Technical university of Liberec

Faculty of mechatronics, informatics
and interdisciplinary studies

Flow123d

version 1.7.1

**Documentation of file formats
and brief user manual.**

Liberec, 2013

Authors (of version 1.7.1)

Jan Březina, Jan Stebel, Jiří Hnídek, David Flanderka, Pavel Exner, Lukáš Zedek

Acknowledgement

This work was supported by the Technology Agency of the Czech Republic under the project no. TA01021331.

Contents

Chapter 1

Quick start

Flow123D is a software for simulation of water flow and reactionary solute transport in a heterogeneous porous and fractured medium. In particular it is suited for simulation of underground processes in a granite rock massive. The program is able to describe explicitly processes in 3D medium, 2D fractures, and 1D channels and exchange between domains of different dimensions. The computational mesh is therefore collection of 3D tetrahedrons, 2D triangles and 1D line segments.

The water flow model assumes a saturated medium described by Darcy law. For discretization, we use lumped mixed-hybrid finite element method. We support both steady and unsteady water flow.

The solute transport model can deal with several dissolved substances. It contains non-equilibrium dual porosity model, i.e. exchange between mobile and immobile pores. There are also models for several types of adsorption in both the mobile and immobile zone. The implemented adsorption models are linear adsorption, Freundlich isotherm and Langmuir isotherm. The solute transport model uses finite volume discretization with up-winding in space and explicit Euler discretization in time. The dual porosity and the adsorption are introduced into transport by operator splitting. The dual porosity model use analytic solution and the non-linear adsorption can be either solved numerically by the $toms748\_solve(...)$ command (which is a part of boost package) or it can be approximated through interpolation which uses precomputed functional values lying on selected type of isotherm.

Reaction between transported substances can be modeled either by a SEMCHEM module, which is slow, but can describe all sorts of reactions. On the other hand, for reactions of the first order, i.e. linear reactions or decays, we provide our own solver which is much faster. Reactions are coupled with transport by the operator splitting method.

The program provides output of the pressure, the velocity and the concentration fields in two file formats. You can use file format of GMSH mesh generator and post-processor or you can use output into widely supported VTK format. In particular we recommend Paraview software for visualization and post-processing of VTK data.

The program is implemented in C/C++ using essentially PETSC library for linear algebra. The water flow as well as the transport simulation and reactions can be computed in parallel using MPI environment.

The program is distributed under GNU GPL v. 3 license and is available on the project

web page: http://dev.nti.tul.cz/trac/flow123d

1.1 Basic usage

1.1.1 How to run the simulation.

On the Linux system the program can be started either directly or through a script `flow123d.sh`. When started directly, e.g. by the command

```
> flow123d -s example.con
```

the program requires one argument after switch `-s` which is the name of the principal input file. Full list of possible command line arguments is as follows.

`--help`
: Parameters interpreted by Flow123d. Remaining parameters are passed to PETSC.

`-s, --solve` *file*
: Set principal CON input file. All relative paths in the CON file are relative against current directory.

`-i, --input_dir` *directory*
: The place holder `${INPUT}` used in the path of an input file will be replaced by given *directory*.

`-o, --output_dir` *directory*
: All paths for output files will be relative to this *directory*.

`-l, --log` *file_name*
: Set base name of log files.

`--no_log`
: Turn off logging.

`--no_profiler`
: Turn off profiler output.

`--full_doc`
: Prints full structure of the main input file.

`--JSON_template`
: Prints a description of the main input file as a valid CON file template.

`--latex_doc`
: Prints a description of the main input file in LaTeX format using particular macros.

All other parameters will be passed to the PETSC library. An advanced user can influence lot of parameters of linear solver. In order to get list of supported options use parameter `-help` together with some valid input. Options for various PETSC modules are displayed when the module is used for the first time.

Alternatively, you can use script `flow123d.sh` to start parallel jobs or limit resources used by the program. This script accepts the same parameters as the program itself and further following additional parameters:

-h
: Usage overview.

-t *timeout*
: Upper estimate for real running time of the calculation. Kill calculation after *timeout* seconds. Can also be used by PBS to choose appropriate job queue.

-np *number of processes*
: Specify number of parallel processes for calculation.

-m *memory limit*
: Limits total available memory to *memory limit* bytes.

-n *priority*
: Change (lower) priority for the calculation. See `nice` command.

-r *out file*
: Stdout and stderr will be redirected to *out file*.

On the Windows system we use Cygwin libraries in other to emulate Linux API. Therefore you have to keep the Cygwin libraries within the same direcotry as the program executable. The Windows package that can be downloaded from project web page contains both the Cygwin libraries and the mpiexec command for starting parallel jobs on the Windows workstations.

Then you can start the sequential run by the command:

```
> flow123d.exe -s example.con
```

or the parallel run by the command:

```
> mpiexec.exe -np 2 flow123d.exe -s example.con
```

The program accepts the same parameters as the Linux version, but there is no script similar to `flow123d.sh` for the Windows system.

1.1.2 Tutorial problem

CON file format

The main input file uses a slightly extended JSON file format which together with some particular constructs forms a CON (C++ object notation) file format. Main extensions of the JSON are unquoted key names (as long as they do not contain whitespaces), possibility to use `=` instead of `:` and C++ comments, i.e. `//` for a one line and `/* */` for a multi-line comment. In CON file format, we prefer to call JSON objects "records" and we introduce also "abstract records" that mimic C++ abstract classes, arrays of a

CON file 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), there are few special upper case keys that are interpreted by the reader: `REF` key for references, `TYPE` key for specifing actual type of an abstract record. For detailed description see Section 4.1.

Geometry

In the following, we shall provide a commented input for the tutorial problem:

`tests/03_transport_small_12d/flow_vtk.con`

We consider a simple 2D problem with a branching 1D fracture (see Figure 1.1 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:

```
cl1 = 0.16;
Point(1) = {0, 1, 0, cl1};
Point(2) = {1, 1, 0, cl1};
Point(3) = {1, 0, 0, cl1};
Point(4) = {0, 0, 0, cl1};
Point(6) = {0.25, -0, 0, cl1};
Point(7) = {0, 0.25, 0, cl1};
Point(8) = {0.5, 0.5, -0, cl1};
Point(9) = {0.75, 1, 0, cl1};
Line(19) = {9, 8};
Line(20) = {7, 8};
Line(21) = {8, 6};
Line(22) = {2, 3};
Line(23) = {2, 9};
Line(24) = {9, 1};
Line(25) = {1, 7};
Line(26) = {7, 4};
Line(27) = {4, 6};
Line(28) = {6, 3};
Line Loop(30) = {20, -19, 24, 25};
Plane Surface(30) = {30};
Line Loop(32) = {23, 19, 21, 28, -22};
Plane Surface(32) = {32};
Line Loop(34) = {26, 27, -21, -20};
Plane Surface(34) = {34};
Physical Point(".1d_top") = {9};
Physical Point(".1d_left") = {7};
Physical Point(".1d_bottom") = {6};
Physical Line("1d_upper") = {19};
Physical Line("1d_lower") = {21};
Physical Line("1d_left_branch") = {20};
Physical Line(".2d_top") = {23, 24};
Physical Line(".2d_right") = {22};
Physical Line(".2d_bottom") = {27, 28};
Physical Line(".2d_left") = {25, 26};
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.

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

```
{
  problem = {
    TYPE = "SequentialCoupling",
    description = "Transport 1D-2D, (convection, dual porosity, sorption)",
    mesh = {
      mesh_file = "./input/mesh_with_boundary.msh",
      sets = [
          { name="1d_domain",
            region_labels = [ "1d_upper", "1d_lower", "1d_left_branch" ]
          }
        ]
    },
```

The file starts with a particular problem type selection, currently only the type `SequentialCoupling` is supported, and a textual problem description. Next, we specify the computational mesh, here it consists of the name of the mesh file and the declaration of one *region set* composed 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 numbers (e.g to have leakage on single element), defining boundary regions, and set operations with region sets, see Section 4.2.1 for details.

Flow setting

Next, we setup the flow problem. We shall consider a flow driven only by the pressure gradient (no gravity), setting the Dirichlet boundary condition on the whole boundary with the pressure head equal to $x+y$. The conductivity will be 1 on the 2D domain and 10 on the 1D domain. The fracture width will be $\delta_1 = 1$ (quite unnatural) as well as the transition parameter $\sigma_2 = 1$ which describes a "conductivity" between dimensions. These are currently the default values.

```
    primary_equation = {
      TYPE = "Steady_MH",

      bulk_data = [
        { r_set = "1d_domain", conductivity = 10 },
        { region = "2d",       conductivity = 1  }
      ],

      bc_data = [
        { r_set = "BOUNDARY",
          bc_type = "dirichlet",
          bc_pressure = { TYPE="FieldFormula", value = "x+y" }
        }
      ],

```

```
28        output = {
29          output_stream = { REF = "/system/output_streams/0" },
30          pressure_p0 = "flow_output_stream",
31          pressure_p1 = "flow_output_stream",
32          velocity_p0 = "flow_output_stream"
33        },
34
35        solver = { TYPE = "Petsc", accuracy = 1e-07 }
36      }, // primary equation
```

On line 11, we specify particular implementation (numerical method) of the flow solver, in this case the Mixed-Hybrid solver for unsteady problems. On lines 16 – 19, we set mathematical fields that live on the computational domain (i.e. the bulk domain), we set only the conductivity field since other bulk fields have appropriate default values. On lines 21 – 26, we set fields for boundary conditions (`bc_data`). We use implicitely defined set "BOUNDARY" that contains all boundary regions and 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 `TYPE=FieldFormula`. See Section **??** for other field types. On lines 28 – 33, we specify which output fields should be written into which output stream (that means particular output file, with given format). Currently, we support only one output stream per equation, so this allows at least switching individual output fields on or off. Notice the reference used on line 29 pointing to the definition of the output streams at the end of the file. Finally, we specify type of the linear solver and its tolerance.

Transport setting

We also consider subsequent transport problem with the porosity $\theta = 0.25$ and zero initial concentration. The boundary condition is equal to 1 and is automatically applied only on the inflow part of the boundary. There are also some adsorption and dual porosity models in this particular test case. Adsorption and simple reactions model inputs are particularly described in subsections 1.1.2.

```
37      secondary_equation = {
38        TYPE = "TransportOperatorSplitting",
39
40        dual_porosity = true,
41        substances = [ "age", "U235", "U235s-lin", "U235s-lang", "U235s-freund"],
42
43        bulk_data = [
44          { r_set = "ALL",
45            init_conc = 0,
46            por_m = 0.25,
47            por_imm = 0.25
48          }
49        ],
50
51        bc_data = [
```

```
52          { r_set = "BOUNDARY",
53            bc_conc = 1.0
54          }
55        ],
56
57        output = {
58          output_stream = { REF = "/system/output_streams/1" },
59          save_step = 0.01,
60          mobile_p0 = "transport_output_stream"
61        },
62
63        time = { end_time = 1.0 }
64
65        reactions = {
66          TYPE="Sorptions",
67          solvent_dens = 1.0,
68          species = [ "age", "U235", "U235s-lin", "U235s-lang", "U235s-freund"],
69          molar_masses = [1.0, 1.0, 1.0, 1.0, 1.0],
70          solubility = [1.0, 1.0, 1.0, 1.0, 1.0],
71          substeps = 50,
72          bulk_data = [
73            {
74              region = "2d",
75              rock_density = {TYPE="FieldFormula", value = "1.0*x/x"},
76              sorption_types = [ "none",  "none", "linear", "langmuir", "freundlich"],
77              mult_coefs = [0, 0, 0.6, 0.4, 0.6],
78              second_params = [0, 0, 0, 0.6, 0.4]
79            }
80          ]
81        }
82      } // secondary_equation
83    }, // problem
```

For the transport problem we use implementation called "TransportOperatorSplitting" which is explicit finite volume solver of the convection equation (without diffusion), the operator splitting is used for the equilibrium adsorption as well as for the dual porosity model and first order reactions simulation. Dual porosity is switched on as we can see on lines 40. On the next line, we set names of transported substances, here it is the age of the water and the uranium 235. On lines 44 – 55, we set the bulk fields in particular the porosity 'por_m' and the initial concentrations (one for every substance). However, on line 46, we see only single value since an automatic conversion is applied to turn the scalar zero into the zero vector (of size 2). On line 53, we can see vector that set different adsorption coefficients for the two substances. Then, on lines 57 – 61, we set the boundary fields namely the concentration on the inflow part of the boundary. We need not to specify type of the condition since currently this is the only one available. In the output record we have to specify the save step (line 65) for the output fields. And finally, we have to set the time setting, here only the end time of the simulation since the step size is determined from the CFL condition, however you can set smaller time step if you want.

Sorption settings

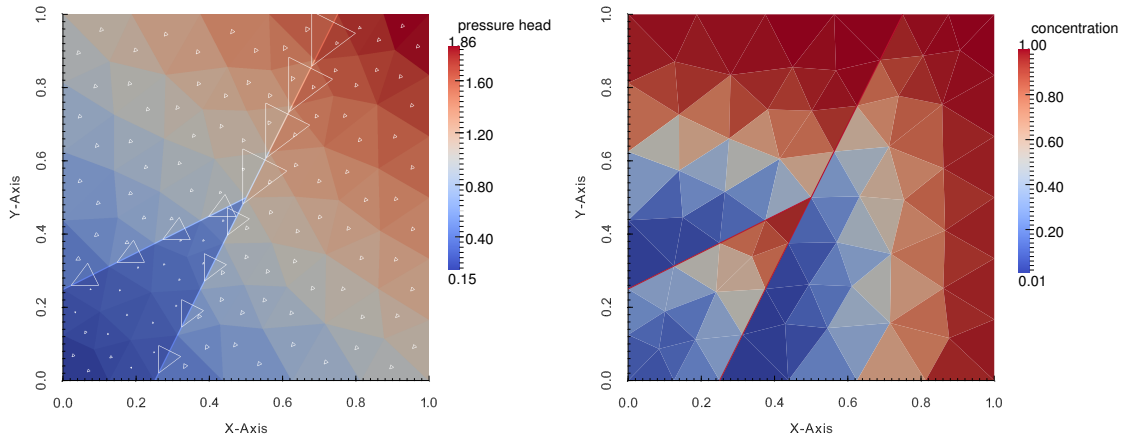
The input information for equilibrial sorption are enclosed in the record *reactions* = {...}. The type of simulated, chemical interactions is determinated by the label $TYPE$ = "*Sorptions*",. It is possible to set the value of the solvent (obviously water) density *solvent_dens* = 1.0,. The solvent density is supposed to be constant all over the simulated area. The vector *species* containes the list of soluted substances whose concentrations is concidered to be affected by sorptions. Material characteristics of all the sorbing species can be defined by vectors *molar_masses* and *solubility*. Elements of the vector *sollubility* defines the upper bound of an aqueous concentrations which can appear, because some substances have limmited solubility and if the solubility exceeds this limit they start to precipitate. *solubility* is a crucial parameter for solving further described set of nonlinear equations. The parameter *substeps* is important when interpolation is used to search approximative solution of the adsorption problem. It is the number of precomputed points lying on the isotherm. Default value is set to be 100.

The record *bulk_data* colects information about regions specific, several isotherms desribed adsorption. Particular region (bulk Physical Entity), where one kind of adsorption takes place, can be specified by its label from gmsh-file. All implemented types of adsorption can take the rock density in region into account. The value of *rock_density* can be either constant or specified by $FieldFormula$. The implemented *sorption_types* can have one of four possible values {"*none*","*linear*","$freundlich$","*langmuir*"} and except of first one are those types empiricaly described by appropriate type of isotherm. Linear isotherm described adsorption needs just one parameter to be given whereas Freundlichs' and Langmuirs' isotherm have two parameters. For further details about mathematical description see the section 2.3. Isothermaly described sorption simulation can be used in the case of low concenttrated solutions without competition between multiple disoved species.

Output streams and results

```
84    system = {
85      output_streams = [
86        {
87          file = "test3.pvd",
88          format = { TYPE = "vtk", variant = "ascii" },
89          name = "flow_output_stream"
90        },
91        {
92          file = "test3-transport.pvd",
93          format = { TYPE = "vtk", variant = "ascii" },
94          name = "transport_output_stream"
95        }
96      ]
97    }
98  }
```

The end of the input file contains declaration of two output streams, one for the flow problem and one for the transport problem. Currently, we support output into VTK



(a) Elementwise pressure head and velocity field (triangles).

(b) Propagation of U235 from the inflow part of the boundary.

Figure 1.1: Results of the tutorial problem.

format and GMSH data format. On Figure 1.1 you can see the results, the pressure and the velocity field on the left and the concentration of U235 at time $t = 0.9$ on the right. Even if the pressure gradient is the same on the 2D domain as on the fracture, the velocity field is ten times faster on the fracture. Since porosity is same, the substance is transported faster by the fracture and then appears in the bottom left 2D domain before the main wave propagating solely through the 2D domain.

The output files can be either `*.msh` files accepted by the GMSH or one can use VTK format that can be post-processed by Paraview.

In the following chapter, we briefly describe structure of individual input files in particular the main INI file. In the last chapter, we describe mathematical models and numerical methods used in the Flow123d.

Chapter 2

Mathematical models of physical reality

Flow123d provides models for Darcy flow in porous media as well as for the transport and reactions of soluted substances. In this section, we describe mathematical formulations of these models together with physical meaning and units of all involved quantities. Common and unique feature of all models is 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 open set $\Omega_2 \subset \mathbf{R}^2$ representing 2D fractures and open set $\Omega_1 \subset \mathbf{R}^3$ of 1D channels or preferential paths (see Fig 2.1). We assume that Ω_2 and Ω_1 are polygonal. 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 simplexes that are tetrahedrons, triangles and lines.

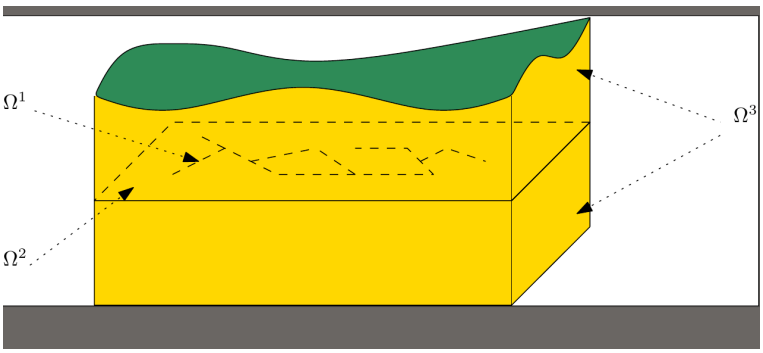


Figure 2.1: Scheme of a problem with domains of multiple dimensions.

Present numerical methods requires 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.

2.1 Darcy flow model

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

$$\boldsymbol{w} = -\mathbb{K}\nabla H \quad \text{on } \Omega_d, \text{ for } d = 1, 2, 3. \tag{2.3}$$

We drop the dimension index of quantities in equations if it is the same as the dimension of the set where the equation holds. In (2.3), $\boldsymbol{w}_d$ [ms^{-1}] is the superficial velocity, $\mathbb{K}_d$ is the conductivity tensor, and H_d [m] is the piezometric head. The velocity is related to the flux $\boldsymbol{q}_d$ with units [m^{4-d}s^{-1}] through

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

where δ_d [m^{3-d}] is a 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 q_d is the volume of the liquid (water) that pass 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 $k_d > 0$ is the hydraulic conductivity [ms^{-1}] and $\mathbb{A}_d$ is 3×3 dimensionless anisotropy tensor which has to be symmetric and positive definite. The piezometric-head H_d has units [m] and is related to the pressure head h_d by $H_d = h_d + z$ assuming that the gravity force acts in negative direction of the z-axes. Combining these relations we get Darcy low in the form:

$$\boldsymbol{q} = -\delta k \mathbb{A} \nabla (h + z) \qquad \text{on } \Omega_d, \text{ for } d = 1, 2, 3. \tag{2.4}$$

Next, we employ continuity equation for saturated porous medium:

$$\partial_t (S\, h) + \operatorname{div} \boldsymbol{q} = F \qquad \text{on } \Omega_d, \text{ for } d = 1, 2, 3, \tag{2.5}$$

where S_d is the storativity and F_d is a source term. In our setting the principal unknowns of the system (2.4, 2.5) are the pressure head h_d and the flux $\boldsymbol{q}_d$.

The storativity $S_d > 0$ or the volumetric specific storage [m^{-1}] can be expressed as

$$S_d = \gamma_w (\beta_r + \nu \beta_w), \tag{2.6}$$

where γ_w [kgm^{-2}s^{-2}] is the specific weight of water, ν 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 [m^{3-d}s^{-1}] on the right hand side of (2.5) consists of the volume density of prescribed sources f_d [s^{-1}] and flux from higher dimension. Exact formula is slightly different for every dimension and will be discussed presently.

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

On the set $\Omega_2 \cap \Omega_3$ the fracture is surrounded by one 3D surface from every side (or just one surface since we allow also 2D models on the boundary). On $\partial\Omega_3 \cap \Omega_2$ we prescribe boundary condition of Robin type

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

where $\boldsymbol{q}_3 \cdot \boldsymbol{n}^{+/-}$ [ms^{-1}] is the outflow from Ω_3, $h_3^{+/-}$ is a trace of the pressure head on Ω_3, h_2 is the pressure head on Ω_2, and $\sigma_3^{+/-} = \sigma_{32}$ [s^{-1}] is the transition coefficient that will be discussed later. On the other hand, the sum of the interchange fluxes $q_{32}^{+/-}$ forms a volume source on Ω_2. Therefore F_2 [ms^{-1}] on the right hand side of (2.5) is given by

$$F_2 = \delta_2 f_2 + (q_{32}^{+} + q_{32}^{-}). \tag{2.7}$$

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

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

where $\sigma_2^i = \delta_2^i \sigma_{21}$ [ms^{-1}] is the transition coefficient integrated over the width of the fracture i. Sum of the fluxes forms part of F_1 [m^{2}s^{-1}]

$$F_1 = \delta_1 f_1 + \sum_i q_{21}^i. \tag{2.8}$$

The transition coefficients σ_{32} [s^{-1}] and σ_{21} [s^{-1}] are independent parameters in our setting however in practice they should be related to the conductivity. According to [4] one can use

$$\sigma_{31} = \frac{2\mathbb{K}_2 : \boldsymbol{n}_2 \otimes \boldsymbol{n}_2}{\delta_2}, \sigma_{21} = \frac{2\mathbb{K}_1 : \boldsymbol{n}_1^i \otimes \boldsymbol{n}_1^i}{\delta_1}$$

where $\boldsymbol{n}_2$ is normal to the fracture (sign doesn't matter) and $\boldsymbol{n}_1^i$ is normal to the channel that is tangential to the fracture i.

In order to obtain unique solution we have to prescribe boundary conditions. Currently we support three basic types of boundary condition. Consider disjoint decomposition of the boundary

$$\partial\Omega_d = \Gamma_d^D \cap \Gamma_d^N \cap \Gamma_d^R$$

into Dirichlet, Neumann, and Robin parts. We prescribe

$$h_d = h_d^D \qquad \text{on } \Gamma_d^D, \tag{2.9}$$

$$\boldsymbol{q}_d \cdot \boldsymbol{n} = q_d^N \qquad \text{on } \Gamma_d^N, \tag{2.10}$$

$$\boldsymbol{q}_d \cdot \boldsymbol{n} = \sigma_d^R (h_d - h_d^R) \qquad \text{on } \Gamma_d^R. \tag{2.11}$$

where h_d^D, h_d^R is the prescribed pressure head [m], which alternatively can be prescribed through the piezometric head H_d^D, H_d^R respectively. q_d^N is the prescribed surface density of the boundary outflow [m^{4-d}s^{-1}], and σ_d^R is the transition coefficient [m^{3-d}s^{-1}]. The problem is well posed only if there is Dirichlet or Robin boundary condition on every component of the set $\Omega_1 \cup \Omega_2 \cup \Omega_3$ and $\sigma_d > 0$ for $d = 2, 3$.

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

2.2 Transport of substances

Flow123d can simulate transport of substances dissolved in water. The transport mechanism is governed by the *advection*, and the *hydrodynamic dispersion*. Moreover the substances can move between ground and fractures.

In the domain Ω_d of dimension $d \in \{1, 2, 3\}$, we consider a system of mass balance equations in the following form:

$$\delta_d \partial_t(\vartheta c^i) + \operatorname{div}(\boldsymbol{q}_d c^i) - \operatorname{div}(\vartheta \delta_d \mathbb{D}^i \nabla c^i) = F_S^i + F_C(c^i) + F_R(c^1, \dots, c^s). \tag{2.12}$$

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

- ϑ [] is the porosity, i.e. fraction of space occupied by water and the total volume.
- The hydrodynamic dispersivity tensor $\mathbb{D}^i$ [m^{2}s^{-1}] has the form

  $$\mathbb{D}^i = D_m^i \tau \mathbb{I} + |\boldsymbol{v}| \left(\alpha_T^i \mathbb{I} + (\alpha_L^i - \alpha_T^i) \frac{\boldsymbol{v} \times \boldsymbol{v}}{|\boldsymbol{v}|^2} \right),$$

  which represents (isotropic) molecular diffusion, and mechanical dispersion in longitudal and transversal direction to the flow. Here D_m^i [m^{2}s^{-1}] is the molecular diffusion coefficient of the i-th substance (usual magnitude in clear water is 10^{-9}), $\tau = \vartheta^{1/3}$ is the tortuosity (by [5]), α_L^i [m] and α_T^i [m] is the longitudal dispersivity and the transversal dispersivity, respectively. Finally, $\boldsymbol{v}$ [ms^{-1}] is the *microscopic* water velocity, related to the Darcy flux $\boldsymbol{q}_d$ by the relation $\boldsymbol{q}_d = \vartheta \delta_d \boldsymbol{v}$. The value of D_m^i for specific substances can be found in literature (see e.g. [1]). For instructions on how to determine α_L^i, α_T^i we refer to [2, 3].
- F_S^i [kgm^{-d}s^{-1}] represents the density of concentration sources. Its form is:

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

  Here ϱ_S^i [kgm^{-d}s^{-1}] is the density of concentration sources, c_S^i is an equilibrium concentration and σ_S^i is the concentration flux.
- $F_C(c^i)$ [kgm^{-d}s^{-1}] is the density of concentration sources due to exchange between regions with different dimensions, see (2.15) below.
- The reaction term $F_R(\dots)$ [kgm^{-d}s^{-1}] is currently neglected.

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 two parts:

$$\Gamma_D(t) = \{\boldsymbol{x} \in \partial\Omega_d \,|\, \boldsymbol{q}(t, \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) < 0\},$$
$$\Gamma_N(t) = \{\boldsymbol{x} \in \partial\Omega_d \,|\, \boldsymbol{q}(t, \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) \geq 0\},$$

where $\boldsymbol{n}$ stands for the unit outward normal vector to $\partial\Omega_d$. On the inflow part Γ_D, the user must provide Dirichlet boundary condition for concentrations:

$$c^i(t, \boldsymbol{x}) = c_D^i(t, \boldsymbol{x}) \text{ on } \Gamma_D(t),$$

while on Γ_N we impose homogeneous Neumann boundary condition:

$$-\vartheta \delta_d \mathbb{D}^i(t, \boldsymbol{x}) \nabla c^i(t, \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) = 0 \text{ on } \Gamma_N(t).$$

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 comunication on the interface between Ω_{d+1} and Ω_d is described by:

$$q^c = \delta_{d+1}\sigma^c(\vartheta_{d+1}c_{d+1} - \vartheta_d c_d) + \begin{cases} q^w c_{d+1} & \text{if } q^w \geq 0, \\ q^w c_d & \text{if } q^w < 0, \end{cases} \tag{2.13}$$

where

- q^c [kgm^{-d}s^{-1}] is the density of concentration flux from Ω_{d+1} to Ω_d,
- σ^c [ms^{-1}] is a transition parameter. Its nonzero value causes mass exchange between dimensions whenever the concentrations differ. It is recommended to set either $\sigma^c = 0$ (exchange due to water flux only) or, similarly as in (2.1),

$$\sigma^c \approx \frac{\delta_{d+1}}{\delta_d}\mathbb{D} : \boldsymbol{n} \otimes \boldsymbol{n}.$$

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

Equation (2.13) is incorporated as the total flux boundary condition for the problem on Ω_{d+1} and a source term in Ω_d:

$$-\vartheta\delta_{d+1}\mathbb{D}\nabla c_{d+1} \cdot \boldsymbol{n} + q^w c_{d+1} = q^c, \tag{2.14}$$

$$F_C^d = q^c. \tag{2.15}$$

Dual porosity Up to now we have described the transport equation for the single porosity model. The dual porosity model splits the mass into to zones the mobile zone and the immbile 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 traped and can not pass through. The rest of the pore volume is ocuppied 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 nonequilibrium linear model:

$$\theta_m \partial_t c_m = \alpha(c_i - c_m), \tag{2.16}$$

$$\theta_i \partial_t c_i = \alpha(c_m - c_i) \tag{2.17}$$

where c_m is the concentration in the mobile zone, c_i is the concentration in the immobile zone, α is a diffusion parameter, θ_m and θ_i are porosities of the mobile zone and the immobile zone respectively, while

$$\theta_m + \theta_i = \theta.$$

The solution of this system is:

$$c_m(t) = (c_m(0) - c_a(0))\exp(-\alpha(\frac{1}{\theta_m} + \frac{1}{\theta_i})t) + c_a(0), \tag{2.18}$$

$$c_i(t) = (c_i(0) - c_a(0))\exp(-\alpha(\frac{1}{\theta_m} + \frac{1}{\theta_i})t) + c_a(0) \tag{2.19}$$

where c_a is weighted average:

$$c_a = \frac{\theta_m c_m + \theta_i c_i}{\theta_m + \theta_i}.$$

2.3 Equilibrial Adsorption

The simulation of monolayer, equilibrial adsorption is based on solution of the couple of equations representing mass balance law and empirical description of adsorption represented by comon types of isotherm. Considered types of isotherm folows:

- Without adsorption, *sorption_type* is "*none*".
- Linear isotherm $c_s = f(c_a) = k_l \cdot c_a$, *sorption_type* is "*linear*".
- Freundlichs' isotherm $c_s = f(c_a) = k_F \cdot c_a^{\alpha}$, *sorption_type* is "*freundlich*".
- Langmuirs' isotherm $c_s = f(c_a) = k_L \cdot \frac{\alpha \cdot c_a}{1+\alpha \cdot c_a}$, *sorption_type* is "*langmuir*". Langmuirs isotherm has been derived form thermodynamic laws. k_L denotes the maximal amount of sorbing specie which can be capt per unit volume of a bulk matrix. Coefficient α is a fraction of adsorption and desorption rate constant $\alpha = \frac{k_a}{k_d}$

Notification:

- Sorbed concentration $[c_s] = \frac{[n]}{[m_H]} = \frac{N}{M} = \frac{Mol}{kg}$, where m_H is the mass of bulk and n denotes amount of substance in an element.
- Aqueous concentration $[c_a] = \frac{[m]}{[m_w]} = \frac{M}{M} = \frac{kg}{kg} = 1$, where m_w is the mass of water/solvent in element. Just water ($\rho_w = 1\ kg \cdot l^{-1}$) is supposed to be solvent in version 1.7.0.
- Multiplication parameters, $k_i,\ i \in \{l, F, L\}$, *mult_coefs* can have various physical dimensions.
- Aditional parameters, $[\alpha] = 1$, *second_params*.

The mass balance equation can be derived from 2.20,

$$m_{Total} = m_{aqueous} + m_{sorbed} = c_a \cdot V_{elm} \cdot \rho_w \cdot n + c_s \cdot M_s \cdot V_{elm} \cdot \rho_H \cdot (1 - n) \tag{2.20}$$

where ρ_w denotes solvent density *solvent_dens*, ρ_H is the symbol for *rock_density*, V_{elm} denotes element volume, n is the symbol for porosity, here, and M_s denotes *molar_masses*.

The equation 2.20 depends on volume of an element V_{elm}. We devide both sides by V_{elm} to suppress this dependency and we get resulting mass balance equation 2.21.

$$\begin{aligned} konst._T &= k_a \cdot c_a + k_s \cdot c_s \\ k_a &= \rho_w \cdot n \\ k_s &= M_s \cdot \rho_H \cdot (1 - n) \end{aligned} \tag{2.21}$$

After the substitution $|c_s = f(c_a)|$ we become the equation, which can be either solved iteratively or aproximated through interpolation. This equation has following form.

$$konst._T = k_a \cdot c_a + k_s \cdot f(c_a) \tag{2.22}$$

To solve the equation 2.22 iteratively, it is very important to define interval where to look for solution (unknown c_a). The lower bound is 0. Concentration can not reach negative

values. The upper bound is derived using simple imagination. Lets suppose limmited *solubility* of selected transported substance and lets denote the limmit c_a^{limmit}. We keep the maximal "total mass" $konst._T^{limit} = k_a \cdot c_a^{limmit} + k_s \cdot f(c_a^{limmit})$, but we dissolve all the mass to get maximal $c_a^{max} > c_a^{limmit}$. That means $c_s = 0$ for this moment. To understand this step lets look at the figure 2.2. We can slightly enlarge the interval by setting the upper bound equal to $c_a^{max} + const_{small}$.

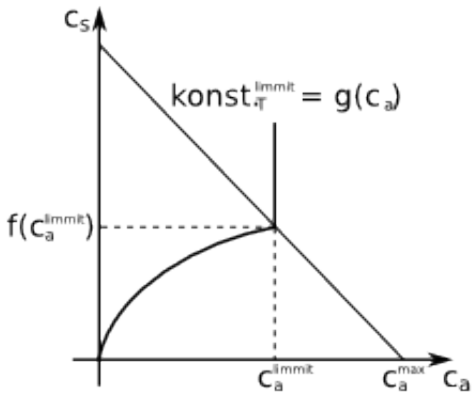


Figure 2.2: Sorption in combination with limmited solubility.

To approximate the equation 2.22 using interpolation we need to prepare the set of values which represent $[c_a, f(c_a)]$, with c_a equidistantly distributed in transformed (rotated and rescaled) coordination system at first. The approach for construction of interpolation table follows.

1. Maximal "total mass" $konst._T^{limit} = k_a \cdot c_a^{limmit} + k_s \cdot f(c_a^{limmit})$ is computed.
2. Total mass step is derived $mass\_step = konst._T^{limit}/n\_steps$. n_steps is equal to *substeps* in con-file.
3. Appropriate $c_a^i = (mass\_step \cdot i)/k_a,\ i \in \{0, \ldots, n\_steps\}$ are computed.
4. The equations $k_a \cdot c_a^i = k_a \cdot c_a + k_s \cdot f(c_a)\ i \in \{0, \ldots, n\_steps\}$ are solved for c_a as unknown. The solution is the set of ordered couples (points) $[c_a^l, f(c_a^l)],\ l \in \{0, \ldots, n\_steps\}$.

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

$$\begin{gathered}\overrightarrow{c}^R = \mathbf{A} \cdot \overrightarrow{c} \\ \begin{bmatrix} c_a^{R,l} \\ c_s^{R,l} \end{bmatrix} = \begin{bmatrix} n \cdot \rho_w & M_s(1-n)\rho_H \\ -M_s(1-n)\rho_H & n \cdot \rho_w \end{bmatrix} \cdot \begin{bmatrix} c_a^l \\ c_s^l \end{bmatrix} \\ l \in \{0, \ldots, n\_steps\}\end{gathered} \tag{2.23}$$

The values $c_a^{R,l}$ are equidistantly distributed and there is no reason to save them, but the values $c_s^{R,l}$ are stored in onedimensional interpolation table.

Once we have the interpolation table, we can use it for projection of $[c_a, c_s]$ transport results on the isotherm under concideration. The approach look as folows.

1. Achieved concentrations are transformed to the coordination system through multiplication with the matrix $\mathbf{A}$, see 2.23.

2. Transrformed values are interpolated.

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

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

2.3.1 Limmited Solubility

When $k_a \cdot c_a + k_s \cdot f(c_a) > k_a \cdot c_a^{limmit} + k_s \cdot f(c_a^{limmit})$ neither iterative solver nor interpolation table is used. The aqueous concentration is set to be c_a^{limmit} and sorbed concentration is computed $c_s = (k_a \cdot c_a + k_s \cdot f(c_a) - k_a \cdot c_a^{limmit})/k_s$.

Chapter 3

Numerical methods

In this chapter we briefly describe numerical methods used in Flow123d for solution of physical models presented in Chapter 2. For the steady or unsteady Darcy flow we use mixed-hybrid discretization using Raviart-Thomas finite elements described in details in Section 3.1. For the numerical approximation of the advection-dispersion equation (2.12) we distinguish whether the dispersion $\mathbb{D}$ is present or not. Since the true solution has qualitatively different properties, we also choose different numerical methods for each case. For purely advection problems (i.e. without dispersion term) one can use finite volume method with up–winding and explicit Euler method fully described in Section 3.2. For problems with dispersion term (even advection dominated) we provides implicit discontinuous Galerkin method covered in Section 3.3.

3.1 Mixed-Hybrid method for Darcy flow (TO BE DONE)

Based on publications: [?], [?], [?], [?], [?], [?], [?]

3.2 Pure advection (NEEDS ACTUALIZATION)

3.2.1 Advection-Diffusion equation

Solute transport is governed by advection equation which can be written in the form

$$\frac{\partial c}{\partial t} + \boldsymbol{v}\frac{\partial c}{\partial x} = 0, \tag{3.1}$$

where c is concentration $[M^3 \cdot L^{-3}]$, t is time $[T]$, v is velocity $[L \cdot T^{-1}]$, and x is coordinate in cartesian system $[L]$. Assuming solution which is constant on every element (cell centered finite volume method) and integrating equation (3.1) we get

$$\int_{e_i} \frac{\partial c}{\partial t} dV + \int_{e_i} \boldsymbol{v}\frac{\partial c}{\partial x}\, dV = 0.$$

After some rearrangements we obtain on i-th element (e_i)

$$\frac{\partial c_i}{\partial t} V_i + c \int\limits_{\partial e_i} \boldsymbol{v}\, \mathbf{dS} = 0, \tag{3.2}$$

where c_i is average concentration in e_i and V_i its volume, c will be specified later (there are two main possibilities - c_i or concentration from neighbouring element). Term $\frac{\partial c}{\partial t}$ we approximate by explicit Euler difference

$$\frac{\partial c}{\partial \mathrm{t}} \approx \frac{c_i^{n+1} - c_i^n}{\Delta t}. \tag{3.3}$$

Where Δt is a time step and upper index at c_i means values in the discrete time steps $n+1$ and n. We assume that all elements have piecewise smooth element boudary ∂e with outwards directed normal. Inside the area Ω we introduce internal flows. With respect to e_i, we define internal flow intake U_{ij}^- (from element e_j) and internal flow drain U_{ij}^+ (to element e_j) as follows

$$\begin{aligned} U_{ij}^- &= \min(\int\limits_{\partial e_i \cap \partial e_j, i \neq j} \boldsymbol{v}\, \mathbf{dS}, 0), \\ U_{ij}^+ &= \max(\int\limits_{\partial e_i \cap \partial e_j, i \neq j} \boldsymbol{v}\, \mathbf{dS}, 0). \end{aligned} \tag{3.4}$$

Those flows realizes solute transport in the area Ω. On the $\partial\Omega$ we define external flows which will be important for transport Dirichlet boundary conditions. In the same way as for internal flows we assume (with respect to element e_i) external flow intake U_{ij}^{e-} (from $\partial\Omega$) and external flow drain U_{ij}^{e+} (to $\partial\Omega$).

$$\begin{aligned} U_{ik}^{e-} &= \min(\int\limits_{\partial e_i \cap \partial\Omega} \boldsymbol{v}\, \mathbf{dS}, 0), \\ U_{ik}^{e+} &= \max(\int\limits_{\partial e_i \cap \partial\Omega} \boldsymbol{v}\, \mathbf{dS}, 0). \end{aligned} \tag{3.5}$$

Direction of the velocity $\boldsymbol{v}$, which affects sign of the U-terms is significant for the construction solution. For the solution stability it is suitable to use an upwind scheme, which can by written for finite difference on simple 1D geometry in the form

$$\begin{aligned} v > 0 : \frac{\partial c}{\partial \mathrm{x}} &\approx \frac{c_i^n - c_{i-1}^n}{\Delta x}, \\ v < 0 : \frac{\partial c}{\partial \mathrm{x}} &\approx \frac{c_{i+1}^n - c_i^n}{\Delta x}. \end{aligned} \tag{3.6}$$

This scheme can be interpreted as well as in finite volume method - in convection term one can get c value opposite the flow of the quantity $\boldsymbol{v}$ direction. For every e_i we introduce itemsets $\mathcal{N}_i, \mathcal{B}_i$ which contains indexes of neighbourging elements, local boundary conditions respectivelly. Assuming upwind scheme, using (3.4), (3.5), and (3.3) we can write solution of the equation (3.2) (relation between two consecutive time steps) on e_i in the form

$$c_i^{n+1} = c_i^n - \frac{\Delta t}{V_i} \left[\sum_{j \in \mathcal{N}_i} \left[U_{ij}^+ c_i + U_{ij}^- c_j \right] + \sum_{k \in \mathcal{B}_i} \left[U_{ik}^{e+} c_i + U_{ik}^{e-} c_{B_{ik}} \right] \right]. \tag{3.7}$$

Where $c_{B_{ik}}$ are values of Dirichlet boundary conditions which belong to e_i. Formula (3.7) can be rewritten into the matrix notation

$$\mathbf{c}^{n+1} = (\mathbf{I} + \Delta t \mathbf{A}) \cdot \mathbf{c}^n + \Delta t \mathbf{B} \cdot {\mathbf{c_B}}^n \tag{3.8}$$

Where $\mathbf{c}$ is vector of c_i^{n+1}, $\mathbf{A}$ is a square matrix composed from $\frac{U_{ij}^+}{V_i}$, $\frac{U_{ij}^-}{V_i}$, and $\frac{U_{ij}^{e+}}{V_i}$. $\mathbf{B}$ is in general rectangular matrix composed from $\frac{U_{ij}^{e-}}{V_i}$ and ${\mathbf{c_B}}^n$ is vector of Dirichlet boundary conditions.matrix definition. There is one stability condition for time step which is called Courant-Friedrich-Levy condition. For the problem without sources/sinks it can be written as

$$\Delta t_{max} = \min_i \left(\frac{V_i}{\sum_j U_{ij}^+ + \sum_k U_{ik}^{e+}} \right) = \min_i \left(\frac{V_i}{\sum_j |U_{ij}^-| + \sum_k |U_{ik}^{e-}|} \right). \tag{3.9}$$

This condition has a physical interpretation, which can be understood as conservation law - volume that intakes/drains to/from element e_i can not be higher then element volume V_i. From algebraical point of view this condition can be seen as a condition which bounds norm of the evolution operator as follows

$$\|\mathbf{I} + \Delta t \mathbf{A} \quad \Delta t \mathbf{B}\| \leq 1. \tag{3.10}$$

3.2.2 Generalization

This approach can be used as well as for more general element connections – for compatible/non-compatible element interconnection, if we know the flow integral values (U_{ij}^+ or U_{ij}^-). The most general case of connection is relation among n elements like

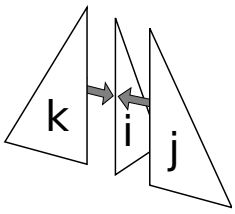


Figure 3.1: Edge with 3 elements

in figure (3.1). For this case we define edge element indexset $\mathcal{G}_l$ that contains all the indexes of elements which sides make l-th edge (g_l), so that $\mathcal{G}_l = \{i, j, k\}$. For $\mathcal{G}_l$ we introduce its subsets $\mathcal{G}_{ij}$, $\mathcal{G}_{ji}$, $\mathcal{G}_{ik}$, $\mathcal{G}_{ki}$, $\mathcal{G}_{kj}$, and $\mathcal{G}_{jk}$, where $\mathcal{G}_{ij} = \mathcal{G}_{ik} = \mathcal{G}_l \backslash i = \{j, k\}$, $\mathcal{G}_{ji} = \mathcal{G}_{jk} = \mathcal{G}_l \backslash j = \{i, k\}$, and $\mathcal{G}_{ki} = \mathcal{G}_{kj} = \mathcal{G}_l \backslash k = \{i, j\}$. It can be written in the same way for any edge g with more than 3 elements, it is hold $|\mathcal{G}_g| - 1 = |\mathcal{G}_{ab}|; \forall a, b \in \mathcal{G}_g$. For

l-th edge (g_l) we can define total edge flow U_{g_l} eg. as

$$\begin{aligned} U_{g_l} &= \sum_{m\in\mathcal{G}_{ji}} \left[U^+_{mj} + \frac{U^+_{jm}}{|\mathcal{G}_{ji}|}\right] = \sum_{m\in\mathcal{G}_{jk}} \left[U^+_{mj} + \frac{U^+_{jm}}{|\mathcal{G}_{jk}|}\right] \\ &= \sum_{m\in\mathcal{G}_{ij}} \left[U^+_{mi} + \frac{U^+_{im}}{|\mathcal{G}_{ij}|}\right] = \sum_{m\in\mathcal{G}_{ik}} \left[U^+_{mi} + \frac{U^+_{im}}{|\mathcal{G}_{ik}|}\right] \\ &= \sum_{m\in\mathcal{G}_{ki}} \left[U^+_{mk} + \frac{U^+_{km}}{|\mathcal{G}_{ki}|}\right] = \sum_{m\in\mathcal{G}_{kj}} \left[U^+_{mk} + \frac{U^+_{km}}{|\mathcal{G}_{kj}|}\right], \end{aligned} \tag{3.11}$$

U_{g_l} with respect to any e_m; $m \in \mathcal{G}_l$ has to have the same value because continuity equation, for assumed incompresible flow, has to be fulfilled in every edge. Edges with more than two elements and two and more nonzero intakes to edge realize an ideal mixing (to an average concentration) with weights which will be specified later. This fact modifies equation (3.7) on the general mesh into the form

$$c_i^{n+1} = c_i^n - \frac{\Delta t}{V_i}\left[\sum_{j\in\mathcal{N}_i}\left[U^+_{ij}c_i + \frac{U^-_{ij}}{\sum\limits_{k\in\mathcal{G}_{ij}}\left[U^+_{ki} + \frac{U^+_{ik}}{|\mathcal{G}_{ij}|}\right]}\sum_{k\in\mathcal{G}_{ij}} U^+_{ki}c_k\right] + \sum_{k\in\mathcal{B}_i}\left[U^{e+}_{ik}c_i + U^{e-}_{ik}c_{B_{ik}}\right]\right]. \tag{3.12}$$

The edges with total edge flow $U_{g_l} = 0$ can occur breakdown in the equation (3.12) via term $\sum\limits_{k\in\mathcal{G}_{ij}}\left[U^+_{ki} + \frac{U^+_{ik}}{|\mathcal{G}_{ij}|}\right] = 0$. This fact implies as well as numerator $U^-_{ij} = 0$. In order to avoid dividing by zero we have to assume computation only for nonzero flows. Concentrations c_k, $k \in \mathcal{G}_{ij}$ that may intakes into element e_i are weighted with weights

$$\alpha_k = \frac{U^+_{ki}}{\sum\limits_{k\in\mathcal{G}_{ij}}\left[U^+_{ki} + \frac{U^+_{ik}}{|\mathcal{G}_{ij}|}\right]}, \tag{3.13}$$

so that the ideal mixing in this edge leads to the average concentration

$$c_{av} = \frac{\sum\limits_{k\in\mathcal{G}_{ij}} U^+_{ki}c_k}{\sum\limits_{k\in\mathcal{G}_{ij}}\left[U^+_{ki} + \frac{U^+_{ik}}{|\mathcal{G}_{ij}|}\right]}. \tag{3.14}$$

Matrix notation is the same as in (3.8). Finally ...

3.3 Advection with dispersion

For the general case we use the discontinuous Galerkin space approximation described in [**?**] and implicit Euler time discretization. Let τ, h be the time step and the space discretization parameter, respectively. We assume that $\mathcal{T}_h^d$ is a regular partition of the domain Ω^d into simplices, $d = 1, 2, 3$. We define the set $\mathcal{E}_h^d$ of all edges of elements in $\mathcal{T}_h$ (triangles for $d = 3$, line segments for $d = 2$ and points for $d = 1$). Further, $\mathcal{E}_{h,I}^d$, $\mathcal{E}_{h,B}^d$ will stand for interior and boundary edges, respectively, $\mathcal{E}_{h,D}^d(t)$ for edges that coincide with $\Gamma_D^d(t)$ and $\mathcal{E}_{h,C}^d$ for edges on interface with Ω^{d-1}.

Let us fix one substance and the space dimension d. At each time instant $t_k = k\tau$ we search for the concentration field $c_d^{h,k} \in V_d^h$, where

$$V_d^h = \{v : \overline{\Omega^d} \to \mathbb{R} \,|\, v_{|T} \in P_1(T)\ \forall T \in \mathcal{T}_h^d\}$$

is the space of functions piecewise affine on the elements of $\mathcal{T}_h^d$, possibly discontinuous across the element interfaces. The initial concentration $c_d^{h,0}$ is set to the projection of the initial data c_0 onto V_d^h. For $k = 1, 2, \ldots$, the discrete problem reads:

$$\left(\delta^d \vartheta \frac{c_d^{h,k} - c_d^{h,k-1}}{\tau}, v\right)_{\Omega^d} + a_d^{h,k}(c_d^{h,k}, v) = b_d^{h,k}(v) \quad \forall v \in V_d^h.$$

Here $(f,g)_{\Omega^d} = \int_{\Omega^d} fg$, $c_d^{h,k-1}$ is the solution from the previous time step and the forms $a_d^{h,k}$, $b_d^{h,k}$ are defined as follows:

$$\begin{aligned} a_d^{h,k}(u,v) = {} & \left(\delta^d \vartheta \mathbb{D} \nabla u, \nabla v\right)_{\Omega^d} - (\boldsymbol{q} u, \nabla v)_{\Omega^d} \\ & - \sum_{E \in \mathcal{E}_{h,I}^d} \left(\left(\left\{\delta^d \vartheta \mathbb{D} \nabla u\right\}_\omega \cdot \boldsymbol{n}, [v]\right)_E + \theta \left(\left\{\delta^d \vartheta \mathbb{D} \nabla v\right\}_\omega \cdot \boldsymbol{n}, [u]\right)_E \right) \\ & + \sum_{E \in \mathcal{E}_{h,I}^d} \left(\boldsymbol{q} \cdot \boldsymbol{n} \{u\}, [v]\right)_E + \sum_{E \in \mathcal{E}_{h,B}^d} (\boldsymbol{q} \cdot \boldsymbol{n} u, v)_E \\ & + \sum_{E \in \mathcal{E}_{h,I}^d} \gamma_E \left([u], [v]\right)_E + \sum_{E \in \mathcal{E}_{h,D}^d(t_k)} \gamma_E \left(u, v\right)_E, \end{aligned}$$

$$b_d^{h,k}(v) = \sum_{E \in \mathcal{E}_{h,D}^d(t_k)} \gamma_E \left(c_D, v\right)_E.$$

For an interior edge E we denote by $T^-(E)$ and $T^+(E)$ the elements sharing E. By $\boldsymbol{n}$ we mean the unit normal vector to E pointing from $T^-(E)$ towards $T^+(E)$, the inter-element jump is defined as $[f] = f_{|T^-(E)} - f_{|T^+(E)}$, $\{f\} = \frac{1}{2}(f_{|T^-(E)} + f_{|T^+(E)})$ and $\{f\}_\omega = \omega f_{|T^-(E)} + (1-\omega) f_{|T^+(E)}$ denotes the usual and weighted average of the quantity f. The weight ω is chosen in a specific way (see [**?**] for details) taking into account possible inhomogeneity of $\mathbb{D}$. The stabilization parameter $\gamma_E > 0$ is user dependent; its value affects the inter-element jumps of the solution. The constant θ can take the value -1, 0 or 1, which corresponds to the nonsymmetric, incomplete and symmetric variant of the interior penalty DG method.

In case that lower dimensional domains (Ω^1, Ω^2) have complex topology, e.g. if there are more triangles sharing one line segment, then we consider ideal mixing, i.e. the concentration entering the edge through every inlet element ($\boldsymbol{q}$ points out of this element) is divided among all outlet elements proportionally to their fluxes.

If there are interfaces between adjacent dimensions, then the following terms are added to the bilinear forms:

$$\begin{aligned} & a_{d+1,C}^{h,k}((u_{d+1}, u_d), v_{d+1}) \\ & \qquad = \sum_{E \in \mathcal{E}_{h,C}^{d+1}} \left(\delta_{d+1} \sigma^c (\vartheta_{d+1} u_{d+1} - \vartheta_d u_d) + (\boldsymbol{q} \cdot \boldsymbol{n})^+ u_{d+1} - (\boldsymbol{q} \cdot \boldsymbol{n})^- u_d, v_{d+1}\right)_E, \end{aligned}$$

$$a_{d,C}^{h,k}((u_{d+1},u_d),v_d)$$
$$= -\sum_{E\in\mathcal{E}_{h,C}^{d+1}} \left(\delta_{d+1}\sigma^c(\vartheta_{d+1}u_{d+1}-\vartheta_d u_d)+(\boldsymbol{q}\cdot\boldsymbol{n})^+u_{d+1}-(\boldsymbol{q}\cdot\boldsymbol{n})^-u_d,v_d\right)_E.$$

Here $f^+=\max\{0,f\}$ and $f^-=-\min\{0,f\}$ is the positive, negative part, respectively and $\mathcal{E}_{h,C}^4=\mathcal{E}_{h,C}^1=\emptyset$.

A priori error estimates for the equation in a single domain are done in [**?**], for a posteriori estimates see [**?**].

Chapter 4

File formats

4.1 Main input file (CON file format)

In this section, we shall describe structure of the main input file that is given through the parameter `-s` on the command line. The file formats of other files that are referenced from the main input file and used for input of the mesh or large field data (e.g. the GMSH file format) are described in following sections. The input subsystem was designed with the aim to provide uniform initialization of C++ classes and data structures. Its structure is depicted on Figure 4.1. The structure of the input is described by the Input Types Tree (ITT) of (usually static) objects which follows the structure of the classes. The data from an input file are read by apropriate reader, their structure is checked against ITT and they are pushed into the Internal Storage Buffer (ISB). An accessor object to the root data record is the result of the file reading. The data can be retrieved through accessors which combine raw data stored in in IBS with their meaning described in ITT. ITT can be printed out in various formats providing description of the input structure both for humans and other software.

Currently, the JSON input file format is only implemented and in fact it is slight extension of the JSON file format. On the other hand the data for initialization of the C++ data structures are coded in particular way. Combination of this extension and restriction of the JSON file format produce what we call CON (C++ object notation) file format.

4.1.1 JSON for humans

Basic syntax of the CON file is very close to the JSON file format with only few extensions, namely:

- You can use C++ (or JavaScript) comments. One line comments `//` and multi-line comments `/* */`.
- The quoting of the keys is optional if they do not contain spaces (holds for all CON keys).
- You can use equality sign `=` instead of colon `:` for separation of keys and values in JSON objects.

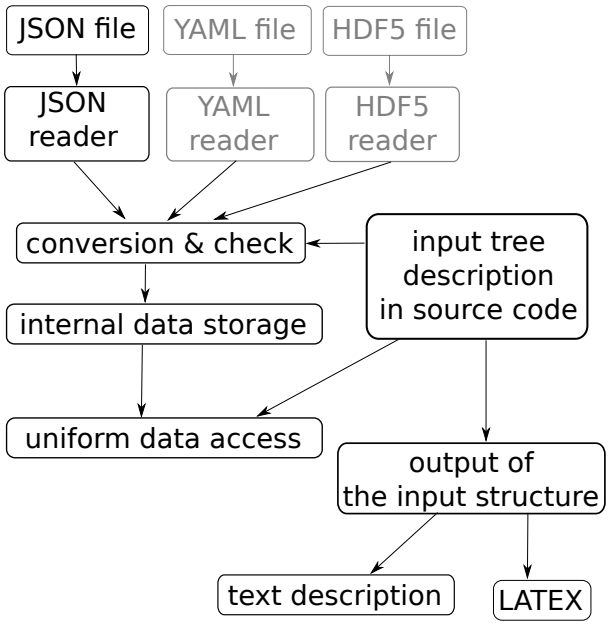


Figure 4.1: Sturucture of the input subsystem. Grey boxes are not implemented yet.

- You can use any whitespace to separate tokens in JSON object or JSON array.

The aim of these extensions is to simplify writing input files manually. However these extensions can be easily filtered out and converted to the generic JSON format. For the description of the JSON format we refer to `http://www.json.org/`.

4.1.2 CON constructs

The CON file format constructs are designed for initialization of C++ strongly typed variables. The primitive data types can be initialized from the primitive CON constructs:

- *Bool* — initialized from the JSON keywords `true` and `false`.
- *Double*, *Integer* — initialized from JSON numeric data.
- *String*, *FileName*, *Selections* — initialized from JSON strings

Selections are typed like the C++ enum types that are initialized from them. Various kind of containers can be initialized by the *Array* construct, that is an JSON array with elements of the same CON type. The C++ structures and classes can be initialize from the *Record* construct, which is represented by a JSON object. However, in constrast to JSON, these Records have different types in similar way as the strong typed C++ structures. The types are described by ITT of the particular program which can be printed out in several formats, in particular description of ITT for Flow123d forms content of Chapter 9. In order to allow certain kind of polymorphism, we introduce also the *AbstractRecord* construct, where the type of the record is not given by ITT but can be chosen as part of the input.

4.1.3 CON special keys

All keys in Records should be in lower case, possibly using digits and underscore. The keys all in upper case are reserved for special function in the CON file. These are:

TYPE key :

```
TYPE=<Selection of AbstractRecord>
```

Is used to specify particular type of an AbstractRecord. This way you can choose which particular implementation of an abstract C++ class should be instantiated. The value of the key is a string from the Selection that consists of names of Records that was declared as descendants of the AbstractRecord.

REF key :

```
{ REF=<address> }
```

The record in input file that contains only the key `REF` is replaced by the JSON entity that is referenced by the `<address>`. The address is a string with format similar to UNIX path, i.e. with grammar

```
<address> = <address> / <item>
          = <item>
          = <null>
<item> = <index>
       = <key>
       = ..
```

where `index` is non-negative integer and key is valid CON record key (lowercase, digits, underscores). The address can be absolute or relative identification of an entity. The relative address is relative to the entity in which the reference record is contained. One can use two dots `".."` to move to parent entity.

Example:

```
mesh={
        file_name="xyz"
}
array=[
        {x=1 y=0}
        {x=2 y=0}
        {x=3 y=0}
]
outer_record={
        output_file="x_out"
        inner_record={
                output_file={REF="../output_file"} // value "x_out"
        }
        x={REF="/array/2/x"}                       // value "3"
        f_name={REF="/mesh/file_name"}             // value "xyz"
}
```

4.1.4 Record types

A Record type is given by the set of key specifications, which in turn consist from: key name, type of value 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.

Implicit creation of composed entities

Consider a Record type in which all keys have default values (possibly except one). Then the specification of the Record can contain a *key for default construction*. User can specify only the value of this particular key instead of the whole record, all other keys are initialized from its default values. Moreover, an AbstractRecord type may have a default value for the `TYPE` key. This allows to express simple tasks by simple inputs but still make complex inputs possible. Similar functionality holds for arrays. If the user sets a non-array value where an array is expected the reader provides an array with a unique element holding the given value.

4.2 Important Record types of Flow123d input

4.2.1 Mesh record

The mesh record provides initialization for the computational mesh consisting of points, lines, triangles and tetrahedrons in 3D 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 either by ID number or by string label. 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*. User can create new labeled regions through the key `regions`, the new region can be specified either by its ID or by list of IDs of its elements. The latter possibility overrides original region assigned to the elements, which can be useful for specification of small areas "ad hoc". The key `sets` allows specification of sets of regions in terms of list of region IDs or labels and basic set operations. The difference between regions and sets is that regions form disjoint covering of elements, the sets, however, may overlap. There are three predefined region sets: "ALL", "BOUNDARY", "BULK".

4.2.2 Field records

A general time and space dependent, scalar, vector, or tensor valued function can be specified through the family of abstract records Field $R^m -> \mathcal{S}$, where m is currently always $m = 3$ and $\mathcal{S}$ is a specification of the target space, which can be:

- $\mathcal{T}$ — scalar valued field, with scalars of type $\mathcal{T}$
- $\mathcal{T}[d]$ — vector valued field, with vector of fixed size d and elements of type $\mathcal{T}$
- $\mathcal{T}[\mathtt{n}]$ — vector valued field, with vector of variable size (given by some input) and elements of type $\mathcal{T}$
- $\mathcal{T}[d, d]$ — tensor valued field, with square tensor of fixed size and elements of type $\mathcal{T}$

the scalar types can be

- **Real** — scalar real valued field
- **Int** — scalar integer valued field
- **Enum** — scalar non negative integer valued field, should be convertible to appropriate C++ enum type

Each of these abstract record has the same set of descendants which implement various algorithms to specify and compute values of the field. These are

FieldConstant — field that is constant in space

FieldFormula — field that is given by runtime parsed formula using x, y, z, t coordinates. The Function Parser library is used with syntax rules described here.

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:

```
constant_scalar_function = 1.0
// is same as
constant_scalar_function = {
  TYPE=FieldConstant,
  value=1.0
}

conductivity_tensor = [1 ,2, 3]
// is same as
conductivity_tensor = {
    TYPE=FieldConstant,
    value=[[1,0,0],[0,2,0],[0,0,3]]
}

concentration = {
    TYPE=FieldFormula,
    value="x+y+z"
}
//is same as (provided the vector has 2 elements)
concentration = {
    TYPE=FieldFormula,
    value=["x+y+z", "x+y+z"]
}
```

4.2.3 Field data for equations

Every equation record has keys `bulk_data` and `bc_data`. Both have the same structure, however, the first one is intended to set the bulk fields (on bulk regions) while the second serves for initialization of the boundary fields (on boundary regions). These keys contains an array of region-time initialization records like the BulkData record of the DarcyFlow equation. Every such record specify fields on particular region (keys `region` and `rid`) or on a region set (key `r_set`) starting from the time specified by the key `time`. The array is processed sequentially and latter values overwrites the previous ones. Times should form a non-decreasing sequence.

Example:

```
bulk_data = [
    { // time=0.0  - default value
        r_set="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={
```

```
            // from time=1.0 we switch to the linear function in time
            TYPE=FieldFormula
            value="2+t"
        }
    }
    {   time=2.0
        region="2d_part"
        conductivity={
            // from time=2.0 we switch to elementwise field, but only
            // on the region "2d_part"
            TYPE=FieldElementwise
            gmsh_file="./input/data.msh"
            field_name="conductivity"
        }
    }
]
```

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:

http://geuz.org/gmsh/doc/texinfo/gmsh.html#MSH-ASCII-file-format

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 aritmetic sequance. Coordinates are float numbers.

`Elements` : Element IDs are unique but need not to form an aritmetic 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 folowed by field data on individual elements. Flow123d assumes that elements are sorted by `element ID`, but doesn't need to form a continuos sequence.

4.4 Output files

Flow123d support output of scalar and vector 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 softwares. However, our primal goal is to support data transfer into the Paraview visualization software. See key `format`.

4.4.1 Output data fields of water flow module

Water flow module provides output of four data fields.

pressure on elements Pressure head in length units $[L]$ piecewise constant on every element. This field is directly produced by the MH method and thus contains no postprocessing error.

pressure in nodes Same pressure head field, but interpolated into $P1$ continuous scalar field. Namely you lost dicontinuities on fractures.

velocity on elements Vector field of water flux volume unit per time unit $[L^3/T]$. For every element we evaluate discrete flux field in barycenter.

piezometric head on elements Piezometric head in length units $[L]$ 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`).

4.4.2 Output data fields of transport

Transport module provides output only for concentrations (in mobile phase) as a field piecewise constant over elements. There is one field for every substance and names of those fields contain names of substances given by key `substances`. The physical unit is mass unit over volume unit $[M/L^3]$.

4.4.3 Auxiliary output files

Profiling information

On every run we collect some basic profiling informations. 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.

Water flux information

File contains water flow balance, total inflow and outflow over boundary segments. Further there is total water income due to sources (if they are present).

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 fo 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, infulence 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

Chapter 5

Test and tutorial problems (WORK IN PROGRES)

Chapter 6

Units

Quantity	**Unit**
lenght	L
time	T
conductivity	
concentration	
diffusivity	

Table 6.1: The table of units used in the document.

Chapter 7

Tests

7.1 Test 01 – Steady flow

This test considers steady Darcy flow in a cube domain which is cutted by 2D fractures which are further separated by a 1D channel in their cross section. The multidimensional connections between 1D, 2D and 3D elements are involved in the computation. Dirichlet boundary condition is prescribed for flow.

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid

Geometry and boundary conditions

A cube with its side 1.0 L is cutted by two diagonal 2D fractures which are further separated by a 1D channel in their cross section.

Geometry parameters are defined for different physical domains. Thickness of the 2D fractures is set to 1.0 L and the area of the cross section is set to 1.0 L^2. These parameters are unrealistic (the side of the cube is 1.0 L long) but it is compensated in the equation in the fraction with conductivity.

There are only simple Dirichlet boundary conditions. Pressure gradient in direction from one corner of the cube to the oposite corner is applied on all boundary faces of all dimensions.

Parameters

- **Cross section area:** 1D channel is set to 1.0 L^2.
- **Thickness:** 2D fractures are set to 1.0 L.
- **Conductivity:** The conductivity of materials:
  - 1D channel is set to $K = 10$.
  - 2D fractures are set to $\mathbf{K} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

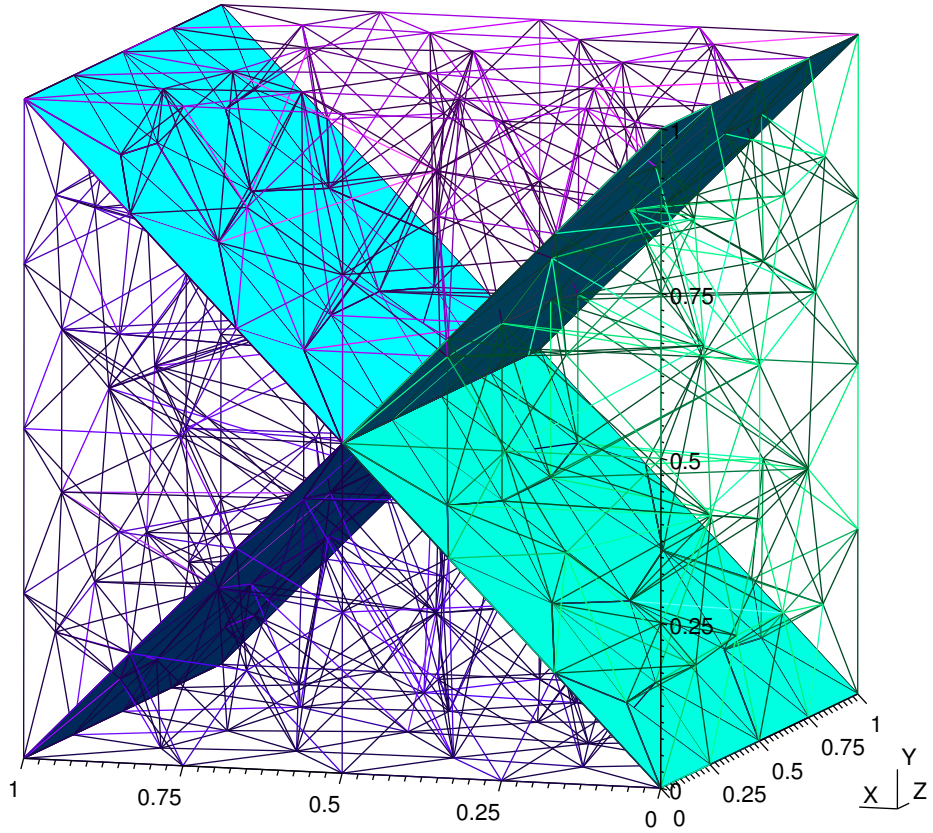


Figure 7.1: Test 01 – mesh

– cube material is set to $\mathbf{K} = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{pmatrix}$.

- There is no transport so there are not any other parameters.

Verification

This test verifies solving steady Darcy flow by mixed hybrid method. There are different dimensional connections which are 2D-3D connection between the cube and the flat fractures and 1D-2D connection between the 1D channel and the two flat fractures in their crossection.

7.2 Test 02 – Steady flow in 2D and transport

This test involves steady Darcy flow in 2D, connections of 1D-2D elements, Dirichlet boundary condition for flow and transport, transport of two substances with zero initial condition for concentration. There are acutally two different cases computed in this test. Dual porosity and sorption features in explicit transport. Dispersion is defined in implicit transport.

The coefficient of diffusive transfer through a fracture (means between the fracture and

the surrounding material) is set to zero so the substance cannot be diffused through the fracture's boundary.

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid
- *secondary equation* – transport operator splitting (explicit), discontinuous Galerkin method (implicit)

Geometry and boundary conditions

The domain is two-dimensional slice through a part of a relief which involves several one-dimensional fractures.

Simple Dirichlet flow boundary condition is defined on left and right side where pressure heads are prescribed. There is no flow through the upper and lower boundary of the model. This all causes a flow along the x axis.

Dirichlet boundary condition for transport is prescribed on both sides as it is for flow boundary condition and the value of concentration is 1.0 for both substances. Initial concentration of the substances is zero in the whole area.

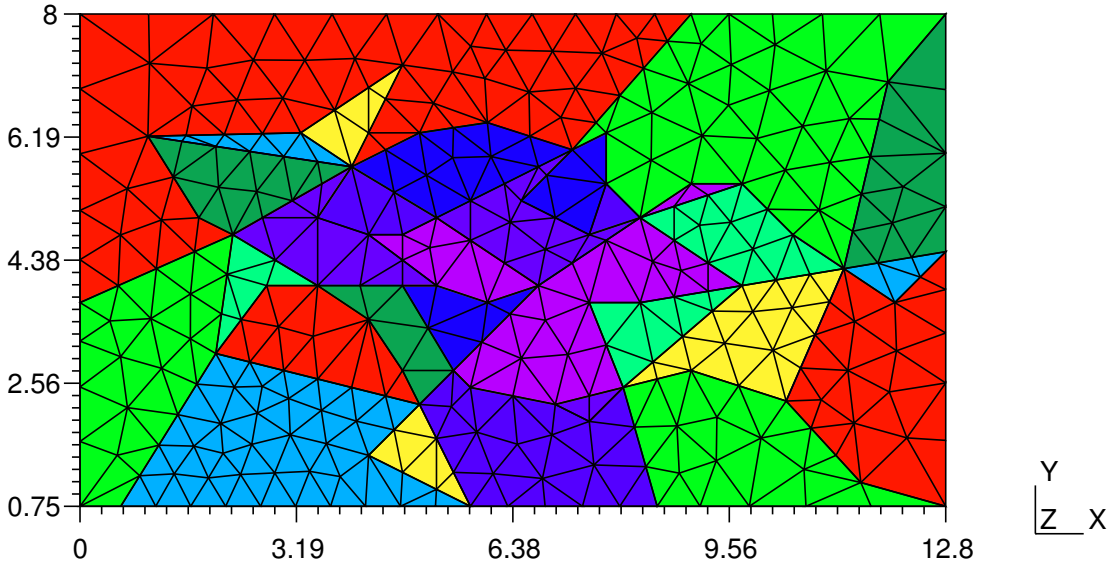


Figure 7.2: Test 02 – mesh

Parameters

The flow is steady and the transport is solved in time interval $(0, 5.0)$. The output is written every 0.5. Time parameters for implicitly computed transport are the same only initial time step is set to 0.5.

- **Cross section area:** 1D fractures are set to 1.0 L^2.
- **Thickness:** domain is set to 1.0 L.
- **Conductivity:** The conductivity of materials:
  - fracture material is set to $K = 10$.
  - plane material is set to $\mathbf{K} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.
- **Sorption:** The sorption parameters are for both materials equal:
  - linear sorption isotherm parameter of the first substance is set to $k_d = 0.02$.
  - Freundlich sorption isotherm parameters of the second substance are set to $k_f = 0.02$, $\alpha = 0.5$
- **Dual porosity:** The dual porosity parameters are for both materials equal:
  - mobile porosity coefficient is set to 0.25
  - immobile porosity coefficient is set to 0.25
  - nonequilibrium coefficient of both substances 0.01
- **Sorption fraction:** The sorption fraction parameters are for both materials equal and set to $SF = 0.5$.
- **Diffusivity coefficients:** These are not set so default values are applied $\sigma = 0$, $\alpha_l = 0$, $\alpha_t = 0$, $d_m = 1e - 6$.

Verification

This test verifies explicitly computed transport considering only convection with dual porosity and sorption and implicitly computed transport considering both convection and dispersion. Transport through 1D-2D element connections is computed in addition to the first test.

7.3 Test 03 – Steady flow in 2D and transport

This test differs from the previous one only by simpler structure of its geometry. It shows how the substace flows in the main fracture and divides in two other fractures. The substance spreads in the fractures much faster in comparision to transport in the plane.

Geometry and boundary conditions

There is a plane with side 1.0 which is cutted by fractures. The main fracture divides in two other fractures.

Parameters

The flow is steady and the transport is solved in time interval $(0, 1.0)$. The output is written every 0.01. Initial time step for transport computed implicitly is set to 0.1 and the output is written every 0.1.

Other parameters are the same as in test 02.

Verification

This test verifies the same features as the test 02 does but on a simpler geometry.

7.4 Test 05 – Darcy flow boundary conditions

There are three types of boundary conditions – Dirichlet, Neumann and Robin that are tested. All three test have the same geometry and boundary conditions are derived from the same analytical solution.

We will prescribe analytical solution $u = xy$ of Laplace equation $-\Delta u = 0$.

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid

Geometry and boundary conditions

The geometry is simple – square plane in xy coordinates with corner points [0,0] and [1,1]. Each side has its own boundary regions called `.bc_south`, `.bc_east`, `.bc_north`, `.bc_west`.

Dirichlet test. All sides have pressure prescribed. These are south: $u_D = 0$; east: $u_D = y$; north: $u_D = x$; west: $u_D = 0$.

Neumann test. Two sides have pressure prescribed for the Dirichlet boundary condition: east: $u_D = y$; west: $u_D = 0$. Two other sides have flux prescribed: south: $q_N = x$; north $q_N = -x$.

Robin test. Two sides have pressure prescribed for the Dirichlet boundary condition: east: $u_D = y$; west: $u_D = 0$. For Robin boundary condition we get from the equation boundary pressure

$$u_R = \frac{1 + \sigma_R}{\sigma_R} x. \tag{7.1}$$

We choose $\sigma_R = 0.5$ and then we get $u_R = -2x$ on the south side and $u_R = 3x$ on the north side.

Parameters

- **Conductivity:** on region `plane` is 1.0.

- **Thickness:** on region `plane` is by default 1.0 L.
- There are no other regions, no transport so there are not any other parameters.

Verification

This test verifies prescribing different types boundary conditions.

7.5 Test 06 – Coupling between dimensions in Darcy flow

There are two tests – `flow_32d.con` for compatible coupling between 3D-2D and `flow_21d.con` for compatible coupling between 2D-1D.

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid

We will discuss both the geometry and parameters at once.

3D-2D. There is a cube with vertices [0,0,0] [-1,-1,-1] which couples with a 2D crack in the bottom (i.e. $z = -1$). We consider solution of the piezometric head in the cube $p_3(x, y, z) = z$. We will use it to prescribe Dirichlet boundary condition on the non-coupling parts of the cube. There are no sources of a flow in the cube. We can write the outflow through the bottom of the cube in following term $q_{32} = \mathbf{q_3} \cdot \mathbf{n} = (-K_3 \nabla p_3(x, y, -1)) \cdot \mathbf{n}$, where $\mathbf{n} = (0, 0, -1)$. To obtain Laplace equation with zero right hand side on the 2D crack, we prescribe a new source term (7.2) that eliminates the inflow coming from the cube.

$$\begin{aligned} F_2 &= \delta_2 f_2 + q_{32} = 0 \\ f_2(x, y) &= -\frac{q_{32}}{\delta_2}. \end{aligned} \tag{7.2}$$

When $\delta_2 = 10$, $K_3 = 2$ are choosed then the source term is equal $f_2 = -0.2$. Homogenous Neumann condition is prescribed on the boundary of the fraction (zero outflow and inflow).

From the flow coupling equation we can get the pressure (7.3) on the crack which we can verify.

$$\begin{aligned} \sigma_{32}(p_3(x, y, -1) - p_2(x, y)) &= q_{32} \\ p_2(x, y) &= -\frac{q_{32}}{\sigma_{32}} + p_3(x, y, -1). \end{aligned} \tag{7.3}$$

When $\sigma = 1$ is set then the pressure in the crack is constant and equal $p_2(x, y) = -3$.

2D-1D. The other tests geometry consists of a 2D crack in xy plane with vertices [0,0] [1,1] and a 1D channel coupling with the crack on the left side ($x = 0$). Everything is then analogical to the first test. We consider solution of the pressure in the crack $p_2(x, y) = x$. We will use it to prescribe Dirichlet boundary condition on the non-coupling parts of

the crack. There are no sources of a flow in the crack. We can write the outflow through the left side of the crack in following term $q_{21} = \mathbf{q_2} \cdot \mathbf{n} = (-\delta_2 K_2 \nabla p_2(0, y)) \cdot \mathbf{n}$, where $\mathbf{n} = (-1, 0)$. To obtain Laplace equation with zero right hand side on the channel, we prescribe a new source term (7.4) that eliminates the inflow coming from the crack.

$$\begin{aligned} F_1 &= \delta_1 f_1 + q_{21} = 0 \\ f_1(x, y) &= -\frac{q_{21}}{\delta_1}. \end{aligned} \tag{7.4}$$

When $\delta_1 = 20$, $\delta_2 = 10$ and $K_2 = 5$ are choosed then the source term is equal $f_2 = -2.5$. Homogenous Neumann condition is prescribed on the boundary of the fraction (zero outflow and inflow).

From the flow coupling equation we can get the pressure (7.5) on the plane which we can verify.

$$\begin{aligned} \delta_2 \sigma_{21}(p_2(0, y) - p_2(y)) &= q_{21} \\ p_1(y) &= -\frac{q_{21}}{\delta_2 \sigma_{21}} + p_2(0, y). \end{aligned} \tag{7.5}$$

When $\sigma_{21} = 1$ is set then the pressure in the crack is constant and equal $p_1(y) = -2.5$.

Verification

This test verifies correct communication between dimensions 3D-2D and 2D-1D in compatible coupling. We can also observe the results in the `water_balance` file. We should see that all the inflow to the cube should be fully compensated by the negative source. Homogenous Neumann boundary condition is prescribed on the crack so there should be no outflow or inflow through the boundary of the crack. The situation in the second test is similar. The inflow to the crack should be fully compensated by the negative source in the channel which should have no other inflow or outflow.

7.6 Test 08 – Steady Darcy flow with source

This test is aimed at verifying steady Darcy flow with source which is prescribed by space function formula. This formula is processed by the function parser.

We will solve Laplace equation $-\Delta u = f$ where source f is prescribed by function: $f = 2(1 - y^2) + 2(1 - x^2)$.

We can easily prove that the analytic solution is $u = (1 - x^2)(1 - y^2)$ by replacing it in the Laplace equation.

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid

Geometry and boundary conditions

The domain is a square with opposite vertices $[-1, -1]$ and $[1, 1]$. Zero dirichlet boundary condition is prescribed on all boundaries – along the circumference of the square.

Parameters

- **Conductivity:** The conductivity of plane material is 1.0.
- There are no other materials, no transport so there are not any other parameters.

Verification

As it was mentioned above, this test mainly verifies that the function parser works properly. The source formula to be parsed is given in the key `source_formula`. The solution (pressure) is a paraboloid with a top in $[0, 0, 1]$.

7.7 Test 10 – Unsteady flow in 2D

Unsteady flow in 2D domain is simulated in this test and is computed by both mixed hybrid and lumped mixed hybrid method. No transport is involved.

- *problem type* – sequential coupling,
- *primary equation* – unsteady mixed hybrid, unsteady lumped mixed hybrid

Geometry and boundary conditions

The domain is a square with oposite vertices $[0, 0]$ and $[1, 1]$. Different Dirichlet boundary condition for flow is prescribed on two opposite sides – 0.0 on the left and 100.0 on the right.

Parameters

The flow is solved in time interval $(0, 0.5)$ with step 0.01. The output is written every 0.1.

- **Conductivity:** The conductivity of plane material is 0.02.
- Initial pressure is set to zero everywhere.
- There are no other materials, no transport so there are not any other parameters.

Verification

This test verifies two different numerical methods – the problem is computed by both mixed hybrid and lumped mixed hybrid method.

7.8 Test 11 – Radioactive decay chain with more branches

8 isotopes are members of considered decay chain with three branches. Transport boundary conditions does not matter because zero presure gradient is considered. Final concentrations of all isotopes except C decrease to zero after 20 time steps, whereas C concentration grows to 0.36.

$$E \to D \to F \to B \quad \begin{matrix} 0.2B \to A & A \to G & \\ 0.6B \to H & H \to G & G \to C \\ 0.2B \to G & & \end{matrix}$$

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid
- *secondary equation* – transport operator splitting
- *reactions* – linear reactions

Geometry

The domain is a prism which base is a right-angled triangle with its ordinates 3.0 units long. There are then only three tetrahedron elements in the mesh.

Parameters

The flow is steady and the transport is solved in time interval $(0, 10.0)$. The output is written every 0.5.

Half-lives are equal to 0.5 for all isotopes. Initial concentrations are summarized in the table below:

isotop	A	B	C	D	E	F	G	H
initial concentration	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08

Verification

7.9 Test 12 – Radioactive decay

There are actually two tests of the radioactive decay. The first one considers first order reaction of two isotopes determined by kinetic constant and the other one describes radioactive decay chain of three isotopes.

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid

- *secondary equation* – transport operator splitting
- *reactions* – linear reactions

Geometry and boundary conditions

The domain is a prism which base is a right-angled triangle with its ordinates 3.0 units long. There are then only three tetrahedron elements in the mesh.

There are two Dirichlet boundary conditions for flow prescribed.

- **Conductivity:** The conductivity of the prism material is 0.01.
- There is no other parameter for flow or transport.

7.9.1 First order reaction determined by kinetic constant

The only linear reaction between D and F substances.

$$D \xrightarrow{k} F$$

Parameters

The flow is steady and the transport is solved in time interval $(0, 10.0)$. The output is written every 0.5.

- **Substances:** 6 substances to be transported – A, B, C, D, E, F
- **Kinetic constant:** $k = 0.277258872$

Verification

7.9.2 Radioactive decay chain

The considered radioctive decay chain is:

$$D \xrightarrow{t_{1/2,D}} F \xrightarrow{t_{1/2,F}} B$$

Parameters

Time parameters are the same as they are above.

- **Substances:** 6 substances to be transported – A, B, C, D, E, F
- **Decay half-lives:** $t_{1/2,D} = t_{1/2,F} = 2.5$

7.10 Test 13 – Solute mixing on the edge

This test realizes mixing of substances on the edges of planes and also does quantitative test on a trivial transport problem. The problem is computed with both explicit and implicit transport.

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid
- *secondary equation* – transport operator splitting (explicit), discontinuous Galerkin method (implicit)

Geometry and boundary conditions

The domain is a fork where the main branch of length 5 with the incoming solute is in the xy plane. Then it is divided into two other branches of length $5\sqrt{2}$, one with positive and the another with negative z coordinate. There are different conductivities in each branch.

Dirichlet boundary conditions for flow and transport are prescribed at the beginning of the main plane ($x = 0$) and at the end of the secondary branches ($x = 10$).

flow: $h_D = -x - z + 10.0$ which gives 10.0 at point [0,0,0] and ± 5 at points [10,0,$\mp$5]
transport: concentration is 1.0 at point [0,0,0] and 0.0 at points [10,0,$\mp$5]

Initial concentration of the substances is zero in the whole area.

Parameters

The flow is steady and the transport is solved in time interval $(0, 100.0)$. The output is written every 0.5. Time parameters for implicitly computed transport are the same only initial time step and output time is set to 5.0.

- **Thickness:** all planes are set to 1.0 L.
- **Conductivity:** The conductivity of materials (isotropic planes):
  - main branch (material num. 17): $K = 1$.
  - branch (positive z, material num. 18): $K = 0.1$.
  - branch (negative z, material num. 19): $K = 0.1$.
- **Diffusivity coefficients:** are used in implicit transport wth dispersion. Default parameters are set ($d_m = 1e - 6$, others are zero, see manual or parameters in test02 in 7.2).

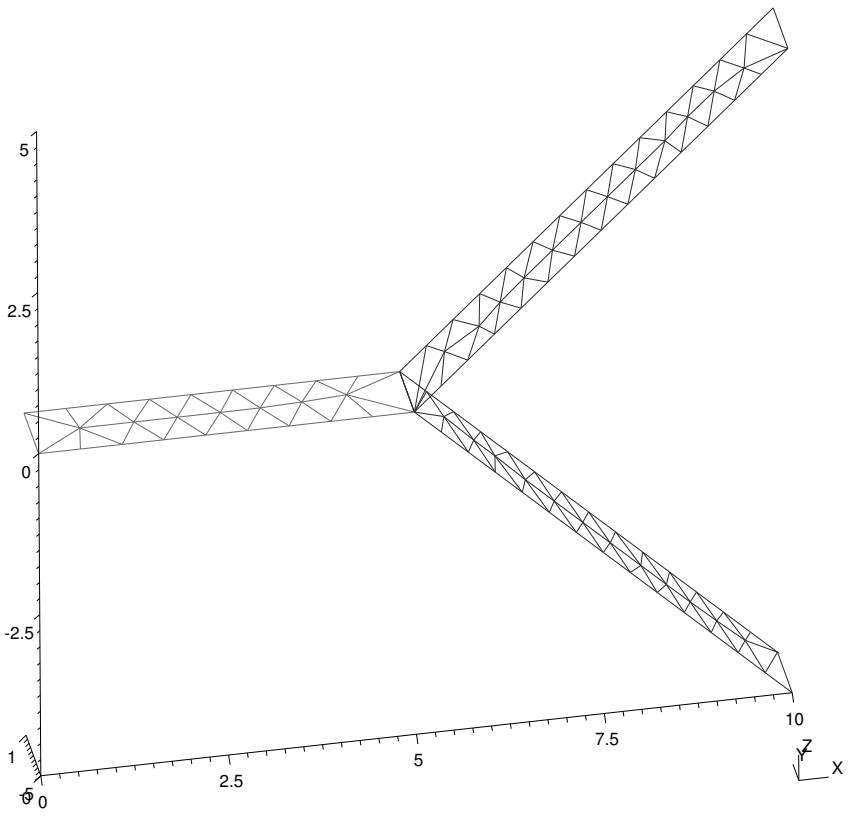


Figure 7.3: Test 13 – mesh

Verification

7.11 Test 14 – Variable transport boundary condition

There is considered a time variable boundary condition for transport in this test. Steady flow with constant velocity is caused by a pressure gradient from one side of a 2D strip to another. Dirichlet boundary condition for transport evolving in time is prescribed on the right side.

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid
- *secondary equation* – transport operator splitting (explicit), discontinuous Galerkin method (implicit)

Geometry and boundary conditions

Flow boundary condition is prescribed all around the plane and is equal x and causes constant flow from right to left (pressure prescribed on the upper and lower sides are along x equal so have no influence). Transport boundary condition has the same prescribtion as flow only the values evolves in time.

Initial concentration is zero on the whole plane. Two pulses of nonzero concentration are applied on the boundary. The changes of the boundary condition at specified times are shown in the following table:

time	0	1	3	6	7
concentration	0	20	0	40	0

- **Thickness:** all planes are set to 1.0 L.
- **Conductivity:** The conductivity of material (isotropic plane): $K = 0.1$.
- **Diffusivity coefficients:** are used in implicit transport wth dispersion. Default parameters are set except $dg_penalty = 1e-4$ (see manual or parameters in test02 in 7.2).

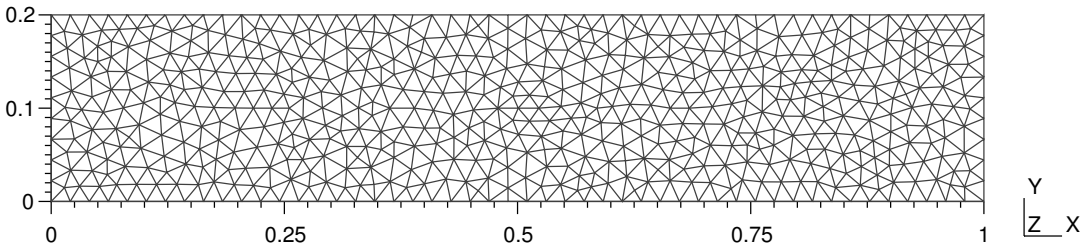


Figure 7.4: Test 14 – mesh

Parameters

The flow is steady and the transport is solved in time interval $(0, 10.0)$. The output is written every 1.0. Time parameters for implicitly computed transport are the same only initial time step is set to 1.0.

Verification

7.12 Test 15 – Unsteady flow with transport

Transport of a single pulse of concentration moving along a 2D strip is solved. This test involves unsteady flow computed by lumped hybrid method, transport is solved both with explicit and implicit (involves dispersion) scheme.

- *problem type* – sequential coupling,
- *primary equation* – unsteady lumped mixed hybrid
- *secondary equation* – transport operator splitting (explicit), discontinuous Galerkin method (implicit)

Geometry and boundary conditions

The domain is a 2D strip with dimensions 1.0x16.0. Zero Dirichlet boundary for flow is prescribed at $x = 0$, zero Neumann boundary is elsewhere.

Dirichlet transport boundary condition is set on the left side to 10.0 only at the beginning. Then is this boundary condition zero.

Parameters

Initial pressure is zero everywhere. The source is prescribed with function $f = -x$ along the strip.

- **Thickness:** all planes are set to 1.0 L.
- **Conductivity:** The conductivity of material (isotropic plane): $K = 1.0$.
- **Source formula:** $f = -x$
- **Diffusivity coefficients:** are used in implicit transport wth dispersion. Default parameters are set ($d_m = 1e - 6$, others are zero, see manual or parameters in test02 in 7.2).

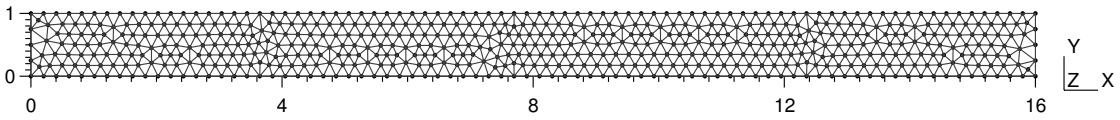


Figure 7.5: Test 15 – mesh

Verification

The test is similar to the test 10 but here in addition the computation of a transport in an unsteady flow field is verified.

7.13 Test 16 – Substance concentration source in transport

This test include a source of concentration of a substance. The domain is a 2D strip in vertical direction. There is a steady flow with constant velocity in the vertical direction. Two sources are situated on two elements at the top of the strip and the substance is transported down along the strip. The concentration values of the sources are defined in the `tso` input file.

- *problem type* – sequential coupling,

- *primary equation* – steady mixed hybrid
- *secondary equation* – transport operator splitting

Geometry

Parameters

Verification

7.14 Test 17 – Radioactive decay – Pade approximation

This test solves radioactive decay chain of five isotopes using Pade approximation. The considered radioctive decay chain is:

$$A \xrightarrow{t_{1/2,A}} B \xrightarrow{t_{1/2,B}} C \xrightarrow{t_{1/2,C}} D \xrightarrow{t_{1/2,D}} E$$

- *problem type* – sequential coupling,
- *primary equation* – steady mixed hybrid
- *secondary equation* – transport operator splitting
- *reactions* – Pade approximation

Geometry

The geometry and material and transport parameters are the same as in test 12.

Parameters

- **Substances:** 5 substances to be transported – A, B, C, D, E
- Polynomial degree of the nominator and the denominator of Pade approximation is 3.
- **Decay half-lives:**

$t_{1/2,A}$	$t_{1/2,B}$	$t_{1/2,C}$	$t_{1/2,D}$
1.3863	2.3105	1.5403	1.1552

Verification

7.15 Test 18 – Diffusion through fractures

This test is aimed at transport caused just by diffusion.

There is a triangular domain with zero pressure everywhere so no flow is present. Triangular element with high concentration of a substance lies in the middle of the domain and its sides neighbour with fractures. The coeffients of molecular diffusion and diffusive transfer through fractures are the parameters of the implicit transport and are set in the configuration file.

Geometry

Parameters

Verification

Chapter 8

Comparision of versions

In this chapter we are comparing the computing efficiency of released versions of Flow123d. We provide some of the time profiling information gathered from benchmarks, point out and discuss the differences umong different versions.

We use the class `Profiler` to get time profiling information – measured times that are taken by running different pieces of code. Selected problems are solved with different versions and different number of processors is used in parallel computation also to see scaling efficiency.

Two consecutively released versions 1.6.7 and 1.7.0 has been profiled so far. Benchmarks are computed on cluster 'hal' in the Supercomputing Centre of Czech Technical University.

8.1 Problem of transport in Melechov region

date	June 6, 2013
mesh size	56355 elements
compared versions	1.6.7 (rev. 2392) flow123d/branches/1.6.7
	1.7.0 (rev. 2416) flow123d/trunk

Table 8.1: Benchmark general information

This benchmark is based on the real data. We are solving transport of a single substance from the concentration source in the Melechov region which is considered as one of the localities suitable for building nuclear waste deposit. The results of profiling can be seen in the table 8.2. One can see in the header of the table that the problem was computed several times on different number of proccessors, with both versions 1.6.7 and 1.7.0. Number of calls of different routines and maximum amount of time which is spent on one of the proccessor during the routine *Tmax* have been measured.

- **Assembly.** The amount of time needed for *assembly* of the mixed hybrid system (in *Darcy constructor*) was mainly increased by moving some of the precomputations from mesh reading routines directly into the assembly proccess. This is the part which we can optimize in the future and futhermore we would like to make

it faster by implementing numerical integration using new system of classes for handling degrees of freedom, finite element values etc.

- **Mesh reading.** Another thing that is slowing down the computation is reading of the mesh. Version 1.7.0 seems to be slower because the neighboring of elements is computed directly in runtime. By contrast the older version uses still the utility 'NGH' to generete neighboring file which is used later (this procedure is not measured). We can also see along the row *Reading mesh* that this routine is not scaling at all. Until we implement classes for handling the mesh in parallel way this will not be any faster.

- **Solving MH System.** We can see the increase in time needed for solving mixed hybrid system. The computation of Schur complements needs to be optimized.

- **Convection matrix assembly.** In the *convection matrix assembly* routine we can see improvement from version 1.6.7 to 1.7.0. This routine scales better in newer version and this probably can be still farther optimized.

- **Transport Operator Splitting – one step.** It seems the newer version 1.7.0 is a little bit faster. The difference is caused by the change of computing the time step of convection. There are fewer convection steps in version 1.7.0 so the routine does not last so long.

- **Output.** There two parts of the output – output of the Darcy flow computation (row *Darcy output*) and output of the transport operator splitting method (row *TOS-output*). We can see that both have been slowed down (output of water flow almost twice) and that both of them are not scaling at all. The problem of slowing down is currently being solved. Next step in parallelism will be taken as soon as we have parallel mesh available.

8.2 Transport sources

The problem of transport sources can be viewed in different ways. We can prescribe a source term in a region of a mesh or we can cut off that region from the mesh and then prescribe a Dirichlet boundary condition on the created boundary. The second way was used in some models when transport sources had not been implemented in Flow123d yet.

In this section we would like to show the user the comparision of both techniques used on the real problem.

proccessors	1				2				4				8			
version	1.6.7		1.7.0		1.6.7		1.7.0		1.6.7		1.7.0		1.6.7		1.7.0	
measurement	calls	Tmax	calls	Tmax	calls	Tmax	calls	Tmax	calls	Tmax	calls	Tmax	calls	Tmax	calls	Tmax
Whole Program	1	98.23	1	86.92	1	31.17	1	35.48	1	21.58	1	26.42	1	18.39	1	23.67
HC run simulation	1	92.83	1	79.05	1	25.87	1	29.3	1	16.36	1	20.51	1	13.25	1	17.84
convection matrix assembly	1	0.16	1	0.25	1	1.32	1	0.1	1	1.31	1	0.07	1	1.3	1	0.06
TOS-output data	5	2.3	5	2.45	5	2.31	5	2.59	5	2.32	5	2.55	5	2.32	5	2.54
Solving MH system	1	4.29	1	5.19	1	1.95	1	2.97	1	1.06	1	1.7	1	0.73	1	1.28
solve system	1	3.05	1	2.52	1	1.02	1	1.02	1	0.57	1	0.57	1	0.38	1	0.43
iteration-PETSC solver	12	3.05	20	2.52	12	1.02	13	1.01	12	0.56	12	0.57	12	0.38	12	0.42
Schur 1	1	0.9	1	2.14	1	0.67	1	1.46	1	0.36	1	0.84	1	0.26	1	0.63
schur1 - create,inverse	1	0.17	1	1.46	1	0.07	1	0.4	1	0.05	1	0.22	1	0.04	1	0.15
schur1 - form	1	0.73	1	0.69	1	0.6	1	1.06	1	0.31	1	0.63	1	0.22	1	0.48
Schur 2	1	0.33	1	0.32	1	0.25	1	0.44	1	0.14	1	0.26	1	0.1	1	0.21
TOS-one step	5	82.64	5	64.04	5	17.66	5	15.97	5	9.14	5	8.56	5	6.37	5	6.36
convection-one step	16050	82.59	13790	64	16050	17.66	13790	15.97	16050	9.13	13790	8.55	16050	6.36	13790	6.36
mat mult	16050	39.16	13790	26.02	16050	10.66	13790	9.05	16050	5.75	13790	4.67	16050	4.15	13790	3.29
data reinit			13790	1.66			13790	1.05			13790	1.05			13790	1.13
compute concentration sources	16050	43.31	13790	36.25	16050	7.31	13790	6.99	16050	3.6	13790	3.03	16050	2.52	13790	2.22
Darcy output	2	3.28	2	6.99	2	3.31	2	7.14	2	3.32	2	7.12	2	3.32	2	7.12
HC constructor	1	5.35	1	7.79	1	5.16	1	6.32	1	5.05	1	6.05	1	5.04	1	5.99
Darcy constructor	1	2	1	3.28	1	1.61	1	1.74	1	1.5	1	1.45	1	1.48	1	1.41
preallocation	1	0.45	1	0.55	1	0.13	1	0.15	1	0.08	1	0.09	1	0.06	1	0.07
data init			1	0.32			1	0.32			1	0.32			1	0.32
prepare parallel	1	0.41	1	0.35	1	0.58	1	0.62	1	0.6	1	0.65	1	0.61	1	0.71
precalculation in mesh	1	0.5			1	0.62			1	0.62			1	0.62		
assembly	1	0.57	1	2.01	1	0.2	1	0.56	1	0.12	1	0.3	1	0.09	1	0.22
Reading mesh	1	2.27	1	3.94	1	2.47	1	3.94	1	2.48	1	3.95	1	2.5	1	3.93
MESH - setup topology	1	1.71	1	0.49	1	1.9	1	0.49	1	1.91	1	0.49	1	1.92	1	0.5
GMSHReader - read mesh	1	0.51	1	3.46	1	0.51	1	3.45	1	0.51	1	3.46	1	0.51	1	3.43

Table 8.2: Benchmark results on the transport problem in Melechov region.

Chapter 9

Main input file reference

---

record: **Root**

Root record of JSON input for Flow123d.

`problem` = <*abstract type: Problem*>

Default: <*obligatory*>

Simulation problem to be solved.

`pause_after_run` = <*Bool*>

Default: false

If true, the program will wait for key press before it terminates.

`output_streams` = <*Array of record: OutputStream*>

Default: <*optional*>

Array of formated output streams to open.

---

abstract type: **Problem**

Descendants:

The root record of description of particular the problem to solve.

`SequentialCoupling`

---

record: **SequentialCoupling** implements abstract type: Problem

---

Record with data for a general sequential coupling.

`TYPE` = <*selection: Problem_TYPE_selection*>

Default: SequentialCoupling

Sub-record selection.

`description` = <*String (generic)*>

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.

`time` = <*record: TimeGovernor*>

Default: <*optional*>

Simulation time frame and time step.

`primary_equation` = <*abstract type: DarcyFlowMH*>

Default: <*obligatory*>

Primary equation, have all data given.

`secondary_equation` = <*abstract type: Transport*>

Default: <*optional*>

The equation that depends (the velocity field) on the result of the primary equation.

---

record: **Mesh**

Record with mesh related data.

`mesh_file` = <*input file name*>

Default: <*obligatory*>

Input file with mesh description.

`regions` = <*Array of record: Region*>

Default: <*optional*>

List of additional region definitions not contained in the mesh.

`sets` = <*Array of record: RegionSet*>

Default: <*optional*>

List of region set definitions. There are three region sets implicitly defined: ALL (all regions of the mesh), BOUNDARY (all boundary regions), and BULK (all bulk regions)

`partitioning` = <*record: Partition*>

Default: any_neighboring

Parameters of mesh partitioning algorithms.

---

record: **Region**

Definition of region of elements.

`name` = <*String (generic)*>

Default: <*obligatory*>

Label (name) of the region. Has to be unique in one mesh.

`id` = <*Integer [0,]*>

Default: <*obligatory*>

The ID of the region to which you assign label.

`element_list` = <*Array of Integer [0,]*>

Default: <*optional*>

Specification of the region by the list of elements. This is not recomended

---

record: **RegionSet**

Definition of one region set.

`name` = <*String (generic)*>

Default: <*obligatory*>

Unique name of the region set.

`region_ids` = <*Array of Integer [0,]*>

Default: <*optional*>

List of region ID numbers that has to be added to the region set.

`region_labels` = <*Array of String (generic)*>

Default: <*optional*>

List of labels of the regions that has to be added to the region set.

`union` = <*Array [2, 2] of String (generic)*>

Default: <*optional*>

Defines region set as a union of given pair of sets. Overrides previous keys.

`intersection` = <*Array [2, 2] of String (generic)*>

Default: <*optional*>

Defines region set as an intersection of given pair of sets. Overrides previous keys.

`difference` = <*Array [2, 2] of String (generic)*>

Default: <*optional*>

Defines region set as a difference of given pair of sets. Overrides previous keys.

---

record: **Partition** constructible from key: graph_type

Setting for various types of mesh partitioning.

`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 type: **PartTool**

Select the partitioning tool to use.

Possible values:

`PETSc` : Use PETSc interface to various partitioning tools.

`METIS` : Use direct interface to Metis.

---

selection type: **GraphType**

Different algorithms to make the sparse graph with weighted edges from the multidimensional mesh. Main difference is dealing with neighborings of elements of different dimension.

Possible values:

`any_neighboring` : Add edge for any pair of neighboring elements.

`any_wight_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_neghboring` : Add edge for any pair of neighboring elements of same dimension (bad for matrix multiply).

---

record: **TimeGovernor**

Setting of the simulation time. (can be specific to one eqaution)

`start_time` = <*Double* >

Default: 0.0

Start time of the simulation.

`end_time` = <*Double* >

Default: <*obligatory*>

End time of the simulation.

`init_dt` = <*Double [0,]*>

Default: <*optional*>

Initial guess for the time step. The time step is fixed if hard time step limits are not set.

`min_dt` = <*Double [0,]*>

Default: "Machine precision or 'init_dt' if specified"

Hard lower limit for the time step.

`max_dt` = <*Double [0,]*>

Default: "Whole time of the simulation or 'init_dt' if specified"

Hard upper limit for the time step.

---

abstract type: **DarcyFlowMH**

Descendants:

Mixed-Hybrid solver for saturated Darcy flow.

Steady_MH

Unsteady_MH

Unsteady_LMH

---

record: **Steady_MH** implements abstract type: DarcyFlowMH

Mixed-Hybrid solver for STEADY saturated Darcy flow.

`TYPE` = *<selection: DarcyFlowMH_TYPE_selection>*

Default: Steady_MH

Sub-record selection.

`n_schurs` = *<Integer [0, 2]>*

Default: 2

Number of Schur complements to perform when solving MH sytem.

`solver` = *<abstract type: Solver>*

Default: *<obligatory>*

Linear solver for MH problem.

`output` = *<record: DarcyMHOutput>*

Default: *<obligatory>*

Parameters of output form MH module.

`mortar_method` = *<selection: MH_MortarMethod>*

Default: None

Method for coupling Darcy flow between dimensions.

`bc_data` = *<Array of record: DarcyFlowMH_Steady_BoundaryData>*

Default: *<obligatory>*

`bulk_data` = *<Array of record: DarcyFlowMH_Steady_BulkData>*

Default: *<obligatory>*

---

abstract type: **Solver**

Descendants:

Solver setting.

Petsc

Bddc

---

record: **Petsc** implements abstract type: Solver

Solver setting.

`TYPE` = *<selection: Solver_TYPE_selection>*

Default: Petsc

Sub-record selection.

`a_tol` = *<Double [0,]>*

Default: 1.0e-9

Absolute residual tolerance.

`r_tol` = *<Double [0, 1]>*

Default: 1.0e-7

Relative residual tolerance (to initial error).

`max_it` = *<Integer [0,]>*

Default: 10000

Maximum number of outer iterations of the linear solver.

`options` = *<String (generic)>*

Default:

Options passed to the petsc instead of default setting.

---

record: **Bddc** implements abstract type: Solver

Solver setting.

`TYPE` = *<selection: Solver_TYPE_selection>*

Default: Bddc

Sub-record selection.

`a_tol` = *<Double [0,]>*

Default: 1.0e-9

Absolute residual tolerance.

`r_tol` = *<Double [0, 1]>*

Default: 1.0e-7

Relative residual tolerance (to initial error).

`max_it` = *<Integer [0,]>*

Default: 10000

Maximum number of outer iterations of the linear solver.

record: **DarcyMHOutput**

Parameters of MH output.

`save_step` = <*Double [0,]*>

Default: 1.0

Regular step between MH outputs.

`output_stream` = <*record: OutputStream*>

Default: <*obligatory*>

Parameters of output stream.

`velocity_p0` = <*String (generic)*>

Default: <*optional*>

Output stream for P0 approximation of the velocity field.

`pressure_p0` = <*String (generic)*>

Default: <*optional*>

Output stream for P0 approximation of the pressure field.

`pressure_p1` = <*String (generic)*>

Default: <*optional*>

Output stream for P1 approximation of the pressure field.

`piezo_head_p0` = <*String (generic)*>

Default: <*optional*>

Output stream for P0 approximation of the piezometric head field.

`balance_output` = <*output file name*>

Default: water_balance.txt

Output file for water balance table.

`subdomains` = <*String (generic)*>

Default: <*optional*>

Output stream for subdomain indices (partitioning of mesh elements) used by DarcyFlow module.

`raw_flow_output` = <*output file name*>

Default: <*optional*>

Output file with raw data form MH module.

record: **OutputStream**

Parameters of output.

`name` = <*String (generic)*>

Default: <*obligatory*>

The name of this stream. Used to reference the output stream.

`file` = <*output file name*>

Default: <*obligatory*>

File path to the connected output file.

`format` = <*abstract type: OutputFormat*>

Default: <*optional*>

Format of output stream and possible parameters.

---

abstract type: **OutputFormat**

Descendants:

Format of output stream and possible parameters.

`vtk`

`gmsh`

---

record: **vtk** implements abstract type: OutputFormat

Parameters of vtk output format.

`TYPE` = <*selection: OutputFormat_TYPE_selection*>

Default: vtk

Sub-record selection.

`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.

`compression` = <*selection: Type of compression of VTK file format*>

Default: none

Compression used in output stream file format.

---

selection type: **VTK variant (ascii or binary)**

Possible values:

`ascii` : ASCII variant of VTK file format

`binary` : Binary variant of VTK file format (not supported yet)

---

selection type: **Type of compression of VTK file format**

Possible values:

`none` : Data in VTK file format are not compressed

`zlib` : Data in VTK file format are compressed using zlib (not supported yet)

record: **gmsh** implements abstract type: OutputFormat

Parameters of gmsh output format.

`TYPE` = *<selection: OutputFormat_TYPE_selection>*

Default: gmsh

Sub-record selection.

selection type: **MH_MortarMethod**

Possible values:

`None` : Mortar space: P0 on elements of lower dimension.

`P0` : Mortar space: P0 on elements of lower dimension.

`P1` : Mortar space: P1 on intersections, using non-conforming pressures.

record: **DarcyFlowMH_Steady_BoundaryData**

Record to set BOUNDARY fields of the equation 'DarcyFlowMH_Steady'. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any DarcyFlowMH_Steady_BoundaryData record that comes later in the boundary data array.

`r_set` = *<String (generic)>*

Default: *<optional>*

Name of region set where to set fields.

`region` = *<String (generic)>*

Default: *<optional>*

Label of the region where to set fields.

`rid` = *<Integer [0,]>*

Default: *<optional>*

ID of the region where to set fields.

`time` = *<Double [0,]>*

Default: 0.0

Apply field setting in this record after this time. These times have to form an increasing sequence.

`bc_type` = *<abstract type: Field:R3 -¿ Enum>*

Default: <*optional*>

Boundary condition type, possible values:

`bc_pressure` = <*abstract type: Field:R3 -¿ Real*>

Default: <*optional*>

Dirichlet BC condition value for pressure.

`bc_flux` = <*abstract type: Field:R3 -¿ Real*>

Default: <*optional*>

Flux in Neumman or Robin boundary condition.

`bc_robin_sigma` = <*abstract type: Field:R3 -¿ Real*>

Default: <*optional*>

Conductivity coefficient in Robin boundary condition.

`bc_piezo_head` = <*abstract type: Field:R3 -¿ Real*>

Default: <*optional*>

Boundary condition for pressure as piezometric head.

`flow_old_bcd_file` = <*input file name*>

Default: <*optional*>

---

abstract type: **Field:R3 -¿ Enum** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`FieldConstant`

`FieldFormula`

`FieldPython`

`FieldElementwise`

---

record: **FieldConstant** implements abstract type: Field:R3 -¿ Enum constructible from key: value

---

R3 -¿ Enum Field constant in space.

`TYPE` = <*selection: Field:R3 -¿ Enum_TYPE_selection*>

Default: FieldConstant

Sub-record selection.

`value` = <*selection: EqData_bc_Type*>

Default: <*obligatory*>

Value of the constant field. For vector values, you can use scalar value to enter constant vector. For square NxN-matrix values, you can use: * vector of size N to enter diagonal matrix * vector of size (N+1)*N/2 to enter symmetric matrix

(upper triangle, row by row) * scalar to enter multiple of the unit matrix.

---

selection type: **EqData_bc_Type**

Possible values:

`none` : Homogeneous Neoumann BC.

`dirichlet` :

`neumann` :

`robin` :

`total_flux` :

---

record: **FieldFormula** implements abstract type: Field:R3 -¿ Enum

---

R3 -¿ Enum Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 -¿ Enum_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<String (generic)>*

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 NxN-matrix values, you can use: * array of strings of size N to enter diagonal matrix * array of strings of size (N+1)*N/2 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 and usual operators and functions.

---

record: **FieldPython** implements abstract type: Field:R3 -¿ Enum

---

R3 -¿ Enum Field given by a Python script.

`TYPE` = *<selection: Field:R3 -¿ Enum_TYPE_selection>*

Default: FieldPython

Sub-record selection.

`script_string` = *<String (generic)>*

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

`script_file` = *<input file name>*

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

`function` = *<String (generic)>*

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: **FieldElementwise** implements abstract type: Field:R3 -¿ Enum

---

R3 -¿ Enum Field constant in space.

`TYPE` = *<selection: Field:R3 -¿ Enum_TYPE_selection>*

Default: FieldElementwise

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

abstract type: **Field:R3 -¿ Real** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`FieldConstant`

`FieldPython`

`FieldFormula`

`FieldElementwise`

`FieldInterpolatedP0`

---

record: **FieldConstant** implements abstract type: Field:R3 -¿ Real constructible from key: value

---

R3 -¿ Real Field constant in space.

`TYPE` = *<selection: Field:R3 -¿ Real_TYPE_selection>*

Default: FieldConstant

Sub-record selection.

`value` = *<Double >*

Default: *<obligatory>*

Value of the constant field. For vector values, you can use scalar value to enter constant vector. For square NxN-matrix values, you can use: * vector of size N

to enter diagonal matrix * vector of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row) * scalar to enter multiple of the unit matrix.

---

record: **FieldPython** implements abstract type: Field:R3 -¿ Real

---

R3 -¿ Real Field given by a Python script.

`TYPE` = *<selection: Field:R3 -¿ Real_TYPE_selection>*

Default: FieldPython

Sub-record selection.

`script_string` = *<String (generic)>*

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

`script_file` = *<input file name>*

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

`function` = *<String (generic)>*

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: **FieldFormula** implements abstract type: Field:R3 -¿ Real

---

R3 -¿ Real Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 -¿ Real_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<String (generic)>*

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 NxN-matrix values, you can use: * array of strings of size N to enter diagonal matrix * array of strings of size (N+1)*N/2 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 and usual operators and functions.

---

record: **FieldElementwise** implements abstract type: Field:R3 -¿ Real

---

R3 -¿ Real Field constant in space.

`TYPE` = *<selection: Field:R3 -¿ Real_TYPE_selection>*

Default: FieldElementwise

Sub-record selection.

gmsh_file = <*input file name*>

Default: <*obligatory*>

Input file with ASCII GMSH file format.

field_name = <*String (generic)*>

Default: <*obligatory*>

The values of the Field are read from the $ElementData section with field name given by this key.

---

record: **FieldInterpolatedP0** implements abstract type: Field:R3 -¿ Real

---

R3 -¿ Real Field constant in space.

TYPE = <*selection: Field:R3 -¿ Real_TYPE_selection*>

Default: FieldInterpolatedP0

Sub-record selection.

gmsh_file = <*input file name*>

Default: <*obligatory*>

Input file with ASCII GMSH file format.

field_name = <*String (generic)*>

Default: <*obligatory*>

The values of the Field are read from the $ElementData section with field name given by this key.

---

record: **DarcyFlowMH_Steady_BulkData**

Record to set BULK fields of the equation 'DarcyFlowMH_Steady'. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any DarcyFlowMH_Steady_BulkData record that comes later in the bulk data array.

r_set = <*String (generic)*>

Default: <*optional*>

Name of region set where to set fields.

region = <*String (generic)*>

Default: <*optional*>

Label of the region where to set fields.

rid = <*Integer [0,]*>

Default: *<optional>*

ID of the region where to set fields.

`time` = *<Double [0,]>*

Default: 0.0

Apply field setting in this record after this time. These times have to form an increasing sequence.

`anisotropy` = *<abstract type: Field:R3 -¿ Real[3,3]>*

Default: *<optional>*

Anisotropy of the conductivity tensor.

`cross_section` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Complement dimension parameter (cross section for 1D, thickness for 2D).

`conductivity` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Isotropic conductivity scalar.

`sigma` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Transition coefficient between dimensions.

`water_source_density` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Water source density.

`init_pressure` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Initial condition as pressure

`storativity` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Storativity.

`init_piezo_head` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Initial condition for pressure as piezometric head.

---

abstract type: **Field:R3 -¿ Real[3,3]** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`FieldConstant`

FieldPython

FieldFormula

FieldElementwise

FieldInterpolatedP0

---

record: **FieldConstant** implements abstract type: Field:R3 -¿ Real[3,3] constructible from key: value

---

R3 -¿ Real[3,3] Field constant in space.

TYPE = *<selection: Field:R3 -¿ Real[3,3]_TYPE_selection>*

Default: FieldConstant

Sub-record selection.

value = *<Array [1,] of Array [1,] of Double >*

Default: *<obligatory>*

Value of the constant field. For vector values, you can use scalar value to enter constant vector. For square NxN-matrix values, you can use: * vector of size N to enter diagonal matrix * vector of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row) * scalar to enter multiple of the unit matrix.

---

record: **FieldPython** implements abstract type: Field:R3 -¿ Real[3,3]

R3 -¿ Real[3,3] Field given by a Python script.

TYPE = *<selection: Field:R3 -¿ Real[3,3]_TYPE_selection>*

Default: FieldPython

Sub-record selection.

script_string = *<String (generic)>*

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

script_file = *<input file name>*

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

function = *<String (generic)>*

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: **FieldFormula** implements abstract type: Field:R3 -¿ Real[3,3]

R3 -¿ Real[3,3] Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 -¿ Real[3,3]_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<Array [1,] of Array [1,] of String (generic)>*

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 NxN-matrix values, you can use: * array of strings of size N to enter diagonal matrix * array of strings of size (N+1)*N/2 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 and usual operators and functions.

---

record: **FieldElementwise** implements abstract type: Field:R3 -¿ Real[3,3]

R3 -¿ Real[3,3] Field constant in space.

`TYPE` = *<selection: Field:R3 -¿ Real[3,3]_TYPE_selection>*

Default: FieldElementwise

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

record: **FieldInterpolatedP0** implements abstract type: Field:R3 -¿ Real[3,3]

R3 -¿ Real[3,3] Field constant in space.

`TYPE` = *<selection: Field:R3 -¿ Real[3,3]_TYPE_selection>*

Default: FieldInterpolatedP0

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name

given by this key.

---

record: **Unsteady_MH** implements abstract type: DarcyFlowMH

Mixed-Hybrid solver for unsteady saturated Darcy flow.

`TYPE` = <*selection: DarcyFlowMH_TYPE_selection*>

Default: Unsteady_MH

Sub-record selection.

`n_schurs` = <*Integer [0, 2]*>

Default: 2

Number of Schur complements to perform when solving MH sytem.

`solver` = <*abstract type: Solver*>

Default: <*obligatory*>

Linear solver for MH problem.

`output` = <*record: DarcyMHOutput*>

Default: <*obligatory*>

Parameters of output form MH module.

`mortar_method` = <*selection: MH_MortarMethod*>

Default: None

Method for coupling Darcy flow between dimensions.

`time` = <*record: TimeGovernor*>

Default: <*obligatory*>

Time governor setting for the unsteady Darcy flow model.

`bc_data` = <*Array of record: DarcyFlowMH_Steady_BoundaryData*>

Default: <*obligatory*>

`bulk_data` = <*Array of record: DarcyFlowMH_Steady_BulkData*>

Default: <*obligatory*>

---

record: **Unsteady_LMH** implements abstract type: DarcyFlowMH

Lumped Mixed-Hybrid solver for unsteady saturated Darcy flow.

`TYPE` = <*selection: DarcyFlowMH_TYPE_selection*>

Default: Unsteady_LMH

Sub-record selection.

`n_schurs` = <*Integer [0, 2]*>

Default: 2

Number of Schur complements to perform when solving MH sytem.

`solver` = <*abstract type: Solver*>

Default: <*obligatory*>

Linear solver for MH problem.

`output` = <*record: DarcyMHOutput*>

Default: <*obligatory*>

Parameters of output form MH module.

`mortar_method` = <*selection: MH_MortarMethod*>

Default: None

Method for coupling Darcy flow between dimensions.

`time` = <*record: TimeGovernor*>

Default: <*obligatory*>

Time governor setting for the unsteady Darcy flow model.

`bc_data` = <*Array of record: DarcyFlowMH_Steady_BoundaryData*>

Default: <*obligatory*>

`bulk_data` = <*Array of record: DarcyFlowMH_Steady_BulkData*>

Default: <*obligatory*>

---

abstract type: **Transport**

Descendants:

Secondary equation for transport of substances.

`TransportOperatorSplitting`

`AdvectionDiffusion_DG`

---

record: **TransportOperatorSplitting** implements abstract type: Transport

---

Explicit FVM transport (no diffusion) coupled with reaction and sorption model (ODE per element) via operator splitting.

`TYPE` = <*selection: Transport_TYPE_selection*>

Default: TransportOperatorSplitting

Sub-record selection.

`time` = <*record: TimeGovernor*>

Default: <*obligatory*>

Time governor setting for the transport model.

`substances` = <*Array of String (generic)*>

Default: <*obligatory*>

Names of transported substances.

`sorption_enable` = <*Bool*>

Default: false

Model of sorption.

`dual_porosity` = <*Bool*>

Default: false

Dual porosity model.

`output` = <*record: TransportOutput*>

Default: <*obligatory*>

Parameters of output stream.

`reactions` = <*abstract type: Reactions*>

Default: <*optional*>

Initialization of per element reactions.

`adsorptions` = <*record: Sorptions*>

Default: <*optional*>

Initialization of per element sorptions.

`bc_data` = <*Array of record: TransportOperatorSplitting_BoundaryData*>

Default: <*obligatory*>

`bulk_data` = <*Array of record: TransportOperatorSplitting_BulkData*>

Default: <*obligatory*>

---

record: **TransportOutput**

Output setting for transport equations.

`output_stream` = <*record: OutputStream*>

Default: <*obligatory*>

Parameters of output stream.

`save_step` = <*Double [0,]*>

Default: <*obligatory*>

Interval between outputs.

`output_times` = <*Array of Double [0,]*>

Default: <*optional*>

Explicit array of output times (can be combined with 'save_step'.

`conc_mobile_p0` = <*String (generic)*>

Default: <*optional*>

Name of output stream for P0 approximation of the concentration in mobile phase.

`conc_immobile_p0` = *<String (generic)>*

Default: *<optional>*

Name of output stream for P0 approximation of the concentration in immobile phase.

`conc_mobile_sorbed_p0` = *<String (generic)>*

Default: *<optional>*

Name of output stream for P0 approximation of the surface concentration of sorbed mobile phase.

`conc_immobile_sorbed_p0` = *<String (generic)>*

Default: *<optional>*

Name of output stream for P0 approximation of the surface concentration of sorbed immobile phase.

---

abstract type: **Reactions**

Descendants:

Equation for reading information about simple chemical reactions.

`Sorptions`

`LinearReactions`

`PadeApproximant`

`Isotope`

---

record: **Sorptions** implements abstract type: Reactions

Information about all the limited solubility affected sorptions.

`TYPE` = *<selection: Reactions_TYPE_selection>*

Default: Sorptions

Sub-record selection.

`solvent_dens` = *<Double >*

Default: 1.0

Density of the solvent.

`substeps` = *<Integer >*

Default: 100

Number of equidistant substeps, molar mass and isotherm intersections

`species` = *<Array of String (generic)>*

Default: *<obligatory>*

Names of all the sorbing species

`molar_masses` = *<Array of Double >*

Default: <*obligatory*>

Specifies molar masses of all the sorbing species

`solubility` = <*Array of Double* >

Default: <*obligatory*>

Specifies solubility limits of all the sorbing species

`bulk_data` = <*Array of record: Sorption_BulkData*>

Default: <*obligatory*>

Containes region specific data necessery to construct isotherms.

---

record: **Sorption_BulkData**

Record to set BULK fields of the equation 'Sorption'. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any Sorption_BulkData record that comes later in the bulk data array.

`r_set` = <*String (generic)*>

Default: <*optional*>

Name of region set where to set fields.

`region` = <*String (generic)*>

Default: <*optional*>

Label of the region where to set fields.

`rid` = <*Integer [0,]*>

Default: <*optional*>

ID of the region where to set fields.

`time` = <*Double [0,]*>

Default: 0.0

Apply field setting in this record after this time. These times have to form an increasing sequence.

`rock_density` = <*abstract type: Field:R3 -¿ Real*>

Default: <*optional*>

Rock matrix density.

`sorption_types` = <*abstract type: Field:R3 -¿ Enum[n]*>

Default: <*optional*>

Considered adsorption is described by selected isotherm.

`mult_coefs` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

Multiplication parameters (k, omega) in either Langmuir c_s = omega * (al-

pha*c_a)/(1- alpha*c_a) or in linear c_s = k * c_a isothermal description.

`second_params` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

Second parameters (alpha, ...) defining isotherm c_s = omega * (alpha*c_a)/(1- alpha*c_a).

`mob_porosity` = <*abstract type: Field:R3 -¿ Real*>

Default: <*optional*>

Mobile porosity of the rock matrix.

---

abstract type: **Field:R3 -¿ Enum[n]** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`FieldConstant`

`FieldFormula`

`FieldPython`

`FieldElementwise`

---

record: **FieldConstant** implements abstract type: Field:R3 -¿ Enum[n] constructible from key: value

---

R3 -¿ Enum[n] Field constant in space.

`TYPE` = <*selection: Field:R3 -¿ Enum[n]_TYPE_selection*>

Default: FieldConstant

Sub-record selection.

`value` = <*Array [1,] of selection: SorptionType*>

Default: <*obligatory*>

Value of the constant field. For vector values, you can use scalar value to enter constant vector. For square NxN-matrix values, you can use: * vector of size N to enter diagonal matrix * vector of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row) * scalar to enter multiple of the unit matrix.

---

selection type: **SorptionType**

Possible values:

`none` : No sorption considered

`linear` : Linear isotherm described sorption considered.

`langmuir` : Langmuir isotherm described sorption considered

`freundlich` : Freundlich isotherm described sorption considered

record: **FieldFormula** implements abstract type: Field:R3 -¿ Enum[n]

R3 -¿ Enum[n] Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 -¿ Enum[n]_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<Array [1,] of String (generic)>*

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 NxN-matrix values, you can use: * array of strings of size N to enter diagonal matrix * array of strings of size (N+1)*N/2 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 and usual operators and functions.

record: **FieldPython** implements abstract type: Field:R3 -¿ Enum[n]

R3 -¿ Enum[n] Field given by a Python script.

`TYPE` = *<selection: Field:R3 -¿ Enum[n]_TYPE_selection>*

Default: FieldPython

Sub-record selection.

`script_string` = *<String (generic)>*

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

`script_file` = *<input file name>*

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

`function` = *<String (generic)>*

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: **FieldElementwise** implements abstract type: Field:R3 -¿ Enum[n]

R3 -¿ Enum[n] Field constant in space.

`TYPE` = *<selection: Field:R3 -¿ Enum[n]_TYPE_selection>*

Default: FieldElementwise

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

abstract type: **Field:R3 -¿ Real[n]** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`FieldConstant`

`FieldPython`

`FieldFormula`

`FieldElementwise`

`FieldInterpolatedP0`

---

record: **FieldConstant** implements abstract type: Field:R3 -¿ Real[n] constructible from key: value

---

R3 -¿ Real[n] Field constant in space.

`TYPE` = *<selection: Field:R3 -¿ Real[n]_TYPE_selection>*

Default: FieldConstant

Sub-record selection.

`value` = *<Array [1,] of Double >*

Default: *<obligatory>*

Value of the constant field. For vector values, you can use scalar value to enter constant vector. For square NxN-matrix values, you can use: * vector of size N to enter diagonal matrix * vector of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row) * scalar to enter multiple of the unit matrix.

---

record: **FieldPython** implements abstract type: Field:R3 -¿ Real[n]

R3 -¿ Real[n] Field given by a Python script.

`TYPE` = *<selection: Field:R3 -¿ Real[n]_TYPE_selection>*

Default: FieldPython

Sub-record selection.

`script_string` = *<String (generic)>*

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

`script_file` = *<input file name>*

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

`function` = *<String (generic)>*

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: **FieldFormula** implements abstract type: Field:R3 -¿ Real[n]

R3 -¿ Real[n] Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 -¿ Real[n]_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<Array [1,] of String (generic)>*

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 NxN-matrix values, you can use: * array of strings of size N to enter diagonal matrix * array of strings of size (N+1)*N/2 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 and usual operators and functions.

---

record: **FieldElementwise** implements abstract type: Field:R3 -¿ Real[n]

R3 -¿ Real[n] Field constant in space.

`TYPE` = *<selection: Field:R3 -¿ Real[n]_TYPE_selection>*

Default: FieldElementwise

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

record: **FieldInterpolatedP0** implements abstract type: Field:R3 -¿ Real[n]

R3 -¿ Real[n] Field constant in space.

`TYPE` = *<selection: Field:R3 -¿ Real[n]_TYPE_selection>*

Default: FieldInterpolatedP0

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

record: **LinearReactions** implements abstract type: Reactions

Information for a decision about the way to simulate radioactive decay.

`TYPE` = *<selection: Reactions_TYPE_selection>*

Default: LinearReactions

Sub-record selection.

`decays` = *<Array of record: Substep>*

Default: *<obligatory>*

Description of particular decay chain substeps.

`matrix_exp_on` = *<Bool>*

Default: false

Enables to use Pade approximant of matrix exponential.

record: **Substep**

Equation for reading information about radioactive decays.

`parent` = *<String (generic)>*

Default: *<obligatory>*

Identifier of an isotope.

`half_life` = *<Double >*

Default: *<optional>*

Half life of the parent substance.

`kinetic` = *<Double >*

Default: *<optional>*

Kinetic constants describing first order reactions.

`products` = *<Array of String (generic)>*

Default: *<obligatory>*

Identifies isotopes which decays parental atom to.

`branch_ratios` = *<Array of Double >*

Default: 1.0

Decay chain branching percentage.

---

record: **PadeApproximant** implements abstract type: Reactions

---

Abstract record with an information about pade approximant parameters.

`TYPE` = *<selection: Reactions_TYPE_selection>*

Default: PadeApproximant

Sub-record selection.

`decays` = *<Array of record: Substep>*

Default: *<obligatory>*

Description of particular decay chain substeps.

`nom_pol_deg` = *<Integer >*

Default: 2

Polynomial degree of the nominator of Pade approximant.

`den_pol_deg` = *<Integer >*

Default: 2

Polynomial degree of the nominator of Pade approximant

---

record: **Isotope** implements abstract type: Reactions

Definition of information about a single isotope.

`TYPE` = *<selection: Reactions_TYPE_selection>*

Default: Isotope

Sub-record selection.

`identifier` = *<Integer >*

Default: *<obligatory>*

Identifier of the isotope.

`half_life` = *<Double >*

Default: *<obligatory>*

Half life parameter.

record: **TransportOperatorSplitting_BoundaryData**

Record to set BOUNDARY fields of the equation 'TransportOperatorSplitting'. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any TransportOperatorSplitting_BoundaryData record that comes later in the boundary data array.

`r_set` = *<String (generic)>*

Default: *<optional>*

Name of region set where to set fields.

`region` = *<String (generic)>*

Default: *<optional>*

Label of the region where to set fields.

`rid` = *<Integer [0,]>*

Default: *<optional>*

ID of the region where to set fields.

`time` = *<Double [0,]>*

Default: 0.0

Apply field setting in this record after this time. These times have to form an increasing sequence.

`bc_conc` = *<abstract type: Field:R3 -¿ Real[n]>*

Default: *<optional>*

Boundary conditions for concentrations.

`old_boundary_file` = *<input file name>*

Default: *<optional>*

Input file with boundary conditions (obsolete).

`bc_times` = *<Array of Double >*

Default: *<optional>*

Times for changing the boundary conditions (obsolete).

record: **TransportOperatorSplitting_BulkData**

Record to set BULK fields of the equation 'TransportOperatorSplitting'. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any TransportOperatorSplitting_BulkData record that comes later in the bulk data array.

`r_set` = *<String (generic)>*

Default: *<optional>*

Name of region set where to set fields.

`region` = *<String (generic)>*

Default: *<optional>*

Label of the region where to set fields.

`rid` = *<Integer [0,]>*

Default: *<optional>*

ID of the region where to set fields.

`time` = *<Double [0,]>*

Default: 0.0

Apply field setting in this record after this time. These times have to form an increasing sequence.

`init_conc` = *<abstract type: Field:R3 -¿ Real[n]>*

Default: *<optional>*

Initial concentrations.

`por_m` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Mobile porosity

`sources_density` = *<abstract type: Field:R3 -¿ Real[n]>*

Default: *<optional>*

Density of concentration sources.

`sources_sigma` = *<abstract type: Field:R3 -¿ Real[n]>*

Default: *<optional>*

Concentration flux.

`sources_conc` = *<abstract type: Field:R3 -¿ Real[n]>*

Default: *<optional>*

Concentration sources threshold.

`por_imm` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Porosity material parameter of the immobile zone. Vector, one value for every substance.

`alpha` = *<abstract type: Field:R3 -¿ Real[n]>*

Default: *<optional>*

Diffusion coefficient of non-equilibrium linear exchange between mobile and immobile zone (dual porosity). Vector, one value for every substance.

`sorp_type` = *<abstract type: Field:R3 -¿ Enum[n]>*

Default: <*optional*>

Type of sorption isotherm.

`sorp_coef0` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

First parameter of sorption: Scaling of the isothem for all types. Vector, one value for every substance.

`sorp_coef1` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

Second parameter of sorption: exponent(Freundlich isotherm), limit concentration (Langmuir isotherm). Vector, one value for every substance.

`phi` = <*abstract type: Field:R3 -¿ Real*>

Default: <*optional*>

Fraction of the total sorption surface exposed to the mobile zone, in interval (0,1). Used only in combination with dual porosity model. Vector, one value for every substance.

---

record: **AdvectionDiffusion_DG** implements abstract type: Transport

---

DG solver for transport with diffusion.

`TYPE` = <*selection: Transport_TYPE_selection*>

Default: AdvectionDiffusion_DG

Sub-record selection.

`time` = <*record: TimeGovernor*>

Default: <*obligatory*>

Time governor setting for the transport model.

`substances` = <*Array of String (generic)*>

Default: <*obligatory*>

Names of transported substances.

`sorption_enable` = <*Bool*>

Default: false

Model of sorption.

`dual_porosity` = <*Bool*>

Default: false

Dual porosity model.

`output` = <*record: TransportOutput*>

Default: <*obligatory*>

Parameters of output stream.

`solver` = <*abstract type: Solver*>

Default: <*obligatory*>

Linear solver for MH problem.

`bc_data` = <*Array of record: TransportDG_BoundaryData*>

Default: <*obligatory*>

`bulk_data` = <*Array of record: TransportDG_BulkData*>

Default: <*obligatory*>

`dg_variant` = <*selection: DG_variant*>

Default: non-symmetric

Variant of interior penalty discontinuous Galerkin method.

`dg_order` = <*Integer [0, 2]*>

Default: 1

Polynomial order for finite element in DG method (order 0 is suitable if there is no diffusion/dispersion).

---

record: **TransportDG_BoundaryData**

Record to set BOUNDARY fields of the equation 'TransportDG'. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any TransportDG_BoundaryData record that comes later in the boundary data array.

`r_set` = <*String (generic)*>

Default: <*optional*>

Name of region set where to set fields.

`region` = <*String (generic)*>

Default: <*optional*>

Label of the region where to set fields.

`rid` = <*Integer [0,]*>

Default: <*optional*>

ID of the region where to set fields.

`time` = <*Double [0,]*>

Default: 0.0

Apply field setting in this record after this time. These times have to form an increasing sequence.

`bc_conc` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

Boundary conditions for concentrations.

`old_boundary_file` = *<input file name>*

Default: *<optional>*

Input file with boundary conditions (obsolete).

`bc_times` = *<Array of Double >*

Default: *<optional>*

Times for changing the boundary conditions (obsolete).

---

record: **TransportDG_BulkData**

Record to set BULK fields of the equation 'TransportDG'. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any TransportDG_BulkData record that comes later in the bulk data array.

`r_set` = *<String (generic)>*

Default: *<optional>*

Name of region set where to set fields.

`region` = *<String (generic)>*

Default: *<optional>*

Label of the region where to set fields.

`rid` = *<Integer [0,]>*

Default: *<optional>*

ID of the region where to set fields.

`time` = *<Double [0,]>*

Default: 0.0

Apply field setting in this record after this time. These times have to form an increasing sequence.

`init_conc` = *<abstract type: Field:R3 -¿ Real[n]>*

Default: *<optional>*

Initial concentrations.

`por_m` = *<abstract type: Field:R3 -¿ Real>*

Default: *<optional>*

Mobile porosity

`sources_density` = *<abstract type: Field:R3 -¿ Real[n]>*

Default: *<optional>*

Density of concentration sources.

`sources_sigma` = *<abstract type: Field:R3 -¿ Real[n]>*

Default: <*optional*>

Concentration flux.

`sources_conc` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

Concentration sources threshold.

`disp_l` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

Longitudal dispersivity (for each substance).

`disp_t` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

Transversal dispersivity (for each substance).

`diff_m` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

Molecular diffusivity (for each substance).

`sigma_c` = <*abstract type: Field:R3 -¿ Real[n]*>

Default: <*optional*>

Coefficient of diffusive transfer through fractures (for each substance).

`dg_penalty` = <*abstract type: Field:R3 -¿ Real[n]*>

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 type: **DG_variant**

Type of penalty term.

Possible values:

`non-symmetric` : non-symmetric weighted interior penalty DG method

`incomplete` : incomplete weighted interior penalty DG method

`symmetric` : symmetric weighted interior penalty DG method

Bibliography

[1] Milena Číslerová and Tomáš Vogel. *Transportní procesy*. ČVUT, 1998.

[2] Ghislain De Marsily. *Quantitative hydrogeology: Groundwater hydrology for engineers*. Academic Press, New York, 1986.

[3] Patrick A Domenico and Franklin W Schwartz. *Physical and chemical hydrogeology*, volume 824. Wiley New York, 1990.

[4] Vincent Martin, Jérôme Jaffré, and Jean E. Roberts. Modeling fractures and barriers as interfaces for flow in porous media. *SIAM Journal on Scientific Computing*, 26(5):1667, 2005.

[5] RJ Millington and JP Quirk. Permeability of porous solids. *Transactions of the Faraday Society*, 57:1200–1207, 1961.