RICHY1D Manual

June 2, 2015

This software is being developed at:

Applied Mathematics I
Modelling and Simulation
Friedrich-Alexander-University Erlangen-Nürnberg

Cauerstr. 11
D-91058 Erlangen

Contents

Chapter 1

Introduction

RICHY1D is a simulation software for physical transport problems modeled by partial differential equations. RICHY1D

- works in **1 space dimension**,
- simulates **time and space dependent** problems,
- includes the solution of inverse problems to identify model parameters... and provides many more features for mathematicians and engineers, described within this manual.

The main purpose of **R. I. C. H. Y.** is, to support the evaluation of soil **r**emedi**a**tion and **ri**sk assessment scenarios. A typical situation of interest is a **c**ontaminated site, where the spreading of **ch**emicals in the soil exhibits a **h**azard for the environment. Model equations for (un-)saturated water flow and for solute transport were the first problems of **hy**drology, numerical**ly** solved with RICHY1D.

An earlier state of the software is also documented with some examples in Schneid et al. (2000). RICHY1D is an outstanding tool due to its combination of state-of-the-art mathematical techniques (like locally mass-conserving mixed hybrid finite elements) with a unique variety of advanced model descriptions (like preferential flow or carrier-facilitated multicomponent transport). To put it in a nutshell:

> **R**eally **i**nteresting **c**ontaminant **hy**drology!

At present, numerical solutions to the following single and coupled partial differential equations are implemented:

1. Heat Conduction, p. 57,
2. Solute Transport, p. 62,
3. Saturated/Unsaturated Water Flow (Richards Equation), p. 78,
4. Saturated/Unsaturated Preferential Water Flow (Richards Equation), p. 84,
5. Coupled Water-Surfactant Transport, p. 91,
6. Biodegradation, p. 99,

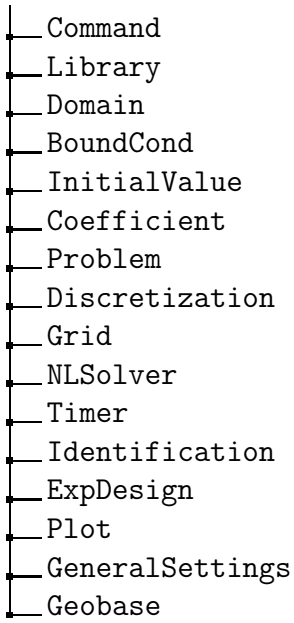
7. Multiphase Flow *(currently no documentation available)*,
8. Reactive Multicomponent Transport, p. 113.

1.1 Operating Instruction

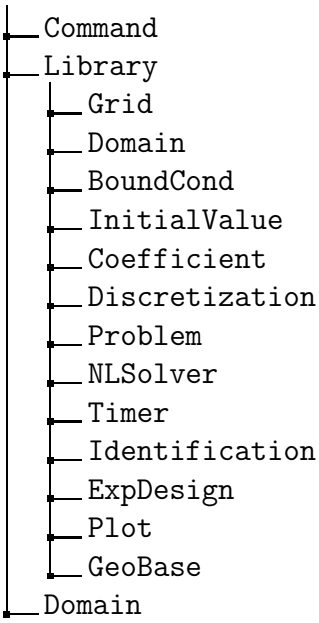
During a session an environment is built up in the main memory, which has a structure (tree) comparable to that of a filesystem directory. When you start RICHY1D, you face the toplevel of that directory. Each entry in this environment consists of two parts: a symbol indicating the type of the entry and its name. Clicking with the left mouse button on the name invokes it.

Navigating

The top level (the root directory) contains the following directories



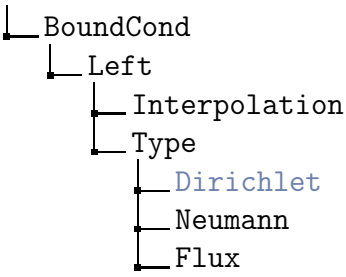
You can navigate through this environment with the left mouse button. To change into the top level directory `/Library`, just click on its name with your left mouse button. You will face its subdirectory:



Click a second time on `/Library`, to close it.

Selecting

Some directories allow the selection of one of its entries. This is done by clicking with the right mousebutton on that entry. The name of the selected entry is highlighted with a different color.



Editing

Flags are modified directly by clicking with the left mousebutton on the corresponding entry. The value toggles between on and off.

To edit any number or string entry of the environment, you simply have to click on it and an edit-window is opened, where you can enter your changes. Strings must not contain any blanks. Within the array editor you can jump from one cell to the next by pressing the Tab key. Note that the array windows always display a list of several entries even if your specific problem at hand requires less. The redundant entries are ignored.

Executing

Execution of a command is invoked by clicking with the left mousebutton on a command entry of the environment.

If a command is a member of the `/Library`, its execution is related to the generation of a new entry in one of the other toplevel directories.

Other commands are related to plotting, solving, etc.

Hiding

➟ `/GeneralSettings/DisplayAll`

Environment entries can be hidden. To enable hiding of entries, set the `/GeneralSettings/DisplayAll` off. To toggle between hide and show for an entry hold the Shift key and click the entry with the left mousebutton. Be aware that you pass the point of no return, when you first enable hiding and then hide the `/GeneralSettings/DisplayAll`.

Symbols

Each icon[1] indicates the type of an entry.

icon	symbol	meaning
		closed folder
		open folder
	✲	command
	?	flag (: checked, : unchecked)
	N	integer
	$\underline{\mathbf{N}}$	array of integers
	R	float
	$\underline{\mathbf{R}}$	array of floats
	$\underline{\underline{\mathbf{R}}}$	matrix of floats
	A	string

Array Data Input

Figure 1.1 shows the array data input dialog. Layout is as follows:

- ⟨ name of the variable being edited ⟩
- (data type of variable's values [♯values − 1])

[1] License: The icons , , , , , are taken from the Tango Icon Gallery (`http://tango.freedesktop.org/`), licensed under the Creative Commons Attribution Share-Alike license (`http://creativecommons.org/licenses/by-sa/2.5/`).

- value[0], ..., value[♯values − 1]
- | array data load | save | link controls | ok | cancel |

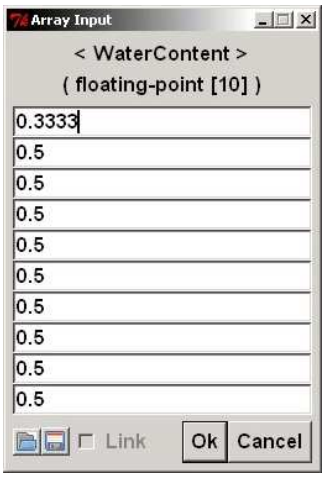


Figure 1.1: Example of an array data input window.

The load/save controls in the lower, left of the dialog window allow loading and saving of the array input data. The accepted file format is given in A.3 Scriptfile, p. 148. The 'Link' checkbox is used in conjunction with RICHY1D's logging facility. If array data is loaded from a file, the state of the 'Link' checkbox indicates whether the array data values are logged directly to the script file or if only the filename of the picked file is logged. In the last case, it is possible to manually edit the data file without touching the script file.

1.2 Work Scheme — The 12 Steps to Happiness

RICHY1D provides a menu driven frontend to set up a numerical simulation. To make use of RICHY1D's features, the user has to familiarize with RICHY1D's approach to define, set up and solve flow and transport phenomena of one of the implemented types. The proper utilization can be summarized by a schedule consisting of 12 consecutive steps. It is mandatory that these steps are performed according to the given order. Disregard of the work scheme may lead to unpredictable results!

1. Choose a PDE, suitable as a model equation for the transport problem you want to simulate and become familiar with the details of the corresponding problem class (cf.3 Problem Classes, p. 57).
2. Create and specify the Model Domain (cf.2.3.1 Domain, p. 10).
3. Create and specify proper Boundary Conditions (cf.2.3.2 Boundary Conditions, p. 11).

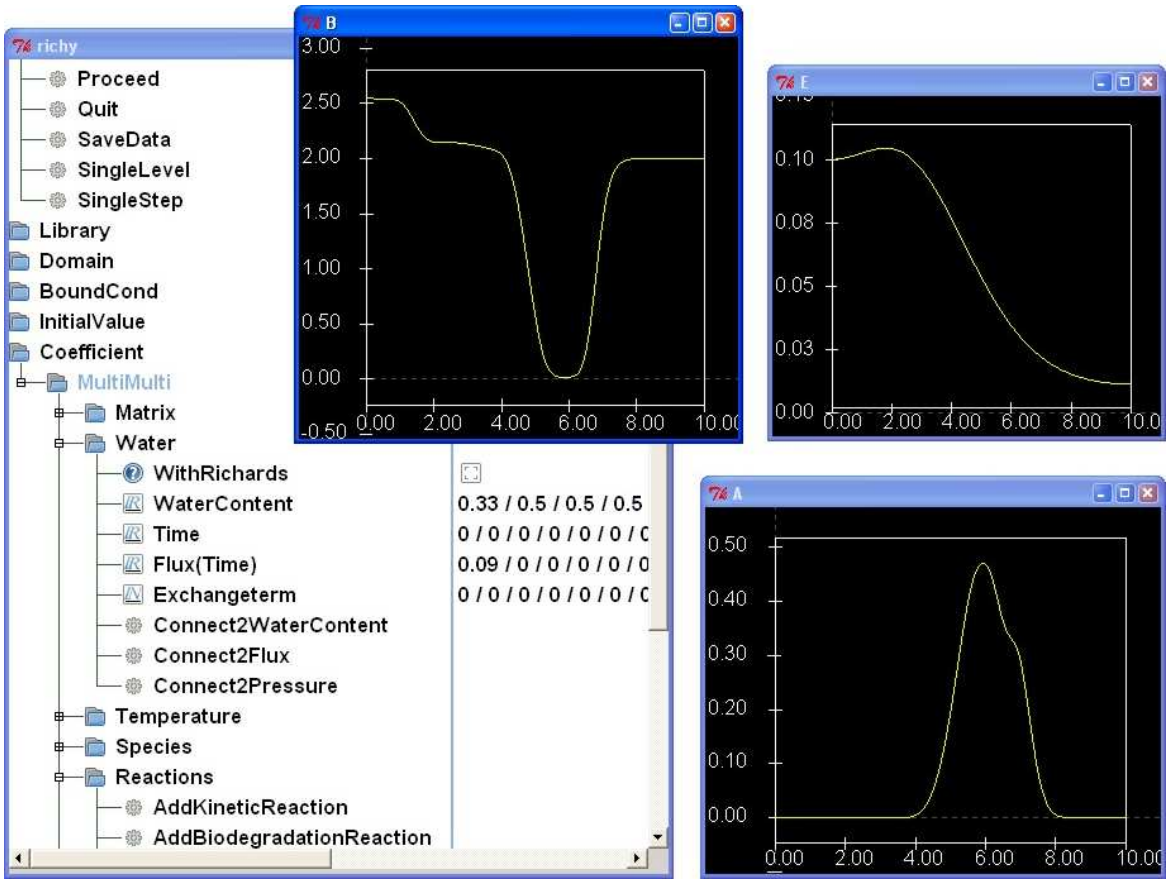


Figure 1.2: Screenshot of RICHY1D running on a WinXP system.

4. Create and specify proper Initial Values (cf.2.3.3 Initial Values, p. 14).
5. Select and specify the model's Coefficient Functions (cf.2.3.4 Coefficient Functions, p. 15).
6. Select the model's Discretization Scheme (cf.2.4 Discretization, p. 17).
7. Prepare the Numerical Grid.
8. Add the Problem and its Discretization to the Grid.
9. Initialize the Grid.
10. Apply a Numercial Solver.
11. Apply a Time Stepping Scheme.
12. Prepare the Plots (cf.2.9 Plot, p. 48).
13. Initialize and Execute the Simulation.

For an introducing example, see A.2.1 Simulation of Heat Conduction, p. 130.

Chapter 2

Items of Richy1D's Menu

2.1 Command

Here you find a list of commands of RICHY1D.

Commands within the toplevel directory `Command`

- `Command`
  - `AdaptGrid`
  - `AddP&D2Grid`
  - `EnableInterrupt` ..... see below
  - `EstimateGrid`
  - `ExecScript` ..... see A.3 Execution of Scriptfiles, p. 148
  - `ExpDesigning` ..... see 2.8 Experimental Design, p. 44
  - `Initialize`
  - `InitializeGrid`
  - `MultiLevel` ..... see 2.7 Identification, p. 22
  - `ParaFitting` ..... see 2.7 Identification, p. 22
  - `Plot` ..... see 2.9 Plot, p. 48
  - `Proceed`
  - `Quit`
  - `SaveData` ..... see 2.9 Plot, p. 48
  - `SingleLevel` ..... see 2.7 Identification, p. 22
  - `SingleStep`

Commands within the subdirectory `GeoBase` (cf.2.11 Geological Database, p. 50)

- `GeoBase`
  - `Data2Transport` ..... see 2.11 Soil Texture, p. 53
  - `Data2Heat` ..... see 2.11 Soil Texture, p. 53
  - `SoilTexture`
    - `AddLayer` ..... see 2.11 Interface to Problem, p. 55
    - `RemoveLayer` ..... see 2.11 Interface to Problem, p. 55

➟ /Command/EnableInterrupt

The command EnableInterrupt opens a small menu with the following buttons:

- Interrupt Simulation: Stops a (forward) simulation. To continue press Proceed in the main menu like usual.
- Pause ParaFitting: Pause an identification (of standard parametrisation). This way you are e. g.able to have a look at intermediate results. To continue uncheck the button again.
- Close: Close EnableInterrupt menu again. Then it is not possible to halt the simulation or identification before the end time is reached or the identification is completed, respectively.

2.2 Library

The /Library is the toolbox of RICHY1D. It provides the user with all components, that can be combined to define the transport problem for simulation. The generation of components from the /Library is explained in the documentation part of each component.

Library
- Grid
- Domain
- BoundCond
- InitialValue
- Coefficient
- Discretization
- Problem
- NLSolver
- Timer
- Identification
- ExpDesign
- Plot
- GeoBase

2.3 Problem

The problem is (from a mathematical point of view) either a boundary value problem (stationary, timeindependent case) or an initial boundary value problem (instationary, timedependent case). The problem therefore consists of a domain, a boundary condition, a set of coefficient functions and in case of timedependent problems of an initial value. The type of problem you have to generate, depends on the problem class (cf.3 Problem Classes, p. 57) you want to use.

The generation of any of these parts of a problem is explained within its documentation.

To generate a timedependent problem, follow these steps:

- Select the desired components within the directories `/Domain`, `/BoundCond`, `/InitialValue` and `/Coefficient`,
- go to `/Library/Problem` and execute `Parabolic`.

The four parts are bound together into one problem as an entry in the toplevel directory `/Problem` with the name you specified.

2.3.1 Domain

The domain is the explicit numerical representation of the region in which the transport phenomena occur. A domain has to be specified prior to all other definitions.

To generate a domain, go to the directory `/Library/Domain`, execute the command `1DDomain` and enter a domain name. To specify your domain, go to the directory `/Domain/domainname` and change the corresponding values.

```
Domain
└─ domainname
   ├─ LeftBoundary
   ├─ RightBoundary
   ├─ Subdonains
   ├─ InnerBoundaries
   └─ ElemsPerSubdom
```

The meaning of the entries is as follows

`LeftBoundary`	[`Length`]	left boundary of the domain
`RightBoundary`	[`Length`]	right boundary of the domain
`Subdomains`	[–]	number of subdomains
`InnerBoundaries`	[`Length`]	points between subdomains
`ElemsPerSubdom`	[–]	number of elements per subdomain

The number of subdomains you specify here is important for all the coefficients of your problem.

Note

Ensure that `LeftBoundary` < `RightBoundary`.

➟ `/GeneralSettings/ZDirection`

The domain is horizontal orientated with positiv direction from left to right and

counterclockwise rotated by $\varphi = \arcsin(\texttt{ZDirection})$ (cf.fig. 2.1). Admissable values of `ZDirection` are $[-1, 1]$, i. e., $\varphi \in [-\pi/2, \pi/2]$. The value of `ZDirection` enters directly the discretization as measure for gravitation (i. e., no gravitation for horizontal orientation). The orientation of the domain has no other influences on the problem.

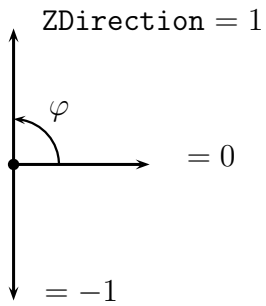


Figure 2.1: Illustration of the possible directions for the domain.

2.3.2 Boundary Conditions

After the domain, we have to specify proper boundary conditions. To define the boundary conditions, go to the directory `/Library/BoundCond`. You are offered three general commands (and the command `Closed4Transport` for the problem class 'solute transport', see Closed-Flow Design, p. 76 for details) to build the problem-specific boundary condition and some other commands to build the boundary condition for a specific problem class.

```
Library
└─ BoundCond
   ├─ ScalarBC
   ├─ VectorBC
   └─ AddComponent
```

The type of boundary condition you have to use depends on and is documented in 3 Problem Classes, p. 57.

Scalar Boundary Conditions

If you have a scalar problem (e. g.heat conduction), you have to use `ScalarBC` (e. g.to specify boundary values for the temperature). Follow these steps:

- Go to the directory `/Library/BoundCond`, execute `ScalarBC` and specify a name, e. g., 'SBC',
- go to the directory `/BoundCond/SBC` and specify for each side (`Left` and `Right`) the values as described below for a single component (see below).

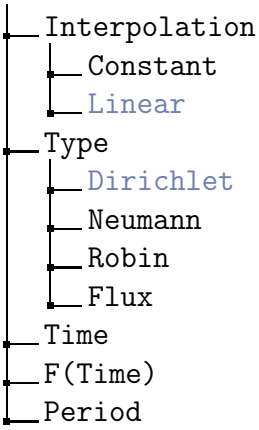
Vector Boundary Conditions

If you have a vectorial problem (e. g.surfactant influenced water transport), then you have to use `VectorBC` and further `AddComponent` for each component of the unknown vector of the problem class (e. g.one component to specify boundary values for the water pressure and a second component for the boundary values of surfactant concentration). Follow these steps:

- Go to the directory `/Library/BoundCond`, execute `VectorBC` and specify a name, e. g., 'VBC',
- for each component execute AddComponent and specify a name, e. g., 'Comp',
- for each component `Comp` go to the directories `/BoundCond/VBC/Left/Comp` and `/BoundCond/VBC/Right/Comp` and specify the values as described below for a single component (see below).

Specifying the Boundary Condition for a Component

The menu entries of a boundary condition for a single comoponent looks like:



The meanings of these entries are as follows:

{`Constant`, `Linear`}	*selectable*	type of interpolation
{`Dirichlet`, `Neumann`, `Robin`, `Flux`}	*selectable*	type of boundary condition
`Time`	[`Time`]	array of timepoints
`F(Time)`	[∗]	array of values corresponding to timepoints
`Period`	[`Time`]	time interval for periodical repetition

➟ `/BoundCond/[name]/[Left or Right]/Interpolation`

- Define the type of interpolation (piecewise constant or piecewise linear, see below).
- Piecewise constant interpolation is mainly used for experimental design (cf.2.8 Experimental Design, p. 44).

➟ `/BoundCond/[name]/[Left or Right]/Type`

- The problem class gives information about admissable types of boundary condition and details of their meaning.

➟ `/BoundCond/[name]/[Left or Right]/F(Time)`

- The boundary condition is a continuous piecewise linear function or a discontinous piecewise constant function of time.
- `Time` and `F(Time)` are float arrays that provide the interface to this function.
- Define at least two values for `Time` and for `F(Time)`
- the time interval, you want to simulate, should be inside of the time intervals of the array `Time`.
- The evaluation of the boundary condition is piecewise constant or piecewise linear between timepoints with increasing time—the function is kept constant if the time interval of two consecutive timepoints is less or equal 0.
- Piecewise constant interpolation: the value of the function is `F(Time[i])` on the half-open interval [`F(Time[i])`, `F(Time[i+1])`[.

➟ `/BoundCond/[name]/[Left or Right]/Period`

- The boundary condition function is periodical in time: `F(Time+Period)` = `F(Time)`.

Special Forms of Boundary Conditions

If a special form of the boundary conditions is available for a problem class, detailed information can be found in 3 Problem Classes, p. 57.

2.3.3 Initial Values

After the boundary condition, we have to specify a proper initial value. To define the initial value, go to the directory `/Library/InitialValue`. You are offered at least three different commands to build the problem-specific initial value:

```
Library
├─Grid
├─Domain
├─BoundCond
└─InitialValue
  ├─Create
  ├─AddVector
  └─StdFunctions
```

The type of initial value you have to use depends on and is documented in 3 Problem Classes, p. 57.

In general you have to follow these steps:

- Go to the directory `/Library/InitialValue`,
- generate a basic entry in the toplevel directory `/InitialValue` with the command `Create` and specify its name, e. g., 'Cold',
- for each vector of your problem class add an entry to `/InitialValue/Cold` by executing `AddVector` and specify its name, e. g., 'Temperature',
- for each component of an already defined vector, e. g., `/InitialValue/Cold/Temperature` add an entry by executing, e. g., `StdFunctions` and specify the name for the initial value function of this component, e. g., 'Comp1'.

In the simplest case of heat conduction (a scalar initial boundary value problem) the initial value `Cold` has to set values for one vector `Temperature` with one component `Comp1`. The result looks like:

```
InitialValue
└─Cold
  └─Temperature
    └─Comp1
      ├─Coord
      └─F(Coord)
```

The meaning of these entries is as follows:

`Coord`	[`Length`]	array of coordinates
`F(Coord)`	[∗]	array of values corresponding to coordinates

➟ `/InitialValue/[...]/Coord, /InitialValue/[...]/F(Coord)`

- The initial value is a continous, piecewise linear function of coordinate.
- `Coord` and `F(Coord)` are float arrays that provide the interface to this piecewise linear function.
- Define at least two values for `Coord` and for `F(Coord)`.
- The space interval, of your domain, should be inside of the coordinate intervals of the array `Coord`.

Note
The toplevel directory `/InitialValue` and any directory below (e. g.`Cold`) allow selection of one of their entries. The commands `AddVector` and `StdFunctions` generate their entries below the selected directories.

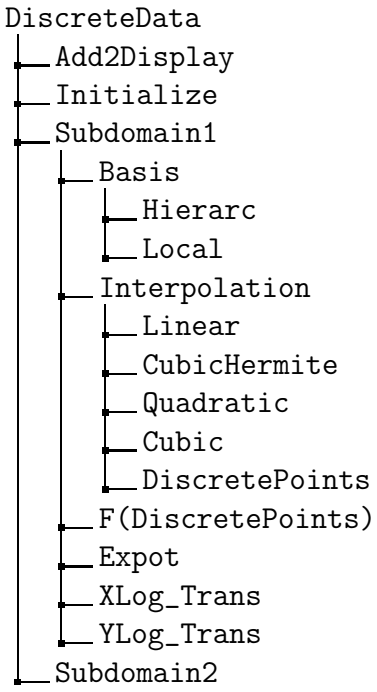
2.3.4 Coefficient Functions

The last part to complement a problem is a set of coefficient functions. To generate such a set follow these steps:

- Go to the directory `/Library/Coefficient`,
- execute that entry, that corresponds to your problem class, and specify the name under which this set of coefficent functions will be available in the toplevel directory `/Coefficient`,
- change into the directory `/Coefficient/[name]` to edit parameters of that set of functions. If your domain is divided in several subdomains, many parameters have to be specified for each subdomain (even if they may not vary in every subdomain of your specific problem at hand).

Generic Coefficient Functions for Spline Interpolation

For some coefficient functions exists a generic type (`ParaType` 4) for spline interpolation (e. g.a coefficient function can be discretized to a piecewise polynomial function). The subdirectory `/Coefficient/Kermit/Functions` contains the names of these coefficient functions. The data of spline interpolation are located at a directory called `../DiscreteData`.



The meanings of these entries are as follows:

{Hierarc, Local}	*selectable*	type of basis functions
{Linear, CubicHermite, Quadratic, Cubic}	*selectable*	type of splines/interpolation
DiscretePoints	[∗]	array of discretization points
F(DiscretePoints)	[∗]	array of values corresponding to discretization points
XLog_Trans	[−]	logarithmic transformation, x-domain
YLog_Trans	[−]	logarithmic transformation, y-domain

Note

- The command `Add2Display` creates above entries for the next subdomain.
- The float array `DiscretePoints` describes a discretization $x_1 < x_2 < \ldots < x_n$ of the domain of definition.
- The evaluation of the coefficient function corresponds to the selected interpolation and basis type.
- The coefficient function is constant for $x < x_1$ and $x > x_n$.
- If `XLog_Trans`/`YLog_Trans` is active, the discrete data describe the according logarithmic function.
- The generic coefficient functions must be initialized by executing the command `Initialize`.

Any further information about coefficient functions can be found in 3 Problem Classes, p. 57.

2.4 Discretization

Each problem class provides a discretization for a problem. Therefore more detailed information about discretizations can be found at the corresponding problem classes, see 3 Problem Classes, p. 57. After the generation of a problem, a proper discretization has to be specified. Follow this step:

- Go to `/Library/Discretization`, execute the entry with the name defined in the documentation of your problem class and specify any name, say 'Disc1'.

The discretization is now the entry name of the toplevel directory `/Discretization`. There is one common subdirectory `/Discretization/Disc1/Variables`, under which the available variables of a discretization can be selected. If a variable is displayed like a flag, you have to activate the flag and initialize the grid, to make the variable accessible. The use of other entries below `/Discretization/Disc1` is explained when needed.

2.5 Solvers

Nonlinear Solvers

The nonlinear system of equations resulting of the discretization of nonlinear problems is solved by Newton's Method. The advantage of this iterative method is that it converges quadratically to the solution of the problem, i. e., the number of significant digits is doubled in every time step (minus a fixed number). However, Newton's method is locally convergent, i. e., you may not find a solution that satisfies the expected error,

if your initial iterate is too far from the true solution. We use a damped version of the algorithm with Armijo's rule (see Knabner and Angermann 2003).

The interface of the `NLSolver` reads

```
NLSolver
└─ [name]
   ├─ AbsError
   ├─ RelError
   ├─ MaxNewIterates
   ├─ MaxLineSearchSteps
   ├─ GoodIterates
   ├─ BadIterates
   └─ PerformanceInfo
      ├─ TotalCPUTime(s)
      ├─ Time4Assembling(s)
      ├─ Time4LinearSolver(s)
      ├─ TotalNewtonSteps
      └─ AverageNewtonSteps
```

➟ `/NLSolver/[name]/PerformanceInfo`

The subdirectory `PerformanceInfo` contains—seperately for every problem—information about

- the total CPU time in seconds of your simulation,
- the effort in CPU seconds for assembling routines (composing the Jacobian, right hand side, evaluation of the norm of the right hand side),
- the time for the linear solver,
- the total number of Newton steps, and
- the number of average Newton steps per time step.

Note

If the number of Newton steps per time step is high (e. g.greater than 10), you should try to reduce the time step size (see 2.6 Timesteppers, p. 20). Of course, if your problem is linear (e. g.heat conduction), you need only one Newton step per time step to find your solution.

Linear Solvers

In each iteration step of the nonlinear solver, the linear solver is called. There are two different linear solvers implemented:

- The LU factorization,

- the preconditioned GMinRes(m) method.

The user should select the solver which requires less CPU time (see below). In most situations, the LU factorization is the more efficient solver.

Description of the methods and their parameters, efficiency. The LU factorization is a direct, i. e., a non-iterative method, that decomposes the system matrix into a lower and an upper triangular matrix. Details on the method can be found in any book on numerics. For problems with one space dimension, the computational costs of the LU factorization are $\mathcal{O}(S\,N^3)$, where S is the number of species and N is the number of elements.

The GMinRes(m) method is an iterative method. As a preconditioner, the matrix

$$(\mathbf{D} + \omega\,\mathbf{L})\,\mathbf{D}^{-1}\,(\mathbf{D} + \omega\,\mathbf{U}) \tag{2.1}$$

is used, where $\mathbf{D}$, $\mathbf{L}$, $\mathbf{U}$ are the diagonal/lower/upper part of the system matrix and ω is a relaxation parameter. If GMinRes(m) is used, the user can choose 3 parameters:

- The convergence criterion ε. ε should definitely be chosen at least one or two powers of ten smaller than the corresponding value for the nonlinear solver. Otherwise, the nonlinear solver might not converge.

- The parameter m. Since the computational complexity of each iteration step and the amount of computer memory increase during the GMinRes iteration, after m iteration steps, a restart is done. The literature recommends to choose m between 6 and 20.

- The relaxation parameter ω. The convergence rate can be optimized by choosing the right relaxation parameter for the preconditioner. ω must be chosen from the interval [1.0, 2.0]. The finer the spatial discretization of the problem, the larger ω should be chosen. For the script file `def4multispec.scr` (multicomponent reactive transport with 10 species), we found the following optimal values for ω:

N	ω_{opt}
125	1.35
250	1.45
500	1.7
1000	1.85
2000	1.8

The CPU time for the preconditioned GMinRes(m) method depends on the number of nonzero entries in the system matrix (between $\mathcal{O}(S\,N)$ and $\mathcal{O}(S^2\,N)$, depending on the reaction network) and the number of iteration steps (about $\mathcal{O}(\sqrt{N})$ required to reach the convergence criterion. Hence, (only) if the number of species S is large

and the number of elements N is small, the GMinRes can be the more efficient choice. To give an idea of the range in which GMinRes(m) can be more efficient, we give the following table:

- For $S = 20$ and $N < 300$ choose GMinRes(m),
- for $S = 10$ and $N < 150$ choose GMinRes(m),
- for $S = 1$ the direct solvers seems to be faster for arbitrary N.

2.6 Timesteppers

For the temporal discretization, one time stepper has to be selected. The selected time stepper handles all the problems which are defined by the user. For the solute transport equation (including sorption, convection, diffusion-dispersion and serial decay reactions, cf.3.2 Solute Transport, p. 62), the following time steppers are implemented:

- The Implicit Euler Method (IE),
- the Crank–Nicolson Method (CN),
- the 2nd order Backward Difference Method (BDF2),
- the 2nd order Numerical Differentiation Method (NDF2).

For other problem classes, only IE is implemented presently.

Notice the following restriction: If CN or BDF2 or NDF2 is selected while problems different from the transport equation (i. e., Richards equation, biodegradation, heat conduction, etc.) are to be solved, or while those problems are coupled to the transport equation, those problems are still handled with the standard IE method, while only the solute transport equation solver uses the chosen timestepper. No error message or warning is generated in this case. The overall error will be of first order in this case.

Properties of the Methods

General classification. IE and CN are one-step methods, i. e., the state at time t_n is computed from the state at time t_{n-1}. BDF2 and NDF2 are 2-step/3-step methods, respectively, i. e., the state at time t_n is computed from the state at the previous timesteps t_{n-1} / t_{n-2}. NDF2 is a modification of the well-known BDF2 method having a smaller leading error term and similar stability properties.

Accuracy. IE is of first order accuracy while CN, BDF2, NDF2 are of second order. Thus, the latter methods should yield a higher accuracy for the same timestep size, provided the solution is smooth enough. To put it in different words: The second order methods allow to choose larger timesteps to gain the same accuracy as IE.

To take advantage from this property, it is necessary that temporal and spatial error of the scheme are equilibrated. (If the spatial discretization is too coarse, the total error is dominated by the spatial error, i. e., a second order temporal scheme would bring no advantage at all.) CN is the method with the smallest temporal discretization error (see table). If the solution is not smooth enough, the second order accuracy of CN, BDF2, NDF2 may not be recovered. However, in this case, the second order methods usually lead at least to a smaller leading error term compared to IE.

Stability. All the methods IE, CN, BDF2, NDF2 are A-stable, i. e., for a linear problem, the numerical solution decays, if the exact solution decays. However, for CN, this decay may take very long, and during the decay, temporal and spatial oscillations of the numerical solution may be present which decrease very slowly. These oscillations may lead to (locally in space and time) negative solutions! In mathematical terms: IE, BDF2, NDF2 are L-stable, but CN is not. A- and L-stability are stability criteria for linear problems. These creteria are also meaningful for nonlinear problems (e. g.nonlinear sorption processes). However, the stability for nonlinear problems is guaranteed by the criterion of B-stability, and its generalization for multistep methods, the G-stability. IE, CN are B-stable, BDF2 and NDF2 are G-stable.

Stepper	Leading Error Term	A-stability	L-stability	B-/G-stability
IE	$\Delta t/2$	yes	yes	yes
CN	$\Delta t^2/12$	yes	no	yes
NDF2	$\Delta t^2/6$	yes	yes	yes
BDF2	$\Delta t^2/3$	yes	yes	yes

Advice for the Optimal Choice of Timestepper

1. Get RICHY1D running by using IE.

2. If you are only solving transport equation(s):
   Switch to a higher order method (BDF2, NDF2, CN). If the problem is stiff or if you want to choose very large timesteps, choose BDF2 or NDF2. Otherwise, also CN should work. If oscillations or negative concentrations occur (most likely for CN), choose a different stepper or smaller timesteps.

3. Optimize timestep size Δt in relation to spatial discretization parameter h ($h =$ domain length divided by number of mesh points): For a given h and Δt, run the program with Δt, h then with $\Delta t/2$, h and with Δt, $h/2$. Save the solutions each time. If the change of h has more influence on the solution than the change of Δt, the spatial error is dominant, i. e., you may choose larger Δt (or smaller h). Otherwise, your temporal error is dominant, i. e., increase h or decrease Δt.

A detailed overview on timestepping methods and their properties is given in the book of Strehmel and Weiner (1995).

2.7 Identification

To define the identification, go to the directory `/Library/Identification` and proceed according to one of the following schemes. If you want to identify certain parameters (coefficients) of a given, fixed functional form (e. g.a Freundlich isotherm), then choose `StandardIdent` (see 2.7.1 Identification of Standard Parametrization, p. 22). If you do not want to restrict yourself to a special functional form of your coefficient function you can choose `FormfreeIdent` (see 2.7.2 Formfree Identification with Spline Parametrization, p. 32) which allows you to identify e. g.the best piecewise cubic spline function.

Details of the formfree identification method are given in Igler (1998) for reactive transport and in Bitterlich (2003) for the Richards equation. Applications can be found, e. g., in Igler et al. (1998) and Bitterlich et al. (2004).

```
Library
└─Identification
  ├─StandardIdent
  └─FormfreeIdent
```

2.7.1 Identification of Standard Parametrization

General Framework

For some problem classes an identification procedure for coefficient functions exists. To create the toolbox necessary for the identification of the coefficients of a given standard parametrization (i. e., to identify parameters of a given coefficient function), go to the directory `/Library/Identification/StandardIdent`. This directory contains the commands to install the identification tools:

```
Library
└─Identification
  ├─StandardIdent
  │ └─CreateParaFit
  ├─AddMeasurement
  ├─InputMeasData
  └─InputWeights
```

In general you have to proceed according to the following steps:

- Set up the problem-specific (forward) simulation,
- go to the directory `/Library/Identification/StandardIdent`,
- generate a basic entry with the command `CreateParaFit` and specify its name,
- select the variable which is observed for the identification in the directory `/Discretization/[discretization name]/Variables`,

- go to the directory `/Library/Identification` and add the selected variable to the identification with the command `AddMeasurement` and specify its name.

In this way one or more coefficient functions and/or measurements can be added to an identification.

Note
Your problems have to be established with fixed grid and time step size. Do not use adaption, if you want to identify parameters.

The result looks in general like this:

```
Identification
└─ [identification name]
   ├─ Measurements
   ├─ Observation
   │  └─ Linear
   ├─ ErrorFunctional
   │  ├─ OLS
   │  ├─ OLS_TW
   │  ├─ OLS_SW
   │  └─ OLS_manW
   ├─ PseudoInvSensMat
   ├─ Gradient
   │  ├─ FDAppr
   │  └─ Direct
   ├─ Optimization
   ├─ ErrorAnalysis
   ├─ SaveIdentData
   └─ ComputeMeasWeight
```

The meanings of these entries are explained in the following table and in the next few paragraphs.

`Measurements`	*folder*	
`Linear`	*selectable*	type of interpolation of observations

`OLS`
`OLS_TW`
`OLS_SW`
`OLS_manW`
	selectable	type of error functional: ordinary least squares; OLS with weights according to the time interval around the data point; OLS with weights according to the (inverse of the) sensitivity of the observation; OLS with weights given by the user manually or by computing `PseudoInvWFactors`

`PseudiInvSensMat`	*folder*	
`FDAppr` `Direct`	*selectable*	type of gradient computation: Finite Difference Approximation or direct gradient computation
`Optimization`	*folder*	
`ErrorAnalysis`	*folder*	
`SaveIdentData`	*folder*	
`ComputeWeight`	*command*	computes a weighting factor for the selected measurement (yields the inverse of the square of the mean value)

Measurements

The entries of the directory `/Identification/[identification name]/Measurements` are selectable for the command `ComputeMeasWeight` or for plotting with the command `GLMeasPlot` in the directory `/Library/Plot`. The menu entries of a measurement looks like this:

```
Measurements
└─ [measurement name]
   ├─ Coordinate
   ├─ Weight
   ├─ NumInBlock
   ├─ MeasTime
   ├─ F(MeasTime)
   ├─ W(MeasTime)
   ├─ ErrorAnalysis
   └─ Epsilon
```

The meaning is as follows:

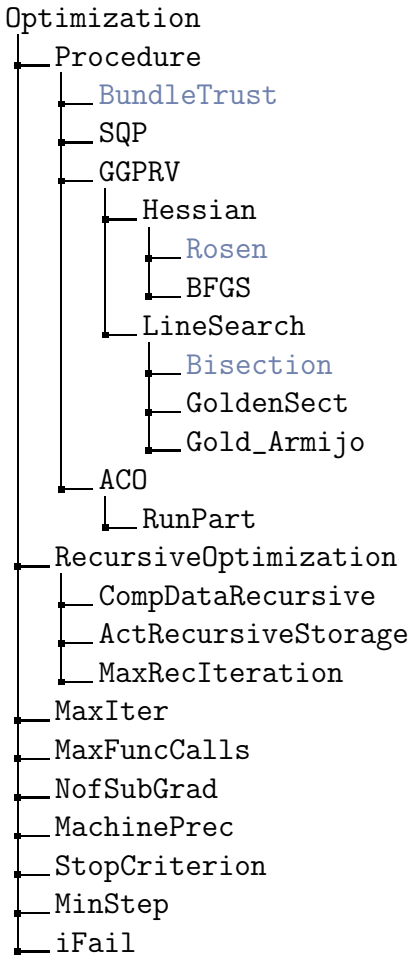
`Coordinate`	[`Length`]	measurement point for variable
`Weight`	[−]	weighting factor in error functional *(By using the value* 0 *the current parameter will not be considered for the inverse problem.)*
`NumInBlock`	[−]	component if variable is a vector
`MeasTime`	[`Time`]	array of timepoints when variable has been observed
`F(MeasTime)`	[∗]	array of measured values corresponding to timepoints
`Epsilon`	[−]	data error

Measured data (`MeasTime`, `F(MeasTime)`) for the selected measurement can also be read from a file by executing the command `InputMeasData` in the directory `/Library/Identification`. In case of more than one identification or measurements you have to assign one of the them before reading the datafile.

Note
The entry `Epsilon` is only used in the problem class experimental design (cf.2.8 Experimental Design, p. 44).

Optimization

The necessary settings for the applied optimization procedure are included in the subdirectory `/Optimization` in every entry of the toplevel directory `/Identification`.



The meaning of these entries is as follows:

{`BundleTrust`, `SQP`, `GGPRV`, `ACO`}	*selectable*	type of optimization procedure
{`Rosen`, `BFGS`}	*selectable*	type of approximation of Hessian
{`Bisection`, `GoldenSect`, `Gold_Armijo`}	*selectable*	type of line search for `GGPRV`
`RunPart`	[−]	considered pathways for `ACO` in percent
`MaxIter`	[−]	maximum number of iterations
`MaxFuncCalls`	[−]	maximum number of function calls
`NofSubGrad`	[−]	maximum number of stored subgradients (≥ 3, only `BundleTrust`)
`MachinePrec`	[−]	machine precision
`StopCriterion`	[−]	stopping criterion
`MinStep`	[`Time`]	minimum time stepsize during identification
`ifail`	[−]	Parameter for showing reason of terminating: 0: The optimality conditions are satisfied. 1: The algorithm has been stopped after `MaxIter` iterations. 2: The algorithm computed an uphill search direction. 3: Underflow occured when determining a new approximation matrix for the Hessian of the Lagrangian. 4: More than `MaxFunCalls` function calls are required during the line search. 5: Length of a working array is too short. 7: The search direction is close to zero, but the current iterate is still infeasible. > 10: The solution of the quadratic of least squares subproblem has been terminated with an error message IFQL > 0 and ifail is set to IFQL $+ 10$.

Note — Optimization Procedures

- `BundleTrust`:
  algorithm for nonsmooth functions.
- `SQP`:
  sequential quadratic programming method for smooth functions; modified BFGS-update; $L2$-augmented Lagrangian penalty function/$L1$-penalty function for the line search.
- `GGPRV`:
  gradient-projection method for smooth functions; (unmodified) BFGS-update or method of Rosen; bisection method, golden section method, or Goldstein-Armijo principle for the line search.
- `ACO`:
  ant colony optimization algorithm (non-derivative method, for standard identification only).

Pseudo Inverse Sensitivity Matrix

The subdirectory `/Identification/[identification name]/PseudoInvSensMat` contains some options for computing and saving the pseudo inverse sensitivity matrix and the resulting weighting factors for the error functional. In general it looks like this:

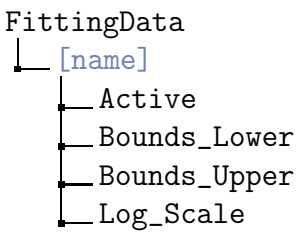
```
PseudoInvSensMat
├── PInvWeightFactors
├── SavePInvWFactors
├── SavePseudoInv
├── PInvWFactorsFileName
├── PseudoInvFileName
└── DirName4SavingPFile
```

The meanings of these above entries are as follows:

`PInvWeightFactors`	[−]	computes weighting factors for the error functional based on the pseudo inverse sensitivity matrix. The factors can be used by recursive identification in the next identification procedure
`SavePInvWFactors`	[−]	saves the computed pseudo inverse sensitivity weighting factors
`SavePseudoInv`	[−]	saves the pseudo inverse sensitivity matrix
`PInvWFactorsFileName`	[−]	name of the file including the weighting factors. The default filename is 'PInvWFactors.dat'
`PseudoInvFileName`	[−]	name of the file including the pseudo inverse sensitivity matrix. The default filename is 'PInvSensMat.dat'
`DirName4SavingPFile`	[−]	name of the directory where the data are saved (if the directory does not exist, the directory will be created)

Coefficient Functions

After executing the command `CreateParaFit` the directory `/Coefficient/[coefficient name]` contains subdirectories named `/FittingData` with entries of the following type:



The meanings of these entries are as follows:

`Active`	*flag*	flag for activating/deactivating the parameter for optimization
`Bounds_Lower`	[∗]	lower boundaries (per subdomain) for the search of the parameter value
`Bounds_Upper`	[∗]	upper boundaries (per subdomain) for the search of the parameter value
`Log_Scale`	*flag*	flag for optimization on logarithmic scale
`EstDeviation`	[∗]	estimated deviation of the optimized parameter
`99%Confidence`	[∗]	99% confidence interval for the parameter

Note

The default of the item `Active` is 'off'. Before starting the current identification at least one coefficient has to be active.

Error Analysis

The subdirectory `/Identification/[identification name]/ErrorAnalysis` contains some data for the error analysis and looks similar to this:

```
ErrorAnalysis
├── MueCond
├── MueMax
├── WEuklidNorm
├── WMaxNorm
├── Trace
├── Determinant
└── Func_Value
```

The meanings of the above entries are as follows:

`MueCond`	[–]	condition number
`MueMax`	[–]	maximum error amplification
`EuklidNorm`	[–]	Euklidian norm of the residuals
`MaxNorm`	[–]	maximum norm of the residuals
`Trace`	[–]	trace of the covariance matrix
`Determinant`	[–]	determinant of the covariance matrix (product of the non-zero eigenvalues)
`Func_Value`	[–]	value of the error functionals

Save Identification Data

The subdirectory `/Identification/[identification name]/SaveIdentData` contains some saving options. It looks in general like this:

```
SaveIdentData
├── SaveOptData
├── SaveSensitivities
├── SaveCorrelCoeff
├── SaveDataRecursive
├── ActRecursiveStorage
└── DirectoryName4Saving
```

The meanings of the above entries are as follows:

`SaveData`	[−]	save option for the identification results: `ParameterValues.dat` includes the identified optimal value of the coefficients; `ErrorData.dat` gives information about the minimized function; `measurementname.dat` prints the measured and the simulated data for the appropriate variable
`SaveSensitivities`	[−]	save option for the sensitivities of the identification results: `Sensitivities.dat` prints the sensitivities of each measurement variable and each time point
`SaveCorrelCoeffs`	[−]	save option for the correlation coefficients: `CorrelCoeffs.dat`
`SaveDataRecursive`	[−]	save option for recursive identification calls
`ActRecursiveStorage`	[−]	parameter show the actual number of recursive identification
`DirectoryName4Saving`	[−]	name of the directory where the data are saved (if the directory does not exist, the directory will be created)

Plots

For the identification in the case of standard parametrization we can generate particular plots where the fixed measured data points are given along with the calculated solution corresponding to the current coefficients.

First, you have to single out one of the measurements for the current plot:

- Go to the directory `/Identification`,
- choose one of the identification sets you have defined before,
- go to the directory `/[identification name]/Measurements` corresponding to your favorite identification,
- choose one of the measurements *(by clicking the right mouse button)*.

Note

In the case of a single identification with just one measurement these steps can be omitted because because the corresponding measurement will be highlighted automatically.

Now we can create the appropriate plot:

- Go to the directory `/Library/Plot`,
- choose the command `GLMeasPlot` and specify a name for the plot,
- execute the command `Plot` in the `/Command` directory.

Starting the Identification

If an identification with at least one measurement was created the inverse simulation procedure can be started:

- select a identification in the toplevel directory `/Identification`
- specify the corresponding entries *(choose the active variables etc.)*.
- The selected identification finally runs by executing the command `ParaFitting` in the `/Command` directory.

Results of the Current Identification

After the identification of a standard parametrization you will find the calculated data of the identified parameters in the directory `/Coefficient/[coefficient name]`. They have replaced the initial values you have inserted before. Furthermore you will find more information about the quality of the results in the subdirectory `/ParaFitting`.

2.7.2 Formfree Identification with Spline Parametrization

General Framework

For some problem classes exists an identification procedure for coefficient functions. To define the formfree identification, go to the directory `/Library/Identification/FormfreeIdent`. This directory contains some commands to set up the identification:

```
Library
└─Identification
  ├─FormfreeIdent
  │ ├─1DFFIdent
  │ │ ├─Create1DFFIdent
  │ │ └─Add1DFunction
  │ └─3DFFIdent
  │   ├─Create3DFFIdent
  │   └─Add3DFunction
  ├─AddMeasurement
  ├─InputMeasData
  └─InputWeights
```

In general you have to proceed according to the following steps:

- Set up the problem-specific simulation,
- go to the directory `/Library/Identification`,

- generate a basic entry in the subdirectory `/FormfreeIdent/1DFFIdent` (`3DFFIdent`) with the command `Create1DFFIdent` (`Create3DFFIdent`) and specify its name,
- go to the directory `/Coefficient/[coefficient name]/Functions/ScalarFunctions` and select the function, which is to be identified,
- go to the directory `/Library/Identification/FormfreeIdent/1DFFIdent` (`3DFFIdent`) and add the selected function to the identification with the command `Add1DFunction` (`Add3DFunction`),
- select the variable, which is observed for identification, in the directory `/Discretization/[discretization name]/Variables`,
- go to the directory `/Library/Identification` and add the selected variable to the identification with the command `AddMeasurement` and specify its name.

In this way one or more coefficient functions can be added to an identification.

Note
The problems have to be established with fixed grid and time step size. Do not use adaption, if you want to identify parameters.

The result looks like this:

```
Identification
└─ [identification name]
   ├─ Measurements
   ├─ Strategy.
   │  ├─ Adaptive
   │  │  ├─ Local
   │  │  └─ Global
   │  ├─ Equidistant
   │  ├─ DyadicLeft
   │  ├─ DyadicRight
   │  └─ OneScale
   ├─ Observation
   │  └─ Linear
   ├─ ErrorFunctional
   │  ├─ OLS
   │  ├─ OLS_TW
   │  ├─ OLS_SW
   │  ├─ OLS_manW
   │  └─ Mualem_Pen
   ├─ PseudoInvSensMat
   ├─ Gradient
   │  ├─ FDAppr
   │  │  └─ StepSize
   │  └─ Direct
   ├─ Optimization
   ├─ ErrorAnalysis
   ├─ Options .4 Start
   │  ├─ End
   │  ├─ Step
   │  ├─ NumDPoints
   │  └─ Scale
   ├─ SaveIdentData
   └─ ComputeMeasWeight
```

The meanings of these entries are as follows:

`Measurements`	*folder*	
`Linear`	*selectable*	type of interpolation of observations

`Adaptive`
`Equidistant`
`DyadicLeft`
`DyadicRight`
`OneScale`
	selectable	type of refinement strategy

`Local` `Global`	*selectable*	type of adaptive refinement strategy

`OLS`
`OLS_TW`
`OLS_SW`
`OLS_manW`
`OLS_Mualem_Pen`
	selectable	type of error functional: ordinary least squares; OLS with weights according to the time interval around the data point; OLS with weights according to the (inverse of the) sensitivity of the observation; OLS with weights given by the user manually or by computing `PseudoInvWFactors`

`FDAppr` `Direct`	*selectable*	type of gradient computation: Finite Difference Approximation or direct gradient computation
`StepSize`	[−]	relative stepsize for finite difference approximation
`Optimization`	*folder*	
`Options`	*folder*	
`ErrorAnalysis`	*folder*	
`SaveIdentData`	*folder*	
`ComputeWeight`	*command*	computes a weighting factor for the selected measurement (yields the inverse of the square of the mean value)

Measurements

The entries of the directory `/Identification/[identification name]/Measurements` are selectable for the command `ComputeWeight` or for plotting with the command `GLMeasPlot` in the directory `/Library/Plot`. The menu entries of a measurement look like this:

```
Measurements
└─ [name]
   ├─ Coordinate
   ├─ Weight
   ├─ NumInBlock
   ├─ MeasTime
   ├─ F(MeasTime)
   ├─ ErrorAnalysis
   └─ Epsilon
```

The meaning is as follows:

`Coordinate`	[`Length`]	measurement point for variable
`Weight`	[−]	weighting factor in error functional
`NumInBlock`	[−]	component if variable is a vector
`MeasTime`	[`Time`]	array of timepoints of measuring
`F(MeasTime)`	[∗]	array of measured values corresponding to timepoints
`Epsilon`	[−]	data error

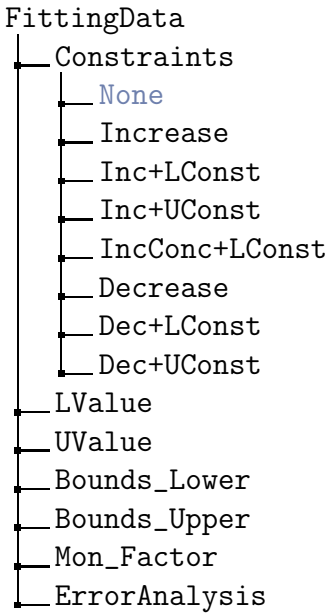
Measured Data (`MeasTime`, `F(MeasTime)`) for the selected measurement can be read also from a file by executing the command `InputMeasData` of the directory `/Library/Identification`. In the case of more than one identification or measurements you have to assign one of the them before reading the datafile.

Note

The entry `Epsilon` is only used in the problem class experimental design (cf.2.8 Experimental Design, p. 44).

Coefficient Functions

If a coefficient function is added to a identification, the directory `/Coefficient/[coefficient name]/Functions/ScalarFunctions/[function name]` contains a subdirectory `/FittingData` with further informations for the optimization.



The meanings of these entries are as follows:

{ None, Increase, Inc+LConst, Inc+UConst, IncConc+LConst, Decrease, Dec+LConst, Dec+UConst }	*selectable*	type of constraints corresponding to the physical properties of the coefficient function
LValue	[∗]	lower value
UValue	[∗]	upper value
Bounds_Lower	[∗]	lower boundaries (per subdomain) for search of the param. valu
Bounds_Upper	[∗]	upper boundaries (per subdomain) for search of the param. valu
Mon_Factor	[−]	monotonicity factor
MaxEstDeviation	[∗]	maximum estimated deviation of identified formfree coefficient
Max99%Confidence	[∗]	max. 99% confidence interval of identified formfree coefficient

Note

For the formfree identification the ParaType of the to identified coefficient function have set to generic type.

Pseudo Inverse Sensitivity Matrix

The subdirectory `/Identification/[identification name]/PseudoInvSensMat` contains some options for computing and saving the pseudo inverse sensitivity matrix and the resulting weighting factors for the error functional. In general it looks like this:

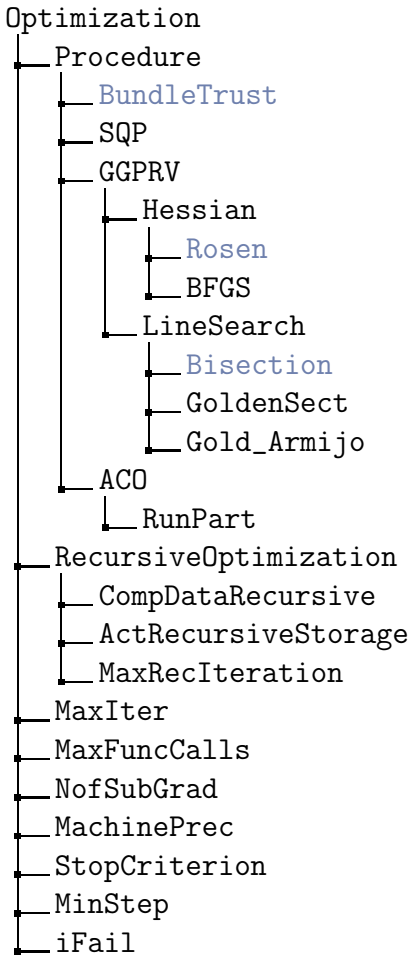
```
PseudoInvSensMat
├── PInvWeightFactors
├── SavePInvWFactors
├── SavePseudoInv
├── PInvWFactorsFileName
├── PseudoInvFileName
└── DirName4SavingPFile
```

The meanings of these above entries are as follows:

`PInvWeightFactors`	[–]	computes weighting factors for the error functional based on the pseudo inverse sensitivity matrix. The factors can be used by recursive identification in the next identification procedure
`SavePInvWFactors`	[–]	saves the computed pseudo inverse sensitivity weighting factors
`SavePseudoInv`	[–]	saves the pseudo inverse sensitivity matrix
`PInvWFactorsFileName`	[–]	name of the file including the weighting factors. The default filename is 'PInvWFactors.dat'
`PseudoInvFileName`	[–]	name of the file including the pseudo inverse sensitivity matrix. The default filename is 'PInvSensMat.dat'
`DirName4SavingPFile`	[–]	name of the directory where the data are saved (if the directory does not exist, the directory will be created)

Optimization

The necessary settings for the applied optimization procedure are included in the subdirectory `/Optimization` in every entry of the toplevel directory `/Identification`.



The meaning of these entries is as follows:

{`BundleTrust`, `SQP`, `GGPRV`, `ACO`}	*selectable*	type of optimization procedure
{`Rosen`, `BFGS`}	*selectable*	type of approximation of Hessian
{`Bisection`, `GoldenSect`, `Gold_Armijo`}	*selectable*	type of line search for `GGPRV`
`RunPart`	[−]	considered pathways for `ACO` in percent
`MaxIter`	[−]	maximum number of iterations
`MaxFuncCalls`	[−]	maximum number of function calls
`NofSubGrad`	[−]	maximum number of stored subgradients (≥ 3, only `BundleTrust`)
`MachinePrec`	[−]	machine precision
`StopCriterion`	[−]	stopping criterion
`MinStep`	[`Time`]	minimum time stepsize during identification
`ifail`	[−]	Parameter for showing reason of terminating: 0: The optimality conditions are satisfied. 1: The algorithm has been stopped after `MaxIter` iterations. 2: The algorithm computed an uphill search direction. 3: Underflow occured when determining a new approximation matrix for the Hessian of the Lagrangian. 4: More than `MaxFunCalls` function calls are required during the line search. 5: Length of a working array is too short. 7: The search direction is close to zero, but the current iterate is still infeasible. > 10: The solution of the quadratic of least squares subproblem has been terminated with an error message IFQL > 0 and ifail is set to IFQL $+ 10$.

Note — Optimization Procedures

- `BundleTrust`:
  algorithm for nonsmooth functions.
- `SQP`:
  sequential quadratic programming method for smooth functions; modified BFGS-update; $L2$-augmented Lagrangian penalty function/$L1$-penalty function for the line search.
- `GGPRV`:
  gradient-projection method for smooth functions; (unmodified) BFGS-update or method of Rosen; bisection method, golden section method, or Goldstein-Armijo principle for the line search.
- `ACO`:
  ant colony optimization algorithm (non-derivative method, for standard identification only).

Error Analysis

A subdirectory `/ErrorAnalysis` contains some data for the error analysis and looks similar to this:

```
ErrorAnalysis
├─StopCriterion
│  ├─Type
│  │  ├─None
│  │  ├─FuncReduce
│  │  ├─MueCond
│  │  └─MueMax
│  └─Tolerance
├─MueCond
├─MueMax
├─EuklidNorm
├─MaxNorm
├─Func_Value
├─Est_Exp
└─Est_Dev
```

The meanings of the above entries are as follows:

{None, FuncReduce, MueCond, MueMax}	*selectable*	type of stopping criterion for the multi-level algorithm
Tolerance	[−]	tolerance parameter
MueCond	[−]	condition number
MueMax	[−]	maximum error amplification
EuklidNorm	[−]	Euklidian norm of the residuals
MaxNorm	[−]	maximum norm of the residuals
Trace	[−]	trace of the covariance matrix
Determinant	[−]	determinant of the covariance matrix (product of the eigenvalues $\neq 0$
Func_Value	[−]	value of the error functionals
Est_Exp	[∗]	estimated expectation of the residuals
Est_Dev	[∗]	estimated standard deviation of the residuals

Options

The subdirectory `/Identification/[identification name]/SaveIdentData` contains some options for the formfree identification. It looks like this:

```
Options
├─ Start
├─ End
├─ Step
├─ NumDPoints
└─ Scale
```

The meanings of the above entries are as follows:

Start	[−]	number of discrete points or scale index at the start
End	[−]	number of discrete points or scale index at the end
Step	[−]	step length
NumDPoints	[−]	current number of discrete points
Scale	[−]	current scale index

Save Identification Data

The subdirectory `/Identification/[identification name]/SaveIdentData` contains some saving options. It looks in general like this:

```
SaveIdentData
  ├─ SaveOptData
  ├─ SaveSensitivities
  ├─ SaveCorrelCoeff
  ├─ ErrorDataFileName
  ├─ OptParamValuesFileName
  ├─ SensitivitiesFileName
  ├─ CorrelCoeffsFileName
  └─ DirectoryName4Saving
```

The meanings of the above entries are as follows:

`SaveData`	[–]	save option for the identification results: `ParameterValues.dat` includes the identified optimal value of the coefficients; `ErrorData.dat` gives information about the minimized function; `measurementname.dat` prints the measured and the simulated data for the appropriate variable
`SaveSensitivities`	[–]	save option for the sensitivities of the identification results: `Sensitivities.dat` prints the sensitivities of each measurement variable and each time point
`SaveCorrelCoeffs`	[–]	save option for the correlation coefficients: `CorrelCoeffs.dat`
`SaveDataRecursive`	[–]	save option for recursive identification calls
`ActRecursiveStorage`	[–]	parameter show the actual number of recursive identification
`DirectoryName4Saving`	[–]	name of the directory where the data are saved (if the directory does not exist, the directory will be created)

Plots

For the identification in the case of standard parametrization we can generate particular plots where the fixed measured data points are given along with the calculated solution corresponding to the current coefficients.

First, you have to single out one of the measurements for the current plot:

- Go to the directory `/Identification`,
- choose one of the identification sets you have defined before,
- go to the directory `/[identification name]/Measurements` corresponding to your favorite identification,
- choose one of the measurements *(by clicking the right mouse button)*.

Note
In the case of a single identification with just one measurement these steps can be omitted because because the corresponding measurement will be highlighted automatically.

Now we can create the appropriate plot:

- Go to the directory `/Library/Plot`,
- choose the command `GLMeasPlot` and specify a name for the plot,
- execute the command `Plot` in the `/Command` directory.

Starting the Identification

If an identification was created and at least one coefficient function was added, the identification can be started:

- Select a identification in the toplevel directory `/Identification`,
- specify the corresponding entries,
- the formfree identification runs
  - one step by executing the command `SingleLevel` or
  - a complete multi-level process by executing the command `MultiLevel`.

2.8 Experimental Design

General Framework

For the identification of coefficient functions we can specify a experiment design problem (currently available only for identification by standard parametrization).

To define the experiment design problem, go to the directory `/Library/ExpDesign`, which offers several commands to build the experiment design problem.

```
Library
└─ExpDesign
   ├─CreateExpDesign
   └─AddScalarBC
```

First, you have to create the identification problem. Then you have to follow these steps:

- Go to the top level directory `/Identification` and select the entry pertaining the experiment design problem,
- go to the directory `/Library/ExpDesign`,

- generate a basic entry in the top level directory `/ExpDesign` with the command `CreateExpDesign` and specify the name.

A basic entry in the toplevel directory `/ExpDesign` looks like:

```
ExpDesign
└─ [name]
   ├─ OptData
   ├─ OptCriterion
   │  ├─ MueCond
   │  ├─ MueMax
   │  ├─ Trace
   │  └─ Determinant
   ├─ Procedure
   │  ├─ BundleTrust
   │  ├─ Hooke
   │  └─ SimAnnealing
   ├─ ObjFuncEvals
   ├─ MaxFuncCalls
   ├─ NofSubGrad
   ├─ MaxIter
   ├─ Rho
   ├─ StopCriterion
   ├─ StepSize4FD
   ├─ StartTemp
   ├─ DeltaTemp
   ├─ WithDataError
   └─ WithCompWeights
```

The meaning of the entries is as follows:

{MueCond, MueMax, Trace, Determinant}	*selectable*	type of the objective function
{BundleTrust, Hooke, SimAnnealing}	*selectable*	type of the optimization procedure
ObjFuncEvals	[−]	number of used evaluations of the objective function
MaxFuncCalls	[−]	max. number of evaluations of the objective function
NofSubGrad	[−]	max. number of stored subgradients (≥ 3, only for BundleTrust)
MaxIter	[−]	max. number of iterations
Rho	[−]	stepsize multiplicator ($0 <$ Rho < 1, only for Hooke)
StopCriterion	[−]	stopping criterion (for Hooke and BundleTrust)
StepSize4FD	[−]	relative stepsize for finite difference approximation of the subgradient of the objective function (only for BundleTrust)
StartTemp	[−]	start temperature for simulated annealing
DeltaTemp	[−]	stepsize of temperature for simulated annealing
WithDataError	*flag*	flag for using a data noise in measurements for the identification
WithCompWeights	*flag*	flag for using the command ComputeWeight in the identification

Note

- If the flag WithDataError is on, the generated measurements for the identification are disturbed by a Gaussian noise with the amplitude specified in the entries Epsilon of the subdirectories of the corresponding directory /Identification/[identification name]/Measurements.
- The directory /OptData is used to include problem-specific entries.

Problem Classes

The experiment design problem must be completed by the specification of the optimization variables. You can consider the following problem:

Optimization of scalar boundary conditions. To complete the experiment design problem of optimization a scalar boundary condition you have to follow these steps:

- Go to the top level directory /BoundCond and select the boundary condition, which is to be optimized,
- go to the the directory /Library/ExpDesign and execute the command

`AddScalarBC` to add the boundary condition to the selected entry in the top level directory `/ExpDesign`.

In this case the directory `/OptData` contains the following entries:

```
OptData
├─Boundary
│  ├─Left
│  └─Right
├─Type
│  ├─AllFree
│  ├─FirstFix
│  ├─LastFix
│  └─First+LastFix
├─LowerBound
└─UpperBound
```

The meaning of these entries is as follows:

{`Left`, `Right`}	*selectable*	optimization of the left or right boundary condition
{`AllFree`, `FirstFix`, `LastFix`, `First+LastFix`}	*selectable*	selection to fix some variables
`LowerBound`	[*]	lower boundary of optimization variables
`UpperBound`	[*]	upper boundary of optimization variables

Note — Optimization variables

- The optimization variables are the elements of the array `F(Time)` of the subdirectory `Left` or menuitemRight (selected in `/OptData/Boundary`) of the added scalar boundary condition.
- Only the part of `F(Time)` is optimized, which corresponds to the monotone part of `Time` (cf.2.3.2 Specifying the Boundary Condition for a Component, p. 12).
- By selecting an entry of the directory `/OptData/Type` you can optimize all variables or fix some variables.

Starting the Experiment Designing

If the experiment design problem is correctly created and the corresponding entries are specified, the command `ExpDesigning` of the top level directory `/Command` starts the experiment designing for the selected problem in the top level directory `/ExpDesign`.

2.9 Plot

Graphical output of calculated data and coefficient functions can be invoked with RICHY1D's plotting feature. To set up a plot, you have to specify first the variable or coefficient function you are going to plot and secondly, whether you want to visualize its space or time dependence.

To select a problem specific variable, follow these steps:

- Go to the toplevel directory `/Discretization`,
- select the discretization name linked to the problem, you are going to choose a variable from,
- go to the directory `/Discretization/[discretization name]/Variables` and select the variable.

To select a problem spezific coefficient function, follow these steps:

- Go to the toplevel directory `/Coefficient`,
- select the coefficient name linked to the problem, you are going to choose a coefficient function from,
- go to the directory `/Coefficient/[coefficient name]/Functions` and select the function.

To generate a plot of this variable or coefficient function, follow these steps:

- Go to the directory `/Library/Plot` (you are offered at least free commands: one for the time-, one for the space dependent plot and one for plotting a function),
- execute one of these commands and specify the name of the plot.

To edit plot specific parameters go to the directory `/Plot/[name]` and change the corresponding values:

```
Plot
└─ [name]
   ├─ Active
   ├─ Coordinate
   ├─ NumInBlock
   ├─ Subdomain
   ├─ XAutoRange
   ├─ XRange
   ├─ YAutoRange
   └─ YRange
```

The meanings of the entries are as follows:

`Active`	*flag*	flag for activating/deactivating the plot
`Coordinate`[1]	[`Length`]	Evaluation point for variable
`NumInBlock`[2]	[–]	component if variable is a vector
`Subdomain`[3]	[–]	selection subdomain
`XAutoRange`	*flag*	flag for autoscaling of horizontal axis
`XRange`	[`Length`] or [`Time`]	left and right boundary of horizontal axis (unit depends on type of plot)
`YAutoRange`	*flag*	flag for autoscaling of vertical axis
`YRange`	[∗]	lower and upper boundary of vertical axis
`Steps2Save`[1]	[–]	`SaveData` in `/Command` directory saves every `Steps2Save` th time point and corresponding value
`Resolution`[1]	[–]	sets the number of lines to be drawn at least in the plot (only graphical output, not calculation)

Note

- For a `TimePlot` specify the point of evaluation and ensure that this `Coordinate` is inside of your domain.
- If the variable of the plot is vector valued (see description of the problem class, 3 Problem Classes, p. 57), you can select the component of the vector with `NumInBlock`.
- The plot will be scaled automatically, if you choose `Left` > `Right` or `Lower` > `Upper`, even if `AutoRange` is not active.
- You may not generate a `GridPlot` or `TimePlot` before the timer is present.

➟ `/Command/Plot`
This command sequentially executes all plots.

➟ `/Command/SaveData`
This command saves the data of the selected `GridPlot` or `TimePlot` into a file. You are asked for a file name. If no plot is available or a `FunctionPlot` is selected, the selected variable of the selected discretization is saved.

[1] only for `TimePlot`
[2] only for `GridPlot` and `TimePlot`
[3] only for `FunctionPlot`

2.10 General Settings

The variables of the toplevel directory `/GeneralSettings` are used to control arbitrary behavior of RICHY1D. Further details can be found in the documentation of the corresponding parts.

➟ `/GeneralSettings/ShowMessages`

This command enables/disables the popping up of message boxes, e. g.when new species are added (in the multicomponent problems) or different problems are connected to each other (carrier facilitation or water flow with solute transport). You normally have to hit the 'Ok' button to confirm the messages. If you do not want that messages pop up (e. g.because you do not want that the execution of script files is interrupted by this confirmation), just switch the button off (by default, it is on).

➟ `/GeneralSettings/SaveDataSettings`

In this directory you can define the number of digits `NumOfDigits` of the values that you save when the `SaveData` command is executed for the current plot. The `CutOff` value sets every datum that should be saved to 0.0, if the absolute value of the datum is smaller or equal to `CutOff`. `CutOff` is only relevant for grid plots or time plots, it has no effect on the calculations themselves, though!

➟ `/GeneralSettings/ShowHistoryLog`

The command `ShowHistoryLog` envokes a window that displays all the operations/actions that you perform during your session (e. g.execution of any command, changing directories, setting parameters or selecting items).

2.11 Geological Database

Overview

`GeoBase` is an interface to transfer physicochemical properties of the soil and the contaminant to the input data of certain problem classes of RICHY1D, e. g.transport.

The physicochemical properties are available through ASCII files which are loaded into the application and build up a database that can be found in the toplevel directory `/GeoBase`. The user may also modify any properties within RICHY1D, choose any contaminant from the database and build up a soil structure from the textures defined in the database. A single command will transfer the user's settings into the structures of RICHY1D.

By default, the database is not loaded automatically when RICHY1D is started, therefore you have to execute the script file `./appl/scripts/geobase.scr`. Then you have access to a number of soil properties and contaminant data, which are mainly based on data of the Umweltbundesamt UBA. Of course the user is free to modify these databases, or create its own database files. Therefore he just has to modify the files `./appl/scripts/geobase/contamiants/contaminants.scr` or `./appl/scripts/geobase/soiltextures/soiltextures.scr` or create similar new files of that kind. The directory structure of GeoBase

```
GeoBase
├─ Data2Transport
├─ Data2Heat
├─ Contaminant
├─ SorptionModel
└─ SoilTexture
```

is explained in the following sections.

Contaminants

Under the toplevel directory `/GeoBase` there is a directory `Contaminants`, which itself has the subdirectory `DataBase`. The database contains one directory for each contaminant. The entries in this directory are

```
GeoBase
└─Contaminant
  └─DataBase
    └─[name of contaminant]
      ├─MolarMass
      ├─Density
      ├─MeltingPoint
      ├─BoilingPoint
      ├─VapourPressure
      ├─RelGasDensity
      ├─WaterSolubility
      ├─SaturationConcentration
      ├─DynamicViscosity
      ├─KinematicViscosity
      ├─DiffusionWater
      ├─DiffusionAir
      ├─DissociationConstant
      ├─HenryCoefficient
      ├─K_OW
      └─K_OC
```

Most of these entries are still unused and we will only explain and give units for the used entries.

`WaterSolubility`	[g/cm^3]	
`DiffusionWater`	[cm^2/min]	molecular diffusion coefficient in water
`K_OC`	[−]	partition coefficient between organic carbon and water

Within this database you only have to select the desired contaminant, then the specified parameter will be used for the equilibrium sorption model, see the following section, and the diffusion constant.

Sorption Models

The second directory under `/GeoBase` is `SorptionModel`. The sorption models are mathematical formulations of isotherms built from physical properties. They differ from each other by the set of physical properties used to formulate an isotherm and by the mathematical formula that is used to define the isotherm. The following model in the `GeoBase` framework is available:
Linear, depending on organic carbon and clay content. The directory of this sorption model looks like

```
GeoBase
├─Contaminant
├─SorptionModel
│  └─Linear(OC,T)
│     ├─ClayCutOff
│     ├─KT(S)_Factor
│     └─KT(S)_Exponent
```

This model represents a linear sorption isotherm for equilibrium sorption. It is a familiar way to write the sorbed concentration as product of bulk density, K_d value and concentration of the contaminant. K_d is calculated from the contaminant's `WaterSolubility`, it's partition coefficient K_{OC}, the soil's `OrganicCarbon` and `Clay` content.

$$K_d = K_{OC}\,\frac{\texttt{OrganicCarbon}}{100} + \frac{\texttt{Clay}}{100}\,\texttt{KT(S)\_Factor}\,\exp\left(\texttt{KT(S)\_Exponent}\,\ln(\texttt{WaterSolubility})\right)$$

The second term in this sum, containing the clay dependence of sorption, is neglected if `Clay` $<$ `ClayCutOff`. You have to select the desired sorption model.

Soil Texture

The last subdirectory under `/GeoBase` is `SoilTexture`

```
GeoBase
├─Contaminant
├─SorptionModel
├─SoilTexture
│  ├─DataBase
│  ├─AddLayer
│  ├─RemoveLayer
│  └─Layers
```

and contains a database of soil properties, commands to add a layer of the selected property or to remove a selected layer and a list of layers that builds the soil.

➟ `/GeoBase/SoilTexture/DataBase`
The database contains one directory for each specified soil type. The entries in this directory are

```
GeoBase
  ├─Contaminant
  ├─SorptionModel
  ├─SoilTexture
  │  ├─DataBase
  │  │  ├─[name of soil property]
  │  │  │  ├─BulkDensity
  │  │  │  ├─ClayContent
  │  │  │  ├─SiltContent
  │  │  │  ├─SandContent
  │  │  │  ├─FieldCapacity
  │  │  │  ├─AvailableWater
  │  │  │  ├─ResidualWaterContent
  │  │  │  ├─Porosity
  │  │  │  ├─Permeability
  │  │  │  ├─QuartzContent
  │  │  │  ├─OtherMineral
  │  │  │  ├─SesquioxidesContent
  │  │  │  └─OrganicMatter
```

Most of these entries are still unused by the available sorption models within `GeoBase` and we will only explain and give units for the used entries.

`BulkDensity`	[g/cm^3]
`ClayContent`	[%]
`OrganicCarbon`	[%]
`FieldCapacity`	[%]
`QuartzContent`	[%]
`OtherMinerals`	[%]
`SesquioxidesContent`	[%]
`OrganicMatter`	[%]

Within this database you have to select the soil property, from which you want to add a layer to your soil.

➟ `/GeoBase/SoilTexture/AddLayer`
This command adds a layer of the previously selected soil property to the list of layers in that directory (see below).

➟ `/GeoBase/SoilTexture/RemoveLayer`
This command removes the selected layer from that directory (see below).

➟ `/GeoBase/SoilTexture/Layers`
The directory `Layers` contains all layers, which you

- add by the command `AddLayer`, after
- selecting the soil property from the `DataBase`.

You can also remove a layer from this directory by selecting this layer and using the command `RemoveLayer`.

Each layer is represented by a directory named by the soil property, containing the same information like the entries of any soil property and in addition the `Thickness` of the layer and the content of `OrganicCarbon`

```
GeoBase
├─Contaminant
├─SorptionModel
└─SoilTexture
  ├─DataBase
  ├─AddLayer
  └─Layers
    └─[name of soil property
      ├─Thickness
      ├─OrganicCarbon
      ├─BulkDensity
      └─ClayContent
```

The sequence of layers in this directory is arranged from top to bottom. The first layer is the toplayer in your soil. The second layer is the layer below the first layer and so on.

Interface to Problem

➟ `Data2[problem]`

The command `Data2[roblem]` is only available for heat and transport problems. This command requires that a domain is allocated (it uses the selected domain) and that the coefficient functions of the corresponding problem are allocated (it uses the selected coefficients). If these requirements are fullfilled, the command

1. fills the information of your soil texture (the sequence of layers) into the domain definition and
2. for each layer it combines the soil properties and the information for the problem coefficents (detailed below) and fills the corresponding data into the `Coefficients` of the `Problem`.

Note

Be aware that RICHY1D enumerates the layers along the axis in positive direction. If you read any data outside of `GeoBase` which are defined per subdomain, the index 0 is that of the bottom layer and the toplayer has the largest index number.

Some parameters in the database are given with units. You have to ensure that all other parameters are defined consistently to these units.

In detail, currently the following information is transferred from the `GeoBase` to the `Coefficients` of the `Problem`:

➟ `/Geobase/Data2Heat`

Besides the configuration of the layers which is used to build the domain, the following data are transferred to the heat coefficients:

The contents of quartz, other minerals, sesquioxides and natural matter of the layers.

➟ `/Geobase/Data2Transport`

Besides the configuration of the layers which is used to build the domain, the following data are transferred to the transport coefficients:

The (equilibrium) sorption model is transferred as explained in the section sorption models, p. 52. Furthermore the values for the field capacity (→ water content), the molecular diffusion of the contaminant species, and the bulk density of the soil layers are read in.

Chapter 3

Problem Classes

Each problem class deals with a certain model equation and provides at least a set of coefficient functions and a discretization for this equation. The model equations are in vectorial form and have to be condidered for only one spatial dimension in the framework of RICHY1D.

3.1 Heat Conduction

Notation

T	[`Temperature`]	temperature
c_ν	[`Mass`/(`Time`^{2} `Length`^{2} `Temperature`^{2})]	volumetric storage capacity of soil
λ	[`Mass Length`/(`Time`^{3} `Temperature`)]	thermal conductivity of soil
c_w	[`Mass`/(`Time`^{2} `Length Temperature`)]	volumetric heat capacity of water
$\boldsymbol{q}$	[`Length`/`Time`]	water flux
f	[`Mass`/(`Time`^{3} `Length`)]	source
θ	[$-$]	volumetric water content
θ_sat	[$-$]	saturated water content
f_M	[$-$]	volumetric fraction of minerals
f_O	[$-$]	volumetric fraction of organic matter

Model Equation

The conduction of heat in a porous medium by convection and diffusion is described by this equation (cf.e. g. de Vries and Afgan 1975):

$$\partial_t \left(c_\nu(\theta)\, T\right) - \boldsymbol{\nabla} \cdot \left(\lambda(\theta)\, \boldsymbol{\nabla} T\right) + c_\mathrm{w}\, \boldsymbol{\nabla} \cdot \left(T\, \boldsymbol{q}\right) \;=\; f \tag{3.1}$$

It reduces to the classical equation of heat conduction (in a bar, e. g.) for constant storage capacity c_ν and conductivity λ, and vanishing convective part (i. e., $\boldsymbol{q} = \boldsymbol{0}$).

Coefficient Functions

To create the set of coefficient functions for heat conduction follow these steps:

- Go to the directory `/Library/Coefficient`,
- execute `HeatConduction` and define a name, under which the corresponding parameter set appears in the toplevel directory `/Coefficient`.

Heat Conduction

To assign or change the parameter values, RICHY1D offers the following interface:

```
Coefficient
└─ [name]
   ├─ StorageCapacity
   ├─ Conductivity
   ├─ HomogSource
   └─ Water
```

The parameters correspond to those, introduced in the notations table above. The parameters are displayed as arrays. You have to specify values for each subdomain.

To calculate the solution of the heat equation (3.1) define the coefficients `StorageCapacity`, `Conductivity` and `HomogSource`. These coefficients are piecewise constant on subdomains.

Heat Transport

To simulate heat transport in connection with water flow in soils, you have to switch on the flag `WaterDependend` in the `Water` subdirectory. Now additional terms in your equation are considered that account for convective transport and relate the other coefficients to the water content (see the model equation (3.1)). The formulations of these coefficient functions are based on work of de Vries and Afgan (1975), see also Totsche (1996).

You have either the possibility to define values for the water content θ and the water flux q in the menu (in this case, switch `WithRichards` off), or couple a simulation of a water flow problem (the Richards equation) to the heat conduction problem (in this case, switch `WithRichards` on). In the latter case, the current water contents and fluxes are taken from the water flow problem, as well as the value for the saturated water content θ_{sat} (in this case, the menu variable for `ThetaSat` is ignored.). RICHY1D offers you the following interface:

```
Coefficient
└─ [name]
   ├─ StorageCapacity
   ├─ Conductivity
   ├─ HomogSource
   └─ Water
      ├─ WaterDependend
      ├─ WithRichards
      ├─ WaterContent
      ├─ Time
      ├─ Flux
      ├─ HeatCapWater
      ├─ LambdaWater
      ├─ LambdaQuartz
      ├─ LambdaOtherMinerals
      ├─ LambdaSesquiOxides
      ├─ LambdaOrganicMatter
      ├─ LambdaAir
      ├─ ThetaSat
      ├─ XQuartz
      ├─ XOtherMinerals
      ├─ XSesquiOxides
      ├─ XOrganicMatter
      ├─ PrefFlowNumInBlock
      ├─ Connect2WaterContent
      └─ Connect2Flux
```

To provide the information concerning the water content θ and the water flux q for the heat conduction problem follow these steps:

1. Select `WithRichards`.
2. Define `HomogSource`, `OrganicMatter` and `HeatCapWater`.
3. Go to the toplevel directory `/Discretization` and select the discretization corresponding to the water flow (e. g.`WaterDisc`).
4. Go into the directory `/Discretization/WaterDisc/Variables` and select the water content.
5. Execute the command `/Coefficient/[name]/Water/Connect2WaterContent`.
6. Go into the directory `/Discretization/[water discretization name]/Variables` and select `flux`.
7. Execute the command `/Coefficient/SoluteCoeff/Water/Connect2Flux`.

8. Set `PrefFlowNumInBlock` to define the number of Richards equation in problem class `PrefFlow` (cf.3.4 Saturated/Unsaturated Preferential Water Flow (Richards Equation), p. 84). If water flow is simulated by problem class `Richards`, `PrefFlowNumInBlock` has to be set to 0.

Note
The simulation of water flow has to be added to the grid before the heat transport.

If `WaterDependend` is selected, the volumetric storage capacity of soil c_ν is defined with the help of empirically derived weighting factors according to de Vries and Afgan (1975)

$$c_\nu \;=\; 0.46\, f_\mathrm{M} + 0.60\, f_\mathrm{O} + 1.0\, \theta\;. \tag{3.2}$$

f_M and f_O are the volumetric fractions of minerals and organic matter, respectively. The thermal conductivity of the soil λ has the following form:

$$\lambda(\theta) \;=\; \frac{\sum_{i=1}^{5} \lambda_i\, x_i\, k_i}{\sum_{i=1}^{5} x_i\, k_i} \qquad\qquad x_i \;=\; \frac{V_i}{\sum_{j=0}^{6} V_j}\;. \tag{3.3}$$

The volumetric fractions for water and air are not given in the menu, because x_water corresponds to the water content θ and $x_\mathrm{air} = \theta_\mathrm{sat} - \theta$. k_i is currently set to 1.0. Otherwise (in the case you switch `WaterDependend` off) c_ν is set to `StorageCapacity`, λ to `Conductivity`.

Discretization

The heat conduction is discretized with a conforming finite element method, mass lumpimg and a backward Euler scheme. To create the corresponding discretization follow these steps:

- Go to the directory `/Library/Discretization`,
- execute `FEM4HeatConduction` and define a name, under which the discretization appears in the toplevel directory `/Discretization`.

Now, you have access (e. g.for selection) to the variables of this discretization in the toplevel directory `/Discretization`.

```
Discretization
└─ [name]
   └─ Variables
      └─ temperature
```

The entry in the directory `Variables` corresponds to that, explained in the notations table above.

Grid-and time adaption. The discretization `FEM4HeatConduction` does not provide any indicators for grid and time adaption.

Instructions for Usage

The problem class heat conduction handles a scalar initial boundary value problem. Therefore you have to add

- an initial boundary value problem consisting of
  - a domain,
  - a scalar boundary condition (`Type` `Dirichlet`, `Neumann` or `Flux` for temperature),
  - a scalar initial value (1 vector with 1 component, prescribing temperature at the beginning of the simulation),
  - the set of coefficient functions `HeatConduction` and
- the discretization `FEM4HeatConduction`

to the grid. The result of this discretization is a piecewise linear approximation of the temperature on the grid. As the problem is linear, a nonlinear solver should only need one iteration to solve one time step.

3.2 Solute Transport

Table of Contents

Notation

t	[Time]	time
c	[Mass/Length^3]	mass concentration of solute
s	[−]	mass concentration of sorbate at nonequilibrium sorption sites
θ	[−]	volumetric water content
$\boldsymbol{q}$	[Length/Time]	water flux
ρ_b	[Mass/Length^3]	bulk density
f_ϕ, f_φ	[−]	mass fraction of sorption sites
$\phi(c)$	[−]	equilibrium sorption isotherm
$\varphi(c)$	[−]	nonequilibrium sorption isotherm
r	[1/Time]	rate parameter for nonequilibrium sorption
$\mathbf{D}$	[Length^2/Time]	diffusion-dispersion tensor
α_l	[Length]	longitudinal dispersivity
d	[Length^2/Time]	molecular diffusion
k	[1/Time]	first order decay rate
const	[Mass/(Length^3 Time)]	constant zeroth order decay
k_r	[1/Time]	exchange rate between solution and organic phase
X	[−]	mole fraction within organic phase
S_p	[Mass/Length^3]	pure solubility in water

Model Equation

The solute transport is described with a transport equation including diffusion-dispersion, convection, equilibrium and nonequilibrium sorption, decay and Raoult's law driven by first order exchange kinetic:

$$\partial_t(\theta\, c) - \boldsymbol{\nabla}\cdot(\mathbf{D}\nabla c - \boldsymbol{q}\, c) + \rho_\mathrm{b}\,\partial_t\left(f_\phi\,\phi(c) + f_\varphi\, s(c)\right) \;=\; -\theta\, k\, c - \theta\,\mathrm{const} - \theta\, k_\mathrm{r}\,(X\, S_\mathrm{p} - c)\;, \tag{3.4a}$$

$$\partial_t s \;=\; r\,(\varphi(c) - s)\;. \tag{3.4b}$$

Coefficient Functions

The main directory looks like:

```
Coefficient
└─ [name]
   ├─ Functions
   ├─ Transport
   ├─ Carrier
   ├─ Water
   ├─ Decay
   └─ Stabilization4Transport
```

Solute Transport

The interface to these parameters is as follows:

```
Coefficient
└─ [name]
   └─ Transport
      ├─ Diffusion
      ├─ Dispersion
      ├─ BulkDensity
      ├─ EquiSorpt
      │  ├─ ParaType
      │  ├─ MassFraction
      │  ├─ Kd-Value
      │  ├─ Exponent
      │  ├─ Regularization
      │  ├─ MaxSorption
      │  └─ Link2Coeff
      ├─ NonEquiSorpt
      │  └─ AddIsotherm
      └─ Raoult
         ├─ MolarFraction
         ├─ PureSolubility
         └─ ExchangeRate
```

In case of existing nonequilibrium sorbtion the subdirectory `Coefficient/[name]/Transport/Non\ EquiSorpt` looks like

```
NonEquiSorpt
  AddIsotherm
  NEIsothermA
    ParaType
    MassFraction
    Kd-Value
    Exponent
    Regularization
    MaxSorption
    RateParameter
    Link2Coeff
```

Note

By using the default setting `RateParameter` $= -1$ the corresponding nonequilibrium sorption isotherm will be ignored.

The solute transport provides four different sorption isotherms both for equilibrium and for nonequilibrium sorption. The `ParaType` controls the evaluation of this isotherms on each subdomain in the following way:

ParaType	name	equation
0	linear	$\varphi_{\text{lin}} = K_{\text{d}}\, c$
1	Freundlich	$\varphi_{\text{F}} = K_{\text{d}}\, c^{\texttt{Exponent}}$
2	Langmuir	$\varphi_{\text{L}} = \dfrac{K_{\text{d}}\, c}{1 + \frac{K_{\text{d}}}{\texttt{MaxSorption}}\, c}$
3	Freundlich–Langmuir	$\varphi_{\text{FL}} = \dfrac{K_{\text{d}}\, c^{\text{Exponent}}}{1 + \frac{K_{\text{d}}}{\texttt{MaxSorption}}\, c^{\text{Exponent}}}$

The Freundlich and the Freundlich–Langmuir isotherms can be regularized within the interval [0, `Regularization`]. For `Exponent` < 1 the isotherms are continuously substituted within this interval.

The diffusion-dispersion tensor has the following form in one space dimension:

$$D = \alpha_{\text{l}}\, q + d\, \theta\,. \tag{3.5}$$

In more space dimensions there exist several matrix representations.

Release of organic substances from the Non-Aqueous Phase Liquid (NAPL) phase is often controlled by Raoult's law. The parametrization of this source/sink term comprises the `MolarFraction` of the organic substance in the NAPL phase, the solubility of the pure substance in water (`PureSolubility`) and the`ExchangeRate` between aqueous solution and NAPL.

Multiple Non-Equilibrium Isotherms

It is possible to add several kinetic isotherms to the transport problem. An isotherm is added by clicking the command-button `AddIsotherm`:

```
Coefficient
  [name]
    Transport
      Diffusion
      Dispersion
      BulkDensity
      EquiSorpt
      NonEquiSorpt
        AddIsotherm
      Raoult
```

The added kinetic isotherms are displayed and the coefficients can be set in the same way as they are set for the equilibrium isotherms.

It is necessary to specify new initial values for each added isotherm. In order to do this, add a vector to describe the initial values for the sorbate and a function for each new isotherm to this vector.

In order to select plots for the different sorption places, the corresponding `NumInBlock` values must be specified according following table:

[isotherm name]	NumInBlock
NEIsothermA	0
NEIsothermB	1
NEIsothermC	2
...	...

If a domain is divided into subdomains, it is possible to deactivate these isotherms in some subdomains by specifying negative rateparameters for these subdomains.

Note

It is necessary to create a dicretisation prior to the addition of non equilibrium isotherms. The grid has to be reinitialized after adding new isotherms.

Carrier Facilitation

The effect of mobile and immobile carriers (e. g.dissolved organic carbon, DOC) on solute transport can be included. According to the work of Knabner et al. (1996) and Totsche et al. (1996) the contaminant's sorption isotherm ψ is replaced by an effective sorption isotherm ψ_{eff}. Following experimental studies (Magee et al. (1991), Roy and Dzombak (1997)) the partitioning of the solute (phenanthrene) to DOC is modelled by a linear exchange mechanism (cf.Prechtel et al. (2002) for an application). The

resulting effective isotherm consists of a term for the sorption of the free contaminant and a term for the sorption of the carrier bound contaminant:

$$f_\psi \psi_{\text{eff}}(c) = \underbrace{f_{\psi_{Hf}} \psi_{Hf} \left(\frac{c}{1 + c_D K_{HD}} \right)}_{\text{sorption of free HOC}} + \underbrace{f_{\psi_D} \psi_D(c_D) K_{HD} \frac{c}{1 + c_D K_{HD}}}_{\text{sorption of carrier bound HOC}} \tag{3.6}$$

where c is the total (free + carrier bound) dissolved concentration of the contaminant, $f_{\psi_{Hf}}$ and f_{ψ_D} are the mass fractions of the solid matrix providing equilibrium sorption sites for free HOC (phenanthrene) and for DOC respectively, ψ_{Hf} and ψ_D are the equilibrium sorption isotherms for free phenanthrene and for DOC, c_D is the current solute concentration of the carrier, and K_{HD} is the partition coefficient for the linear sorption of HOC to DOC. Note that the term describing the sorption of carrier bound HOC only takes cumulative sorption into account, i. e. the binding of dissolved HOC to sorbed DOC. A term describing co-sorption, i. e. the sorption of HOC-loaded DOC particles could be easily added, but at the moment, the experimental distinction between the two effects is still questionable. For the derivation, discussion and further properties of the model (also with respect to other approaches), see Knabner et al. (1996), Totsche et al. (1996) and **KS1996**.

Sorption of the free solute is described in the menu `Transport` as usual. Additionally the solute now may sorb to a carrier substance, which itself is subject to transport, sorption etc.

Carrier facilitation is switched on and off with the flag `WithCarrier`. The interface to carrier facilitation is as follows:

```
Coefficient
└─SoluteCoeff
  ├─Transport
  ├─Carrier
  │ ├─WithCarrier
  │ ├─PartitioningCoefficient
  │ ├─Connect2Solute
  │ └─Connect2Sorbate
  ├─Water
  └─Decay
```

To use this feature, you need another problem on your grid, that describes the transport of the carrier. You can use e. g.the problem class solute transport twice. First, you create the problem and its discretization for the carrier. And then, you add the same problem and discretization a second time to the grid for the solute that is subject to carrier facilitated transport. To establish the connection between the current solute transport and a previously defined carrier transport follow these steps:

- Switch `WithCarrier` on,
- define a `PartitioningCoefficient` for the sorption of your solute to the carrier,

- go to the toplevel directory `/Discretization` and select the discretization corresponding to the carrier (e. g.`CarrierDisc`),
- go into the directory `/Discretization/CarrierDisc/Variables` and select the concentration of the carrier,
- execute the command `/Coefficient/SoluteCoeff/Carrier/Connect2Solute`,
- go into the directory `/Discretization/CarrierDisc/Variables` and select the sorbed concentration of the carrier,
- execute the command `/Coefficient/SoluteCoeff/Carrier/Connect2Sorbate`.

Note
The carrier transport has to be added to the grid before the (carrier facilitated) solute transport.

Water Flow

RICHY1D's interface to define the water flow for your transport problem is as follows:

```
Coefficient
└─SoluteCoeff
  ├─Transport
  ├─Carrier
  └─Water
    ├─WithRichards
    ├─WaterContent
    ├─Time
    ├─Flux(Time)
    ├─Connect2WaterContent
    └─Connect2Flux
```

The simulation of solute transport depends on a description of water flow i. e., of the water flux q and the water content θ. There are two possibilities to provide this information for solute transport:

1. Predetermined constant values for water content θ on each subdomain and possibly time dependent values for water flux q. To make use of this method, follow these steps:
   - Switch `WithRichards` off,
   - define `WaterContent` for each subdomain and
   - define `Time` and `Flux(Time)` for the whole domain. Values are interpolated piecewise linear at intermediate time points.

2. Simulation of water flow. To make use this method, you need the problem and its discretization, that describes water flow, on your grid. Follow these steps:
   - Switch `WithRichards` on,
   - go to the toplevel directory `/Discretization` and select the discretization corresponding to the water flow (e. g.`WaterDisc`),
   - enter the directory `/Discretization/WaterDisc/Variables` and select the water content,
   - execute the command `/Coefficient/SoluteCoeff/Water/Connect2WaterContent`,
   - enter the directory `/Discretization/WaterDisc/Variables` and select the water flux,
   - execute the command `/Coefficient/SoluteCoeff/Water/Connect2Flux`.

Note
The simulation of water flow has to be added to the grid before the solute transport.

Serial Decay Reactions

Let introduce the following notations:

c_i	[`Mass/Length`^{3}]	concentration of species i
c_{i-1}	[`Mass/Length`^{3}]	concentration of species $i-1$
k_i, k_{i-1}	[`1/Time`]	first order decay rates for c_i and c_{i-1}
y_i	[$-$]	stoichiometric factor
const	[`Mass/Time`]	constant zeroth order decay

It is possible to couple several transport/decay reactions that are described by the following model equations:

$$\partial_t(\theta\, c_i) - \boldsymbol{\nabla}\cdot(\mathbf{D}\nabla c_i - \boldsymbol{q}\, c_i) + \rho_\mathrm{b}\,\partial_t\left(f_\phi\,\phi(c_i) + f_\varphi\, s_i(c_i)\right) \;=\; y_i\, k_{i-1}\, c_{i-1} - k_i\, c_i - \mathrm{const} \tag{3.7a}$$

$$\partial_t s_i \;=\; r\,(\varphi(c_i) - s_i) \tag{3.7b}$$

Note
A constant zeroth order decay is independent of the current solute concentration and thus may lead to negative concentrations! This is not a numerical error but a consequence of the zerothth order model (which is not appropriate in such a situation).

For each species only one father species may be given, for the first problem $i = 0$ you have to set c_{i-1} to zero. Two examples of serial decay rates are shown in figure 3.1.

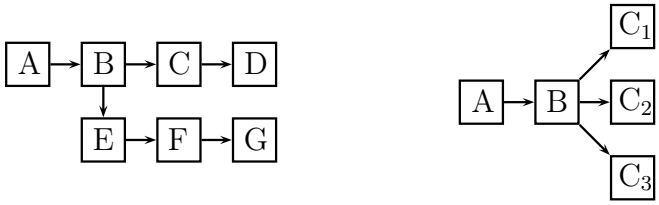


Figure 3.1: Example of two decay series.

Coupling the reactions. The interface to serial decay is as follows:

```
Coefficient
└─ReactionB
   ├─Transport
   ├─Carrier
   ├─Water
   └─Decay
      ├─0thOrderRate
      ├─1stOrderRate
      ├─WithFatherSpecies
      ├─YieldFactor
      ├─Connect2Father
      └─Connect2SubComp
```

To use this feature, you have to create several transport problems (and their discretization) on your grid, that describe the transport of the species 1 to i. It is also possible to connect your transport problem with a biodegradation problem. To establish the connection between two reactions (A $\rightarrow$ B) follow these steps:

- Switch `WithFatherSpecies` on,
- define the yield factor of the reaction,
- if the solute variable of the father reaction consists of two (or more) components (as e. g.in the biodegradation problem), define the component by setting `Connect2SubComp`. Look out: '0' identifies the first component!
- Enter the toplevel directory `/Discretization` and select the discretization corresponding to the father reaction (e. g.'ReactionA'),
- enter the directory `/Discretization/ReactionA/Variables/` and select the sink-term. If `ReactionA` is a transport problem, you have to select the variable `sink-trans` (then the sink term corresponds to a first order term). If `ReactionA` is defined in the problem class biodegradation, you have to select `sink-biodeg` (then the sink term corresponds to a Monod model).
- Enter the directory `/Coefficient/ReactionB/Decay` and execute the command `Connect2Father`.

Temperature-Dependent Kinetics of Degradation

The influence of the temperature on the kinetics of microbially mediated degradation can be described by multiplying an inhibition term

$$I(T) \;=\; \exp\left(-\kappa(T - T_{\text{opt}})^2\right) \;. \tag{3.8a}$$

Thus the zeroth order term becomes

$$\text{const} \;\longrightarrow\; \text{const}_{\text{opt}}\, I(T) \tag{3.8b}$$

and the zeroth order rate constant can now be interpreted as the zeroth order constant at optimal temperature. Analogously the first order rate becomes

$$k \;\longrightarrow\; k_{\text{opt}}\, I(T) \tag{3.8c}$$

where

T	[Temp]	current temperature
T_{opt}	[Temp]	optimal temperature for biodegradation
κ	[1/Temp^2]	inhibition coefficient

The Interface to this feature is as follows:

```
Coefficient
└─ Transport
   └─ Decay
      ├─ 0thOrderF(Temp)
      ├─ OptimalT40thOrder
      ├─ Kappa40thOrder
      ├─ 1stOrderF(Temp)
      ├─ OptimalT41stOrder
      ├─ Kappa41stOrder
      ├─ WithHeatConduction
      ├─ Connect2HeatConduction
      ├─ Temperature
      │  ├─ Add2Display
      │  └─ TempMeasurement1
```

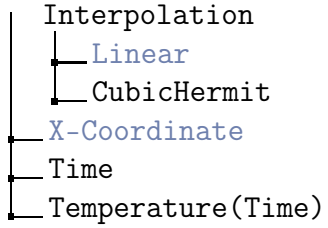
These entries have the following meaning:

0thOrderF(Temp) 1stOrderF(Temp)	flag to specify whether temperature has an influence on each degradation
OptimalT40thOrder OptimalT41stOrder	temperature at which the biodegradation takes place at maximum speed for zeroth order and first order decay
Kappa40thOrder Kappa41stOrder	inhibition term for first and zeroth order decay
WithHeatConduction	flag which specifies whether temperature distribution is calculated by heat conduction
Connect2HeatConduction	command to connect heat conduction problem to transport problem
Add2Display	command to add temperature measurements for a new coordinate
TempMeasurement1	directory in which the temperature measurements for one coordinate can be specified

In order to model temperature dependent decay reactions, the temporal evolution of the temperature in the whole domain has to be specified. This can be done in two ways:

1. The temperature can be specified explicitly as a function of time and space.
2. The transport problem can be coupled with a heat conduction problem.

In order to specify the temperature explicitly, use the command `Add2Display` in order to add new temperature measurements for each coordinate: The directory `/TemperatureMeasurement` contains following entries:



The interpolation which has to be specified refers to the evaluation with respect to time. The interpolation between the coordinates is done linearily. It is important to specify the coordinates in an ordered way ($x_1 < x_2 < \ldots < x_n$). Outside of the specified intervals the values will be kept constant.

In order to couple the transport problem with the heat conduction problem, a problem of the type heat conduction must be defined prior to the following steps:

- Switch WithHeatConduction on,
- go to the toplevel directory `/Discretization` and select the discretization corresponding to the heat conduction problem (e. g.`HcDisc`),
- enter the directory `/Discretization/HcDisc/Variables` and select `temperature`,

- execute the command `/Coefficient/Transport/Decay/Connect2HeatConduction`,
- select the flag `/Coefficient/Transport/Decay/WithHeatConduction`.

It is possible to determine the parameters κ and T_{opt} by inverse modelling as well as to model temperature dependent serial decay reactions. Though it is necessary to specify the explicit temporal evolution of the temperature for each species as well as the coupling with the heat conduction must be done for each species.

Linking Isotherms of Different Problems

If at least two transport problems with separate `Coefficient` are defined, then an isotherm can be linked to an isotherm from another `Coefficient`. This is especially of interest to identify an isotherm from two experiments which use, e. g., different flux conditions.

Example

Let two entries—named `CoefficientA` and `CoefficientB`—are defined in the toplevel directory `/Coefficient`. The equilibrium isotherm of `CoefficientB` is to be linked to the equilibrium isotherm of `CoefficientA`. The link is created in the following way:

- Go to the toplevel directory `/Coefficient` and select the entry `CoefficientA`,
- go into the directory `/CoefficientA/Functions` and select `EquiIsotherm`,
- go into the directory `/CoefficientB/Transport/EquiSorpt` and execute `Link2Coeff`.

Now, `CoefficientB` uses the same equilibrium isotherm data as `CoefficientA` (for simulation and identification).

Usage

To create a set of coefficient functions for solute transport follow these steps:

- Go to the directory `/Library/Coefficient`,
- execute `Transport` and define a name, under which the corresponding parameter set appears in the toplevel directory `/Coefficient`.

Discretization

The solute transport is discretized with a conforming finite element method, mass lumping and a backward Euler scheme. To create the corresponding discretization follow these steps:

- Go to the directory `/Library/Discretization`,

- execute `FEM4Transport` and define a name, under which the discretization appears in the toplevel directory `/Discretization`.

Now you have access (e. g.for selection) to the variables of this discretization in the toplevel directory `/Discretization`.

```
Discretization
├── Stabilization4Transport
│   └── ArtificialDiffusion
└── [name]
    └── Variables
        ├── solute
        ├── sorbate(neq)
        ├── sorbate(eq)
        ├── solute_sens
        ├── sorbate(neq)_sens
        ├── sorbate(eq)_sens
        └── sink-trans
```

The entry `ArtificialDiffusion` is the parameter of the stabilization (cf.A.1 Stabilization, p. 130). The entries in the directory `Variables` correspond to those explained in the notations table above. The variable `sink-trans` refers to the sink-term by first-order decay of your transport problem. This sink-term may enter as a source term in a subsequent transport problem (see the section about serial decay reactions above). Note that it does not contain the zeroth order decay term!

A variable `[name]_sens` has 22 components and includes the sensitivities of the variable `[name]` with respect to small changes in the parameters of the coefficient functions. The assignment is as follows:

NumInBlock	[parameter]	NumInBlock	[parameter]
0	Diffusion	13	RateParameter
1	Dispersion	14	PartitioningCoefficient
2	BulkDensity	15	K_sat(Richards)
3	Flux	16	T_sat(Richards)
4	WaterContent	17	T_res(Richards)
5	MassFraction(EquiSorpt)	18	n(Richards)
6	Kd-Value(EquiSorpt)	19	alpha(Richards)
7	Exponent(EquiSorpt)	20	p(Richards)
8	MaxSorption(EquiSorpt)	21	beta(Richards)
9	MassFraction(NonEquiSorpt)	22	1stOrderRate
10	Kd-Value(NonEquiSorpt)	23	YieldFactor
11	Exponent(NonEquiSorpt)	24	1stOrderRate(FatherSpecies)
12	MaxSorption(NonEquiSorpt)	25	0thOrderRate

Instructions for Usage

The problem class solute transport handles a scalar initial boundary value problem for the solute concentration together with an initial value problem for the(nonequilibrium-)sorbed concentration. Therefore you have to add

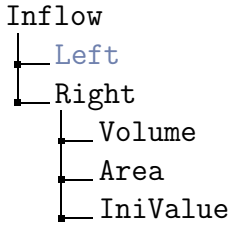
- an initial boundary value problem consisting of
  - a domain,
  - a scalar boundary condition (`Type Dirichlet`, `Neumann` or `Flux`) or the closed-flow boundary condition (see below), an initial value with two vectors (1^{st} vector for the solute and 2^{nd} vector for the (nonequilibrium-)sorbed concentration). If `Flux` conditions (q_{in}, c_{in}) are specified, the given values are the inflow concentrations, which will be multiplied by the given water flux, taken from the coefficients menu, or the Richards equation, if you have coupled the problems (so you need not recalculate the boundary conditions when the water flux changes),
  - the set of coefficient functions `Transport` and
- the discretization `FEM4Transport` to the grid.

The result of this discretization is a piecewise linear approximation of the solute- and the (nonequilibrium-)sorbed concentration. Any further variables

(e. g.(equilibrium-)sorbed concentration) can be derived from these variables. The variable that a nonlinear solver has to bring to zero is... By this way the stopping criterion of the nonlinear solver bounds the...

Closed-Flow Design

For the solute transport a closed-flow design of a column can be created by executing the command `Closed4Transport` in the directory `/Library/BoundCond` and specifying a name. The menu entries of the boundary condition for the closed-flow design look like:



The meanings of these entries are as follows:

{`Left` / `Right`}	*selectable*	position of the inflow boundary
`Volume`	[`Length`^{3}]	volume of the reservoir
`Area`	[`Length`^{2}]	inflow (=outflow) area
`IniValue`	[`Mass`/`Length`^{3}]	initial value of mass concentration in the reservoir

Identification

The problem class solute transport allows the identification of sorptions isotherms. For identification you can use the `OLS` error functional (output least squares), the `OLS_TW` error functional (output least squares with weighting by time differences) or the `OLS_SW` error functional (output least squares with adaptive weighting by sensitivity values).

The directory `/Coefficient/[coefficient name]/Functions/NonEquiIsotherm/FittingData` contains two additional entries for the formfree identification:

`RB_Lower`	[`1/Time`]	lower boundaries for rate parameter
`RB_Upper`	[`1/Time`]	upper boundaries for rate parameter

Additionally to the identification of sorption isotherms and the according coefficient functions it is possible to identify initial conditions. This requires to put up the initial conditions in the following way:

- Go to the directory `/Library/InitialValue`,

- generate a basic entry in the toplevel directory `/InitialValue` with the command `Create` and specify its name (e. g.'Pollutant'),
- for each vector of your problem class add an entry to `/InitialValue/Pollutant` by executing `AddVector` and specify its name (e. g.'Sorbate'),
- for each component of an already defined vector (e. g.`/InitialValue/Pollutant/Sorbate`) add an entry by executing e. g.`ConstFunctions` and specify the name for the initial value function of this component (e. g.'Comp').
- Within an array constant in the subdirectory `/InitialValue/Pollutant/Sorbate/Comp` a concentration value can be specified for each subdomain.

After creation of the identification (`CreateParaFit` or `CreateIdentify`) it is possible to specify the identification controls.

3.3 Saturated/Unsaturated Water Flow (Richards Equation)

Table of Contents

Notation

t	[`Time`]	time
z	[`Length`]	coordinate, opposite to the gravitational direction
ψ	[`Length`]	pressure head
$\boldsymbol{q}$	[`Length/Time`]	water flux
θ	[$-$]	volumetric water content (coefficient function)
θ_{res}	[$-$]	residual water content
θ_{sat}	[$-$]	saturated water content
$\mathbf{K}$	[`Length/Time`]	hydraulic conductivity (coefficient function; $\mathbf{K} = K_{\text{rel}}\,\mathbf{K}_{\text{sat}}$)
K_{rel}	[$-$]	relative hydraulic conductivity ($K_{\text{rel}} = K_{\text{rel}}(\theta)$)
$\mathbf{K}_{\text{sat}}$	[`Length/Time`]	saturated hydraulic conductivity
α, β	[`1/Length`]	parameter for the pressure–saturation curve
m, n, p	[$-$]	parameter for the pressure–saturation curve

Model Equation

The saturated/unsaturated flow of water in a porous media is described by the Richards equation (Richards 1931):

$$\partial_t \theta + \boldsymbol{\nabla} \cdot \boldsymbol{q} \;=\; 0\,, \qquad\qquad \boldsymbol{q} \;=\; -\mathbf{K}(\theta)\,\boldsymbol{\nabla}(\psi + z)\,. \tag{3.9}$$

In one spatial dimension, the *unsaturated hydraulic conductivity* tensor $\mathbf{K}$ becomes a scalar K. The primary unknown variable of this model equation is the *pressure head* ψ.

Coefficient Functions

In saturated regions, i. e., where $\psi > 0$ holds, the water content and the conductivity are equal to $\theta = \theta_{\text{sat}}$, $K = K_{\text{sat}}$ (due to $K_{\text{rel}} = 1$). Regarding unsaturated regions three parametrizations of the coefficient functions are available for the one-dimensional *Richards equation*:

Gardner

$$\theta_{\text{exp}}(\psi) \;=\; \theta_{\text{res}} + (\theta_{\text{sat}} - \theta_{\text{res}})\, e^{\alpha\,\psi}\;, \tag{3.10a}$$

$$K_{\text{exp}}(\psi) \;=\; K_{\text{sat}}\, e^{\alpha\,\psi}\;. \tag{3.10b}$$

Haverkamp

$$\theta_{\text{Hav}}(\psi) \;=\; \theta_{\text{res}} + \frac{\theta_{\text{sat}} - \theta_{\text{res}}}{1 + (-\alpha\,\psi)^n}\;, \tag{3.11a}$$

$$K_{\text{Hav}}(\psi) \;=\; \frac{K_{\text{sat}}}{1 + (-\beta\,\psi)^p}\;. \tag{3.11b}$$

van-Genuchten–Mualem

$$\theta_{\text{vG}}(\psi) \;=\; \theta_{\text{res}} + (\theta_{\text{sat}} - \theta_{\text{res}})\,\Phi(\psi)\;, \tag{3.12a}$$

$$K_{\text{vG}}(\psi) \;=\; K_{\text{sat}}\sqrt{\Phi(\psi)}\left(1 - \left(1 - \Phi(\psi)^{1/m}\right)^m\right)^2 \tag{3.12b}$$

$$\text{with} \qquad \Phi(\psi) \;=\; \frac{1}{(1 + (-\alpha\,\psi)^n)^m}\;, \qquad m \;=\; 1 - 1/n\;.$$

To create this set of coefficient functions for the Richards equation we follow these steps:

- go to the directory `/Library/Coefficient`,
- execute `Richards` and define a name, under which the corresponding parameter set appears in the toplevel directory `/Coefficient`.

Typical values for the parameters of the van-Genuchten parametrization are found in the next table (Blume 2011, Tab. 2.1):

	θ_{res} [−]	θ_{sat} [−]	K_{sat} [cm s^{-1}]	α [cm^{-1}]	n [−]
sand	0.03	0.32	2.2×10^{-3}	2.3×10^{-2}	4.17
silt	0.01	0.41	1.0×10^{-3}	7.0×10^{-3}	1.30
loam	0.00	0.43	3.0×10^{-4}	1.6×10^{-2}	1.25

To select the parametrization type, assign or change the parameter values, RICHY1D offers the following interface:

```
Coefficient
└─ [name]
   ├─ Functions
   │  └─ ...
   ├─ ParaType
   ├─ K_sat
   ├─ T_sat
   ├─ T_sat_approx
   │  ├─ Connect2Porosity
   │  └─ WithPorosity
   ├─ T_res
   ├─ n
   ├─ alpha
   ├─ p
   ├─ beta
   └─ Regularization
```

The meaning of most of the parameters in this directory is explained in the notation table above. The parameters are displayed as arrays. You have to specify values for each subdomain. The `ParaType` defines the used parameter function per subdomain as follows from this table:

name of estimation model	`ParaType`
Gardner	0
Haverkamp	1
van-Genuchten–Mualem	2
generic spline interpolation	4

Variable Porosity

In the case you want to include the effect of varying mineral concentrations on the total porosity (which we do not distinguish from the saturated water content θ_{sat} in our models), you can connect a reactive multispecies problem (see Reactive Multicomponent Transport, p. 113) with the water flow problem.

The porosity can be described in terms of mineral concentrations by

$$\theta_{\text{sat}} = 1 - \sum_m V m_m \bar{c}_m \qquad [\texttt{Length}^3\ \texttt{pore}/\texttt{Length}^3\ \texttt{bulk}] \tag{3.13}$$

where Vm_m [$\texttt{Length}^3\ \texttt{m}^{\texttt{th}}\ \texttt{mineral}/\texttt{Mass}$] denotes the **molar volume** of the mth mineral species which is assumed to be constant. According to eqn. (3.13), a change in mineral concentrations directly influences the porosity and therewith the water movement. The residual water content θ_{res} is a measure for the minimum water volume held in the pore space by capillary forces. Assuming that the microscopic structure of the

soil matrix does not change with respect to variable porosity (i.e. there is no forming of ridges or fingers) the following scaling is performed:

$$\theta_{\text{res}} \longleftarrow \phi/\phi^0\theta_{\text{res}}$$

where ϕ^0 denotes the initial porosity.

In the subdirectory `T_sat_approx` set the variable `withPorosity` to on. Then go to

➟ `/Discretization/[MultispecProblem]/Variables/porosity`
Set porosity on and highlight the variable with a right-click. Go to

➟ `/Coefficiemt/[RichardsProblem]/T_sat_approx`
and execute the command `Connect2Porosity`.

Note — Regularization
Not all the parameters are active for each function although the menu always contains the complete set of parameters for every model. The entry `Regularization` refers to the van-Genuchten–Mualem model: for $n < 2$, the permeability function K_{vG} must be regularized (cf. Blume 2011, Sec. 2.1.5). The permeability is continuous differentiable substituted by a quadratic function within the range $[1 - \texttt{Regularization}, 1]$.

Initial Value

The problem class Richards equation provides the additional initial value function `RichyEqui`, which can be used instead of the standard piecewise linear function defined within the kernel. Instead of the parametrization of piecewise linear function you have now only the parameter `Flux` to define the initial pressure profile. This works as follows:

➟ `/Library/InitialValue/RichyEqui`
The command `RichyEqui` invokes the computation of an equilibrium initial condition for the (un)saturated water flow problem. Initialization calculates the pressure and water content distribution within the domain using the `Dirichlet` boundary condition at the left side of the domain for time $t = 0$ and the specified `Flux` within the domain.

Note
If you adjust the boundary condition (`Type Flux`) at the right hand side of the domain to the value you specified for the initial value and if you keep boundary condition values constant in time, your problem will remain stationary.

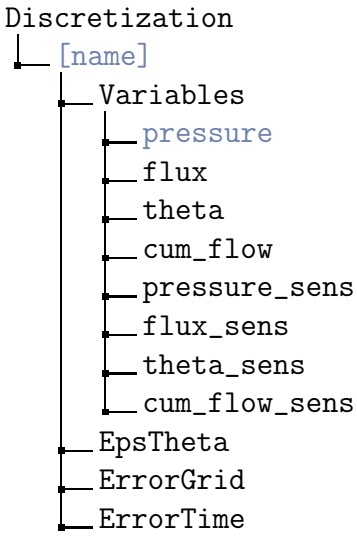
Discretization

The Richards equation is discretized with a mixed hybrid finite element method. To create the corresponding discretization follow these steps:

- go to the directory `/Library/Discretization`,

- execute `HMFEM4Richards` and define a name, under which the discretization appears in the toplevel directory `/Discretization`.

Now, you have access (e. g. for selection) to the variables of this discretization in the toplevel directory `/Discretization`.



The meaning of the entries `pressure`, `flux` and `theta` in the directory `Variables` is explained in the notations table above. The variable `cum_flow` contains the cumulative flow.

A variable `[name]_sens` has seven components and includes the sensitivities of the variable name with respect to small changes in the parameters of the coefficient functions. The assignment is as follows:

NumInBlock	parameter
0	K_{sat}
1	θ_{sat}
2	θ_{res}
3	n
4	α
5	p
6	β

Grid- and Time Adaptation

The discretization `HMFEM4Richards` provides error indicators for grid and time adaptation. If you activate adaptivity grid size and timestep size is controlled by the corresponding parameters: `ErrorGrid` and `ErrorTime`.

Instructions for Usage

The problem class Richards equation handles a scalar initial boundary value problem. Therefore you have to add

- an initial boundary value problem consisting of
  - a domain,
  - a scalar boundary condition (`Type Dirichlet` for pressure or `Type Flux` for water flux),
  - a scalar initial value (1 vector with 1 component, prescribing pressure at the beginning of the simulation),
  - the set of coefficient functions `Richards` and
- the discretization `HMFEM4Richards`

to the grid. The result of this discretization is

- a piecewise constant approximation of the pressure on each element,
- a pressure value on the edges between two elements and
- a piecewise linear and continous flux.

Any further variables (e. g.`theta`, `cum_flow`) can be derived from these variables.

The variable that a nonlinear solver has to bring to zero is the flux difference on every edge between two elements. By this way the stopping criterion of the nonlinear solver bounds the discontinouity of the flux across edges.

Identification

The Richards equation allows us the identification of the hydraulic properties. For the identification you can use the output least squares error functionals `OLS`, `OLS_TW` and `OLS_SW`. The error functional `Mualem_Pen` combines the output least squares identification with a regularization by the Mualem relation. The subdirectory `/Mualem_Pen` contains two additional parameters:

`PenaltyDom`	[−]	tolerance parameter for the penalty term
`PenaltyPar`	[−]	regularization parameter

3.4 Saturated/Unsaturated Preferential Water Flow (Richards Equation)

Table of Contents

Notation

t	[`Time`]	time
z	[`Length`]	coordinate, opposite to the gravitational direction
ψ	[`Length`]	pressure head
$\boldsymbol{q}$	[`Length/Time`]	water flux
θ	[$-$]	volumetric water content (coefficient function)
θ_{res}	[$-$]	residual water content
θ_{sat}	[$-$]	saturated water content
$\mathbf{K}$	[`Length/Time`]	hydraulic conductivity (coefficient function; $\mathbf{K} = K_{\mathrm{rel}}\,\mathbf{K}_{\mathrm{sat}}$)
K_{rel}	[$-$]	relative hydraulic conductivity ($K_{\mathrm{rel}} = K_{\mathrm{rel}}(\theta)$)
$\mathbf{K}_{\mathrm{sat}}$	[`Length/Time`]	saturated hydraulic conductivity
α, β	[`1/Length`]	parameter
m, n, p	[$-$]	parameter
Γ	[`1/Time`]	exchange term
$a_{\mathrm{Pl}}^{(ij)}$	[`1/(Length Time)`]	mass exchange coefficient reg. phase $i \to$ phase j
$a_{\mathrm{Pnl}}^{(ij)}$	[`1/Length`^{2}]	mass exchange coefficient reg. phase $i \to$ phase j
$a_{\mathrm{S}}^{(ij)}$	[`1/Time`]	mass exchange coefficient reg. phase $i \to$ phase j
N_{P}	[$-$]	number of pore systems/Richards equations

Model Equation

Consider a structured porous media consisting of N_P pore systems. The saturated/unsaturated flow of water in pore system i is described by the Richards equation:

$$\partial_t \theta_i + \boldsymbol{\nabla} \cdot \boldsymbol{q}_i + \Gamma_i \;=\; 0 \,, \qquad\qquad \boldsymbol{q}_i \;=\; -\mathbf{K}_i(\theta_i)\, \boldsymbol{\nabla}(\psi_i + z) \,. \tag{3.14}$$

In one spatial dimension, the unsaturated hydraulic conductivity tensor $\mathbf{K}_i$ becomes a scalar K_i. The primary unknown variable of this model equation is the pressure head ψ_i.

Coefficient Functions

In saturated regions, i. e., where $\psi > 0$ holds, the water content and the conductivity are equal to $\theta = \theta_\mathrm{sat}$, $K = K_\mathrm{sat}$ (due to $K_\mathrm{rel} = 1$). Regarding unsaturated regions three parametrizations of the coefficient functions are available for the one-dimensional Richards equation:

Gardner

$$\theta_\mathrm{exp}(\psi) \;=\; \theta_\mathrm{res} + (\theta_\mathrm{sat} - \theta_\mathrm{res})\, e^{\alpha\,\psi} \,, \tag{3.15a}$$

$$K_\mathrm{exp}(\psi) \;=\; K_\mathrm{sat}\, e^{\alpha\,\psi} \,. \tag{3.15b}$$

Haverkamp

$$\theta_\mathrm{Hav}(\psi) \;=\; \theta_\mathrm{res} + \frac{\theta_\mathrm{sat} - \theta_\mathrm{res}}{1 + (-\alpha\,\psi)^n} \,, \tag{3.16a}$$

$$K_\mathrm{Hav}(\psi) \;=\; \frac{K_\mathrm{sat}}{1 + (-\beta\,\psi)^p} \,. \tag{3.16b}$$

van-Genuchten–Mualem

$$\theta_\mathrm{vG}(\psi) \;=\; \theta_\mathrm{res} + (\theta_\mathrm{sat} - \theta_\mathrm{res})\, \Phi(\psi) \,, \tag{3.17a}$$

$$K_\mathrm{vG}(\psi) \;=\; K_\mathrm{sat} \sqrt{\Phi(\psi)} \left(1 - \left(1 - \Phi(\psi)^{1/m}\right)^m\right)^2 \tag{3.17b}$$

$$\text{with} \qquad \Phi(\psi) \;=\; \frac{1}{(1 + (-\alpha\,\psi)^n)^m} \,, \qquad m \;=\; 1 - 1/n \,.$$

To create this set of coefficient functions for Richards equation follow these steps:

- Go to the directory `/Library/Coefficient`,
- execute `PrefFlow` and define a name, under which the corresponding parameter set appears in the toplevel directory `/Coefficient`.

To add a Richards equation use the interface

```
Coefficient
└─ [name]
   ├─ Functions
   ├─ Source
   ├─ Exchangeterm
   ├─ SingleEquations
   │  └─ AddRichy
```

and execute `AddRichy` and enter an equation name.

To select the parametrization type, assign or change the parameter values, RICHY1D offers the following interface:

```
Coefficient
└─ [name]
   ├─ Functions
   ├─ Source
   ├─ Exchangeterm
   ├─ SingleEquations
   │  ├─ AddRichy
   │  └─ Equation01:[equation name]
   │     ├─ ParaType
   │     ├─ K_sat
   │     ├─ T_sat
   │     ├─ T_res
   │     ├─ n
   │     ├─ alpha
   │     ├─ p
   │     ├─ beta
   │     ├─ Regularization
   │     └─ a
```

The meaning of most of the parameters in this directory is explained in the notations table above. The parameters are displayed as arrays. You have to specify values for each subdomain. The `ParaType` defines the used parameter function per subdomain as follows from this table:

name of estimation model	ParaType
Gardner	0
Haverkamp	1
van-Genuchten–Mualem	2

Note

Not all the parameters are active for each function although the menu always contains the complete set of parameters for every model. The `Regularization` is used for the van-Genuchten–Mualem model. For $n < 2$ the permeability function must be

regularized. The permeability is continuous differentiable substituted by a quadratic function within the range [1 − `Regularization`, 1].

Exchange Term

For the Richards equations four exchange terms are available:

No exchange

$$\Gamma_0^{(i)} \;=\; 0\,. \tag{3.18}$$

Pressure difference—linear

$$\Gamma_{\mathrm{Pl}}^{(i)} \;=\; \sum_{\substack{j=1\\ j\neq i}}^{R} a_{\mathrm{Pl}}^{(ij)}\,(\psi_i - \psi_j)\,. \tag{3.19}$$

Pressure difference—nonlinear

$$\Gamma_{\mathrm{Pn}}^{(i)} \;=\; \sum_{\substack{j=1\\ j\neq i}}^{R} a_{\mathrm{Pnl}}^{(ij)}\,\frac{K_i(\psi_i) - K_j(\psi_j)}{2}\,(\psi_i - \psi_j)\,. \tag{3.20}$$

Saturation difference

$$\Gamma_{\mathrm{S}}^{(i)} \;=\; \sum_{\substack{j=1\\ j\neq i}}^{R} a_{\mathrm{S}}^{(ij)}\,(\Theta^{(i)} - \Theta^{(j)})\,, \qquad \Theta^{(k)} \;=\; \frac{\theta_k - \theta_{\mathrm{res}}^{(k)}}{\theta_{\mathrm{sat}}^{(k)} - \theta_{\mathrm{res}}^{(k)}}\,. \tag{3.21}$$

An exchange term can be chosen by

`Exchangeterm`	**exchange type**
0	no exchange
1	pressure difference linear
2	pressure difference nonlinear
3	saturation difference

All equations of the model use the same type of exchange term. To choose the type of exchange term and the parameters use the interface above.

Initial Value

The problem class preferential flow provides the additional initial value function `PrefFlowEqui`, which can be used instead of the standard piecewise linear function defined within the kernel. Instead of the parametrization of piecewise linear function you have now only the parameter `Flux` to define the initial pressure profile. This works as follows:

➟ `/Library/InitialValue/PrefFlowEqui`

The command `RichyEqui` invokes the computation of an equilibrium initial condition for the (un)saturated water flow problem. Initialization calculates the pressure and water content distribution within the domain using the `Dirichlet` boundary condition at the left side of the domain for time $t = 0$ and the specified `Flux` within the domain.

Note

If you adjust the boundary condition (`Type Flux`) at the right hand side of the domain to the value you specified for the initial value and if you keep boundary condition values constant in time, your problem will remain stationary.

Discretization

The Richards equation is discretized with a mixed hybrid finite element method. To create the corresponding discretization follow these steps:

- Go to the directory `/Library/Discretization`,
- execute `HMFEM4PrefFlow` and define a name, under which the discretization appears in the toplevel directory `/Discretization`.

Now, you have access (e. g.for selection) to the variables of this discretization in the toplevel directory `/Discretization`.

```
Discretization
└── [name]
    ├── Variables
    │   ├── pressure
    │   ├── flux
    │   ├── theta
    │   ├── cum_flow
    │   ├── pressure_sens
    │   ├── flux_sens
    │   ├── theta_sens
    │   └── cum_flow_sens
    ├── EpsTheta
    ├── ErrorGrid
    └── ErrorTime
```

The meaning of the entries `pressure`, `flux` and `theta` in the directory `Variables` is explained in the notations table above. The variable `cum_flow` contains the cumulative flow.

A variable `[name]_sens` has seven components and includes the sensitivities of the variable name with respect to small changes in the parameters of the coefficient functions. The assignment is as follows:

NumInBlock	parameter
0	K_{sat}
1	θ_{sat}
2	θ_{res}
3	n
4	α
5	p
6	β

Grid- and Time Adaptation

The discretization `HMFEM4PrefFlow` provides error indicators for grid and time adaptation. If you activate adaptivity grid size and timestep size is controlled by the corresponding parameters: `ErrorGrid` and `ErrorTime`.

Instructions for Usage

The problem class preferential flow handles a scalar initial boundary value problem. Therefore you have to add

- an initial boundary value problem consisting of
  - a domain,
  - a scalar boundary condition (`Type Dirichlet` for pressure or `Type Flux` for water flux),
  - a scalar initial value (1 vector with 1 component, prescribing pressure at the beginning of the simulation),
  - the set of coefficient functions `PrefFlow` and
- the discretization `HMFEM4PrefFlow`

to the grid. The result of this discretization is

- a piecewise constant approximation of the pressure on each element,
- a pressure value on the edges between two elements and

- a piecewise linear and continous flux.

Any further variables (e. g.`theta`, `cum_flow`) can be derived from these variables.

The variable that a nonlinear solver has to bring to zero is the flux difference on every edge between two elements. By this way the stopping criterion of the nonlinear solver bounds the discontinouity of the flux across edges.

Identification

The identification of parameters will be included in a later version. For problems with only one Richards equation use the problem class (un-)saturated water flow (see 3.3 Saturated/Unsaturated Water Flow (Richards Equation), p. 78).

3.5 Coupled Water-Surfactant Transport

Table of Contents

Notation

t	[Time]	time
z	[Length]	coordinate, opposite to the gravitational direction
ψ	[Length]	pressure head
$\boldsymbol{q}$	[Length/Time]	water flux
c	[Mass/Length^3]	surfactant concentration
θ	[−]	volumetric water content (coefficient function)
θ_{res}	[−]	residual water content
θ_{sat}	[−]	saturated water content
$\mathbf{K}$	[Length/Time]	hydraulic conductivity (coefficient function; $\mathbf{K} = K_{\text{rel}}\,\mathbf{K}_{\text{sat}}$)
K_{rel}	[−]	relative hydraulic conductivity ($K_{\text{rel}} = K_{\text{rel}}(\theta)$)
$\mathbf{K}_{\text{sat}}$	[Length/Time]	saturated hydraulic conductivity
$\mathbf{K}_{\text{coarse}}$, $\mathbf{K}_{\text{clay}}$	[Length/Time]	hydraulic conductivity for a component of the soil
ν, ν_{clay}	[−]	volume fraction of a comp. of the soil relative to total bulk volume
σ_0		surface tension of water (without surfactant)
$\sigma(c)$		surface tension of water as function of surfactant concentration
ρ_{b}	[Mass/Length^3]	bulk density
$\rho_{\text{surfactant}}$	[Mass/Length^3]	mass density of surfactant
ϕ	[−]	equilibrium sorption isotherm
K_{d}		distribution coefficient
$\mathbf{D}$	[Length^2/Time]	diffusion-dispersion tensor
d	[Length^2/Time]	molecular diffusion
α_{l}	[Length]	longitudinal dispersivity
α, β	[1/Length]	parameter
a, b, ℓ, m, n	[−]	parameter

Model Equations

The coupled water surfactant transport is is described by Richards equation (for water flow) and by a solute transport equation (for surfactant transport) including convection, diffusion-dispersion and sorption.

$$\partial_t \theta + \boldsymbol{\nabla} \cdot \boldsymbol{q} \;=\; 0\,, \qquad \boldsymbol{q} \;=\; -\mathbf{K}(\theta)\,\boldsymbol{\nabla}(\psi + z)\,, \tag{3.22a}$$

$$\partial_t(\theta\, c) + \rho_{\text{b}}\,\partial_t \phi(c) - \boldsymbol{\nabla} \cdot (\mathbf{D}\,\boldsymbol{\nabla} c - \boldsymbol{q}\, c) \;=\; 0\,. \tag{3.22b}$$

In one spatial dimension, the unsaturated hydraulic conductivity tensor $\mathbf{K}$ becomes a scalar K. The primary unknown variable of this model equation are the pressure head of water, ψ, and the solute concentration of the surfactant c.

Coefficient- and Coupling Functions

Richards Equation

For the one-dimensional Richards equation two parametrizations of the coefficient functions are available:

Gardner

$$\theta_{\exp}(\psi) \;=\; \theta_{\text{res}} + (\theta_{\text{sat}} - \theta_{\text{res}})\, e^{\alpha\,\psi}\;, \tag{3.23a}$$

$$K_{\exp}(\psi) \;=\; K_{\text{sat}}\, e^{\alpha\,\psi}\;. \tag{3.23b}$$

van-Genuchten–Mualem

$$\theta_{\text{vG}}(\psi) \;=\; \theta_{\text{res}} + (\theta_{\text{sat}} - \theta_{\text{res}})\, \Phi(\psi)\;, \tag{3.24a}$$

$$K_{\text{vG}}(\psi) \;=\; K_{\text{sat}} \Phi(\psi)^{\ell} \left(1 - \left(1 - \Phi(\psi)^{1/m}\right)^{m}\right)^{2} \tag{3.24b}$$

$$\text{with} \qquad \Phi(\psi) \;=\; \frac{1}{(1 + (\alpha\,\psi)^{n})^{m}}\;, \qquad m \;=\; 1 - 1/n\;.$$

The interface to these parameters is as follows:

```
Coefficient
└─ [name]
   ├─ ParaType
   ├─ Water
   │  ├─ Ks
   │  ├─ Theta_s
   │  ├─ Theta_r
   │  ├─ n
   │  ├─ alpha
   │  ├─ n
   │  └─ l
   ├─ Coupling
   └─ Transport
```

Surfactant Transport Equation

For the surfactant transport equation four different parametrizations of the equilibrium sorption isotherm are available:

ParaType	name	equation
0	linear	$\phi_{\text{lin}} = K_{\text{d}}\, c$
1	Freundlich	$\phi_{\text{F}} = K_{\text{d}}\, c^{\texttt{Exponent}}$
2	Langmuir	$\phi_{\text{L}} = \dfrac{K_{\text{d}}\, c}{1 + \frac{K_{\text{d}}}{\texttt{MaxSorption}}\, c}$
3	Freundlich–Langmuir	$\phi_{\text{FL}} = \dfrac{K_{\text{d}}\, c^{\text{Exponent}}}{1 + \frac{K_{\text{d}}}{\texttt{MaxSorption}}\, c^{\text{Exponent}}}$

The Freundlich and the Freundlich–Langmuir isotherms can be regularized within the interval [0, `Regularization`]. For `Exponent` < 1 the isotherms are continuously substituted within this interval.

The diffusion-dispersion tensor has the following form in one space dimension:

$$D = \alpha_{\text{l}}\, q + d\, \theta\,. \tag{3.25}$$

In more space dimensions there exist several matrix representations.

The interface to these parameters is as follows:

```
Coefficient
└─ [name]
   ├─ ParaType
   ├─ Water
   ├─ Coupling
   ├─ Transport
   │  ├─ Diffusion
   │  ├─ Dispersion
   │  ├─ Kd
   │  ├─ Exponent
   │  ├─ Regularization
   │  ├─ MaxSorption
   │  └─ BulkDensity
```

Coupling

The Richards equation and the surfactant transport equation are coupled by the following mechanisms:

Pressure scaling. The following model is based on the work of Smith and Gillham (1994). The surface activity of surfactant inluences the surface tension of the water

phase. This physical property of the solute is represented by a scaling factor within the pressure saturation relation:

$$\theta(\psi) \;\longrightarrow\; \theta\left(\frac{\sigma_0}{\sigma(c)}\,\psi\right) \;=:\; \tilde{\theta}(c,\psi)\,, \qquad \frac{\sigma_0}{\sigma(c)} \;=\; \frac{1}{1-b\ln(c/a+1)}\,. \tag{3.26}$$

Permeability scaling. This model is based on the work of Renshaw et al. (1997). It is assumed that soil consists of a mixture of two components. One component with high permeability like for example sand, and clay as the second component. Surfactants sorb to the clay and therefore increase the volume fraction of the clay-surfactant conglomerate. It follows that the effective permeability decreases:

$$K_{\text{sat}} \;\longrightarrow\; K_{\text{eff}}(c) \;=\; K_{\text{coarse}}^{1-\nu}\,K_{\text{clay}}^{\nu}\,, \qquad \nu \;=\; \nu_{\text{clay}} + \frac{\rho_{\text{b}}}{\rho_{\text{surfactant}}}\,\phi(c)\,. \tag{3.27a}$$

Altogether, the hydraulic conductivity becomes

$$K_{\text{rel}}(\theta)\,K_{\text{sat}} \;=\; K(\theta) \;\longrightarrow\; \widetilde{K}(c,\theta) \;=\; K_{\text{rel}}(\theta)\,K_{\text{eff}}(c)\,. \tag{3.27b}$$

The conductivity of clay is adjusted, such that at vanishing surfactant concentration the saturated conductivity is equal to the value defined for Richards equation.

The interface to these parameters is as follows:

```
Coefficient
└─ [name]
   ├─ ParaType
   ├─ Water
   ├─ Coupling
   │  ├─ PressureScaling
   │  ├─ scale_a
   │  ├─ scale_b
   │  ├─ PermeabilityScaling
   │  ├─ Ks_coarse
   │  ├─ ClayVolumeFraction
   │  └─ SurfactantDensity
   └─ Transport
```

The variables `PressureScaling` and `PermeabilityScaling` are flags, that allow to switch the coupling on and off.

Usage

To create this set of coefficient- and coupling functions for the coupled water surfactant transport follow these steps:

- Go to the directory `/Library/Coefficient`,

- execute `SurfactantWater` and define a name, under which the corresponding parameter set appears in the toplevel directory `/Coefficient`.

The parameters of this set of coefficient functions are displayed as arrays. You have to specify values for each subdomain. The `ParaType` defines the used parametrization type per subdomain as follows from this table:

	exponential	van Genuchten
linear	0	10
Freundlich	1	11
Langmuir	2	12
Freundlich–Langmuir	3	13

Initial Value

The problem class surfactant water transport provides the additional initial value function `SurfWaterEqui`, which can be used instead of the standard piecewise linear function defined within the kernel. Instead of the parametrization of a piecewise linear function you have now only the parameters `Flux`, to define the initial pressure and concentration profile. `SurfWaterEqui` invokes the computation of an equilibrium initial condition for the coupled water surfactant transport. Initialization calculates the pressure and concentration distribution within the domain using the Dirichlet boundary condition at the left side of the domain for time $t = 0$ and the specified flux within the domain.

Note

If you adjust the boundary condition (`Type Flux`) at the right hand side of the domain to the value you specified for the initial value and if you keep boundary condition values constant in time, your problem will remain stationary.

Discretization

The Richards equation is discretized with a mixed hybrid finite element method. To create the corresponding discretization follow these steps:

- Go to the directory `/Library/Discretization`,
- execute `HMFEM4SurfWater` and define a name, under which the discretization appears in the toplevel directory `/Discretization`.

Now, you have access (e. g.for selection) to the variables of this discretization in the toplevel directory `/Discretization`.

```
Discretization
└─ [name]
   ├─ Variables
   │  ├─ press/conc
   │  ├─ Theta
   │  ├─ flux_water/surf
   │  └─ permeability
   └─ Weights
```

The meaning of the entries in the directory `Variables` is explained in the notations table above. The two variables `press/conc` and `flux_water/surf` have two components. In both cases the first component corresponds to water and the second to surfactant.

The `Weights` are applied to each component (water, surfactant) of the defect vector when the norm of the defect is calculated. By this method you can equilibrate the defect of water- and of surfactant flux. Depending on the units you choose, water- and surfactant flux may have values in different orders of magnitude.

Grid- and Time Adaptation

The discretization `HMFEM4SurfWater` does not provide error indicators.

Instructions for Usage

The problem class water surfactant transport handles a vector valued initial boundary value problem. The solution has two components. The first component is the pressure head and the second is the surfactant concentration. Therefore you have to add

- an initial boundary value problem consisting of
  - a domain,
  - a vector boundary condition (`Type Dirichlet` for pressure or `Type Flux` for water flux) with two components,
  - a vector initial value (1 vector with 2 components, prescribing pressure and concentration at the beginning of the simulation),
  - the set of coefficient functions `WaterSurfactant` and
- the discretization `HMFEM4WaterSurf`

to the grid. The result of this discretization is

- a piecewise constant approximation of the pressure and of the concentration on each element,
- a pressure and a concentration value on the edges between two elements and
- a piecewise linear and continous flux for water and for surfactant.

Any further variables (e. g.`Theta`) can be derived from these variables.

The variable that a nonlinear solver has to bring to zero is the flux difference on every edge between two elements. By this way the stopping criterion of the nonlinear solver bounds the discontinouity of the flux across edges.

3.6 Biodegradation

Table of Contents

Notation

Parameters for solute transport of donator and acceptor

t	[Time]	time
c_D, c_A	[Mass/Length^3]	mass concentration of solute (donator, acceptor)
c_X	[Mass/Length^3]	mass concentration of biomass
s_D, s_A	[Mass/Length^3]	mass conc. of sorbate (don., acc.) at nonequ. sorption sites
$\boldsymbol{q}$	[Length/Time]	water flux
θ	[−]	volumetric water content (coefficient function)
ρ_b	[Mass/Length^3]	bulk density
f_ϕ, f_φ	[−]	mass fraction of sorption sites
$\phi(c)$	[−]	equilibrium sorption isotherm
$\varphi(c)$	[−]	nonequilibrium sorption isotherm
r_D, r_A	[1/Time]	rate parameter for nonequilibrium sorption (don., acc.)
k_D, k_A	[1/Time]	first order decay rates for donator and acceptor
$\mathbf{D}_\mathrm{A}$, $\mathbf{D}_\mathrm{D}$	[Length^2/Time]	diffusion-dispersion for acceptor and donator
α_l	[Length]	longitudinal dispersivity
d	[Length^2/Time]	molecular diffusion
$\mathrm{const}_\mathrm{D}$, $\mathrm{const}_\mathrm{A}$	[Mass/(Length^3 Time)]	zeroth order decay constant (donator, acceptor)

Parameters for biodegradation

$\alpha_\mathrm{A/D}$	[−]	yield coefficient for acceptor
μ_max	[1/Time]	maximum substrate utilization rate
μ_D	[Mass/(Time Length^3)]	degradation rate
$c_{\mathrm{X}_\mathrm{max}}$	[Mass/Length^3]	maximum biomass concentration
K_A	[Mass/Length^3]	half maximum rate parameter for acceptor (Monod parameter)
K_D	[Mass/Length^3]	half maximum rate parameter for donator (Monod parameter)
$K_{\mathrm{I}_\mathrm{A}}$	[Mass/Length^3]	inhibition parameter for acceptor (Haldane parameter)
$K_{\mathrm{I}_\mathrm{D}}$	[Mass/Length^3]	inhibition parameter for donator (Haldane parameter)
Y	[Mass/Mass]	microb. yield coeff. (biomass formed / mass of substrate consumed)
k_X	[1/Time]	decay rate (death coefficient) for biomass

Model Equation

Biodegradation is described by a three component model:

- 1st component: Electron donator (mobile),
- 2nd component: Electron acceptor (mobile, e. g.oxygen),

- 3rd component: Biomass (immobile).

The transport model for the mobile species includes diffusion-dispersion, convection, equilibrium and nonequilibrium sorption; double Monod kinetics are used for the decay:

$$\begin{aligned} \partial_t(\theta\, c_\mathrm{D}) - \boldsymbol{\nabla}\cdot(\mathbf{D}_\mathrm{D}\,\boldsymbol{\nabla} c_\mathrm{D} - \boldsymbol{q}\, c_\mathrm{D}) + \rho_\mathrm{b}\,\partial_t\left(f_\phi\,\phi(c_\mathrm{D}) + f_\varphi\, s_\mathrm{D}(c_\mathrm{D})\right) + \\ + \mu_\mathrm{D} + \theta\, k_\mathrm{D}\, c_\mathrm{D} - \theta\,\mathrm{const}_\mathrm{D} \;=\; 0\;, \end{aligned} \tag{3.28a}$$

$$\begin{aligned} \partial_t(\theta\, c_\mathrm{A}) - \boldsymbol{\nabla}\cdot(\mathbf{D}_\mathrm{A}\,\boldsymbol{\nabla} c_\mathrm{A} - \boldsymbol{q}\, c_\mathrm{A}) + \rho_\mathrm{b}\,\partial_t\left(f_\phi\,\phi(c_\mathrm{A}) + f_\varphi\, s_\mathrm{A}(c_\mathrm{A})\right) + \\ + \alpha_{\mathrm{A/D}}\,\mu_\mathrm{D} + \theta\, k_\mathrm{A}\, c_\mathrm{A} - \theta\,\mathrm{const}_\mathrm{A} \;=\; 0\;, \end{aligned} \tag{3.28b}$$

$$\partial_t c_\mathrm{X} - \frac{Y}{\theta}\left(1 - \frac{c_\mathrm{X}}{c_{\mathrm{X}_\mathrm{max}}}\right)\mu_\mathrm{D} + \theta\, k_\mathrm{X}\, c_\mathrm{X} \;=\; 0 \tag{3.28c}$$

with

$$\partial_t s_\mathrm{D} \;=\; r_\mathrm{D}\;(\varphi(c_\mathrm{D}) - s_\mathrm{D})\;, \tag{3.28d}$$

$$\partial_t s_\mathrm{A} \;=\; r_\mathrm{A}\;(\varphi(c_\mathrm{A}) - s_\mathrm{A}) \tag{3.28e}$$

and

$$\mu_\mathrm{D} \;:=\; \theta\,\mu_\mathrm{max}\left(\frac{c_\mathrm{D}}{K_\mathrm{D} + c_\mathrm{D} + c_\mathrm{D}^2/K_{\mathrm{I}_\mathrm{D}}}\right)\left(\frac{c_\mathrm{A}}{K_\mathrm{A} + c_\mathrm{A} + c_\mathrm{A}^2/K_{\mathrm{I}_\mathrm{A}}}\right) c_\mathrm{X}\;. \tag{3.28f}$$

Coefficient Functions

The interface for a biodegradation problem in the highest level looks like:

```
Coefficient
└─ [coefficient name]
   ├─ Functions
   ├─ Donator
   ├─ Acceptor
   ├─ Transport
   ├─ Water
   ├─ Biodegradation
   └─ Stabilization4Biodeg
```

Solute Transport

The interface to the parameters for the mobile components of the donator (and the acceptor as well) is as follows:

```
Coefficient
  [coefficient name]
    Donator
      Diffusion
      K
      WithInhibition
      K_I
      Carrier
      EquiSorpt
        ParaType
        MassFraction
        Kd-Value
        Exponent
        Regularization
        MaxSorption
        DiscreteData
      NonEquiSorpt
        ParaType
        MassFraction
        Kd-Value
        Exponent
        Regularization
        MaxSorption
        RateParameter
        DiscreteData
      Decay
```

The solute transport provides four different sorption isotherms both for equilibrium and for nonequilibrium sorption, with is controlled by the variable `ParaType`

ParaType	**name**	**equation**
0	linear	$\varphi_{\text{lin}} = K_{\text{d}}\, c$
1	Freundlich	$\varphi_{\text{F}} = K_{\text{d}}\, c^{\texttt{Exponent}}$
2	Langmuir	$\varphi_{\text{L}} = \dfrac{K_{\text{d}}\, c}{1 + \frac{K_{\text{d}}}{\texttt{MaxSorption}}\, c}$
3	Freundlich–Langmuir	$\varphi_{\text{FL}} = \dfrac{K_{\text{d}}\, c^{\text{Exponent}}}{1 + \frac{K_{\text{d}}}{\texttt{MaxSorption}}\, c^{\text{Exponent}}}$
4	generic spline interpolation	

The Freundlich and the Freundlich–Langmuir isotherms can be regularized within the interval [0, `Regularization`]. For `Exponent` < 1 the isotherms are continuously substituted within this interval.

The diffusion-dispersion tensor has the following form in one space dimension:

$$D \;=\; \alpha_l\, q + d\,\theta\;. \tag{3.29}$$

In more space dimensions there exist several matrix representations.

Biodegradation

The interface to the parameters of biodegradation is as follows:

```
Coefficient
└─ [coefficient name]
   ├─ Donator
   │  ├─ Diffusion
   │  ├─ K
   │  ├─ WithInhibition
   │  ├─ K_I
   │  ├─ Carrier
   │  ├─ EquiSorpt
   │  ├─ NonEquiSorpt
   │  └─ Decay
   ├─ Acceptor
   │  ├─ Diffusion
   │  ├─ K
   │  ├─ WithInhibition
   │  ├─ K_I
   │  ├─ Carrier
   │  ├─ EquiSorpt
   │  ├─ NonEquiSorpt
   │  └─ Decay
   ├─ Transport
   ├─ Water
   └─ Biodegradation
      ├─ ParaType
      ├─ MuMax
      ├─ YieldBio
      ├─ DeathCoeff
      ├─ Alpha
      ├─ WithMaxBiomass
      └─ MaxBiomass
```

The restriction by a maximum biomass concentration can be controlled by the flag `WithMaxBiomass` in the `/Biodegradation` subdirectory. In case of no existing bound-

aries the appropriate biomass differential equation simplifies to

$$\partial_t c_{\mathrm{X}} - \frac{Y}{\theta}\,\mu_{\mathrm{D}} + k_{\mathrm{X}}\,c_{\mathrm{X}} \;=\; 0\;. \tag{3.30}$$

You can choose independently an inhibition term of the electron donator and -acceptor by the flag `WithInhibition` in the corresponding `/eDonator` or `/eAcceptor` subdirectory. The model equation for the maximum substrate utilization rate $\mu_{\max}$ will simplify in that way that the appropriate addend in the denominator $c_{\mathrm{D}}^2/K_{\mathrm{I_D}}$ and/or $c_{\mathrm{A}}^2/K_{\mathrm{I_A}}$ are omitted.

If you are using a zeroth order decay constant, verify carefully that this model makes sense in the range of concentration where your simulation proceeds. The model may otherwise result in negative concentrations!

Carrier Facilitation

The effect of mobile and immobile carriers (e. g.dissolved organic carbon, DOC) on solute transport can be included. According to the work of Knabner et al. (1996) the sorption isotherm is replaced by an effective sorption isotherm. Carrier facilitation is switched on and off with the flag `WithCarrier`. The interface to carrier facilitation is as follows:

```
Coefficient
└─ [coefficient name]
   ├─ Donator
   │  ├─ Diffusion
   │  ├─ Functions
   │  ├─ Carrier
   │  │  ├─ WithCarrier
   │  │  ├─ PartitioningCoefficient
   │  │  ├─ Connect2Solute
   │  │  └─ Connect2Sorbate
   │  ├─ EquiSorpt
   │  └─ NonEquiSorpt
   └─ Acceptor
```

To use this feature, you need another problem on your grid, that describes the transport of the carrier. You can use e. g.the problem class solute transport twice. First, you create this problem and its discretization for the carrier. Then, you add the same problem and discretization a second time to the grid for the solute. To establish the connection between the current solute transport for the donator and a previously defined carrier transport follow these steps *(This has not been tested so far!)*:

- Switch `WithCarrier` on,
- define a `PartitioningCoefficient` for the sorption of your solute to the carrier,

- go to the toplevel directory `/Discretization` and select the discretization corresponding to the carrier (e. g.`CarrierDisc`),
- go into the directory `/Discretization/CarrierDisc/Variables` and select the concentration of the carrier,
- execute the command `/Coefficient/[coefficient name]/Donator/Carrier/Connect2\ Solute`,
- enter the directory `/Discretization/CarrierDisc/Variables` and select the sorbed concentration of the carrier,
- execute the command `/Coefficient/[coefficient name]/Donator/Carrier/Connect2\ Sorbate`.

Note
The carrier transport has to be added to the grid before the (carrier facilitated) solute transport.

Water Flow

RICHY1D's interface to define the water flow for your transport problem is as follows:

```
Coefficient
└─ [coefficient name]
   ├─ Donator
   ├─ Acceptor
   ├─ Transport
   └─ Water
      ├─ WithRichards
      ├─ WaterContent
      ├─ Time
      ├─ Flux(Time)
      ├─ Connect2WaterContent
      └─ Connect2Flux
```

The simulation of solute transport depends on a description of water flow, i. e., of the water flux q and the water content θ. There are two possibilities to provide this information for solute transport:

1. Predetermined constant values for water content θ on each subdomain and possibly time dependent values for water flux q. To make use of this method, follow these steps:
   - Switch `WithRichards` off,
   - define `WaterContent` for each subdomain and

- define `Time` and `Flux(Time)` for the whole domain. Values are interpolated piecewise linear at intermediate time points.

2. Simulation of water flow. To make use this method, you need the problem and its discretization, that describes water flow, on your grid. Follow these steps:
   - Switch `WithRichards` on,
   - go to the toplevel directory `/Discretization` and select the discretization corresponding to the water flow (e. g.`WaterDisc`),
   - enter the directory `/Discretization/WaterDisc/Variables` and select the water content,
   - execute the command `/Coefficient/[coefficient name]/Water/Connect2Water\ Content`,
   - go into the directory `/Discretization/WaterDisc/Variables` and select the water flux,
   - execute the command `/Coefficient/[coefficient name]/Water/Connect2Flux`

Note
The simulation of water flow has to be added to the grid before the biodegradation problem.

Serial Decay Reactions

It is possible to couple several transport/biodegradation reactions (cf.3.2 Serial Decay Reactions, p. 69).

Coupling the reactions. The interface to serial decay is as follows (analogous for the acceptor):

```
Coefficient
└─ReactionB
   ├─Donator
   │  ├─Diffusion
   │  ├─K
   │  ├─WithInhibition
   │  ├─K_I
   │  ├─Carrier
   │  ├─EquiSorpt
   │  ├─NonEquiSorpt
   │  └─Decay
   │     ├─0thOrderRate
   │     ├─1stOrderRate
   │     ├─WithFatherSpecies
   │     ├─YieldFactor
   │     ├─Connect2Father
   │     └─Connect2SubComp
   ├─Acceptor
   └─Transport
```

To use this feature, you have to create several transport problems (and their discretization) on your grid, that describe the transport of the species 1 to i. To establish the connection between two reactions (A $\rightarrow$ B) follow these steps:

- Switch `WithFatherSpecies` on,
- define the yield factor of the reaction,
- if the solute variable of the father reaction consists of two (or more) components, define the component by setting `Connect2SubComp`. Look out: '0' identifies the first component!
- Enter the toplevel directory `/Discretization` and select the discretization corresponding to the father reaction (e. g.'ReactionA'),
- enter the directory `/Discretization/ReactionA/Variables/` and select the sink term (e. g.`sink-biodeg` for the problem class biodegradation or `sink-trans` for the problem class transport),
- enter the directory `/Coefficient/ReactionB/Donator/Decay` and execute the command `Connect2Father`.

The links between the reactions for donator and acceptor can be established independently from each other.

Temperature-Dependent Kinetics of Degradation

The influence of the temperature on the kinetics of microbially mediated degradation can be described by multiplying an inhibition term

$$I(T) \;=\; \exp\left(-\kappa(T - T_{\text{opt}})^2\right) \;. \tag{3.31a}$$

Thus the zeroth order term becomes

$$\text{const} \;\longrightarrow\; \text{const}_{\text{opt}}\, I(T) \tag{3.31b}$$

and the zeroth order rate constant can now be interpreted as the zeroth order constant at optimal temperature. Analogously the first order rate becomes

$$k \;\longrightarrow\; k_{\text{opt}}\, I(T) \tag{3.31c}$$

where

T	[Temp]	current temperature
T_{opt}	[Temp]	optimal temperature for biodegradation
κ	[1/Temp^2]	inhibition coefficient

The Interface to this feature is as follows:

```
Coefficient
└─ [transport name]
   ├─ Donator
   │  └─ Decay
   │     ├─ 0thOrderRate
   │     ├─ 0thOrderF(Temp)
   │     ├─ OptimalT40thOrder
   │     ├─ Kappa40thOrder
   │     ├─ 1stOrderRate
   │     ├─ 1stOrderF(Temp)
   │     ├─ OptimalT41stOrder
   │     └─ Kappa41stOrder
   ├─ Acceptor
   │  └─ Decay
   └─ Biodegradation
      ├─ BiodegF(Temp)
      ├─ WithHeatConduction
      ├─ Kappa4Monod
      ├─ MonodTopt
      ├─ Temperature
      │  ├─ Add2Display
      │  └─ TempMeasurement1
      └─ Connect2HeatConduction
```

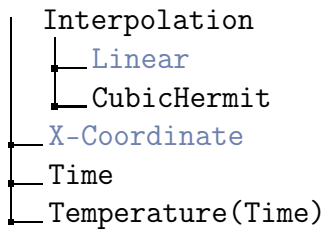
These entries have the following meaning:

0thOrderF(Temp) 1stOrderF(Temp) BiodegF(Temp)	flag to specify whether temperature has an influence on each type of degradation
OptimalT40thOrder OptimalT41stOrder MonodTopt	temperature at which the biodegradation takes place at maximum speed for zeroth order, first order decay and for Monod kinetics
Kappa40thOrder Kappa41stOrder Kappa4Monod	inhibition coeff. for first and zeroth order decay and for Monod kinetics
WithHeatConduction	flag which specifies whether temperature distribution is calculated by heat conduction
Connect2HeatConduction	command to connect the heat conduction to the biodegredation problem
Add2Display	command to add temperature measurements for a new coordinate
TempMeasurement1	directory in which the temperature measurements for one coordinate can be specified

The directory /Decay exists as well for the donator as for the acceptor. Its entries are identical. There are two ways to specify the temperature development:

1. It can be given explicitly as a function of time and space.
2. The biodegradation problem can be coupled with the heat conduction problem.

In order to specify the temperature explicitly, use the command Add2Display in order to add new temperature measurements for each coordinate: The directory /TemperatureMeasurement contains following entries:



The interpolation which has to be specified refers to the evaluation with respect to time. The interpolation between the coordinates is done linearily. It is important to specify the coordinates in an ordered way ($x_1 < x_2 < \ldots < x_n$). Outside of the specified intervals the values will be kept constant.

In order to couple the transport problem with the heat conduction problem, a problem of the type heat conduction must be defined prior to the following steps:

- Go to the toplevel directory /Discretization and select the discretization corresponding to the heat conduction problem (e. g.HcDisc),

- enter the directory `/Discretization/HcDisc/Variables` and select `temperature`,
- execute the command `/Coefficient/[transport name]/Biodegradation/Connect2Heat\Conduction`,
- select the flag `/Coefficient/[transport name]/Biodegradation/WithHeatConduction`.

It is possible to determine the parameters κ and T_{opt} by inverse modelling as well as to model temperature dependent serial decay reactions. Though it is necessary to specify the explicit temporal evolution of the temperature for each species as well as the coupling with the heat conduction must be done for each species.

Usage

To create a set of coefficient functions for biodegradation follow these steps:

- Go to the directory `/Library/Coefficient`,
- execute `Biodeg` and define a name, under which the corresponding parameter set appears in the toplevel directory `/Coefficient`.

Discretization

The solute transport is discretized with a conforming finite element method, mass lumping and a backward Euler scheme. To create the corresponding discretization follow these steps:

- Go to the directory `/Library/Discretization`,
- execute `FEM4Biodeg` and define a name, under which the discretization appears in the toplevel directory `/Discretization`.

Now you have access (e. g.for selection) to the variables of this discretization in the toplevel directory `/Discretization`.

```
Discretization
├── Stabilization4Transport
│   └── ArtificialDiffusion
└── [name]
    └── Variables
        ├── solute
        ├── sorbate(neq)
        ├── sorbate(eq)
        ├── sink-biodeg
        ├── solute_sens
        ├── sorbate(neq)_sens
        └── sorbate(eq)_sens
```

The entry `ArtificialDiffusion` is the parameter of the stabilization (cf.A.1 Stabilization, p. 130). The entries in the directory `Variables` correspond to that, explained in the notations table above. The variable `sink-biodeg` refers to the sink term by Monod kinetics of your biodegradation problem. This sink term may enter as a source term in a subsequent transport problem. Note that it does not contain the zeroth order decay terms!

A variable `[name]_sens` contains the sensitivities of the variable `[name]` with respect to the parameters for biodegradation. Starting with donator, then acceptor and finally biomass, the arrangement of the parameters is as follows:

number	0	1	2	3	4	5	6	7	8	9	10
parameter	$\mu_{\max}$	Y	k_{X}	$\alpha_{\mathrm{A/D}}$	K_{D}	$K_{\mathrm{I_D}}$	K_{A}	$K_{\mathrm{I_A}}$	κ	T_{opt}	$c_{\mathrm{X_{max}}}$

Example
The subcomponent $15 = 1 \times 11 + 4$ of the variable `solute_sens` contains the sensitivity values of the solute mass concentration of the acceptor.

Instructions for Usage

The problem class biodegradation handles a scalar initial boundary value problem for the solute concentration together with an initial value problem for the (nonequilibrium-)sorbed concentration. Therefore you have to add

- an initial boundary value problem consisting of
  - a domain,
  - a scalar boundary condition for donator and acceptor (`Type Dirichlet`, `Neumann` or `Flux`) for all three species an initial value with two vectors (first vector for the solute and second vector for the (nonequilibrium-)sorbed concentration). If flux conditions (q_{in}, c_{in}) are specified, the given values are the inflow concentrations, which will be multiplied by the given water flux, taken from the coefficients menu, or the Richards equation, if you have coupled the problems (so you need not recalculate the boundary conditions when the water flux changes),
  - the set of coefficient functions `Biodeg` and
- the discretization `FEM4Biodeg` to the grid.

Note
Even if the biomass is immobile, you have to specify homogenous Neumann boundary conditions for it! This is due to technical reasons and must not be changed.

The result of this discretization is a piecewise linear approximation of the solute- and the (nonequilibrium-)sorbed concentration. Any further variables (e. g.(equilibrium-)sorbed concentration) can be derived from these variables.

Identification

The problem class biodegradation allows the identification of the parameters of biodegradation. For the identification you can use the `OLS` error functional (output least squares), the `OLS_TW` error functional (output least squares with weighting by time differences) or the `OLS_SW` error functional (output least squares with adaptive weighting by sensitivity values).

In the case of formfree identification (parametrization of the growth rates in the Monod model with splines) the directory `/Coefficient/[coefficient name]/Functions/GrowthRateDonator/FittingData` contains the additional entries:

`MuMax-BdLower`	[`1/Time`]	lower boundaries for maximum decay rate
`MuMax-BdUpper`	[`1/Time`]	upper boundaries for maximum decay rate
`YieldBio-BdLower`	[−]	lower boundaries for yield coefficient for biomass
`YieldBio-BdUpper`	[−]	upper boundaries for yield coefficient for biomass
`DeathCoeff-BdLower`	[`1/Time`]	lower boundaries for decay rate for biomass
`DeathCoeff-BdUpper`	[`1/Time`]	upper boundaries for decay rate for biomass

and the directory `/Coefficient/[coefficient name]/Functions/GrowthRateAcceptor/FittingData` contains the additional entries:

`Alpha-BdLower`	[−]	lower boundaries for yield coefficient for acceptor
`Alpha-BdUpper`	[−]	upper boundaries for yield coefficient for acceptor

3.7 Reactive Multicomponent Transport

Table of Contents

Notation

Parameters for solute transport of species

t	[Time]	time
c_i	[Mass/Length^3]	mass concentration of ith solute or immobile species
γ_i	[−]	activity coefficient for ith species
c_{X_ℓ}	[Mass/Length^3]	mass concentration of ℓth biomass species
s_i	[Mass/Mass]	mass concentration of sorbate at nonequ. sorption sites
$\boldsymbol{q}$	[Length/Time]	water flux
θ	[−]	volumetric water content (coefficient function)
ρ_{b}	[Mass/Length^3]	bulk density
f_ϕ, f_φ	[−]	mass fraction of sorption sites
$\phi(c)$	[−]	equilibrium sorption isotherm
$\varphi(c)$	[−]	nonequilibrium sorption isotherm
r_i	[1/Time]	rate parameter for nonequilibrium sorption
k_i	[1/Time]	first order decay rates
$\mathbf{D}_i$	[Length^2/Time]	diffusion-dispersion
α_{l}	[Length]	longitudinal dispersivity
d_i	[Length^2/Time]	molecular diffusion
const_i	[Mass/(Length^3 Time)]	zeroth order constant (source or loss)
R_j	[Mass/(Length^3 Time)]	rate of jth reaction
ν_{ij}	[−]	stoichiometric factor of ith species in jth reaction

Model Equation

The module for reactive multicomponent transport allows you to simulate the spreading of multiple species simultaneously, which may be coupled by kinetic reactions in a flexible and rather arbitrary way. There exist four basic types of species: mobile, immobile, mineral and microbial species. Microbial species are also immobile, but are additionally taken into account for the calculation of the maximum biomass concentration, that may limit the biomass growth (see Biodegredation Reaction, p. 121). Mineral species are immobile and their volume is part of the solid phase, and thus you can include the influence of varying mineral concentrations on the porosity, if you connect the subproblems (see p. 116)

The transport model for the mobile species includes diffusion-dispersion, convection, and reaction rates of different kinds. Furthermore, we also kept the possibility of including an equilibrium and a nonequilibrium sorption isotherm.

The general form of the equation reads

$$\partial_t(\theta\, c_i) - \boldsymbol{\nabla}\cdot(\mathbf{D}_i\nabla c_i - \boldsymbol{q}\, c_i) + \rho_{\mathrm{b}}\,\partial_t\left(f_{\phi_i}\,\phi_i(c_i) + f_{\varphi_i}\,s_i(c_i)\right) \;=\; \theta\sum_{j=1}^{N_R}\nu_{ij}\,R_j\;, \tag{3.32a}$$

$$\partial_t s_i \;=\; r_i(\varphi_i(c_i) - s_i)\;.$$

The specific reaction rates R_j for zeroth and first order decay, general stoichiometric kinetic reactions, and general microbial reactions will be depicted in detail in the following. Mineral, microbial or other immobile species are not transported, but transformed by reactions and thus equation (3.32a) simplifies for those species to:

$$\partial_t c_i \;=\; \sum_{j=1}^{N_R}\nu_{ij}\,R_j\;. \tag{3.32b}$$

Zeroth and First Order Decay

For zeroth and first order decay of the ith species, we get the rate expressions

$$R_i \;=\; -k_i\,c_i - \mathrm{const}_i\;. \tag{3.33}$$

Note that here, the (additional) stoichiometric coefficients will be unity. Decay chains (with father and product concentrations) can be included in the framework of the stoichiometric reactions. Note that these rates are multiplied by the water content θ for the mobile species.

Biodegradation Reactions

The biodegradation reaction may involve arbritrary species, but one reaction is catalyzed by exactly one microbial species, which is denoted here by the index X_ℓ. The model combines Monod growth terms and the inhibition terms of an arbitrary number and combination of species (a substance may as well be necessary and inhibiting in different concentration ranges), see also Widdowson et al. (1988). The index sets $\mathcal{M}_j$ and $\mathcal{I}_j$ refer to those species involved in the jth reaction, that are necessary for the microbial (Monod) growth and that inhibit that reaction, respectively:

$$R_j \;=\; -\mu_{\max_j}\,c_{X_\ell}\prod_{i\in\mathcal{M}_j\subset\{1,\dots,N_S\}}\left(\frac{c_i}{K_{\mathrm{M}_i}+c_i}\right)\prod_{i\in\mathcal{I}_j\subset\{1,\dots,N_S\}}\frac{K_{\mathrm{I}_i}}{K_{\mathrm{I}_i}+c_i}\;. \tag{3.34}$$

This reaction term can slightly be modified in the equations of the microbial species, to account for growth limitations of the microorganisms by pore space restrictions, production of toxic metabolites or lack of nutrients (cf.(Schirmer et al. 2000)). This restriction term sums up the concentrations of all the present microbial species. The resulting equation is

$$R_j \;=\; \left(1 - \frac{\sum_i c_{X_i}}{c_{X_{\max}}}\right)\mu_{\max_j}c_{X_\ell}\prod_{i\in\mathcal{M}_j\subset\{1,\dots,N_S\}}\left(\frac{c_i}{K_{\mathrm{M}_i}+c_i}\right)\prod_{i\in\mathcal{I}_j\subset\{1,\dots,N_S\}}\frac{K_{\mathrm{I}_i}}{K_{\mathrm{I}_i}+c_i}\;. \tag{3.35}$$

You may also switch off this restriction term (see Biodegredation Reaction, p. 121).

Stoichiometric Kinetic Reactions

The general reaction rate of the rth stoichiometric kinetic reaction is given as the net rate between forward and backward reaction rates, which can be formulated by the rate constants of forward and backward reaction (e. g.(Bethke 1996)) according to the mass action law of thermodynamics:

$$R_j = \left(k_j^{\mathrm{f}} \prod_{\{i|\nu_{ij}>0\}} c_i^{\nu_{ij}} - k_j^{\mathrm{b}} \prod_{\{i|\nu_{ij}<0\}} c_i^{-\nu_{ij}} \right) . \tag{3.36}$$

The first product refers to the reactants, the second to the product species, where by convention their stoichiometric coefficients are negative. Note that we do not make activity corrections for the concentrations.

Mineral Stoichiometric Kinetic Reactions

The general reaction rate of the jth surface-controlled mineral stoichiometric kinetic reaction is given as the net rate between forward and backward reaction rates, which can be formulated by the rate constants of forward and backward reaction:

$$R_j = \tilde{s}_m \left(k_j^{\mathrm{f}} \prod_{\{m\neq i|\nu_{ij}>0\}} c_i^{\nu_{ij}} - k_j^{\mathrm{b}} \prod_{\{m\neq i|\nu_{ij}<0\}} c_i^{-\nu_{ij}} \right) , \tag{3.37}$$

where $\tilde{s}_m = \tilde{s}_m^0 (c_m / c_m^0)^{\sigma_m}$ is the surface area of the mineral species. $\tilde{s}_m^0$ denotes the initial surface area of the mineral species and c_m^0 is the initial concentration of the mineral species. Please note that there is exactly one mineral species per mineral reaction. This species has index m (in equation 3.37). The first product refers to the reactants, the second to the product species, where by convention their stoichiometric coefficients are negative. The shape factor $\sigma_m \in (0, 1)$ may be chosen arbitrarily, it is set to 2/3 by default. For more details see (Frank 2008).
Surface-controlled mineral stoichiometric kinetic reactions may influence porosity as the mineral is a substantial part of the solid matrix, and thus changes in the mineral concentration also change porosity.

If you want to include this effect in your simulation, you must connect the reactive transport problem with the water flow problem (i.e. the Richards Equation). The variation of the porosity is described in Section 3.4. Therefore use the following instructions:

1. Connect multispecies problem to Richards Problem.
   This provides the current water flux and water content values of an unsaturated flow problem and uses them in the Multicomponent transport problem.
   - Go to `Discretization` and select directory `HMFEM4Richards`. Then enter this directory and select `theta`.

- Go to `Coefficient` and select directory `[Coefficient name]`. Then enter directory `/Coefficient/[Coefficient name]/Water` and execute `Connect2WaterContent`.
- Go to `Discretization` and select directory `HMFEM4Richards`. Then enter this directory and select `pressure`.
- Go to `Coefficient` and select directory `[Coefficient name]`. Then enter directory `/Coefficient/[Coefficient name]/Water` and execute `Connect2Pressure`.
- Go to `Discretization` and select directory `HMFEM4Richards`. Then enter this directory and select `flux`.
- Go to `Coefficient` and select directory `[Coefficient name]`. Then enter directory `/Coefficient/[Coefficient name]/Water` and execute `Connect2Flux`.

2. Connect Richards Problem to multispecies problem.
   This takes into account the variations of the mineral concentrations and updates the resulting porosity changes in the flow problem.
   - Go to `Discretization` and select directory `FEM4Multispec`. Then enter directory `/Discretization/FEM4Multispec/Variables` and select `porosity`.
   - Enter directory `/Coefficient/Richards/T_sat_approx`. Then execute `Connect2Porosity` and switch on `WithPorosity`.

Coefficient Functions

Some coefficient functions of a reactive multicomponent transport problem are identical for all species. This holds true for the water flow, transport parameters of the porous medium like the dispersion length α_l, the bulk density ρ_b, and the temperature (if needed). Others are specific for every species.

Water Flow and Matrix Menu

Mobile species are transported in the liquid phase. The menu's `Matrix` and `Water` apply to all mobile species. RICHY1D's interface to define the water flow for your transport problem is as follows:

```
Coefficient
└─ [coefficient name]
   ├─ Matrix
   ├─ Water
   │  ├─ WithRichards
   │  ├─ WaterContent
   │  ├─ Time
   │  ├─ Flux(Time)
   │  ├─ Exchangeterm
   │  ├─ Connect2WaterContent
   │  ├─ Connect2Flux
   │  └─ Connect2Pressure
   ├─ Temperature
   ├─ Species
   └─ Reactions
```

The simulation of solute transport needs the specification of the water content θ and the water flux q (see also model equation (3.32a)). There are two possibilities to provide this information for solute transport:

1. Predetermined constant values for water content θ on each subdomain and possibly time dependent values for water flux q. To make use of this method, follow these steps:
   - Switch `WithRichards` off,
   - define `WaterContent` for each subdomain and
   - define `Time` and `Flux(Time)` for the whole domain. Values are interpolated piecewise linear at intermediate time points.
2. Simulation of water flow. To make use this method, you need the problem and its discretization, that describes water flow, on your grid. Follow these steps:
   - Switch `WithRichards` on,
   - go to the toplevel directory `/Discretization` and select the discretization corresponding to the water flow (e. g.`WaterDisc`),
   - enter the directory `/Discretization/WaterDisc/Variables` and select the water content,
   - execute the command `/Coefficient/[coefficient name]/Water/Connect2Water\Content`,
   - enter the directory `/Discretization/WaterDisc/Variables` and select the water flux,
   - execute the command `/Coefficient/[coefficient name]/Water/Connect2Flux`.

Note
The simulation of water flow has to be added to the grid before the multispecies transport problem.

In the Matrix menu, properties of the porous medium (the solid matrix) like the dispersion length α_l and the bulk density ρ_b are given:

```
Coefficient
└─ [coefficient name]
   ├─ Matrix
   │  ├─ Dispersion
   │  └─ BulkDensity
   ├─ Water
   ├─ Temperature
   ├─ Species
   └─ Reactions
```

Species Dependent Coefficient Functions

If you have added a species by the `AddSpecies` command, you will be prompted for the name of the species (e. g.'Benzene'), and a new subdirectory is created including the running number of the species to identify it, and its name (note—the running number starts with 0). The menu structure could look like:

```
Coefficient
└─ [coefficient name]
   ├─ Matrix
   ├─ Water
   ├─ Temperature
   ├─ Species
   │  ├─ AddSpecies
   │  ├─ Species00:Benzene
   │  └─ Species01:Xylene
   └─ Reactions
```

The species subdirectory has the following entries:

```
Species00:Benzene
├─Type
│  ├─mobile
│  ├─immobile
│  ├─microbial
│  └─mineral
├─Diffusion
├─Charge
├─MolarVolume
├─Sigma
├─Carrier
├─EquiSorpt
│  ├─WithIsotherm
│  ├─ParaType
│  ├─MassFraction
│  ├─Kd-Value
│  ├─Exponent
│  ├─Regularization
│  └─MaxSorption
├─NonEquiSorpt
│  ├─WithIsotherm
│  ├─ParaType
│  ├─MassFraction
│  ├─Kd-Value
│  ├─Exponent
│  ├─Regularization
│  ├─MaxSorption
│  ├─RateParameter
│  └─DiscreteData
├─0thOrderRate
├─0thOrderF(Temp)
├─OptimalT40thOrder
├─Kappa40thOrder
├─1stOrderRate
├─1stOrderF(Temp)
├─OptimalT41stOrder
├─Kappa41stOrder
├─RichyNumInBlock
└─a
```

First, select the type of the species. For a mobile species, you should give its molecular diffusion constant d_i. The diffusion-dispersion tensor has the following form in one space dimension:

$$D_i \;=\; \alpha_l \, q + d_i \, \theta \;, \tag{3.38}$$

where the molecular diffusion d_i depends on the concentration of the ith species. In more space dimensions there exist several matrix representations.

If you want to describe sorption of mobile species by isotherms, you can define them in the equilibrium sorption menu and/or the nonequilibrium sorption menu. You can switch the isotherms on and off with the flags `WithIsotherm`. For immobile, microbial and mineral species, these settings will be ignored. The sorption menus are explained in the solute transport section. Note that we currently allow only one nonequilibrium sorption isotherm per species (no multiple kinetic sites). Furthermore you can specify decay rates of zeroth and first order, which may be temperature dependent. If they depend on the temperature (switch the flags `1stOrderF(Temp)` or `0thOrderF(Temp)` on), you have to define the optimal temperature and the inhibition constant `Kappa` (see Temperature-Dependent Kinetics of Degradation, p. 123). If you are using a zeroth order decay constant, verify carefully that this model makes sense in the range of concentration where your simulation proceeds. The model may otherwise result in negative concentrations!

Immobile and microbial species are only transformed by decay and reactions.

Biodegredation Reaction

When you add a biodegradation reaction, the following interface for the parameters will be available:

```
Coefficient
└─ [coefficient name]
   ├─ Transport
   ├─ Water
   ├─ Temperature
   ├─ Species
   ├─ Reactions
   │  ├─ AddKineticReaction
   │  ├─ AddBiodegradationReaction
   │  └─ Reaction01:Biodegradation
   │     ├─ Mumax
   │     ├─ SpeciesList
   │     ├─ NrOfBioSpecies
   │     ├─ StoichCoeff
   │     ├─ Monod
   │     ├─ Haldane
   │     ├─ WithMaxBiomass
   │     ├─ MaxBiomass
   │     ├─ BiodegF(Temp)
   │     ├─ Kappa
   │     └─ OptimalTemp
```

`Mumax` is the maximum microbial growth rate (in each subdomain), the `SpeciesList` contains the indices of the species that take part in this reaction, also the possibly inhibiting species. The order is not of importance, but note that the Monod coefficients etc. refer to the order of the indices given in `SpeciesList`, i. e., if the first entry in `SpeciesList` is 4, the first Monod coefficient refers to species no. 4. Note that the entry '−1' means that the end of the list is reached. No further values will be read.

The index of the microbial species that catalyses this reaction, is additionally given in `NrOfBioSpecies`. The list of the corresponding stoichiometric coefficients is given in `StoichCoeff`, again note that `StoichCoeff[i]` refers to species number `SpeciesList[i]`. Purely inhibitory substances have, of course, stoichiometric coefficient 0, because they do not grow or decay through the reaction.

Monod and Haldane are matrices of dimension `MaxSpecies` × `MaxSubdomains`, so for each line (i. e., each species, corresponding again to the order given in `SpeciesList`), you specify the species coefficients in the subdomains. In case a species is not inhibitory, this corresponds to an extremely large inhibition concentration (say, e. g., 1×10^9), then the inhibition term equals approximately 1.

The restriction by a total maximum biomass concentration can be controlled by the flag `WithMaxBiomass`. The other parameters are as listed in the notations table above.

Parameters for jth biodegradation reaction

ν_{ij}	[−]	stoichiometric coefficient for ith species in jth reaction
$\mu_{\max_j}$	[1/Time]	maximum substrate utilization rate
$c_{\mathrm{X}_{\max}}$	[Mass/Length^3]	maximum total biomass concentration
K_{M_i}	[Mass/Length^3]	half maximum rate parameter (Monod constant)
K_{I_i}	[Mass/Length^3]	inhibition concentration

For the microbial species, the stoichiometric coefficient is equivalent with the yield coefficient for the microbial species in this reaction. For the calculation of the total biomass concentration, all microbial species concentrations of the multicomponent problem are summed up. Temperature dependency of the reaction is described in the section Temperature-Dependent Kinetics of Degradation, p. 123.

Chemical Kinetic Reactions

When you add a reaction according to mass action kinetics, the following interface for the parameters will be available:

```
Coefficient
  [coefficient name]
    Transport
    Water
    Temperature
    Species
    Reactions
      AddKineticReaction
      AddBiodegradationReaction
      Reaction01:KineticRateLaw
        SpeciesList
        StoichCoeff
        ForwardRateConst
        BackwardRateConst
```

The `SpeciesList` contains the identification number of the participating species in that reaction (e. g.`Species03:NaCl` is number 3), as they are given in the `Species` menu. Note that the entry '−1' means that the end of the list is reached. No further values will be read. The stoichiometric coefficients of the species in that reaction are given in the array `StoichCoeff` in the same order as the species in `SpeciesList` are given. `ForwardRateConst` and `BackwardRateConst` have to be specified for each subdomain.

Parameters for kinetic rate law

k^{f}	[1/Time]	forward rate constant
k^{b}	[1/Time]	backward rate constant
ν_{ij}	[−]	stoichimetric coefficient of species i in reaction j

Carrier Facilitation

The effect of mobile and immobile carriers (e. g.dissolved organic carbon, DOC) on the transport of mobile species can be included in the simulation. This feature is explained in the solute transport documentation (cf.3.2 Solute Transport, p. 62).

Note

The carrier transport problem has to be added to the grid before the (carrier facilitated) solute transport problem. This must be two seperate problems, it is not possible to choose the carrier among the species of the same multicomponent problem.

Temperature-Dependent Kinetics of Degradation

The influence of the temperature on the kinetics of microbially mediated degradation can be described by multiplying an inhibition term to the standard degradation rate.

This rate may be of zeroth or first order, or the rate of a biodegradation reaction. The inhibition term reads

$$I(T) \;=\; \exp\left(-\kappa(T - T_{\text{opt}})^2\right) \;, \tag{3.39a}$$

where

T	[Temp]	current temperature
T_{opt}	[Temp]	optimal temperature for biodegradation
κ	[1/Temp^2]	inhibition coefficient

The degradation process is inhibited (i. e., the optimal rate decreased) whenever the current temperature T deviates of the optimal temperature T_{opt}. The rates you specify now are interpreted as the optimal rates at T_{opt}. Thus the zeroth order term becomes

$$\text{const} \;\longrightarrow\; \text{const}_{\text{opt}}\, I(T) \;. \tag{3.39b}$$

Analogously the first order decay rate becomes

$$k \;\longrightarrow\; k_{\text{opt}}\, I(T) \;. \tag{3.39c}$$

You also may include this effect for the biodegradation reaction (cf.Biodegredation Reaction, p. 121). The relevant menu items for this feature are the following:

```
Coefficient
└─ [multispecies transport name]
   ├─ Temperature
   │  ├─ Connect2HeatConduction
   │  ├─ Add2Display
   │  └─ WithHeatConduction
   ├─ Species
   │  └─ SpeciesXX:Carbonsource
   │     └─ Decay
   │        ├─ 0thOrderRate
   │        ├─ 0thOrderF(Temp)
   │        ├─ OptimalT40thOrder
   │        ├─ Kappa40thOrder
   │        ├─ 1stOrderRate
   │        ├─ 1stOrderF(Temp)
   │        ├─ OptimalT41stOrder
   │        └─ Kappa41stOrder
   └─ Reactions
      └─ ReactionXX:Biodegradation
         ├─ BiodegF(Temp)
         ├─ Kappa
         └─ OptimalTemp
```

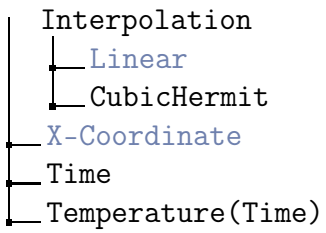
These entries have the following meaning:

0thOrderF(Temp) 1stOrderF(Temp) BiodegF(Temp)	flag to specify whether temperature has an influence on each type of degradation
OptimalT40thOrder OptimalT41stOrder OptimalTemp	temperature at which the biodegradation takes place at maximum speed for zeroth order decay, first order decay and for the Monod type degradation reaction
Kappa40thOrder Kappa41stOrder Kappa	inhibition coeff. for first and zeroth order decay and for Monod kinetics
WithHeatConduction	flag which specifies whether temperature distribution is calculated by heat conduction
Connect2HeatConduction	command to connect the heat conduction to the biodegredation proble
Add2Display	command to add temperature measurements for a new coordinate
TempMeasurement1	directory in which the temperature measurements for one coordinate can be specified

The decay rates of zeroth and first order exist for every species. The specified temperature profile is relevant for every temperature dependent rate, but it is not necessary that every rate is temperature dependent. There are two ways to specify the temperature development:

1. It can be given explicitly as a function of time and space.
2. The transport problem can be coupled with a heat conduction problem.

In order to give the temperature explicitly, use the command `Add2Display` in order to add a new temperature measurement for each coordinate: The directory `/TemperatureMeasurement` contains following entries:



The interpolation which has to be specified refers to the evaluation with respect to time. The interpolation between the coordinates is done linearily. It is important to specify the coordinates in an ordered way ($x_1 < x_2 < \ldots < x_n$). Outside of the specified intervals the values will be kept constant.

In order to couple the transport problem with the heat conduction problem, a problem of the type heat conduction must be defined prior to the following steps:

- Go to the toplevel directory `/Discretization` and select the discretization corresponding to the heat conduction problem (e. g.`HcDisc`),
- enter the directory `/Discretization/HcDisc/Variables` and select temperature,
- execute the command `/Coefficient/MultispecTransport/Temperature/Connect2Heat\Conduction`,
- select the flag `/Coefficient/MultispecTransport/Temperature/WithHeatConduction`.

It is possible to determine the parameters κ and T_{opt} by inverse modelling.

Activity Correction

Solutions with low salinity can be regarded as ideal solutions - electrostatic interaction betweeen ions can be neglected. For higher ionic concentrations those have to be taken into account. Consequently the **activity** as "effective concentration" for ionic species is smaller than the actual concentration due to the decrease of the chemical potential in consequence of the interaction between opposed-charged ions. Within the reaction rates (3.36) and (3.37) in the model equation (3.32a) the concentrations c_i have to be substituted by the activities for each dissolved species:

$$c_i \longleftarrow \gamma_i c_i, \qquad i \in \mathcal{S}^{\text{dissolved}} \qquad [\texttt{Mass/Length}^3\ \texttt{water}],$$

where γ_i $[-]$ denotes the **activity coefficient** for the ith dissolved, ionic species, which depends on the concentrations of all dissolved ionic species.

Estimation of the Activity Coefficient

A_γ		constant, $A_\gamma \approx 0.511$ for water at 25°C and 1 atm
B_γ		constant, $B_\gamma \approx 0.33$ for water at 25°C and 1 atm
γ_i	$[-]$	activity coefficient for the ith dissolved species
I	$[\texttt{Mass/Length}^3\ \texttt{water}]$	ionic strength
z_i		ionic charge of the ith dissolved species

The activity coefficient varies with the amount of cat- and anions in a solution. The **ionic strength** I of an electrolyte solution is given by (W. Stumm 1996):

$$I = \frac{1}{2}\sum_i c_i z_i^2 \qquad [\texttt{Mass/Length}^3\ \texttt{water}]$$

where z_i denotes the **ionic charge** of the ith ionic species (Menu entry `Charge`). For different ranges of values of I, the activity coefficient γ_i can be estimated via the model equations given in the table below. The parameters A_γ and B_γ depend in general on temperature and density of the solution.

Remark - Richy1D - Dimensions. The values of the ionic strength I must be converted to the explicit dimension [`mol/dm`^{3}] to fit the definition range of the approximation models of the activity coefficients (cf. table below). This is an open problem since the activity correction thus abolishes the option to choose arbitrary dimensions for simulations with RICHY1D.

Approximation	Activity Coefficient	Approx. Applicability
DEBYE-HÜCKEL	$\log_{10}\gamma_i = -A_\gamma z_i^2 I^{1/2}$	$I < 10^{-2.3}$ mol/dm^3
GÜNTELBERG	$\log_{10}\gamma_i = -A_\gamma z_i^2 \frac{I^{1/2}}{1+I^{1/2}}$	$I < 10^{-1}$ mol/dm^3
DAVIES	$\log_{10}\gamma_i = -A_\gamma z_i^2 \left(\frac{I^{1/2}}{1+I^{1/2}} - 0.24I\right)$	$I < 0.5$ mol/dm^3

Usage

To create a set of coefficient functions for reactive multispecies transport problems follow these steps:

- Go to the directory `/Library/Coefficient`,
- execute `Multispecies` and define a name (e. g.'MS'), under which the corresponding parameter set appears in the toplevel directory `/Coefficient`.

Adding species. Note that before you can add species in the coefficients menu, you must have defined the discretization of your multispecies problem. After that, you may define the coefficients corresponding to every single species of your problem:

- Go to the directory `/Coeffient/MS/Species` and execute the command `AddSpecies` for every species you want to include in your simulation. You are asked for a name (e. g.'CaCO3'). A new entry in the species menu is created of the type `SpeciesXX:CaCO3` with a number `XX`, that is generated automatically. This number is the so called `NumInBlock` of the concentration vector, which may be needed for plotting, identifying the species in the specieslists of the reactions, or for connecting different problems. Note that it starts with 00.
- Go to the directory `/Coeffient/MS/Species/SpeciesXX:CaCO3` and define the coefficients for that species.

Discretization

The solute transport is discretized with a conforming finite element method, mass lumping and a backward Euler scheme. The reactive multicomponent/transport problem is accurately solved fully coupled without operator splitting (and its errors) by Newton's method. To create the corresponding discretization follow these steps:

- Go to the directory `/Library/Discretization`,

- execute `FEM4Multispec` and define a name, under which the discretization appears in the toplevel directory `/Discretization`.

Now you have access (e. g.for selection) to the variables of this discretization in the toplevel directory `/Discretization`.

```
Discretization
├─Stabilization4Transport
│  └─ArtificialDiffusion
└─[name]
   └─Variables
      ├─concentration
      ├─sorbate(neq)
      └─sorbate(eq)
```

The entry `ArtificialDiffusion` is the parameter of the stabilization (cf.A.1 Stabilization, p. 130). The entries in the directory `Variables` correspond to that, explained in the notations table above. All the variables in a multispecies problem must be understood as vectors that contain all the components (i. e., the single species, e. g.the concentrations). You have access to a species concentration (e. g.for plotting) by specifying its `NumInBlock`. This number corresponds to the species number in the species menu (attention: this number has nothing to do with the name you specify for the species).

Example

The menu entry `Species04:Solute138` tells you, that the relevant `NumInBlock` is 4.

Instructions for Usage

The problem class reactive multispecies transport handles a coupled system of initial boundary value problems for the solute concentrations together with initial value problems for the (nonequilibrium-)sorbed concentrations of the mobile species, and initial value problems for the species of immobile or microbial type. Therefore you have to add

- an initial boundary value problem consisting of
  - a domain,
  - a vector boundary condition with components for every species (even the immobile and microbial) of type Dirichlet, Neumann or flux. If flux conditions (q_{in}, c_{in}) are specified, the given values are the inflow concentrations, which will be multiplied by the given water flux, taken from the coefficients menu, or the Richards equation, if you have coupled the problems (so you need not recalculate the boundary conditions when the water flux changes),

  - a **vector initial condition** with components for every species for specifying the initial concentrations. This concentration corresponds to the solute concentration for species of the type `mobile`, and to sorbed concentrations for species of the type `immobile`, `mineral`, or `microbial`. As you still have the possibility to include sorption kinetics in your simulation by sorption isotherms, you also have to specify a **vector initial condition** with components for the nonequilibrium sorbed concentration for every species, and a **vector initial condition** to define the initial surface area for mineral species.
    You always need these three types of initial values (concentration, non equilibrium sorbed concentration, surface area) for every species you add - no matter which type it is. The entries in the vector will only be read for the species with noneq. sorption, or the minerals with initial surface area, respectively. However the number of components in each vector must correspond to the total number of species (see also note below).
  - the set of coefficient functions and

- the discretization `FEM4Multispec` to the grid.

Note

Even if the microbial and other immobile species do not need boundary conditions, you have to add a component for them in the vector of boundary conditions. You do not need to specify any values, as they are ignored, but there have to exist as many components as species. This is only due to technical reasons and may be changed in a later version. The same holds true for the initial conditions of the nonequilibrium sorbate. Even if not needed for every species, you have to add a component for every species. Of course the simulation of the sorbate part makes only sense for mobile species, but due to technical reasons, the vectors must all have the same number of components (may be changed in a later version). So you just have to specify initial values for the sorbate concentrations of mobile species, that have a nonequilibrium sorption isotherm, but the other components must at least exist (although they certainly have no influence on the simulation).

The result of this discretization is a piecewise linear approximation of the solute concentration for mobile species, sorbed concentration for immobile and microbial species. For mobile species, additionally a piecewise linear approximation of the (nonequilibrium-)sorbed concentration may be given. Any further variables (e. g.(equilibrium-)sorbed concentration) can be derived from these variables.

We have included a sample script-file `def4multispec.scr`, which you may modify for your own purposes.

Appendix A

Annotations and Support

A.1 Stabilization

If the problem is convection dominant, i. e., the convective forces are much bigger than the diffusive, the numerical solution can have instabilities. The solution may be oscillating. To reduce or avoid this effect you can use special numerical methods for solving the problem. In RICHY1D the SUPG (Streamline Upwind Petrov–Galerkin method) as described in the book of Knabner and Angermann (2003) is implemented. It has the parameter `ArtificalDiffusion` which can be set in the menu `/Discretization/Stabilization4name`, where 'name' is the one of the problem class. If you set the parameter to zero the SUPG-Method is switched off and the problem will be solved with the standard finite element method.

How to choose the parameter The bigger the parameter the more the oscillations are damped. But if you choose the parameter too big the numerical solution becomes wrong. For example negative values of concentrations can occur or at boundary layers the solution is flattened. So a good value of the paramter may be about 20 in the case of biodegradation (see 3.6 Biodegradation, p. 99) and reactive multicomponent transport (see 3.7 Reactive Multicomponent Transport, p. 113) or about 1 in the case of solute transport (see 3.2 Solute Transport, p. 62). To avoid the negative concentrations which occur at strong stabilization you can increase the number of elements in the domain (see 2.3.1 Domain, p. 10).

A.2 Simulation Examples

A.2.1 Simulation of Heat Conduction

This example for the solution of a heat conduction problem is a step-by-step introduction to set up the problem specific simulation scenario. The commands and definitions given herein can also be found in the corresponding scriptfile `def4heat.scr` in `./appl/scripts/`. The content of this course corresponds to the work scheme (cf.1.2 Work Scheme, p. 5).

Statement of the Problem

This example provides a solution to the one-dimensional heat conduction equation. We use a homogeneous domain with constant coefficients, i. e., thermal conductivity and heat capacity. Details about the model equation are given in section 3.1 Heat Conduction, p. 57.

Domain

We create a domain named 'Oven', represented by the interval $[0, 1]$, consisting of one subdomain (where the coefficients do not vary) and choose a spatial grid of 100 elements:

- go to the directory `/Library/Domain`,
- allocate a domain with the command `1DDomain` and enter the name, e. g., 'Oven',
- go to the directory `/Domain/Oven`,
- set the variables
  - `LeftBoundary` to 0.0 *(default)*,
  - `RightBoundary` to 1.0,
  - `NumOfSubdomains` to 1 *(default)*,
  - *`InnerBoundaries` will be ignored because of the coherent domain*,
  - `ElemsPerSubDom` to 100 *(integer array, but due to the single subdomain just the first entry will be used)*.

Boundary Condition

We now have to assign the type and values for the left and the right boundary conditions:

- go to the directory `/Library/BoundCond`,
- define a scalar boundary condition with the command `ScalarBC` and name it, e. g., 'FireFromRight',
- enter the directory `/BoundCond/FireFromRight/Left`,
- select *(by right mouse button)* `Linear` in the corresponding subdirectory `Interpolation` *(default)*, this defines the type of interpolation between the values that you specify,
- select `Dirichlet` within the subdirectory `Type` *(default)*,
- set the variables

  - `Time` to (0.0, 10.0),
  - `F(Time)` to (0.0, 0.0) *(default)*,
  - `Period` is set by default to 1.0×10^{20}, this will have no effect here, because `Period` > `Endtime`.

- Go to the directory `/BoundCond/FireFromRight/Right`,
- select *(by right mouse button)* `Linear` in the corresponding subdirectory `Interpolation` *(default)*,
- select `Dirichlet` within the subdirectory `Type` *(default)*,
- set the variables
  - `Time` to (0.0, 2.0, 10.0),
  - `F(Time)` to (0.0, 2.0, 2.0).

Initial Value

In the next step we assign the initial values for the heat conduction problem:

- go to the directory `/Library/InitialValue`,
- create an inital condition with the command `Create` and name it, e. g., 'Cold',
- create a vector with the command `AddVector` and name it, e. g., 'Temperature',
- assign a component to the vector with the command `StdFunctions` and name it, e. g., 'Component0',
- enter the directory `/IntialValue/Cold/Temperature/Component0`,
- set the variables
  - `Time` to (0.0, 1.0),
  - `F(Time)` to (0.0, 0.0) *(default)*.

Coefficient Functions

Now, the coefficient functions for the heat conduction problem have to be defined. In our example, these are the conductivities and the heat capacities for each subdomain:

- go to the directory `/Library/Coeffcient`,
- create a coefficent function set with the command `HeatConduction` and name it, e. g., 'HeatConduction',
- enter the directory `/Coefficients/HeatConduction`,

- set the variables per subdomain *(in this case only one)*
  - `StorageCapacity` to 1.0 *(default)*,
  - `Conductivity` to 0.1,
  - `HomogSource` to 0.0.

In the subdirectory `/Water` the flag `WaterDependend` is inactive *(default)*, in this case the remaining options will not be used.

Discretization Scheme

To complete the set of definitions, we have to choose an appropriate discretization scheme. Therefore

- go to the directory `/Library/Discretization`,
- select the discretization scheme with the command `FEM4HeatConduction`, and name it, e. g., 'Fem4HeatConduction'.

Building the Grid

Now, the grid has to be build up:

- go to the directory `/Library/Grid`,
- select the grid with the command `StandardGrid` and name it, e. g., 'Transport'.

Adding Problem and Discretization to the Grid

We add the problem and its discretization to the grid:

- go to the directory `/Library/Problem`,
- define your problem with the command `Parabolic` and name it, e. g., 'Getting-Warm',
- go to the directory `/Command`,
- add the problem and the discretization to the grid with the command `AddP&D2Grid` *(don't worry! Nothing will happen on screen)*.

Initialization of the Grid

We initialize the grid, i. e., the domain will be discretized by elements:

- Execute the command `InitializeGrid` in the /Command directory *(there is also no changing visible on screen)*.

Solver

In the next step we have to apply the solver:

- go to the directory `/Library/NLSolver`,
- select the Newton solver with the command `NewtonLS` and name it, e. g., 'Cracker'.

Timer

The set up of the simulation algorithm is finished by a timer:

- go to the directory `/Library/Timer`,
- select the implicit Euler method with the command `ImplicitEuler` and name it, e. g., 'Stepper',
- go to the directory `/Timer/Stepper`,
- set the variables
  - `Start` to 0.0 *(default)*,
  - `End` to 10.0,
  - `StepSize` to 0.05.

Use defaults for the other menu items.

Creating Plots

To display and follow up the progress in calculation, the interface provides a plotting feature. To make use of this, the user has to set up and define an output window for the simulation problem. We want to see the course of temperature during simulation. To do so

- enter the directory `/Discretization/FEM4HeatConduction/Variables`,
- select the entry `Temperature` *(default in this case)*,
- enter the directory `/Library/Plot`,
- select the space dependence of the temperature as plot with the command `GLGridPlot` and name it, e. g., 'Temperature',
- go to the directory `/Command`,
- invoke the plot with the command `Plot`.

RICHY1D now opens a graphical pop-up window entitled 'Temperature'.

Simulation

Now we are ready to run the simulation. To do so

- the initial values are set by execution of the command `InitializeData` in the `/Command` submenu.
- the simulation runs
  - until the time `End` is reached (see directory `/Timer`) by executing the command `Proceed` or
  - by one timestep of `StepSize` by executing the command `SingleStep`.

Remarks

In the following some useful features are listed:

- If you want to use the set up of a problem again it will be useful to create a scriptfile (cf.A.3 Scriptfile, p. 148). To save your interactive settings done with the GUI into a scriptfile, you can log your session. Therefore you just have to activate the flag `Log2File` in the toplevel directory `/GeneralSettings` before starting. You may edit the scriptfile afterwards with any text editor, if desired.
- The heat conduction discretization `FEM4HeatConduction` expect exactly one initial value vector which prescribe the temperature at the beginning of the simulation. In this session it is `Temperature` *(have a look to step 3)*. If you define more than one initial value vectors in `Cold` RICHY1D will ignore them. But you can create other scenarios by executing the command `Create` in the directory `/Library/InitialValue` and for each of this circumstances you can allign an other initial value vector. The current scenario has to be marked *(by the right mouse button)* then.
- In case of using more than one, say 4 subdomains you have to modify the variables
  - `/Domain/Oven/Subdomains` to 4,
  - `/Domain/Oven/InnerBoundaries` to (0.25, 0.5, 0.75) *(float vector of size `NumOfSubdomains` − 1)*,
  - `/Domain/Oven/ElemsPerSubDom` to (25, 25, 25, 25) *(integer vector of size `NumOfSubdomains`, you can expediently use different values in various subdomains)*,
  - `/Coefficient/HeatConduction/StorageCapacity` to (1.0, 1.0, 1.0, 1.0),
  - `/Coefficient/HeatConduction/Conductivity` to (0.1, 0.1, 0.1, 0.1),
  - `/Coefficient/HeatConduction/HomogSource` to (0.0, 0.0, 0.0, 0.0).
- If you have changed parts of the current grid it will be necessary to execute the command `InitializeGrid` in the `/Command` directory for restarting the calculation. For resetting the computation you have furthermore to click on the

command `Initialize` and optionally on the command `Plot`. Only this way you can start the simulation again.

A.2.2 Simulation of Solute Transport

This example session on the solution of the solute transport problem is a step-by-step introduction to the set up of problem specific simulation scenario. The instructions given herein can also be found in the corresponding scriptfile `def4trans.scr` in the directory `./appl/scripts/`. The content of this course corresponds to the work scheme (cf.1.2 Work Scheme, p. 5).

Statement of the Problem

This example provides a solution to the one-dimensional solute transport equation (3.4) including diffusion-dispersion, convection and equilibrium sorption. More information is given in section 3.2 Solute Transport, p. 62.

Domain

We create a domain named 'SoilColumn', represented by the interval [0, 10], consisting of one subdomain (it is not splitted into layers) and we choose a spatial grid of 200 elements:

- go to the directory `/Library/Domain`,
- allocate a domain with the command `1DDomain` and enter the name, e. g., 'SoilColumn',
- go to the directory `/Domain/SoilColumn`,
- set the variables
  - `LeftBoundary` to 0.0 *(default)*,
  - `RightBoundary` to 10.0,
  - `NumOfSubdomains` to 1 *(default)*,
  - *`InnerBoundaries` will be ignored because of the coherent domain*,
  - `ElemsPerSubDom` to 200 *(integer array, but due to the single subdomain just the first entry will be used)*.

Boundary Condition

We now have to assign proper values for the left and the right boundary conditions:

- go to the directory `/Library/BoundCond`,

- define a scalar boundary condition *(because we consider one single substance)* with the command `ScalarBC` and name it, e. g., 'TopInflow',
- enter the directory `/BoundCond/TopInflow/Left`,
- select *(by right mouse button)* `Linear` in the subdirectory `Interpolation` *(default)*, this defines the type of interpolation between the values that you specify,
- select `Flux` within the subdirectory `Type`,
- set the variables
  - `Time` to $(0.0,\ 1.0 \times 10^9)$ *(the last time point should be larger than the end time of your simulation which is specified in the `Timer` menu)*,
  - `F(Time)` to $(0.16,\ 0.16)$.
  - `Period` is set by default to 1.0×10^{20}, this will have no effect here since `Period` > `Endtime`.
- Go to the directory `/BoundCond/TopInflow/Right`,
- select *(by right mouse button)* `Linear` in the corresponding subdirectory `Interpolation` *(default)*,
- select `Neumann` within the subdirectory `Type`,
- set the variables
  - `Time` to $(0.0,\ 1.0 \times 10^9)$,
  - `F(Time)` to $(0.0,\ 0.0)$,
  - `Period` to 1.0×10^{20} *(default)*

Initial Value

In the next step we assign the initial values for the solute transport problem. We will not have any nonequilibrium sorption 'sorbate(neq)'. Therefore just one initial value vector has to be created. For getting the correct number of initial vectors have a look at remarks on p. 142.

- Go to the directory `/Library/InitialValue`,
- create an inital value condition with the command `Create` and name it, e. g., 'Empty',
- create a vector with the command `AddVector` and name it, e. g., 'solute',
- assign a component to the vector with the command `StdFunctions` and name it, e. g., 'const',

- enter the directory `/IntialValue/Empty/solute/const`,
- set the variables
  - `Coords` to $(-0.1,\ 10.1)$,
  - `F(Coords)` to $(0.0,\ 0.0)$ *default*.

Coefficient Functions

Now, the coefficient functions for the solute transport problem have to be defined. These are molecular diffusion d, dispersion length α_l, needed for the diffusion/dispersion tensor D, and the bulk density ρ_b. Furthermore the modelling of equilibrium sorption requires the mass fraction of the equilibrium sorption site f_ψ and a sorption isotherm with all corresponding parameters:

- go to the directory `/Library/Coeffcient`,
- create a coefficent function set with the command `Transport` and name it, e. g., 'ContaminantTransport',
- enter directory `/Coefficients/ContaminantTransport/Transport`,
- set the variables *(just an array of size* `NumOfSubdomains` $= 1$ *will be used respectively)*
  - `Diffusion` to 0.04,
  - `Dispersion` to 0.003,
  - `BulkDensity` to 1.0,
- enter the subdirectory `/Transport/EquiSorpt`,
- set the variables
  - `ParaType` to 1 *(i. e., Freundlich isotherm)*,
  - `MassFraction` to 1.0,
  - `Kd-Value` to 1.0,
  - `Exponent` to 0.5,
  - `Regularisation` to 0.001,
  - `MaxSorption` *(default, will be ignored since not needed in Freundlich isotherm)*.

We will neither simulate nonequilibrium sorbate concentrations nor releases caused by Raoult's law. Therefore the default settings (for being deactivated) in the subdirectories `/Transport/NonEquiSorpt` and `/Transport/Raoult` should be used. Additionally to the already defined values, the water flow has to be defined. Here we do not want to simulate the water flow by solving the Richards equation (3.9) but give explicit values for water flux q and water content Θ that define a stationary flow regime:

- Return to the directory `/Coefficient/ContaminantTransport`,
- enter the directory `/Water`,
- deactivate the flag `WithRichards` *(default)*,
- set the variables
  - `WaterContent` to 1.0,
  - `Time` to 0.0 *(default)*,
  - `Flux(Time)` to 1.0.

The commands `Connect2WaterContent` and `Connect2Flux` are not needed here. Use defaults for all the other parameters in the other subdirectories `carrier`, `decay` and `Stabilization4Transport`.

Discretization Scheme

To complement the set of definitions, we have to choose an appropriate discretization scheme. Therefore

- go to the directory `/Library/Discretization`,
- select the discretization scheme with the command `FEM4Transport`, and name it, e. g., 'FEM4Transport'.

Note

The variable solute and the created isotherms for sorbates in the equilibrium will be discretizised automatically without any more settings. If you want to calculate other values like `sorbate(neq)` or `solute_sens`, `sorbate(neq)_sens`, `sorbate(eq)_sens`, `sink-trans` and `sink-trans_sens` you have to appoint that additionally.

In this example we have no isotherms for the sorbate in the nonequilibrium but we want to simulate the sorbate in the equilibrium. Therefore

- go to the directory `/Discretization/Variables`,
- activate the flag `sorbate(eq)`.

Remark

It is not neccesary and even wrong to define a new unkown vector called 'sorbate(eq)' in step 3, Initial Value. This is a result of the fact that `sorbate(eq)` depends directly on the `solute` concentration and is not an additional unknown in the problem.

Building the Grid

Now, the grid has to be build up:

- go to the directory `/Library/Grid`,
- select the grid with the command `StandardGrid` and name it, e. g., 'Transport-Grid'.

Adding Problem and Discretization to the Grid

We add the problem and its discretization to the grid:

- go to the directory `/Library/Problem`,
- create the problem with the command `Parabolic` and name it, e. g.,'TransportProblem',
- go to the directory `/Command`,
- add the problem and the discretization to the grid with the command `AddP&D2Grid` *(nothing will happen on screen)*.

Initialization of the Grid

We initialize the grid, i. e., the domain will be discretized by elements:

- Execute the command `InitializeGrid` in the `/Command` directory *(there is also no changing visible on screen)*.

Solver

In the next step we have to apply the solver:

- go to the directory `/Library/NLSolver`,
- select the Newton solver with the command `NewtonLS` and name it, e. g., 'Cracker'.

The the solver's settings have to be defined:

- go to the directory `/NLSolver/Cracker`,
- set the variable `AbsError` to 1.0×10^{-6}.

Use the defaults for all the other variables.

Timer

The set up of the simulation algorithm is finished by a timer:

- go to the directory `/Library/Timer`,
- select the implicit Euler method with the command `ImplicitEuler` and name it, e. g., 'Stepper',
- go to the directory `/Timer/Stepper`,
- set the variables
  - `Start` to 0.0 *(default)*,
  - `End` to 30.0,
  - `StepSize` to 0.1.

Use defaults for the other menu items.

Creating Plots

To display and follow up the progress in calculation, the interface provides a plotting feature. To make use of this, the user has to set up and define an output window for the simulation problem. We want to see the course of the `solute` and the `sorbate(eq)` during simulation. Therefore we have to create two plots:

- Enter the directory `/Discretization/FEM4Transport/Variables`,
- select the entry `solute` *(by right mouse button)*,
- go to the directory `/Library/Plot`,
- to display the evolution of the unknown `solute` in the spatial domain execute the command `GLGridPlot` and name the plot, e. g.,'PlotSolute',
- enter the directory `/Discretization/FEM4Transport/Variables`,
- select this time the entry `sorbate(eq)`,
- return to the directory `/Library/Plot` again,
- execute the command `GLGridPlot` and name the plot, e. g.., 'PlotSorbate(eq)',
- go to the directory `/Command`,
- invoke the plot with the command `Plot`.

RICHY1D now opens two graphical pop-up windows entitled 'PlotSolute' and 'PlotSorbate(eq)' respectively.

Simulation

Now we are ready to run the simulation. To do so

- the initial values are set by execution of the command `InitializeData` in the `/Command` menu,
- the simulation runs
  - until the time `End` is reached (see directory `/Timer`) by executing the command `Proceed` or
  - by one timestep of `StepSize` by executing the command `SingleStep`.

Remarks

In the following some useful features are listed:

- If you want to use the set up of a problem again it will be useful to create a scriptfile (cf.A.3 Scriptfile, p. 148). To save your interactive settings done with the GUI into a scriptfile, you can log your session. Therefore you just have to activate the flag `Log2File` in the toplevel directory `/GeneralSettings` before starting. You may edit the scriptfile afterwards with any text editor, if desired.
- The number of initial and boundary value vectors which are expected by RICHY1D in case of a solute transport discretization depends on the number of unknowns of the problem. The unknowns are the solute concentration and optionally up to five nonequilibrium sorption concentrations (note that the equilibrium sorbate concentration is not an additional unknown of the problem—it can be calculated directly from the solute concentration). For each unknown initial and boundary conditions have to be provided! Therefore the unchangeable sequence of the vectors reads as follows:
  1. Vector describing the solute.
  2. Vector describing the first nonequilibrium sorbate, corresponding to the first type of sorption sites (described by an isotherm).
  3. Vector describing the second nonequilibrium sorbate, etc.

  If you define to less vectors RICHY1D will print an error message without starting the simulation. In case of to much defined vectors the program will ignore them. Other problem classes can handle the number of initial vectors with have to be defined different. Have a look at section 3.6 Biodegradation, p. 99). You have the possibility to create as much other scenarios as you want by executing the command `Create` in the directory `/Library/InitialValue` and for each of these scenarios you can align different initial value vectors. The current scenario has to be selected then. In this session due to nonexisting isotherms of sorbates in the nonequilibrium just one vector solute exists.

- If you have changed parts of the current grid it will be necessary to execute the command `InitializeGrid` in the `/Command` directory for restarting the calculation. For resetting the computation you have furthermore to click on the command `Initialize` and optionally on the command `Plot`. Only now you can start the simulation again.

A.2.3 Simulation of Water-Flow (Richards Equation)

This example session on the solution of the Richards equation describing the water flow is a step-by-step introduction to the set up of problem specific simulation scenario. The instructions given herein can also be found in the corresponding scriptfile `def4richy.scr` in the directory `./appl/scripts/`. The content of this course corresponds to the work scheme (cf.1.2 Work Scheme, p. 5).

Statement of the Problem

This example provides a solution to the one-dimensional Richards equation (3.9) on a two-layered soil column. More information is given in section 3.3 Saturated/Unsaturated Water Flow (Richards Equation), p. 78.

Domain

We create a domain named 'twolayer', represented by the interval $[-1,\ 1]$, consisting of two subdomains and we choose a spatial grid of 50 elements per subdomain:

- go to the directory `/Library/Domain`,
- allocate a domain with the command `1DDomain` and enter the name, e. g., 'twolayer',
- go to the directory `/Domain/twolayer`,
- set the variables
  - `LeftBoundary` to -1.0,
  - `RightBoundary` to 1.0,
  - `NumOfSubdomains` to 2,
  - `InnerBoundaries` to 0.0,
  - `ElemsPerSubDom` to $(50,\ 50)$ *(integer array, following entries will be ignored)*.

Boundary Condition

We now have to assign proper values for the left and the right boundary conditions:

- go to the directory `/Library/BoundCond`,

- define a scalar boundary condition with the command `ScalarBC` and name it, e. g., 'watertableandflux',
- enter the directory `/BoundCond/watertableandflux/Left`,
- select *(by right mouse button)* `Linear` in the subdirectory `Interpolation` *(default)*, this defines the type of interpolation between the values that you specify,
- select `Dirichlet` within the subdirectory `Type`,
- set the variables
  - `Time` to $(0.0,\ 1.0 \times 10^3)$ *(the last time point should be larger than the end time of your simulation which is specified in the `Timer` menu)*,
  - `F(Time)` to $(0.0,\ 0.0)$.
  - `Period` is set by default to 1.0×10^{20}, this will have no effect here since `Period` > `Endtime`.
- go to the directory `/BoundCond/watertableandflux/Right`,
- select *(by right mouse button)* `Linear` in the corresponding subdirectory `Interpolation` *(default)*,
- select `Flux` within the subdirectory `Type`,
- set the variables
  - `Time` to $(0.0,\ 1.0 \times 10^2,\ 1.0 \times 10^3)$,
  - `F(Time)` to $(2.77 \times 10^{-7},\ 2.5 \times 10^{-6},\ 2.5 \times 10^{-6}$,
  - `Period` to 1.0×10^{20} *(default)*

Initial Value

In the next step we assign the initial values for the Richards equation with use of the command `RichyEqui`. As described in section Initial Value, p. 81, the initial value function `RichyEqui` only requires the parameter `Flux` while the pressure head ψ and the water content Θ will be calculated.

- Go to the directory `/Library/InitialValue`,
- create an inital value condition with the command `Create` and name it, e. g., 'PressureProfile',
- create a vector with the command `AddVector` and name it, e. g., 'Pressure',
- execute the command `RichyEqui`,
- enter the directory `/IntialValue/PressureProfile/Pressure/Equilibrium`,
- set `Flux` to -2.77×10^{-7}.

Coefficient Functions

Now, the parametrisation of the coefficient functions—these are water content Θ and hydraulic conductivity K—has to be chosen and included parameters have to be defined:

- go to the directory `/Library/Coeffcient`,
- create a coefficent function set with the command `Richards` and name it, e. g., 'Richards',
- enter directory `/Coefficients/Richards`,
- set the variables *(just an array of size* `NumOfSubdomains` $= 2$ *will be used respectively)*
  - `ParaType` to $(0,\ 0)$ *(i. e., Gardner estimation for both subdomains)*,
  - `K_sat` to $(2.77 \times 10^{-6},\ 2.77 \times 10^{-5})$,
  - `T_sat` to $(0.36,\ 0.36)$,
  - `T_res` to $(0.06,\ 0.06)$,
  - `alpha` to $(10.0,\ 10.0)$.

Preparing the Grid

Now, the grid has to be build up:

- go to the directory `/GeneralSettings`,
- set `ZDirection` to 1.0.
- Go to the directory `/Library/Grid`,
- select the grid with the command `StandardGrid` and name it, e. g., 'flowgrid'.

Building Boundary Value Problem and Finishing the Grid

To complement the set of definitions, we have to create a problem and to choose an appropriate discretization scheme. Therefore

- go to the directory `/Library/Problem`,
- execute the command `Parabolic` and name the problem, e. g., 'flowbvp',
- go to the directory `/Library/Discretization`,
- select the discretization scheme with the command `HMFEM4Richards`, and name it, e. g., 'HMFEM4Richards'.

We want to use time adaption. Therefore the following settings are declared:

- go to the directory `iscretization/HMFEM4Richards`,
- set the variables
  - `ErrorTime` to 1.0×10^{-5},
  - `CoarseningRatio4Time` to 6.

Add the problem and the discretization to the grid by executing the command `AddP&D2Grid` in the command directory. Finish the grid with the command `InitializeGrid` *(in both cases nothing will happen on screen)*.

Solver

In the next step we have to apply the solver:

- go to the directory `/Library/NLSolver`,
- select the Newton solver with the command `NewtonLS` and name it, e. g., 'cracker'.

The the solver's settings have to be defined:

- go to the directory `/NLSolver/Cracker`,
- set the variables
  - `AbsError` to 1.0×10^{-20},
  - `RelError` to 1.0×10^{-6}.

Use the defaults for all the other variables.

Timer

The set up of the simulation algorithm is finished by a timer:

- go to the directory `/Library/Timer`,
- select the implicit Euler method with the command `ImplicitEuler` and name it, e. g., 'stepper',
- go to the directory `/Timer/stepper`,
- set the variables
  - `Start` to 0.0 *(default)*,
  - `End` to 1.0×10^{6},
  - `MinStepSize` to 1.0×10^{-5},
  - `MaxStepSize` to 1.0×10^{5}.

Enable the flag `AdaptTime` and disable the flag `AdaptGrid`. Use defaults for the other menu items. Initialize all datas by executing the command `InitializeData` in the command directory.

Creating Plots

To display and follow up the progress in calculation, the interface provides a plotting feature. To make use of this, the user has to set up and define an output window for the simulation problem. We want to see the course of the pressure head ψ and the water flux q during simulation. Therefore we have to create two plots:

- Enter the directory `/Discretization/HMFEM4Richards/Variables`,
- select the entry `pressure` *(by right mouse button)*,
- go to the directory `/Library/Plot`,
- to display the evolution of the unknown `pressure` in the spatial domain execute the command `GLGridPlot` and name the plot, e. g.,'pressure',
- enter the directory `/Discretization/HMFEM4Richards/Variables`,
- select this time the entry `flux`,
- return to the directory `/Library/Plot` again,
- execute the command `GLGridPlot` and name the plot, e. g., 'flux',
- go to the directory `/Command`,
- invoke the plot with the command `Plot`.

RICHY1D now opens two graphical pop-up windows entitled 'pressure' and 'flux' respectively.

Simulation

Now we are ready to run the simulation. To do so

- the initial values are set by execution of the command `InitializeData` in the `/Command` menu *(done already above)*,
- the simulation runs
  - until the time `End` is reached (see directory `/Timer`) by executing the command `Proceed` or
  - by one timestep of `StepSize` by executing the command `SingleStep`.

Also have a look at remarks on p. 142.

A.3 Scriptfile

RICHY1D can work without any user interaction. You can choose a startup file, to be loaded automatically. Interactively generated sessions can be logged into a file, which can be modified by usual ASCII text editors and executed in later sessions. The execution of scriptfiles can be nested.

Logging a Session

➟ `/GeneralSettings/Log2File`
You start/stop the logging of your session into the file `./appl/scripts/logfile.scr` by activating/deactivating the flag `Log2File` in the toplevel directory `/GeneralSettings`. By default, logging is off. Any previously generated session in `./appl/scripts/logfile.scr` is deleted when starting a new session. If you want to keep the script file of your session, choose a new filename and copy `./appl/scripts/logfile.scr` to `./appl/scripts/filename.scr`.

Execution of Scriptfiles

➟ `/Command/ExecScript`
You can execute a series of commands, logged or written previously into a file, by execution of the command `ExecScript`. You will be asked to enter a filename. This file is searched for in the subdirectory scripts.

Scriptfile Language

The scriptfile language consists of the following actions each of which is written in one line within the scriptfile.

`GOTO [name of toplevel directory]`
Changes the current directory to one of the toplevel directories. You can move only down, but not up. So you always have to start from a toplevel directory.

```
GOTO /Domain
```

`CD [name of subdirectory]`
Changes the current directory to the defined subdirectory.

```
CD Oven
```

`HIDE [name of menuitem]`
Hides the defined menuitem (i. e., variable, command, directory, etc.). The menuitem is searched for under the current directory. If it is not found here, the search starts again from the toplevel.

```
HIDE /Variable
```

`SET [name of variable] [list of values]`
Sets the value of the defined variable. The form of `[list of values]` depends on the type of the variable:

- Floating point or integer number: A single value of the desired type,
- array of floating point or integer numbers: First the number of values and then the sequence of these values,
- matrix of floating point or integer numbers: Specify number of lines i, then number of columns j, then j values for first line, up to j values for ith line.

```
SET #LeftBoundary#          0.0
SET #RightBoundary#         1.0
SET #InnerBoundaries# 3     0.2 0.4 0.9
SET #ElemsPerSubdom#  4     20  20  50  10
SET #Matrix#          2 3   1.0 1.0 1.0
                            2.0 2.0 2.0
```

`SETFROMFILE [name of variable] "[filename]"`
Sets the values of a integer or floating point array variable. The values are read from the file given by `[filename]` which must be enclosed by quotes. The file format is as follows:

```
[first line is for comments/identification. it is ignored]
[n: number of values (integer)]
[value 0]
[ ... ]
[value n-1]
```

Values are either integer or floating point—dependent on the variable's type. If more values are given than the variable holds, the additional values are ignored. If less values are given, the not specified array entries are set to 0.
`SELECT [name or number of menuitem]`
Selects a menuitem, defined by its name or number, within the current directory.

```
GOTO   /Domain
SELECT Oven
```

`CMD [name of command] [string]`
Executes the defined command. The string contains further information for the command (e. g.a filename). Even if the command does not require any further information, the string may not be empty and you should at least write something like 'dummy'.

```
CMD *ExecScript* session.scr
CMD *Plot*       dummy
```

```
FOR [replacement character] [lower value] [upper value] [step size]
[code]
ENDFOR
```

Note that `upper value` − `lower value` mod `stepsize` $\stackrel{!}{=} 0$.

```
GOTO /Plot
CD   SorbatTimePlot
FOR X 8 12 2
SET  #Coordinate# X
CMD  *SaveData* coord_X.dat
ENDFOR
```

```
END
```

Tells the interpreter that the execution of the scriptfile is finished. The execution of commands will be continued by the user interface.

```
END
```

If you want to quit the application you have to use the following command within the scriptfile:

```
CMD *Quit*
```

A.4 Stylesheet

You can adapt RICHY1D's graphical appearance to your own ideas by changing the stylesheet `./appl/tcl/browser_conf.tcl`. The meaning of the variables are very intuitive and commented within this file. I.a. the following fonts are available:

Monospace Fonts	Proportional Fonts
Courier	Arial
LucidaTypewriter	Helvetica
FotinosTypewriter	Times
Terminal	

Bibliography

Bethke, C. M. (1996). *Geochemical Reaction Modeling*. New York: Oxford University Press.

Bitterlich, S. (2003). "Identifizierung der hydraulischen Funktionen poröser Medien unter Verwendung formfreier Ansätze." PhD thesis. Institute of Applied Mathematics, Friedrich–Alexander-University Erlangen–Nürnberg.

Bitterlich, S., W. Durner, S. C. Iden, and P. Knabner (2004). "Inverse estimation of the unsaturated soil hydraulic properties from column outflow experiments using free-form parameterizations." *Vadose Zone Journal* 3, pp. 971–981. DOI: 10.2136/vzj2004.0971.

Blume, M. (2011). "Identifizierung nichtlinearer Koeffizientenfunktionen des reaktiven Transports durch poröse Medien unter Verwendung rekursiver und formfreier Ansätze." PhD thesis. Institute of Applied Mathematics, Friedrich–Alexander-University Erlangen–Nürnberg.

De Vries, D. A. and N. H. Afgan (1975). *Heat and Mass Transfer in the Biosphere*. New York: Wiley.

Frank, F. (2008). "Hydrogeochemical Multi-Component Transport - Mineral Dissolution and Precipitation with Consideration of Porosity-Changes in Variably-Saturated Porous Media." Diploma Thesis. Institute of Applied Mathematics, Friedrich–Alexander-University Erlangen–Nürnberg.

Gardner, W. (1958). "Some steady state solutions for the unsaturated moisture flow equation with application to evaporation from a water table." *Soil Science* 85.4, pp. 228–232.

Igler, B. (1998). "Identification of Nonlinear Coefficient Functions in Reactive Transport through Porous Media." PhD thesis. Institute of Applied Mathematics, Friedrich–Alexander-University Erlangen–Nürnberg.

Igler, B., K. U. Totsche, and P. Knabner (1998). "Identification of Nonlinear Sorption Isotherms by Soil Column Breakthrough Experiments." *Physics and Chemistry of the Earth* 23.2, pp. 215–219.

Knabner, P. and L. Angermann (2003). *Numerical Methods for Elliptic and Parabolic Partial Differential Equations*. Vol. 44. Texts in Applied Mathematics. New York: Springer.

Knabner, P. and E. Schneid (1996). "Qualitative Properties of a Model for Carrier Facilitated Groundwater Contaminant Transport." English. *Scientific Computing in Chemical Engineering*. Ed. by F. Keil, W. Mackens, H. Voß, and J. Werther. Springer Berlin Heidelberg, pp. 129–135. DOI: 10.1007/978-3-642-80149-5_14.

Knabner, P., K. U. Totsche, and I. Kögel-Knabner (1996). "The modeling of reactive solute transport with sorption to mobile and immobile sorbents; 1. Experimental evidence and model development." *Water Resources Research* 32.6, pp. 1611–1622.

Magee, B. R., L. W. Lion, and A. T. Lemley (1991). "Transport of dissolved organic macromolecules and their effect on the transport of phenanthrene in porous media." *Environmental Science & Technology* 25, pp. 323–331.

Prechtel, A., P. Knabner, E. Schneid, and K. U. Totsche (2002). "Simulation of Carrier Facilitated Transport of Phenanthrene in Layered Soil Profile." *Journal of Contaminant Hydrology* 56.3-4, pp. 209–225.

Renshaw, C. E., G. D. Zynda, and J. C. Fountain (1997). "Permeability reductions induced by sorption of surfactant." *Water Resources Research* 33.3, pp. 371–378.

Richards, L. A. (1931). "Capillary conduction of liquids through porous mediums." PhD thesis. New York: Cornell University.

Roy, S. B. and D. A. Dzombak (1997). "Chemical factors influencing colloid-facilitated transport of contaminants in porous media." *Environmental Science & Technology* 31, pp. 656–664.

Schirmer, M., J. W. Molson, E. O. Frind, and J. F. Barker (2000). "Biodegradation modelling of a dissolved gasoline plume applying independent laboratory and field parameters." *Journal of Contaminant Hydrology* 46, pp. 339–374.

Schneid, E., A. Prechtel, and P. Knabner (2000). "A comprehensive tool for the simulation of complex reactive transport and flow in soils." *Land Contamination & Reclamation* 8.4, pp. 357–365.

Smith, J. E. and R. W. Gillham (1994). "The effect of concentration-dependent surface tension on the flow of water and transport of dissolved organic compounds: A pressure head-based formulation and numerical model." *Water Resources Research* 30.2, pp. 343–354.

Strehmel, K. and R. Weiner (1995). *Numerik gewöhnlicher Differentialgleichungen*. Stuttgart: Teubner.

Totsche, K. U. (1996). "SIMFONI — a numerical model for the simultaneous simulation of water, heat and nitrogen dynamics in forest ecosystems." *BITÖK Forschungsbericht 1995* 28.

Totsche, K. U., P. Knabner, and I. Kögel-Knabner (1996). "The modeling of reactive solute transport with sorption to mobile and immobile sorbents; 2. Model discussion and numerical simulation." *Water Resources Research* 32.6, pp. 1623–1634.

W. Stumm, J. M. (1996). *Aquatic chemistry: an introduction emphasizing chemical equilibria in natural waters*. New York: John Wiley and Sons.

Widdowson, M. A., F. J. Molz, and L. D. Benefield (1988). "A numerical Transport model for oxygen- and nitrate-based respiration linked to substrate and nutrient availability in porous media." *Water Resources Research* 24.9, pp. 1553–1565.

Index