Geosci. Model Dev., 15, 1659–1676, 2022
https://doi.org/10.5194/gmd-15-1659-2022

Geoscientific Model Development
Open Access
EGU

The PFLOTRAN Reaction Sandbox

Glenn E. Hammond

Environmental Subsurface Science Group, Pacific Northwest National Laboratory, Richland, WA, USA

Correspondence: Glenn E. Hammond (glenn.hammond@pnnl.gov)

Received: 14 August 2021 – Discussion started: 23 September 2021
Revised: 7 January 2022 – Accepted: 28 January 2022 – Published: 25 February 2022

Abstract. As modern reactive transport simulators evolve to accommodate the demands of a user community, researchers need a platform for prototyping new biogeochemical processes, many of which are niche and specific to laboratory or field experiments. The PFLOTRAN Reaction Sandbox leverages modern, object-oriented Fortran in an attempt to provide such an environment within an existing reactive transport simulator. This work describes the PFLOTRAN Reaction Sandbox concept and implementation through several illustrative examples. Reaction Sandbox Biodegradation Hill customizes the existing microbially mediated biodegradation reaction formulation within PFLOTRAN to better match empirical data. Reaction Sandbox Simple provides an isolated environment for testing numerous preconfigured kinetic rate expressions and developing user intuition. Reaction Sandbox Example serves as a template for creating new sandboxes within PFLOTRAN.

1 Introduction

Modern reactive transport simulators incorporate sophisticated networks of reactions to simulate complex biogeochemical processes within the Earth's subsurface environment in support of scientific research in climate change, fate and transport of contaminants, and water resources management (Steefel et al., 2005). As these simulators mature and evolve over time, they can accumulate a large number of chemical reactions with a wide range of implementations. Many reactions are somewhat common among reactive transport codes (e.g., aqueous complexation, mineral precipitation–dissolution, radioactive decay and sorption; Steefel et al., 2015), while others are niche, prototypical or problem dependent.

There is a multitude of possible reactions to include within a simulator, and the approach to implementing these reactions often varies among simulators. For instance, sorption processes include absorption, adsorption and ion exchange. Surface complexation is an adsorptive process that can be simulated using a constant capacitance, (diffuse) double layer, triple layer or non-electrostatic model (Bethke, 2007). Depending on the reaction timescales, surface complexation may be simulated assuming local equilibrium or driven by a kinetic rate expression, and kinetic approaches may include single and multi-rate models, the latter utilizing a distribution of rate constants associated with size fractions of sediment material (Liu et al., 2008).

It is also common for researchers to develop one-off implementations of reactions when existing capabilities cannot replicate biogeochemical phenomena observed in field or laboratory experiments. For example, Tutolo et al. (2018) demonstrated that the brucite silicification is a serpentinization rate-limiting reaction that is exponentially dependent upon aqueous silica activity. Their brucite silicification reaction required the implementation of a custom rate law that was exponentially dependent upon the activity of aqueous silica. These one-off implementations quickly increase the diversity of reactions supported by a simulator. As the number of reactions implemented within a simulator grows, and the reactions become increasingly diverse, adding new reactions can challenge ongoing simulator development and maintenance, especially within an open-source community where code development is often crowd-sourced.

Due to time and funding limitations, it is difficult for code developers to satisfy the needs of the user community by implementing all variants of a reactive process. Nor does it make sense when reactions are problem-specific and may never be used in the future. Meanwhile, the end-users who are requesting customization are often non-computational

domain scientists with less interest in or limited understanding of numerical methods, programming paradigms, code abstractions and data layout. Their focus is more on improving predictive accuracy through the refinement of reaction conceptual models, not code development. This raises the question as to what steps can be taken in the design and implementation of a reactive transport simulator to facilitate code development and long-term maintenance while flattening the (often) steep learning curve for new developers.

The PFLOTRAN Reaction Sandbox is an attempt to provide such an environment within an existing reactive transport simulator. The purpose of the Reaction Sandbox is to provide a means for testing alternative implementations for kinetically formulated rate expressions or networks of these reactions in conjunction with the existing reactive transport capability within PFLOTRAN. Within computer science, the term *sandbox* often refers to an environment for implementing and vetting new, untested algorithms in isolation. The purpose of a sandbox is to limit the impact on the remainder of the code.

The following sections document the implementation of the PFLOTRAN Reaction Sandbox and demonstrate its application on several problem scenarios. Section 2 provides an overview of PFLOTRAN by presenting the governing equations for reactive transport and the numerical methods employed to solve the resulting discrete nonlinear systems of equations. The section also presents PFLOTRAN's conventional reactive transport capability and discusses the code's modular object-oriented design that facilitates the implementation of the Reaction Sandbox. Section 3 describes the foundational (Fortran) Reaction Sandbox class upon which all Reaction Sandbox classes are coded and the high-level programming interface through which the remainder of the code accesses sandbox reactions. Section 4 documents several example sandboxes from which a researcher may derive their own implementation. Finally, Sect. 5 summarizes the approach.

2 Background

PFLOTRAN is an massively parallel, reactive multiphase flow and transport simulator for modeling subsurface earth system processes (Hammond et al., 2014). The code has been developed under open-source GNU LGPL licensing since 2009 and contains contributions from an international group of developers. PFLOTRAN is written in Fortran 2003/2008, which facilitates object-oriented design through the use of nested derived types, classes (extensible derived types with member procedures) and procedure pointers. The code is designed as a nested hierarchy of objects. The top-level simulation object contains pointers to all process models, solvers and supporting data (state variables, parameters, etc.) needed to run a simulation.

PFLOTRAN simulates biogeochemical transport through its reactive transport process model. Supported biogeochemical reaction capabilities include aqueous speciation with ion activity models, general Nth-order forward (and reversible) kinetic reactions, microbially mediated reactions, mineral precipitation–dissolution, radioactive decay, and ingrowth and sorption (Andre et al., 2022). These reactions are referred to as *conventional* reactive transport capability as the implementations are based on commonly accepted approaches that are well-documented in the literature.

2.1 Governing equations

PFLOTRAN's governing mass conservation equation for reactive transport of aqueous species j is

$$\frac{\partial}{\partial t}\left(\phi s \Psi_j\right) + \nabla \cdot \left(\boldsymbol{q} - \phi s \mathbf{D} \nabla\right) \Psi_j \;=\; Q_j - \sum_r \nu_{j\mathrm{r}} I_\mathrm{r}, \tag{1}$$

with porosity ϕ, liquid saturation s, total aqueous component concentration Ψ_j, Darcy fluid flux $\boldsymbol{q}$, diagonal hydrodynamic dispersion tensor $\mathbf{D}$ and source term Q_j. $\nu_{j\mathrm{r}}$ represents the stoichiometry of species j in kinetic reaction I_r. Following the continuum formulation for reactive transport (Lichtner, 1985) and assuming local equilibrium (Rubin, 1983), the total aqueous component concentration of species j is the sum of the free ion concentration c_j and its stoichiometric contribution ν_{ji} to each secondary aqueous complex X_i,

$$\Psi_j \;=\; c_j + \sum_i \nu_{ji} X_i. \tag{2}$$

Aqueous complex X_i is calculated through mass action as

$$X_i \;=\; \frac{K_i}{\gamma_i} \prod_{j'} \left(\gamma_{j'} c_{j'}\right)^{\nu_{j'i}} \tag{3}$$

with equilibrium constant K_i, activity coefficients γ_i and $\gamma_{j'}$, stoichiometry $\nu_{j'i}$, and primary aqueous species free ion concentration $c_{j'}$. The governing mass conservation equation for the jth primary immobile species Φ is

$$\frac{\partial}{\partial t}\left(\Phi_j\right) \;=\; -\sum_r \nu_{j\mathrm{r}} I_\mathrm{r}. \tag{4}$$

Ignoring the advection, hydrodynamic dispersion and source terms and assuming constant porosity and saturation, the discrete (finite volume) forms of Eqs. (1) and (4) are

$$\frac{\phi s V}{\Delta t}\left(\Psi_j^{k+1} - \Psi_j^k\right) \;\ldots\; =\; \ldots \;- V \sum_r \nu_{j\mathrm{r}} I_\mathrm{r} \tag{5}$$

and

$$\frac{V}{\Delta t}\left(\Phi_j^{k+1} - \Phi_j^k\right) \;=\; -V \sum_r \nu_{j\mathrm{r}} I_\mathrm{r}, \tag{6}$$

respectively. Units for these equations are [mole s^{-1}]. Rate expression I_r represents the primary species mass consumed or produced by each kinetic reaction and has units of mole $\mathrm{m}_{\mathrm{bulk}}^{-3}$ s^{-1}.

2.2 Numerical solution technique

PFLOTRAN employs the finite volume method for spatial discretization, backward Euler time integration and Newton's method for solving the resulting nonlinear system of equations. Newton's method converges to a solution for the primary species concentrations $\boldsymbol{x}$ by iteratively evaluating the residual function

$$f(\boldsymbol{x}^p) = 0 \tag{7}$$

and Jacobian $\mathbf{J}$, and solving the linear system

$$\mathbf{J}\delta\boldsymbol{x} = -f(\boldsymbol{x}^p) \tag{8}$$

for the concentration update δx

$$\boldsymbol{x}^{p+1} = \boldsymbol{x}^p + \boldsymbol{\delta x}. \tag{9}$$

p is the iteration number.

The residual function f is evaluated by rearranging Eqs. (5) and (6) and setting them equal to zero. For example, the residual for aqueous degree of freedom c_n is

$$f(c_n^p) = \frac{\phi s V}{\Delta t}\left(\Psi_n^{k+1} - \Psi_n^k\right) \ldots + V\sum_r \nu_{nr} I_r = 0, \tag{10}$$

while for immobile degree of freedom Φ_n, it is

$$f(\Phi_n^p) = \frac{V}{\Delta t}\left(\Phi_n^{k+1} - \Phi_n^k\right) + V\sum_r \nu_{nr} I_r = 0. \tag{11}$$

Since the focus of this research is the chemical reaction, the advection, dispersion and source terms (represented by ellipses …) are ignored. The Jacobian contains derivatives of the residual with respect to the primary unknowns, i.e.,

$$J_{n,m} = \frac{\partial f(x_n^p)}{\partial x_m^p}. \tag{12}$$

Units for internal PFLOTRAN variables are documented in Table 1. These units must be considered in the development of a Reaction Sandbox.

2.3 Object-oriented Fortran

Modern programming paradigms facilitate the modularity, extensibility and overall longevity of a software product. A common feature among most modern programming paradigms is support for object-oriented design, where objects contain the data structures and procedures necessary to provide functionality, and interfaces are set up for interaction between objects. One major benefit of object-oriented design is that when programmed correctly, modifications to an object's data structures and procedures have little to no impact on other portions of the code. This modularity greatly facilitates the initial development and long-term maintenance of a code.

Modern Fortran facilitates object-oriented design through the use of derived types and classes. A Fortran derived type is a container that encapsulates other Fortran data types (e.g., logicals, integers, reals, other derived types, or pointers to data types). A Fortran class is an extensible version of the derived type that supports inheritance of (type-bound) member variables and procedures. A child class, created by extending the parent derived type, inherits all parent class variables and procedures. New member variables may be added to child classes and member procedures may be overridden. By default, a member procedure receives the class object as the first entry in function or subroutine argument lists. (A *class object* is simply an instantiation of the class.) The procedure uses the object's member variables and the remaining arguments to perform calculations. Chapman (2018) describes Fortran classes in greater detail.

PFLOTRAN is designed as a nested hierarchy of dynamically allocated objects from the highest-level simulation object down to low-level, cell-centric auxiliary objects that store all state variables for each grid cell. Given a pointer to the top-level simulation object, the developer has access to all underlying data structures or objects. This hierarchy engenders modularity and structure within the code. PFLOTRAN employs modern Fortran 2003/2008 where all objects are instantiations of Fortran derived types or classes. It also leverages pointers to procedures and common procedure interfaces to allow users to choose from a suite of constitutive relations, equations of state, flux algorithms, etc., in support of run time options. The modularity afforded through object-oriented Fortran has greatly facilitated the incorporation of new algorithms within PFLOTRAN, as will be shown through the implementation of the Reaction Sandbox.

3 Reaction sandbox approach

The PFLOTRAN Reaction Sandbox provides a simplified interface for implementing new kinetically formulated reactions (rate expressions) within PFLOTRAN that are tailored to specific user needs. The term I_r on the right side of Eqs. (1) and (4) represents these kinetic reactions in the governing equations. The Reaction Sandbox isolates one-off or application-specific reactions from the remainder of the PFLOTRAN code base and facilitates code development and long-term maintenance. For the domain scientist desiring to implement a new reaction, the Reaction Sandbox reduces complexity by exposing only the limited set of variables that are necessary to calculate rate expressions. Thus, the researcher is better shielded from the intricate details of code development. For kinetic reactions that have the potential for wider acceptance, the Reaction Sandbox provides a venue for vetting reactions in isolation prior to adoption within the main code base. This section describes the Reaction Sandbox concept and defines the underlying data structures and user interface.

Table 1. Units for PFLOTRAN internal variables.

Variables	Symbol	Units
Porosity	ϕ	$m^3_{pore}\, m^{-3}_{bulk}$
Liquid saturation	s	$m^3_{water}\, m^{-3}_{pore}$
Volume	V	m^3_{bulk}
Time	t	s
Aqueous free ion concentration	c	mole kg^{-1}_{water} (or molality)
Total aqueous component concentration	Ψ	mole L^{-1}_{water} (or molarity)
Immobile concentration	Φ	mole m^{-3}_{bulk}
Stoichiometry	ν	–
Kinetic rate	I_r	mole $m^{-3}_{bulk}\, s^{-1}$
Residual function	$f(x^p)$	mole s^{-1}
Derivative of residual w.r.t. free ion concentration	$\frac{\partial f(x_n^p)}{\partial c_m^p}$	$kg_{water}\, s^{-1}$
Derivative of residual w.r.t. immobile concentration	$\frac{\partial f(x_n^p)}{\partial \Phi_m^p}$	$m^3_{bulk}\, s^{-1}$

3.1 Concept

Figure 1 illustrates the hierarchical structure of (hypothetical) Reaction Sandboxes within PFLOTRAN where reaction classes are derived as descendants of the Reaction Sandbox Base Class. Each Reaction Sandbox class is implemented as a separate module within the PFLOTRAN source code. The end-user specifies the reaction sandboxes to be employed within the PFLOTRAN input file, and a linked list of sandbox reactions is constructed during simulation initialization. Figure 2 illustrates a representative linked list of sandboxes composed of two reaction objects from Fig. 1. The outer Reaction Sandbox module loops over the linked list to perform all operations (e.g., setup, evaluation, destruction) as illustrated later in Code Block 2. Linked lists improve flexibility for the code developer as reactions may be inserted or appended in any order. Newly developed reaction classes are added to the source code as daughter classes in new modules, and instantiated objects are added to the list during simulation initialization.

3.2 Implementation

The Reaction Sandbox is founded upon two files within the PFLOTRAN source code: `reaction_sandbox_base.F90` and `reaction_sandbox.F90`.

3.2.1 `reaction_sandbox_base.F90`

`reaction_sandbox_base.F90` defines the base (or parent) Fortran class `reaction_sandbox_base_type` that the developer extends to create new (child) Reaction Sandbox classes. Code Block 1 illustrates the member variables and procedures within the `reaction_sandbox_base_type` class. This class contains a single member variable `next`, a pointer to the next class of the same `reaction_sandbox_base_type` type that enables the creation of an abstract linked list of Reaction Sandbox objects. The class also contains empty member procedures with prescribed subroutine interfaces (or argument lists) that the developer overrides in child classes. With the exception of `BaseEvaluate`, these procedures are empty and return immediately if not overridden by the child class. `BaseEvaluate` must be extended, and error messaging is incorporated within `BaseEvaluate` to ensure correct implementation. The other member procedures are optional and do not require implementation in the child classes. Appendix A documents the `BaseXXX` member procedure interfaces.

3.2.2 `reaction_sandbox.F90`

`reaction_sandbox.F90` serves as the main driver interface for the Reaction Sandbox, providing subroutines that manage the creation, reading, setup, execution and destruction of all Reaction Sandbox objects. For example, the subroutine `RSandboxRead` instantiates Reaction Sandbox objects based on the keywords parsed from the `REACTION_SANDBOX` block in the input file. `RSandboxEvaluate` calculates reaction rates by traversing the linked list of Reaction Sandboxes evaluating individual rates as shown in Code block 2 (subroutine argument lists have been omitted for simplicity). With the exception of `RSandboxRead`, all member procedures are executed from within a linked list loop similar to that shown in Code block 2. Thus, subroutines within `reaction_sandbox.F90` serve as interfaces to the linked lists of Reaction Sandbox objects, and all information is exchanged through these routines. The outer `RSandboxXXX` subroutines defined in `reaction_sandbox.F90` may be called from other PFLOTRAN modules (e.g., see the call to

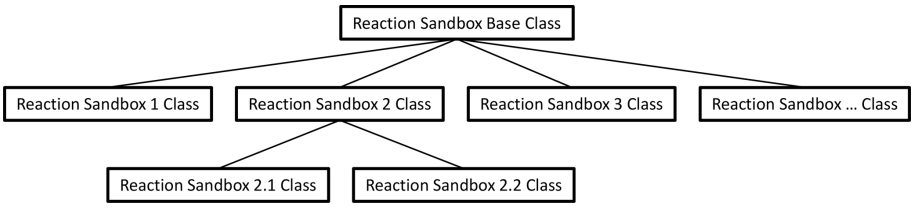

Figure 1. Schematic of a hypothetical Reaction Sandbox class hierarchy.

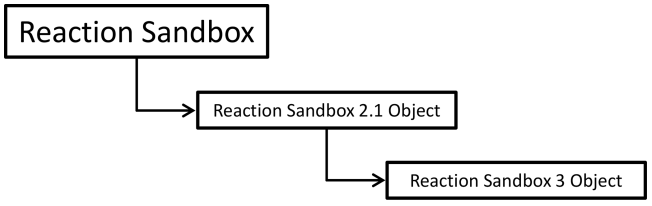

Figure 2. Schematic of a linked list of Reaction Sandboxes composed of two reaction objects instantiated from the class hierarchy shown in Fig. 1.

`RSandboxEvaluate` within subroutine `RReaction` within `reaction.F90`), but member procedures within the Reaction Sandbox classes may not be called (to preserve data encapsulation).

The following section describes several example Reaction Sandboxes. These examples are implemented within the PFLOTRAN source code and may serve as templates for future sandboxes.

4 Example reaction sandboxes

This section illustrates the implementation of a few Reaction Sandboxes within PFLOTRAN. Reaction Sandbox Biodegradation Hill and Reaction Sandbox Flexible Biodegradation Hill were developed to demonstrate the implementation of a microbially mediated biodegradation reaction. Reaction Sandbox Simple provides a set of preconfigured reactions that may be uncommented, compiled and run to better understand kinetic rate expressions. Reaction Sandbox Example implements a first-order decay reaction. The comments within `reaction_sandbox_example.F90` detail the steps necessary to implement a new Reaction Sandbox class.

4.1 Biodegradation

4.1.1 Biodegradation conceptual model

Consider a microbially mediated biodegradation reaction with biomass growth and decay over time. The reaction could be expressed as

$$A_{\mathrm{aq}} + 0.25B_{\mathrm{aq}} \rightarrow 0.33C_{\mathrm{aq}} + D_{\mathrm{aq}}, \quad \text{(R1)}$$

with electron donor A_{aq} and acceptor B_{aq} and products C_{aq} and D_{aq} [M]. The reaction is mediated by the immobile biomass species X_{im} [$\mathrm{mole_{biomass}\,m^{-3}_{bulk}}$] and inhibited above a C_{aq} concentration of 10^{-4}. The reaction rate I_{r} [$\mathrm{mole\,m^{-3}_{bulk}\,s^{-1}}$] can be calculated using Michaelis–Menten kinetics as

$$I_{\mathrm{r}} = k_{\max} X_{\mathrm{im}} \frac{A_{\mathrm{aq}}}{K_{A_{\mathrm{aq}}} + A_{\mathrm{aq}}} \times \frac{B_{\mathrm{aq}}}{K_{B_{\mathrm{aq}}} + B_{\mathrm{aq}}} \times \frac{I_{C_{\mathrm{aq}}}}{I_{C_{\mathrm{aq}}} + C_{\mathrm{aq}}} \quad (13)$$

with maximum specific utilization rate constant $k_{\max}$ [$\mathrm{mole\,mole^{-1}_{biomass}\,s^{-1}}$], half-saturation constants $K_{A_{\mathrm{aq}}}$ and $K_{B_{\mathrm{aq}}}$ [M], and inhibitor concentration $I_{C_{\mathrm{aq}}}$ [M]. The rate of biomass growth and decay can be modeled as

$$\frac{\mathrm{d}X}{\mathrm{d}t} = yield_{X_{\mathrm{im}}} I_{\mathrm{r}} - k_{\mathrm{decay}} X \quad (14)$$

with yield $yield_{X_{\mathrm{im}}}$ [$\mathrm{mole_{biomass}\,mole^{-1}}$] and decay rate constant k_{decay} [$\mathrm{s^{-1}}$]. These rate expressions are implemented as microbial and biomass decay reactions within PFLOTRAN and enabled through the `MICROBIAL_REACTION` and `IMMOBILE_DECAY_REACTION` keywords in the `CHEMISTRY` block of the input file.

Figure 3 shows PFLOTRAN simulation results for an example batch experiment run over 7 d employing the reactions in Eqs. (13) and (14), reaction parameters in Table 2, stoichiometries in Reaction R1, and initial conditions in Table 3. Results plotted in the figure may be replicated through the following commands.

```
cd $PFLOTRAN_DIR/regression_tests/
default/reaction_sandbox
$PFLOTRAN_DIR/src/pflotran/pflotran
 -input_prefix biodegradation
python biodegradation_vs_data.py
 biodegradation-obs-0.pft
```

The plot shows the time evolution of aqueous and immobile species concentrations as the week-long simulation runs with a maximum time step size of 1 h. The results demonstrate that electron donor A_{aq} is the limiting substrate, as the concentrations for acceptor B_{aq} and the reaction products

Code Block 1. Class `reaction_sandbox_base_type` in `reaction_sandbox_base.F90`.

```
type, abstract, public :: reaction_sandbox_base_type
  class(reaction_sandbox_base_type), pointer :: next
contains
  procedure, public :: ReadInput => BaseReadInput
  procedure, public :: Setup => BaseSetup
  procedure, public :: Evaluate => BaseEvaluate
  procedure, public :: UpdateKineticState => BaseUpdateKineticState
  procedure, public :: AuxiliaryPlotVariables => BaseAuxiliaryPlotVariables
  procedure, public :: Destroy => BaseDestroy
end type reaction_sandbox_base_type
```

Code Block 2. Abbreviated version of subroutine `RSandboxEvaluate` in `reaction_sandbox.F90`.

```
subroutine RSandboxEvaluate(...)
  ...
  class(reaction_sandbox_base_type), pointer :: cur_reaction

  cur_reaction => rxn_sandbox_list
  do
    if (.not.associated(cur_reaction)) exit
    call cur_reaction%Evaluate(...)
    cur_reaction => cur_reaction%next
  enddo

end subroutine RSandboxEvaluate
```

C_{aq} and D_{aq} plateau when A_{aq} is nearly depleted at approximately 4 d. Biomass concentration increases through 2.25 d at which time the first-order decay rate exceeds the growth rate, and the population begins to fade. Superimposed on the figure are hypothetical experimental results for species A_{aq} (shown as circles) that deviate from the (blue) simulated curve in the log-scale plot. In this example, Eq. (13) is somewhat inaccurate in predicting the tailing behavior of species A_{aq} at late times.

The discrepancy in A_{aq} concentration can be resolved by employing a Hill function (Hill, 1910) within the Monod expression for A_{aq}. Here, the A_{aq} concentration and corresponding half-saturation constant are raised to the power n.

$$I_r = k_{max} X_{im} \frac{A_{aq}^n}{K_{A_{aq}}^n + A_{aq}^n} \times \frac{B_{aq}}{K_{B_{aq}} + B_{aq}} \times \frac{I_{C_{aq}}}{I_{C_{aq}} + C_{aq}} \quad (15)$$

However, the Hill function is not an option in PFLOTRAN; the code must be altered to accommodate this new feature. To further complicate matters, the researcher cannot modify the existing Monod expression within PFLOTRAN, since in doing so, the Hill function would be applied to both the A_{aq} and B_{aq} Monod expressions. One possible solution is to implement Eq. (15) as a new reaction, and the Reaction Sandbox is designed to facilitate this process.

Table 2. Microbially mediated reaction parameters for the batch biodegradation experiment. n only applies to the reaction incorporating the Hill function (i.e., Eq. 15). M signifies molarity or mole per liter of water.

Parameter	Value	Units
k_{max}	9×10^{-2}	mole mole$_{biomass}^{-1}$ s^{-1}
$K_{A_{aq}}$	2×10^{-4}	M
$K_{B_{aq}}$	1.25×10^{-5}	M
$I_{C_{aq}}$	2.5×10^{-4}	M
$yield_{X_{im}}$	1×10^{-4}	mole biomass mole^{-1}
k_{decay}	$1 \times 1-^{-6}$	1 s^{-1}
n	1.2	–

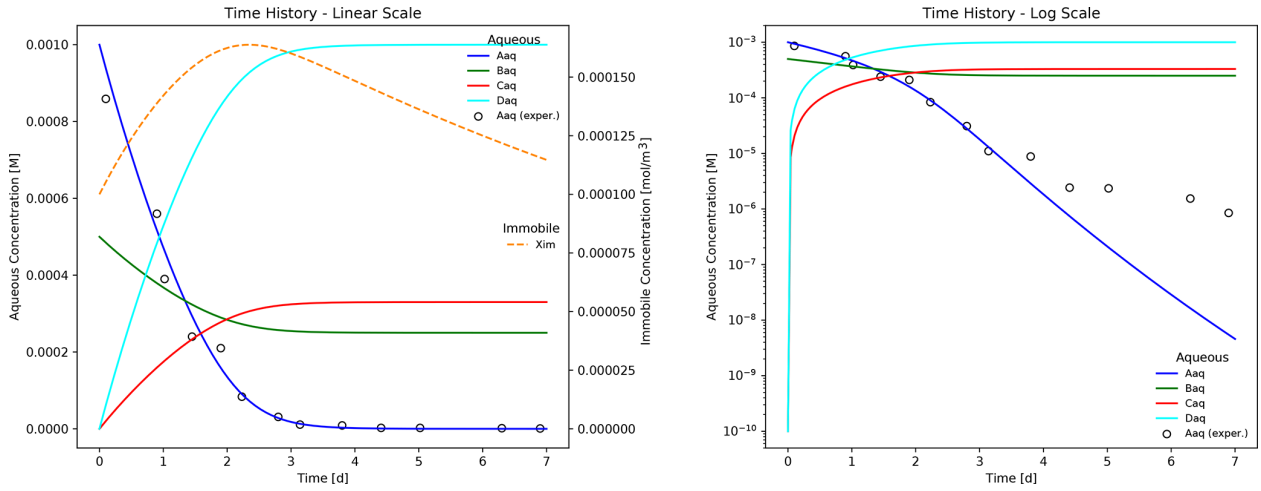

Figure 3. Time evolution of aqueous and immobile biomass species in a week-long batch biodegradation experiment.

Table 3. Initial concentrations for the batch biodegradation experiment.

Species	Concentration	Units
A_{aq}	1×10^{-3}	M
B_{aq}	5×10^{-4}	M
C_{aq}	1×10^{-10}	M
D_{aq}	1×10^{-10}	M
X_{im}	1×10^{-4}	mole m^{-3}_{bulk}

4.1.2 Reaction Sandbox Biodegradation Hill

Reaction Sandbox Biodegradation Hill implements the enhanced biodegradation reaction that incorporates the Hill function in Eq. (15) and the biomass growth and decay reaction in Eq. (14). The reactions are encoded in `reaction_sandbox_biohill.F90`. The `reaction_sandbox_biohill_type` class is presented in Code block 3, where integer IDs for each species are stored as class variables and the `Setup` and `Evaluate` procedures are redirected to local `BioHillXXX` implementations. Code block 4 illustrates the assignment of species IDs in an abbreviated version of `BioHillSetup`, based on keywords for aqueous species `Aaq`, `Baq`, `Caq` and `Daq` and immobile species `Xim` specified in the input file. `BioHillEvaluate` utilizes these IDs to access species concentrations and entries in the residual vector. Code block 5 presents an abbreviated version of the implementation of `BioHillEvaluate` where the naming convention for local variables corresponds closely to reaction parameters and state variables defined in Eqs. (14) and (15). Note that in the calculation of rate `I_r`, the concentration of `Aaq` and half-saturation constant `K_Aaq` are raised to the power `n`. Figure 4 illustrates a more accurate simulation result for the batch reaction experiment where the Hill function exponent n is set to 1.2. It is clear that the addition of the Hill function improves the model's ability to capture the tailing of species A_{aq} at low concentrations.

Results plotted in Fig. 4 may be replicated through the following commands:

```
cd $PFLOTRAN_DIR/regression_tests/
default/reaction_sandbox
$PFLOTRAN_DIR/src/pflotran/pflotran
 -input_prefix biodegradation_hill
python biodegradation_vs_data.py
 biodegradation_hill-obs-0.pft
```

The implementation of the enhanced biodegradation reaction in Reaction Sandbox Biodegradation Hill is somewhat rigid. All reaction parameters (rate constants, half-saturation constants, stoichiometries, etc.) are hardcoded within the source code as shown in Code block 5 and may not be changed without code modifications. However, the implementation may be generalized, and this is demonstrated in Reaction Sandbox Flexible Biodegradation Hill.

4.1.3 Reaction Sandbox Flexible Biodegradation Hill

Reaction Sandbox Flexible Biodegradation Hill employs the same biodegradation, growth and decay reactions as Reaction Sandbox Biodegradation Hill with the added flexibility of specifying reaction parameters at run time through the input file, eliminating the need to re-compile PFLOTRAN every time a parameter changes. The new class is implemented in `reaction_sandbox_flexbiohill.F90`. The `reaction_sandbox_flexbiohill_type` class extends `reaction_sandbox_biohill_type` as presented in Code block 6. The child class inherits the species integer IDs from the parent and adds member variables for storing all reaction parameters and a logical flag for specifying the units of half-saturation constants `K_Aaq` and `K_Baq`

Code Block 3. Class `reaction_sandbox_biohill_type` in `reaction_sandbox_biohill.F90`.

```
type, public, &
  extends(reaction_sandbox_base_type) :: reaction_sandbox_biohill_type
  ! Aqueous species
  PetscInt :: species_Aaq_id
  PetscInt :: species_Baq_id
  PetscInt :: species_Caq_id
  PetscInt :: species_Daq_id
  ! Immobile species (e.g. biomass)
  PetscInt :: species_Xim_id
contains
  procedure, public :: Setup => BioHillSetup
  procedure, public :: Evaluate => BioHillEvaluate
end type reaction_sandbox_biohill_type
```

Code Block 4. Subroutine `BioHillSetup` in `reaction_sandbox_biohill.F90`.

```
subroutine BioHillSetup(this,reaction,option)
  ...
  ! Aqueous species
  word = 'Aaq'
  this%species_Aaq_id = &
    GetPrimarySpeciesIDFromName(word,reaction,option)
  ...
  ! Immobile species
  word = 'Xim'
  this%species_Xim_id = &
    GetImmobileSpeciesIDFromName(word,reaction%immobile,option)

end subroutine BioHillSetup
```

and inhibitor concentration `I_Caq` (molality versus molarity). The dynamic `stoich` array enables the use of "do loops" in the `Evaluate` routine. The class redirects the `ReadInput`, `Setup`, `Evaluate` and `Destroy` procedures to local implementations, though only the implementation of `FlexBioHillEvaluate` will be described below. The `FlexBioHillEvaluate` routine shown in Code block 7 is more succinct than `BioHillEvaluate` (Code block 5). Reaction parameters are no longer hardwired but read from the input file in `FlexBioHillReadInput` and stored as class member variables. The use of the class member `stoich` array within the do loops eliminates the need to hardwire the `Residual` array indexing and reduces the number of lines of code. In addition, `FlexBioHillEvaluate` demonstrates the ability to choose between molality versus molarity for aqueous concentrations and half-saturation constants, a useful option since laboratory data are often available in either format. There is also support for calculating analytical derivatives (for the Jacobian), which can be programmed much more concisely with the `stoich` array and do loops (not shown). Flexible Biodegradation Hill produces results identical to Biodegradation Hill when given identical initial conditions and Biodegradation Hill's reaction parameters. These results may be compared by running the script `compare_biodegradation_results.py` after completing both simulations, as follows.

```
cd $PFLOTRAN_DIR/regression_tests/
default/reaction_sandbox
$PFLOTRAN_DIR/src/pflotran/pflotran
 -input_prefix biodegradation_hill
$PFLOTRAN_DIR/src/pflotran/pflotran
 -input_prefix
  flexible_biodegradation_hill
```

Code Block 5. Subroutine `BioHillEvaluate` in `reaction_sandbox_biohill.F90`.

```
subroutine BioHillEvaluate(this,Residual,Jacobian,compute_derivative, &
                           rt_auxvar,global_auxvar,material_auxvar, &
                           reaction,option)
  ...
  k_max = 9.d-2
  k_decay = 1.d-6
  K_Aaq = 2.d-4
  K_Baq = 1.25d-5
  I_Caq = 2.5d-4

  yield = 1.d-4
  n = 1.2d0

  stoichA = -1.d0
  stoichB = -0.25d0
  stoichC = 0.33d0
  stoichD = 1.d0

  I_r = k_max * Xim * Aaq**n / (K_Aaq**n + Aaq**n) * &
                      Baq / (K_Baq + Baq) * &
                      I_Caq / (I_Caq + Caq)

  I = I_r * volume
  RateA = stoichA * I
  ...
  RateX = yield * I - k_decay * Xim * volume
  ...
end subroutine BioHillEvaluate
```

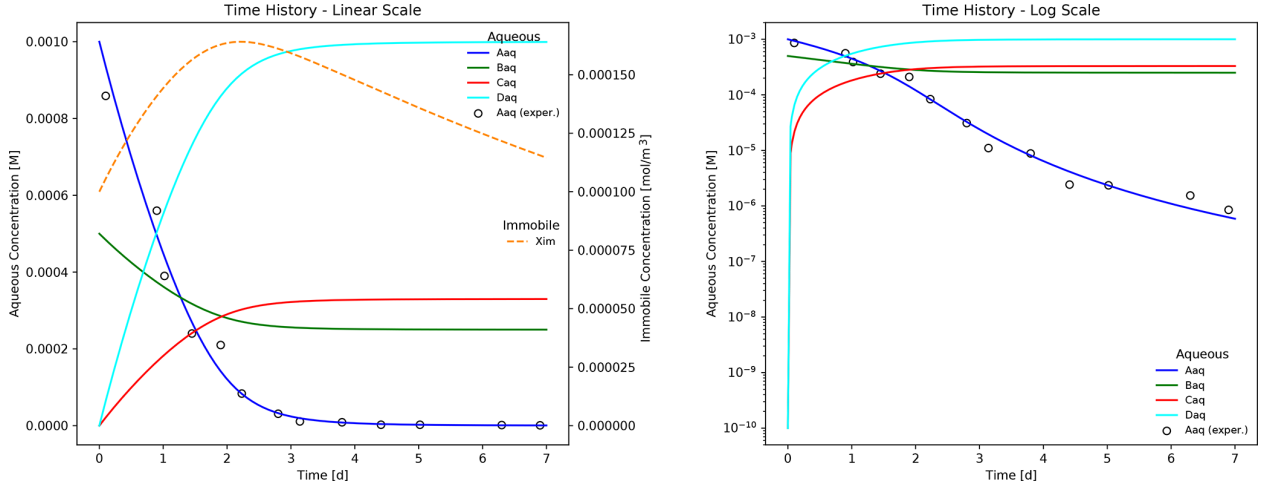

Figure 4. Improved match to the batch experiment results for species A_{aq} after incorporating the Hill function in Eq. (15). Compare with the log-scale plot (without the Hill function) in Fig. 3.

Code Block 6. reaction_sandbox_flexbiohill_type in reaction_sandbox_flexbiohill.F90.

```
type, public, &
  extends(reaction_sandbox_biohill_type) :: reaction_sandbox_flexbiohill_type
  PetscReal :: k_max
  PetscReal :: K_Aaq_n
  PetscReal :: K_Baq
  PetscReal :: I_Caq
  PetscReal :: yield
  PetscReal :: k_decay
  PetscReal :: n
  PetscBool :: molarity_units
  PetscReal, pointer :: stoich(:)
contains
  procedure, public :: ReadInput => FlexBioHillReadInput
  procedure, public :: Setup => FlexBioHillSetup
  procedure, public :: Evaluate => FlexBioHillEvaluate
  procedure, public :: Destroy => FlexBioHillDestroy
end type reaction_sandbox_flexbiohill_type
```

Code Block 7. Subroutine FlexBioHillEvaluate in reaction_sandbox_flexbiohill.F90.

```
subroutine FlexBioHillEvaluate(this,Residual,Jacobian,compute_derivative, &
                               rt_auxvar,global_auxvar,material_auxvar, &
                               reaction,option)
  ...
  I_r = this%k_max * Xim * Aaq**this%n / (this%K_Aaq_n + Aaq**this%n) * &
                           Baq / (this%K_Baq + Baq) * &
                           this%I_Caq / (this%I_Caq + Caq)
  I = I_r * volume
  ...
  do icomp = 1, reaction%ncomp
    Residual(icomp) = Residual(icomp) - this%stoich(icomp) * I
  enddo
  ...
  Residual(Xim_offset) = Residual(Xim_offset) + this%k_decay * Xim * volume
  ...
end subroutine FlexBioHillEvaluate
```

```
python compare_biodegradation
_results.py
```

4.2 Reaction Sandbox Simple

Reaction Sandbox Simple provides a framework for evaluating reactive transport in 1D using common kinetic rate expressions with a preconfigured set of six aqueous and two immobile species (A_{aq}, B_{aq}, C_{aq}, D_{aq}, E_{aq}, F_{aq}, X_{im}, Y_{im}). The conceptual model consists of a 100 m, liquid-saturated column with a Dirichlet boundary condition at the inlet ($x = 0$ m) and a zero-gradient boundary condition at the outlet ($x = 100$ m). Grid spacing is set to 1 m resolution. The prescribed Darcy velocity is $1\,\mathrm{m\,yr^{-1}}$ with a pore water velocity of $4\,\mathrm{m\,yr^{-1}}$ (porosity $= 0.25$). Throughout the simulation, solutes enter at the inlet and react within the domain. Simulation results are stored in two formats: (1) snapshots of the entire domain at select times and (2) continuous observation at

Table 4. Reaction Sandbox Simple conceptual model.

Parameter	Value
Column length	100 m
Cross-sectional area	1 m^2
Grid resolution	1 m
Prescribed Darcy velocity	1 m yr^{-1}
Pore water velocity	4 m yr^{-1}
Porosity	0.25
Tortuosity	1.0
Water density	1000 kg m^{-3}
Aqueous diffusion coefficient	10^{-9} m^2
Observation point location	49.5 m
Initial time step size	1 h
Maximum time step size	0.25 yr
Final simulation time	25 yr
Observation output frequency	Every time step
Snapshot file output times	0, 6.25, 12.5, 18.75, 25 yr

a mid-column observation point ($x = 49.5$ m). Table 4 summarizes simulation parameters, while Table 5 describes the initial and boundary concentrations.

The implementation of the `reaction_sandbox_simple_type` class is nearly identical to that of `reaction_sandbox_biohill_type` in Code block 3 with the addition of member variables `species_Eaq_id`, `species_Faq_id` and `species_Yim_id`. Member procedure `SimpleSetup` links these integer IDs to their respective names from the input file. The user may evaluate reactions in this sandbox with the following steps:

1. Uncomment a rate expression block within subroutine `SimpleEvaluate`.

```
cd $PFLOTRAN_DIR/src/pflotran
[emacs,gedit,nano,vi]
  reaction_sandbox_simple.F90
```

   Remove "`!uncomment:`" prefixes from lines within chosen rate expression in subroutine `SimpleEvaluate`.

   Save the file.

2. Compile the PFLOTRAN executable.

```
make pflotran
```

3. Navigate to `$PFLOTRAN_DIR/regression_tests/default/reaction_sandbox`.

```
cd $PFLOTRAN_DIR/regression_tests/
default/reaction_sandbox
```

4. Change the x-direction grid resolution in `reaction_sandbox_simple.in` to 100.

```
[emacs,gedit,nano,vi]
   reaction_sandbox_simple.in
NXYZ 10 1 1  →  NXYZ 100 1 1
```

   Save the file.

5. Run the simulation.

```
$PFLOTRAN_DIR/src/pflotran/pflotran
  -input_prefix reaction
  _sandbox_simple
```

6. Plot the results with `reaction_sandbox_simple.py`.

```
python reaction_sandbox_simple.py
```

Code block 8 illustrates a *commented* rate expression block for calculating the first-order decay of species `Aaq` to daughter product `Caq`. The user deletes all "`!uncomment:`" prefixes in the code block as shown in Code block 9, compiles PFLOTRAN, navigates to the `reaction_sandbox` folder and runs the code.

Figure 5 illustrates the simulation results. Plotted to the left is concentration breakthrough at the observation point and to the right is a snapshot of concentration profiles at 12.5 years. Species A_{aq} clearly decays into C_{aq} while the other species are transported without reaction. The user may evaluate the other rate expressions in the subroutine with the same approach.

4.3 Reaction Sandbox Example

PFLOTRAN's `reaction_sandbox_example.F90` serves as a template for implementing new Reaction Sandboxes. The sandbox implements the first-order decay of A_{aq} without daughter products, and the developer modifies the source code template to incorporate new reactions. Comments within the source code enumerate steps for implementing new reactions beginning with the renaming of subroutines and variables and ending with the implementation of subroutine `ExampleDestroy` at the bottom of the file. In between, source code is modified to implement the new reaction(s).

Code block 10 illustrates the first two steps embedded within comments in the source code near the top of the file. Comments within `ExampleEvaluate` provide a detailed description of the subroutine's arguments and local variables, including members of the reaction and reactive transport auxiliary variable classes (i.e., `reaction_rt_type` and `reactive_transport_auxvar_type`, respectively) which may be used in rate expression calculations. The units of all variables and the `Residual` and `Jacobian` arrays are also provided. Code block 11 shows several of these detailed comment blocks. Once refactoring is complete, the developer must modify the corresponding class and subroutine names within

Table 5. Initial and boundary concentrations in the Reaction Sandbox Simple input file.

Species	Initial concentration	Boundary concentration	Units
A_{aq}	1×10^{-10}	1×10^{-3}	M
B_{aq}	1×10^{-10}	1×10^{-3}	M
C_{aq}	1×10^{-10}	1×10^{-10}	M
D_{aq}	1×10^{-10}	1×10^{-10}	M
E_{aq}	1×10^{-10}	1×10^{-10}	M
F_{aq}	1×10^{-10}	1×10^{-10}	M
X_{im}	1×10^{-4}	n/a	mole m^{-3}_{bulk}
Y_{im}	1×10^{-10}	n/a	mole m^{-3}_{bulk}

n/a – not applicable

Code Block 8. Commented first-order rate expression block in `SimpleEvaluate` within `reaction_sandbox_simple.F90`.

```
subroutine SimpleEvaluate(...)
  ...
  ! first-order (A -> C)
  !uncomment: k = 1.d-9  ! [1/sec]
  !uncomment: stoichA = -1.d0
  !uncomment: stoichC = 1.d0
  !uncomment: Rate = k * Aaq * L_water  ! [mol/sec]
  !uncomment: RateA = stoichA * Rate
  !uncomment: RateC = stoichC * Rate
  ...
end subroutine SimpleEvaluate
```

`reaction_sandbox.F90` to match those in the newly refactored `reaction_sandbox_example.F90`. The source code may then be compiled without changing the `reaction_sandbox_example.F90` filename. Should this filename be revised, the developer must update the corresponding filenames within the `pflotran_object_files.txt` and `pflotran_dependencies.txt` files referenced by the makefile.

The best approach to compiling the code is to perform a `make clean` to remove all previously built modules and object files and the `pflotran` executable and then `make pflotran` to compile the code. `reaction_sandbox_example.in`, located in `$PFLOTRAN_DIR/regression_tests/default/reaction_sandbox`, provides a representative input deck for this example. Note that the card `EXAMPLE` in the `REACTION_SANDBOX` block of this input file must be updated to match the corresponding keyword added to `reaction_sandbox.F90`.

5 Conclusions

Customization of biogeochemical reaction networks is often necessary in the development of reactive transport simulators employed to simulate problem-specific scenarios in the real world. For researchers in the natural sciences, who may be more focused on biology, chemistry and/or physics than computational science, modifying these codes to incorporate new scientific processes can be challenging. The PFLOTRAN Reaction Sandbox may help remedy this issue.

The Reaction Sandbox provides a modular environment for prototyping new kinetic rate expressions that do not exist within PFLOTRAN. Within the Reaction Sandbox, novel reaction networks can evolve and mature over time; natural selection can run its course. Once vetted, these reactions may be incorporated more efficiently elsewhere within the permanent code base.

This work demonstrates the implementation of reactions within the Reaction Sandbox. Several new Reaction Sandbox classes are conceptualized and implemented based on existing rate expressions within PFLOTRAN for microbially mediated biodegradation to improve the code's ability to match hypothetical empirical data and provide greater flexibility from the end-user perspective. Reaction Sandbox Simple is

Code Block 9. Uncommented first-order rate expression block in `SimpleEvaluate` within `reaction_sandbox_simple.F90`.

```
subroutine SimpleEvaluate(...)
  ...
  ! first-order (A -> C)
  k = 1.d-9  ! [1/sec]
  stoichA = -1.d0
  stoichC = 1.d0
  Rate = k * Aaq * L_water  ! [mol/sec]
  RateA = stoichA * Rate
  RateC = stoichC * Rate
  ...
end subroutine SimpleEvaluate
```

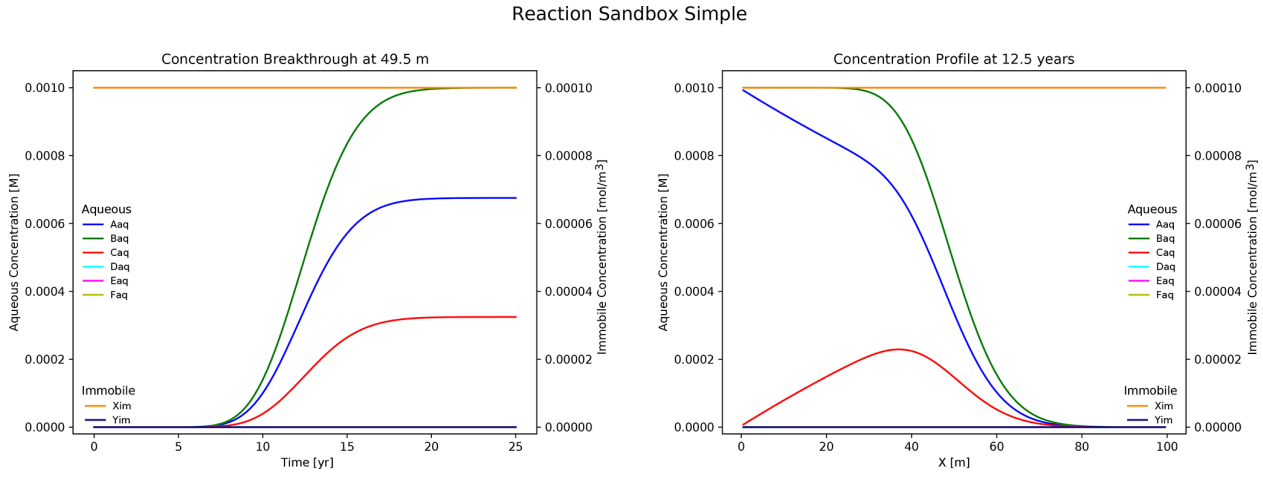

Figure 5. Results from Reaction Sandbox Simple with the first-order A → C rate expression uncommented.

presented as a means of prototyping numerous common kinetic rate expressions within a preconfigured column experiment context. Reaction Sandbox Example provides a template for implementing new reactions within PFLOTRAN.

Appendix A: Reaction sandbox member procedure interfaces

This section documents Fortran class `reaction_sandbox_base_type` member procedure interfaces not covered in Section 3.2.1 (see `reaction_sandbox_base.F90`).

A1 `ReadInput => BaseReadInput`

`ReadInput` (Code block 12) provides a customizable interface for reading parameters associated with the reaction sandbox from a block in the input file. The input block is opened by a unique keyword associated with the child Reaction Sandbox class.

`this` : `reaction_sandbox_base_type` object

`input` : object storing the input file object pointer and line buffers

`option` : object storing run time options including process rank and an error messaging buffer

A2 `Setup => BaseSetup`

`Setup` (Code block 13) initializes the Reaction Sandbox class by allocating dynamic memory, mapping species IDs, assigning stoichiometries and rate constants, etc. The degree to which Reaction Sandbox class settings are customizable at run time is up to the developer and their creativity in implementing `ReadInput` and `Setup`.

`this` : `reaction_sandbox_base_type` object

`reaction` : object storing chemical species and general reaction information

`option` : object storing run time options including process rank and an error messaging buffer

Code Block 10. Source code with embedded comments from the top of `reaction_sandbox_example.F90`.

```
module Reaction_Sandbox_Example_class

#include "petsc/finclude/petscsys.h"
  use petscsys

! 1. Change all references to "Example" as desired to rename the module and
!    subroutines within the module.

  use Reaction_Sandbox_Base_class

  use Global_Aux_module
  use Reactive_Transport_Aux_module

  use PFLOTRAN_Constants_module

  implicit none

  private

! 2. Add module variables here.  Note that one must use the PETSc data types
!    PetscInt, PetscReal, PetscBool to declare variables of type integer
!    float/real*8, and logical respectively.  E.g.,
!
! PetscReal, parameter :: formula_weight_of_water = 18.01534d0
...
```

A3 `Evaluate => BaseEvaluate`

`Evaluate` (Code block 14) calculates kinetic rates for all reactions in the Reaction Sandbox class and adds the (kinetic rate) contributions to the `Residual` and `Jacobian` arrays. This is the only subroutine that must be extended in child Reaction Sandbox classes.

`this` : `reaction_sandbox_base_type` object

`Residual` : 1D array of double precision numbers holding contributions to the residual equations at each grid cell

`Jacobian` : 2D array of double precision numbers holding contributions to the Jacobian matrix at each grid cell

`compute_derivative` : flag toggling on the calculation of analytical derivatives for the Jacobian matrix when true

`rt_auxvar` : object storing reactive transport state variables (e.g., concentrations, rates) at each grid cell

`global_auxvar` : object storing flow state variables (e.g., density, saturation) at each grid cell

`material_auxvar` : object storing material and cell properties (e.g., porosity, volume) at each grid cell

`reaction` : object storing chemical species and general reaction information

`option` : object storing run time options including process rank and an error messaging buffer

A4 `UpdateKineticState => BaseUpdateKineticState`

`UpdateKineticState` (Code block 15) updates state variables associated with the Reaction Sandbox class that are stored in `rt_auxvar` and updated at the end of a time step based on rates calculated in the Reaction Sandbox (e.g., for mass balance calculations, updates to mineral volume fractions).

`this` : `reaction_sandbox_base_type` object

Code Block 11. Representative comments from subroutine `ExampleEvaluate`.

```
  ...
  ! rt_auxvar - Object holding chemistry information (e.g., concentrations,
  !   activity coefficients, mineral volume fractions, etc.).  See
  !   reactive_transport_aux.F90.
  !
  !   Useful variables:
  !     rt_auxvar%total(:,iphase) - total component concentrations
  !                                 [mol/L water] for phase
  !     rt_auxvar%pri_molal(:) - free ion concentrations [mol/kg water]
  !     rt_auxvar%pri_act_coef(:) - activity coefficients for primary species
  !     rt_auxvar%aqueous%dtotal(:,iphase) - derivative of total component
  !                 concentration with respect to free ion [kg water/L water]
  ...
! 10. Add code for the Residual evaluation.

  ! Units of the Residual must be in moles/second.
  ! 1.d3 converts m^3 water -> L water
  L_water = material_auxvar%porosity*global_auxvar%sat(iphase)* &
            material_auxvar%volume*1.d3
  ! Always "subtract" the contribution from the Residual.
  Residual(this%species_id) = Residual(this%species_id) - &
    (-1.d0) * & ! negative stoichiometry
    this%rate_constant * &  ! 1/sec
    L_water * & ! L water
    rt_auxvar%total(this%species_id,iphase) ! mol/L water
  ...
```

Code Block 12. `BaseReadInput` argument list.

```
  subroutine BaseReadInput(this,input,option)
    ...
    class(reaction_sandbox_base_type) :: this
    type(input_type), pointer :: input
    type(option_type) :: option
```

Code Block 13. `BaseSetup` argument list.

```
  subroutine BaseSetup(this,reaction,option)
    ...
    class(reaction_sandbox_base_type) :: this
    class(reaction_rt_type) :: reaction
    type(option_type) :: option
```

Code Block 14. `BaseEvaluate` argument list.

```
subroutine BaseEvaluate(this,Residual,Jacobian,compute_derivative, &
                        rt_auxvar,global_auxvar,material_auxvar, &
                        reaction,option)
  ...
  class(reaction_sandbox_base_type) :: this
  class(reaction_rt_type) :: reaction
  ! the following arrays must be declared after reaction
  PetscReal :: Residual(reaction%ncomp)
  PetscReal :: Jacobian(reaction%ncomp,reaction%ncomp)
  PetscBool :: compute_derivative
  type(reactive_transport_auxvar_type) :: rt_auxvar
  type(global_auxvar_type) :: global_auxvar
  class(material_auxvar_type) :: material_auxvar
  type(option_type) :: option
```

Code Block 15. `BaseUpdateKineticState` argument list.

```
subroutine BaseUpdateKineticState(this,rt_auxvar,global_auxvar, &
                                  material_auxvar,reaction,option)
  ...
  class(reaction_sandbox_base_type) :: this
  type(reactive_transport_auxvar_type) :: rt_auxvar
  type(global_auxvar_type) :: global_auxvar
  class(material_auxvar_type) :: material_auxvar
  class(reaction_rt_type) :: reaction
  type(option_type) :: option
```

`rt_auxvar` : object storing reactive transport state variables (e.g., concentrations, rates) at each grid cell

`global_auxvar` : object storing flow state variables (e.g., density, saturation) at each grid cell

`material_auxvar` : object storing material and cell properties (e.g., porosity, volume) at each grid cell

`reaction` : object storing chemical species and general reaction information

`option` : object storing run time options including process rank and an error messaging buffer

A5 `AuxiliaryPlotVariables` => `BaseAuxiliaryPlotVariables`

`AuxiliaryPlotVariables` (Code block 16) appends Reaction Sandbox-specific state variables stored in `rt_auxvar` to the list of output variables to be printed to observation and snapshot files.

`this` : `reaction_sandbox_base_type` object

`list` : `output_variable_list_type` object storing a linked list of sandbox-specific `output_variable_type` objects

`reaction` : object storing chemical species and general reaction information

`option` : object storing run time options including process rank and an error messaging buffer

A6 `Destroy` => `BaseDestroy`

`Destroy` (Code block 17) deallocates all dynamic memory in the Reaction Sandbox class at the end of a simulation.

`this` : `reaction_sandbox_base_type` object

Code availability. The source code, input files and results presented in this manuscript are based on PFLOTRAN v4.0. A snapshot of this release is available at https://doi.org/10.5281/zenodo.5826289 (Hammond, 2022). The corresponding version of PETSc is v3.16.2 and was configured

Code Block 16. `BaseAuxiliaryPlotVariables` argument list.

```
subroutine BaseAuxiliaryPlotVariables(this,list,reaction,option)
  ...
  class(reaction_sandbox_base_type) :: this
  type(output_variable_list_type), pointer :: list
  class(reaction_rt_type) :: reaction
  type(option_type) :: option
```

Code Block 17. `BaseDestroy` argument list.

```
subroutine BaseDestroy(this)
  ...
  class(reaction_sandbox_base_type) :: this
```

on Ubuntu 18.04 with GCC 7.5 using the following config script:
`./configure -CFLAGS='-O3' -CXXFLAGS='-O3' -FFLAGS='-O3' -with-debugging=no -download-mpich=yes -download-hdf5=yes -download-hdf5-fortran-bindings=yes -download-fblaslapack=yes -download-metis=yes -download-parmetis=yes`.

Competing interests. The contact author has declared that there are no competing interests.

Disclaimer. Publisher's note: Copernicus Publications remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Acknowledgements. This research was supported by the U.S. Department of Energy (DOE), Office of Biological and Environmental Research (BER), as part of BER's Environmental System Science Program (ESS). This contribution originates from the River Corridor Scientific Focus Area (SFA) at the Pacific Northwest National Laboratory (PNNL) and was supported by the partnership with IDEAS-Watersheds. PNNL is operated for the DOE by Battelle Memorial Institute under contract DE-AC05-76RL01830. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

Financial support. This research has been supported by the Office of Science (grant no. DE-AC05-76RL01830).

Review statement. This paper was edited by Richard Mills and reviewed by Allan Leal and Bhavna Arora.

References

Andre, B., Bisht, G., Collier, N., Frederick, J., Hammond, G., Jaysaval, P., Karra, S., Kumar, J., Leone, R., Lichtner, P., Mills, R., Nole, M., Orsini, P., Park, H., and Rousseau, M.: PFLOTRAN Theory Guide, http://documentation.pflotran.org, last access: 23 February 2022.

Bethke, C. M.: Geochemical and Biogeochemical Reaction Modeling, Cambridge University Press, 2nd Edn., https://doi.org/10.1017/CBO9780511619670, 2007.

Chapman, S.: Fortran for Scientists & Engineers, McGraw-Hill Higher Education, 4th Edn., ISBN10: 0073385891, ISBN13: 9780073385891, 2018.

Hammond, G.: PFLOTRAN Reaction Sandbox, Zenodo [code], https://doi.org/10.5281/zenodo.5826289, 2022.

Hammond, G. E., Lichtner, P. C., and Mills, R. T.: Evaluating the performance of parallel subsurface simulators: An illustrative example with PFLOTRAN, Water Resour. Res., 50, 208–228, https://doi.org/10.1002/2012WR013483, 2014.

Hill, A. V.: The possible effects of the aggregation of the molecules of hæmoglobin on its dissociation curves, J. Physiol., 40, i–vii, 1910.

Lichtner, P. C.: Continuum model for simultaneous chemical reactions and mass transport in hydrothermal systems, Geochim. Cosmochim. Ac., 49, 779–800, https://doi.org/10.1016/0016-7037(85)90172-3, 1985.

Liu, C., Zachara, J. M., Qafoku, N. P., and Wang, Z.: Scale-dependent desorption of uranium from contaminated subsurface sediments, Water Resour. Res., 44, W08413, https://doi.org/10.1029/2007WR006478, 2008.

Rubin, J.: Transport of reacting solutes in porous media: Relation between mathematical nature of problem formulation and chemical nature of reactions, Water Resour. Res., 19, 1231–1252, https://doi.org/10.1029/WR019i005p01231, 1983.

Steefel, C., Appelo, C., Arora, B., Jacques, D., Kalbacher, T., Kolditz, O., Lagneau, V., Lichtner, P. C., Mayer, K. U., Meeussen, J. C. L., Molins, S., Moulton, D., Shao, H., Simunek, J., Spycher, N., Yabusaki, S. B., and Yeh, G. T.: Reactive transport codes for subsurface environmental simulation, Comput. Geosci., 19, 445–478, https://doi.org/10.1007/s10596-014-9443-x, 2015.

Steefel, C. I., DePaolo, D. J., and Lichtner, P. C.: Reactive transport modeling: An essential tool and a new research approach for the Earth sciences, Earth Planet. Sc. Lett., 240, 539–558, https://doi.org/10.1016/j.epsl.2005.09.017, 2005.

Tutolo, B. M., Luhmann, A. J., Tosca, N. J., and Seyfried Jr., W. E.: Serpentinization as a reactive transport process: The brucite silicification reaction, Earth Planet. Sc. Lett., 484, 385–395, https://doi.org/10.1016/j.epsl.2017.12.029, 2018.