SMIMfit Documentation

1 General Information

This document gives a high-level description of the functionality of the SMIMfit library, which allows a user to fit the parameters for a stochastic mobile-immobile model (SMIM), given breakthrough curve (BTC) data. The library is based on and adapted from methods originally presented by Kelly at al. [1] and Cortis and Berkowitz [2]. The cited references may be consulted to provide insight into some of the methods employed within SMIMfit. The primary advance provided by this library is the ability to include first-order reaction rates for the mobile and immobile zones, allowing the user to fit reactive breakthrough curves.

2 Library Architecture

Model fits are performed using the master function `SMIMfit()`, which is found in the `GeneralFittingCode` directory. This function specifies, by way of `createModel()`, the: (1) objective function to be minimized (currently only one is available) and (2) SMIM transport model used to create the model BTC (e.g., truncated power law (TPL)–currently the only choice). Tunable inputs for the fitting problem may be found in the `inputs` directory, and include:

- inputs to the optimization algorithm, found in `getOptInputs()`;
  - Currently, Matlab's nonlinear least-squares fitting function, `lsqnonlin()`, is employed.
- initial parameter guesses to be provided to the optimization algorithm, found in `initialGuessTPL()`;
  - These are initial guesses for the parameters that are to be fitted and may be altered based on pre-existing information about the system.
  - **Note:** initial guesses may also be altered by passing them to `SMMfit()`, via the optional input struct `'M'` within the fields:
    - `'M.params_guess'`
    - `'M.params_lower'`
    - `'M.params_upper'`
- tolerances for numerical methods, found in `getTolerance()`.
  - Currently, the inverse Laplace transform "de Hoog" algorithm and the Gauss-Laguerre quadrature algorithm call this function.

The transport model code is found in the `models` directory, and truncated power law (TPL) is currently the only available choice. The `utilities` directory contains other functions

and numerical methods–for example, the objective function, inverse Laplace transform, or numerical quadrature.

A file tree depicting the SMIMfit directory structure is shown in Figure 1, and pseudocode demonstrating the function hierarchy (via a call to the master `SMIMfit()` function) is given in Algorithm 1.

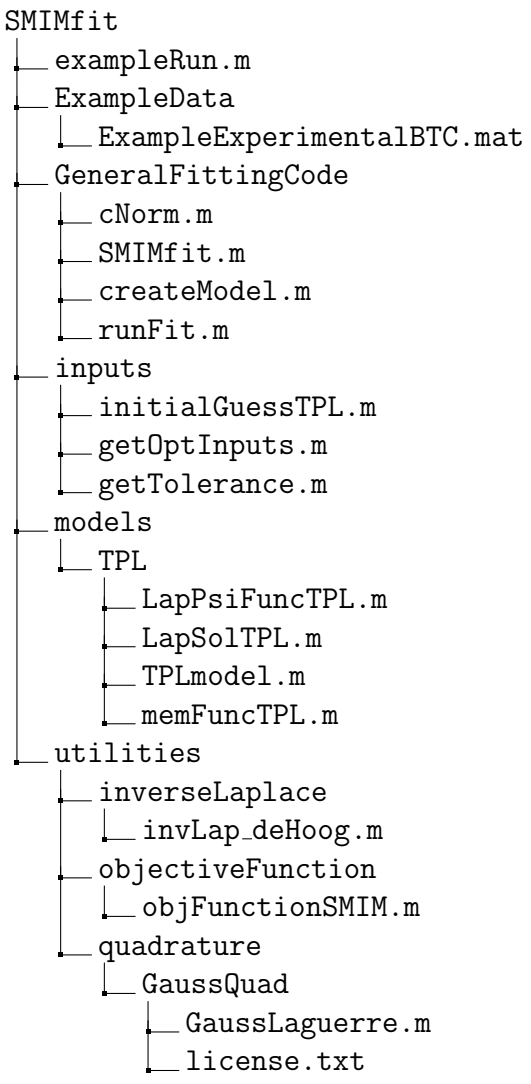

Figure 1: SMIMfit Directory Structure

2.1 Selected Functions

In order to use SMIMfit, the user need only interact with the high-level functions that drive the fitting model. A summary of these functions is given here. If the user wishes to alter or augment the functionality of SMIMfit, the headers of, and comments within, the functions in the library should provide sufficient details.

2.1.1 `SMIMfit()`

Note: SMIMfit requires `'c'` to be a normalized concentration vector (i.e., sums to 1), so an un-normalized concentration must first be normalized by calling `cNorm()` (see Section 2.1.2)

- **Inputs:**
  - `mymeas`: measurement (conservative or reactive)
  - `t`: time vector corresponding to concentration data
  - `c`: concentration vector (mg/L)
  - `varargin`: optional inputs:
    - **Note:** these must be specified as " 'name', value " pairs–e.g., `'model_type', 'TPL'` or `'L', 42`
    - `model_type`: Which model you're using. Only programmed for truncated power law at present time (default TPL).
    - `L`: Length of the reach where measures are being made (default 50 m).
    - `injectDuration`: duration of the injection. Exclude this name/value pair or set to zero if injection is a pulse.
    - `t_end`: latest time you wish to use in the model fit. Set this value when tailing behavior is unphysical due to, e.g., return to background conditions.
    - `M`: Output of a previous run (i.e., the struct output). Use to quickly specify guesses and limits of parameter fits (if non-default) or to supply modeled time series of conservative tracer when fitting a reactive time series.
- **Outputs:**
  - `M`: struct containing results of the fit, containing:
    - `params_fit`: fitted parameters (i.e., the results)
    - `opts`: fit options, including the parameter guesses and upper/lower bounds
    - `obj_fcn_result`: objective function residual vector from best fit
    - `SSE`: sum of squared errors from best fit
    - `fitdate`: time and date of final fit
    - `allfits`: parameter array containing fitted parameters for each fit
    - `obj_fcn_allfits`: norm of objective function residual for each fit
    - `algout`: struct containing output from the optimization function
    - `cXfit`: best fit concentration vector, where `'X'` is `'c'` or `'r'`, depending on whether this is for a conservative or reactive fit

    * `tXfit`: time vector corresponding to cXfit, where `'X'` is `'c'` or `'r'`, depending on whether this is for a conservative or reactive fit
    * `kfits`: only generated when performing reactive fit, contains structs for the type of reactive fit that is conducted (i.e., kboth, ksurf, ksub), and these structs will contain the above fields for the reactive fit

2.1.2 `cNorm()`

- **Inputs:**
  - `t`: time vector
  - `c`: vector of concentrations (mg/L)
  - `mymeas`: measurement (conservative or reactive)
  - `varargin`: optional inputs, based on available data and the specific measurement (conservative or reactive)
    * **Note:** these must be specified as " 'name', value " pairs.
    * `Q`: Discharge. If zero, discharge will be calculated from dilution gauging, assuming full mass recovery. Only add if discharge has been measured independently.
    * `cMass`: Input mass of conservative tracer (mg)
    * `rMass`: Input mass of reactive tracer (mg).
    * `fmc`: Fraction mass recovery of conservative tracer.
- **Outputs:**
  - `cnorm`: normalized concentration
  - `fm`: fraction of mass recovered (optional)
  - `Q`: discharge, estimated from dilution gauging (optional)
  - **Note:** `'cnorm'` is a required output, while `'fm'` and `'Q'` will only be output if the function is called with an output vector of length 2 or 3, respectively.

3 Basic Working Example

This example works through a fit of a combined NaCl/NO3 tracer experiment in a pea gravel stream. The stream discharge was not measured independently, so discharge is estimated via dilution gauging, using the included normalization function. This example is implemented in the included script `exampleRun.m`, and the following steps are executed:

1. **Add Library to Matlab Path**

- **Note:** This is done in the example script by simply adding the entire directory to the path (and later removing it in the same manner). Consider whether other options may be appropriate to your circumstances.

2. **Load the Example Data**

   - Load the dataset `'ExampleExperimentalBTC.mat'`. The dataset contains the following variables:
     - `tc_exp`: vector of measurement times for the conservative tracer
     - `cc_exp`: vector of conservative concentrations (mg/L)
     - `tr_exp`: vector of measurement times for the reactive tracer
     - `cr_exp`: vector of reactive concentrations (mg/L)
     - `xBTC`: sensor location downstream of injection point (48.5 m)
     - `mc`: mass of injected conservative tracer
     - `mr`: mass of injected reactive tracer

3. **Perform Conservative Fit**

   - **Normalize Measured Concentration Vector**
     - The SMIM model requires normalized concentrations as inputs, i.e., it expects that a conservative tracer should be fully recovered, and that mass recovery is unity. The `cNorm()` function will provide normalized conservative concentration (`'ccnorm'`), and discharge estimated from dilution gauging (`'Qdg'`). For example:

       ```
       [ccNorm, ~, Qdg] = cNorm(tc_exp, cc_exp, 'c', 'cMass', mc);
       ```

     - The first 3 inputs are always required. Any additional inputs (e.g., `'cMass'`, `'mc'`, in this case) should be " 'name', value " pairs, as is commonly supplied to built-in Matlab functions. In this case, the additional inputs direct the function to supply a non-default value to the function variable `'cMass'`, assigning it the value of the workspace variable `'mc'`.
       * Note that, if discharge has been measured experimentally, it should be supplied as an additional input for more accurate normalization (otherwise Q will be estimated.
       * In this case, the command

         ```
         [ccNorm, fmc] = cNorm(tc_exp, cc_exp, 'c', 'cMass', mc,
                                                    'Q', qexp);
         ```

         gives the percentage of the injected conservative tracer that is recovered in the BTC (`'fmc'`), where `'qexp'` is the experimentally-measured discharge.

- **Run the Conservative Fit**
  - Use the following command to run the conservative fit:

    ```
    Mcons = SMIMfit(tc_exp, ccNorm, 'c', 'L', xBTC)
    ```

  - The output struct `'Mcons'` stores all of the information related to the conservative model fit. Most important for the reactive model fit are the best fit parameter values are stored in field `'Mcons.params_fit'` and the model fit of the conservative BTC (`'Mcons.tcfit'`, `'Mcons.ccfit'`). This struct can be fed directly to the fitting function when performing the reactive fit, and the appropriate data from the conservative fit will be extracted.
  - **Note:** The optimization inputs are specified within `getOptInputs()`, and may be changed there, based on the needs of the user. As is, the tolerances and number of fit iterations are chosen to favor good visual fit in relatively quick run time.

4. **Perform Reactive Fit**

- **Note:** for both the normalization and fitting step, the reactive case requires outputs from the conservative case. For example, the conservative tracer data is used to measure the transport variables (e.g., velocity, exchange rate, etc.) and the reactive tracer data is used to measure the reaction rates. Thus, the above steps must be completed prior to the following.
- **Normalize Measured Concentration Vector**
  - Much like above, we normalize the concentration vector in the following way:

    ```
    [crNorm, fmr] = cNorm(tr_exp, cr_exp, 'r', 'rMass', mr, 'Q', Qdg);
    ```

  - If we calculated `'fmc'` during the conservative normalization, we may also supply it to `cNorm()`, as follows:

    ```
    [crNorm, fmr] = cNorm(tr_exp, cr_exp, 'r', 'rMass', mr,
                                          'Q', Qdg, 'fmc', fmc);
    ```

    Note that this is not necessary for the model fit, but it provides useful information. In this case, we observe only $\approx 18\%$ of the injected mass in the breakthrough curve.
- **Run the Reactive Fit**
  - Use the following command to run the reactive fit:

    ```
    Mreact = SMIMfit(tr_exp, crNorm, 'r', 'L', xBTC, 'M', Mcons)
    ```

  - **Note:** The best fit concentrations are *not* directly comparable to the experimentally measured concentrations. To do so, you would have to"unnormalize" them. This is left as an exercise for the user.

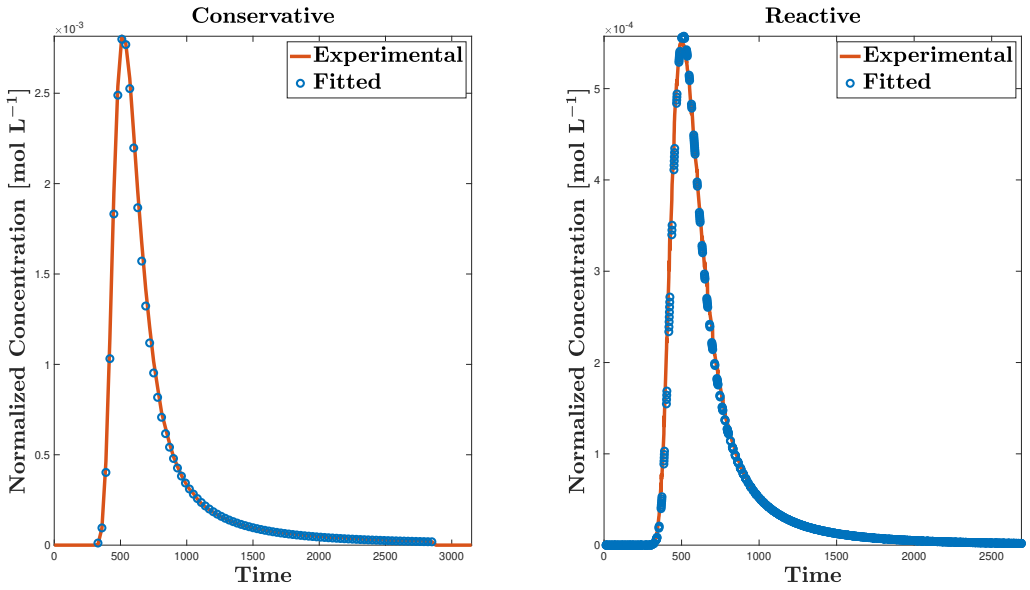

Figure 2: Results of conservative and reactive fits, by running `exampleRun.m` on the provided data, '`ExampleExperimentalBTC.mat`'

.

  - **Note:** The optimization inputs are specified within `getOptInputs()`, and may be changed there, based on the needs of the user. As is, the tolerances and number of fit iterations are chosen to favor good visual fit in relatively quick run time.

5. **Remove Library from Matlab Path**

   - **Note:** As above, this is done in the simplest way possible. Consider whether other options may be appropriate to your circumstances.

6. **Plot the Results**

   - The results of the conservative and reactive fits are depicted in Figure 2.

References

1. Kelly, J. F., Bolster, D., Meerschaert, M. M., Drummond, J. D. & Packman, A. I. FracFit: A robust parameter estimation tool for fractional calculus models. *Water Resources Research* **53,** 2559–2567. https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2016WR019748 (2017).

2. Cortis, A. & Berkowitz, B. Computing "Anomalous" Contaminant Transport in Porous Media: The CTRW MATLAB Toolbox. *Groundwater* **43,** 947–950. https://ngwa.onlinelibrary.wiley.com/doi/abs/10.1111/j.1745-6584.2005.00045.x (2005).

Algorithm 1: `SMIMfit()` function workflow

Input: Measured time and concentration vectors for breakthrough curve.
Output: Struct containing fitted parameters and run data.

```
Function initialGuessTPL():
    ▷ Set initial parameter guesses
end
Function createModel():
    ▷ Specify objective function and transport model
end
Function runFit():
    Function getOptInputs():
        ▷ Set options of optimization function
    end
    for i = 1 to numFitIterations do
        Function lsqnonlin():
            ▷ Matlab's nonlinear least-squares minimization function
            Function objFunctionSMIM():
                ▷ Objective function, defined within createModel()
                Function TPLmodel():
                    ▷ Transport model, defined within createModel()
                    ▷ Employs the following TPL-specific functions
                    Function LapSolTPL():
                        Function memFuncTPL():
                            Function LapPsiFuncTPL():
                            end
                        end
                    end
                end
            end
        end
    end
end
```