MEDIALAB

(Modeling Early Diagenesis using MATLAB)

User Manual

Model version: 1.1

date: 01.02.2022

Main author: Babak Shafei

Revised by: Martin Schmid

Eawag, Kastanienbaum

Table of content

1 Overview

MEDIALAB (**M**odeling **E**arly **DIA**genesis using MAT**LAB**) is an early diagenesis model, which calculates concentrations and fluxes of chemical species as well as rates of all the biogeochemical pathways at each depth of aquatic sediments for a specific time period. It is based on the model MATSEDLAB, which had been created by B. Shafei as part of his Ph.D. thesis project at the Georgia Institute of Technology under the supervision of P. Van Cappellen (Shafei, 2012). The code was further improved and subsequently applied to simulate early diagenesis in a lake with variable boundary conditions by Steinsberger et al. (2019). This manual refers to the version of the code used in that study.

A system of partial differential equations corresponding to early diagenesis equations are automatically generated through MATLAB's symbolic programming capabilities and solved using MATLAB's built-in solver *pdepe* to evaluate the temporal and spatial distribution of chemical species.

MEDIALAB is executed through the MATLAB home screen. The execution of MEDIALAB requires an active installation of MATLAB (Version 7.6 release R2008a or later). It is recommended to allow at least 1GB of space on the hard drive for the model output and 1GB of contiguous random-access memory for the initialization routine.

The code as well as an example application is freely available on GitHub at:

https://github.com/Eawag-AppliedSystemAnalysis/Medialab

2 Theoretical model

2.1 Diagenesis equation

The general, one-dimensional continuum representation of coupled mass transport and biogeochemical reaction in aquatic sediments is expressed by a set of partial differential equations (PDEs) of the form (Aguilera et al., 2005):

$$\frac{\partial(\varepsilon C_i)}{\partial t} = \left[\frac{\partial}{\partial x}\left(D\varepsilon\frac{\partial(C_i)}{\partial x}\right) - \frac{\partial}{\partial x}(\vartheta\varepsilon C_i)\right] + \sum \varepsilon r_i(x, t, C_i, \dots) \quad (1)$$

where C_i is the concentration of constituent i (solid-bound and solute concentrations are expressed in mass per g solid and mass per L porewater, respectively), x is the position along the 1-D vertical domain, with x = 0 corresponding to the sediment-water interface (SWI), and t denotes time. In Eq. (1), $\sum \varepsilon r_i(x, t, C_i, \dots)$ is the sum of the sources and sinks of species i; it includes the rates of all the (bio)geochemical reactions producing or consuming species i, as well as the non-local transport processes that add or remove the dissolved species, most notably through the irrigation of macrofaunal burrows. In a multicomponent reaction network containing N_S species and N_r reactions, Eq. (1) is solved for each species while various chemical species are coupled to one another via N_r reaction rate expressions appearing in $\sum \varepsilon r_i(x, t, C_i, \dots)$.

The variables ε, D and ϑ take different meanings depending on whether the given constituent is solid-bound or dissolved (Table 1).

Table 1: Meaning of the generalized variables in Eq. (1) for solids and solutes in aquatic sediments

Variable	Solids	Solutes
ε	$1-\varphi$	φ
D	D_b	$D_b + D_m/(1-\ln(\varphi^2))$
ϑ	ω	$\omega + V$

φ [-]: porosity; V [LT^{-1}]: externally imposed flow velocity; ω [LT^{-1}]: burial velocity defined with respect to the sediment water interface (SWI); D_b [L^{2}T^{-1}]: bioturbation coefficient; D_m [L^{2}T^{-1}]: molecular diffusion coefficient.

The bioturbation coefficient is depth-dependent as described by Steinsberger et al. (2019):

$$D_b = D_b^0 \frac{1-\tanh\left(\frac{x-H}{\tau_b}\right)}{1-\tanh\left(-\frac{H}{\tau_b}\right)} + D_{bmin} \quad (2)$$

Here, D_b^0 is the bioturbation coefficient at the SWI, τ_b is a characteristic length scale, H is the depth at which the steepest gradient of D_b occurs in the sediment, and D_{bmin} is a small value to avoid numerical problems that can occur if $D_b = 0$.

One major capability of MATLAB is to operate on matrices and arrays. Hence, to implement Eq. (1) in MATLAB, it is practical to convert it to a vector form as follows:

$$\frac{\partial(\varepsilon\vec{C})}{\partial t} = \frac{\partial(\varepsilon\vec{F})}{\partial x} + \varepsilon\vec{S} \quad (3)$$

where $\vec{C}$ is the vector of concentrations of species, $\vec{F}$ is the transport vector encompassing burial and bioturbation terms with molecular diffusion only applied for the solutes:

$$\vec{F} = D\frac{\partial\vec{C}}{\partial x} - \vartheta\vec{C} \quad (4)$$

All biogeochemical reaction rates are embedded in the vector $\vec{S}$ where the rate of the production and consumption of each species is a function of reaction network stoichiometrics and associated reaction rates. It is defined in a matrix notation as follows:

$$\vec{S} = \boldsymbol{\mu} \cdot \vec{R} \quad (5)$$

In this equation $\vec{S}$ (size N_S) is the vector of the components of $\sum r_i(x,t,C_i,\dots)$, defining the rate of change of the concentration of each constituent, $\boldsymbol{\mu}$ (size N_S x N_r) is the matrix of the stoichiometric coefficients, and $\vec{R}$ (size N_r) is the vector containing the rates of each reaction. In a fully kinetic description of the set of the biogeochemical reaction rates, equilibrium reactions are treated by fast kinetic rate constants.

2.2 Boundary conditions

For solutes, the upper boundary condition at x = 0 is the concentration of the solute in the bottom water at the sediment-water interface. In non-steady state simulations this concentration can be time-dependent reflecting e.g. annual or seasonal variations. In vector form we have:

$$\vec{C}(0,t) = \vec{B}(t) \quad (6)$$

where $\vec{B}(t)$ is vector of solute concentrations at the SWI.

For solid-bound species, the flux continuity condition is used at x = 0:

$$D_b \frac{\partial \vec{C}(0,t)}{\partial x} - w\vec{C}(0,t) = \frac{\vec{J}(t)}{\rho_b(1-\varphi)} \quad (7)$$

where $\vec{J}(t)$ is the vector of the depositional fluxes of the given solid-bound species (time-dependent in transient simulations), while w and D_b are the sedimentation rate and the bioturbation coefficient, respectively. The denominator, $\rho_b(1-\varphi)$ ensures consistency among the units of $\vec{J}(t)$ and $\vec{C}$. In the example application on GitHub, the units used are µmol cm^{-2} yr^{-1} for $\vec{J}$, µmol g^{-1} for solid species ($\vec{C}$), and cm yr^{-1} for w and D_b; ρ_b is the dry sediment density (g cm^{-3}) and φ is the sediment porosity.

As lower boundary condition, zero gradients are imposed for all solute and solid-bound species:

$$\frac{\partial \vec{C}(x_L,t)}{\partial x} = 0 \quad (8)$$

3 Implementation in Matlab

3.1 *pdepe* function

The *pdepe* function is designed to solve initial-boundary value problems (IBVPs) consisting of systems of parabolic and elliptic PDEs in one space variable and time. The numerical method is based on a piecewise nonlinear Petrov–Garlekin method with second-order accuracy. The method solves the ordinary differential equations (ODEs) resulting from the spatial discretization of the PDEs, using a built-in MATLAB ODE solver to obtain approximate solutions at specified times within a defined time interval.

The general form of PDE that is solved by *pdepe* in MATLAB within the temporal ($t_0 < t < t_f$) and spatial ($x_0 < x < x_L$) domains is:

$$c(x,t,u,\frac{\partial u}{\partial t})\frac{du}{dt} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right) \quad (9)$$

in which the vector u contains all the unknown variables (here the solid and solute concentrations, i.e. $\vec{C}$). Coupling of the partial derivatives with respect to time is restricted to multiplication by the matrix c. On the right hand side of Eq. (9), m is a parameter corresponding to the symmetry of the problem and can be 0 for slab, 1 for cylindrical, or 2 for spherical. In the case of the sediment diagenesis model, m = 0.

The functions f and s – the flux and sink/source terms, respectively – are vector functions, which depend on depth (x), time (t), and concentrations (u) and their gradients. Comparing the diagenesis Eq. (1) with Eq. (9), f corresponds to the vector $\vec{F}$ as:

$$f = \varepsilon\vec{F} \quad (10)$$

It represents the transport of chemical constituents and encompasses the advective and the diffusive transport of solutes and sediment mixing by benthic organisms. The minus results from fluxes being defined positive upward. Similarly, the term s and the vector $\vec{S}$ are related by:

$$s = \varepsilon\vec{S} \quad (11)$$

They account for the net production or consumption of all the chemical species by (bio)geochemical reactions and, for solutes, also bioirrigation. In order to solve the parabolic PDE of Eq. (9), initial and boundary conditions must be imposed. The initial condition at $t = t_0$ and all depths x has the form of:

$$u(x, t_0) = u_0(x) \tag{12}$$

Eq. (12) returns initial values of all the chemical species at depth x in the vector u. The user may thus specify any set of initial depth profiles, including spatially heterogeneous distribution. General boundary conditions at $x = x_0$ and $x = x_L$ are defined using the equation:

$$p(x, t, u) + q(x, t) f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0 \tag{13}$$

where f is the transport vector from Eq. (9).

At first sight Eq. (13) may not resemble common diagenesis boundary conditions of Eq. (6)-Eq. (8). However, commonly-used formulations for boundary conditions, such as Dirichlet, Neumann and Cauchy/Robin are embedded in Eq. (13) and for each type, the coefficients $p(x, t, u)$ and $q(x, t)$ take different values. The boundary conditions from Eq. (6) and Eq. (8) can be implemented as follows

1) Upper boundary condition for solutes

p and q in Eq. (13) must be chosen such that Eq (6) results.

$$p + q\varepsilon\vec{F} = 0 \Rightarrow p + q\left[\varphi\left(D_b + \frac{D_m}{1 - ln(\varphi^2)}\right)\frac{\partial \vec{C}}{\partial x} - \varphi w\vec{C}\right] = 0 \tag{14}$$

This is the case if:

$$p = u_L - \vec{B}(t) \tag{15}$$

$$q = 0 \tag{16}$$

where $\mathrm{u_L}$ is the approximate solution of pdepe solver for $\vec{C}$ at $x = 0$.

2) Lower boundary condition for solutes

p and q in Eq. (13) must be chosen such that Eq (8) results. Based on Eq. (14), this is the case if:

$$p = \varphi w u_R \tag{17}$$

$$q = 1 \tag{18}$$

where $\mathrm{u_R}$ is the approximate solution of pdepe solver for $\vec{C}$ at $x = x_L$.

3) Upper boundary condition for solids

p and q in Eq. (13) must be chosen such that Eq (7) results.

$$p + q\varepsilon\vec{F} = 0 \Rightarrow p + q\left[(1 - \varphi)\rho_b D_b \frac{\partial \vec{C}}{\partial x} - (1 - \varphi)\rho_b w\vec{C}\right] = 0 \tag{19}$$

This is the case, if:

$$p = \vec{J}(t) \tag{20}$$

$$q = 1 \tag{21}$$

4) Lower boundary condition for solids

In analogy to the lower boundary condition for solutes:

$$p = (1 - \varphi)\rho_b w u_R \quad (22)$$

$$q = 0 \quad (23)$$

The boundary conditions can depend on t, which means that MEDIALAB evaluates the boundary values at each time step. This gives the user the possibility to impose transient boundary conditions. The latter is particularly useful when simulating the fate of compounds whose inputs are changing due to, for example, anthropogenic activity, or when dealing with systems where the bottom-water chemistry varies over time.

3.2 MATLAB code structure

The pdepe function is run with the following command:

```
sol = pdepe(m,pdeFun,icFun,bcFun,x,t)
```

There are 6 main input arguments that must be parsed to the pdepe function. They are :

1) Parameter m which equals to zero in a 1-D diagenesis problem.

2) `pdeFun` is a function that computes the components of the PDE i.e. c,f,s. It has the following format:

```
[c,f,s] = pdeFun(x,t,u,dudx)
```

The input arguments are :

- x: vector of spatial domain
- t: vector of temporal domain
- u: vector of unknown concentrations
- dudx: vector of approximate partial derivative with respect to x or concentration gradients.

3) `icFun` is the initial condition function which evaluates spatially dependent initial concentration of species and returns the values in a vector u. It has the form:

```
u = icFun(x)
```

where the vector x of spatial domain is the only input argument.

4) `bcFun` is the function that evaluates the terms p and q of the boundary conditions in Eq. (13). It has the form

```
[pl,ql,pr,qr] = bcFun(xl,ul,xr,ur,t)
```

ul is the approximate solution at the upper boundary xl and ur is the approximate solution at the lower boundary xr. pl and ql are column vectors corresponding to p and q evaluated at xl, similarly pr and qr correspond to xr. Therefore, 4 vectors of pl, ql, pr and qr, each with Ns elements, are computed by this function.

The ultimate output of the *pdepe* function returns values of the concentrations of the constituents as a multidimensional matrix, `sol`, on a mesh defined by the x and t vectors. The concentration of the ith species at the grid point k and time step j can be extracted as:

```
ujki = sol(tspan(j), xmesh(k), i)
```

3.3 Symbolic Programming

One of the advantages of MATLAB is its symbolic programming capability, which is applied in the MEDIALAB model to automatically construct the reaction vector s. This is achieved through extracting the coefficients of the chemical species in each reaction to build the stoichiometric matrix **μ**. For this purpose, each biogeochemical pathway is assumed to be an algebraic polynomial equation with the products of the reaction all gathered with the reactants on the left side of the reaction with a negative sign. Chemical species are defined as symbols in MATLAB, which facilitates algebraic operations that are required to set up the stoichiometric matrix.

To clarify the process, let's assume nitrification as one of the reactions in the proposed conceptual model reaction network:

$$NH_4^+ + 2\ O_2 + 2\ HCO_3^- \rightarrow NO_3^- + 2\ CO_2 + 3\ H_2O$$

There are 6 species involved in this reaction, all required to be defined as symbols. The corresponding command in MATLAB is as follows:

```
syms nh4 o2 hco3 no3 co2 h2o
```

The symbolic names associated with each species are arbitrary. However, it is handy if they are chosen in a way that can be recognizable in the script. Then, the reaction is converted to a polynomial by rewriting the reaction as below using the symbols defined above:

```
R = nh4 + 2*o2 + 2*hco3 - no3 - 2*co2 - 3*h2o
```

To extract the number of moles of each species that are produced or consumed in this reaction, the following MATLAB command is used. For example, 1 mole of ammonium in this reaction is consumed, thus it has a coefficient of 1 which is attained through following command:

```
c = coeffs(R,nh4)
```

By applying two consecutive 'for' loops over all reactions and species, the stoichiometric matrix, **μ** is obtained, which is saved as the variable `stoichiometrix`. The number of moles of species i that are produced or consumed in reaction j can then be obtained by calling `stoichiometrix (i,j)`. Positive and negative signs are associated with production and consumption, respectively. If reaction j doesn't include species i, then `stoichiometrix(i,j)` equals to zero.

4 Structure of MEDIALAB

4.1 Overview

The main code of MEDIALAB is included in the following .m files:

- mainMEDIALAB.m
- userMEDIALAB.m
- autoMEDIALAB.m
- postprocessMEDIALAB.m

In addition there is a text file, inputMEDIALAB.txt, that includes all the required parameters incorporated in the conceptual model, as a matlab file *iCvalues.mat*, which contains the initial conditions, and a range of .m files that are used for plotting the output. In case of availability of measured field data, most notably sediment profile data, there is also an Excel spreadsheet with

measured field data as a function of depth. The files are listed in *Table 2* and the flowchart in *Figure 1* shows how the individual files are linked. The main files are described in more detail in the following sections.

Table 2: Files included in MEDIALAB package. The files with "Yes" in the "User modified" column need to be modifed by the user to define the system, provide data, and define which plots are created as output. The other files can of course also be modified, e.g. for adapting plots.

File	Type	User modified	Input arguments	Output arguments
mainMEDIALAB.m	MATLAB	No		`simValues, depth, time`
userMEDIALAB.m	MATLAB	Yes		`x, t, speciesName, inputFile, r, R, isSolidReaction, rName, massBalanceElements`
autoMEDIALAB.m	MATLAB	No	`speciesName, inputFile, r, R, reactionUnits`	`isSolidSpecies, stoichiometrix, advection, diffusion, reaction, plBC, prBC, parName, parValue`
plotOptionsMEDIALAB.m	MATLAB	Yes		`pOptions`
postprocessMEDIALAB.m	MATLAB	No	`resultfile`	
plotXXX.m (several files)	MATLAB	No		
inputMEDIALAB.txt	Text	Yes		
iCvalues.mat	MATLAB	Yes		
fieldData.xlsx	Excel	Yes		

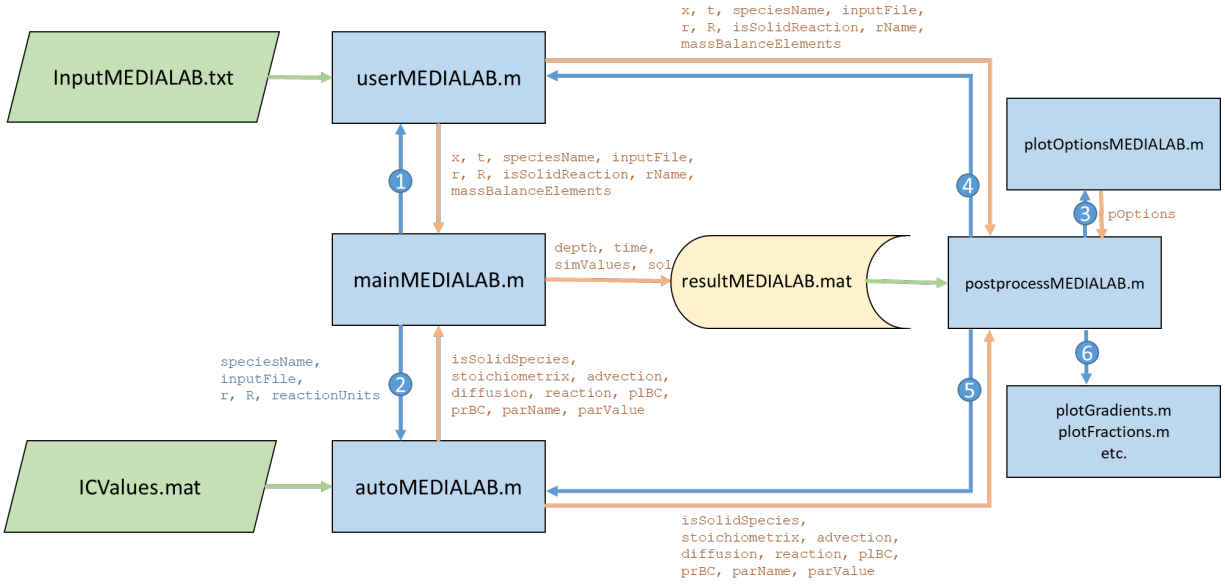

Figure 1: Flowchart of MEDIALAB. Blue rectangles display the .m scripts. The blue arrows indicate function calls from one .m file to another .m file, in the sequence indicated by the numbers on the blue dots. Orange arrows indicate the function outputs that are returned to the calling function. Green parallelograms indicate the input files with the model parameters, boundary conditions and initial conditions provided by the user. The results of the simulation are stored in a .mat file, which is then used to plot the results using the diverse available plotting functions.

4.2 userMEDIALAB

userMEDIALAB.m is the first script that has to be modified by the user to provide all the necessary information of spatial and temporal domains as well as the biogeochemical reaction network. There are some other parameters such as list of species names, organic matter composition coefficients and the names of the input file and the file with the initial conditions that have to be included in the script. Species are characterized by string variables and their assigned names contain either '(s)' or '(aq)' representing solid or aqueous species, respectively. This is an approach to characterize the phase of the species in the script when building the transport matrix (as molecular diffusion is excluded in case of solid species) and the reaction matrix (to ensure consistency between the units of reactions and species). As described above, variable names used for the chemical species are defined as symbols to set up the reaction network. In general for an irreversible reaction:

$$aA + bB \rightarrow cC + dD \quad (24)$$

species A, B, C and D have to be defined as symbols, and the reaction is rewritten as:

```
aA + bB - cC - dD
```

If the reaction is reversible, there will be two algebraic polynomials representing each direction of the reaction:

```
aA + bB - cC - dD
```

and

```
cC + dD - aA - bB
```

Three attributes have to be provided along with every reaction R: the reaction rate `r`, a flag which specifies the unit of the reaction `isSolidReaction` and the reaction name `rname`, which specifies the name of the reaction for plotting purposes.

The reaction rate is a string variable containing reaction rate constants and concentrations of species upon which rates are dependent on. For example if a bimolecular reaction rate is used to define the kinetics of the reaction (24), then `r` equals to:

```
r = 'k_AB * A * B'
```

`k_AB` is the reaction rate constant and provided through *inputMEDIALAB.txt* as an input parameter. In a similar manner, all other parameters appearing in the reaction equations must be included in the input file with the exact same name. `isSolidReaction` is a vector of binary values 1 and 0. If the unit of the reaction rate is the same as that of solid phases, i.e. µmol/g, then `isSolidReaction` equals to 1 otherwise it is 0. This flag variable ensures the consistency between species and reaction units.

The user also needs to provide the MATLAB .mat file that contains the initial conditions to the variable `ICFile`. The file has to include a vector `xic` providing the spatial grid of the initial conditions and a cell array `icValues` with the concentrations at the depths specified in `xic`. If the initial conditions should all be set to zero, `ICFile` can be set to an empty string.

4.3 inputMEDIALAB

inputMEDIALAB.txt is a text file containing the the input parameters provided by the user. It consists of parameters of transport, reaction rate constants, diffusion and bioturbation coefficients as well as

boundary conditions. The input file has two columns: parameters names and their values separated by space. Reaction rate constants must have exactly the same name as used in the *userMEDIALAB.m* script. Molecular diffusion coefficients and boundary conditions are named as 'Dmol_' and 'BC_' added with species name, respectively. For example, molecular diffusion coefficients and boundary condition of ammonium with the species name nh4 are Dmol_nh4 and BC_nh4, respectively. The number of molecular diffusion coefficients and boundary conditions must match with the number of dissolved species and total number of species, respectively.

Boundary conditions can be made time dependent in two ways. The variable BC_ can be provided as a function of the time variable t. Or a time series of scaling factors for BC_ can be provided following the value of BC_ in the format [time1,factor1;time2,factor2;...]. If such a time series is provided, the value of BC_ will be multiplied with the factors given in the time series, interpolated between the given time points. Note that depending on the functions used, time-variable boundary conditions can significantly prolong the simulation time or cause instabilities.

Other parameters that are included in *inputMEDIALAB.txt* are: the bioturbation coefficient D_bio, the sedimentation rate w, sediment dry density rho, and porosity. Since the concentration of H^+ is needed for the calculation of the saturation index of iron sulfide and vivianite, it is also imposed under variable name ph and has the value of 10^{-pH} stated in mM units.

4.4 autoMEDIALAB

autoMEDIALAB.m automatically creates the transport, reaction and boundary condition vectors in form of MATLAB function handles and passes them to *mainMEDIALAB.m*. A function handle is a callable association to a MATLAB function. It enables the user to pass a function to another function.

In the first block of the script, the input parameters are read from the *inputMEDIALAB.txt* file and function handles are created for time-dependent parameters. The names and function handles of the parameters are finally stored in the output variables ParName and parValue. The parName elements will be substituted with the corresponding parValue handles in advection, diffusion, reaction rate and boundary condition terms.

In the 2nd block, the phase of each species is extracted from the speciesName vector and saved as variable speciesPhase in the workspace. Transport and boundary condition functions of solid species and solutes are computed differently as mentioned in previous sections.

The stoichiometric matrix is built in the 3rd block using the symbolic programming capability of MATLAB described above. The unit consistency between species and reaction units is enforced by applying the conversion factor convFactor defined in the 1st block.

The function handles of the reaction rates are computed in the 4th block. Reaction rates described in the string form in *userMEDILAB.m* are not recognizable by *pdepe*. Therefore, the names of the rate constants and the constituents are substituted by the corresponding values and u vector elements, respectively. For example, for the reaction in Eq. (24), if k_AB has the numeric value of 0.2 provided through *inputMEDIALAB.txt* and A and B are the 5th and 8th species in the speciesName vector, then the reaction equation will be transformed to:

```
r = '0.2 * u(5) * u(8)'
```

This string ins then converted to a function handle using the MATLAB built-in function, str2func. str2func('str') constructs a function handle for the function named in the string 'str'.

In a similar manner, function handles of transport and boundary conditions are computed in the 5th block. Terms for advection, diffusion (including molecular diffusion for solutes) and boundary conditions are created in a string format and then are converted to function handles.

The variables containing the function handles of advection (`advection`), diffusion (`diffusion`), boundary conditions (`plBC` and `prBC`) along with the `stoichiometrix` matrix are passed to *mainMEDIALAB.m* to be used as components of *pdepe* solver.

4.5 mainMEDIALAB

mainMEDIALAB.m is the core script file of MEDIALAB which is executed from MATLAB's home screen. It is in this script that *userMEDIALAB.m*, *autoMEDIALAB.m* and *pdepe* solvers are called and the final solution, `sol`, is saved in the *resultMEDIALAB.mat.* `sol` includes the model output for all simulated variables, depth and time values in a MATLAB matrix format. After termination of simulation, variables of `depth`, `time` and `simValues` are saved in the workspace. They are used for plotting and reaction rate calculations through *postprocessMEDIALAB.m.*

4.6 Postprocessing

Once the simulation is finished and the file *ResultsMEDIALAB.mat* has been created, *postprocessingMEDIALAB.m* can be used to calculate further output variables and create graphical model output. Several plotting functions *plotXXX.m* are provided with the package that plot different types of plots. As listed in **Error! Reference source not found.**, the user can select in *plotOptionsMEDIALAB.m* which of the plots are created, and also specify some additional plot properties.

Table 3: Options for plotting provided in plotOptionsMEDIALAB.m, files that are used for plotting the respective plots in addition to the code contained in postprocessMEDIALAB.m, and short summary of plots created if the options are set to 'True'

Option	Code file	Plots created
relativeConcentrationGradients	*plotGradients.m*	relative concentration gradients (dC/dz)/C(z) as a function of depth (at initial time, half the simulation time, and at the end), and maximum relative concentration gradient for each species during the simulation.
timesteps		evolution of internal time steps (created from the file compLevelOutput.txt), only available for the last simulation
measuredProfiles	*plotMEDIALAB.m*	measured profiles of solute and solid species from fieldData.xlsx together with simulated profiles at at initial time, half the simulation time, and at the end*
reactions	*plotAgainstDepth.m*	Time evolution of the vertical profiles of all reaction rates
concentrations	*plotAgainstDepth.m*	Time evolution of the vertical profiles of the concentrations of all solutes and solids
saturationIndices	*plotAgainstDepth.m*	Time evolution of the vertical profiles of saturation indices
fractionFigures	*plotFractions.m*	Figures showing which fractions of certain elements or species are contained in which species or consumed/produced by which reaction
concentrationsTemporal	*plotAgainstTime.m*	Concentrations as a function of time for three (shallow) depths
fluxesTemporal	*plotAgainstTime.m*	Fluxes at the sediment-water interface for all substances as a function of time
massBalances	*plotElementalFluxes.m*	Mass balance for several elements over time.
fRed	*plotFred*	Flux of reduced substances from the sediment as a function of time

*note that the simulated profiles are not expected to match the observed profiles in the test case provided on GitHub.

Simulated vertical concentration profiles can be graphically compared with field data stored in the Excel file *fieldData.xlsx* if `pOptions.doPlot.measuredProfiles` is set to `True`. The file *fieldData.xlsx* needs to contain one worksheet for each simulated species, where depth is given in the first column of that worksheet and observed concentrations (in the same unit as used in the model) in the second column. Further columns can be used for operations (e.g. unit conversion) as these are not included in the plot by MEDIALAB.

postprocessMEDIALAB.m per default uses the model output stored in the file *ResultsMEDIALAB.mat*, but the name of another model output file stored in the same folder can be provided as an argument to plot the output from another simulation run.

5 Running MEDIALAB

In order to run a sediment diagenesis model with MEDIALAB, the user has to perform the following steps:

1) Define the set of reactions to be solved in the model in userMEDIALAB.m (section 4.2).

2) Define the model parameters and boundary conditions in InputMEDIALAB.txt (section 4.3).

3) Define the initial conditions in ICvalues.mat (section 4.2).

4) Run the model by running the *mainMEDIALAB.m* script (section 4.5).

5) Define the plots to be produced in plotOptionsMEDIALAB.m (section 4.6).

6) Do initial postprocessing and plotting by running the *postprocessMEDIALAB.m* script (section 4.6).

Further postprocessing and model output analysis can be performed with own scripts using the model output stored on resultMEDIALAB.mat

6 Troubleshooting

6.1 Time integration failed

This is one of the most frequent problems occurring in this model. It means that the solver crashed and failed to solve the system of the partial differential equations. There are many possibilities why this can occur. Most likely, this might be a result of a wide spacing of the spatial domain, or the effect of certain parameters or boundary conditions. Try increasing the space resolution, e.g. from 500 points to 1000 points (at the expanse of longer running time). Alternatively, the user is encouraged to go back to the set of parameters and boundary conditions, for which the solver worked. Change the parameters one by one to determine which one caused the solver to fail.

6.2 Segmentation Violation

This error is not frequently seen. It is usually a result of running parallel instances of the model at the same time, which might be needed when fitting the parameters. To resolve this problem, reduce the number of model instances to not exceed the number of available CPU cores. Thus, if the CPU has 4 cores, run a maximum of 4 model instances at the same time.

6.3 Long Running Time

Normally, the model should finish running within minutes to an hour. Longer running times occur if there are variable boundary conditions. If the running time is too long, it might be a result of certain values being 'too small'. Try changing the parameters, or even adding 'if' conditions, so that those values will not be 'too small'. Also, alternative formulations for certain reaction rates may be tried, since experience shows that the solver always works faster with rate laws that are first order with respect to the species.

6.4 Specific worksheet not found

This error typically occurs if a worksheet is missing for a species that should be plotted in the the Excel file with the field data. Make sure that this file contains a worksheet for every species specified in `speciesName`. If no data is available for a constitutent, add an empty worksheet with the species name. A template of the Excel file is provided, with the name *fieldData.xlsx*. Note that the names of the sheet have to match the names of the species exactly.

6.5 DAE of index greater than 1

This error occurs when the partial differential equation is not parabolic, e.g. when the bioturbation rate is set to be zero. *pdepe* is only capable for solving parabolic-elliptic problems. Therefore, for that case, set the bioturbation rate to a very small value, e.g. 0.001, to allow the solver to work. Most of the time, the theoretical model can be well expressed as a system of parabolic equations.

6.6 Memory - out of bounds

This error mostly occurs on system with less than 4 GB of random-access memory installed because MEDIALAB requires ~ 1 GB of contiguous RAM available for the initialisation of the matrices. Disabling start up programs and performing a clean reboot often solves the issue.

References

Aguilera DR, Jourabchi P, Spiteri C, Regnier P (2005). A knowledge-based reactive transport approach for the simulation of biogeochemical dynamics in earth systems. Geochemistry Geophysics Geosystems 6(Q07012). https://doi.org/10.1029/2004GC000899

Shafei B (2012). Reactive transport in natural porous media: Contaminant sorption and pore-scale heterogeneity, PhD thesis, Georgia Institute of Tehchnology.

Steinsberger T, Müller B, Gerber C, Shafei B, Schmid M (2019). Modeling sediment oxygen demand in a highly productive lake under various trophic scenarios. PLoS ONE 14(10): e0222318. https://doi.org/10.1371/journal.pone.0222318