SimStrat
1D k-epsilon lake model

Developer Manual

Contents

Swiss Federal Institut of Aquatic Science and Technology, December 2016

I. Introduction

This developer manual is ...

II. Requirements

make
    Cmake
    GNU-Fortran compiler
    Alternatively Intel Fortran Compiler

III. Download and install SimStrat

IV. Code Structure

The code structure of simstrat has been updated to an object-oriented model. In this chapter, the main architecture is documented and a short description of each used class is given. Additionally, the important classes for solving model variables and for updating the grid are explained in more detail.

I. Overview

The following class diagram gives an overview of the coupling of the different classes. The main points

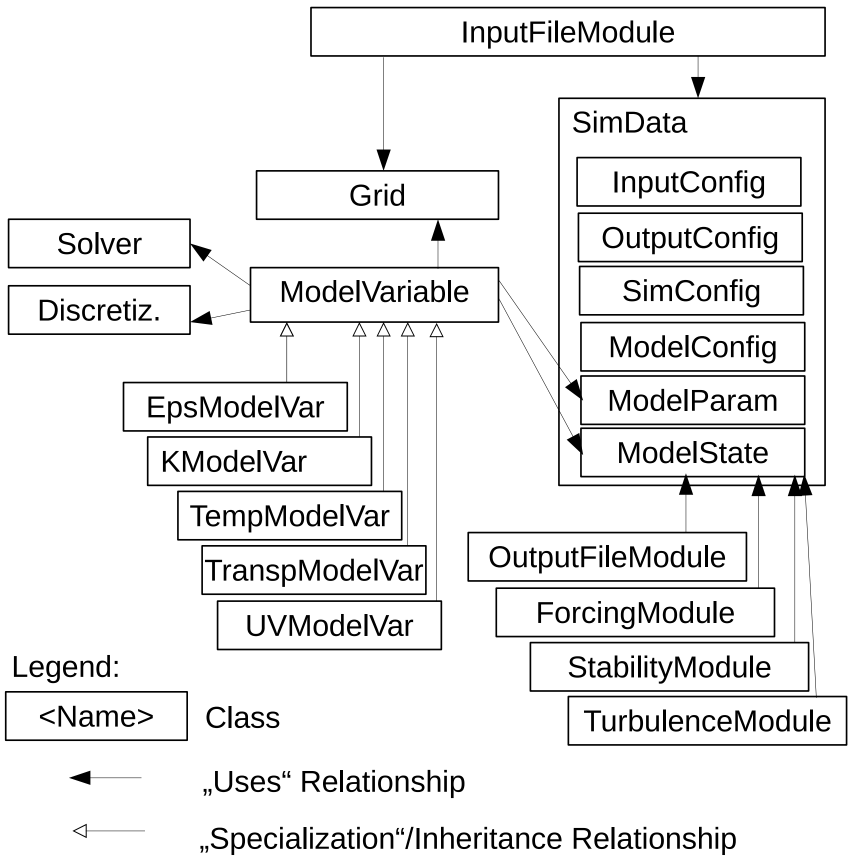

Figure 1: Class diagram of Simstrat2.

of the architecture are:

- All simulation data is kept inside the SimData class and its various subclasses. Additionally, a grid class containing all information and methods (update/interpolate etc) concerning the grid is kept. This information is all read and set up by the InputFileModule.
- All variables that need to be solved to update $(K, \epsilon, T, S, U, V)$ have a corresponding module that contains the logic to calculate the terms needed for constructing a linear system of equations. All these modules are inherited from an abstract base class "ModelVariable" that provides the logic to

combine solver, grid, discretization and variable data. Each inherited subclass only overwrites the needed methods (usually calc_terms).

- Most other parts of the software, such as output data storage, forcing or (stability-)parameter updates are structured into single classes that have access to the model state and/or grid.

Once the simulation has been set up, only the update methods of each module are called during each iteration:

1. Update current simulation date
2. Read & update forcing data (ForcingModule update)
3. Read & update in/outflows
4. Update stability data $N^2, cmue$ (StabilityModule update)
5. Update U and V (UVModelVar update)
6. Update T (TempModelVar update)
7. Update S (TranspModelVar update)
8. Update turbulence data $P, P_{seiche}, \nu_m, \nu_h$ (TurbulenceModule update)
9. Update k (KModelVar update)
10. Update eps (EpsModelVar update)
11. Call logger to log current state

II. Classes

The following classes/modules/interfaces are of importance:

Interfaces

Interfaces that may have multiple implementations to facilitate different variants

LinSysSolver — Interface for a linear system solver, that solves for x in $Ax = b$, where A is tridiagonal.

SimstratOutputLogger — Interface that describes an output module which writes simulation data for further processing (e.g. CSV files or Matlab files etc)

ModelVariable — Interface that defines a generic object that manipulates a model variable (e.g. Transport equation, or update of wind-shear etc). Contains an abstract update method that performs the update process depending on the actual parametrization:

1. "Calculate terms": This calls the subroutine to calculate the source terms and boundary conditions. This method is abstract and has to be overwritten by the effective implementation.
2. Create linear system of equations: Uses the associated discretization method to create a tridiagonal matrix A and the right hand side.
3. Solve: Uses the associated solver to solve the before created linear system
4. "Post Solve": Execute a method to manipulate the variable after solving - default implementation is just an empty method but can be overwritten as needed.

Discretization — Defines the interface for discretization classes that, based on input quantities such as fluxes, current states and timestep, construct a linear system of equation.

DTOs

Data transfer objects that mainly store data, but do not manipulate them (except e.g. converting data)

InputConfig — Holds file system path of all input files

OutputConfig — Holds file system path for all output files and additional information such as logging frequency and similar

SimConfig — Stores fixed simulation configuration, such as start/end date and timestep.

ModelConfig — Stores fixed model configuration, such as turbulence model selection or forcing mode.

ModelParam — Stores fixed model parameters, such as Latitude or ρ_{air}

ModelState — All model variables (i.e. everything that changes during an iteration)

Special classes

Utility and helper classes

Inputfile	Module that reads all the input files and populates the aforementioned data transfer objects. Sets up the simulation.
Grid	Contains all information and routines concerning the grid. Both grids (centered on face / centered on volume) are stored and updated in this class. Additionally, methods such as interpolating from/to a grid and updating area factors belong to the grid.
Forcing	Parses forcing files and updates the current ModelState according to the date.
Stability	Manipulates the NN and cmue1/2 variables of the ModelState

Absorption

Reads absorption input file and modifies corresponding variables of the Modelstate

Lateral

Reads input files for in/outflows (Qinp, Qout, Tin, Sin) and integrates/interpolates values in order to modify Q_{inp}/Q_{vert}

Advection

Based on already read in/outflows (Q_{inp}/Q_{vert}), calculates the advection terms

---

Model state variable classes

Classes that are inherited from ModelVariable

TranspModelVar	Simplest inherited class from ModelVariable. Contains additional method to assign an external source term. Overwrites calc_terms and sets source terms to the afore mentioned source terms. This class can be used for all quantities that need to be transported/distributed without special boundary conditions/equations. Examples include salinity S and additional biogeochemical quantities.
TempModelVar	Used for T variable. In the calc_terms function, radiance into each layer is calculated and added as a source term. Additionally, boundary fluxes according to geothermal/boundary heat fluxes are added.
UVModelVar	Used for U/V variable. Same class can be instantiated twice and assigned to U respectively V, as the code is the same. Additional to the variable, a shear stress variable has to be defined using the assign_shear_stress method.
KModelVar	Used for K variable. Contains calculation for sources/boundary conditions needed to calculate K. Contains post_solve function to adjust first/last element of K after solve
EpsModelVar	Used for ϵ variable. Similar to KModelVar.

III. Implementation of an equation

In this chapter, the detailed approach how a state variable is solved is explained (mostly for code documentation purposes - for more information about the math/discretization used, see next chapters).

IV. Grid

All methods and variables concerning the grid structure are now combined into a grid class. The grid class takes care of

- Updating area factors
- Storing the two grids (volume and face) including heights, areas etc
- Growing/merging grid boxes if needed
- updating upper bounds and sizes
- Interpolate from and to the grid (e.g. interpolate a value with a different z-axis onto the volume grid)

Note that there 2 grids - one that stores the depth of each face, and one that stores the depth of each volume center. Consequently, they are named `z_face` respectively `z_volume`. Both are 1-indexed and `z_face` is always one element longer than `z_volume`. The current upper bounds for both grids can be queried using the `ubnd_fce` and `ubnd_vol` variable in the grid class.

These upper bounds are automatically updated using the `update_nz` method, e.g. when shrinking/growing the grid.

V. Output logger

The interface `SimstratOutputLogger` in file `strat_outputfile.f03` defines a generic logger. Currently, `SimpleLogger` is implemented.

Configuration of the logger is done using the `output_cfg` variable in the `simdata` structure. It consists of an array of pointers and descriptions, so that the logger knows which variables has to be saved to which file.

Example:

```
self%simdata%output_cfg%output_vars(1)%name = "V"
self%simdata%output_cfg%output_vars(1)%values => self%simdata%model%V
self%simdata%output_cfg%output_vars(1)%center_grid = .true.
```

This tells the logger to output variable `self%simdata%model%V` as variable with the name "V" on the volume grid, whenever the log function is called.

VI. Proposed Coupling to AED2

VI.1 Input files

VI.2 Transport Modelvariable

VI.3 Update

VI.4 Output files

VII. Further work on the implementation

The current state (Spring 17) of the code should give a solid foundation for further development. It is by no means ideal - it is more a high-level restructuring than a in-depth refactoring of each part/method. Thus, there are still a lot of things to improve. This chapter should give some ideas on what could be further improved.

- Clean code and readability: Lot of ambigious named variables and hard to debug if/else constructs with code duplication. Methods (e.g. the lateral method) are sometimes very long and could probably be restructured into a combination of well named, short and easy to test methods.
- File reading of input files: Very hard to debug, very unclear and old-style goto- failure modes. For long term development, this should be refactored out into a generic time-series class that can read the desired file format and returns it as a 2d array or so. This class could then be re-used in all modules that have to read input files (such as Qin, Qout, Tin etc).
- List of methods that badly need refactoring:
  - `forcing_update`: Much too long. Should be multiple, well named subfunctions instead of long if/else structure
  - `everything in lateral`: same as above

V. Discretization Scheme and Numerical Approach

I. System of Governing Equations

The governing equations are described in Goudsmit et al. 2000

$$\frac{\partial T}{\partial t} = \frac{1}{A}\frac{\partial}{\partial z}\left(A(\nu_t' + \nu')\frac{\partial T}{\partial z}\right) + \frac{1}{\rho_0 c_p}\frac{\partial H_{sol}}{\partial z} + \frac{dA}{dz}\frac{H_{geo}}{A\rho_0 c_p} \tag{1}$$

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{1}{A}\frac{\partial}{\partial z}\left(A(\nu_t + \nu)\frac{\partial u}{\partial z}\right) + f\nu \\ \frac{\partial v}{\partial t} &= \frac{1}{A}\frac{\partial}{\partial z}\left(A(\nu_t + \nu)\frac{\partial v}{\partial z}\right) - fu \end{aligned} \tag{2}$$

$$\frac{\partial k}{\partial t} = \frac{1}{A}\frac{\partial}{\partial z}\left(A\nu_k\frac{\partial k}{\partial z}\right) + P + P_{seiche} + B - \epsilon \tag{3}$$

$$\frac{\partial \epsilon}{\partial t} = \frac{1}{A}\frac{\partial}{\partial z}\left(A\nu_\epsilon\frac{\partial \epsilon}{\partial z}\right) + \frac{k}{\epsilon}\left(c_{\epsilon 1}(P + P_{seiche}) + c_{\epsilon 3}B - c_{\epsilon 2}\epsilon\right) \tag{4}$$

Note that these equations do not include advection caused by in- and outflow caused by rivers and/or groundwater sources. In Simstrat, advection is decoupled from diffusion and source/sink treatment and computed separately.

II. Numerical Concept

In the above system of partial differential equations the temporal change of a quantity formally consists of a diffusive component D and a source/sink component S:

$$\frac{\partial \phi}{\partial t} = D(t,\phi) + S(t) \tag{5}$$

For the numerical integration an implicit Euler method is applied:

$$\frac{\phi^{n+1} - \phi^{n}}{dt} = D(t^{n+1}, \phi^{n+1}) + S(t^{n}) \tag{6}$$

The resulting system of discretized equations can be written as a system of linear equations of the form:

$$R\phi^{n+1} = \phi^{n} + dt \cdot S^{n} \tag{7}$$

where ϕ represents any of the above quantities (T, u, v, k, ϵ), and R is a tridiagonal matrix of the form:

$$R = \begin{bmatrix} bu & au & 0 & \dots & 0 \\ cu & bu & au & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & au \\ 0 & \dots & 0 & cu & bu \end{bmatrix} \tag{8}$$

where $bu = (1 - au - cu)$. This system of linear equations can be solved very efficiently by the tridiagonal matrix algorithm.

III. Discretization Scheme: Finite Volume Approach on a Staggered Grid

The discretization follows a finite volume approach. The water column is divided into n_{vol} volumes with height h_z, an area A_{z+1} at the top of the volume and A_{z-1} at the bottom (the so-called faces). The volume centers have indices from 1 to n_{vol} (bottom to surface). The faces have indices from 1 to n_{face} which is equal to $n_{vol} + 1$. The mean flow quantities (T, u, v) are placed at the center of the volumes. Turbulent quantities (k, ϵ) are assigned to the volume faces (the interface between two volumes). Within a volume, the diffusion coefficients are assumed to be equal to the mean value of the diffusion coefficients at the upper and lower face.

The temporal change of mean flow quantities within each volume results from the diffusive fluxes through the two interfaces and sources/sinks within the volume:

$$\left.\frac{\partial \phi}{\partial t}\right|_{z} = \frac{A_z F_z - A_{z+1} F_{z+1}}{V_z} + S_z \tag{9}$$

where $V_z = h_z\,(A_z + A_{z+1})/2$.

The flux through the upper and lower interface are:

$$F_{z+1,face} = -\nu \left.\frac{\partial \phi}{\partial z}\right|_{z+1}^{face} = -\nu_{z+1} \frac{\phi_{z+1} - \phi_z}{(h_{z+1} + h_z)/2} \tag{10}$$

$$F_{z,face} = -\nu \left.\frac{\partial \phi}{\partial z}\right|_{z}^{face} = -\nu_{z} \frac{\phi_{z} - \phi_{z-1}}{(h_{z} + h_{z-1})/2} \tag{11}$$

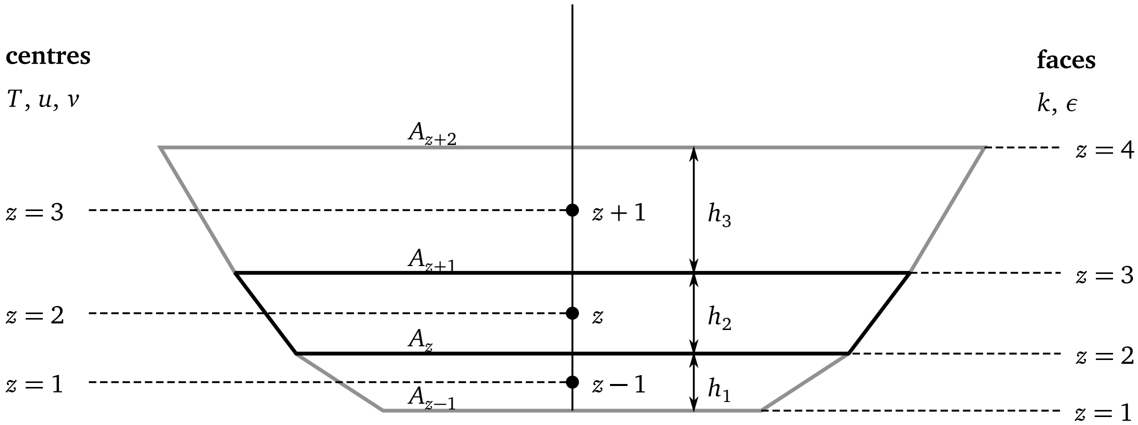

Figure 2: Discretization Scheme

The resulting discretization for mean flow quantities is then:

$$\frac{\phi_z^{n+1}-\phi_z^n}{dt} = \frac{1}{h_z\,(A_z+A_{z+1})/2}\left(A_{z+1}\,\nu_{z+1}^n\frac{\phi_{z+1}^{n+1}-\phi_z^{n+1}}{(h_{z+1}+h_z)/2} - A_z\,\nu_z^n\frac{\phi_z^{n+1}-\phi_{z-1}^{n+1}}{(h_z+h_{z-1})/2}\right)+S_z^n \tag{12}$$

This can be rearranged to:

$$\begin{aligned} & dt\cdot AreaFactor_1\cdot \nu_z^{n+1}\phi_{z-1}^{n+1} \\ & +(1-dt\cdot AreaFactor_1\,\nu_z^n - dt\cdot form_2\,\nu_{z+1}^n)\phi_z^{n+1} \\ & + dt\cdot AreaFactor_2\cdot \nu_{z+1}^{n+1}\phi_{z+1}^{n+1} = \phi_z^n + dt\cdot S_z^n \end{aligned} \tag{13}$$

$$AreaFactor_1 = \frac{-4A_z}{(A_z+A_{z+1})\,(h_z+h_{z-1})} \tag{14}$$

$$AreaFactor_2 = \frac{-4A_{z+1}}{(A_z+A_{z+1})\,(h_{z+1}+h_z)} \tag{15}$$

The temporal change of turbulent quantities mainly follows the same concept except that the control volumes are shifted so that the turbulent quantities are at the center of the control volumes and the mean flow quantities are located at the volume faces. The balance for the turbulent quantities is therefore:

$$\left.\frac{\partial \phi}{\partial t}\right|_z = \frac{(A_z+A_{z-1})\,/2\cdot F_{z-1} - (A_{z+1}+A_z)\,/2\cdot F_z}{V_z} + S_z \tag{16}$$

where $V_z = A_z\,(h_{z-1}+h_z)\,/2$.

To calculate the flux through the upper and lower interface the turbulent kinetic energy and the dissipation at these interfaces are approximated by averaging the TKE and dissipation at the volume centres:

$$\nu_{z,face} = \left(\nu_{z,center} + \nu_{z-1,center}\right)/2 \tag{17}$$

$$\tag{18}$$

The flux through the upper and lower interface are:

$$F_{z,center} = -\nu\left.\frac{\partial \phi}{\partial z}\right|_z^{center} = -\nu_z\frac{\phi_{z+1}-\phi_z}{h_z} \tag{19}$$

$$F_{z-1,center} = -\nu\left.\frac{\partial \phi}{\partial z}\right|_{z-1}^{center} = -\nu_{z-1}\frac{\phi_z-\phi_{z-1}}{h_{z-1}} \tag{20}$$

The resulting discretization for turbulent quantities is then:

$$\frac{\phi_z^{n+1}-\phi_z^n}{dt} = \frac{1}{A_z\left(h_z+h_{z+1}\right)/2}\left(\frac{A_{z+1}+A_z}{2}\,\nu_z^n\,\frac{\phi_{z+1}^{n+1}-\phi_z^{n+1}}{h_z}-\frac{A_z+A_{z-1}}{2}\,\nu_{z-1}^n\,\frac{\phi_z^{n+1}-\phi_{z-1}^{n+1}}{h_{z-1}}\right)+S_z^n \tag{21}$$

This can be rearranged to:

$$\begin{aligned}
&\left(dt\cdot AreaFactor_{k1}\cdot \nu_{z-1}^{n+1}\right)\phi_{z-1}^{n+1}\\
&+\left(1-dt\cdot AreaFactor_{k1}\,\nu_{z-1}^n-dt\cdot AreaFactor_{k2}\,\nu_z^n\right)\phi_z^{n+1}\\
&+\left(dt\cdot AreaFactor_{k2}\cdot \nu_z^{n+1}\right)\phi_{z+1}^{n+1}\\
&=\phi_z^n+dt\cdot S_z^n
\end{aligned} \tag{22}$$

$$AreaFactor_{k1}=\frac{A_z+A_{z-1}}{A_z\left(h_z+h_{z-1}\right)h_{z-1}} \tag{23}$$

$$AreaFactor_{k2}=\frac{A_{z+1}+A_z}{A_z\left(h_z+h_{z-1}\right)h_z} \tag{24}$$

IV. Boundary Conditions at the Sediment and Lake Surface

The basic discretization assumes a no-flux boundary at the sediment and lake surface. Concentration dependent fluxes at these interfaces are implemented as an additional component of the main diagonal.