

# SimStrat

## 1D k-epsilon lake model

### Developer Manual

#### Contents

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>II</b>	<b>Requirements</b>	<b>1</b>
<b>III</b>	<b>Download and install SimStrat</b>	<b>1</b>
<b>IV</b>	<b>Code Structure</b>	<b>2</b>
I	Overview . . . . .	2
II	Classes . . . . .	4
III	Implementation of an equation . . . . .	5
IV	Grid . . . . .	6
V	Proposed Coupling to AED2 . . . . .	6
V1	Input files . . . . .	6
V2	Transport Modelvariable . . . . .	6
V3	Update . . . . .	6
V4	Output files . . . . .	6
<b>V</b>	<b>Discretisation Scheme and Numerical Approach</b>	<b>6</b>
I	System of Governing Equations . . . . .	6
II	Numerical Concept . . . . .	6
III	Discretization Scheme: Finite Volume Approach on a Staggered Grid . . . . .	7
IV	Boundary Condition at the Sediment and Lake Surface . . . . .	9



## **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

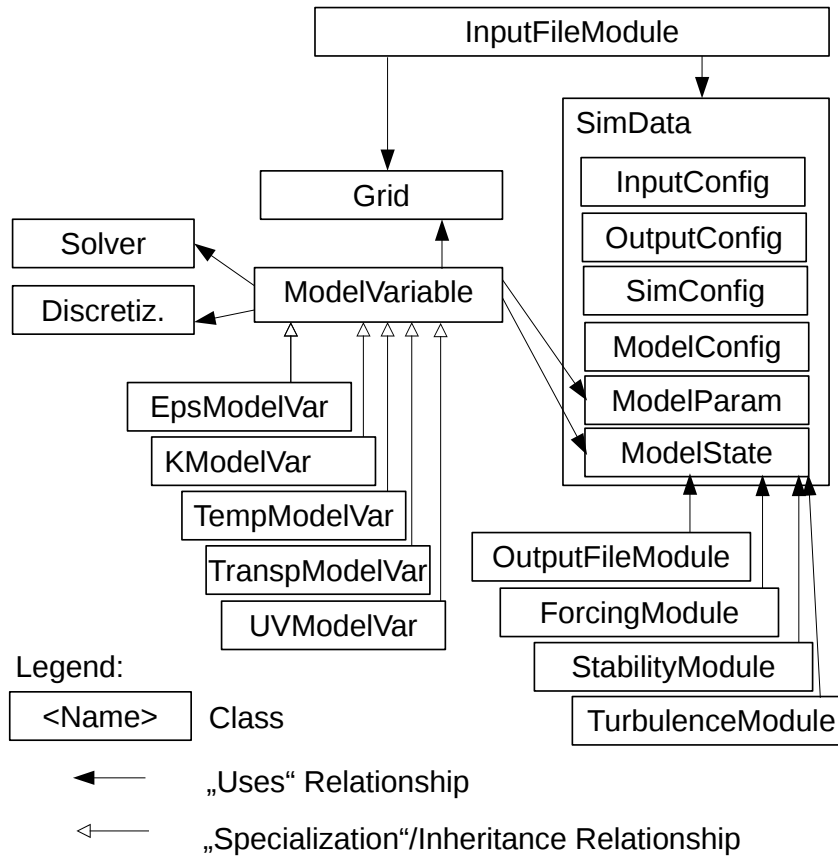


Figure 1: Class diagram of Simstrat2.

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. Update stability data  $N^2$ ,  $cmue$  (StabilityModule update)
4. Update  $U$  and  $V$  (UVModelVar update)
5. Update  $T$  (TempModelVar update)
6. Update  $S$  (TranspModelVar update)
7. Update turbulence data  $P$ ,  $P_{seiche}$ ,  $\nu_m$ ,  $\nu_h$  (TurbulenceModule update)
8. Update  $k$  (KModelVar update)
9. Update  $eps$  (EpsModelVar update)
10. 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

<b>LinSysSolver</b>	Interface for a linear system solver, that solves for $x$ in $Ax = b$ , where $A$ is tridiagonal.
<b>SimstratOutputLogger</b>	Interface that describes an output module which writes simulation data for further processing (e.g. CSV files or Matlab files etc)
<b>ModelVariable</b>	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: <ol style="list-style-type: none"><li>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.</li><li>2. Create linear system of equations: Uses the associated discretization method to create a tridiagonal matrix <math>A</math> and the right hand side.</li><li>3. Solve: Uses the associated solver to solve the before created linear system</li><li>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.</li></ol>
<b>Discretization</b>	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)

<b>InputConfig</b>	Holds file system path of all input files
<b>OutputConfig</b>	Holds file system path for all output files and additional information such as logging frequency and similar
<b>SimConfig</b>	Stores fixed simulation configuration, such as start/end date and timestep.
<b>ModelConfig</b>	Stores fixed model configuration, such as turbulence model selection or forcing mode.
<b>ModelParam</b>	Stores fixed model parameters, such as Latitude or $\rho_{air}$
<b>ModelState</b>	All model variables (i.e. everything that changes during an iteration)

---

## Special classes

Utility and helper classes

<b>Inputfile</b>	Module that reads all the input files and populates the aforementioned data transfer objects. Sets up the simulation.
<b>Grid</b>	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.
<b>Forcing</b>	Parses forcing files and updates the current ModelState according to the date.
<b>Stability</b>	Manipulates the NN and cmue1/2 variables of the ModelState

---

## Model state variable classes

Classes that are inherited from ModelVariable

<b>TranspModelVar</b>	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.
<b>TempModelVar</b>	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.
<b>UVMModelVar</b>	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.
<b>KModelVar</b>	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
<b>EpsModelVar</b>	Used for $\epsilon$ 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

#### V. Proposed Coupling to AED2

##### V1 Input files

##### V2 Transport Modelvariable

##### V3 Update

##### V4 Output files

#### V. Discretisation Scheme and Numerical Approach

##### I. System of Governing Equations

The governing equations are described in Goudsmit et al. 2000

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

$$\frac{\partial u}{\partial t} = \frac{1}{A} \frac{\partial}{\partial z} \left( A(v_t + v) \frac{\partial u}{\partial z} \right) + f v \quad (2)$$

$$\frac{\partial v}{\partial t} = \frac{1}{A} \frac{\partial}{\partial z} \left( A(v_t + v) \frac{\partial v}{\partial z} \right) - f u$$

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

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

##### 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  $S$ :

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

For the numerical integration an implicit Euler method is applied:

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

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

$$R \phi^{n+1} = \phi^n + dt \cdot S^n \quad (7)$$

where  $\phi$  represents any of the above quantities ( $T$ ,  $u$ ,  $v$ ,  $k$ ,  $\epsilon$ ), and  $R$  is a tridiagonal matrix of the form:

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

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 discretisation follows a finite volume approach. The water column is divided into  $nz$  volumes with height  $h_z$ , an area  $A_z$  at the top and  $A_{z-1}$  at the bottom. The volume centres have indices from 1 to  $nz$  (bottom to surface). The volume faces have indices from 0 to  $nz$ . The mean flow quantities ( $T, u, v$ ) are placed at the centre of the volumes. Turbulent quantities ( $k, \epsilon$ ) are assigned to the volume faces (the interface between two volumes). Within a volume, the diffusion coefficients are assumed to be equal to the diffusion coefficient assigned to the top face of the volume.

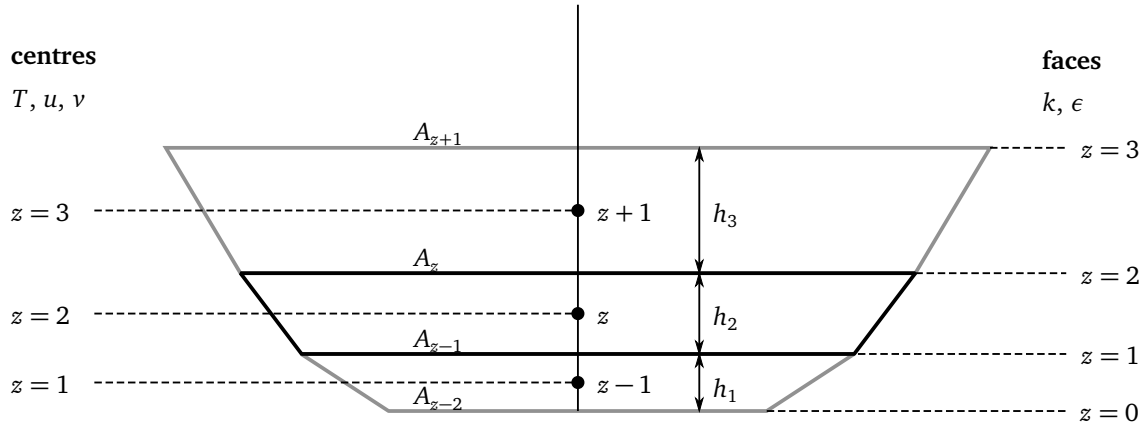


Figure 2: Discretisation Scheme

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

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

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

The flux through the upper and lower interface are:

$$F_{face,z} = -v \left. \frac{\partial \phi}{\partial z} \right|_z^{face} = -v_z \frac{\phi_{z+1} - \phi_z}{(h_{z+1} + h_z)/2} \quad (10)$$

$$F_{face,z-1} = -v \left. \frac{\partial \phi}{\partial z} \right|_{z-1}^{face} = -v_z \frac{\phi_z - \phi_{z-1}}{(h_z + h_{z-1})/2} \quad (11)$$

The resulting discretization for mean flow quantities is then:

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



This can be rearranged to:

$$\begin{aligned} & dt \cdot form_1 \cdot \nu_z^{n+1} \phi_{z-1}^{n+1} \\ & + (1 - dt \cdot form_1 \nu_z^n - dt \cdot form_2 \nu_z^n) \phi_z^{n+1} \\ & + dt \cdot form_2 \cdot \nu_z^{n+1} \phi_{z+1}^{n+1} = \phi_z^n + dt \cdot S_z^n \end{aligned} \quad (13)$$

$$form_1 = \frac{-4A_{z-1}}{(A_z + A_{z-1})(h_z + h_{z-1})} \quad (14)$$

$$form_2 = \frac{-4A_z}{(A_z + A_{z-1})(h_{z+1} + h_z)} \quad (15)$$

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:

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

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

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:

$$\nu_{z,face} = (\nu_{z,centre} + \nu_{z-1,centre})/2 \quad (17)$$

$$(18)$$

The flux through the upper and lower interface are:

$$F_{z+1} = -\nu \left. \frac{\partial \phi}{\partial z} \right|_{z+1} = -\nu_{z,face} \frac{\phi_{z+1} - \phi_z}{h_{z+1}} \quad (19)$$

$$F_z = -\nu \left. \frac{\partial \phi}{\partial z} \right|_z = -\nu_{z,face} \frac{\phi_z - \phi_{z-1}}{h_z} \quad (20)$$

The resulting discretization for turbulent quantities is then:

$$\begin{aligned} & \frac{\phi_z^{n+1} - \phi_z^n}{dt} = \\ & \frac{2}{A_z(h_z + h_{z+1})} \left( \frac{A_{z+1} + A_z}{2} \nu_{z,face}^n \frac{\phi_{z+1}^{n+1} - \phi_z^{n+1}}{h_{z+1}} - \frac{A_z + A_{z-1}}{2} \nu_{z,face}^n \frac{\phi_z^{n+1} - \phi_{z-1}^{n+1}}{h_z} \right) + S_z^n \end{aligned} \quad (21)$$

This can be rearranged to:

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

$$form_{k1} = \frac{A_z + A_{z-1}}{A_z(h_z + h_{z+1})h_z} \quad (23)$$

$$form_{k2} = \frac{A_{z+1} + A_z}{A_z(h_z + h_{z+1})h_{z+1}} \quad (24)$$

#### IV Boundary Condition at the Sediment and Lake Surface

The basic discretisation 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.