**2 Framework Design**

**2.1 Basic Structure**

The HydroModels.jl framework comprises four fundamental types: Flux, Bucket, Route, and Model, through these four basic types we can build a hydrological model from several hydrological formulas. For flexible structure calling, all of the structures have functions with the same name, which obtain output results through input variables and model parameters, and provide input, output and state variables, calculation parameter information, and guide users in how to input data and parameters of the model. The detailed design of each type is as follows:

(1) Flux

The purpose of the *Flux* type is typically to calculate one or more hydrological fluxes based on input variables and parameters. Considering various application demands and computation methods, Flux can be further subdivided into four subtypes: *Simple Flux*, *State Flux, Route Flux*, and *Neural Flux*.

*Simple Flux* is mainly used for conventional hydrological flux calculations, typically can be represented by simple formulas, which is the most common and fundamental calculation method in hydrological models. By inputting the specified variables and parameters based on the flux input and parameter information, the hydrological flux calculated by *Simple Flux* can be obtained.

*State Flux* is applied to the calculation of state fluxes in a hydrological bucket, such as the snowpack value in the exp-hydro model’s snow calculation module. The state flux is usually updated based on the input and output fluxes at each time periods, the change in the state flux of the hydrological bucket, D(state), is equal to the sum of the input fluxes minus the sum of the output fluxes. Thus, it is the key to build the ordinary differential equations (ODE) and support the construction of the hydrological bucket.

*Route Flux* is a special Flux structure, which is usually used for routing computation. It can be used for the routing computation of one flux in the conceptual hydrological model and a distributed model computation unit (vertical computation), and it can also be used for the routing computation of the outflow process of the computation unit (lateral computation). The current confluence computation is generally simplified from the Saint-Venant equation, such as unit hydrograph, Nash Cascade, Muskingum, etc. Each method is computed by constructing ODE function. The detailed computation progress can be found in the **Appendix**.

*Neural Flux*, similar to Simple Flux, can also be used for hydrological flux computation, while the difference is that the computation method of the fluxes is based on the neural network models, which is used to support to building the hot topic in hydrological modelling, named physical informed model.

(2) Bucket

The Bucket represents a hydrological computation module or model. This structure comprises multiple fluxes, corresponding to various hydrological flux calculation formulas within the module. The computation steps are as follows:

Step 1. When the element includes a state flux, the ODE solver should initially be employed to resolve the module's state flux.

Step 2. The state flux, combined with the module's input flux, is utilized for calculating other common fluxes.

对于bucket中ODE问题的求解和其余通量的计算，bucket根据flux在run time阶段构建了ode function和flux function

Bucket结构整合了多个flux

Since the hydrological flux output by some formulas serves as the input flux for other formulas, the calculation order of fluxes within the element will directly impact the computational process. For instance, if the input flux required for a formula's calculation has not yet been determined, a variable non-existence issue will arise. To resolve this problem, it is essential to construct a directed graph based on the input and output flux names of multiple calculation formulas to establish the correct calculation order for each formula.

(3) Unit

A unit represents a hydrological model, typically comprising multiple elements. All hydrological fluxes within the model are calculated by inputting data and integrating the output results of each element through the traversal of each element. It is worth noting that flux, element, and unit share the same invocation method. Therefore, as long as the directed graph constructed from the inputs and outputs remains coherent, a unit can be composed of fluxes, elements, or even other units. Consequently, a unit is primarily used to integrate multiple calculation formulas, modules, and models, thereby facilitating parameter optimization or other analytical computations.

**2.2 Basic Method**

To achieve functions such as model building, runoff simulation, and parameter optimization, the HydroModels.jl framework offers three fundamental methods: *construction, solution, and optimization*. The detailed design concepts of each method are as follows:

2.2.1 Construction

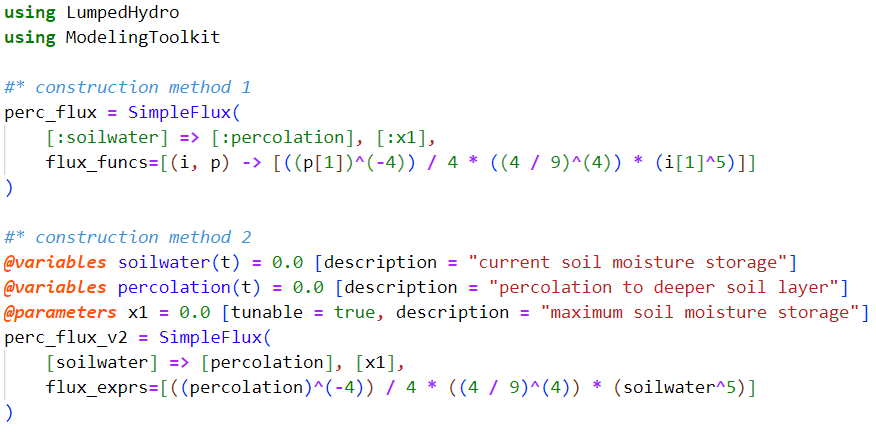
The construction function is fundamental for building a new hydrological model. The design concept of HydroModels.jl framework is akin to the deep learning framework. For instance, a *Flux* can be seen as a calculation layer of a neural network model, such as a linear layer (nn.linear) or an activation function (nn.Relu), both require data input and combine it with parameters to compute output results. The difference lies in the input data for Flux, which only supports one-dimensional vectors (single time point) and two-dimensional matrices (multiple time points with multiple variables). Similarly, an *Element* can be viewed as a calculation block in a neural network, such as the encoder and decoder block of the transformer model. By combining multiple elements, the hydrological model is constructed in a manner akin to the transformer model, resembling the assembly of building blocks.

To ensure ease of use, the construction style of the HydroModels.jl framework draws on Flux.jl and Lux.jl, two Julia-based deep learning frameworks. The various types of construction methods are as follows:

Simple Flux represents a hydrological flux calculation formula, primarily based on the formula. The Simple Flux is constructed according to the inputs, outputs, and parameters of the formula. Taking the percolation calculation formula in the GR4J model as an example:



The formula includes the input flux, soil moisture storage (slw), and the parameter *x1*, and the output flux is percolation (perc). Based on this information, a Simple Flux can be constructed as shown in the diagram:

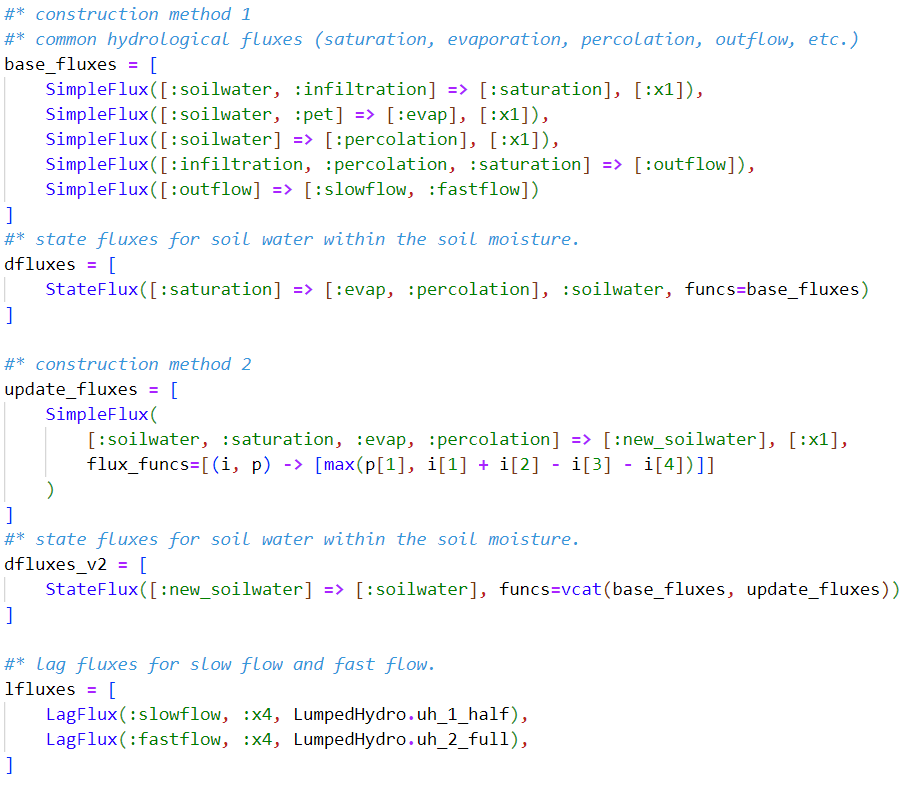


The diagram presents two methods for constructing a SimpleFlux to express the percolation calculation formula. The first method uses the Symbol type to represent the slw and perc fluxes, where "=>" indicates the input-output relationship between slw and perc. The second argument is the parameter required for the formula, such as x1. The flux\_funcs input value is of function type, where arguments include input fluxes and parameters, and the return value is the calculated output fluxes, all in vector type, with the order matching the previously defined arguments. The second method relies on the ModelingToolkit.jl and Symbolic.jl. These packages define slw and perc as fluxes over time t and define the parameter x1 with a default value of 0.0. Using these defined variables, a calculation equation is constructed, which is then used as the keyword argument: flux\_exprs for the Simple Flux. The resulting flux funcs are generated by Symbolic.jl similarly to the first construction method, with other input arguments remaining analogous[[1]](#footnote-1).

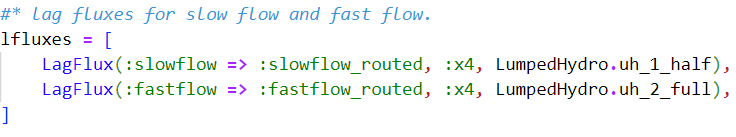
Additionally, HydroModels.jl can pre-construct a function library of conventional hydrological flux calculation formulas, similar to MARRMoT. These are categorized by output flux type, and matched to the corresponding calculation formula based on the input flux names and parameter names (constructing a generic type with flux and parameter names, then utilizing Julia's multiple dispatch feature to match the corresponding calculation formula).

State Flux represents the update formula for state fluxes in the hydrological module. There are two construction methods based on Symbol, as shown in the Figure: one of the method is constructing based on influxes and outfluxes, the "=>" symbol indicates the influx (saturation) and the outfluxes (evaporation and percolation), and the second argument represents the state flux, with its default calculation method corresponding to Eq 1.2. The another method is replacing the original state value with a new state value, where the "=>" symbol indicates the update and replacement relationship between new soil water and original soil water. Both methods require the keyword argument funcs for calculating influxes and outfluxes or new state fluxes.

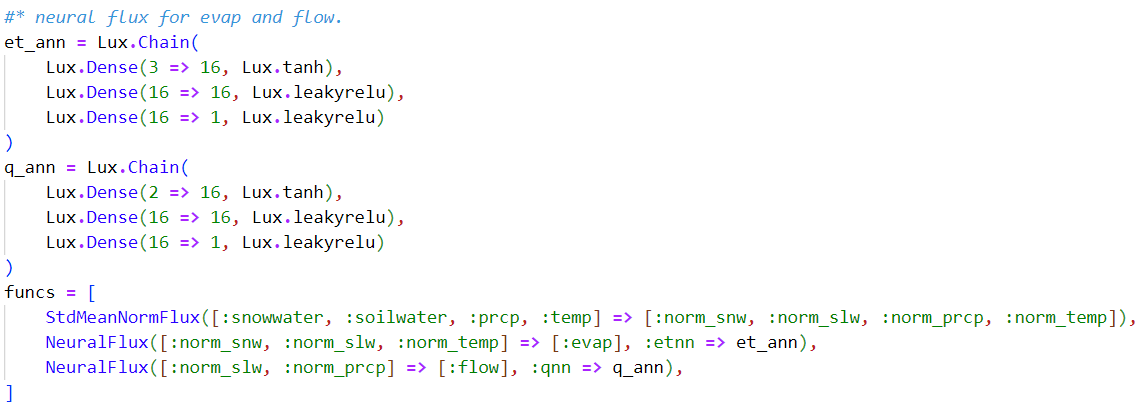
The two construction methods respectively calculate the change in state flux and the updated value. The former directly calculates the difference between input and output fluxes, which is suitable for most scenarios. The latter method is necessary when some state fluxes cannot exceed a certain limit, such as x1 in the diagram, requiring an additional custom update value calculation function.



Lag Flux represents the unit hydrograph calculation for flow routing, as depicted in the diagram. LagFlux requires specifying hydrological fluxes used for unit hydrograph calculation (slowflow and fastflow), the parameter x4 representing lag time, and specifying the unit hydrograph scheme: uh\_1\_half and uh\_2\_full. Subsequently, LagFlux constructs equations 1.3 to 1.5 as ordinary differential equations based on the given information. Then it utilizes a discrete solver to solve these equations and obtain the computed hydrological fluxes. For detailed construction methods, refer to the solve\_lag\_flux function in utils/unithydro.jl.

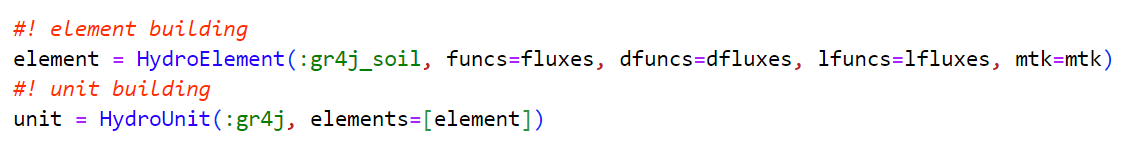


Neural Flux represents hydrological fluxes that require computation using neural networks, such as the implementation of the soil calculation module for M50 shown in the Figure. To build a NeuralFlux, the first step is to define the neural network structures used for flux computation. In the Figure, the et\_ann and q\_ann are neural networks constructed for evaporation and flow calculation. These networks have input and output dimensions consistent with the number of input and output fluxes specified in NeuralFlux, the neural network instances combined with their names is the second argument of NeuralFlux. During calculation and optimization, model parameters necessary for network computation are extracted based on these names.



HydroElement is a structure designed to integrate all aforementioned types of Flux. For the three different flux types: SimpleFlux, StateFlux, and LagFlux, HydroElement provides three input arguments (funcs, dfuncs, lfuncs) to store each flux type. After each flux is input into the element, the construct function extracts the element inputs, outputs, state flux names, and parameter names based on all fluxes, applying them as attributes for subsequent calculations. Additionally, HydroElement has two parametric composite types based on the two ODE function construction methods: HydroElement{false} and HydroElement{true}, corresponding to the conventional function construction and the modeling system construction based on ModelingToolkit.jl. For details on these methods and solution approaches, see section 2.2.2.

HydroUnit is a structure designed to integrate multiple types of components (including flux, element, and unit). It takes various components as input, then achieves unit integration calculation through the computation functions of each component.



2.2.2 Solving

The solution method is the core of model simulation calculations. Hydrological model frameworks such as superflexpy and MARRMoT provide solution methods to solve ordinary differential equations within hydrological models. However, HydroModels.jl does not provide any solvers for ODE problem-solving. Instead, it relies on Julia's mature and robust ecosystem for scientific computing, primarily depending on DifferentialEquation.jl, which offers powerful, comprehensive, and efficient solvers capable of addressing ordinary differential, partial differential, and stochastic differential equations.

HydroModels.jl construct ODE problems based on the combination of fluxes to utilize the solver resources of DifferentialEquation.jl: After constructing the HydroElement, the various state fluxes are not yet integrated with other simple fluxes. Specifically, the input and output fluxes of state fluxes are usually results of secondary calculations. Therefore, before calculating the state flux, it is necessary to compute the input and output fluxes of the state fluxes, which are the input arguments: funcs. The problem construction method varies depending on the HydroElement generic type:

(1) HydroElement{true}

Step 1. Combine the state fluxes with other simple fluxes formulas to construct the ODE system, as detailed in utils/mtk.jl/build\_ele\_system;

Step 2. Construct interpolation functions using observed input data combined with time indices, then combine these interpolation functions and input fluxes formulas with the ODE system constructed in Step 1;

Step 3. Reassign system information based on input parameters and initial states to complete the construction of the current ODE problem.

(2) HydroElement{false}

When multiple fluxes are needed to calculate intermediate fluxes to satisfy state flux requirements, significant computational overhead is often incurred. In HydroModels.jl, intermediate variables calculated by various fluxes are detailed in subsequent calculations, but continuous storage and extraction from these intermediate variables can severely impact code performance. Therefore, HydroElement{false} constructs temporary functions for problem construction and solving based on state fluxes and simple fluxes, as follows:

Step 1. Traverse all simple fluxes in the funcs attribute of state fluxes and replace the intermediate states required for state fluxes calculation with the corresponding formulas of simple fluxes, iterating until the state fluxes formulas only contain element input fluxes;

Step 2. Based on the replaced formulas, construct state flux calculation functions using symbolic.jl's build function, with the inputs being the element's input fluxes and parameters;

Step 3. Construct input generation functions based on the order of function input fluxes and parameters, and input these into the corresponding state flux calculation functions to build temporary ODE functions.

For solving ODE problems, HydroModels.jl decouples the solver from the element. This package constructs a wrapper type based on DifferentialEquations.jl's solvers, storing additional key parameters required for the solve method of the solver.

HydroModels.jl provides ODESolver and DiscreteSolver for two scenarios: continuous (M50 and M100) and discrete (dPL-HBV, PINN) respectively.

After computing the state fluxes, HydroModels.jl stores the solved state fluxes along with input data in a NamedTuple. It then iterates through fluxes in sequence. During each iteration, it extracts the required data from the NamedTuple based on the flux's input flux names and parameter names. After computation, it stores the flux output flux names back into the NamedTuple, continuing until all fluxes have been computed. Finally, when lag fluxes are present, it uses NamedTuple to perform routing calculations for specified variables.

The computation process for Unit is similar, relying on NamedTuple for input to the element and storing computed results back into NamedTuple until all elements have been computed.

2.2.3 Optimizing

Parameter calibration in hydrological models aims to adjust model parameters iteratively using optimization algorithms combined with observed data, thereby improving the model's predictive accuracy. This process is crucial in hydrological research.

Based on differentiability, optimization algorithms can be categorized into non-gradient-based and gradient-based methods. Non-gradient-based algorithms, such as random search, Monte Carlo, Bayesian optimization, and evolutionary algorithms, do not rely on gradient information from the hydrological model. They are widely used in parameter calibration as they allow specifying optimization ranges for parameters.

Gradient-based optimization algorithms, on the other hand, utilize gradient information of the objective function and guide the search direction, examples include gradient descent, Adaptive Moment Estimation (Adam), Newton's Method, etc. These methods perform better when optimizing a large number of parameters and are commonly used in neural network training and preferred in PINN hydrology parameter optimization.

HydroModels.jl relies on the Optimization.jl package for parameter calibration, which provides various optimization algorithms based on the objective function. HydroModels.jl offers two optimization methods: param\_box\_optim and param\_grad\_optim, catering to box optimization and gradient optimization needs respectively. Both methods require constructing an objective function: combining input components with data and algorithm-generated parameters to compute simulation results, then calculating simulation losses based on target flux names and loss functions.

In parameter calibration, conventional conceptual hydrological models often start with non-gradient optimization. After obtaining an optimal parameter combination, gradient-based methods refine parameters further, transitioning from global to local optimization. In the case of PINN hydrology, such as with the M50 model, parameters are first optimized using non-gradient methods for the Exp-hydro model. Subsequently, the optimal model simulation results are used to train a neural network model, which is then integrated into the coupled M50 model and refined using gradient methods. Therefore, both non-gradient and gradient methods play indispensable roles in this process.

**Appendix**

For example, in the GR4J model, the hydrological fluxes fastflow and slowflow are calculated through unit hydrograph convolution to describe watershed runoff. *Lag Flux* firstly computes unit hydrograph weights, constructs an ODE function, and uses a discrete solver to derive the weight calculation results, The calculation steps of *Lag Flux* are as follows:

Step 1. calculate the weight and initial state of the uh based on the uh schemes, considering the lag time and the length of the uh (uh\_len)：



Step 2. update the state of the uh by multiplying the input data (*it*) with the uh weight:





In Equation 1.4, circshift denotes vector shifting method, with -1 indicating that uh\_state shifts forward by one index, replacing the last value with 0. In Equation 1.5, the updated uh\_state value at the next period, uh\_state', is obtained by adding the product of uh\_state and uh\_weight with the input at time t (*it*), and is used for updating the state in the subsequent period.

*Lag Flux* addresses the special calculation methods in hydrological modules: unit hydrograph (uh). The unit hydrograph represents a watershed's hypothetical unit response to a unit rainfall input.

Step 3. Based on the input data for each period, record the updated state values, then extract the first value from each state to obtain the computed result by the uh.



1. Note that all Flux subtypes have two construction methods: Symbol and Variable. The subsequent introduction will use Symbol as an example. [↑](#footnote-ref-1)