https://doi.org/10.5194/gmd-2023-118
Preprint. Discussion started: 3 July 2023

Geoscientific Model Development
Discussions
Open Access

RoGeR v3.0.3 – a process-based hydrological toolbox model in Python

Robin Schwemmle[1], Hannes Leistert[1], Andreas Steinbrich[1], Markus Weiler[1]

[1]Hydrology, Faculty of Environment and Natural Resources, University of Freiburg, Freiburg, Germany

Correspondence to: Robin Schwemmle (robin.schwemmle@hydrology.uni-freiburg.de)

Abstract. Although water availability and water quality are equally important for an effective water resources management, to date, a combined representation of soil water balance components and water quality components in Python is not available. The new RoGeR toolbox contains models that can be used for the quantification of hydrological processes, fluxes and stores, but also solute transport processes based on StorAge selection. This study presents the code structure and functionalities of RoGeR developed as a scientific model toolbox following defined open-source software guidelines. RoGeR uses five different computational back ends covering just-in-time compilation, parallelism and graphical-processing units that might be used for optimizing computational performance. We show that graphical-processing unit computing have the greatest potential to improve computation time and energy usage especially for large modelling experiments. A simple modelling experiment highlights the capabilities of the new RoGeR model toolbox. We simulated the soil water balance, stable water isotope (^{18}O) transport and corresponding travel time distributions through the soil of a grassland plot for a three-year period. Further development of RoGeR as a scientific software is possible and also needed due to the current limitations for a variety of process components and easily possible due to the open software architecture.

1 Introduction

The interplay between the water and solute mass balance (e.g. oxygen-18, chloride or nitrate) and its related flow and transport in the soil-vegetation-atmosphere interface plays an important role for the understanding of hydrologic systems (e.g. Benettin et al., 2017). However, measurements for their states and fluxes are neither in space nor in time ubiquitously available (Beven, 2011). Thus, soil hydrological models, soil-vegetation-atmosphere-transfer (SVAT) models and distributed catchment models are indispensable tools to complement measurements (e.g. for a better process understanding) and to make predictions (e.g. future climate impacts, land cover changes or in ungauged catchments). Currently, there are many models in hydrology and the landscape of models is highly diverse (from simple conceptual models to complex physically-based models). One reason for this large and diverse landscape of models is that hydrologist still disagree about modelling concepts (Weiler and Beven, 2015). Despite the large number of models, however, there is a lack of reproducibility in computational hydrology (Hutton et al., 2016; Reinecke et al., 2022). Main reasons for this lack of reproducibility are poorly documented codes and workflows, code being too complex, code being not availability and input data not available (Reinecke et al., 2022).

Simulating the hydrological processes at the soil-vegetation-atmosphere interface including solute mass balance and transport with a high spatial and/or temporal resolution still requires long computation time. Thus, for reasons of computational performance hydrological models such as HYDRUS (Šimůnek et al., 2016), Daisy (Abrahamsen and Hansen, 2000), HYPE (Lindström et al., 2010), EcH2O-iso (Kuppel et al., 2018) or mHM (Heße et al., 2017; Kumar et al., 2020; Kumar et al., 2013; Samaniego et al., 2010) are written in low-level programming languages such as Fortran or C++. However, these languages are hard to read, to learn and are usually not included in the curriculum of hydrology-related degree programs. By contrast, high-level programming languages are easier to read and to learn, but computation takes about 3-5 times longer than equivalent code in low-level programming languages (Häfner et al., 2021). Therefore, high-level programming languages have the potential to foster reproducibility. Recently, high-level open-source programming languages such as R or Python gained popularity in the hydrological modelling community. Especially Python is currently the most popular programming language among software users (e.g. IEEE Spectrum, 2022; PYPL, 2022; Stack Overflow, 2021). Hydrological models quantifying the hydrological cycle that are written in Python, for example, are SUPERFLEX (Dal Molin et al., 2021; Fenicia et al., 2011), CWatM (Burek et al., 2020) and UniFHy (Hallouin et al., 2022) but none of these models consider transport of solutes and they generally focus at the catchment scale. To date, only rsas (Harman, 2015) implemented a solute transport model written in Python. However, rsas does not quantify the water balance and requires hydrological fluxes as input.

For reasons of longer computation times, high-level programming languages are often avoided in spatially distributed hydrological models. One solution to reduce computation time in high-level programming languages is using a just-in-time compiler (JIT). However, Python does not contain a built-in JIT. Instead, Python requires program libraries such as Numba (Lam et al., 2022) or JAX (Bradbury et al., 2018). However, Numba and JAX provide the opportunity to run the code on graphical processing units (GPUs) to decrease computation times. Veros (Häfner et al., 2018; Häfner et al., 2021), an ocean model written in Python using JAX for acceleration, demonstrated that GPU computations are a competitive alternative to central processing units (CPUs). In addition to that, Häfner et al. (2021) could show that GPU computations saves energy.

The first model version of RoGeR had a focus on the event-based runoff generation (Steinbrich et al., 2021). Therafter, RoGeR had been further developed and by adding a routing scheme, surface runoff and subsurface runoff contributions to flooding events could be explicitly simulated (Steinbrich et al., 2021). Additionally, by considering snow hydrological processes, urban hydrological processes and redistribution processes such as evapotranspiration enabled the estimation of the long-term water balance (Steinbrich et al., 2021). Based on the previous development efforts of the RoGeR model by Weiler (2005), Steinbrich et al. (2016) and Steinbrich et al. (2021), we reimplemented the process-based hydrological model RoGeR in a modular software architecture (e.g. different hydrological processes are implemented in separate modules that can be independently modified) written in Python. Since RoGeR had no implementation for solute transport so far, we include solute transport based on StorAge selection (SAS) functions (e.g. Benettin et al., 2017). We choose a high-level programming language and a modular software paradigm to foster reproducibility and wide-range application in teaching and research. In particular, we aim to facilitate general code understanding, writing new code and debugging code which usually takes most of the time within software projects. To overcome limitations on computational performance, we include the program library JAX.

In the following, we describe the implementation of the new model developed as a scientific software following open-source guidelines. Thereafter, we provide a brief overview about the representation of the hydrological processes and the related solute transport. We further profile the computational performance and energy usage. Finally, we demonstrate the capabilities of the model by simulating a three-year period for a synthetic site.

2 Implementation

2.1 RoGeR as a scientific open-source software

For the development of RoGeR as scientific open-source software, we followed the guidelines presented in Table 1. We defined these guidelines based on van Gompel et al. (2016) and Hall et al. (2022), and on reviewing earth science related software written in Python (e.g. Bakker et al., 2016; Bartos, 2020; Burek et al., 2020; Collenteur et al., 2019; Dal Molin et al., 2021; Häfner et al., 2018; Hallouin et al., 2022; Helmus and Collis, 2016; Kratzert et al., 2022; Mälicke, 2022; May et al., 2022; Rose, 2018; Schwemmle et al., 2021). We suggest that different software concepts might be applied depending on the software complexity. Moreover, including these guidelines in the curriculum of hydrology-related degree programs may lay the foundation for a reproducible future in computational hydrology.

2.1.1 Software architecture

The basic modular structure of the software is adapted from Häfner et al. (2018). The core modules implement hydrological processes and solute transport. As such, these modules represent a toolbox which can be used to build pre-defined models (e.g. a SVAT model by considering only vertical processes). We already provided some pre-defined models but in general new models can be easily assembled and combined to the level of complexity that is required. Moreover, further processes might be added by writing new modules. In addition to that, further modules are available for the pre-/post-processing, writing the model output and handling computational back ends. RoGeR is pure Python, hence not all computational bottlenecks might be solvable. In such cases, we recommend writing extensions using Cython instead of using a low-level language which would require a compiler.

2.1.2 Computational back-ends

The computations are handled by five different back-ends which are implemented through a function decorator (Häfner et al., 2018). Users have to choose a suitable backend beforehand. The choice depends on programming skills, size of the

Table 1 Guidelines for scientific open-source software in computational hydrology

Term/Concept	Description	Advantage
Comments	Write meaningful and clear comments within the source code	Comprehensive and reusable source code
Public access	Store the source code in an online-repository and use version control to enable collaborative development	Transparent and accessible
License	Regulates usage of the software	Reusable
Versioning	Assigns a new version number after each release (i.e. update of the source code)	Traceable
Software environment	Use a virtual environment to avoid software conflicts and provide information about software dependencies	Quick installation
Logger	Captures errors, warnings and progress of computation	Facilitates debugging
Hands-on-tutorials	Guidance on the application of the software for real case examples	Learnable and lower entry bar for new users
Online documentation	Contains essential information for installing, using the software and theoretical background (e.g. equations)	Learnable and understandable
Unit tests	Tests basic functionality of the software	Facilitates maintenance
Continuous integration	Runs unit tests on different operating systems and different software stack	
Profiling	Measures computation time (e.g. of individual modules) or memory usage (e.g. GPU memory)	Supports efficient allocation of computational resources

modelling experiment and available computational resources. In the following, we briefly describe the back ends and give recommendations on the usage:

- numpy: This back-end uses NumPy (Harris et al., 2020) for computation and, hence, is easy to use. However, the interpreted execution of the code and running computations on a single CPU may cause performance limitations. We recommend this back-end to beginners and for small-scale modelling experiments. As long as the modelling experiment fits in the memory, there are no specific requirements for the computational resources.
- numpy-mpi: The numpy-mpi back-end parallelizes the numpy back-end via mpi4py (Dalcin et al., 2011). The size of the modelling experiment might be limited by available memory and number of CPU cores. We recommend this back-end to users with experience in parallelized computations.
- jax: The jax back-end is the same as numpy but code is JIT compiled via JAX (Bradbury et al., 2018). Since JAX transforms NumPy code, it is required that all code is NumPy compatible. The JIT compilation leads to decreasing computation time (see Sect. 3).
- jax-mpi: Same as numpy-mpi but code is JIT compiled via mpi4jax (Häfner and Vicentini, 2021). This leads to computational speedup (see Sect. 3).
- jax-gpu: The code is JIT compiled and computations are performed on GPU which leads to computational speedup (see Sect. 3). The jax-gpu backend requires an appropriate GPU. The size of the modelling experiment is limited to available to GPU memory. We recommend this back-end to users with advanced programming skills.

2.1.3 Data handling

RoGeR requires input data for the following variables:

- precipitation at up to 10 minutes time steps
- air temperature at daily time steps
- potential evapotranspiration at daily time steps
- solute concentrations at daily time steps (only if solute transport is simulated)

The input data can be either provided as NetCDF files (.nc) or text files (.txt). If input data is provided as text files, the data is internally converted to NetCDF.

Metadata (e.g. units, description) for all variables and constants are defined in single modules as dictionaries (Häfner et al., 2018). From these dictionaries, metadata (e.g. units) is automatically added to the model output data. All model output is written to NetCDF files. A major advantage of the NetCDF format is that I-O operations enables parallel writing with compression (Häfner et al., 2018). This reduces time of I-O operations and size of output files.

For 1D models (i.e. no lateral transfer), space can be represented either through grid cells or polygons. By contrast, 2D models (i.e. lateral transfer between grid cells) require a regular grid as spatial representation. In order to generate physically meaningful results, we recommend a spatial resolution between 0.25 m^2 and 25 m^2.

2.2 Hydrological model

Different hydrological processes are implemented as modules. In the following, we list the already implemented processes and refer to the module and declare whether the module is tested or testing is still ongoing:

- Surface water storage (*surface.py*; testing is complete)
- Soil water storage (*soil.py*; testing is complete)
- Root zone water storage (*root_zone.py*; testing is complete)
- Subsoil water storage (*subsoil.py*; testing is complete)
- Groundwater water storage (*groundwater.py*; testing is ongoing)
- Transpiration (*evapotranspiration.py*; testing is complete)
- Soil evaporation (*evapotranspiration.py*; testing is complete)
- Interception (*interception.py*; testing is complete)
- Snow accumulation/Snow melt (*snow.py*; testing is complete)
- Infiltration driven by capillary forces (*infiltration.py*; testing is complete)
- Infiltration driven by gravitational forces (*film_flow.py*; testing is ongoing)
- Surface runoff (*surface_runoff.py*; testing is ongoing)
- Lateral subsurface runoff (sub*surface_runoff.py*; testing is ongoing)
- Lateral groundwater flow (groundwater*_flow.py*; testing is ongoing)
- Percolation (sub*surface_runoff.py*; testing is complete)
- Capillary rise (*capillary_rise.py*; testing is complete)
- Crop phenology (*crop.py*; testing is ongoing)

The modular structure enables the combination of various hydrological processes and thus building a certain model structure. The most basic model structure is shown in Figure 1 and is the basis for more complex model structures. We pre-defined further model structures by adding further hydrological processes (e.g. lateral subsurface flow, crop phenology). For more details about the pre-defined model structures, we refer to the online documentation of RoGeR (Schwemmle, 2023).

RoGeR provides representations for bucket-type interception, degree-day based snow accumulation and snow melt (LARSIM-Entwicklergemeinschaft, 2021), soil matrix, macropore and shrinkage crack infiltration (Steinbrich et al., 2016; Weiler, 2005), soil evaporation (Or et al., 2013), vegetation phenology and vegetation-specific transpiration (Steduto et al., 2009), capillary rise from a groundwater table and percolation to the groundwater (Salvucci, 1993) and lateral subsurface runoff (Steinbrich et al., 2016; Stoll and Weiler, 2010). For detailed information (e.g. model equations), we refer to the online documentation of RoGeR (Schwemmle, 2023).

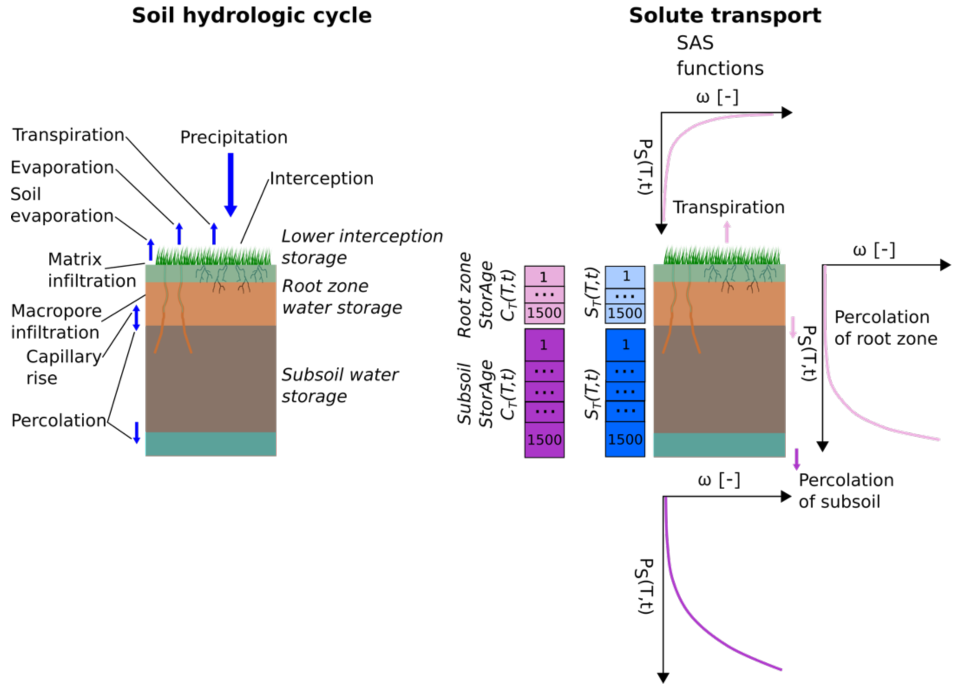

Figure 1 Conceptual implementation of the soil hydrologic cycle and solute transport processes shown for a single grid cell or unit. Water storages are represented in italic.

2.3 Solute transport model

Solute transport is implemented by a travel-time based approach. Particularly, we use StorAge selection (SAS) functions (Rinaldo et al., 2015). SAS is implemented by specific distribution functions. We assign a distribution function to each hydrological process (Figure 1). Here, we introduce two distribution functions which can be used for SAS and are implemented in the toolbox. The first distribution function is based on a power law and requires only a single parameter k_Q (Fig. 2a). The power law distribution function is given as:

$$\omega_Q(T,t) = {k_Q}^{k_Q} \cdot P_s(T,t)^{(k_Q-1)} \quad (1)$$

with

$$P_S(T,t) = \frac{S_T(T,t)}{\mathrm{S(t)}} \quad (2)$$

and the corresponding cumulative power law distribution function:

$$\Omega_Q(T,t) = P_s(T,t)^{k_Q} \quad (3)$$

where S_T(T,t) is the cumulative age-ranked storage (mm), S(t) is the soil water content (mm) and P_S(T,t) is the cumulative probability distribution of the storage.

https://doi.org/10.5194/gmd-2023-118
Preprint. Discussion started: 3 July 2023

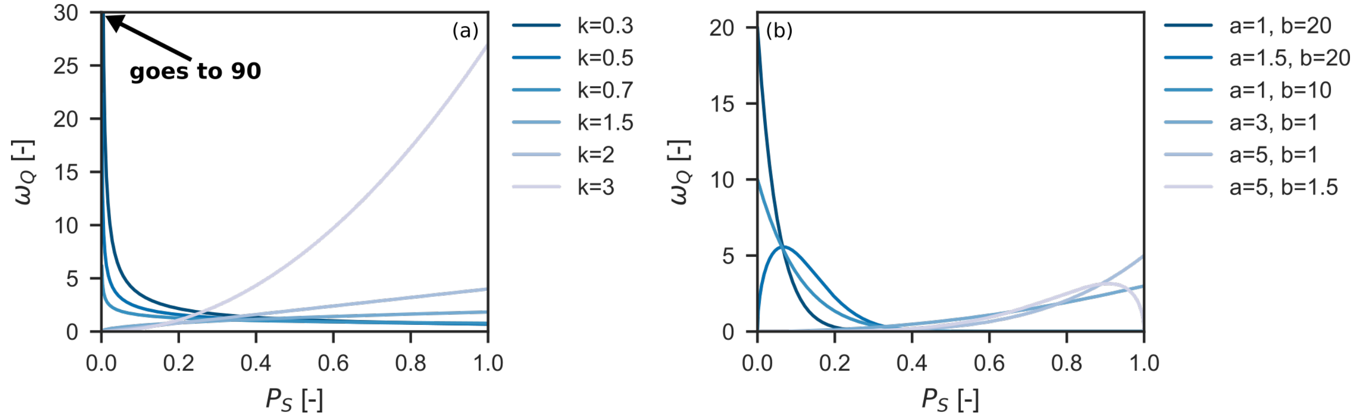

Figure 2 Storage selection with different parameters illustrated for power law distribution function (a; see Eq. (1)) and Kumaraswamy distribution function (b; see Eq. (3))

As a second distribution function, we employ the Kumaraswamy distribution (Kumaraswamy, 1980). With two parameters a_Q and b_Q, the Kumaraswamy distribution provides a greater flexibility than a power law distribution (Fig. 2b). The Kumaraswamy distribution function is formulated as:

$$\omega_Q(T,t) = a \cdot b \cdot P_s(T,t)^{(a_Q-1)} \cdot \left(1 - P_s(T,t)^{a_Q^{(b_Q-1)}}\right) \tag{4}$$

and the corresponding cumulative Kumaraswamy distribution function:

$$\Omega_Q(T,t) = 1 - \left(1 - \left(P_s(T,t)\right)^{a_Q^{b_Q}}\right) \tag{5}$$

Generally, any distribution function might be used as long as a closed form (i.e. probabilities integrates to one) is available (Harman, 2015). We apply the fractional SAS function type (fSAS; van der Velde et al., 2012) and solve the SAS equations for each hydrologic flux Q. To solve the SAS functions, we provide three numerical schemes: (i) deterministic (i.e. solving SAS equations for each flux in a sequential order), (ii) explicit Euler and (iii) explicit Runge-Kutta fourth-order. Transport processes can be defined for conservative and non-conservative solutes:

- Stable water isotopes oxygen-18 (^{18}O) and deuterium (^{2}H): Isotopic fractionation is not yet considered.
- Bromide and chloride: Evapoconcentration, sorption processes and partitioning of root uptake are included.
- Nitrate: Biogeochemical processes denitrification (Kunkel and Wendland, 2012), nitrification, soil nitrogen mineralization and nitrogen uptake by crops are implemented.

Again, we refer to the online documentation of RoGeR for detailed information (Schwemmle, 2023). The following routines are implemented, and we refer to the module and declare whether the module is tested or testing is still ongoing:

- Solute transport and water ages (*transport.py* and *sas.py*; testing is complete)
- Nitrogen cycle (*nitrate.py*; testing is ongoing)

3 Test cases for continuous development, computational performance and energy usage

RoGeR uses unit tests and continuous integration to test and ensure technical functionality (see Table 1). Additionally, we use test cases for continuous development. The idea of these test cases is to guarantee predictive consistency and to track advances in model development (i.e. comparison between model versions). We run the test cases with model parameters that cover a wide range of common parameters and perform simulations with different input data. In contrast to unit tests, the execution time is longer and depends on the number of time steps covered by the input data. The results (see Sect. S1) can be compared to future versions of RoGeR.

Table 2 Hardware specifications of computational benchmarks

	Notebook	Cluster node
CPU	Intel® Core™ i7 @ 2.60 GHz (four physical cores)	2 x Intel® Xeon® E5-2680v4 (Broadwell) @ 2.40 GHz (28 physical cores)
TDP[1] of CPU	45 Watt	280 Watt
RAM	8 GB DDR3	128 GB DDR4
GPU	-	Nvidia Tesla K80 (12 GB GDDR5 memory)
TDP[1] of GPU	-	300 Watt
Software stack	GNU 8.1, Open MPI 4.1.3, HDF5 1.12.2, roger 3.0	GNU 9.2, Open MPI 4.1.3, HDF5 1.12, CUDA 11.4, roger 3.0
PUE[2]	1	1.31

[1] Total power draw
[2] Power usage efficiency

High-level programming languages such as Python still have the reputation of being comparatively slow. We profiled the computation time and energy usage using the five back-ends (see Sect. 2.1.2). For the profiling, we used two different hardware specifications representing commonly available computing resources and high-performance computing (HPC) resources (Table 2). We measured computation time and energy usage with a fixed number of iterations, but varying number of grid cells (Fig. 3).

Model parameters are the same for each grid cell. Figure 3 shows that for small modelling experiments (< 1000 grid cells), the numpy back-end performs equally well than the other back-ends. Parallel computation improves computational speed up only for intermediate to larger modelling experiments (> 1000 grid cells), provided that a greater number of CPU cores are available. Computation on a single GPU device is faster than on multiple CPUs for the RoGeR-SVAT type model while multiple CPUs (numpy-mpi and jax-mpi) are faster than a single GPU device for the RoGeR-SVAT-^{18}O type model. However, a major requirement for GPU computing is that the modelling experiment fits into the GPU memory (< 10^6 grid cells). A solution to the memory limitation would be the usage of multiple GPU devices.

HPC consumes more energy than running computations on a notebook. Depending on the energy source, HPC contributes differently to climate warming (Lannelongue et al., 2021). In order to raise awareness about the energy usage in HPC context and to provide information for a sustainable allocation of computational resources, we profiled the energy usage of RoGeR in an HPC context (see Table 2). Based on the profiling of computation time, we calculated the energy usage of the five back-ends using the method proposed by Lannelongue et al. (2021). The results (Fig. 4) show that using multiple CPUs (numpy-mpi and jax-mpi) consumes more energy than other back-ends. Using a single GPU device decreases energy usage while computation time still competes with multiple CPUs (cp. Fig. 3). For small and intermediate modelling experiments, single CPU (numpy and jax) back-ends use less energy than other back-ends. With these results, we aim to support efficient and sustainable allocation of computational resources. We suggest that computation time and energy usage should be considered equally for the allocation.

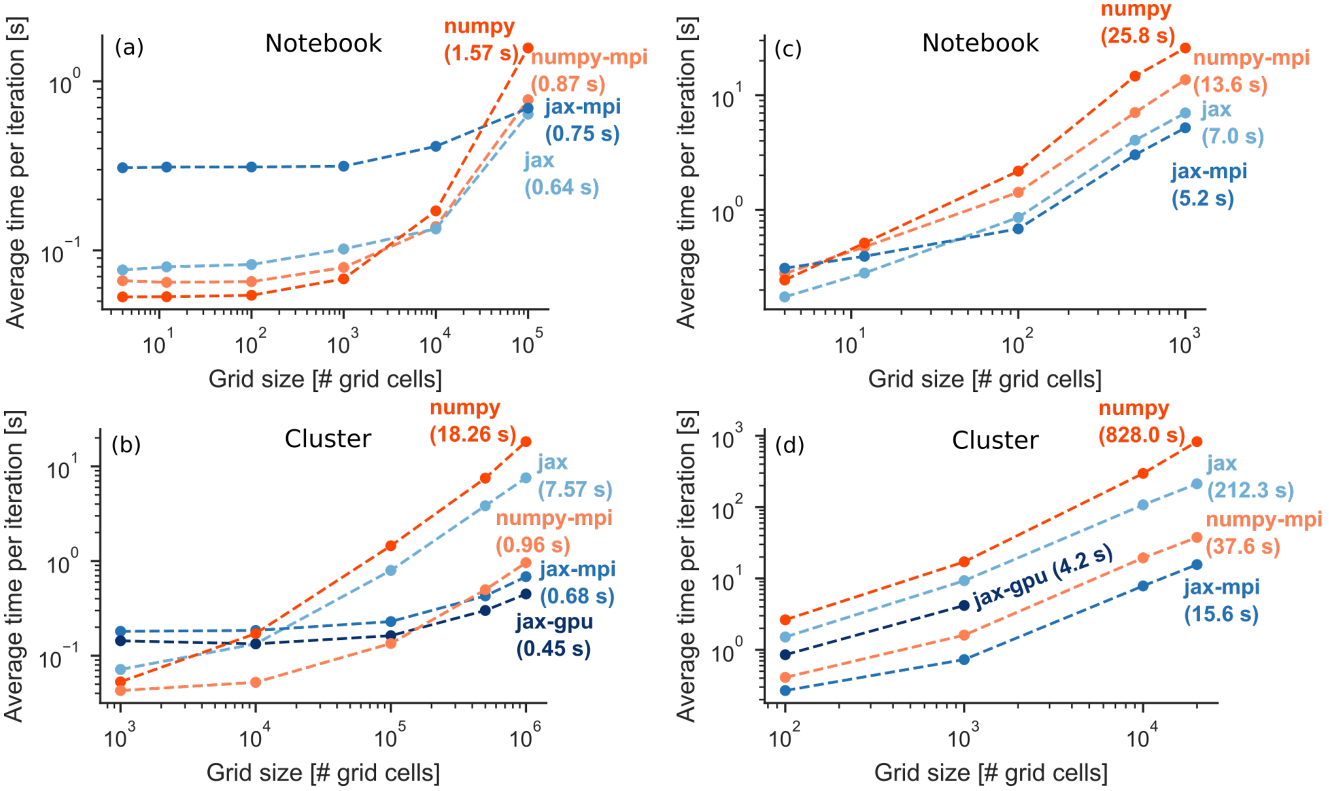

Figure 3 Runtime performance of computational back-ends for the RoGeR-SVAT type model (a, b) and for the RoGeR-SVAT-^{18}O type transport model (c, d). Note that, number of grid cells represents the two horizontal spatial dimensions (e.g. longitudes and latitudes). The total number of elements is greater for transport models due to additional age dimensions and can be derived by multiplying the number of grid cells (i.e. two spatial dimensions) with the number of water ages (e.g. 1500). SVAT model used 100 iterations and SVAT-^{18}O transport model used 20 iterations.

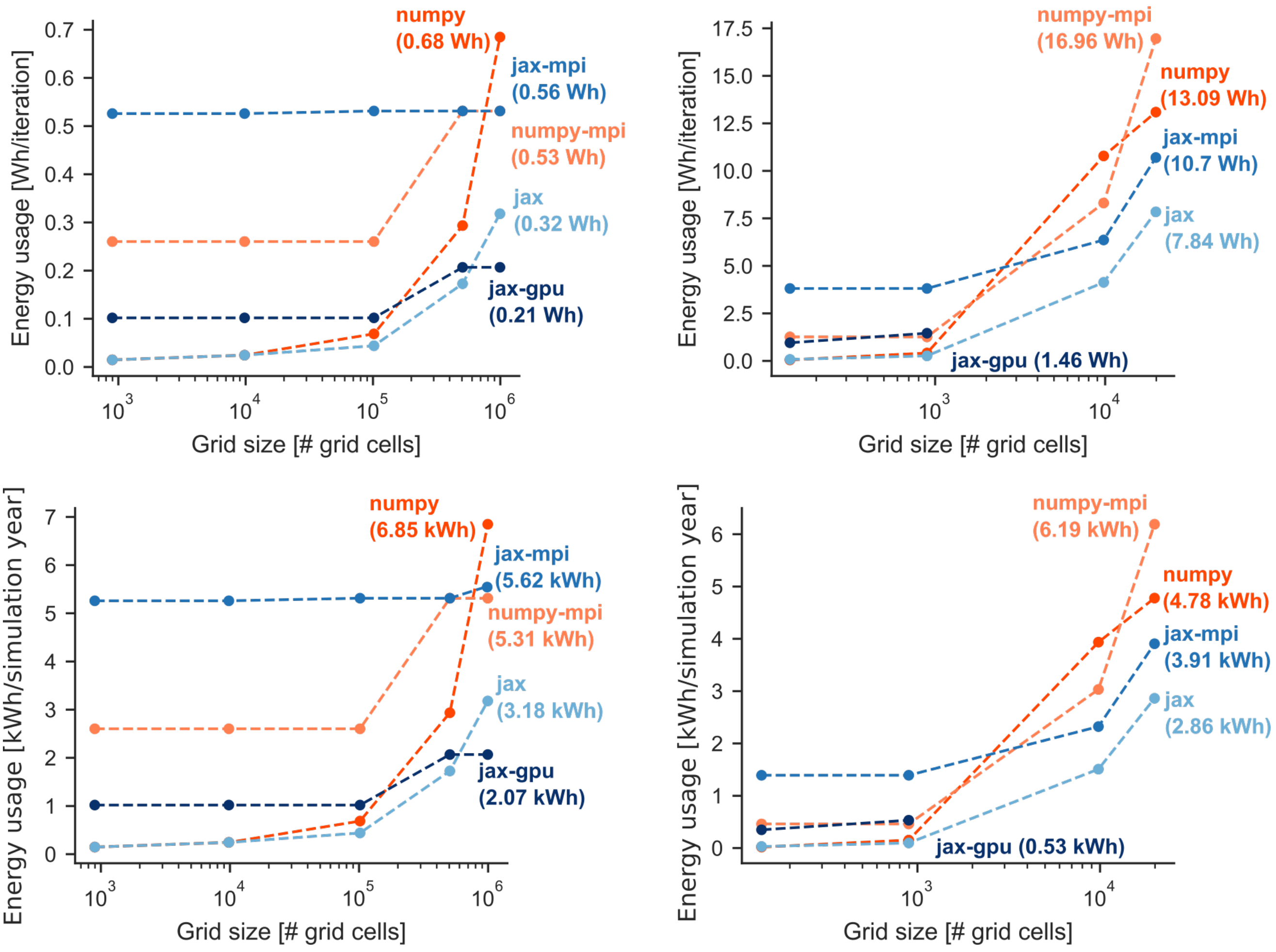

Figure 4 Energy usage of computational back ends on a cluster node for the RoGeR-SVAT type model (a, b) and for the RoGeR-SVAT-^{18}O type transport model (c, d)

4 Application: Soil water balance, ^{18}O transport and water age statistics of a three-year period

To demonstrate the capabilities of RoGeR, we present a simple application example. We simulate the soil water balance and fluxes and ^{18}O fluxes of a flat rectangular grassland (7200 m^2 with a resolution of 25 m^2) for a time period of three years. The input data was retrieved from the database WeatherDB, which provides data from stations operated by Deutscher Wetterdienst (DWD) tailored to the required format of RoGeR (Schmit, 2022). We selected the DWD station at Freiburg airport (station ID: 1443) to obtain precipitation, air temperature and potential evapotranspiration data from November 2019 to October 2022. Since DWD stations do not measure solute concentrations in precipitation, data for ^{18}O in precipitation has been generated by a sinusoidal function with random variation for amplitude and offset (Allen et al., 2018; amplitude=4.3±0.5 [‰], offset=-10±0.5 [‰] and phase=60 [days]). In order to represent the local soil characteristics and their spatial heterogeneity, we randomly generated model parameters for each grid cell with typical parameter ranges provided in Table 3. Additionally, we assumed a deep groundwater table implemented through a high hydraulic conductivity of the bedrock (see Table 3). SAS parameters for the selected power law distribution function are assumed to be spatially and temporally constant for each hydrological process and grid cell. We assigned k=0.1 to soil evaporation and capillary rise, k=0.3 to transpiration, k=2 to percolation of root zone and k=3 to percolation of subsoil. Thus, soil evaporation capillary rise, and transpiration have a preference for younger water, while percolation processes have a preference for older water (see Fig. 2a).

In Figure 5, we display the time series of hydrologic fluxes and soil water content with the corresponding ^{18}O signature and water age distributions of a single grid cell. The temporal pattern exhibits that soil water content and travel times of hydrologic fluxes can be related. This pattern emphasizes the interlinkage between hydrologic states and transport velocities of solutes (Hrachowitz et al., 2016). Figure 6 shows the cumulative distributions of soil hydrologic fluxes, soil water content, δ^{18}O signals and median water ages at four different dates with different soil water content conditions. Soil water content is wetter at 9th December 2021 and drier at 8th August 2022 while the other two dates represent the transition between drier and wetter conditions. The cumulative distributions of δ^{18}O signals and median water ages reveal differences for these different soil water content conditions. The δ^{18}O signals display distinct differences between the considered fluxes and soil water storage. Moreover, spatial variation is more distinct for soil water storage and percolation than for soil evaporation and transpiration. The median water age exposes a more general pattern. For drier conditions, median water age is older, whereas for wetter conditions median water age decreases.

Table 3 Lower and upper boundaries used for random generation of parameter grids

Hydrological model parameter	Symbol	Unit	Parameter boundaries
Land use/Land cover	lu_id	-	grassland
Density of vertical macropores	ρ_{mpv}	m^{-2}	50 - 100
Length of vertical macropores	l_{mpv}	mm	200 - 500
Soil depth	z_{soil}	mm	1000 - 1200
Air capacity of soil	θ_{ac}	-	0.08 - 0.12
Plant available field capacity of soil	θ_{ufc}	-	0.08 - 0.12
Permanent wilting point of soil	θ_{pwp}	-	0.18 - 0.22
Saturated hydraulic conductivity of soil	k_s	mm h^{-1}	5 - 15
Hydraulic conductivity of bedrock	k_f	mm h^{-1}	2500[a]

[a] results in free drainage

The primary objective of the example is to demonstrate the capabilities of RoGeR. Therefore, we kept the complexity of the example at a simple level. Although a comparison between simulations and observations is important to evaluate fidelity of the model, we do not provide such a comparison here. Instead, we refer to Schwemmle and Weiler (2023) for an in-depth evaluation of RoGeR using measurements from a grassland lysimeter site. Since development as scientific software started recently and is still ongoing, further evaluation of RoGeR will be addressed in the future.

The simple application example demonstrates the potential of RoGeR for a combined quantification of the water balance and solute mass balance. The example focusses on vertical soil hydrological processes and a conservative tracer, but this is just an excerpt of the toolbox. Other processes (e.g. lateral subsurface runoff, different SAS function; see Sect. 2.2 and 2.3) or other tracers (e.g. bromide; see Sect. 2.3) could also be considered and implemented.

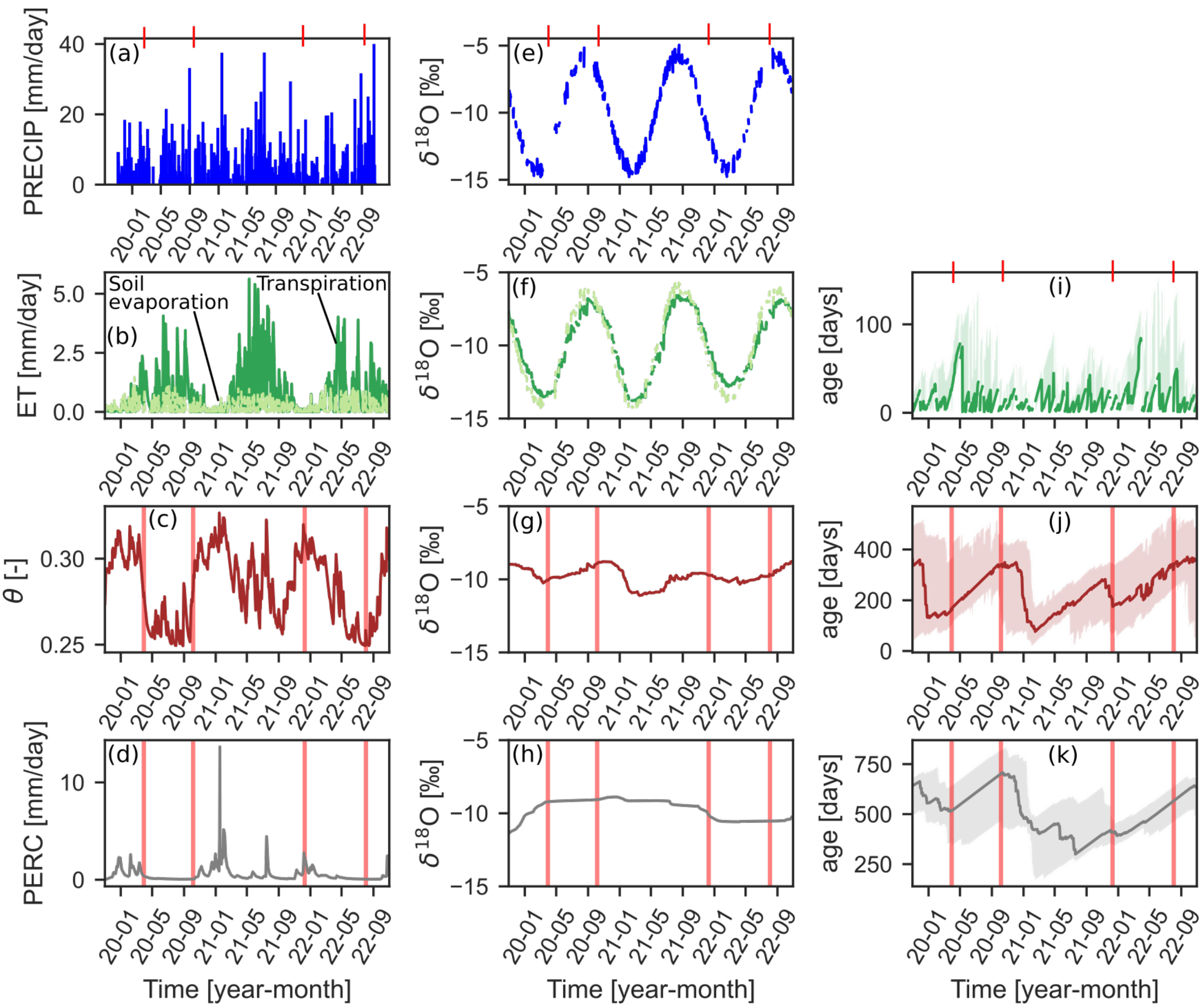

Figure 5 Simulated fluxes and soil water content (a-d), corresponding δ^{18}O signal (e-h) and corresponding 25th, 50th and 75th percentile of water ages (i-k) of a single grid cell. Vertical red lines indicate the four different dates from Figure 6. Power law distribution function serves as SAS function (SAS parameters: $k_{evap\text{-}soil}$=0.1, k_{transp}=0.3, $k_{perc\text{-}rz}$=2, $k_{perc\text{-}ss}$=3; see Figure 2)

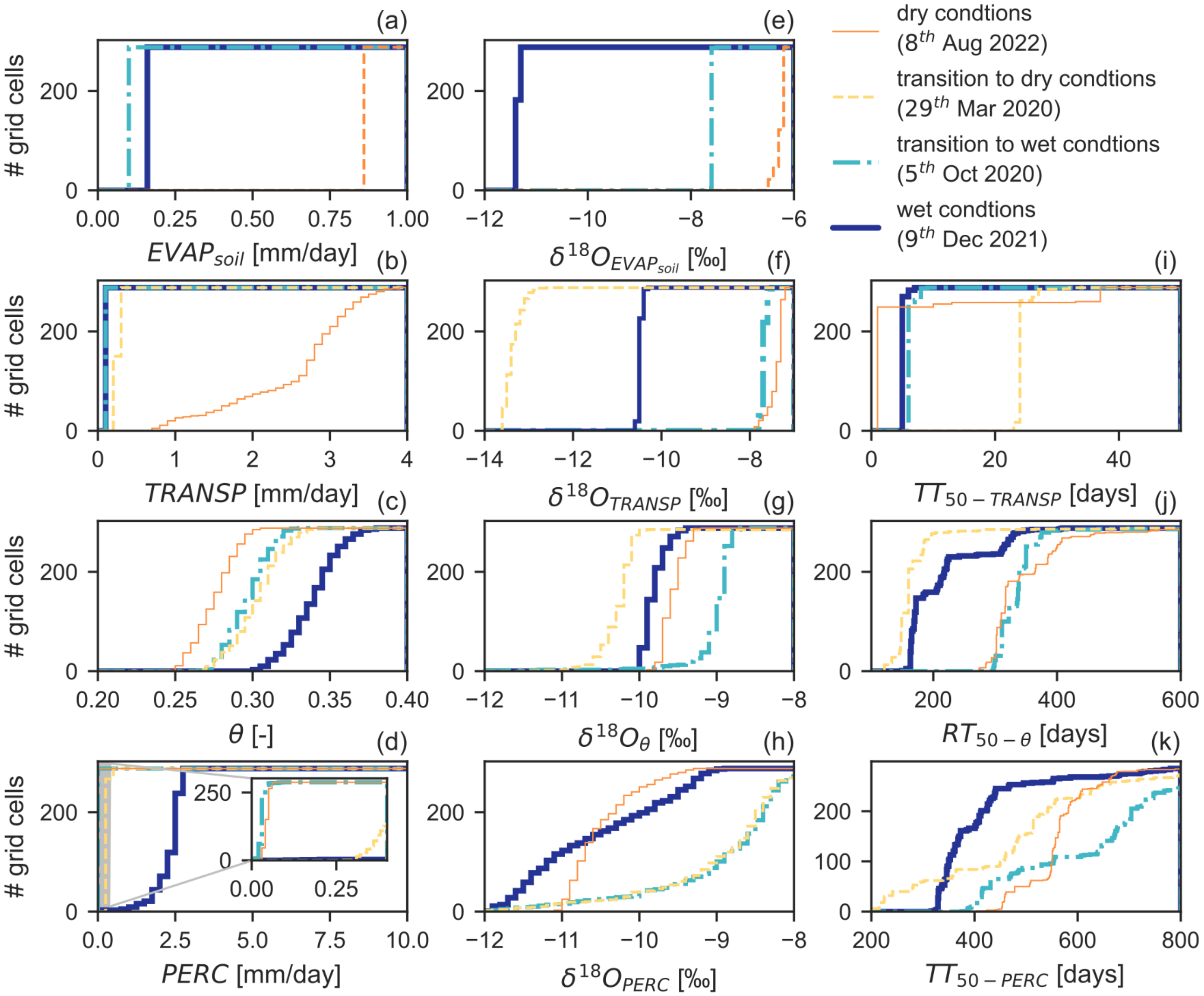

Figure 6 Cumulative distributions of simulated fluxes and soil water content (a-d), corresponding δ^{18}O signal (e-h) and corresponding median travel time median residence time (i-k) of a rectangular grassland (7200 m^2) at four different dates (transition to dry, dry, transition to wet and wet conditions)

285 5 Summary and outlook

The development of the process-based hydrological toolbox RoGeR followed open-source software guidelines (Sect. 2.1). We believe that such guidelines improve the reproducibility in computational hydrology. With the modular code structure (Sect. 2.1.1) and the good readability of Python code, RoGeR is intended to be easy to use (i.e. usable by programmers with little experience) and to be easy to modify (i.e. modification and extension of the code). With using different computational 290 back-ends, we maintained code readability without hampering computational performance (Sect. 2.1.2). The five back-ends provide the opportunity to simulate anything between plot scale and the catchment scale with reasonable computation times. Especially, the GPU back end has great potential to reduce computation time and energy usage of catchment scale modelling experiments (Sect. 3).

In comparison to the publicly available hydrological models written in Python, we combined hydrological processes (Sect. 2.2) and solute transport based on SAS (Sect. 2.3). The combined representation enables the prediction of hydrologic states and fluxes and their corresponding solute concentrations including travel times. The simple application example considering the water balance and ^{18}O transport through the soil of a rectangular grassland showed plausible results. The RoGeR toolbox contains many processes to describe one dimensional hydrological processes (i.e. no lateral transfer between grid cells). The implementation of the lateral transfer between grid cells (i.e. routing schemes for surface and subsurface runoff) will be addressed in future releases. Moreover, we suggest that future work may improve or extend the currently available process representations (e.g. gravity-driven infiltration and percolation; Demand and Weiler, 2021; Germann and Prasuhn, 2018) and further evaluation of RoGeR with measured data may provide insights on the strengths and weaknesses.

RoGeR contributes to a further diversification of the hydrological model landscape and the disagreement about process representation in the hydrological modelling community will continue (Weiler and Beven, 2015). In general, an advantage of this diversification and disagreement is that many different approaches are available and, hence, a great flexibility to address different problems. On the other hand, the theoretical diversification is accompanied by technical diversification (e.g. different programming languages or different data formats) that lead to inconsistencies in the application. We suggest that the diverse hydrological model landscape might benefit from focussing on constrained data interfaces of the models following common data conventions (Hallouin et al., 2022) and implementing standardized model interfaces (Hut et al., 2022; Hutton et al., 2020). This would facilitate model inter-comparison and models could be integrated in earth science models.

Supplement. The supplement related to this article is available online at: https://doi.org/10.5194/gmd-2023-118-supplement

Code availability. The code is open-source and publicly available https://doi.org/10.5281/zenodo.8066302 and https://doi.org/10.5281/zenodo.8095094.

Data availability. The meteorological input data used in the application example has been retrieved from https://weather.hydro.intra.uni-freiburg.de/ (Schmit, 2022) and is available at https://github.com/Hydrology-IFH/roger/tree/main/examples/hillslope_scale/svat_distributed_tutorial.

Author contributions. MW conceived the idea of the hydrological model RoGeR. MW, HL, AS and RS conceptualized RoGeR. HL and AS developed the first model suites of RoGeR in Python. RS developed RoGeR as a software package with support from HL translating python code into a software package. RS wrote the first draft of the manuscript with contributions from all co-authors.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. The authors acknowledge support by the High Performance and Cloud Computing Group at the Zentrum für Datenverarbeitung of the University of Tübingen, the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 37/935-1 FUGG. We are grateful to the Veros development team for providing their software architecture, to the Python community for providing the useful tools and to everyone involved in the development of RoGeR.

Financial support. This research has been supported by Helmholtz Association of German Research Centres (grant no 42-2017). The article processing charge was funded by the Baden-Wuerttemberg Ministry of Science, Research and Art and the University of Freiburg in the funding programme Open Access Publishing.

References

Abrahamsen, P. and Hansen, S.: Daisy: an open soil-crop-atmosphere system model, Environmental Modelling & Software, 15, 313-330, https://doi.org/10.1016/S1364-8152(00)00003-7, 2000.

Allen, S. T., Kirchner, J. W., and Goldsmith, G. R.: Predicting Spatial Patterns in Precipitation Isotope (δ2H and δ18O) Seasonality Using Sinusoidal Isoscapes, Geophysical Research Letters, 45, 4859-4868, https://doi.org/10.1029/2018GL077458, 2018.

Bakker, M., Post, V., Langevin, C. D., Hughes, J. D., White, J. T., Starn, J. J., and Fienen, M. N.: Scripting MODFLOW Model Development Using Python and FloPy, Groundwater, 54, 733-739, https://doi.org/10.1111/gwat.12413, 2016.

Bartos, M.: pysheds: simple and fast watershed delineation in python, https://doi.org/10.5281/zenodo.3822494, 2020.

Benettin, P., Soulsby, C., Birkel, C., Tetzlaff, D., Botter, G., and Rinaldo, A.: Using SAS functions and high-resolution isotope data to unravel travel time distributions in headwater catchments, Water Resources Research, 53, 1864-1878, https://doi.org/10.1002/2016WR020117, 2017.

Beven, K. J.: Rainfall-Runoff Modelling. The Primer., John Wiley & Sons, Chichester, England, 2011.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q.: JAX: composable transformations of Python+NumPy programs, available at: http://github.com/google/jax, (last access: 20 January 2023).

Burek, P., Satoh, Y., Kahil, T., Tang, T., Greve, P., Smilovic, M., Guillaumot, L., Zhao, F., and Wada, Y.: Development of the Community Water Model (CWatM v1.04) – a high-resolution hydrological model for global and regional assessment of integrated water resources management, Geosci. Model Dev., 13, 3267-3298, https://doi.org/10.5194/gmd-13-3267-2020, 2020.

Collenteur, R. A., Bakker, M., Caljé, R., Klop, S. A., and Schaars, F.: Pastas: Open Source Software for the Analysis of Groundwater Time Series, Groundwater, 57, 877-885, https://doi.org/10.1111/gwat.12925, 2019.

Dal Molin, M., Kavetski, D., and Fenicia, F.: SuperflexPy 1.3.0: an open-source Python framework for building, testing, and improving conceptual hydrological models, Geosci. Model Dev., 14, 7047-7072, https://doi.org/10.5194/gmd-14-7047-2021, 2021.

Dalcin, L. D., Paz, R. R., Kler, P. A., and Cosimo, A.: Parallel distributed computing using Python, Advances in Water Resources, 34, 1124-1139, https://doi.org/10.1016/j.advwatres.2011.04.013, 2011.

Demand, D. and Weiler, M.: Potential of a Gravity-Driven Film Flow Model to Predict Infiltration in a Catchment for Diverse Soil and Land Cover Combinations, Water Resources Research, 57, e2019WR026988, https://doi.org/10.1029/2019WR026988, 2021.

Fenicia, F., Kavetski, D., and Savenije, H. H. G.: Elements of a flexible approach for conceptual hydrological modeling: 1. Motivation and theoretical development, Water Resources Research, 47, https://doi.org/10.1029/2010WR010174, 2011.

Germann, P. F. and Prasuhn, V.: Viscous Flow Approach to Rapid Infiltration and Drainage in a Weighing Lysimeter, Vadose Zone Journal, 17, 170020, https://doi.org/10.2136/vzj2017.01.0020, 2018.

Häfner, D., Jacobsen, R. L., Eden, C., Kristensen, M. R. B., Jochum, M., Nuterman, R., and Vinter, B.: Veros v0.1 – a fast and versatile ocean simulator in pure Python, Geosci. Model Dev., 11, 3299-3312, https://doi.org/10.5194/gmd-11-3299-2018, 2018.

Häfner, D., Nuterman, R., and Jochum, M.: Fast, Cheap, and Turbulent—Global Ocean Modeling With GPU Acceleration in Python, Journal of Advances in Modeling Earth Systems, 13, e2021MS002717, https://doi.org/10.1029/2021MS002717, 2021.

Häfner, D. and Vicentini, F.: mpi4jax: Zero-copy MPI communication of JAX arrays, Journal of Open Source Software, 6, 3419, https://doi.org/10.21105/joss.03419, 2021.

Hall, C. A., Saia, S. M., Popp, A. L., Dogulu, N., Schymanski, S. J., Drost, N., van Emmerik, T., and Hut, R.: A hydrologist's guide to open science, Hydrol. Earth Syst. Sci., 26, 647-664, https://doi.org/10.5194/hess-26-647-2022, 2022.

Hallouin, T., Ellis, R. J., Clark, D. B., Dadson, S. J., Hughes, A. G., Lawrence, B. N., Lister, G. M. S., and Polcher, J.: UniFHy v0.1.1: a community modelling framework for the terrestrial water cycle in Python, Geosci. Model Dev., 15, 9177-9196, https://doi.org/10.5194/gmd-15-9177-2022, 2022.

Harman, C. J.: Time-variable transit time distributions and transport: Theory and application to storage-dependent transport of chloride in a watershed, Water Resources Research, 51, 1-30, https://doi.org/10.1002/2014WR015707, 2015.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E.: Array programming with NumPy, Nature, 585, 357-362, https://doi.org/10.1038/s41586-020-2649-2, 2020.

Helmus, J. J. and Collis, S. M.: The Python ARM Radar Toolkit (Py-ART), a library for working with weather radar data in the Python programming language, Journal of Open Research Software, 4, https://doi.org/10.5334/jors.119, 2016.

Heße, F., Zink, M., Kumar, R., Samaniego, L., and Attinger, S.: Spatially distributed characterization of soil-moisture dynamics using travel-time distributions, Hydrol. Earth Syst. Sci., 21, 549-570, https://doi.org/10.5194/hess-21-549-2017, 2017.

Hrachowitz, M., Benettin, P., van Breukelen, B. M., Fovet, O., Howden, N. J. K., Ruiz, L., van der Velde, Y., and Wade, A. J.: Transit times—the link between hydrology and water quality at the catchment scale, Wiley Interdisciplinary Reviews: Water, 3, 629-657, https://doi.org/10.1002/wat2.1155, 2016.

Hut, R., Drost, N., van de Giesen, N., van Werkhoven, B., Abdollahi, B., Aerts, J., Albers, T., Alidoost, F., Andela, B., Camphuijsen, J., Dzigan, Y., van Haren, R., Hutton, E., Kalverla, P., van Meersbergen, M., van den Oord, G., Pelupessy, I., Smeets, S., Verhoeven, S., de Vos, M., and Weel, B.: The eWaterCycle platform for open and FAIR hydrological collaboration, Geosci. Model Dev., 15, 5371-5390, https://doi.org/10.5194/gmd-15-5371-2022, 2022.

Hutton, C., Wagener, T., Freer, J., Han, D., Duffy, C., and Arheimer, B.: Most computational hydrology is not reproducible, so is it really science?, Water Resources Research, 52, 7548-7555, https://doi.org/10.1002/2016WR019285, 2016.

Hutton, E. W., Piper, M. D., and Tucker, G. E.: The Basic Model Interface 2.0: A standard interface for coupling numer ical models in the geosciences, Journal of Open Source Software, 5, 2317, https://doi.org/10.21105/joss.02317, 2020.

IEEE Spectrum: Top Programming Languages 2022, available at: https://spectrum.ieee.org/top-programming-languages-2022, (last access: 12 January 2023).

Kratzert, F., Gauch, M., Nearing, G., and Klotz, D.: NeuralHydrology — A Python library for Deep Learning research in hydro logy, Journal of Open Source Software, 7, 4050, https://doi.org/10.21105/joss.04050, 2022.

Kumar, R., Heße, F., Rao, P. S. C., Musolff, A., Jawitz, J. W., Sarrazin, F., Samaniego, L., Fleckenstein, J. H., Rakovec, O., Thober, S., and Attinger, S.: Strong hydroclimatic controls on vulnerability to subsurface nitrate contamination across Europe, Nature Communications, 11, 6302, https://doi.org/10.1038/s41467-020-19955-8, 2020.

Kumar, R., Samaniego, L., and Attinger, S.: Implications of distributed hydrologic model parameterization on water fluxes at multiple scales and locations, Water Resources Research, 49, 360-379, https://doi.org/10.1029/2012WR012195, 2013.

Kumaraswamy, P.: A generalized probability density function for double-bounded random processes, Journal of Hydrology, 46, 79-88, https://doi.org/10.1016/0022-1694(80)90036-0, 1980.

Kunkel, R. and Wendland, F.: Diffuse Nitrateinträge in die Grund- und Oberflächengewässer von Rhein und Ems - Ist-Zustands- und Maßnahmenanalysen, Forschungszentrum Jülich, Jülich, Germany, 143 pp., 2012.

Kuppel, S., Tetzlaff, D., Maneta, M. P., and Soulsby, C.: EcH2O-iso 1.0: water isotopes and age tracking in a process-based, distributed ecohydrological model, Geosci. Model Dev., 11, 3045-3069, https://doi.org/10.5194/gmd-11-3045-2018, 2018.

Lam, S. K., Pitrou, A., Florisson, M., Seibert, S., Markall, G., Anderson, T. A., Leobas, G., Collison, M., Bourque, J., Meurer, A., Oliphant, T. E., Riasanovsky, N., Wang, M., Pronovost, E., Totoni, E., Wieser, E., Seefeld, S., Grecco, H., Peterson, P., Virshup, I., Matty, G., Turner-Trauring, I., and Bourbeau, J.: numba/numba: Version 0.56.4, available at: https://doi.org/10.5281/zenodo.7289231, (last access: 20 January 2023).

Lannelongue, L., Grealey, J., and Inouye, M.: Green Algorithms: Quantifying the Carbon Footprint of Computation, Advanced Science, 8, 2100707, https://doi.org/10.1002/advs.202100707, 2021.

LARSIM-Entwicklergemeinschaft: Das Wasserhaushaltsmodell LARSIM: Modellgrundlagen und Anwendungsbeispiele, LARSIM-Entwicklergemeinschaft - Hochwasserzentralen LUBW, BLfU, LfU RP, HLNUG, BAFU, 258 pp., 2021.

Lindström, G., Pers, C., Rosberg, J., Strömqvist, J., and Arheimer, B.: Development and testing of the HYPE (Hydrological Predictions for the Environment) water quality model for different spatial scales, Hydrology Research, 41, 295-319, https://doi.org/10.2166/nh.2010.007, 2010.

Mälicke, M.: SciKit-GStat 1.0: a SciPy-flavored geostatistical variogram estimation toolbox written in Python, Geosci. Model Dev., 15, 2505-2532, https://doi.org/10.5194/gmd-15-2505-2022, 2022.

May, R. M., Goebbert, K. H., Thielen, J. E., Leeman, J. R., Camron, M. D., Bruick, Z., Bruning, E. C., Manser, R. P., Arms, S. C., and Marsh, P. T.: MetPy: A Meteorological Python Library for Data Analysis and Visualization, Bulletin of the American Meteorological Society, 103, E2273-E2284, https://doi.org/10.1175/bams-d-21-0125.1, 2022.

Or, D., Lehmann, P., Shahraeeni, E., and Shokri, N.: Advances in Soil Evaporation Physics—A Review, Vadose Zone Journal, 12, vzj2012.0163, https://doi.org/10.2136/vzj2012.0163, 2013.

PYPL: PYPL PopularitY of Programming Language index, available at: https://pypl.github.io/PYPL.html, (last access: 12 January 2023).

Reinecke, R., Trautmann, T., Wagener, T., and Schüler, K.: The critical need to foster computational reproducibility, Environmental Research Letters, 17, 041005, https://doi.org/10.1088/1748-9326/ac5cf8, 2022.

Rinaldo, A., Benettin, P., Harman, C. J., Hrachowitz, M., McGuire, K. J., van der Velde, Y., Bertuzzo, E., and Botter, G.: Storage selection functions: A coherent framework for quantifying how catchments store and release water and solutes, Water Resources Research, 51, 4840-4847, https://doi.org/10.1002/2015WR017273, 2015.

Rose, B. E.: CLIMLAB: a Python toolkit for interactive, process-oriented climate modeling, Journal of Open Source Software, 3, 659, https://doi.org/10.21105/joss.00659, 2018.

Salvucci, G. D.: An approximate solution for steady vertical flux of moisture through an unsaturated homogeneous soil, Water Resources Research, 29, 3749-3753, https://doi.org/10.1029/93wr02068, 1993.

Samaniego, L., Kumar, R., and Attinger, S.: Multiscale parameter regionalization of a grid-based hydrologic model at the mesoscale, Water Resources Research, 46, https://doi.org/10.1029/2008WR007327, 2010.

Schmit, M.: WeatherDB, available at: https://weather.hydro.intra.uni-freiburg.de, (last access: 20 January 2023).

Schwemmle, R.: RoGeR - a process-based hydrological toolbox model in Python, available at: https://doi.org/10.5281/zenodo.7633362 and https://github.com/Hydrology-IFH/roger, (last access: 9 March 2023).

Schwemmle, R., Demand, D., and Weiler, M.: Technical note: Diagnostic efficiency – specific evaluation of model performance, Hydrol. Earth Syst. Sci., 25, 2187-2198, https://doi.org/10.5194/hess-25-2187-2021, 2021.

Schwemmle, R. and Weiler, M.: Consistent modelling of transport processes and travel times - coupling soil hydrologic processes with StorAge Selection functions (in review), submitted to Water Resources Research, https://doi.org/10.22541/essoar.167751575.55537069/v1, 2023.

Šimůnek, J., van Genuchten, M. T., and Šejna, M.: Recent Developments and Applications of the HYDRUS Computer Software Packages, Vadose Zone Journal, 15, vzj2016.2004.0033, https://doi.org/10.2136/vzj2016.04.0033, 2016.

Stack Overflow: Stack Overflow Developer Survey, available at: https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies, (last access: 12 January 2023).

Steduto, P., Hsiao, T. C., Raes, D., and Fereres, E.: AquaCrop—The FAO Crop Model to Simulate Yield Response to Water: I. Concepts and Underlying Principles, Agronomy Journal, 101, 426-437, https://doi.org/10.2134/agronj2008.0139s, 2009.

Steinbrich, A., Leistert, H., and Weiler, M.: Model-based quantification of runoff generation processes at high spatial and temporal resolution, Environmental Earth Sciences, 75, 1423, https://doi.org/10.1007/s12665-016-6234-9, 2016.

Steinbrich, A., Leistert, H., and Weiler, M.: RoGeR – ein bodenhydrologisches Modell für die Beantwortung einer Vielzahl hydrologischer Fragen, Korrespondenz Wasserwirtschaft, 14, https://doi.org/10.3243/kwe2021.02.004, 2021.

Stoll, S. and Weiler, M.: Explicit simulations of stream networks to guide hydrological modelling in ungauged basins, Hydrol. Earth Syst. Sci., 14, 1435-1448, https://doi.org/10.5194/hess-14-1435-2010, 2010.

van der Velde, Y., Torfs, P. J. J. F., van der Zee, S. E. A. T. M., and Uijlenhoet, R.: Quantifying catchment-scale mixing and its effect on time-varying travel time distributions, Water Resources Research, 48, https://doi.org/10.1029/2011WR011310, 2012.

van Gompel, M., Noordzij, J., de Valk, R., and Scharnhorst, A.: Guidelines for Software Quality, Common Lab Research Infrastructure for the Arts and Humanities, Amsterdam, Netherlands, 1-42 pp., 2016.

Weiler, M.: An infiltration model based on flow variability in macropores: development, sensitivity analysis and applications, Journal of Hydrology, 310, 294-315, https://doi.org/10.1016/j.jhydrol.2005.01.010, 2005.

Weiler, M. and Beven, K.: Do we need a Community Hydrological Model?, Water Resources Research, 51, 7777-7784, https://doi.org/10.1002/2014WR016731, 2015.