Geosci. Model Dev., 17, 477–495, 2024
https://doi.org/10.5194/gmd-17-477-2024

Geoscientific Model Development
Open Access

mesas.py v1.0: a flexible Python package for modeling solute transport and transit times using StorAge Selection functions

Ciaran J. Harman[1,2] **and Esther Xu Fei**[1]

[1]Department of Environmental Health and Engineering, Johns Hopkins University, Baltimore, MD, USA
[2]Department of Earth and Planetary Sciences, Johns Hopkins University, Baltimore, MD, USA

Correspondence: Ciaran J. Harman (charman1@jhu.edu)

Received: 14 November 2022 – Discussion started: 12 December 2022
Revised: 22 August 2023 – Accepted: 1 September 2023 – Published: 19 January 2024

Abstract. StorAge Selection (SAS) transport theory has recently emerged as a framework for representing material transport through a control volume. It can be seen as a generalization of transit time theories and lumped-parameter models to allow for arbitrary temporal variability in the rate of material flow in and out of the control volume, and in the transport dynamics. SAS is currently the state-of-the-art approach to interpreting tracer transport. Here, we present mesas.py, a Python package implementing the SAS framework. mesas.py allows SAS functions to be specified using several built-in common distributions, as a piecewise linear cumulative distribution function (CDF), or as a weighted sum of any number of such distributions. The distribution parameters and weights used to combine them can be allowed to vary in time, permitting SAS functions of arbitrary complexity to be specified. mesas.py simulates tracer transport using a novel mass-tracking scheme and can account for first-order reactions and fractionation. We present a number of analytical solutions to the governing equations and use these to validate the code. For a benchmark problem the time-step-averaging approach of the mesas.py implementation provides a reduction in mass balance errors of up to 15 times in some cases compared with a previous implementation of SAS.

1 Introduction

StorAge Selection (SAS) is a theoretical framework for modeling transport dynamics through spatially integrated systems (control volumes). It is applicable for any system in which it is reasonable to assume that the bulk material flowing out of the system (at rate $Q(t)$) at some time t is some conservative mixture of the bulk material that flowed in at earlier times (at rate $J(t)$). For example, it is often reasonable to assume that the streamflow and evapotranspiration leaving a watershed are some conservative mixture of precipitation that fell on that watershed at earlier times. The conservative bulk material in that case is simply the water comprising the rainfall, streamflow, and evapotranspiration. The development of this theory and related issues in watershed hydrology have been recently reviewed in Benettin et al. (2022).

SAS is a generalization of the idea of a transit time distribution (TTD), which has proven useful in a wide range of disciplines, including chemical engineering (Ross et al., 2006), transportation engineering (Tyworth and Zeng, 1998), groundwater hydrology (Dupas et al., 2020; Rinaldo et al., 2015; Danesh-Yazdi et al., 2018), surface water hydrology (Stockinger et al., 2016; Rodriguez and Klaus, 2019; Rodriguez et al., 2021), medicine (Rossum et al., 1989), and others. However TTDs have previously required that the bulk material flow through the system be approximately steady (i.e., $J(t) = Q(t) =$ a constant). SAS relaxes this assumption in a rigorous and general way, so that it can (in principle) be used to characterize transport through any system in which the bulk material flow is conserved. However, to date, SAS functions have not been widely adopted in practice. In part, this is due to the perception that they are too complex and data-hungry.

Our objective here is to provide detailed documentation of mesas.py, a Python implementation of SAS functions that is easy to use, highly flexible, and computationally accurate. This implementation is already the basis of online teaching resources (Harman, 2020), and we hope to develop more in the future. It is essential, therefore, that a peer-reviewed publication exists that supports and documents the software.

In a typical forward-modeling use-case, we wish to predict the concentration $C_Q(t)$ of a conservative tracer in the bulk material outflow $Q(t)$, which is assumed to be a conservative mixture of previous bulk material inflows $J(t)$ in which the tracer concentration was $C_J(t)$. If this is the case, the outflow concentration $C_Q(t)$ will be some weighted average of past values of $C_J(t)$. The transit time distribution $p_Q(T,t)$ gives those weights:

$$C_Q(t) = \int_0^\infty C_J(t-T)\, p_Q(T,t)\, \mathrm{d}T. \tag{1}$$

SAS provides a means to calculate the time-varying distribution $p_Q(T,t)$ for a given system. An overview of SAS and related approaches can be found in Botter (2012), Harman (2015), Rinaldo et al. (2015), and Benettin and Bertuzzo (2018).

The basic equations required to calculate $p_Q(T,t)$ (discussed in Sect. 2 below) are not especially difficult to solve numerically, but some care is required. An implementation of SAS in MATLAB (tran-SAS) is already available (Benettin and Bertuzzo, 2018). mesas.py replicates the functionality of tran-SAS but offers the following features:

- mesas.py offers an extremely flexible framework for specifying SAS functions, allowing them to be arbitrarily complex and variable in time. This includes the ability to specify SAS functions as a time-varying weighted sum of other functions (Rodriguez and Klaus, 2019) and as a (time-varying) piecewise linear cumulative distribution function (CDF) with any number of segments.
- mesas.py uses a novel mass-tracking approach that estimates solute/tracer mass storage as part of the solution.
- mesas.py estimates the time-step-averaged transit times and mass fluxes using a fourth-order Runge–Kutta method, and it provides superior numerical accuracy and mass balance accounting (as we shall demonstrate).
- mesas.py allows for time-varying first-order reactions and time-varying solute/tracer fractionation.
- mesas.py is implemented in Python and Fortran and is designed to be easy to install (through conda-forge) and user-friendly.

The governing equations of the SAS framework are given in Sect. 2 of this paper, including the novel approach to solute/tracer mass tracking. Calculating the storage and release of solutes/tracers continuously in tandem with calculation of the TTD (rather than using the convolution after the TTD has been obtained) makes incorporating reactions and fractionation into the SAS function simple and intuitive. Section 3 gives details of the code, including the numerical implementation, the method for specifying SAS functions, and procedures for running the code.

In Sect. 4 of the paper, we test the code against a number of benchmarks in the form of analytical solutions to the governing equations. These include cases of steady and unsteady flow. We compare the accuracy of mesas.py against that of tran-SAS for the unsteady-flow case.

2 Governing equations

To estimate p_Q and solve Eq. (1), two key pieces of information are required: (1) time series of inflows $J(t)$ and outflows $Q_q(t)$ (there may be more than one outflow, hence the subscript index q) and (2) SAS functions Ω_q (one for each outflow q) that capture the way each outflow is drawn from the water of different ages available to be removed from storage. The inflow and outflow data are used to solve expressions of conservation of mass that describe how the age distribution of the material in storage changes over time as material is added and removed. The SAS functions are needed to calculate this solution because they characterize the relative rate at which material of different relative ages is selected for removal.

2.1 Conservation laws

2.1.1 Conservation law for the bulk material flows

The bulk flow is the material that makes up the inflows and outflows from the system, carrying tracers and other species of material with it. Typically, in hydrologic applications, the bulk flow is water. As is typical in hydrology, we assume that the water is incompressible so we can refer to units of volume for convenience, but the framework is valid for any conservative bulk flow as long as fluxes, storages, and concentrations are expressed in consistent compatible units.

The conservation equation for the bulk flow can be obtained by considering an incremental volume $s_T(T,t)$ that has an age T at time t; therefore, it entered at time $t_i = t - T$. Note that $s_T(T,t)$ has units of volume (or mass) *per time*, as it is the amount that entered in an infinitesimal increment of time. If the inflow rate is $J(t)$ at some time in the past $t = t_i$, then $s_T(0,t) = J(t)$. Over time, the quantity of bulk flow residing in storage represented by s_T depletes due to outflows. Assuming that outflows are indexed by q, and each has an outflow rate Q_q and transit time distribution $p_q(T,t)$, the time evolution of s_T from some initial time t_i to the present time t is given by the following:

$$\frac{\mathrm{d}}{\mathrm{d}t} s_T(t-t_i,t) = J(t)\delta(t-t_i) - \sum_q Q_q(t)\, p_q(t-t_i,t), \tag{2}$$

where $\delta(\cdot)$ is the Dirac delta distribution. Typically, the derivative on the left here is broken up into two terms, as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t}s_T(t-t_i,t)=\frac{\partial s_T}{\partial t}+\frac{\partial s_T}{\partial T}. \tag{3}$$

However, the form given in Eq. (2) serves to remind us that these two derivatives can be thought of as representing the rate at which s_T changes as it simultaneously moves through time and ages. We can think of it moving along a characteristic curve that is a straight line in age–time space, with a slope of one unit of age per unit of time and passing through the point $(T,t)=(0,t_i)$. The computational method of mesas.py is based on numerically integrating along this characteristic curve.

Integrating s_T over all ages up to some age T gives the cumulative form S_T, known as the *age-ranked storage*:

$$S_T(T,t)=\int_0^T s_T(\tau,t)\,\mathrm{d}\tau. \tag{4}$$

This is the volume of bulk material residing in storage that is younger than T at time t. The bulk material conservation equation is often expressed in terms of this cumulative quantity and $P_q(T,t)$, the cumulative form of $p_q(T,t)$. Equation (2) can be obtained from the cumulative form by taking the derivative with respect to T.

S_T is also essential for solving Eq. (2) through its role in evaluating the SAS function (see Sect. 2.2 below). Therefore, even though the primary state variable that mesas.py solves for is $s_T(T,t)$, the code must also keep track of the accumulating values of $S_T(T,t)$.

2.1.2 Conservation law for solutes

Consider a conservative solute or tracer that travels ideally with the bulk material. We can define $m_T(T,t)$ as the incremental tracer mass that entered at time t_i and is now remaining in storage (m_T). We can also define a notion of *age-ranked concentration* in storage as the increment of age-ranked solute mass per increment of age-ranked storage:

$$C_T(T,t)=\frac{m_T(T,t)}{s_T(T,t)}. \tag{5}$$

Note that this is not a concentration in the usual sense. C_T may not correspond to an actual measurable concentration anywhere in the system if materials of different ages have intermingled sufficiently. However, it does (by definition) equal the input concentration $C_J(t)$ just as water enters the system; thus,

$$m_T(0,t)=C_T(0,t)s_T(0,t)=C_J(t)J(t). \tag{6}$$

It also gives the effective concentration of the solute in the increment of water at age $T=t-t_i$ that is contributing to each outflow, and thus controls the mass flux out for a given age increment, which we will term $\dot{m}_q$:

$$\dot{m}_q(T,t)=Q_q(t)C_T(T,t)p_q(T,t). \tag{7}$$

Putting these together, we can write a conservation law for the solute as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t}m_T(t-t_i,t)=J(t)C_J(t)\delta(t-t_i)-\sum_q\dot{m}_q(t-t_i,t). \tag{8}$$

This equation is analogous to Eq. (2), but instead of tracking the bulk material along a characteristic curve in age–time space, it tracks the mass of solute/tracer.

2.1.3 Accounting for fractionation and reactions

We can easily generalize Eq. (8) in two ways. First, we can account for the effect of fractionation in the outflows. Sometimes the concentration of a solute in an outflow is different from its concentration in storage. For example, chloride (which has been used as a tracer in catchment studies, such as Harman, 2015) can leave in discharge, but it is not carried out of the catchment by evapotranspiration. Thus, the effective concentration of chloride in the evaporative flux must be zero. A less extreme example is where stable water isotope ratios in evaporation tend to be lighter than those in the water left behind.

We can account for this fractionation in a simple way by assuming that the concentration of the solute in outflow q is some (possibly time-varying) multiple $\alpha_q(t)$ of the concentration in storage. To accomplish this we modify Eq. (7) to include the following:

$$\dot{m}_q(T,t)=\alpha_q(t)Q_q(t)C_T(T,t)p_q(T,t). \tag{9}$$

When $\alpha_q=1$, there is no fractionation. $\alpha_q<1$ will result in reduced concentrations in the given outflow, and $\alpha_q=0$ excludes the solute from the outflow. It is also possible to set $\alpha_q>1$ if the solute is preferentially entrained in the outflow. Note that if isotope fractionation is being modeled, the isotope data must be given as an isotope ratio, rather than a δ value (per mill). For example, values of δ^{18}O must be converted using $R=\delta^{18}\mathrm{O}/1000+1$. The value of α is simply the regular fractionation factor $\alpha_{A/B}$ (Kendall and Caldwell, 1998). If fractionation is being neglected, δ values may be used.

Second, we can account for first-order reactions. The change in C_T resulting from mass introduced or removed by such a reaction can be modeled as follows:

$$\frac{\mathrm{d}C_T}{\mathrm{d}t}=k_1(C_\mathrm{eq}-C_T). \tag{10}$$

From this we can define a reaction term $\dot{m}_R$ as follows:

$$\dot{m}_R(T,t)=s_T(T,t)\frac{dC_T}{\mathrm{d}t}=k_1(t)(C_\mathrm{eq}(t)s_T(T,t)-m_T(T,t)), \tag{11}$$

where k_1 is a first-order reaction rate and C_eq is an equilibrium concentration (both of which may vary in time).

Including fractionation and the reaction terms in the solute conservation law gives

$$\frac{\mathrm{d}}{\mathrm{d}t}m_T(t-t_i,t) = J(t)C_J(t)\delta(t-t_i) - \sum_q \dot{m}_q(T,t) + \dot{m}_R(T,t), \tag{12}$$

where $\dot{m}_q$ is given by Eq. (9) and $\dot{m}_R$ is given by Eq. (11).

The actual outflow concentration at time t is obtained by integrating $\dot{m}_q(T,t)$ over all ages $T \leq T_{\max}$ and then adding the "old-water" contribution:

$$C_q(t) = \frac{1}{Q_q(t)} \int_0^{T_{\max}} \dot{m}_q(T,t)\mathrm{d}T + C_{\text{old}} \times (1 - P_q(T_{\max},t)), \tag{13}$$

where C_{old} is the concentration assigned to all the water in storage with an age greater than $T_{\max}$. $T_{\max} = t$ usually, but it may be set to be less than t to reduce memory demands or speed up computation (at the cost of potentially truncating the contributions of water that entered early in the simulation to outflows later).

2.2 StorAge Selection (SAS) functions

The equations above cannot be solved on their own, as p_Q is not known. The SAS functions provide the required additional relationship linking the age-ranked storage and the transit time distribution.

Given a volume of age-ranked storage S_T representing all the bulk material in a control volume with an age of T or less, the cumulative SAS function Ω_q is defined as a function that gives the fraction of outflow Q_q drawn from $S_T(T,t)$. This is also (by definition) the fraction of discharge with an age of T or less, which is simply $P_q(T,t)$. Thus, we can write the following:

$$P_q(T,t) = \Omega_q(S_T,t), \tag{14}$$

where $S_T = S_T(T,t)$. That is, the SAS function and the cumulative transit time distribution both give the fraction of discharge with age T or less, but the SAS function expresses the age in terms of age-ranked storage $S_T(T,t)$, rather than age T. This has proven to be very useful because $P_q(T,t)$ varies in time due to variations in fluxes, whereas $\Omega_q(S_T,t)$ only varies when the manner in which storage turns over varies. In many applications, SAS functions have been approximated by a variety of continuous distributions with good results (Benettin et al., 2022). We will discuss several that have been implemented in mesas.py in the next section.

By taking the derivative of the equation above and applying the chain rule, we can see that

$$p_q(T,t) = \omega_q(S_T,t)s_T(T,t), \tag{15}$$

where ω_q is the density form of Ω_q. The left-hand side of this equation is the rate of discharge of water with age T. The right-hand side is the rate at which water is removed from the age-ranked storage at $S_T(T,t)$.

The SAS functions needed to represent a particular system are typically obtained by first choosing a functional form from those presented below and then tuning the parameters of that functional form such that the model predictions match the tracer observations. It should be noted that there is currently an element of subjectivity and imprecision here, as multiple functional forms may produce equally acceptable fits to the available data (Harman, 2019). In previous applications to watersheds, streamflow has been represented with a heavily right-skewed distribution whose mean varies inversely with catchment wetness, whereas evapotranspiration (ET) has been represented by uniform distributions over the youngest water in storage. More physically based parameterizations may be available in the future.

2.2.1 Continuous SAS functions available in mesas.py

The SAS function must be specified so that it accurately captures how a system turns over, releasing storage as bulk outflow. At present, three continuous distributions commonly used for specifying SAS functions are available as built-in options in mesas.py: the beta, Kumaraswamy, and gamma distributions. More details on each distribution are given below.

These distributions each have at least two parameters: a *location* parameter ($S_{\min}$) and a *scale* parameter (S_0). These parameters both have units of storage, and they serve to shift and scale the values of S_T into a normalized form:

$$x = \frac{S_T - S_{\min}}{S_0}. \tag{16}$$

Here, $x = 0$ if $S_T < S_{\min}$. In mesas.py, the values of S_0, $S_{\min}$, and all other parameters can be given as constant values, or different values for every time step can be provided.

Briefly, the three aforementioned built-in distributions are as follows:

– *Beta distribution.* The CDF of the beta distribution is given by

$$\text{Beta}(S_T; S_{\min}, S_0, \alpha, \beta) = \frac{B(x;\alpha,\beta)}{B(\alpha,\beta)} \quad S_{\min} \leq S_T \leq S_{\min} + S_0, \tag{17}$$

where $B(x;\alpha,\beta)$ and $B(\alpha,\beta)$ are the incomplete and complete beta functions, and $\alpha, \beta > 0$. This distribution has been used to represent the SAS function of systems whose total active storage volume $S(t)$ has been estimated (Benettin et al., 2022). In those cases, S_m is set to zero and $S_0 = S(t)$.

- *Kumaraswamy distribution.* The CDF of the Kumaraswamy distribution is given by

  $$\begin{aligned}&\mathrm{Kumaraswamy}(S_T; S_{\min}, S_0, a, b) = 1-(1-x^a)^b\\ &S_{\min} \leq S_T \leq S_{\min}+S_0, \qquad (18)\end{aligned}$$

  where $a, b > 0$. This distribution has also been used to represent the SAS function of systems whose total active storage volume has been estimated.

- *Gamma distribution.* The gamma distribution is given by

  $$\begin{aligned}&\mathrm{Gamma}(S_T; S_{\min}, S_0, \alpha) = \frac{\Gamma(\alpha, x)}{\Gamma(\alpha)}\\ &\text{for } S_{\min} \leq S_T < \infty. \qquad (19)\end{aligned}$$

  This distribution has been used to represent the SAS function of systems whose total volume is unknown, and the right tail of the SAS function is assumed to taper exponentially for sufficiently large S_T (Harman, 2015; Benettin et al., 2022).

In addition, any distribution specified by the scipy.stats library can be used in mesas.py by converting its CDF into a piecewise linear form (see next section). This is done automatically within mesas.py. The accuracy of the results obtained this way may be poor. The piecewise linear form may be inaccurate where the PDF changes rapidly (as a gamma distribution does near $S_T = 0$ when $\alpha < 1$) or where it approaches an asymptotic value at the tails.

2.2.2 Piecewise linear SAS functions in mesas.py

SAS functions can also be specified as a piecewise linear CDF with N segments. When $N = 1$, this is simply a uniform distribution. These linear segments join control points $(S_{T\,q,0}, \Omega_{q,0})$, $(S_{T\,q,1}, \Omega_{q,1})\ldots(S_{T\,q,N}, \Omega_{q,N})$. To ensure that the result is a probability distribution, we require $0 = \Omega_{q,0} \leq \Omega_{q,1} \leq \ldots \leq \Omega_{q,N} = 1$ and $0 \leq S_{T\,q,0} \leq S_{T\,q,1} \leq \ldots \leq S_{T\,q,N}$. The SAS function is specified by giving these control points, which may vary in time. The PDF $\omega_q(S_T, t)$ is piecewise constant:

$$\omega_q(S_T,t) = \begin{cases} \frac{\Omega_{q,1}-\Omega_{q,0}}{S_{T\,q,1}-S_{T\,q,0}} & S_{T\,q,0} \leq S_T < S_{T\,q,1} \\ \frac{\Omega_{q,2}-\Omega_{q,1}}{S_{T\,q,2}-S_{T\,q,1}} & S_{T\,q,1} \leq S_T < S_{T\,q,2} \\ \ldots & \\ \frac{\Omega_{q,N}-\Omega_{q,N-1}}{S_{T\,q,N}-S_{T\,q,N-1}} & S_{T\,q,N-1} \leq S_T < S_{T\,q,N} \\ 0 & \text{otherwise.} \end{cases} \qquad (20)$$

The parameters of the N-segment piecewise SAS function are the $N+1$ values of $S_{T\,q,n}$ and the $N-1$ values of $\Omega_{q,n}$ (recall that $\Omega_{q,0} = 0$ and $\Omega_{q,N} = 1$).

2.2.3 SAS functions built from weighted sums of components

mesas.py also allows an SAS function to be specified as a (time-varying) weighted sum of component SAS functions, each of which may specified in any of the available ways. This approach was first suggested by Rodriguez and Klaus (2019), and Wilusz et al. (2020) provided evidence supporting its validity.

Given M *component* SAS functions (indexed by m) defined using one of the methods presented above, the overall SAS function can be obtained as follows:

$$\omega_q(S_T,t) = \sum_{m=1}^{M} f_{q,m}(t)\omega_{q,m}(S_T,t), \qquad (21)$$

where $f_{q,m}(t)$ is a (possibly time-varying) weight. These weights must be provided as inputs to the model. The weights should sum to 1 at each time step (although this is not enforced by mesas.py).

3 Methods

3.1 Numerical implementation

The numerical implementation of the governing equations in mesas.py is reminiscent of a numerical finite-volume scheme. We will assume that time steps Δt and age steps ΔT are equal.

First, age-step-averaged forms of the state variables are obtained by integrating in T over an interval of past input times $[t_i - \Delta T, t_i]$:

$$s_i(t) = \frac{1}{\Delta T}\int_{t_i-\Delta T}^{t_i} s_T(t-\tau,t)\mathrm{d}\tau, \qquad (22)$$

$$m_i(t) = \frac{1}{\Delta T}\int_{t_i-\Delta T}^{t_i} m_T(t-\tau,t)\mathrm{d}\tau. \qquad (23)$$

For notational consistency, we can define the cumulative version of this as $S_i(t)$, but it is precisely equal to $S_T(t-t_i,t)$. We can use these to write age-step-averaged forms of Eqs. (2) and (12), respectively:

$$\begin{aligned}\Delta T\frac{\mathrm{d}s_i}{\mathrm{d}t} = J(t)\int_{t_i-\Delta T}^{t_i}\delta(t-\tau)\mathrm{d}\tau - \sum_q Q_q(t)\\ \int_{t_i-\Delta T}^{t_i} p_q(t-\tau,t)\mathrm{d}\tau, \qquad (24)\end{aligned}$$

$$\Delta T \frac{\mathrm{d}m_i}{\mathrm{d}t} = J(t)C_J(t) \int\limits_{t_i-\Delta T}^{t_i} \delta(t-\tau)\mathrm{d}\tau - \sum_q \left(\int\limits_{t_i-\Delta T}^{t_i} \dot{m}_q(t-\tau,t)\mathrm{d}\tau \right) - \int\limits_{t_i-\Delta T}^{t_i} \dot{m}_R(t-\tau,t)\mathrm{d}\tau. \tag{25}$$

There are a few things to unpack here. First, note that, due to the properties of the Dirac δ-function, the integral $\int_{t_i-\Delta T}^{t_i}\delta(t-\tau)\mathrm{d}\tau$ is 1 if $t_i - \Delta T < t < t_i$ but is 0 otherwise. Thus, for all times after t_i, the first term disappears in both equations above. This behavior can be represented using the indicator function $\mathbf{1}_{[t_i-\Delta T,t_i)}(t)$, which we will write as $\mathbf{1}_i(t)$ for short.

The age-step-averaged TTD $p_{qi}(t)$ can be expressed in terms of the cumulative TTD, and thus in terms of the SAS function, as follows:

$$\begin{aligned} p_{qi}(t)\Delta T &= \int\limits_{t_i-\Delta T}^{t_i} p_q(t-\tau,t)\mathrm{d}\tau = P_q(t-t_i+\Delta T,t) \\ &\quad - P_q(t-t_i,t) \\ &= \Omega_q(S_T(t-(t_i-\Delta T),t),t) \\ &\quad - \Omega_q(S_T(t-t_i,t),t) \\ &= \Omega_q(S_i(t)+s_i(t),t) - \Omega_q(S_i(t),t). \end{aligned} \tag{26}$$

The time-step-averaged values of $\dot{m}_q$ and $\dot{m}_R$ can then be obtained if we are willing to approximate that the age-ranked concentration C_T is constant over the interval ΔT at $C_T = m_i/s_i$. This amounts to approximating the concentration C_J in the corresponding input bulk material as constant over this interval.

Thus, we can express the governing equations as the set of ordinary differential equations (ODEs):

$$\begin{aligned} \Delta T \frac{\mathrm{d}s_i}{\mathrm{d}t} &= J(t)\mathbf{1}_i(t) - \sum_q Q_q(t)p_{qi}(t)\Delta T \\ &= f_s(s_i,S_i,t)\Delta T, \end{aligned} \tag{27}$$

$$\begin{aligned} \Delta T \frac{\mathrm{d}m_i}{\mathrm{d}t} &= J(t)C_J(t)\mathbf{1}_i(t) - \frac{m_i(t)}{s_i(t)} \\ &\quad \left(\sum_q \alpha_q(t)Q_q(t)p_{qi}(t)\Delta T\right) \\ &\quad - k_1(t)(C_{\mathrm{eq}}(t)s_i(t) - m_i(t)) \\ &= f_m(m_i,s_i,S_i,t)\Delta T. \end{aligned} \tag{28}$$

If the right-hand side of these equations were functions of the state variables s_i and m_i and a number of time-variable coefficients (J, Q_q, C_J, α_q, k_1, and C_{eq}), these would be ODEs. The dependence of the SAS function on the cumulative value $S_i(t)$ complicates matters only slightly.

The numerical solution of these equations involves two core tasks:

1. estimating the rates of change $f_s(s_i,S_i,t)$ and $f_m(m_i,s_i,S_i,t)$ and
2. using these to estimate the state variables s_i and m_i (and S_i) at a future time.

Let us assume $t = j\Delta t$ and $T = i\Delta T$, and recall that age and time steps are equal in size. Now define $\mathtt{s}_\mathtt{i}^\mathtt{j} = s_i(j\Delta t)$, and define $\mathtt{m}_\mathtt{i}^\mathtt{j}$ similarly. Particular care must be taken to ensure that an accurate value of S_T is used to evaluate the SAS function. Let

$$\mathtt{S}_\mathtt{i}^\mathtt{j} = \sum_{k=i+1}^{j} \mathtt{s}_\mathtt{k}^\mathtt{j}\Delta t. \tag{29}$$

Note that, by this definition, the value of $\mathtt{S}_\mathtt{i}^\mathtt{j}$ depends on $\mathtt{s}_{\mathtt{i}+1}^\mathtt{j}$ but not on $\mathtt{s}_\mathtt{i}^\mathtt{j}$. For the fourth-order Runge–Kutta (RK4) and midpoint methods, the state variables must be evaluated at an intermediate point halfway between the regular time steps. The value of $\mathtt{S}_\mathtt{i}^{\mathtt{j}+\frac{1}{2}}$ can be estimated as follows:

$$\mathtt{S}_\mathtt{i}^{\mathtt{j}+\frac{1}{2}} = \frac{\mathtt{S}_\mathtt{i}^\mathtt{j} + \mathtt{S}_\mathtt{i}^{\mathtt{j}+1}}{2}. \tag{30}$$

However, note that $\mathtt{S}_\mathtt{i}^{\mathtt{j}+1}$ can be found only after an estimate of $\mathtt{s}_{\mathtt{i}+1}^{\mathtt{j}+1}$ is obtained. This suggests that, in order to calculate how $\mathtt{s}_\mathtt{i}^\mathtt{j}$ becomes $\mathtt{s}_\mathtt{i}^{\mathtt{j}+1}$, we must first have determined both $\mathtt{s}_{\mathtt{i}+1}^\mathtt{j}$ and $\mathtt{s}_{\mathtt{i}+1}^{\mathtt{j}+1}$. In other words, for an accurate numerical solution, we need to know the fate of younger material before we can determine the fate of older material. Because of this, mesas.py solves $j \to j+1$ for all i before moving on to $i+1$, and so on.

By default, mesas.py uses the RK4 method to estimate the value of $\mathtt{s}_\mathtt{i}^{\mathtt{j}+1}$ as follows:

$$\begin{aligned} k_1 &= f_s\left(\mathtt{s}_\mathtt{i}^\mathtt{j}, \mathtt{S}_\mathtt{i}^\mathtt{j}, j\Delta t\right), \\ k_2 &= f_s\left(\mathtt{s}_\mathtt{i}^\mathtt{j} + \frac{1}{2}k_1\Delta t, \mathtt{S}_\mathtt{i}^{\mathtt{j}+\frac{1}{2}}, \left(j+\frac{1}{2}\right)\Delta t\right), \\ k_3 &= f_s\left(\mathtt{s}_\mathtt{i}^\mathtt{j} + \frac{1}{2}k_2\Delta t, \mathtt{S}_\mathtt{i}^{\mathtt{j}+\frac{1}{2}}, \left(j+\frac{1}{2}\right)\Delta t\right), \\ k_4 &= f_s\left(\mathtt{s}_\mathtt{i}^\mathtt{j} + k_3\Delta t, \mathtt{S}_\mathtt{i}^{\mathtt{j}+1}, (j+1)\Delta t\right), \\ k_* &= \frac{k_1 + 2k_2 + 2k_2 + k_1}{6}, \\ \mathtt{s}_\mathtt{i}^{\mathtt{j}+1} &= \mathtt{s}_\mathtt{i}^\mathtt{j} + k_*\Delta t. \end{aligned} \tag{31}$$

Similar steps are followed to estimate $\mathtt{m}_\mathtt{i}^{\mathtt{j}+1}$ using $f_m(m_i,s_i,S_i,t)$. During the calculation, the intermediate values of p_q, $\dot{m}_q$, and $\dot{m}_R$ are also tracked and time-step-averaged according to the same scheme. It is also possible to run mesas.py with a midpoint method or with a forward Euler scheme, although at some sacrifice in terms of numerical accuracy.

The state variables are held in memory in arrays whose columns are times and whose rows are ages. Thus, stepping through time steps j for a fixed input time i corresponds to stepping along a diagonal of a state variable matrix.

mesas.py allows initial conditions to be specified for the age-ranked storage s_T and solute mass m_T. This is useful for restarting calculations or spinning up the simulation. The initial conditions are supplied as a vector of values $\mathtt{s}_\mathtt{k}^0$ and $\mathtt{m}_\mathtt{k}^0$ for ages k. Each entry represents the value of s_T and m_T at $t = 0$ averaged over each age interval and is used to populate the first column of the matrices holding the state variables.

mesas.py proceeds by solving all the time steps $j \rightarrow j+1$ for the first age step before moving on to second, and so forth. Typically, when there are N time steps, the solution is found for N age steps also. This is sometimes excessive because the contributions of bulk flow from the start of the simulation to the final time step may be negligible. The number of age steps can also be larger than N, although this will only have value if an initial condition is provided that gives values of $\mathtt{s}_\mathtt{i}^0$ for more than $> N$ ages. Note that all outflow with an unknown age is assigned concentration C_{old}.

3.2 Model specification and input structure

Inputs to mesas.py come in two main forms:

1. parameters specifying the SAS function(s), solute properties, and other model settings and
2. time series of inflows, outflows, and other variables.

The parameters are specified using a nested data structure that can be stored and read from a JSON-formatted text file or fed into a model object instance directly as a Python dictionary. The time series can be provided as a .csv text file or as a pandas data frame.

The parameter data structure consists of a dictionary of key:value pairs, where a "key" is an immutable label (typically a string) and a "value" is an object that can be retrieved from the dictionary using the associated key. The values can themselves be dictionaries, allowing for a nested structure to the data.

The top-level dictionary in the parameter specification must have a key `"sas_specs"`. The associated value must be a dictionary of SAS specifications. It may also have two optional entries: `"solute_parameters"`, which provides information about the solutes to be routed through the model, and `"options"`, which can be used to set a number of model options.

3.2.1 SAS function specification

A basic example of the `"sas_specs"` key:value pair is shown below.

```
"sas_specs":{

    "Q":{
        "Q SAS function 1":{
            "func": "gamma",
            "args": {
                "loc": 0.0,
                "scale": "S_scale",
                "a": 0.6856
            }
        },
        "Q SAS function 2":{
            "func": "beta",
            "args": {
                "loc": 0.0,
                "scale": 150.0,
                "a": 1.0,
                "b": 3.0
            }
        }
    },

    "ET":{
        "ET SAS function":{
            "ST": [0.0, "S_ET"],
            "P":  [0.0, 1.0]
        }
    }
},
```

The `"sas_specs"` dictionary should contain one key for each bulk flux out of the control volume, and each key must exactly match the heading of a column in the time series dataset giving that flux rate. In the example above, mesas.py would expect the time series dataset to contain columns `"Q"` and `"ET"`.

Each of the keys naming a bulk flux in `"sas_specs"` is associated with a dictionary specifying the SAS functions for that flux. That dictionary can also include multiple SAS functions, which are combined together using time-varying weights. In the example above, mesas.py would expect to find columns in the time series dataset titled `"Q SAS function 1"` and `"Q SAS function 2"` containing weights to multiply each SAS function. These weights should add up to 1, although this is not checked. If the dictionary associated with each flux contains only one key:value pair, it is not necessary to provide a weights column in the time series dataset.

Presently, each SAS function can be specified in three different ways:

- as a gamma, beta, or Kumaraswamy distribution;
- using any distribution from scipy.stats;
- as a piecewise linear CDF.

Table 1. The relationship between the parameters in Eqs. (17), (18), and (19) and the keys used to specify the value of these parameters in the SAS function specification.

Gamma distribution (19)			Beta distribution (17)			Kumaraswamy distribution (18)		
$S_{\min}$	$\leftrightarrow$	`"loc"`	$S_{\min}$	$\leftrightarrow$	`"loc"`	$S_{\min}$	$\leftrightarrow$	`"loc"`
S_0	$\leftrightarrow$	`"scale"`	S_0	$\leftrightarrow$	`"scale"`	S_0	$\leftrightarrow$	`"scale"`
α	$\leftrightarrow$	`"a"`	α	$\leftrightarrow$	`"a"`	a	$\leftrightarrow$	`"a"`
			β	$\leftrightarrow$	`"b"`	b	$\leftrightarrow$	`"b"`

The gamma, beta, or Kumaraswamy distributions are coded into the core computational code, whereas the scipy.stats distributions are approximated as piecewise linear CDFs. In either case, the distribution is selected based on the value associated with `"func"`. In the example above, gamma and beta distributions are combined to produce the SAS function for outflow `"Q"`.

The distribution parameters are given by the dictionary associated with the key `"args"`. The expected content of this varies between distributions (see Table 1). Any parameter value can be specified as a fixed number or can be allowed to vary in time. Time-varying parameters are given as a string identical to a column in the time series dataset where the time-varying values are provided. In the example `"sas_specs"` above, the scale parameter of the gamma distribution used in `"Q SAS function 1"` is set to `"S_scale"`. This tells mesas.py to use the time series of values found in that column of the input dataset for the scale parameter.

To use a distribution from scipy.stats the key:value pair `"use":"scipy.stats"` should be included. An optional parameter `"nsegments"` sets the number of segments used to approximate the distribution. Note that this approach is included for convenience but is not recommended when the tails of the SAS distribution are important for the problem being considered, as they may not be well captured by the piecewise linear CDF.

Alternatively, the SAS function can be specified as a piecewise linear CDF. In the example above, this option is used to specify a uniform SAS function for ET using a single linear segment. The cumulative age-ranked storage values `"ST"` and corresponding cumulative probabilities `"P"` (varying from zero to one) must be provided as lists of increasing values. Any of the values in these lists may be allowed to vary in time by instead providing a string corresponding to the heading of a column in the input time series dataset – see, for example, `"S_ET"` in the `"ET SAS function"` above.

3.2.2 Solute parameters

Solute properties are given in a dictionary associated with the top-level key `"solute_parameters"`. The keys in this dictionary should correspond to columns in the time series dataset giving inflow concentrations. Each key should be associated with a dictionary giving additional parameters. If defaults are to be used, the associated dictionary may be empty and is then simply given as `{}`. An example is given below.

```
"solute_parameters":{

    "Cl mg/l":{
        "C_old": 7.11,
        "alpha": {"Q": 1.0, "ET": 0.0}
    }
},
```

In this case, mesas.py will look for a time series of solute inflows in column `"Cl mg/l"` and produce predictions of the outflow concentrations associated with this input. Two additional parameters are specified: `"C_old"` gives the old-water concentration (C_{old}) and `"alpha"` corresponds to the α_q partitioning parameter in Eq. (9). In the given example, no chloride can leave the system through ET, as the corresponding value of α is zero. See Table 2 for more information.

3.2.3 Options

Additional options can also be set in the `"options"` dictionary of the parameter inputs. The available options are described in Table 3.

3.2.4 Time series

The time series input can be provided as a .csv file or as a Python pandas data frame. The order of the columns is not important, but the column names should be consistent with references to time series data in the SAS function specification, solute parameters, and options.

For example, to be consistent with the specifications given in the example in Sect. 3.2.1 and 3.2.2, the input data frame would have the following columns:

– `"Q"` and `"ET"`, which refer to the outflow rates (e.g., discharge and evapotranspiration, respectively) at

Table 2. Description of the keys that may optionally be associated with each solute in the "solute_parameters" section of the configuration file.

Key	Symbol(s)	Description
"C_old"	C_{old}	The old-water concentration: the concentration of all water released of unknown age. If "sT_init" is not specified, this will be all water in storage at $t = 0$; if "sT_init" is specified, it will be all water older than the last nonzero entry in "sT_init". The default is 0.0. It cannot be set as variable in time.
"k1"	k_1	The first-order reaction rate constant in Eq. (11). The default is 0.0. It may be variable in time if a time series column name is provided.
"C_eq"	C_{eq}	The equilibrium concentration in Eq. (11). The default is 0.0. It may be variable in time if a time series column name is provided.
"alpha"	α_q	A dictionary giving the partitioning coefficients for each outflow, as in Eq. (9). The default is 1.0. Dictionary keys must correspond to named outflow columns in the SAS specification. Each α_q may be variable in time if a time series column name is provided.
"sT_init"	$s_T(T,0)$	A list or array of values specifying the initial age-ranked storage distribution in the system. This is useful if the system is initialized by some sort of spin-up. Each entry is age-ranked storage in an age interval of duration Δt.
"mT_init"	$m_T(T,0)$	A list or array of values specifying the initial age-ranked mass in the system. This is useful if the system is initialized by some sort of spin-up. Each entry is age-ranked mass in an age interval of duration Δt. If "mT_init" is specified, "sT_init" must also be specified in the "options" and be of the same length. The element-wise ratio "mT_init"/"sT_init" gives the age-ranked concentration C_T of the water in storage at time zero. Note that if "sT_init" is specified but "mT_init" is not, the concentrations associated with each nonzero value of "sT_init" will be zero. The default is "None". It cannot be set as variable in time.

each time step. These are assumed to be average rates over the time step (rather than instantaneous rates at the start or end).

- "J", which gives the average inflow rate over each time step.
- "Cl mg/l", which gives the inflow concentration at each time step.
- "Q SAS function 1" and "Q SAS function 2", which are weights associated with the two component SAS functions that will be combined to give the SAS function for "Q". Note that a column for "ET SAS function" is not required because there is only one component.
- "S_ET" and "S_scale", which are time-varying parameters of the SAS functions.

After running, the output time series would include the following new columns:

- "Cl mg/l -> Q",
- "Cl mg/l -> ET".

These time series represent the concentration of the solute in those outflow fluxes. The values of "Cl mg/l -> ET" would all be zero, as the partitioning coefficient "alpha" associated with that solute and outflow was set to zero.

3.3 Running the model and querying results

The model is set up and run by instantiating a model object provided with all the needed input data and then calling its run method.

```
from mesas.sas.model import Model
my_model = Model(data_df='/path/to/data.csv',
 config='/path/to/config.json')
my_model.run()
```

The time series inputs and outputs will then be available in a data frame accessible as an attribute of the model object. For example, this would allow the user to plot the input and output concentrations.

```
import matplotlib.pyplot as plt

# Extract the timeseries
C_in = my_model.data_df['Cl mg/l']
C_out = my_model.data_df['Cl mg/l --> Q']
```

Table 3. Description of the keys that may optionally be in the `"options"` entry in the configuration file.

Key	Description
`"influx"`	This key is a string (the default is ”J”). It gives the name of the column in the time series dataset containing the inflow rate.
`"dt"`	This key is the time step Δt, such that Δt multiplied by any of the fluxes in the time series dataset gives the total volume of flux over the time step. The default is 1.0. It cannot be set as variable in time.
`"n_substeps"`	This key is an integer (the default is 1) that gives the number of substeps used in each time step of the calculation. Subdividing the time steps can increase the numerical accuracy of the solution and address some numerical issues, at the cost of longer run times. Note that the substep calculations are not retained in the output – only aggregate time step results are provided.
`"num_scheme"`	This key is an integer (the default is 4) defining the numerical scheme used to solve the SAS function. The default is a fourth-order Runge–Kutta scheme (`num_scheme=4`). Valid alternatives are a second-order accurate midpoint scheme (`num_scheme=2`) and a first-order accurate forward Euler scheme (`num_scheme=1`).
`"max_age"`	This key is an integer (the default is the length of the time series dataset) that gives the maximum number of age steps that will be calculated. This controls the number of rows in the output matrices. It is set to a smaller value than the default to reduce the calculation time (at the cost of replacing calculated concentrations of older water with the value of `"C_old"`)
`"sT_init"`	This key is a list or array (the default is a zero array of the length of the time series dataset) of the initial distribution of age-ranked storage (in density form). It is useful for starting a run using output from another model run, e.g., for the spin-up. If the length of this array is less than the length of the time series dataset, `"max_age"` will be set to the length of `"sT_init"`.
`"record_state"`	This key is a record the state variables at some or all time steps. A default value of `false` will still record the final state of the model. Note that setting this to `True` can greatly increase the memory requirements for large datasets. It can also be set to a string representing a column of Boolean values in `data_df`.
`"verbose"`	This key is a Boolean value (the default is `false`): print information about the calculation progress.
`"warning"`	This key is a Boolean value (the default is `true`): print warnings about calculation issues.
`"debug"`	This key is a Boolean value (the default is `false`): print a very large amount of information about the calculation progress. Do not use.

```
t = my_model.data_df.index

# Make the plots
plt.plot(t, C_in, label = "Cl in precip")
plt.plot(t, C_out, label = "Cl in discharge")

# Finishing touches
plt.legend(frameon=False)
plt.xlabel('time')
plt.ylabel('Cl [mg/l]')
```

Users can access further results by employing accessor functions. These can return the values for a particular time step, age step, or input time. The latter is useful for examining how water that entered at a particular time evolves in time. If none of these are given, the entire array is returned. Both density (`sT`, `pQ`, `mT`, `mQ`, and `mR`) and cumulative (`ST`, `PQ`, `MT`, `MQ`, and `MR`) forms are available.

```
# Make an array of ages to plot against
T = my_model.options['dt'] *
  np.arange(my_model.options['max_age'])

# Extract and plot the TTD at a particular
  timestep
pQ = my_model.get_pQ(timestep=100, flux='Q')
plt.figure()
plt.step(T, pQ, where='post')

# Extract and plot the volume of water
 in storage with an age less than 90
ST = my_model.get_ST(agestep=90)
plt.figure()
plt.step(T+1, ST, where='pre')

# Extract and plot the concentration
 of water in storage as it evolves due
  to evapoconcentration
sT = my_model.get_sT(inputtime=328)
mT = my_model.get_mT(inputtime=328,
  sol='Cl mg/l')
CT = mT/sT
plt.step(t, CT, where='post')
```

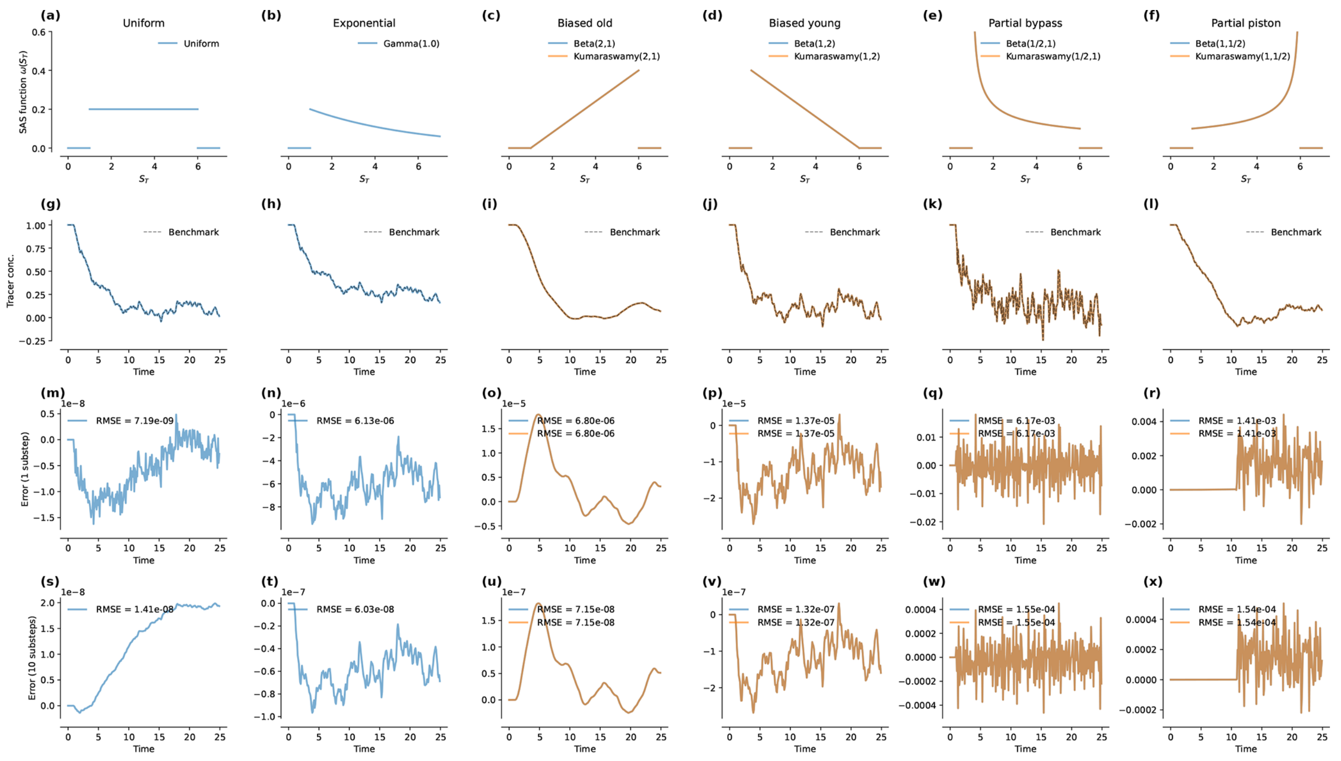

Figure 1. Results of the benchmark runs under steady flow. Panels **(a)** to **(f)** present the SAS functions used in each case. In each case, `"loc"`$= 1$ and `"scale"`$= 5$. The steady flow rate was $Q = 1$ and the time step was $\Delta t = 0.1$. Panels **(g)** to **(l)** show the predictions produced by mesas.py and the analytical benchmark solutions (dashed black lines). The inflow concentrations were Gaussian random variables with a mean of 1 and a standard deviation of 1. The initial concentration in storage was 1. Panels **(m)** to **(r)** give the absolute error relative to the benchmark with `"n_substeps"`$= 1$. Panels **(s)** to **(x)** give the error with `"n_substeps"`$= 10$. Note that the annotation “1e-6” on an axis indicates that the axis values are multiples of 10^{-6}.

More information on these functions is available in the documentation (see the “Code and data availability” section of this paper).

4 Code validation and comparison

To validate the numerical implementation, mesas.py was tested against several analytical benchmark solutions. Six of these are analytical solutions for different SAS functions under steady-flow conditions. Additional benchmark solutions for unsteady flow are identical to those presented for tran-SAS in Benettin and Bertuzzo (2018) and can, therefore, be used for comparison.

4.1 Validation against benchmarks: steady flow

4.1.1 Approach

For certain SAS functions, it is possible to find a closed-form expression for the corresponding TTD under steady flow. For the six cases considered here, the details of the derivations are given in Appendix A and the mathematical results are listed in Table 4. Several of these have been found previously (Botter, 2012; Harman, 2015; Berghuijs and Kirchner, 2017), although others are new.

The six cases (also shown in the top row of Fig. 1) are a uniform distribution; an exponential distribution; a “biased old” distribution and a “biased young” distribution, which encode a bias for older or younger storage, respectively, that varies linearly with age rank in storage; and a “partial piston” distribution and a “partial bypass” distribution, both of which encode a strong preference for the oldest and youngest storage, respectively. The latter four scenarios are special cases of both beta and Kumaraswamy distributions.

To assess the validity of our implementation of the numerical solution against these closed-form expressions, we can either (a) use very fine time steps, and thus more closely approximate the continuous result, or (b) find an analytical form of the discrete solution. The latter is preferable, as we can compare the numerical and analytical results directly, rather than asymptotically at the limit of small time steps. We have, therefore, taken the additional step of obtaining discrete versions of each expression (rightmost column of Table 4), which (when convolved with a synthetic time series of input concentrations) yield the average output concentration over each time step. These exact values can be compared

Table 4. Analytic solutions for the continuous and discrete TTD for a number of benchmark SAS functions with uniform, gamma, and beta distributions. The discrete form is obtained by averaging the value of p_Q over each age step and time step. In the time-variable cases, it is assumed that fluxes are constant over each time step, so $S_{j+1} = S_j + \Delta t(J_j - Q_j)$. For notational convenience, we have defined $\delta_j = \Delta T Q_j / S_j$; $\kappa = k\Delta t$; $\eta_j = S_{j+1}/S_j - 1$; $n = \frac{2}{\delta} - \frac{1}{2}$; and $\phi_j = \log(\eta_j + 1)/\eta_j$ if $\eta_j \neq 0$ or $\phi_j = 1$ otherwise.

Case	Flux $Q(t) =$	SAS function $\Omega(S_T, t) =$	Continuous TTD $p_Q(T,t) =$	Discrete TTD $\mathtt{p_Q}_{i,j} = \frac{1}{\Delta t \delta_j} \times$
Uniform	Q	Uniform$(0, S_0)$	$\frac{Q}{S_0} e^{-T\frac{Q}{S_0}}$	$\begin{cases} \delta + e^{-\delta} - 1 & i = 0 \\ e^{-(1+i)\delta}\left(e^{\delta} - 1\right)^2 & i > 0 \end{cases}$
Exponential	Q	Gamma$(1, S_0)$	$\frac{Q}{S_0}\left(T\frac{Q}{S_0} + 1\right)^{-2}$	$\begin{cases} \delta + \log\left(\frac{1}{\delta+1}\right) & i = 0 \\ \log\left(\frac{(\delta i+1)^2}{(\delta(i-1)+1)(\delta(i+1)+1)}\right) & i > 0 \end{cases}$
Biased old	Q	Kumaraswamy$(2, 1)$ or Beta$(2, 1)$ on $S_T \in [0, S_0]$	$\frac{2Q\tanh\left(T\frac{Q}{S_0}\right)}{S_0 \cosh^2\left(T\frac{Q}{S_0}\right)}$	$\begin{cases} \delta - \tanh(\delta) & i = 0 \\ \frac{2\sinh^2(\delta)\tanh(\delta i)}{\cosh((i-1)\delta)\cosh((i+1)\delta)} & i > 0 \end{cases}$
Biased young	Q	Kumaraswamy$(1, 2)$ or Beta$(1, 2)$ on $S_T \in [0, S_0]$	$2\frac{Q}{S_0}\left(T\frac{Q}{S_0} + 1\right)^{-3}$	$\begin{cases} \frac{\delta^2}{\delta+1} & i = 0 \\ \frac{2\delta^2}{(\delta(i-1)+1)(\delta i+1)(\delta(i+1)+1)} & i > 0 \end{cases}$
Partial bypass	Q	Kumaraswamy$(\frac{1}{2}, 1)$ or Beta$\left(\frac{1}{2}, 1\right)$ on $S_T \in [0, S_0]$	$-\frac{Q}{2S_0}\left(W\left(-e^{-\frac{QT}{2S_0}-1}\right)^{-1} + 1\right)^{-1}$	$\begin{cases} \delta + M(1) - 1 & i = 0 \\ M(i-1) - 2M(i) + M(i+1) & i > 0 \end{cases}$ $M(\ell) = W\left(-e^{-\frac{\delta\ell}{2}-1}\right)\left(W\left(-e^{-\frac{\delta\ell}{2}-1}\right) + 2\right)$
Partial piston	Q	Kumaraswamy$(1, \frac{1}{2})$ or Beta$\left(1, \frac{1}{2}\right)$ on $S_T \in [0, S_0]$	$\frac{Q}{2S_0}$ on $T \in [0, 2S_0/Q]$	$\begin{cases} \frac{\delta^2}{4} & i = 0 \\ \frac{\delta^2}{2} & 0 < i \leq n \\ (n - \lfloor n \rfloor)\frac{\delta^2}{2} & i = \lceil n \rceil \end{cases}$
Uniform and time-varying flux	$Q(t)$	Uniform$(0, S(t))$	$\frac{J(t-T)}{S(t)} \exp\left(-\int_{t-T}^{t} \frac{Q(\tau)}{S(\tau)} \mathrm{d}\tau\right)$	$\begin{cases} e^{-\delta_j\phi_j} + \delta_j - 1 & i = 0 \\ \frac{S_{j-i}}{S_j} \exp\left(-\sum_{k=0}^{i} \delta_{j-k}\phi_{j-k}\right) & \\ \times\left(e^{\delta_j\phi_j} - 1\right)\left(e^{(\delta_{j-i}+\eta_{j-i})\phi_{j-i}} - 1\right) & i > 0 \end{cases}$

directly to the numerical results, which are also intended to represent the average value over each time step.

In each scenario, the flow rate was set to $J(t) = Q(t) = 1$ and the time step to $\Delta t = 0.1$. The value of the scale parameter was set to $S_0 = 5$, and an offset of $S_{\min} = 1$ was used. Consequently, outflow concentrations are delayed relative to inflows by five time steps. Inflow concentrations were synthetically generated as independent, identically distributed random values (white noise) that were normally distributed with a mean of 0.0 and a standard deviation of 1.0. Initial concentration in storage (C_{old}) was set to 1.0. The `n_substeps` parameter was initially set to 1 and then increased to 10 to examine how a greater number of numerical substeps improved the solution accuracy.

4.1.2 Results

Figure 1g–l plots the tracer concentration predicted by mesas.py (blue and orange lines, as labeled in the first row) and analytical benchmark solutions (dashed black line). The random inputs are smoothed most by the biased old case, and they retain much of the input variability in the partial bypass case. The last two rows of Fig. 1 present the percent errors relative to the benchmark when one (Fig. 1m–r) or 10 (Fig. 1s–x) numerical steps are taken each time step.

When `n_substeps=1`, the root-mean-square errors (RMSEs) for the uniform case were the smallest (at 10^{-9}), whereas they were largest for the partial bypass and partial piston cases (whose errors are closer to 10^{-3}). For the other three cases, the errors were around 10^{-6}. When the number

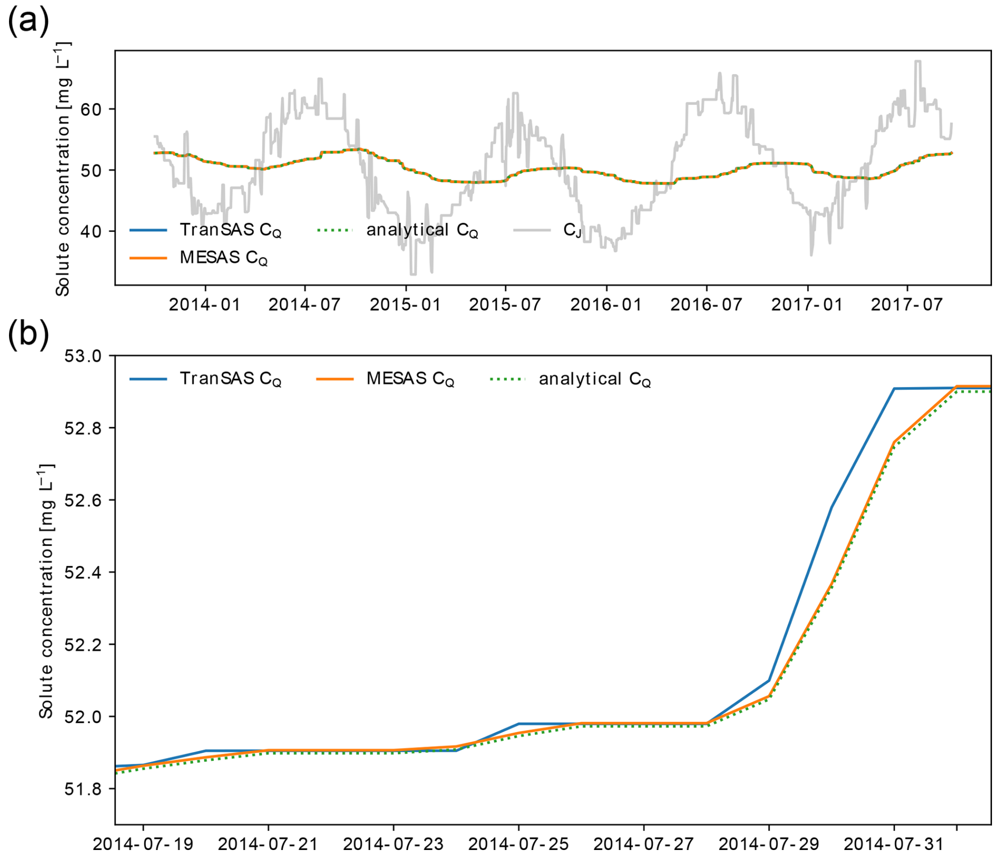

Figure 2. Comparison of mesas.py and tran-SAS for the case with uniform sampling and $S_0 = 1000$ mm. Panel **(a)** shows C_Q estimations from mesas.py (solid orange line) and tran-SAS (solid blue line) compared to the analytical solution (dashed green line); the solid gray line represents the input concentration from precipitation (C_J). Panel **(b)** presents a section of panel **(a)** in more detail, showing the errors resulting from the numerical scheme.

of substeps was increased to 10, the errors for those three cases (exponential, biased old, and biased young) decreased by a factor of 100. For the partial piston and partial bypass cases, the reduction was smaller (a factor of 9 and 40, respectively). For the uniform case, the error actually increased slightly, apparently due to a persistent (very small) bias associated with the initial concentration. When C_{old} was instead set to zero, the errors decrease by a factor of 2 when the substeps were increased.

In the partial piston case, errors are initially very low but suddenly become much larger later in the simulation. This occurs when the age-ranked storage of the oldest water in storage (that which entered in the first time step) reaches the "spike" on the right-hand end of the distribution. That is, using the definitions in Table 4, when $S_T(t,t) \approx S_0$. The high curvature of Ω_Q at that point most stresses the ability of the numerical solution to accurately determine p_Q.

4.2 Validation against benchmarks: unsteady flow

4.2.1 Approach

The power of the SAS approach comes from its ability to handle time-variable inflows and outflows. The bottom row of Table 4 gives the analytical solution for a case in which the SAS function is uniform but the flow rate is variable in time. The general analytical solution was presented in Botter (2012), but the discrete form given here is novel. The discrete form is derived from the general case by assuming that inflows and outflow are constant over a time step, so that the storage varies linearly. Further, the solution given is not the instantaneous PDF p_Q but rather p_Q averaged over an age step/time step along a characteristic curve. Therefore, it gives a precise estimate of the expected value of the fraction of discharge over each time step drawn from inputs in previous time steps.

This benchmark was used to validate the mesas.py code for the same dataset that Benettin and Bertuzzo (2018) used to validate the performance of tran-SAS. The dataset was downloaded from the repository cited in Benettin and

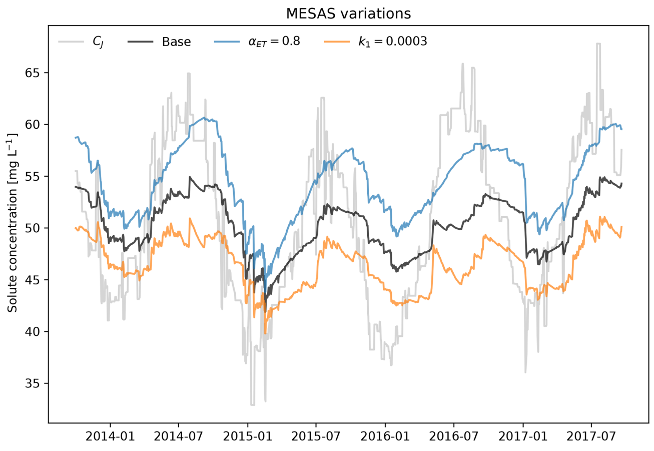

Figure 3. Variations of mesas.py applied to the time series and SAS model from Benettin and Bertuzzo (2018). The gray line shows the input concentration C_J from precipitation; the black line presents the base case, using the same setting as the tran-SAS model in Benettin and Bertuzzo (2018); the blue line denotes $\alpha_{\mathrm{ET}} = 0.8$; and the orange line denotes a first-order reaction rate $k_1 = 0.0003\,\mathrm{d}^{-1}$.

Bertuzzo (2018) and includes 8 years of 12 h precipitation, discharge, and evapotranspiration data. Benettin and Bertuzzo (2018) generated input concentrations by adding noise to a seasonal sinusoidal signal. The evapotranspiration was assumed to be drawn uniformly from the total storage in all simulations. Total storage at the end of each time step is calculated from the water balance assuming an initial storage S_{init}. The S_0 parameter used in mesas.py was calculated by averaging the total storage at the start and end of each time step.

4.2.2 Results

Figure 2 showcases the effect of activating some of the features of mesas.py: the ability to account for first-order reactions (for the case where the reaction rate is $3 \times 10^{-4}\,\mathrm{h}^{-1}$) and fractionation (for a case in which $\alpha_{\mathrm{ET}} = 0.8$, so that evapotranspiration enriches the tracer concentration in storage).

Figure 3 shows the input solute concentration as well as the output concentration predictions of mesas.py and tran-SAS for a case in which the discharge SAS function is uniform and $S_{\mathrm{init}} = 1000\,\mathrm{mm}$. Errors resulting from the numerical scheme are reduced in mesas.py relative to tran-SAS.

Closer inspection of the residuals between the model concentration predictions and the analytical benchmarks reveals differences between the performance of tran-SAS and mesas.py. Figure 4a and b show the time series and distribution of errors for tran-SAS and mesas.py for a case in which $S_{\mathrm{init}} = 1000\,\mathrm{mm}$. Although the overall distribution of absolute error magnitudes is similar, tran-SAS produces relatively large errors about 15 % of the time. Overall, the RMSE of mesas.py is 0.3 % (of the output standard deviation), whereas the RMSE for tran-SAS is 1.5 %.

These differences become larger when we consider the error in the solute mass flux, as shown in Fig. 4c and d. The RMSE of mass flux in mesas.py is 0.016 %, whereas it is about 15 times larger for tran-SAS (0.21 %).

The differences can be almost entirely attributed to the fact that tran-SAS provides estimates of the *instantaneous* transit time distribution at the end of each time step, whereas mesas.py estimates *time-step-averaged* values. It is also possible to obtain the TTDs for the end of the time step from mesas.py output and use them to estimate outflow concentrations. Those estimates have errors (shown in green in Fig. 4d) very similar to those of tran-SAS, as we would expect.

mesas.py performs better than tran-SAS for other configurations of the problem, although the size of the difference changes. The normalized RMSEs of each implementation are shown in Fig. 4f for four different values of initial storage. The results show that the normalized RMSE is larger for both codes when storage is small; however, mesas.py has a lower RMSE than tran-SAS in all cases.

We can also compare the performance of these models for a case in which the discharge SAS function is not uniform. Again, following Benettin and Bertuzzo (2018), we consider the case in which $\Omega_Q(S_T, t) = \left[\frac{S_T(T,t)}{S(t)}\right]^k$. This is equivalent to a Kumaraswamy distribution with $S_{\mathrm{min}} = 0$, $S_0 = S(t)$, $a = k$, and $b = 1$. The required parameter JSON file is given below.

```
"sas_specs":{

    "Q":{
        "Q SAS function":{
            "func": "kumaraswamy",
            "args": {
                "loc": 0.0,
                "scale": "S",
                "a": "k",
                "b": 1.0
            }
        }
    },

    "ET":{
        "ET SAS function":{
            "ST": [0.0, "S"],
            "P":  [0.0, 1.0]
        }
    }
},

"solute_parameters":{
    "C":{
        "C_old": 50,
    }
},
```

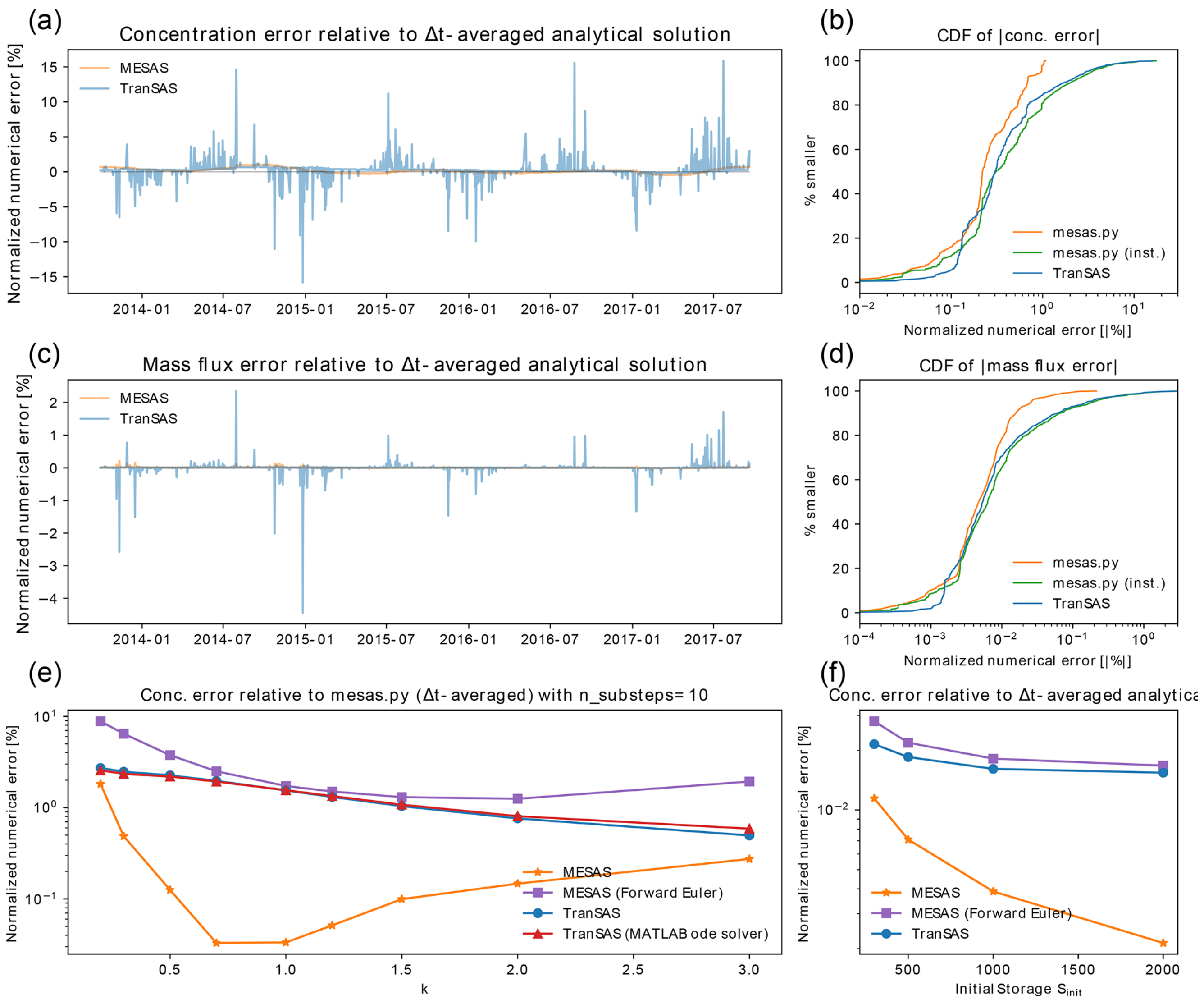

Figure 4. Comparison of mesas.py with tran-SAS for a benchmark problem presented by Benettin and Bertuzzo (2018). Panel **(a)** shows the normalized RMSE of outflow concentration estimates from mesas.py (orange) and tran-SAS (blue) relative to the time-step-averaged analytical solution with $S_{\text{init}} = 1000$ and $k = 1$. Panel **(b)** presents the absolute concentration error CDF for mesas.py and tran-SAS. Panels **(c)** and **(d)** are the same as panels **(a)** and **(b)** but show errors in the mass flux, rather than the concentration. Panel **(e)** displays the effect of varying $k \in [0.2, 0.3, 0.5, 1, 1.2, 1.5, 2, 3]$ with $S_{\text{init}} = 1000$ on the difference between each code's predictions and a run of mesas.py with 10 substeps as a high-accuracy estimate. Panel **(f)** shows the effect of varying $S_{\text{init}} \in [300, 500, 1000, 2000]$ with $k = 1$ on the concentration estimates of tran-SAS and mesas.py relative to the analytical solution.

```
"options":{
    "influx": "J"
    "n_substeps": 1
},
```

The time series dataset includes columns "Q", "J", "ET", "S", "k", and input concentrations "C".

As analytical solutions are unavailable for this more general case, the results obtained from tran-SAS and mesas.py were compared against a higher-accuracy mesas.py solution (obtained by setting "n_substeps" to 10). The RMSEs for a range of values of k are shown in Fig. 4c. The mesas.py RMSE was consistently lower than that of tran-SAS. The RMSE for the mesas.py forward Euler is less than that of tran-SAS because it does not include the modification introduced by Benettin and Bertuzzo (2018) to improve the accuracy of the first age step.

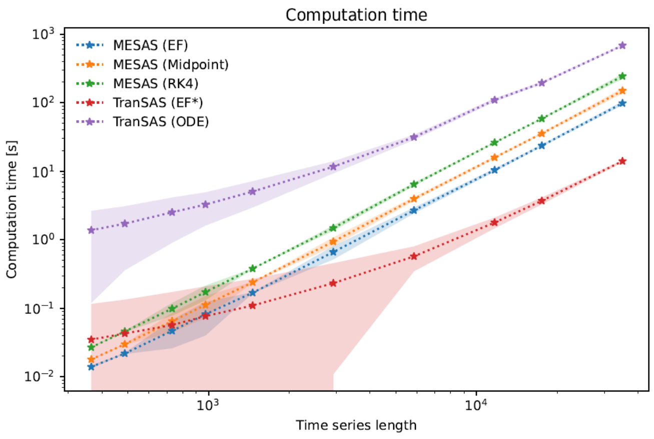

Figure 5. Comparison of the computational time and time series length for three mesas.py and two tran-SAS numerical computation settings. The plot shows the mean computational time and 95 % confidence interval for each case, represented by a line and corresponding colored region, respectively. The mesas.py cases include the EF (forward Euler), midpoint (midpoint method), and RK4 (fourth-order Runge–Kutta) methods, whereas the tran-SAS cases include the EF* (modified forward Euler) and ODE (MATLAB ODE solver) methods.

4.3 Performance

Computation time was compared by running all models for a benchmark configuration (with various lengths of time series) on a single-core Intel Xeon Gold Cascade Lake 6248R CPU with 192 GB DDR4 2933 MHz RAM. The benchmark configuration is the uniform sampling discharge SAS function case described above. Each configuration was run 100 times to establish confidence bounds on the run time.

As the results in Fig. 5 show, mesas.py and tran-SAS had comparable performance for time series less than 1000 time steps, but tran-SAS was generally computationally faster than mesas.py for longer time series. mesas.py was always faster than tran-SAS with the MATLAB ODE solver enabled. Using the midpoint and forward Euler numerical schemes in mesas.py resulted in a modest increase in performance (at the cost of numerical accuracy, as shown above).

5 Conclusions

SAS transport theory provides a very general framework for modeling material transport through control volumes (Benettin et al., 2022). At its core, it is based on a statement of conservation of mass of bulk material of different ages. This must be augmented with an SAS function that captures how outflow preferentially removes bulk material from storage according to the rank of its age. The mesas.py code presented here implements this theory, allows SAS functions to be expressed in a very flexible way, and solves the underlying equations with high accuracy with regard to mass balance.

The code is also intended to be user-friendly. A number of resources are available, including a free online course via HydroLearn. The course, entitled "JHU 570.412 Tracers and transit times in time-variable hydrologic systems: A gentle introduction to the StorAge Selection (SAS) approach" can be found at https://edx.hydrolearn.org/courses/course-v1:JHU+570.412+Sp2020/course/ (last access: 8 November 2023) (free registration required). This course includes three sections of introductory SAS function theory accompanied by a mesas.py walk-through.

Further work is needed to augment the code with additional useful tools. Three sets of tools are particularly important. First, tools for generating ensembles of input concentration data. In hydrology, observations of input concentrations are often bulk samples that represent amount-weighted averages over multiple time steps. These must be disaggregated to the resolution at which we want to run the model. Second, tools for parameterizing SAS functions and fitting them to data, preferably in a way that can adapt to any specification of the SAS function. Third, tools for assessing uncertainty in both the disaggregated inputs, the SAS function shape, and the model predictions.

Appendix A: Analytical benchmarks

A1 Steady-state solutions

At steady state with one outflow and $J = Q$, Eq. (2) can be written as follows:

$$\frac{\mathrm{d}s_T}{\mathrm{d}T} = -Q p_Q(T), \tag{A1}$$

$$s_T(0) = Q. \tag{A2}$$

This is a separable differential equation, which we can integrate twice to give the following:

$$S_T(T) = Q\left(T - \int_0^T P_Q(\tau)\,\mathrm{d}\tau\right). \tag{A3}$$

This gives the steady-state cumulative age-ranked storage in terms of the cumulative transit time distribution P_Q.

Alternatively, we can use the fact that $p_Q = \omega_Q s_T$ and $s_T = \mathrm{d}S_T/\mathrm{d}T$ to write the conservation law as follows:

$$\frac{\mathrm{d}S_T}{\mathrm{d}T} = Q - Q\Omega_Q(S_T(T)), \tag{A4}$$

$$S_T(0) = 0. \tag{A5}$$

This can likewise be integrated to obtain the age of water at a given location in age-ranked storage in terms of the SAS function:

$$T = \frac{1}{Q}\int_0^{S_T(T)} \frac{1}{1-\Omega_Q(\sigma)}\mathrm{d}\sigma. \tag{A6}$$

Using these equations, we can (in principle) find the steady-state transit time distributions if we know the SAS function, and vice versa. In many cases, a closed-form solution is not possible.

For example, for a uniform distribution $U(0, S_0)$, the SAS function is given by $\Omega_Q(S_T) = S_T/S_0$ for $S_T \in [0, S_0]$. Substituting this into Eq. (A6) gives $QT = -S_0 \log(1 - S_T/S_0)$, and rearranging gives $S_T(T) = S_0\left(1 - e^{\frac{Q}{S_0}T}\right)$. This is the cumulative age-ranked storage when Ω_Q is a uniform distribution and the flow is steady. Substituting this into the definition of the uniform Ω_Q yields the cumulative transit time distribution (as $P_Q(T,t) = \Omega_Q(S_T, t)$ by definition). Consequently, we can say that

$$\Omega_Q(S_T) = U(0, S_0) \Leftrightarrow P_Q(T) = 1 - e^{-\frac{Q}{S_0}T} \tag{A7}$$

at steady state. That is, a uniform SAS function is equivalent to an exponential TTD.

A similar set of steps can be used to show that an exponential SAS function (which is a special case of a gamma distribution with shape parameter $\alpha = 1$) is equivalent to a TTD following a Lomax distribution with exponent 1:

$$\Omega_Q(S_T) = \Gamma(1, S_0) \Leftrightarrow P_Q(T) = 1 - (1 + QT/S_0)^{-1}. \tag{A8}$$

We were not able to obtain a solution for the general case of a gamma distribution. Similarly, solutions for the TTD when the SAS function is given by a beta distribution $B(\alpha, \beta)$ over $0 \le S_T/S_0 \le 1$ could only be found for particular values of (α, β). For example, with $\alpha = 1$ and $\beta = 2$, we have the biased young case:

$$\Omega_Q(S_T) = B(1,2) \Leftrightarrow P_Q(T) = 1 - (1 + TQ/S_0)^{-2}, \tag{A9}$$

which is a Lomax distribution with exponent 2. Similarly the biased old, partial bypass, and partial piston cases are as follows:

$$\Omega_Q(S_T) = B(2,1) \Leftrightarrow P_Q(T) = \tanh^2(TQ/S_0), \tag{A10}$$

$$\begin{aligned}\Omega_Q(S_T) = B\left(\frac{1}{2}, 1\right) &\Leftrightarrow P_Q(T) \\ &= W\left(-e^{-\frac{QT}{2S_0}-1}\right) + 1,\end{aligned} \tag{A11}$$

$$\begin{aligned}\Omega_Q(S_T) = B(1,\beta) &\Leftrightarrow P_Q(T) = 1 - ((\beta - 1) \\ &QT/S_0 + 1)^{\frac{\beta}{1-\beta}}.\end{aligned} \tag{A12}$$

Here, $W(\cdot)$ is the Lambert W function.

A2 Accounting for discretization effects

To obtain a discrete form of the analytical solution, we can make two assumptions. First, that the time series of inflow concentrations and of water inflows and outflows are in fact constant within a time step Δt, so

$$\mathtt{C_J}_j = C_J(t) \;\text{ for }\; j \le t/\Delta t < j+1, \tag{A13}$$

$$\mathtt{J}_j = J(t) \;\text{ for }\; j \le t/\Delta t < j+1, \tag{A14}$$

$$\mathtt{Q}_j = Q(t) \;\text{ for }\; j \le t/\Delta t < j+1. \tag{A15}$$

Here, $\mathtt{C_J}_j$ and $\mathtt{C_Q}_j$ are the discrete forms of $C_J(t)$ and $C_Q(t)$.

Second, we assume that the numerical estimates of the outflow concentration time series should reflect the average value of the analytical solution over each time step. That is,

$$\mathtt{C_Q}_j = \frac{1}{\Delta t}\int_0^{\Delta t} C_Q(j\Delta t + \nu)\,\mathrm{d}\nu. \tag{A17}$$

In the continuous form, C_Q is obtained by the convolution of p_Q with C_J, as shown in Eq. (1). If we assume that the input concentrations are constant over each time step, Eq. (1) can be expressed as the sum of integrals over each time step interval $j\Delta t \le t < (j+1)\Delta t$ with the subsequent addition the old-water contribution:

$$\begin{aligned}
C_Q(t) &= \int_0^t C_J(t-T)p_Q(T,t)\,\mathrm{d}T + C_{\text{old}}P_Q(t,t) \\
&= \left(\sum_{i=0}^{j-1}\int_{t-(j-i))\Delta t}^{t-(j-i-1)\Delta t} C_J(t-T)p_Q(T,t)\,\mathrm{d}T\right) \\
&+ \int_0^{t-j\Delta t} C_J(t-T)p_Q(T,t)\,\mathrm{d}T + C_{\text{old}}P_Q(t,t) \\
&= \left(\sum_{i=0}^{j-1}\mathtt{C_J}_{j-i-1}\int_{t-(j-i))\Delta t}^{t-(j-i-1)\Delta t} p_Q(T,t)\,\mathrm{d}T\right) \\
&+ \mathtt{C_J}_j\int_0^{t-j\Delta t} p_Q(T,t)\,\mathrm{d}T + C_{\text{old}}P_Q(t,t) \\
&= \left(\sum_{i=0}^{j-1}\mathtt{C_J}_{j-i-1}\big(P_Q(t-(j-i-1)\Delta t, t)\right. \\
&\left. - P_Q(t-(j-i)\Delta t, t)\big)\right) + \mathtt{C_J}_j P_Q \\
&(t - j\Delta t, t) + C_{\text{old}}P_Q(t,t).
\end{aligned} \tag{A18}$$

To obtain the discrete outflow concentrations, we must apply the time step averaging in Eq. (A17) to Eq. (A18), which yields the discrete convolution:

$$\mathtt{C_Q}_j = \sum_{i=0}^{j}\mathtt{C_J}_{j-i}\mathtt{p_Q}_{i,j}\Delta t + C_{\text{old}}\mathtt{P_Q}_{j,j}. \tag{A19}$$

Here, the time-step-averaged TTD $\mathtt{P_Q}_{i,j}$ is given by

$$\mathtt{P_Q}_{i,j} = \frac{1}{\Delta t}\int_0^{\Delta t} P_Q(i\Delta t + \nu, j\Delta t + \nu)\,\mathrm{d}\nu. \tag{A20}$$

$\mathtt{p}_{\mathtt{Q}i,j}$ is obtained via the discrete derivative

$$\mathtt{p}_{\mathtt{Q}i,j} = \begin{cases} \dfrac{\mathtt{P}_{\mathtt{Q}i,j}}{\Delta t} & i = 0 \\ \dfrac{\mathtt{P}_{\mathtt{Q}i,j} - \mathtt{P}_{\mathtt{Q}i-1,j}}{\Delta t} & i > 0. \end{cases} \tag{A21}$$

For the elementary case of steady flow and uniform sampling, this gives

$$\mathtt{p}_{\mathtt{Q}i,j} = \frac{1}{\Delta t \delta} \times \begin{cases} \delta + e^{-\delta} - 1 & i = 0 \\ e^{-(1+i)\delta}\left(e^{\delta} - 1\right)^2 & i \geq 1, \end{cases} \tag{A22}$$

where $\delta = \Delta t Q / S_0$. Other forms are given in Table 4. Note that δ must be less than 1 in the exponential and biased young cases.

Code and data availability. mesas.py v1.0 is open source and distributed under the terms of the MIT License. The code is available on Zenodo (https://doi.org/10.5281/zenodo.7144730, Harman et al., 2023). Version 1.0 is tagged as https://github.com/charman2/mesas/releases/tag/v1.20231108 (last access: 8 November 2023). Users are encouraged to use GitHub's issue-tracking framework to submit bug reports and feature requests. Python 3.8 or higher is required.

The most up-to-date version of mesas.py along with its dependencies can be installed into a new environment from the command line using conda with the following command: `conda install -y -c conda-forge mesas`. You may need to run `conda config -append channels conda-forge` first.

The model can also be installed by building from the source code (see the documentation for steps required). A FORTRAN compiler is required to do so (but is not required when installing through conda). Documentation for mesas.py is available at https://mesas.readthedocs.io/en/latest/ (last access: 16 January 2024). This documentation is also stored on the GitHub repository. No datasets were used in this article.

Author contributions. CJH contributed to the conceptualization, methodology, formal analysis, investigation, software development, validation and evaluation, visualization, original draft preparation, funding acquisition, project administration, and supervision. EXF contributed to the software development, validation and evaluation, visualization, original draft preparation, and review and editing.

Competing interests. The contact author has declared that neither of the authors has any competing interests.

Acknowledgements. The authors are grateful to Eric Hutter, Oliver Evans and Fei Lu for their contributions to the mesas.py code. We would also like to thank Paolo Benettin and the anonymous reviewer for their helpful reviews.

Financial support. This work was supported by the US National Science Foundation Directorate of Geosciences (grant no. EAR-1654194).

Review statement. This paper was edited by Carlos Sierra and reviewed by Paolo Benettin and one anonymous referee.

References

Benettin, P. and Bertuzzo, E.: *tran*-SAS v1.0: a numerical model to compute catchment-scale hydrologic transport using StorAge Selection functions, Geosci. Model Dev., 11, 1627–1639, https://doi.org/10.5194/gmd-11-1627-2018, 2018.

Benettin, P., Rodriguez, N. B., Sprenger, M., Kim, M., Klaus, J., Harman, C. J., van der Velde, Y., Hrachowitz, M., Botter, G., McGuire, K. J., Kirchner, J. W., Rinaldo, A., and McDonnell, J. J.: Transit Time Estimation in Catchments: Recent Developments and Future Directions, Water Resour. Res., 58, e2022WR033096, https://doi.org/10.1029/2022WR033096, 2022.

Berghuijs, W. R. and Kirchner, J. W.: The Relationship between Contrasting Ages of Groundwater and Streamflow, Geophys. Res. Lett., 44, 8925–8935, https://doi.org/10.1002/2017GL074962, 2017.

Botter, G.: Catchment mixing processes and travel time distributions, Water Resour. Res., 48, https://doi.org/10.1029/2011WR011160, 2012.

Danesh-Yazdi, M., Klaus, J., Condon, L. E., and Maxwell, R. M.: Bridging the gap between numerical solutions of travel time distributions and analytical storage selection functions, Hydrol. Process., 32, 1063–1076, https://doi.org/10.1002/hyp.11481, 2018.

Dupas, R., Ehrhardt, S., Musolff, A., Fovet, O., and Durand, P.: Long-term nitrogen retention and transit time distribution in agricultural catchments in western France, Environ. Res. Lett., 15, 115011, https://doi.org/10.1088/1748-9326/abbe47, 2020.

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

Harman, C. J.: Age-ranked Storage-discharge Relations: A Unified Description of Spatially Lumped Flow and Water Age in Hydrologic Systems, Water Resour. Res., 55, 7143–7165, https://doi.org/10.1029/2017WR022304, 2019.

Harman, C. J.: Tracers and transit times in time-variable hydrologic systems: A gentle introduction to the StorAge Selection (SAS) approach, https://apps.edx.hydrolearn.org/learning/course/course-v1:JHU+570.412+Sp2020/home (last access: 8 November 2023), 2020.

Harman, C. J., Hutton, E., Xu Fei, E., Evans, O., and Lu, F.: charman2/mesas: mesas.py v1.20230728 (v1.20231108), Zenodo [code], https://doi.org/10.5281/zenodo.7144730, 2023.

Kendall, C. and Caldwell, E. A.: Chapter 2 – Fundamentals of Isotope Geochemistry, in: Isotope Tracers in Catchment Hydrology, edited by: Kendall, C. and McDonnell, J. J., 51–86, Elsevier, https://doi.org/10.1016/B978-0-444-81546-0.50009-4, 1998.

Rinaldo, A., Benettin, P., Harman, C. J., Hrachowitz, M., McGuire, K. J., Velde, Y. V. D., Bertuzzo, E., and Botter, G.: A coherent framework for quantifying how catchments store and release water and solutes, Water Resour. Res., 51, 4840–4847, https://doi.org/10.1002/2015WR017273, 2015.

Rodriguez, N. B. and Klaus, J.: Catchment Travel Times From Composite StorAge Selection Functions Representing the Superposition of Streamflow Generation Processes, Water Resour. Res., 55, 9292–9314, https://doi.org/10.1029/2019WR024973, 2019.

Rodriguez, N. B., Pfister, L., Zehe, E., and Klaus, J.: A comparison of catchment travel times and storage deduced from deuterium and tritium tracers using StorAge Selection functions, Hydrol. Earth Syst. Sci., 25, 401–428, https://doi.org/10.5194/hess-25-401-2021, 2021.

Ross, J., Schreiber, I., and Vlad, M. O.: Lifetime and transit time distributions and response experiments in chemical kinetics, in: Determination of Complex Reaction Mechanisms, Oxford University Press, https://doi.org/10.1093/oso/9780195178685.003.0014, 2006.

Rossum, J. M. V., Bie, J. E. G. M. D., Lingen, G. V., and Teeuwen, H. W. A.: Pharmacokinetics from a Dynamical Systems Point of View, J. Pharmacokinet. Biop., 17, 365–392, https://doi.org/10.1007/BF01061902, 1989.

Stockinger, M. P., Bogena, H. R., Lücke, A., Diekkrüger, B., Cornelissen, T., and Vereecken, H.: Tracer sampling frequency influences estimates of young water fraction and streamwater transit time distribution, J. Hydrol., 541, 952–964, https://doi.org/10.1016/j.jhydrol.2016.08.007, 2016.

Tyworth, J. E. and Zeng, A. Z.: Estimating the effects of carrier transit-time performance on logistics cost and service, Transport. Res. A-Pol., 32, 89–97, 1998.

Wilusz, D. C., Harman, C. J., Ball, W. B., Maxwell, R. M., and Buda, A. R.: Using Particle Tracking to Understand Flow Paths, Age Distributions, and the Paradoxical Origins of the Inverse Storage Effect in an Experimental Catchment, Water Resour. Res., 56, e2019WR025140, https://doi.org/10.1029/2019WR025140, 2020.