Technical note: Quadratic solution of the approximate reservoir equation (QuaSoARe)

Julien Lerat [1]

[1]CSIRO Environment, Canberra, ACT, 2601, Australia

Correspondence to: Julien Lerat (julien.lerat@csiro.au)

Abstract. This paper presents a method to solve the reservoir equation, a special type of scalar ordinary differential equation controlling the dynamic of conceptual reservoirs found in most hydrological models. The method called "Quadratic Solution of the Approximate Reservoir Equation" (QuaSoARe) applies to any reservoir equation regardless of its non-linearity or the number of fluxes entering and leaving the reservoir. The method is based on a piecewise quadratic interpolation of the flux functions, which leads to an analytical and mass-conservative solution. It is applied to two routing models and two rainfall-runoff stores that are representative of hydrological model components and evaluated on six catchments in eastern Australia that experienced one of the most extreme floods in recent Australian history. A comparison of the method against two standard numerical schemes, the Radau fifth-order implicit and Runge–Kutta of order 5(4) explicit schemes, suggests that it can reach similar accuracy while reducing runtime by a factor of 10 to 50 depending on the model considered. At the same time, our method is simple enough to be presented as a short pseudo-code included in our paper. Beyond solving a given reservoir equation, the method constitutes a promising avenue to define flexible models where flux functions are defined as piecewise quadratic functions, which can be solved exactly with QuaSoARe.

1 Introduction

1.1 Reservoirs as ubiquitous components of environmental models

Environmental models, including hydrological models, often rely on components that can be modelled conceptually as a reservoir receiving inputs and generating outputs that are sole functions of the volume stored in the reservoir. Such reservoirs are extensively used in rainfall-runoff models such as GR4J (Perrin et al., 2003), HBV (Bergstrom and Forsman, 1973), IHACRES (Croke and Jakeman, 2004) and SAC-SMA (Burnash and Ferral, 1981) where the reservoir dynamic is described by a differential equation relating the change in storage to inputs and outputs fluxes. If the model is applied to a time step long enough for storage to change significantly, this equation must be integrated to obtain the storage level at the end of the time step and the total of each flux. However, apart from a few simple cases, there is no analytical solution to this mathematical problem, and one has to revert to numerical approximations (Clark and Kavetski, 2010; Kavetski and Clark, 2010). Furthermore, flux functions in hydrological models are often highly non-linear, which magnifies numerical errors when using inappropriate numerical schemes (Kavetski and Clark, 2011). This, in turn, degrades model simulation and calibration due to the extra parameterisation needed to compensate for these errors (Kavetski et al., 2006). In this context, our paper presents an approximate analytical method called Quadratic Solution of the Approximate Reservoir Equation (QuaSoARe) to solve the scalar ordinary differential equation (ODE) underlying most conceptual reservoirs used in hydrology.

The integration of ODEs represents an entire field in applied mathematics with a history as old as differential calculus. The reader might wonder why a paper is needed on such a well-beaten scientific track. Despite the voluminous literature written on the topic, the large number of software packages available and the importance of proper ODE integration flagged by Kavetski and Clark (2010), the use of proven ODE numerical schemes remains rare in hydrological modelling. We suggest that this troubling fact may come about because the methods described in reference textbooks (Hairer et al., 2009; Shampine, 2020) aim at solving very general ODEs and hence require complex algorithms to handle every extreme scenario in the equation set-up. However, this complexity might be superfluous if one wants to solve a simple scalar equation such as a conceptual reservoir used in hydrological modelling. Our paper aims to bridge this gap by proposing a simple yet effective and mass-conservative approximate solution designed specifically for the scalar reservoir equation ODE.

More precisely, let us denote the volume stored in the reservoir by S and assume that the reservoir is submitted to forcing variables $\tilde{V}$ that remain constant over the time step. This assumption corresponds to most practical hydrological modelling scenarios where the time distribution of the forcings during the time step is unknown (e.g. time average of rainfall or potential evapotranspiration). In this case, the reservoir equation ODE is formulated as:

$$\frac{dS}{dt} = \sum_{i=1}^{n} f_i\left(S, \tilde{V}\right) \qquad \textbf{Eq. 1}$$

where f_i are arbitrary continuous functions representing input or output fluxes from the reservoir. Here we assume that the f_i are Lipschitz continuous, which ensures that Eq. 1 has a unique solution (Hairer et al., 2009). We also assume that Eq. 1 is input-to-state stable, which guarantees that its solution is bounded if the inputs $\tilde{V}$ are bounded (Mironchenko, 2023). Note that demonstrating the global stability of an ODE is a complex problem much beyond the scope of this paper. LaSalle (1960) presents a method using Lyapunov functions, which by Sontag (1989) later generalised for systems such as Eq. 1. Our mathematical assumptions and the theoretical limitations they introduce are further discussed in Section 4.

Solving Eq. 1 is an initial value problem over a time interval $[0,\ \delta]$ where δ is the time step. S_0 is defined as the initial condition at $t = 0$. To obtain a simulation from the reservoir, Eq. 1 is solved repeatedly for each time step with varying forcing variables (e.g. a daily series of rainfall values). It is highlighted that the presence of several functions f_i in Eq. 1 is common in hydrological models, for example, to account for multiple runoff generation processes such as infiltration and overland flow, or fluxes between surface water and groundwater stores (Clark et al., 2008). Other model components often require these fluxes (e.g. infiltration excess runoff being used as input to a routing model), which means that, in addition to solving for variable S, one must compute the total of each flux over the time step given by:

$$O_i = \int_0^{\delta} f_i(S, \tilde{V}) dt$$ **Eq. 2**

It is important to note that the computation of O_i does not affect the solution $S(t)$. To follow the terminology of atmospheric modelling, O_i is a diagnostic variable (American Meteorological Society, 2024) whereas S is a prognostic variable.
Eq. 1 has a broad range of applications beyond hydrological modelling, for example, to estimate storage in an artificial reservoir (Fiorentini and Orlandini, 2013) or to solve the gradually varied flow equation in hydraulics (Gill, 1976). Unfortunately, there are very few cases where both Eq. 1 and Eq. 2 have an analytical solution, a problem that is considerably more difficult than solving Eq. 1 alone. Consequently, most reservoir equations are solved using numerical approximation methods.

1.2 Numerical methods to solve the reservoir equation

The most common approach relies on discrete methods that estimate S from $t = 0$ to δ at incremental steps using Runge–Kutta methods (Kavetski and Clark, 2010; Knoben et al., 2019; La Follette et al., 2021). These schemes are the topic of a voluminous literature, including several reference textbooks (Butcher, 2003; Hairer et al., 2009; Press et al., 2007; Shampine, 2020 to name but a few). Nonetheless, applying them requires significant expertise to (1) select the most appropriate scheme among the multiple variants available (e.g. Euler, Runge–Kutta, or Huen schemes), (2) decide if the scheme is explicit (when the solution depends on S_0 only) or implicit (if it depends on both S_0 and $S(\delta)$, which requires iterative optimisation), and (3) choose between fixed or variable step lengths to automatically slow down computation when facing numerical difficulties. These choices are not trivial and far from harmless, as warned by Michel et al. (2003) and Kavetski and Clark (2011) who show the disastrous consequences of solving the exponential store with an explicit Euler scheme. As a result, these techniques are not simple to code, and a modeller unfamiliar with them will often require a third-party software package. This adds a dependency to the model code, complicates maintenance and increases runtime, sometimes significantly when using implicit methods, compared to an analytical solution. At the same time, despite the exponential growth of computing power, runtime is still a limiting factor in hydrology when a large number of runs is required, for example, in a Monte Carlo uncertainty analysis or for the calibration of distributed models.

In addition, besides applying the right ODE integrator to Eq. 1, solving the reservoir equation requires a numerical integration of Eq. 2 using a potentially different algorithm. For example, a simple quadrature method can be used to estimate the integral in the right-hand-side of Eq. 2 based on the two values S_0 and $S(\delta)$. However, this approach can be highly inaccurate if the solution $s(t)$ does not vary linearly with time, as demonstrated in the supplementary material. A more accurate method is to expand Eq. 1 into a system of differential equations by adding one differential equation for each flux:

$$\frac{dO_i}{dt} = f_i(S, \tilde{V}) \quad i = 1, \dots n$$ **Eq. 3**

The solution can then be obtained by applying the selected integrator to the system of ODEs combining Eq. 1 and Eq.3. However, because of the wide range of magnitudes observed in hydrological fluxes, this may lead to a system that could be characterised as “stiff”, for which many numerical schemes become unstable (Kavetski and Clark, 2011; Shampine, 2020).

Another angle of attack for solving an ODE is to replace the original equation with one for which an analytical solution exists. The simplest ODEs being linear, it is not surprising that linearisation of Eq. 1 around a specific regime (e.g. steady-state) has constituted the first approach proposed by mathematicians (Hartman, 2002). Linearisation has often been used to solve complex differential equations in hydrology and hydraulics, for example, the Saint-Venant 1D hydrodynamic equation by Hayami (1951) or more recently by Fan and Li (2006) or Munier et al. (2008). This approach is efficient if the true solution does not depart significantly from the linearisation regime. Unfortunately, hydrological systems often exhibit variations of several orders of magnitude that violate this assumption. A logical extension of the linearisation approach is to define several linear approximation regimes between which the solution can switch. This idea has been explored extensively in the control literature (Johansson, 2003) starting from early work by Kalman (1955). More recently, this approach has been formalised in the field of electrical engineering under the name Trajectory Piecewise-Linear Approximation (TPLA), a method introduced by Rewienski and White (2003) to solve large non-linear differential equation systems. The theory presented by Rewienski and White (2003) along with its subsequent refinements (Bond and Daniel, 2009; Kalra and Nabi, 2020) aims to solve equation systems far more complex than the reservoir equation studied here. Adapting this theory for the scalar reservoir equation, along with a clear algorithmic description, would be a valuable contribution from our side. In addition, the TPLA theory relies on a transition between linearised states which is not necessarily continuous. Finally, the choice of the transition function is arbitrary, which adds subjectivity to the process and does not guarantee that the derivative of the solution remains continuous. This can be an issue if the model shows strong non-linearity. This problem was raised by Litrico et al. (2010), who proposed a piecewise-linear model to approximate the Saint-Venant hydrodynamic equations. Overall, the TPLA method is useful for representing complex non-linear dynamics. Still, a simpler approach restricted to a scalar system such as the reservoir equation would likely make its adoption easier. Extending the linearisation idea, Pope (1963) introduced the concept of exponential integrator, where the linearisation of an ODE is combined with a discrete method such as a Runge–Kutta scheme applied to the residual between the linearised part and the original function. Hochbruck and Ostermann (2006) demonstrated this method's efficacy in solving large systems of stiff equations. However, its reliance on a discrete method to correct the linearised solution faces the same issues raised earlier for an application to hydrological models.

Overall, the review of the literature above highlighted the following research gaps:

- The reservoir equation is a standard tool in hydrology that requires fast and robust numerical solutions in the absence of general analytical approaches.
- Classical ODE numerical solvers such as Runge–Kutta methods are not straightforward, especially if the equations are stiff and lead to potentially unstable solutions.

- Existing theoretical developments from control theory based on multiple linearisation points are complex and require care when switching between linearised regimes.

1.3 Objectives of the paper

This paper aims to:

- present an approximate analytical solution for the reservoir equation. The solution solves Eq. 1 and computes all input and output fluxes from Eq. 2
- demonstrate the application of the method to four reservoir equations of increasing complexity and non-linearity and compare the results with classical implicit and explicit discrete methods.

The method, including its pseudo-code, is presented in Sections 2.2 and 2.3 while an accompanying python package is released as supporting material. Section 4 details some limitations of the method and recommendations to remediate them. The protocol used to compare our method with existing discrete methods is detailed in Section 5 with results presented in Section 6 and discussed in Section 7. The paper is concluded in Section 8.

2 Approximate analytical solution of the reservoir equation using piecewise quadratic functions

The method presented in this section is inspired by the TPLA and exponential integrators methods, where the functions f_i in Eq. 1 are approximated by functions for which an analytical solution of the reservoir equation exists.

2.1 Illustrative example

Before presenting our method, we introduce an illustrative example using the following reservoir equation:

$$\frac{dS}{dt} = -\frac{S^3}{2} \qquad \textbf{Eq. 4}$$

This equation is a special case of the cubic flow routing model, further discussed in Section 5.1. Assuming an initial condition S_0 at $t = 0$, the analytical solution of Eq. 4 is:

$$s(t) = \frac{S_0}{\sqrt{1 + t\, S_0^2}} \qquad \textbf{Eq. 5}$$

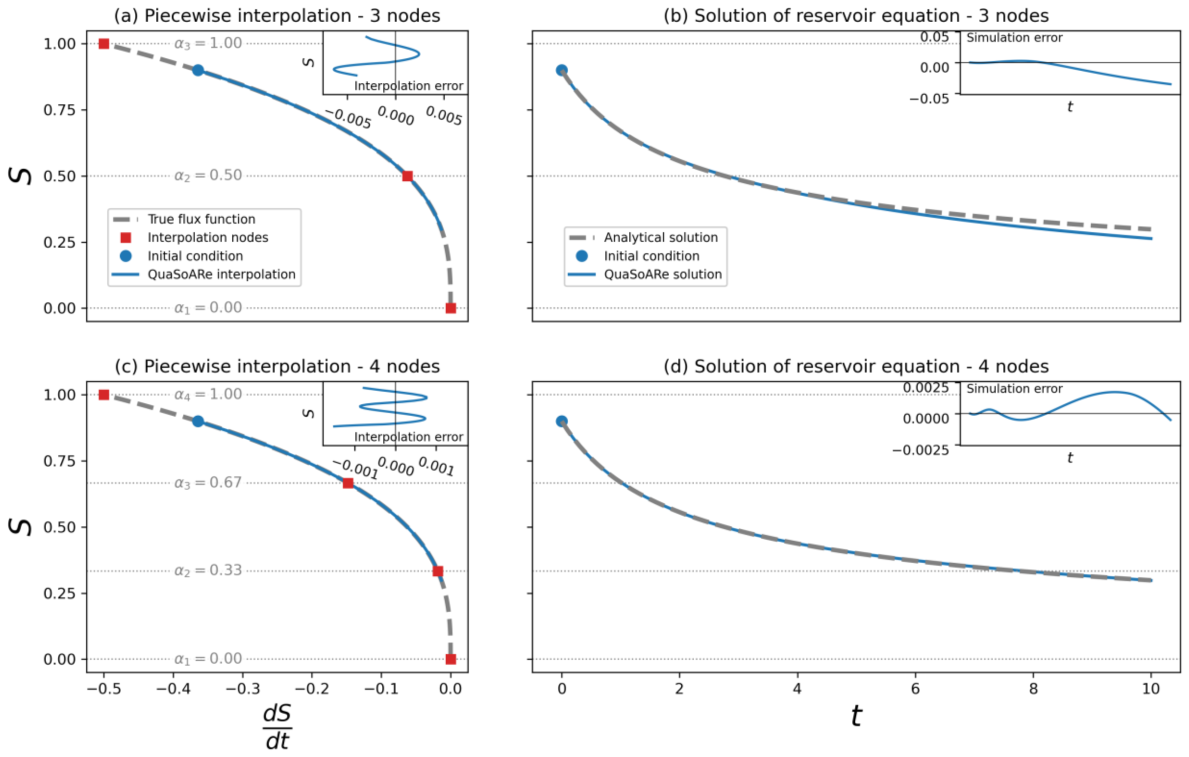


Figure 1: QuaSoARe method applied to the example reservoir equation. The left plots (a and c) present the reservoir equation (i.e. the right-hand side of Eq. 4) as a grey dashed line and its QuaSoARe approximation as a blue line. The right plots (b and d) show this equation's analytical and QuaSoARe solutions using the same colours. A QuaSoARe configuration with three interpolation nodes is used in the two top plots (a and b), while four nodes are used in the two bottom plots (c and d).

The reservoir equation function from the right-hand side of Eq. 4 is plotted in Figure 1.a as a grey dashed line. The corresponding solution from Eq. 5 is shown in Figure 1.b as a grey dashed line using a value of S_0 equal to 0.9. Other elements in this figure are related to the QuaSoARe method and are described in the following sections.

2.2 Reservoir equation approximation by quadratic functions

Before presenting our approximate solution, we need to establish that the solution $s(t)$ of Eq. 1 is monotonic, a result that is used repeatedly across this paper. This can be proved by contradiction: let us assume that $s(t)$ is not monotonic. Consequently, s being derivable as a solution of Eq. 1, there exists a time $t_0 > 0$ where the derivative of s changes sign and $ds/dt \neq 0$ for $t > t_0$. Let us now introduce the function s_2 which is identical to s for $t < t_0$ and which remains constant and equal to $s(t_0)$ for $t \geq t_0$. This function is a solution of Eq. 1 for all $t > 0$ and is distinct from s for $t > t_0$ because its derivative is zero, while the derivative of s is not. This is a contradiction with the uniqueness of the solution of Eq. 1 imposed by the fact that f is Lipschitz continuous, as stated in the previous section. Consequently, s is monotonic for $t \geq 0$.

Back to solving Eq. 1, let us assume that each function f_i can be approximated by a function f_i^* written as:

$$f_i^*(S,\tilde{V}) = \sum_{j=1}^{m-1} \left[a_{i,j}\, S^2 + b_{i,j}\, \mathrm{S} + c_{i,j}\right] \mathrm{I}_j(S) \qquad \textbf{Eq. 6}$$

where the series $\alpha_1 < \alpha_2 < \cdots < \alpha_m$ defines a partition of the interval $[\alpha_1, \alpha_m]$ into $m-1$ intervals referred to as "interpolation bands". In addition, the interval $[\alpha_1, \alpha_m]$ is assumed to contain the bounds of $s(t)$ (solution to Eq. 1 is assumed bounded in Section 1.1). $\mathrm{I_j}(x)$ is the indicator function equal to 1 if $x \in [\alpha_j, \alpha_{j+1}[$ and zero elsewhere. The α_j are referred to as "interpolation nodes" in this paper. The coefficients $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$ are fixed. The process to obtain $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$ is detailed in Appendix A so that the approximated function f_i^* matches the original function f_i at the nodes α_j and at the midpoint between the α_j.

The approximation described above is applied to our example equation given in Eq. 4: two approximations are presented in Figure 1.a and in Figure 1.c using three and four nodes, respectively. Both lead to a highly accurate interpolation where true and approximated functions are visually indistinguishable. The interpolation error, that is, the difference between the true function and its approximated counterpart, is shown as inset in both plots. This error is reduced by a factor of approximately five when the number of nodes increases from three to four.

The form of Eq. 6 was chosen for several reasons. First, a piecewise quadratic function is Lipschitz continuous on a bounded interval. As a result, the solution to the corresponding reservoir equation, referred to as the "approximated reservoir equation", is unique like the one of Eq. 1. Second, this equation can be solved analytically, as is shown in Section 2.3. Third, a quadratic function can approximate a wide range of reservoir functions used in hydrology. For example, if $a_{i,j} = 0$, the equation becomes a linear function of S, which is the most common reservoir equation used in hydrology. Finally, the steady-state solution of the approximate reservoir equation (i.e. when the derivative of S is zero) can be determined analytically, which greatly facilitates the analysis of the behaviour of the reservoir and the selection of the α_j as discussed in Section 3. However, despite these appealing attributes, there are also downsides to this approximation, which are presented in Section 4 along with potential remediation.

Replacing the original functions f_i by their approximated counterpart f_i^*, the approximate reservoir equation is:

$$\frac{dS}{dt} = \sum_{i=1}^{n} f_i^*(S,\tilde{V}) = \sum_{j=1}^{m} \left[A_j\, S^2 + B_j\, \mathrm{S} + C_j\right] \mathrm{I}_j(S) \qquad \textbf{Eq. 7}$$

where A_j, B_j and C_j are the sums of the flux coefficients for the interpolation band j. For example, A_j is given by:

$$A_j = \sum_{i=1}^{n} a_{i,j} \qquad \textbf{Eq. 8}$$

Importantly, Eq. 7 maintains an equality between the change in storage and the sum of fluxes, meaning it conserves mass. This statement is trivial, but it ensures that the QuaSoARe algorithm generates mass conservative simulations, which is a key requirement in hydrological modelling and is not always guaranteed by ODE numerical schemes, as pointed out by Clark and Kavetski (2010).

2.3 Analytical solution of the approximated reservoir equation

The solution to Eq. 7 can be obtained analytically as follows. Let us assume that the initial condition S_0 is falling in the interpolation band $[\alpha_{j_0}, \alpha_{j_0+1}[$ where $j_0 \in [1, m-1]$. For example, $j_0 = 2$ (second band) when $S_0 = 0.9$ in our illustrative example shown in Figure 1.a. When S_0 falls into the j_0^{th} interpolation band at $t = 0$, Eq. 7 can be simplified as:

$$\frac{dS}{dt} = A_{j_0} S^2 + B_{j_0} S + C_{j_0} \qquad \textbf{Eq. 9}$$

The solution of this equation has an analytical expression, referred to as $s(t)$. In addition, all fluxes from Eq. 2 can also be computed analytically. The process to obtain these expressions is not complex but tedious so its presentation is deferred to Appendix B. Using solution $s(t)$, one can compute the value $s(\delta)$ at the end of the time step, leading to three cases:

- **Case 1**: $s(\delta)$ lies in the interval $[\alpha_{j_0}, \alpha_{j_0+1}[$: because it is monotonic (see beginning of Section 2.2), $s(t)$ remains bounded by α_{j_0} and α_{j_0+1} for all t in $[0, \delta]$. Consequently, Eq. 7 remains identical to Eq. 9 for $t \in [0, \delta]$, which means that $s(t)$ is the solution of equation Eq. 7 over the interval $[0, \delta]$.
- **Case 2**: $s(\delta) < \alpha_{j_0}$: here, $s(t)$ is a continuous function with $s(0) = S_0 \geq \alpha_{j_0}$ and $s(\delta) < \alpha_{j_0}$. Hence, by the intermediate value theorem, there exists a time $t_l < \delta$ such that $s(t_l) = \alpha_{j_0}$. Moreover, $s(t)$ is decreasing, which means that $s(t)$ remains in the interval $[\alpha_{j_0}, S_0]$ for all $t < t_l$. In other words, $s(t)$ is the solution of Eq. 7 for t in $[0, t_l]$. The expression for t_l has an analytical expression equal to $\nu(\alpha_{j_0}) - \nu(S_0)$ where function ν is given in Eq. 23 (see Appendix B).

  When $t > t_l$, Eq. 7 is no longer equivalent to Eq. 9 because $s(t)$ becomes lower than α_{j_0}. As a result, $s(t)$ is no longer the solution of Eq. 7. However, Eq. 7 becomes equivalent to an equation similar to Eq. 9 where A_{j_0} and B_{j_0} are replaced by A_{j_0-1} and B_{j_0-1}, respectively.
- **Case 3**: $s(\delta) > \alpha_{j_0+1}$: following a similar reasoning than above, $s(t)$ is a strictly increasing function until it reaches the value α_{j_0+1} at time $t_u < \delta$ where t_u is equal to $\nu(\alpha_{j_0+1}) - \nu(S_0)$. This means that s is the solution of Eq. 7 for $t \in [0, t_u]$.

  For $t > t_u$, s becomes greater than α_{j_0+1} and is no longer the solution of Eq. 7. However, Eq. 7 then becomes equivalent to an equation similar to Eq. 9 where A_{j_0} and B_{j_0} are replaced by A_{j_0+1} and B_{j_0+1}, respectively.

Overall, the first case above leads to an immediate resolution of Eq. 7 over the interval $[0, \delta]$, while the other cases provide a solution over two shorter intervals $[0, t_l]$ and $[0, t_u]$ corresponding to cases 2 and 3, respectively. For these last two cases, if t is greater than t_u or t_l, Eq. 7 becomes equivalent to an equation similar to Eq. 9 where the index j_0 is replaced by either $j_0 - 1$ or $j_0 + 1$, which suggests that the whole process can be repeated iteratively towards the end of the interval $t = \delta$.

It is worth mentioning that the value $s(\delta)$ in the above algorithm may not be defined. Specific values of the coefficients A_j, B_j, C_j can lead to a solution $s(t)$ becoming infinite before reaching $t = \delta$. The last column of Table 3 in Appendix B indicates the time interval during which $s(t)$ remains bounded depending on these coefficients. This situation obviously excludes Case 1 above but can be captured under Case 2 or 3 if the invalid value $s(\delta)$ is replaced by $+\infty$ or $-\infty$, if s is increasing or decreasing, respectively. Note that this adjustment of the algorithm ensures that it can cope with an invalidity of $s(\delta)$ but does not guarantee its systematic convergence. This question is discussed further in Section 4.

It is important to highlight that each step in the algorithm described above is explicit because the calculation of $s(\delta)$ depends on past values of S only. In addition, the underlying analytical solution (see Appendix B) is computed using a limited set of standard mathematical functions (exponential, logarithm, hyperbolic tangent and tangent). Both elements suggest that the QuaSoARe method is simple and fast to implement in any programming language. Finally, each QuaSoARe iteration is repeated at most $m + 1$ times if the entire range $[\alpha_1, \alpha_m]$ is traversed by the solution. Consequently, the runtime required to compute the approximate solution is bounded by the number of interpolation nodes. It cannot reach high values like what can happen with algorithms relying on variable time step size like certain Runge–Kutta methods.

QuaSoARe is applied to our illustrative example with the approximated solution shown in Figure 1.b (interpolation using three nodes) and Figure 1.c (four nodes) as blue lines. QuaSoARe solutions closely match the analytical solution (dashed grey line) for both interpolation configurations. However, the match degrades towards the end of the simulation in Figure 1.b with QuaSoARe underestimating the true solution. This is due to the negative interpolation error when S is lower than 0.5, as seen in the inset of Figure 1.a, which leads to a more rapid decrease of the approximated solution and, hence, progressive underestimation. This example highlights the potential of QuaSoARe to generate accurate simulations, as well as the need for high accuracy in the interpolation of flux functions.

In summary, the solution presented in this section provides a way to solve an approximate reservoir equation where piecewise quadratic interpolations replace the original reservoir functions. The solution is fully analytical, including the computation of all reservoir fluxes. It can be implemented using the pseudo-code presented in Figure 2. Alternatively, Lerat (2025) released an open-source software package where QuaSoARe is coded in both Python and C languages.

```
QuaSoARe($\boldsymbol{\alpha}, \mathbf{A}, \mathbf{B}, \mathbf{C}, t_0, S_0, \delta$)
// $\boldsymbol{\alpha} \in \mathbb{R}^m$ : interpolation nodes
// $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^{m-1\times n}$ : interpolation coefficients for each band and flux
// $t_0$: start time
// $S_0$: initial storage level
// $\delta$ : time step

// Initialisation
$t_1 = t_0$
$\mathbf{O} \in \mathbb{R}^n = \mathbf{0}$ // fluxes set to 0
$j = \arg\max_k (\boldsymbol{\alpha}[k] < S_0)$ // Find the starting interpolation band $j_0$

while $t_1 \leq \delta$
    // Sum of interpolation coefficients across fluxes
    $A_j = \sum_{i=1}^{n} \mathbf{a}[i, j]$
    $B_j = \sum_{i=1}^{n} \mathbf{b}[i, j]$
    $C_j = \sum_{i=1}^{n} \mathbf{c}[i, j]$

    // Bounds of interpolation band
    $S^- = \boldsymbol{\alpha}[j]$
    $S^+ = \boldsymbol{\alpha}[j + 1]$

    // Tentative solution at the end of the time step
    // (Table 3, second column)
    $s(\delta) = S(A_j, B_j, C_j, S_0, \delta)$

    // Move iteration forward in time
    if $s(\delta) \in [S^-, S^+[$
        // Solution stays in the current interpolation band
        $t_1 = \delta$
        $S_1 = s(\delta)$
    else
        // Solution leaves the current interpolation band
        if $s(\delta) < S^-$
            $j^* = j - 1$
            $S^* = S^-$
        else
            $j^* = j + 1$
            $S^* = S^+$

        // Find time when solution leaves the band
        // (Table 3, first column)
        $t_1 = t_0 + \nu(A_j, B_j, C_j, S^*) - \nu(A_j, B_j, C_j, S_0)$

        // Update band number
        $j = j^*$

    // Increment fluxes (Equation 29, Appendix B)
    for $i = 1$ to $n$
        $\mathbf{O}[i] = \mathbf{O}[i] + O_i^*(t_0, t_1)$

    // Loop
    $S_0 = S_1$
    $t_0 = t_1$

return $S_1, \mathbf{O}$
```

Figure 2: QuaSoARe pseudo-code.

3 Selecting interpolation nodes from the range of steady-state solutions

The QuaSoARe method presented in Section 2 relies on an interpolation of functions f_i using nodes $\{\alpha_j\}_{j=1,\ldots,m}$. The choice of these nodes is crucial because it conditions the quality of the interpolation and, hence, the accuracy of the approximate solution. Let us now assume that the reservoir equation is solved for a fixed time step δ and a time series of p forcing values $\{\tilde{V}_k\}_{k=1,\ldots p}$. In most reservoirs used in hydrology, the reservoir equation has at least one steady-state solution for each time step k, denoted $\bar{S}_k$, that is solution of:

Eq. 10

$$\sum_{i=1}^{n} f_i\left(\bar{S}_k, \tilde{V}_k\right) = 0$$

If the set of all solutions $\{\bar{S}_k\}$ is not empty, a simple approach is to set α_1 to $\min(\{\bar{S}_k\})$ and α_m to $\max(\{\bar{S}_k\})$. Subsequently, the α_i are obtained by partitioning the interval $[\alpha_1, \alpha_m]$ into $m - 1$ equal sub-intervals. This approach relies on the fact that extreme values of $s(t)$ are reached during time steps where the forcings $\tilde{V}_k$ are likely to be extremes. At the same time, steady-state solutions are storage values that are reached when the integration time step δ tends to infinity. Consequently, when an extreme forcing value is used, a simulation run for an infinite time step is likely to result in storage values that are higher than any other values of $s(t)$ seen when using a finite time step, hence constituting a conservative estimate for their bounds.

In addition, the steady-state solutions, and hence their bounds, are straightforward to compute with QuaSoARe because, if they exist, they are the roots of a quadratic polynomial (right-hand side of Eq. 9) which can be computed analytically from the coefficients A_j, B_j and C_j. The corresponding functionality is included in the QuaSoARe software package (Lerat, 2025).

Unfortunately, the existence of steady-state solutions is not guaranteed for all reservoir equations. For example, the exponential reservoir with zero inflows does not have one (Michel et al., 2003). In cases like that, we recommend using a trial-and-error approach to obtain a set of α_j that covers the entire range of $s(t)$ observed during the simulation.

4 Limitations of the method and recommendations

The QuaSoARe method is designed to solve a scalar ordinary differential equation. Hence, it cannot be used to solve a coupled system of equations. This is an important limitation of QuaSoARe as most hydrological models contain multiple stores that could benefit from a joint solution using a Runge–Kutta method, as presented by Clark and Kavetski (2010). Extension of QuaSoARe to higher dimensions is not straightforward because the analytical solutions underpinning the method do not have a vector equivalent. However, in the case where the model reservoirs operate in sequence with no feedback, a simple solution is to apply QuaSoARe to each reservoir in turn at a finer time interval than the desired time step. The fluxes generated at this finer time step can subsequently be used to feed the next reservoir in the model structure. This is arguably less efficient from a runtime perspective than applying QuaSoARe over the whole-time step but probably not dissimilar to discrete methods that often shorten the time step to very short sub-steps to control the error.

The second limitation of QuaSoARe is the quality of the piecewise quadratic interpolation. Figure 1 clearly shows that minor discrepancies between the true and interpolated flux functions can lead to noticeable simulation errors. The solution to this problem is to increase the number of interpolation nodes, as was shown in our illustrative example in Figure 1 where the interpolation errors are reduced by a factor of five when switching from three to four nodes. This is straightforward to implement if a modeller starts with a high number of nodes, leading to an interpolation error smaller than their machine precision, as is done in Section 5. Such a small error level is theoretically achievable as the reservoir functions are assumed to be continuous and hence can be approximated up to any error level by a piecewise polynomial. If this configuration exceeds the modellers' runtime requirement, then the number of nodes is progressively reduced to match this constraint.

Particular care should be taken with the interpolation of the reservoir function close to steady-state values discussed in the previous section. Functions with sharp transitions (e.g. rational fractions) cannot be interpolated accurately by a quadratic polynomial over large intervals. Consequently, if applied without constraint, the piecewise interpolation can overshoot and create erroneous steady-state values that do not exist in the original equation (i.e. values of S where $f_i^*(S,\tilde{V})$ is null but not $f_i(S,\tilde{V})$). To avoid this problem, a constraint is imposed in the computation of the interpolation coefficients presented in Appendix A to restrict the quadratic functions to be monotonic and prevent them from crossing the 0 line if the original flux function did not.

Another limitation of QuaSoARe comes from the mathematical assumptions introduced in Section 1. More specifically, there is a need for flux functions to be Lipschitz continuous, which is equivalent to having bounded derivatives if the function is absolutely continuous. This assumption is required to ensure the unicity of the solutions of Eq. 1, and hence its monotonous nature, as demonstrated at the beginning of Section 2.2, but eliminates many common flux functions encountered in hydrological systems (e.g. power functions of S with an exponent lower than 1). A first solution to this problem is to alter the flux functions to obtain smoother functions with bounded derivatives following Kavetski et al. (2006, see Section 5 of their paper) and, hence, revert to the domain of applicability of QuaSoARe. If this is not an option, QuaSoARe may still generate reasonable simulations if the solution of Eq. 1 does not come close to any discontinuity in the flux functions derivatives. This is, of course, case specific. An example of such a case is presented in supplementary material.

The discussion above highlights that QuaSoARe, like all numerical ODE solvers, is not guaranteed to converge for all reservoir equations and initial conditions. Reservoir equations often define stability regions where solutions starting from similar initial conditions remain close for all $t > 0$. If the interpolation accuracy is low, it is possible that the approximate and original stability regions do not coincide. As a result, certain values of the initial conditions may lead to a stable solution for the original equation but not for QuaSoARe. Fortunately, this problem is related to interpolation accuracy and can be diagnosed before running QuaSoARe. Similarly to what is mentioned above, it is recommended that QuaSoARe is first run using a high number of interpolation nodes to ensure that stability regions are approximated accurately. Ultimately, the validity interval of the analytical solutions can be verified at runtime using expressions given in Table 3. Consequently, it is possible to catch this type of problem before attempting a QuaSoARe iteration. Note that this case was never encountered in practice while running the tests presented in the following section, where QuaSoARe is applied to a range of non-linear reservoir models using challenging hydro-climate data.

5 Comparison of the method with alternative numerical schemes

5.1 Reservoir equations tested

The QuaSoARe method is applied to four reservoir equations detailed in Table 1 representing common hydrological models of increasing complexity and non-linearity. All equations in this table depend on a storage scaling factor θ expressed in the same unit as S. This factor acts like a storage capacity and controls the dynamic of the reservoir response. Ten values of θ are evaluated for each equation using bounds for θ indicated in the last column of Table 1.

The first two equations simulate the routing of an inflow time series from the upstream to downstream end of a river reach. This type of routing model is used in semi-distributed hydrological or flood forecasting models to approximate the solution of hydrodynamic equations (Hapuarachchi et al., 2022). The reach receives an inflow, denoted $Q_{inflow}(t)$, assumed fixed over the time step t. Outflow from the reach is computed as a power function of the reach storage following Yevdjevitch (1959) where β is the power exponent multiplied by a reference flow, denoted Q_{ref}, introduced to simplify dimensional analysis (see Table 1). The routing model is run at an hourly time step. The case where $\beta = 1$ corresponds to the linear routing model (Meyer, 1941), while $\beta = 2$ is the quadratic routing store solved analytically by Bentura and Michel (1997). Both cases can be treated as quadratic functions of the storage and hence be solved exactly with QuaSoARe. To provide a more meaningful challenge for our method, the two cases where $\beta = 3$ (cubic reservoir, denoted CR) and $\beta = 6$ (bi-cubic reservoir, denoted BCR) are selected.

The third equation corresponds to the production store of the GR4J model (Perrin et al., 2003). This model is a well-established daily conceptual rainfall-runoff model used worldwide. Its production store, referred to as GR, receives rainfall (P) and potential evapotranspiration (E) and generates three fluxes: infiltrated rainfall into the store, actual evapotranspiration from the store and percolation leaked from the store. The first two fluxes are quadratic functions of the store level, which could be solved exactly with QuaSoARe. However, the percolation flux introduces a strong nonlinearity with a fifth-order polynomial. Note that the version of GR used in this paper computes the three fluxes simultaneously in a single equation, similarly to Santos et al. (2018), whereas they are solved sequentially using the operator splitting method in the original GR4J model presented by Perrin et al. (2003).

The fourth equation, denoted GRM (GR modified), is inspired by the GR equation but increases the nonlinearity of the fluxes by converting the infiltrated rainfall and the actual evapotranspiration to fifth-order polynomials. In addition, a fourth flux is added to represent groundwater recharge in the form of a rational fraction. Rational fractions are difficult to interpolate with polynomials as they possess asymptotes, which leads to a challenging case for QuaSoARe.

We highlight that the main objective of introducing these four reservoir equations is to obtain challenging tests for QuaSoARe that are representative of hydrological models in use. The aim is not to improve existing models such as GR4J or reproduce accurately observed data from existing catchments. Consequently, one should not be surprised by the unusual formulation of certain equations in Table 1.

Table 1: Reservoir equations used for testing QuaSoARe

Name	**Description**	**Fluxes**	**Range of scaling factor range $\boldsymbol{\theta}$**
CR	Cubic and bi-cubic routing models Hourly time step	Inflow: $f_1(S) = Q_{inflow}$ Outflow: $f_2(S) = -Q_{ref}\ (S/\theta)^3$	$[1\,800\ Q_{ref},\ 18\,000\ Q_{ref}]$
BCR	Bi-cubic routing models Hourly time step	Inflow: $f_1(S) = Q_{inflow}$ Outflow: $f_2(S) = -Q_{ref}\ (S/\theta)^6$	$[1\,800\ Q_{ref},\ 18\,000\ Q_{ref}]$
GR	GR4J production store Daily time step	Infiltrated rainfall: $f_1(S) = P[1-(S/\theta)^2]$ Actual evapotranspiration: $f_2(S) = -E\ (S/\theta)(2-S/\theta)$ Percolation: $f_3(S) = -\frac{2.25^4}{4}\theta\ (S/\theta)^5$	$[100, 1\,000]$
GRM	Modified GR4J production store including groundwater recharge Daily time step	Infiltrated rainfall: $f_1(S) = P\ [1-(S/\theta)^3(10-15\,S/\theta+6(S/\theta)^2)]$ Actual evapotranspiration: $f_2(S) = -E\ \left[16\left(S/\theta-\frac{1}{2}\right)^5+\frac{1}{2}\right]$ Percolation: $f_3(S) = -\frac{2.25^4}{4}\theta\ (S/\theta)^7$ Groundwater recharge: $f_4(S) = -0.1\ S/\theta/(1+10\ S/\theta)$	$[100, 1\,000]$

5.2 Performance evaluation

The solution of the four reservoir equations is obtained with QuaSoARe using 5, 50, and 500 interpolation nodes. These three configurations lead to increasing accuracy of the interpolation of flux functions, which is expected to translate into higher solution accuracy.

QuaSoARe is compared with two discrete numerical schemes: the Radau IIA implicit Runge–Kutta method of order 5 (Hairer and Wanner, 1996), referred to as "Radau", and the explicit Runge–Kutta method of order 5(4) (Dormand and Prince, 1980), denoted "RK45". The Radau method was chosen because it is implicit (hence able to handle stiff equations) and of high order, both characteristics leading us to qualify its outputs as the reference for comparison with other methods. The RK45 method was selected because it is the de facto explicit ODE solver that is widely recognised for its combination of speed and accuracy (Shampine, 2020). Both methods are run using the SciPy package implementation where these two algorithms are coded in Python (Virtanen et al., 2020).

Three performance criteria are used to compare the performance of these integration methods. The first criterion measures the maximum absolute error between the fluxes computed from one of the methods above and the Radau method:

$$E^m = max\left\{ \left|\hat{F}_i^{\ m}(k) - \hat{F}_i^{radau}(k)\right|, k = 1, \dots, p \quad i = 1, \dots, n\right\} \qquad \textbf{Eq. 11}$$

where $\hat{F}_i^m(k)$ is the i^{th} flux computed with method m for time step k. E^m is measured in the unit of the reservoir fluxes, hence $m^3 s^{-1}$ for CR and BCR, and $mm\ day^{-1}$ for the two remaining equations.

The second criterion is the maximum mass balance error between the fluxes computed with one of the methods and Radau:

$$B^m = \max\left\{\left|\frac{\sum_{k=1}^{p} \hat{F}_i^{\ m}(k) - \hat{F}_i^{radau}(k)}{\sum_{k=1}^{p} \hat{F}_i^{radau}(k)}\right|, i = 1, \dots, n\right\} \times 100 \qquad \textbf{Eq. 12}$$

B^m is dimensionless and reported in percentage. Finally, the third criterion compares the runtime of a method against the Radau runtime:

$$R^m = \frac{T^m}{T^{radau}} \times 100 \qquad \textbf{Eq. 13}$$

The runtime was measured on a laptop computer using a quad-core processor with a clock speed of 3GHz. Note that the runtime of QuaSoARe is assessed using a version of the code written in pure Python so that it can be compared with Radau and RK45 methods. A faster version of the code written in C within the same package is recommended for application purposes (Lerat, 2025). Measuring runtime is dependent on the machine used and may thus lack generality. An alternative metric to measure computational speed was tested (not reported here) where runtime is assessed by the number of flux function evaluations following Clark and Kavetski (2010). This metric provided similar results to R^m but is more challenging to interpret because QuaSoARe and the two discrete methods do not evaluate the same functions (analytical solutions for QuaSoARe and original flux functions for RK45 and Radau). Consequently, the simple runtime metric R^m is preferred here.

Table 2: Characteristic of the case study catchments

Name	Outlet station	Upstream station	Hydraulic distance upstream-outlet [km]	2022 Rainfall [mm y^{-1}]	2022 PET* [mm y^{-1}]	2022 Outlet peak [m^3 s^{-1}]
Casino	Richmond River at Casino Site ID 203004, 1790 km^2	Eden Creek at Doubtful Site ID 203034, 555 km^2	32	1941	1455	2077
Wiangaree	Richmond River at Wiangaree Site ID 203005, 702 km^2	Richmond River at Lavelles Road Site ID 203056, 337 km^2	16	1908	1430	1486
Eltham	Wilsons River at Eltham Site ID 203014, 223 km^2	Byron Creek at Binna Burra Site ID 203012, 36 km^2	20	3291	1472	573
Ewing Bridge	Coopers Creek at Ewing Bridge Site ID 203024, 148 km^2	Coopers Creek at Repentance Site ID 203002, 52 km^2	22	3401	1440	770
Fairmeadow	Coopers Creek at Fairmeadow Site ID 203060, 177 km^2	Coopers Creek at Repentance Site ID 203002, 57 km^2	26	3360	1441	365
Kyogle	Richmond River at Kyogle Site ID 203900, 899 km^2	Richmond River at Wiangaree Site ID 203005, 710 km^2	19	1992	1435	1358

*Potential evapotranspiration

5.3 Case study area

QuaSoARe performance is evaluated for six sites in the Richmond River catchment in eastern Australia, close to the city of Lismore. The catchments are presented in Table 1, and their location is shown in Figure 3. This area was chosen because it experienced a devastating flood in February 2022, which prompted an in-depth analysis of the event (Lerat et al., 2022). In addition, the maximum rainfall totals observed during this flood exceeded 700 mm in 24 hours. Such extreme values constitute a challenging test for ODE solvers, as pointed out by La Follette et al. (2021). For each catchment, the outlet station and one station located upstream of the outlet, as shown in Figure 3, are selected. The hourly streamflow data from the 1st February 2022 to 10th April 2022 at the upstream station are used to run the two routing models (CR and BCR). The daily catchment average climate data from the 1st January 2010 to 31st December 2022 are used to run the two hydrological models (GR and GRM).

As highlighted in Section 5.2, our aim is not to simulate hydrological processes accurately in these catchments but only to obtain realistic forcing data for our numerical experiments.

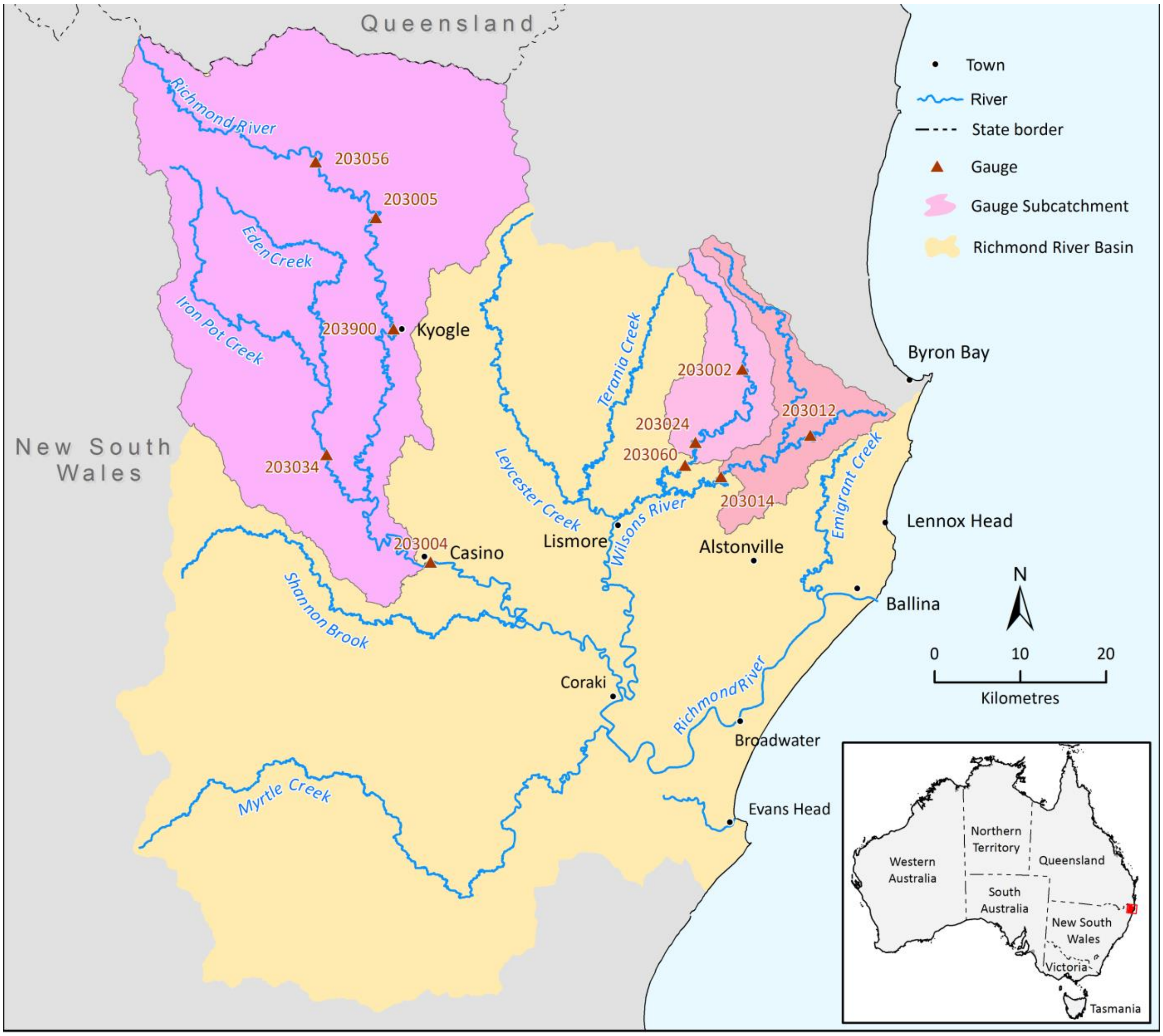


Figure 3: Location of the test catchments

6 Results

6.1 Simulation of the GR4J production store

As an example of the application of QuaSoARe, this section details the results obtained with the GR4J production store (GR) presented in Table 1 for the Coopers Creek at Ewin Bridge catchment. The GR simulation is run with a storage scaling factor of 500 mm, a typical value for this parameter in Australia. Figure 4 shows the interpolation performed by QuaSoARe to approximate the three fluxes of the GR model using 10 nodes. The plots show the storage level S divided by the scaling factor θ on the x-axis and the instantaneous flux on the y-axis. The values for P and E required to compute the flux functions, as indicated in Table 1, are set to 4.2 and 3.1 $\mathrm{mm\ day^{-1}}$ corresponding to the mean daily rainfall and potential evapotranspiration over the simulation period, respectively. The interpolation error is shown as a thin black line on the same plot using the a secondary y-axis of a different scale. In the three plots, the QuaSoARe flux appears indistinguishable from the "true" flux, which is visually satisfying but insufficient to guarantee an accurate solution, as already seen in Figure 1. The interpolation error for the first two fluxes is smaller in magnitude than 2×10^{-15} $\mathrm{mm\ day^{-1}}$, which is comparable to our machine precision. This is expected as the first two GR flux functions are quadratic functions that can be interpolated exactly by QuaSoARe. Note that using ten nodes is superfluous in this case as two nodes would suffice. The third flux is a power function with an exponent of 5, which cannot be interpolated exactly. Consequently, the interpolation error is larger, reaching a magnitude up to 2.5×10^{-3} $\mathrm{mm\ day^{-1}}$. This error remains small and shows an oscillating behaviour characteristic of polynomial interpolation using nodes with constant spacing. A potential improvement on this point is discussed in Section 7.

Figure 5 shows the daily storage levels and fluxes from the GR reservoir for the year 2022 using the Radau method (orange lines), considered the "truth", and the QuaSoARe method (blue lines) configured with ten interpolation nodes. In this figure, the Radau and QuaSoARe simulations are visually indistinguishable, which suggests that the interpolation errors shown in Figure 4 remain small enough to have no lasting impact. The simulation errors are shown as thin black lines using a secondary y-axis. These errors are lower in magnitude than 5×10^{-3} $\mathrm{mm\ day^{-1}}$, which is negligible compared to typical errors of climate input data. Observation errors are rarely below 10^{-2} $\mathrm{mm\ day^{-1}}$ in a research catchment and probably much higher in our study catchment (e.g. Chubb et al. (2016) report root mean squared errors of gridded daily rainfall data above 4.5 $\mathrm{mm\ day^{-1}}$). This result confirms that, despite the low number of nodes used, QuaSoARe can simulate the dynamic of the GR store and all its fluxes accurately. All flux errors exhibit oscillating patterns already noted in the interpolation error shown in Figure 4.c, which are likely due to a combination of oscillating interpolation errors and the alternation of wet and dry days leading to the sudden variations in actual evapotranspiration visible in Figure 5.c. This suggests that quantifying simulation errors in a highly non-linear reservoir equation is not straightforward and depends on both the quality of the numerical solution and the statistical characteristics of input forcings. More generally, the GR flux functions constitute a challenging case study because they combine the slow dynamic of the storage visible in Figure 5.a and rapid changes of actual ET seen in Figure 5.c. QuaSoARe appears to resolve both with a high accuracy.

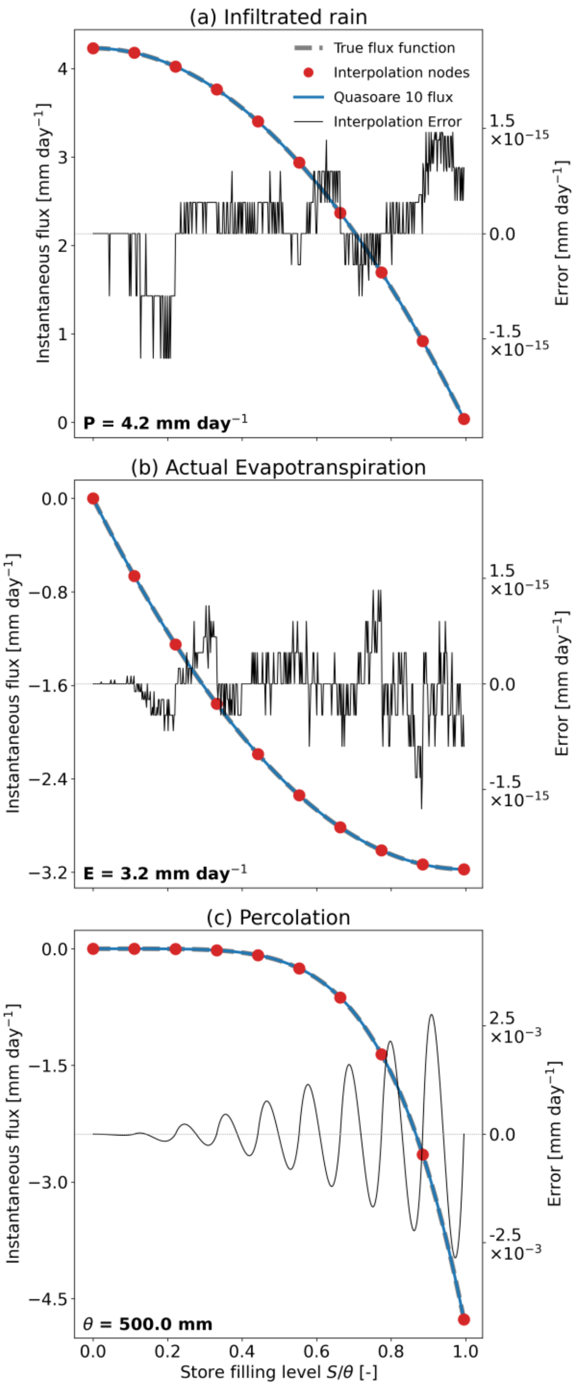
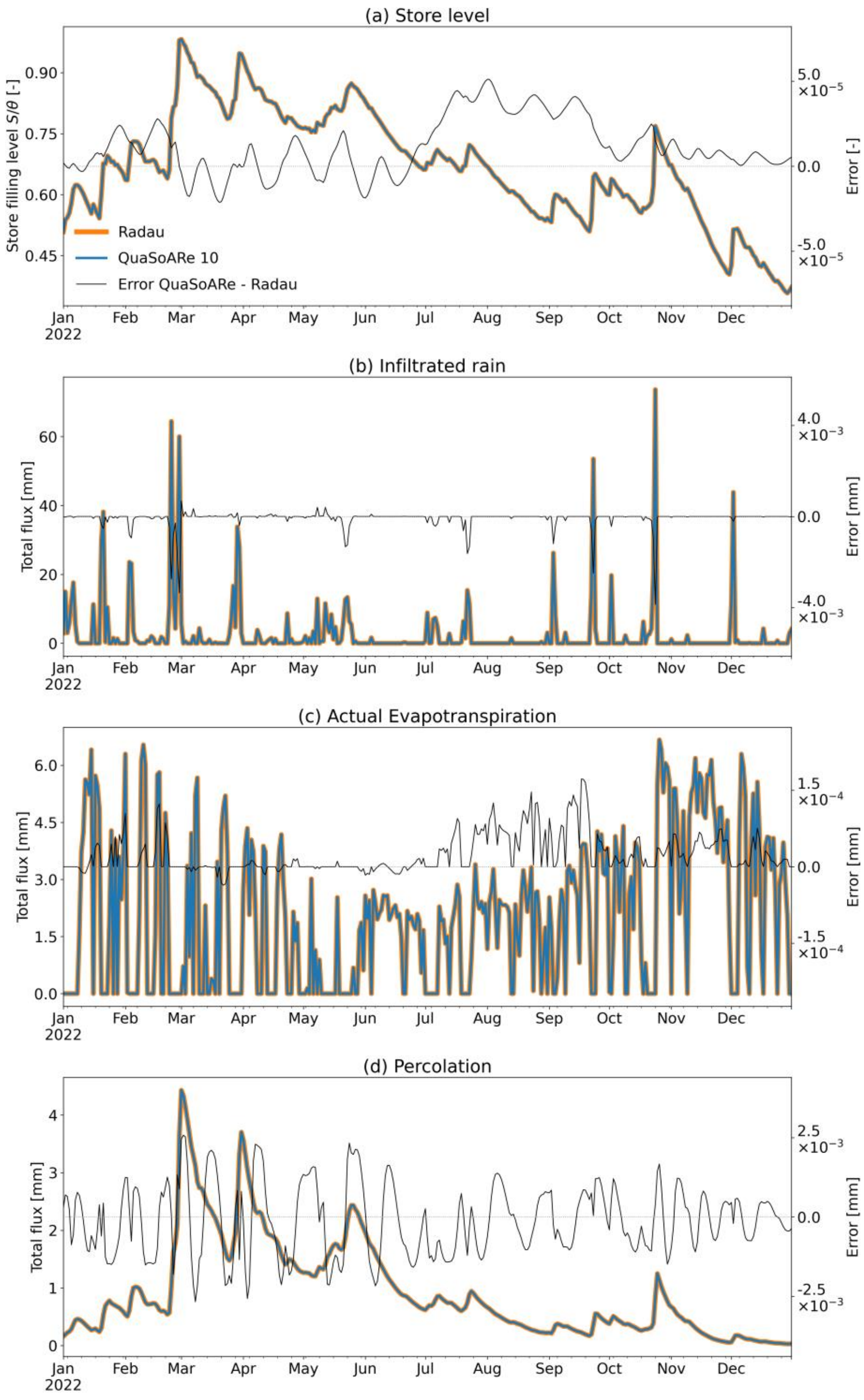


Figure 4: True and approximated flux functions of the GR reservoir equation with a scaling factor θ set to 500 mm. P and E variables are set to 4.2 and 3.1 mm, respectively, and QuaSoARe interpolation uses 10 nodes.



415 **Figure 5: Radau and QuaSoARe storage level (plot a) and three fluxes (plots b to c) for the GR reservoir with a scaling factor θ set to 500 mm and using data from the Coopers Creek at Ewing Bridge catchment. QuaSoARe is configured with 10 interpolation nodes.**

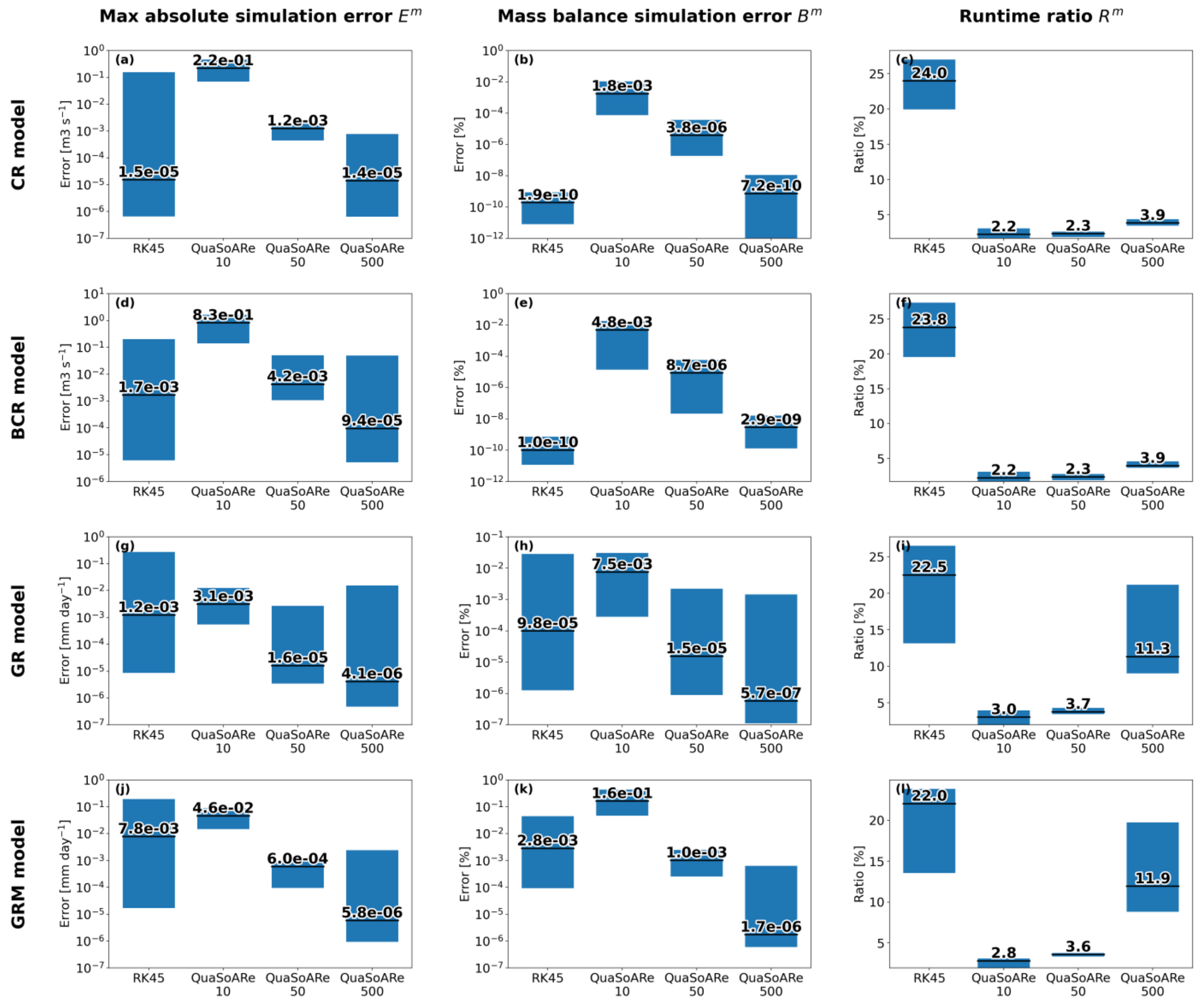


Figure 6: Comparison of performance metrics between the RK45 method and QuaSoARe using 10, 50 and 500 interpolation nodes. The Radau method is used as a reference simulation against which performance metrics are computed. Numbers in black report the median value of the performance metric while the blue boxes show their minimum and maximum range.

6.2 Performance assessment

Expanding the analysis of the previous section, Figure 6 presents statistics of the three performance metrics introduced in Section 5.2. The metrics are computed for the four models (CR, BCR, GR and GRM, see Table 1); the six study catchments (see Table 2); 10 values of the storage capacity parameter θ for each model (see Table 1); and five ODE solvers, including the Radau and RK45 methods and three configurations of QuaSoARe using 10, 50 and 500 nodes. Overall, 60 simulations are produced for each model and ODE solver combination. The Radau method is considered the reference against which the error of other methods is computed. The numbers shown in each plot of Figure 6 report the median value of the metrics over the 60 corresponding simulations, while the blue boxes show their minimum to maximum range.

All the plots in Figure 6 reveal that QuaSoARe simulations are extremely close to the Radau outputs if QuaSoARe is configured with a high number of nodes (500). The median absolute simulation error of QuaSoARe (E^m) varies between 4.4×10^{-6} mm day^{-1} for the GR model (Figure 6.g) to $9.4 \times 10^{-5} \mathrm{m}^3\mathrm{s}^{-1}$ for the BCR model. Both values are several orders of magnitude lower than observation errors, which means that simulations from QuaSoARe and Radau can be considered interchangeable from a modeller's perspective. Importantly, these results also hold for the mass balance metric B^m where the average difference between the flux totals from QuaSoARe and Radau remains lower than 2×10^{-6} % (Figure 6.k). At the same time, the QuaSoARe runtime is only a fraction of the Radau runtime with median values varying from 3.8% for the CR and BCR models (Figure 6.c and f) to 11.8% for GRM (Figure 6.l). Finally, QuaSoARe performance is better than RK45 for most models and performance metrics. For example, the maximum absolute error of QuaSoARe is several orders of magnitude lower than RK45 except for the CR model, where the error of both methods is close to 1.5×10^{-5} $\mathrm{m}^3\mathrm{s}^{-1}$ (Figure 6.a).

When fewer nodes are used, the errors of QuaSoARe increase and worsen the performance metrics. For example, the median maximum error for the GR model, shown in Figure 6.g, increases from 4.1×10^{-6} $\mathrm{m}^3\mathrm{s}^{-1}$ when using 500 nodes to 3.1×10^{-3} $\mathrm{m}^3\mathrm{s}^{-1}$ with 10 nodes. This degradation is expected as the interpolation of the reservoir functions worsens with fewer nodes. Nonetheless, the errors remain insignificant compared to observation errors, which suggests that a low number of nodes remains an attractive configuration when applying QuaSoARe to hydrological models. The runtime efficacy of QuaSoARe when using ten nodes is noticeable with values remaining lower than 3% of the Radau runtime for the four models. We highlight that the results presented here correspond to challenging test cases with highly non-linear flux functions. For example, the GRM reservoir includes functions with polynomials of order up to seven. In addition, the selected catchments exhibit a particularly challenging hydro-climate regime to simulate, with maximum rainfall intensity exceeding several hundred millimetres per day.

7 Discussion

The QuaSoARe method presented in this paper is a valuable alternative to existing numerical schemes for solving the scalar reservoir equation for three reasons. First, the algorithm is simple to understand and code, relying essentially on the interpolation of the reservoir functions by piecewise quadratic polynomials. Consequently, we believe it is straightforward to integrate into an existing modelling platform and help strengthen the numerical solution of hydrological models without adding a significant burden in code maintenance. Second, the method is much faster than standard alternatives such as the RK45 (explicit) and Radau (implicit) schemes tested in this paper. QuaSoARe reduces the runtime by a factor of 20 to 50 compared to Radau, depending on the model. This point further reinforces the value of the method for hydrological modelling tasks requiring repeated model evaluation, such as automated calibration or data assimilation. Finally, the configuration of

QuaSoARe can be modified simply by changing the number of interpolation nodes to vary between highly accurate but slow simulations (say with 500 nodes) to lower accuracy and fast runtime (say with 10 nodes). As a result, the modeller remains in control of the algorithm via a single and easily interpretable configuration parameter.

The QuaSoARe performance reported in this paper is satisfactory, but one may wonder if there could be ways to improve it further without changing the algorithm's core. The first point that could be improved is the quality of the interpolation. Our method relies on piecewise quadratic polynomials, which could be replaced by more flexible functions. However, the requirements of QuaSoARe in terms of interpolation functions are stringent because they should be at the same time stable by linear combination (to compute all fluxes) and leading to an analytical solution of the approximate reservoir equation. A powerful alternative to quadratic polynomials explored in early versions of this work is a combination of exponential functions. This choice was later abandoned because it was prone to numerical overflow and did not provide significant performance improvements. Nonetheless, we believe that other functions could meet the QuaSoARe requirements and help improve its performance in difficult situations such as when the flux functions are rational fractions, which are notoriously tough to interpolate with polynomials (Berrut and Trefethen, 2004).

Related to this point, one may wonder about the performance of a linear interpolation in comparison to its quadratic counterpart presented in this paper. Tests not shown here revealed that the linear interpolation worsens the maximum absolute error by an approximate factor of ten with a similar runtime. Interestingly, the difference between the two approaches seems to persist even when using a large number of nodes, hence when the interpolation of both linear and quadratic is expected to be close. Overall, we recommend the quadratic interpolation but kept the linear approach as a degraded functionality in our code (Lerat, 2025).

Several ideas could be explored to improve QuaSoARe further. In this paper, the interpolation nodes are placed at equal intervals between the two extremes α_1 and α_m. This could be easily modified to find the nodes providing the lowest interpolation errors. This might reduce the oscillations of the interpolation errors visible in Figure 4.c. However, this would also increase the algorithm's complexity, and we favoured simplicity in this first version of QuaSoARe. Finally, we flag that our method could be combined with a discrete method in a way similar to the exponential integrator approach of Pope (1963). This will likely improve performance further but also require using a discrete method, which we discarded in our introduction. However, this could be an interesting avenue for a modeller familiar with discrete methods.

Beyond potential improvements of QuaSoARe, we highlight that the flux functions defined in an empirical modelling context, like most rainfall-runoff models, remain arbitrary and are not derived from physically based equations. In this case, one could argue that piecewise quadratic functions could themselves constitute valid flux functions. This approach would have two merits: first, the instantaneous model equations could be characterised by a small set of coefficients (the interpolation coefficients), hence becoming fully parametric and amenable to optimisation or data assimilation; and second, the model could be solved exactly with QuaSoARe. As a result, models formulated in this way could become powerful components of flexible modelling environments. This avenue is currently being explored to extend the work of Lerat et al. (2024) in which improvement of model structure is obtained by updating state equations via data assimilation methods.

8 Conclusion

This paper presents a simple numerical algorithm called QuaSoARe to solve the instantaneous reservoir equation and provide a basis to build hydrological models based on this equation and generate fast and accurate solutions. The method was tested on a range of highly non-linear models that are representative of hydrological models in use. Its performance suggests that the method matches the accuracy of a high-order implicit discrete method while requiring a fraction of its runtime. Yet, the method algorithm is simple and can be described with a short pseudo-code, which is included in our paper.
The method is limited by its applicability to scalar equations and by the quality of the quadratic interpolation underlying its analytical solution. To model a series of reservoirs, we suggest implementing sub-time step integration. Higher accuracy in the interpolation of reservoir functions can always be achieved by increasing the number of interpolation nodes. Our results showed that using 10 to 50 nodes leads to an accuracy level that is an order of magnitude smaller than the typical error in hydro-climate observation data.

Competing interests

The contact author declares no competing interests.

Acknowledgements

This study was funded by Rous County Council as part of the Richmond and Wilsons Rivers Flood Risk Mitigation Project. The contribution from Tewfik Sari and Vazken Andréassian from INRAE, who pointed out references to the TPLA method and commented on early versions of this manuscript, is greatly acknowledged. Comments provided by Justin Hughes, CSIRO, David Robertson, CSIRO, and Charles Perrin, INRAE, as well as the site map prepared by Steve Marvanek, CSIRO, and editing by Margie Beilharz are also acknowledged. Finally, we thank the two anonymous reviewers who raised important questions and helped improve the quality of the manuscript significantly.

Code and data availability

The QuaSoARe method is released as part of a Python package written by Lerat (2025). The package contains two implementations of the method written in the Python and C languages. The C version is recommended due to its significantly faster runtime. All scripts and hydro-climate data used to generate the results in this paper are included in the code repository.

References

American Meteorological Society: Glossary of Meteorology: Diagnostic Equation, 2024.

Bentura, P. L. F. and Michel, C.: Flood routing in a wide channel with a quadratic lag-and-route method, Hydrological Sciences Journal, 42, 169–189, https://doi.org/10.1080/02626669709492018, 1997.

Bergstrom, S. and Forsman, A.: Development of a conceptual deterministic rainfall-runoff model, Hydrology Research, 4, 147–170, https://doi.org/10.2166/nh.1973.0012, 1973.

Berrut, J.-P. and Trefethen, L. N.: Barycentric Lagrange Interpolation, SIAM Rev., 46, 501–517, https://doi.org/10.1137/S0036144502417715, 2004.

Bond, B. N. and Daniel, L.: Stable Reduced Models for Nonlinear Descriptor Systems Through Piecewise-Linear Approximation and Projection, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28, 1467–1480, https://doi.org/10.1109/TCAD.2009.2030596, 2009.

Burnash, R. J. C. and Ferral, R. L.: A systems approach to real-time runoff analysis with a deterministic rainfall-runoff model, United States, National Weather Service., Western Region.;United States, National Oceanic and Atmospheric Administration., 1981.

Butcher, J. C.: Numerical methods for ordinary differential equations, Wiley, Chichester, 2003.

Chubb, T. H., Manton, M. J., Siems, S. T., and Peace, A. D.: Evaluation of the AWAP daily precipitation spatial analysis with an independent gauge network in the Snowy Mountains, JSHESS, 66, 55–67, https://doi.org/10.1071/es16006, 2016.

Clark, M. P. and Kavetski, D.: Ancient numerical daemons of conceptual hydrological modeling: 1. Fidelity and efficiency of time stepping schemes, Water Resources Research, 46, https://doi.org/10.1029/2009WR008894, 2010.

Clark, M. P., Slater, A. G., Rupp, D. E., Woods, R. A., Vrugt, J. A., Gupta, H. V., Wagener, T., and Hay, L. E.: Framework for Understanding Structural Errors (FUSE): A modular framework to diagnose differences between hydrological models, Water Resources Research, 44, 2008.

Croke, B. F. W. and Jakeman, A. J.: A catchment moisture deficit module for the IHACRES rainfall-runoff model, Environmental Modelling & Software, 19, 1–5, https://doi.org/10.1016/j.envsoft.2003.09.001, 2004.

Dormand, J. R. and Prince, P. J.: A family of embedded Runge-Kutta formulae, Journal of Computational and Applied Mathematics, 6, 19–26, https://doi.org/10.1016/0771-050X(80)90013-3, 1980.

Fan, P. and Li, J. C.: Diffusive wave solutions for open channel flows with uniform and concentrated lateral inflow, Advances in Water Resources, 29, 1000–1019, https://doi.org/10.1016/j.advwatres.2005.08.008, 2006.

Fiorentini, M. and Orlandini, S.: Robust numerical solution of the reservoir routing equation, Advances in Water Resources, 59, 123–132, https://doi.org/10.1016/j.advwatres.2013.05.013, 2013.

Gill, M. A.: Exact Solution of Gradually Varied Flow, Journal of Hydraulic Engineering, 1976.

Hairer, E. and Wanner, G.: Solving Ordinary Differential Equations II, Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-05221-7, 1996.

Hairer, E., Nørsett, S. P., and Wanner, G.: Solving ordinary differential equations I: nonstiff problems, 2nd rev. ed., Springer, Heidelberg ; London, 528 pp., 2009.

Hapuarachchi, H. A. P., Bari, M. A., Kabir, A., Hasan, M. M., Woldemeskel, F. M., Gamage, N., Sunter, P. D., Zhang, X. S., Robertson, D. E., Bennett, J. C., and Feikema, P. M.: Development of a national 7-day ensemble streamflow forecasting service for Australia, Hydrology and Earth System Sciences, 26, 4801–4821, https://doi.org/10.5194/hess-26-4801-2022, 2022.

Hartman, P.: Ordinary Differential Equations, Second., Society for Industrial and Applied Mathematics, https://doi.org/10.1137/1.9780898719222, 2002.

Hayami, S.: On the propagation of flood waves, Bulletins-Disaster Prevention Research Institute, Kyoto University, 1, 1–16, 1951.

Hochbruck, M. and Ostermann, A.: Explicit Exponential Runge-Kutta Methods for Semilinear Parabolic Problems, SIAM Journal on Numerical Analysis, 43, 1069–1090, 2006.

Johansson, M.: Piecewise Linear Control Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, https://doi.org/10.1007/3-540-36801-9, 2003.

Kalman, E. R.: Phase-plane analysis of automatic control systems with nonlinear gain elements, Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry, 73, 383–390, https://doi.org/10.1109/TAI.1955.6367086, 1955.

Kalra, S. and Nabi, M.: Automated scheme for linearisation points selection in TPWL method applied to non-linear circuits, The Journal of Engineering, 2020, 41–49, https://doi.org/10.1049/joe.2019.0876, 2020.

Kavetski, D. and Clark, M. P.: Ancient numerical daemons of conceptual hydrological modeling: 2. Impact of time stepping schemes on model analysis and prediction, Water Resources Research, 46, https://doi.org/10.1029/2009WR008896, 2010.

Kavetski, D. and Clark, M. P.: Numerical troubles in conceptual hydrology: Approximations, absurdities and impact on hypothesis testing, Hydrological Processes, 25, 661–670, https://doi.org/10.1002/hyp.7899, 2011.

Kavetski, D., Kuczera, G., and Franks, S. W.: Calibration of conceptual hydrological models revisited: 1. Overcoming numerical artefacts, Journal of Hydrology, 320, 173–186, https://doi.org/10.1016/j.jhydrol.2005.07.012, 2006.

Knoben, W. J. M., Freer, J. E., Fowler, K. J. A., Peel, M. C., and Woods, R. A.: Modular Assessment of Rainfall–Runoff Models Toolbox (MARRMoT) v1.2: an open-source, extendable framework providing implementations of 46 conceptual hydrologic models as continuous state-space formulations, Geoscientific Model Development, 12, 2463–2480, https://doi.org/10.5194/gmd-12-2463-2019, 2019.

La Follette, P. T., Teuling, A. J., Addor, N., Clark, M., Jansen, K., and Melsen, L. A.: Numerical daemons of hydrological models are summoned by extreme precipitation, Hydrology and Earth System Sciences, 25, 5425–5446, https://doi.org/10.5194/hess-25-5425-2021, 2021.

LaSalle, J.: Some Extensions of Liapunov's Second Method, IRE Transactions on Circuit Theory, 7, 520–527, https://doi.org/10.1109/TCT.1960.1086720, 1960.

Lerat, J.: pyquasoare: a python and C package for the QuaSoARe method, Zenodo [code], https://doi.org/10.5281/zenodo.13928253, 2025.

Lerat, J., Vaze, J., Marvanek, S., and Ticehurst, C.: Characterisation of the 2022 floods in the Northern Rivers region, CSIRO, Australia, 2022.

Lerat, J., Chiew, F., Robertson, D., Andréassian, V., and Zheng, H.: Data Assimilation Informed Model Structure Improvement (DAISI) for Robust Prediction Under Climate Change: Application to 201 Catchments in Southeastern Australia, Water Resources Research, 60, e2023WR036595, https://doi.org/10.1029/2023WR036595, 2024.

Litrico, X., Pomet, J.-B., and Guinot, V.: Simplified nonlinear modeling of river flow routing, Advances in Water Resources, 33, 1015–1023, https://doi.org/10.1016/j.advwatres.2010.06.004, 2010.

Meyer, O. H.: Flood-routing on the Sacramento River, Eos, Transactions American Geophysical Union, 22, 118–124, https://doi.org/10.1029/TR022i001p00118, 1941.

Michel, C., Perrin, C., and Andreássian, V.: The exponential store: a correct formulation for rainfall—runoff modelling, Hydrological Sciences Journal, https://doi.org/10.1623/hysj.48.1.109.43484, 2003.

Mironchenko, A.: Input-to-State Stability, in: Input-to-State Stability: Theory and Applications, edited by: Mironchenko, A., Springer International Publishing, Cham, 41–115, https://doi.org/10.1007/978-3-031-14674-9_2, 2023.

Munier, S., Litrico, X., Belaud, G., and Malaterre, P.-O.: Distributed approximation of open-channel flow routing accounting for backwater effects, Advances in Water Resources, 31, 1590–1602, https://doi.org/10.1016/j.advwatres.2008.07.007, 2008.

Perrin, C., Michel, C., and Andréassian, V.: Improvement of a parsimonious model for streamflow simulation, Journal of Hydrology, 279, 275–289, https://doi.org/10.1016/S0022-1694(03)00225-7, 2003.

Pope, D. A.: An exponential method of numerical integration of ordinary differential equations, Commun. ACM, 6, 491–493, https://doi.org/10.1145/366707.367592, 1963.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P.: Numerical Recipes 3rd Edition: The Art of Scientific Computing, 3rd edition., Cambridge University Press, Cambridge, UK ; New York, 1256 pp., 2007.

Rewienski, M. and White, J.: A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 22, 155–170, https://doi.org/10.1109/TCAD.2002.806601, 2003.

Santos, L., Thirel, G., and Perrin, C.: Continuous state-space representation of a bucket-type rainfall-runoff model: a case study with the GR4 model using state-space GR4 (version 1.0), Geoscientific Model Development, 11, 1591–1605, https://doi.org/10.5194/gmd-11-1591-2018, 2018.

Shampine, L. F.: Numerical Solution of Ordinary Differential Equations, Routledge, New York, 632 pp., https://doi.org/10.1201/9780203745328, 2020.

Sontag, E. D.: Smooth stabilization implies coprime factorization, IEEE Transactions on Automatic Control, 34, 435–443, https://doi.org/10.1109/9.28018, 1989.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods, 17, 261–272, https://doi.org/10.1038/s41592-019-0686-2, 2020.

Yevdjevich, V. M.: Analytical integration of the differential equation for water storage, Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics, 63B, 43, https://doi.org/10.6028/jres.063B.007, 1959.

Appendix A: Interpolation coefficients

For each function f_i and each interval $[\alpha_j, \alpha_{j+1}]$, we seek values of the coefficients $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$ so that the approximated function f_i^* matches f_i as best as possible. This is achieved by matching f_i at the two points α_j and α_{j+1} and at their midpoint. Assuming that $\alpha_j < \alpha_{j+1}$, quadratic interpolation leads to:

$$a_{i,j} = \frac{2f_0 + 2f_1 - 4f_m}{\left(\alpha_{j+1} - \alpha_j\right)^2} \qquad \textbf{Eq. 14}$$

$$b_{i,j} = -2\alpha_j\, a_{i,j} + \frac{4f_m - 3f_0 - f_1}{\alpha_{j+1} - \alpha_j} \qquad \textbf{Eq. 15}$$

$$c_{i,j} = \alpha_j^2\, a_{i,j} - \alpha_j\, \frac{4f_m - 3f_0 - f_1}{\alpha_{j+1} - \alpha_j} + f_0 \qquad \textbf{Eq. 16}$$

Where:

$$f_0 = f_i(\alpha_j), \qquad f_1 = f_i(\alpha_{j+1}), \qquad f_m = f_i\left(\frac{\alpha_j + \alpha_{j+1}}{2}\right) \qquad \textbf{Eq. 17}$$

This solution is straightforward but can lead to a function f_i^* that is non-monotonous, for example, when interpolating a function f_i that presents sharp transitions or asymptotes. This situation can create an issue related to the steady-state solution of Eq. 1 as discussed in Section 4. To avoid this undesirable effect, we replace the value of f_m used in Eq. 14 to Eq. 16 by a value f_m^* defined as follows:

$$f_m^* = \max\left(f_m^-, \min(f_m^+, f_m)\right) \qquad \textbf{Eq. 18}$$

Where the two bounds f_m^- and f_m^+ are given by:

$$f_m^- = \min\left(\frac{3f_0 + f_1}{4}, \frac{f_0 + 3f_1}{4}\right) \qquad \textbf{Eq. 19}$$

$$f_m^+ = \max\left(\frac{3f_0 + f_1}{4}, \frac{f_0 + 3f_1}{4}\right) \qquad \textbf{Eq. 20}$$

It can be verified that using f_m^* instead of f_m in Eq. 14 to Eq. 16 leads to a monotonous quadratic function f_i^*.

Appendix B: Analytical solution of the approximated reservoir equation

In this appendix, we solve the approximate reservoir equation (Eq. 9) over an interval $[t_0, t_1]$ such that $\forall t,\ S(t) \in [\alpha_j, \alpha_{j+1}]$ starting from the initial condition $S(t_0) = S_0$. This solution is obtained by separating the variables leading to the following integral equation:

$$\int_{S_0}^{S_1} \frac{dS}{A_j\, S^2 + B_j\, S + C_j} = t \qquad \text{Eq. 21}$$

640 Where $S_1 = S(t_1)$. Here we assume that either $A_j \neq 0$ or $B_j \neq 0$. If this is not the case, the interpolation function f_i^* is simply the constant C_j leading to:

$$S(t_1) = S_0 + C_j\, \tau \qquad \text{Eq. 22}$$

Where $\tau = t_1 - t_0$. If $A_j \neq 0$ or $B_j \neq 0$, the solutions of Eq. 21 can be obtained from any calculus textbook. They are presented succinctly in Table 3. In this table, the subscript "j" is dropped to simplify notations. Function ν mentioned in the second column of Table 3 is defined as the following primitive:

$$\nu\left(A_j, B_j, C_j, s\right) = \int \frac{dS}{A_j\, S^2 + B_j\, S + C_j} \qquad \text{Eq. 23}$$

645 This function is used in our algorithm to compute the times t_l and t_u mentioned in Cases 2 and 3 in Section 2.3. Finally, Table 3 uses the following expressions to simplify notations further:

$$\bar{S} = -\frac{B_j}{2A_j} \qquad \text{Eq. 24}$$

$$\Delta = B_j^2 - 4A_jC_j \quad \sigma_\Delta = sgn(\Delta) \quad q_\Delta = \frac{\sqrt{|\Delta|}}{2} \qquad \text{Eq. 25}$$

$$\omega_\Delta(x) = \begin{cases} \tan(x) & if\ \Delta < 0 \\ \tanh(x) & if\ \Delta > 0 \end{cases} \qquad \text{Eq. 26}$$

$$\eta_\Delta(x) = \begin{cases} \operatorname{atan}(x) & if\ \Delta < 0 \\ -\operatorname{atanh}(x) & if\ \Delta > 0\ and\ |x| < 1 \\ -\operatorname{atanh}(1/x) & if\ \Delta > 0\ and\ |x| > 1 \end{cases} \qquad \text{Eq. 27}$$

Where sgn is the sign function.

We now focus on finding an expression for the total of the i^{th} flux introduced in Eq. 2. Combining the solution of Eq. 21 given in the third column of Table 3, denoted $S(t)$, with Eq. 2 gives:

$$O_i^*(t_0, t_1) = \int_{t_0}^{t_1} f_i^*(S(t))\, dt = a_{i,j} \int_{t_0}^{t_1} S(t)^2\, dt + b_{i,j} \int_{t_0}^{t_1} S(t)\, dt + c_{i,j}\tau \qquad \text{Eq. 28}$$

650 Where $\tau = t_1 - t_0$. Note that, in the above equations, the integral boundaries used in Eq. 2 ($[0, \delta]$) are generalised to $[t_0, t_1]$ because fluxes are often computed on an smaller interval than $[0, \delta]$ in the algorithm described in Section 2.3. Solving Eq. 28 is equivalent to integrating $S(t)$ and $S(t)^2$ between t_0 and t_1. To compute these two integrals, we consider different cases for A_j and B_j starting with $A_j = B_j = 0$. In this case, we obtain from Eq. 22:

$$\int_{t_0}^{t_1} S(t)\, dt = S_0\tau + C_j \frac{\tau^2}{2} \qquad \text{Eq. 29}$$

$$\int_{t_0}^{t_1} S(t)^2\, dt = S_0^2\tau + S_0C_j\tau^2 + C_j^2 \frac{\tau^3}{3} \qquad \text{Eq. 30}$$

We now assume that $A_j = 0$ and $B_j \neq 0$. The two integrals can be obtained by manipulating and integrating Eq. 9 as follows:

$$\int_{t_0}^{t_1} \frac{dS}{dt}\, dt = B_j \int_{t_0}^{t_1} S(t)\, dt + C_j\tau \;\Rightarrow\; \int_{t_0}^{t_1} S(t)\, dt = \frac{1}{B_j}(S_1 - S_0 - C_j\tau) \qquad \text{Eq. 31}$$

$$\int_{t_1}^{t_2} S(t) \frac{dS}{dt}\, dt = B_j \int_{t_1}^{t_2} S(t)^2\, dt + C_j \int_{t_1}^{t_2} S(t) dt \qquad \text{Eq. 32}$$

$$\Rightarrow \int_{t_0}^{t_1} S(t)^2\, dt = \frac{1}{B_j}\left(\frac{S_1^2 - S_0^2}{2} - C_j \int_{t_0}^{t_1} S(t)\, dt\right)$$

With these two equations, integrating S is done first using Eq. 31. This integral is then used in Eq. 32 to compute the integral of S^2.

Finally, we now address the general case where $A_j \neq 0$. Following a similar approach than above, we rearrange and integrate Eq. 9 as follows:

$$\int_{t_0}^{t_1} \frac{dS}{dt}\, dt = A_j \int_{t_0}^{t_1} S(t)^2\, dt + B_j \int_{t_0}^{t_1} S(t)\, dt + C_j\tau \qquad \text{Eq. 33}$$

$$\Rightarrow \int_{t_0}^{t_1} S(t)^2\, dt = \frac{1}{A_j}\left(S_1 - S_0 - B_j \int_{t_0}^{t_1} S(t)\, dt - C_j\tau\right)$$

Consequently, the integration of S^2 can be deduced from the one of S. For the integration of S, we use the expression of $S(t)$ given in the third column of Table 3. If $\Delta = 0$ (see row before last in Table 3), we obtain:

$$\int_{t_0}^{t_1} S(t)\, dt = \bar{S}\,\tau - \frac{1}{a}\log[1 - a\,\tau\,(S_0 - \bar{S})] \qquad \text{Eq. 34}$$

A value $\Delta \neq 0$ leads to (see last row in Table 3):

$$\int_{t_0}^{t_1} S(t)\, dt = \bar{S}\,\tau + \frac{1}{2a}\log\big(1 - \sigma_\Delta\, \omega_\Delta^2(q_\Delta\tau)\big) - \frac{1}{a}\log\left(1 - a\frac{S_0 - \bar{S}}{q_\Delta}\, \omega_\Delta(q_\Delta\tau)\right) \qquad \text{Eq. 35}$$

Table 3: Solutions of approximate reservoir equation

Case	Primitive $\boldsymbol{v(A,B,C,s)}$	Solution $\boldsymbol{S(A,B,C,S_0,t)}$	Domain of validity of solution $\boldsymbol{S(A,B,C,S_0,t)}$ denoted $\boldsymbol{[0,T_m[}$
$A=0$ $B\neq 0$	$\frac{1}{B}\log\lvert B\,s+C\rvert$	$-\frac{C}{B}+\left(S_0+\frac{C}{B}\right)\exp(b\,t)$	$T_m=+\infty$
$A\neq 0$ $\Delta=0$	$-\frac{1}{A(s-\bar{S})}$	$\bar{S}+\frac{S_0-\bar{S}}{1-A\,t\,(S_0-\bar{S})}$	$T_m=+\infty$ if $A(S_0-\bar{S})\leq 0$ Otherwise $T_m=\frac{1}{A(S_0-\bar{S})}$
$A\neq 0$ $\Delta\neq 0$	$\frac{1}{q_\Delta}\eta_\Delta\left(A\frac{s-\bar{S}}{q_\Delta}\right)$	$\bar{S}+\frac{S_0-\bar{S}-\sigma_\Delta\frac{q_\Delta}{A}\omega_\Delta(q_\Delta t)}{1-A\frac{S_0-\bar{S}}{q_\Delta}\omega_\Delta(q_\Delta t)}$	If $\Delta<0$ $T_m=\frac{1}{q_\Delta}\left[\frac{\pi}{2}-\eta_\Delta\left(A\frac{S_0-\bar{S}}{q_\Delta}\right)\right]$ Otherwise, $T_m=\begin{cases}+\infty & if\ A(S_0-\bar{S})<q_\Delta\\ -\frac{1}{q_\Delta}\eta_\Delta\left(A\frac{S_0-\bar{S}}{q_\Delta}\right) & otherwise\end{cases}$