An implicit 1D Shallow Water Equation Solver using SciML *Computational Science NL Hackathon 2025*

Sven Westerbeek & Floris Buwalda

I. Introduction

At Deltares, we aim to make living in delta areas safe. A significant contribution to that goal is related to the development of numerical solvers for the rapid and accurate modelling of shallow water bodies (e.g. rivers, estuaries, lakes, flood plains and seas.) Successful simulations can be used for the visualization of transport of toxic matter or salinity and the consequences of for example heavy storms, tidal floods or upstream rainfall on the water levels. In turn, the results translate to requirements for prevention and safety measures such as dikes. Contemporary research at Deltares is focused on the development of an implicit solver for the shallow water equations and investigating how novel software tools can aid in reaching this goal.

II. The 1D Shallow Water Equations

Neglecting the effects of the wind shear stress, viscosity and the Coriolis term, the 1D shallow-water equations in integral form read as follows:

$$\int \frac{\partial \vec{U}}{\partial t} dV + \oint \vec{F} \cdot \vec{n} dS = \int \vec{S} dV \tag{1}$$

with:

$$\vec{U} = \begin{pmatrix} h \\ q \end{pmatrix} \tag{2}$$

$$\vec{F} = \begin{pmatrix} \underbrace{q}_{\text{Mass flux}} \\ \underbrace{\frac{q^2}{h}}_{\text{Convective flux}} \end{pmatrix} \tag{3}$$

$$\vec{S} = \begin{pmatrix} 0 \\ \underbrace{-gh\frac{\partial \zeta}{\partial x}}_{\text{Pressure gradient}} \quad \underbrace{-c_f \frac{q|q|}{h^2}}_{\text{Bed shear stress}} \end{pmatrix} \tag{4}$$

or, in differential form:

$$\frac{\partial h}{\partial t} + \frac{\partial q}{\partial x} = 0, \tag{5a}$$

$$\frac{\partial q}{\partial t} + \frac{\partial (q^2/h)}{\partial x} = -gh\frac{\partial \zeta}{\partial x} - c_f \frac{q|q|}{h^2}, \tag{5b}$$

where:

h	is the total water depth [m],
q	is the unit discharge in the x-direction [m s^{-1}],
ζ	is the free-surface level, $\zeta = h + z_b$ [m],
z_b	is the bed level [m],
g	is the gravitational acceleration [m s^{-2}], and
c_f	is a friction coefficient [-].

A thorough explanation and derivation of the equations can be found in [1]. To leverage the SciML ecosystem, the problem should be set up using a delta formulation, i.e.:

$$J \Delta U = b, \tag{6}$$

with J the Jacobian of the problem, b the residual function and ΔU the solution update. SciML can fully take over the left-hand side of this problem. We need only provide the spatial discretization present in b.

III. Objective

This challenge requires a 1D Shallow Water Equation Solver in the Julia programming language, leveraging the SciML ecosystem. Experience with Julia or SciML is **not** a prerequisite, but we do recommend having a look at Appendix A for Julia-related preparation material. During this hackathon, implement a solver for the 1D Shallow Water Equations.

As for the boundary conditions, we request 3 levels of functionality:

1) Periodic boundary conditions.
2) Dirichlet and Neumann boundary conditions (always a combination of a discharge description upstream and a water level downstream).
3) BONUS: Generating-Absorbing Boundary Conditions (GABC), see appendix B. Use it to prescribe a sinusoidal signal entering the domain.

You are free to choose any discretization scheme in space and time as long as it is second-order accurate in at least the interior domain (for comparison purposes). It is recommended to research this choice thoroughly. Perhaps you can even compare different methods and deliver your best solution.

Try run the following test case:

TABLE I: Input parameters for the simulation

Parameter	**Value / Expression**
Domain depth D	10.0
Flat bottom $z_{b,1}$	$-D$
Wavy bottom $z_{b,2}$	$-D + 0.4\sin\left(2\pi \frac{x}{x_{max}} \cdot \frac{nx-1}{nx} \cdot 5\right)$
Grid points nx	200
Domain x	$[0.0, 5.0]$
Initial time t_0	0.0 s
Final time t_{stop}	1.0 s
Initial discharge q_0	0.0
Initial height h_0	$0.1\exp\left(-100\left(\left(\frac{x}{x_{max}} - 0.5\right)x_{max}\right)^2\right) - z_b$

IV. Deliverables

The deliverables are as follows:

1) Implement a 1D Shallow water solver with varying boundary condition capabilities using SciML's nonlinear problem setup in the provided template. Your timeloop function should be able to run using only a parameters tuple as input. This tuple should contain the following: (; g, N, x, D, zb, $tstart$, $tstop$), where g (Float64) the gravitational acceleration, N (Int) the discretization parameter, x (Vector of Float64) the domain points where the bed level is described, D (Float64) the depth, zb (Vector of Float64) the bed levels at the point given in x as a function of the Depth D (see e.g. Table I) and lastly the initial and end time $tstart$ and $tstop$ in seconds (Float64).

V. Appendix A: Julia

If you are not familiar or comfortable yet with using Julia, it is recommended to have a look at this Matlab-Python-Julia cheat sheet: https://cheatsheets.quantecon.org/ to convert your syntax knowledge of Matlab and Python to Julia. There is also a tutorial that you can either use to prepare for the hackathon or as a reference: https://www.matecdev.com/posts/julia-tutorial-science-engineering.html. And, of course, having a look beforehand at the documentation of SciML is recommended.

VI. Appendix B: Generating-Absorbing boundary conditions

GABC are a specific set of equations that can be prescribed locally at the domain boundary that aim to allow for outgoing signals to leave the domain while still being able to prescribe incoming signals. At first glance, this might seem paradoxical; How can one both prescribe a value at a boundary while also allowing an outgoing signal pass through this boundary?

An analysis of the system of equations reveals that a diagonalization of the matrix describing the system is possible in 1 dimension. Sadly, this is not possible in 2 dimensions, only up to some order of the angle of incidence but that is out of the scope of this hackathon. The diagonalization of the system reveals that the problem can be described in terms of characteristics (with velocities given by the eigenvalues) that travel either left (i.e. outward at $x = 0$ and inward at $x = L$) or right (i.e. inward at $x = 0$ and outward at $x = L$) with a speed equal to the wave speed corrected by the velocity of the underlying flow. The equations are given in Table II

TABLE II: Generating-Absorbing Boundary Conditions (GABCs)

Boundary	**Wave Direction**	**Equation**
$x = 0$ (left)	Incoming	$\frac{1}{h}\left(\frac{\partial q}{\partial t} - \frac{q}{h}\frac{\partial h}{\partial t} + \sqrt{gh}\frac{\partial h}{\partial t}\right)$
	Outgoing	$\frac{1}{h}\left(\frac{\partial q}{\partial t} - \frac{q}{h}\frac{\partial h}{\partial t} - \sqrt{gh}\frac{\partial h}{\partial t}\right)$
$x = L$ (right)	Incoming	$\frac{1}{h}\left(\frac{\partial q}{\partial t} - \frac{q}{h}\frac{\partial h}{\partial t} - \sqrt{gh}\frac{\partial h}{\partial t}\right)$
	Outgoing	$\frac{1}{h}\left(\frac{\partial q}{\partial t} - \frac{q}{h}\frac{\partial h}{\partial t} + \sqrt{gh}\frac{\partial h}{\partial t}\right)$

A direct implementation of the equations in Table II as is will not work with SciML and requires some more manipulation. Can you make it work? Two hints are available upon request.

References

[1] Cornelis Boudewijn Vreugdenhil. *Numerical methods for shallow-water flow*. Vol. 13. Springer Science & Business Media, 2013.