PDEs and Numerical Methods : Lab session 6

Malik Hacini

Last updated: **December 14, 2025**

Contents

1: Introduction

In this lab, we solve the 1D Shallow Water Equations (SWE) using finite difference methods. The equations describe the evolution of the water height deviation h and velocity v:

$$h_t = -Hv_x \quad [1]$$

$$v_t = -gh_x \quad [2]$$

2: Spatial Discretization

We implemented the first-order spatial derivative using a second-order centered finite difference scheme. For a variable u on a grid with spacing Δx, the derivative at index i is approximated as:

$$\frac{\partial u}{\partial x}\Big|_i \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x} \quad [3]$$

The implementation in `Operators::diff1` reflects this:

```cpp
GridData<T> diff1(const GridData<T> &i_d) const {
    GridData<T> o;
    o.setup_like(i_d);

    // Note: inv_dx2 pre-calculates 1/(2*dx)
    T inv_dx2 = discConfig->inv_dx2;
    int N = i_d.size;

    for (int i = 0; i < N; i++) {
        // Handle periodic boundaries
        int im1 = (i == 0) ? N - 1 : i - 1;
        int ip1 = (i == N - 1) ? 0 : i + 1;

        // Centered difference: (u_{i+1} - u_{i-1}) / (2*dx)
        o.data[i] = (i_d.data[ip1] - i_d.data[im1]) * inv_dx2;
    }
    return o; }
```

3: Time Integration

The shallow water equations are given by:

$$h_t = -Hv_x \tag{4}$$

$$v_t = -gh_x \tag{5}$$

We implemented the time derivative computation in `TimeStepperBase::df_dt`:

```C++
void df_dt(const std::array<GridData<T_>,NArraySize_> &i_U,
    const Operators<T_> &i_ops, std::array<GridData<T_>,NArraySize_> &o_U) {
    T_ g = config.sim_g;
    T_ h_bar = std::abs(config.sim_bavg);
    // h_t = - h_bar * v_x
    o_U[0] = -h_bar * i_ops.diff1(i_U[1]);
    // v_t = - g * h_x
    o_U[1] = -g * i_ops.diff1(i_U[0]); }
```

Forward Euler (RK1)

The Forward Euler method is defined as:

$$U^{n+1} = U^n + \Delta t \cdot f(U^n) \tag{6}$$

We observed that this method is unconditionally unstable for the centered difference discretization of the wave equation. As shown in Figure 1, the solution rapidly develops high-frequency oscillations and explodes.

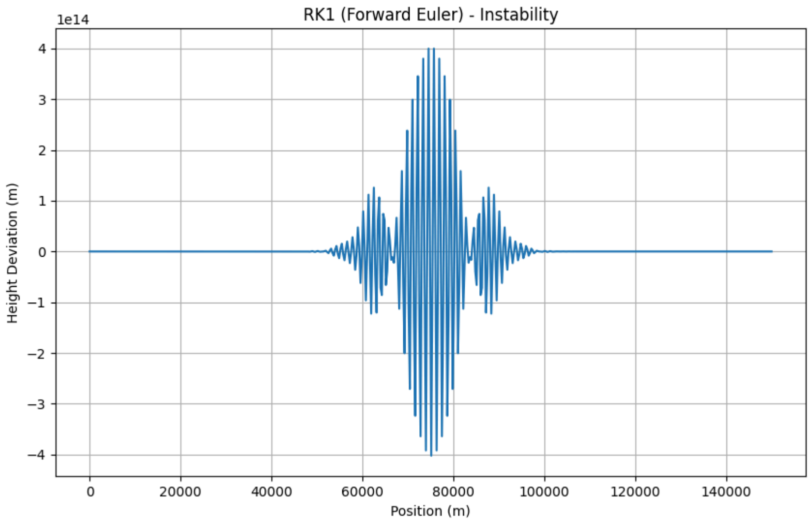

Figure 1: Instability of RK1 (Forward Euler) with centered differences

Runge-Kutta 2 (RK2)

We implemented Heun's method (Explicit Trapezoidal Rule):

$$\begin{aligned} k_1 &= f(U^n) \\ k_2 &= f(U^n + \Delta t k_1) \\ U^{n+1} &= U^n + \frac{\Delta t}{2}(k_1 + k_2) \end{aligned} \tag{7}$$

This method also showed instability for the simulation parameters used (CFL ≈ 1). While it is more stable than RK1 for some problems (e.g., diffusion), for the purely hyperbolic wave equation with centered differences, it remains unstable.

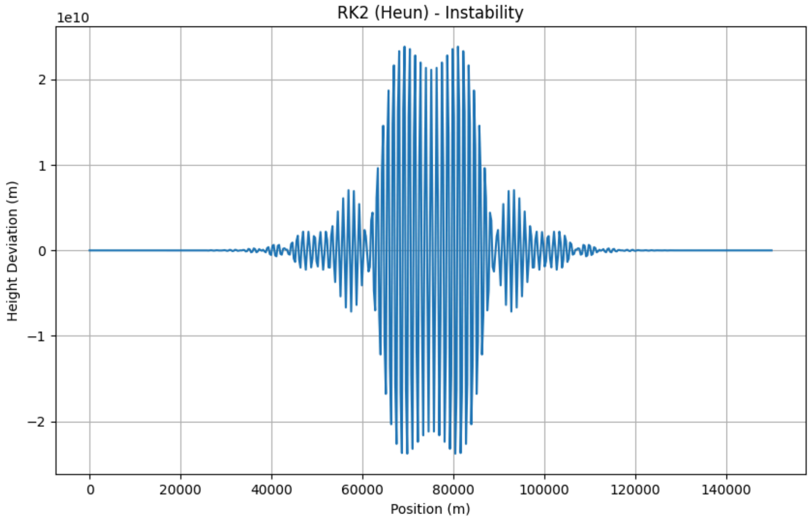

Figure 2: Instability of RK2 (Heun) with centered differences

Runge-Kutta 4 (RK4)

We implemented the classical 4th-order Runge-Kutta method:

$$\begin{aligned} k_1 &= f(U^n) \\ k_2 &= f\left(U^n + \frac{\Delta t}{2} k_1\right) \\ k_3 &= f\left(U^n + \frac{\Delta t}{2} k_2\right) \\ k_4 &= f(U^n + \Delta t k_3) \\ U^{n+1} &= U^n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \quad [8]$$

This method provides a stable solution for the wave equation, preserving the shape of the Gaussian bump as it propagates.

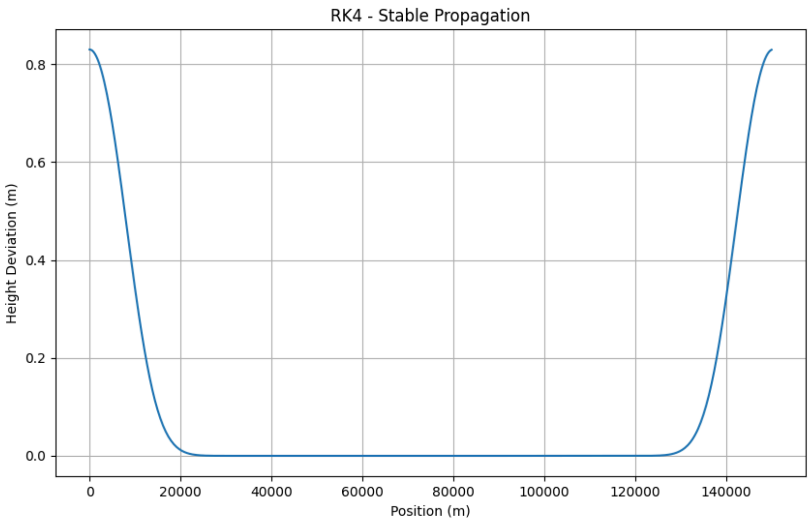

Figure 3: Stable propagation with RK4

4: Convergence Analysis

Analytical Solution

The linearized shallow water equations are given by:

$$\frac{\partial h}{\partial t} + H\frac{\partial v}{\partial x} = 0 \quad [9]$$

$$\frac{\partial v}{\partial t} + g\frac{\partial h}{\partial x} = 0 \quad [10]$$

Differentiating the first equation with respect to t and the second with respect to x:

$$\frac{\partial^2 h}{\partial t^2} + H\frac{\partial^2 v}{\partial x \partial t} = 0 \quad [11]$$

$$\frac{\partial^2 v}{\partial t \partial x} + g\frac{\partial^2 h}{\partial x^2} = 0 \quad [12]$$

Substituting the mixed derivative term:

$$\frac{\partial^2 h}{\partial t^2} - gH\frac{\partial^2 h}{\partial x^2} = 0 \quad [13]$$

This is the classical wave equation $h_{tt} - c^2 h_{xx} = 0$ with wave speed $c = \sqrt{gH}$. The general solution consists of left- and right-traveling waves. For a purely right-traveling wave $h(x,t) = f(x - ct)$, we have:

$$\frac{\partial h}{\partial t} = -cf'(x - ct) \quad [14]$$

$$\frac{\partial h}{\partial x} = f'(x - ct) \quad [15]$$

Substituting into the continuity equation:

$$-cf' + H\frac{\partial v}{\partial x} = 0 \Rightarrow \frac{\partial v}{\partial x} = \frac{c}{H}f' = \frac{c}{H}\frac{\partial h}{\partial x} \quad [16]$$

Integrating with respect to x (assuming $v = 0$ when $h = 0$):

$$v = \frac{c}{H}h = \frac{\sqrt{gH}}{H}h = \sqrt{\frac{g}{H}}h \quad [17]$$

This confirms the relation used for the solitary wave initial condition.

Convergence Results

We performed a convergence analysis using the RK4 time stepper and centered finite differences. The simulation was run for one full period ($T = \frac{L}{c}$) on grids with $N = 64, 128, \ldots, 1024$ points.

To isolate the spatial discretization error, we used a small CFL number (CFL = 0.1). This ensures that the time integration error (which is 4th order) is negligible compared to the spatial error.

The L^2 error was computed between the final state and the initial state.

Table 1: Convergence table for Solitary Wave benchmark (CFL=0.1)

N	L2 Error	Order
64	1.39e-01	-
128	5.12e-02	1.44
256	1.37e-02	1.91
512	3.44e-03	1.99
1024	8.60e-04	2.00

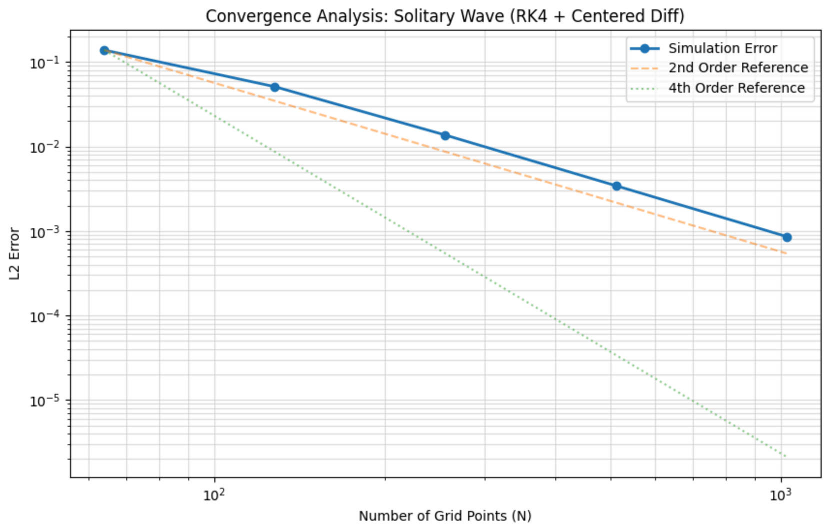

Figure 4: Convergence plot showing L2 error vs Number of DoFs

The results clearly show a convergence order of 2. This is consistent with the second-order centered finite difference scheme used for spatial discretization. The error decreases by a factor of 4 when the grid resolution is doubled.

(Note: We observed that using a larger time step corresponding to CFL ≈ 1 resulted in anomalously high convergence rates (approx. 4th order) and lower absolute errors, likely due to error cancellation between spatial and temporal discretization terms. We chose CFL $= 0.1$ to focus on the order of the spatial operator.)

5: Individual Assignment: Solitary Wave

We implemented the solitary wave benchmark (Assignment 6.2). The goal is to set an initial condition for the velocity v such that a single wave travels to the right.

For the linear wave equation $h_t + ch_x = 0$ (right-traveling wave), the relationship between height and velocity is given by:

$$v = \sqrt{\frac{g}{H}}h \qquad [18]$$

We implemented this in `main.cpp`:

C++
```cpp
else if (config.benchmark_name == "solitary_wave") {
    for (int i = 0; i < discConfig.num_dofs; i++) {
        // Setup relative surface height
        double x = i*discConfig.dx;
        double y = (x - discConfig.domain_size*1.0/2.0)/discConfig.domain_size;
        h.data[i] = std::exp(-300*(y*y));

        // Setup velocity: v = sqrt(g/h_bar) * h
        double g = config.sim_g;
        double h_bar = std::abs(config.sim_bavg);
        v.data[i] = std::sqrt(g/h_bar) * h.data[i];

        // Setup bathymetry
        b.data[i] = config.sim_bavg;    } }
```

Running the simulation with RK4 and this benchmark resulted in a stable wave propagating to the right, confirming the derivation.