Contents

1 Stability of Numerical Schemes

We will work with the given general scheme:

$$U^{n+1} = Q(t_n)U^n + \Delta t F^n$$

$$U^0 = g$$

where $U^n \in \mathbb{R}^d$.

1.1 Discrete Duhamel's Principle

We want to probe that the following discrete Duhamel's principle holds:

$$U^n = S_{\Delta t}(t_n, 0)g + \Delta t \sum_{\nu=0}^{n-1} S_{\Delta t}(t_n, t_{\nu+1})F^\nu$$

where $t_n = n\Delta t$ and

$$S_{\Delta t}(t,t) = I, \forall t \in \mathbb{R}$$

$$S_{\Delta t}(t_{n+1}, t_\mu) = Q(t_n)S_{\Delta t}(t_n, t_\mu)$$

We will start deducing the general formula for U^{n+1} based on U^0, that we can know developing the 3 first terms:

$$U^0 = g$$

$$U^1 = Q(t_0)U^0 + \Delta t F^0 = Q(t_0)g + \Delta t F^0$$

$$U^2 = Q(t_1)\cdot\left(Q(t_0)g + \Delta t F^0\right) + \Delta t F^1 = Q(t_1)Q(t_0)g + Q(t_1)\Delta t F^0 + \Delta t F^1$$

$$U^3 = Q(t_2)\cdot\left(Q(t_1)Q(t_0)g + Q(t_1)\Delta t F^0 + \Delta t F^1\right) + \Delta t F^2 =$$

$$= Q(t_2)Q(t_1)Q(t_0)g + Q(t_2)Q(t_1)\Delta t F^0 + Q(t_2)\Delta t F^1 + \Delta t F^2$$

So we can see that the general case is:

$$U^{n+1} = \left(\prod_{i=0}^{n} Q(t_i)\right)\cdot g + \sum_{\nu=0}^{n}\left(\prod_{j=\nu+1}^{n} Q(t_j)\right)\cdot \Delta t F^\nu$$

Now, we want to see what is the true value of $S_{\Delta t}(t_{n+1}, t_\mu)$, that we know is defined recursively:

$$S_{\Delta t}(t_{n+1}, t_\mu) = Q(t_n)\cdot S_{\Delta t}(t_n, t_\mu) = Q(t_n)Q(t_{n-1})S_{\Delta t}(t_{n-1}, t_\mu) =$$

$$= Q(t_n)Q(t_{n-1})Q(t_{n-2})S_{\Delta t}(t_{n-2}, t_\mu) = ... = Q(t_n)Q(t_{n-1})Q(t_{n-2})\cdots Q(t_\mu)S_{\Delta t}(t_\mu, t_\mu) =$$

$$= Q(t_n)Q(t_{n-1})Q(t_{n-2})\cdots Q(t_\mu)\cdot I = Q(t_n)Q(t_{n-1})Q(t_{n-2})\cdots Q(t_\mu) = \prod_{i=\mu}^{n} Q(t_i)$$

So then, we can replace the products in the general equation for U^{n+1} and we finally get:

$$U^{n+1} = S_{\Delta t}(t_{n+1}, 0)\cdot g + \Delta t \cdot \sum_{\nu=0}^{n} S_{\Delta t}(t_{n+1}, t_{\nu+1})\cdot F^\nu$$

that is the same formula as the one we were looking to probe.

If we consider U^n to be the solution at the time level n, and we assume that it should be *sum of all the solutions that satisfy each of the initial conditions*, where at each time level we get additional "initial conditions" that are hold by F, then in the formula of U: $U^n = S_{\Delta t}(t_n, 0)g + \Delta t \sum_{\nu=0}^{n-1} S_{\Delta t}(t_n, t_{\nu+1})F^\nu$ we can see that $S_{\Delta t}(t_n, 0)$ is the solution at the time level 0, multiplied by the initial conditions g, and then we add to this solutions the different solutions at each time level $\nu + 1$, $S_{\Delta t}(t_n, t_{\nu+1})$, multiplied each one by its time level initial conditions F^ν, and each one of this solutions are multiplied by the time variation Δt (the size of the time level) that, as all time levels have the same size, is taken out of the sum as a common factor.

1.2 Solutions Norm Inequality

We want to show that if $||S_{\Delta t}(t_{\nu+1}, t_\nu)||_h = ||Q(t_\nu)||_h \leq Ke^{\alpha \Delta t}$ for some constants α and K independents of Δt, then the following inequality holds:

$$||U^n||_h \leq K \left(e^{\alpha t_n} ||g||_h + \int_0^{t_n} e^{\alpha(t_n - s)} ds \cdot \max_{0 \leq \nu \leq n-1} ||F^\nu||_h \right)$$

We will start taking the formulation for U^n from the Task 1.1 and we will make the norm h of this formula:

$$||U^n||_h = ||S_{\Delta t}(t_n, 0)g + \Delta t \sum_{\nu=0}^{n-1} S_{\Delta t}(t_n, t_{\nu+1})F^\nu||_h$$

Now, by the triangular inequality property of the norm, we have the following formulation:

$$||U^n||_h \leq ||S_{\Delta t}(t_n, 0)g||_h + ||\Delta t \sum_{\nu=0}^{n-1} S_{\Delta t}(t_n, t_{\nu+1})F^\nu||_h \leq$$

$$\leq ||S_{\Delta t}(t_n, 0)||_h \cdot ||g||_h + ||\Delta t||_h \cdot || \sum_{\nu=0}^{n-1} S_{\Delta t}(t_n, t_{\nu+1})F^\nu||_h \leq$$

$$\leq ||S_{\Delta t}(t_n, 0)||_h \cdot ||g||_h + ||\Delta t||_h \cdot \sum_{\nu=0}^{n-1} ||S_{\Delta t}(t_n, t_{\nu+1})F^\nu||_h \leq$$

$$\leq ||S_{\Delta t}(t_n, 0)||_h \cdot ||g||_h + ||\Delta t||_h \cdot \sum_{\nu=0}^{n-1} ||S_{\Delta t}(t_n, t_{\nu+1})||_h \cdot ||F^\nu||_h$$

Now, as $||S_{\Delta t}(t_{\nu+1}, t_\nu)||_h = ||Q(t_\nu)||_h \leq Ke^{\alpha \Delta t}$, the continuous statement would be $||S_{\Delta t}(t_n, t_\nu)||_h = ||\prod_{i=\nu}^{n-1} Q(t_i)||_h \leq K \prod_{i=\nu}^{n-1} e^{\alpha \Delta t} \leq K \left(e^{\alpha \Delta t}\right)^{n-\nu}$, so we have the following formulation:

$$||U^n||_h \leq K \left(e^{\alpha \Delta t}\right)^n \cdot ||g||_h + ||\Delta t||_h \cdot \sum_{\nu=0}^{n-1} K \left(e^{\alpha \Delta t}\right)^{n-(\nu+1)} \cdot ||F^\nu||_h =$$

$$= Ke^{n\alpha \Delta t} \cdot ||g||_h + ||\Delta t||_h \cdot \sum_{\nu=0}^{n-1} Ke^{(n-(\nu+1))\alpha \Delta t} \cdot ||F^\nu||_h =$$

$$= K \cdot \left(e^{\alpha t_n} \cdot ||g||_h + ||\Delta t||_h \cdot \sum_{\nu=0}^{n-1} e^{\alpha \cdot (t_n - t_{\nu+1})} \cdot ||F^\nu||_h \right) \leq$$

$$\leq K \cdot \left(e^{\alpha t_n} \cdot ||g||_h + 1 \cdot \left(\sum_{\nu=0}^{n-1} e^{\alpha \cdot (t_n - t_{\nu+1})} \right) \cdot \max_{0 \leq \nu \leq n-1} ||F^\nu||_h \right) \leq$$

$$\leq K \cdot \left(e^{\alpha t_n} \cdot ||g||_h + \int_0^{t_n} e^{\alpha \cdot (t_n - s)} ds \cdot \max_{0 \leq \nu \leq n-1} ||F^\nu||_h \right)$$

So finally we show that the inequality holds, and this means that *stability of the homogeneous problem ($F \equiv 0$) implies stability for the inhomogeneous problem.*

1.3 Solutions Norm Inequality with Δt dependence

If $\alpha = \Delta t^{-1/2}$ in the previous inequality, then $||S_{\Delta t}(t_{\nu+1}, t_\nu)||_h = ||Q(t_\nu)||_h \leq K e^{\alpha \Delta t} = K e^{\Delta t^{-1/2} \Delta t} = K e^{\Delta t^{1/2}}$ and then

$$||U^n||_h \leq K \left(e^{\Delta t^{1/2}} \right)^n \cdot ||g||_h + ||\Delta t||_h \cdot \sum_{\nu=0}^{n-1} K \left(e^{\Delta t^{1/2}} \right)^{n-(\nu+1)} \cdot ||F^\nu||_h =$$

$$= K e^{n \Delta t^{1/2}} \cdot ||g||_h + ||\Delta t||_h \cdot \sum_{\nu=0}^{n-1} K e^{(n-(\nu+1)) \Delta t^{1/2}} \cdot ||F^\nu||_h =$$

$$= K \cdot \left(e^{\Delta t^{-1/2} t_n} \cdot ||g||_h + ||\Delta t||_h \cdot \sum_{\nu=0}^{n-1} e^{\Delta t^{-1/2} \cdot (t_n - t_{\nu+1})} \cdot ||F^\nu||_h \right) \leq$$

$$\leq K \cdot \left(e^{\Delta t^{-1/2} t_n} \cdot ||g||_h + \int_0^{t_n} e^{\Delta t^{-1/2} \cdot (t_n - s)} ds \cdot \max_{0 \leq \nu \leq n-1} ||F^\nu||_h \right)$$

As we can see, now the exponents depends on the value of Δt, so depending on the grid size we have selected we will get different values for the formula, but the norm of the solution U^n is always the same, so with a big enough selection of Δt we can get a value lower than $||U^n||_h$.

Then α and K can't depend on Δt if we want an upper bound for $||U^n||_h$.

2 The shallow water model

2.1 Numerical solution

2.1.1 Derivation of numerical formulas

For compressing the algorithms we insert the Lax-Friedrichs flux into the equation of the method resulting in

$$\begin{aligned} u_j^{n+1} &= u_j^n - \frac{h_t}{h_x}(F^{LxF}(u_{j+1}^n, u_j^n) - F^{LxF}(u_j^n, u_{j-1}^n)) \\ &= u_j^n - \frac{h_t}{h_x}(\frac{1}{2}(f(u_{j+1}^n) + f(u_j^n) - \alpha(u_{j+1}^n - u_j^n)) - \frac{1}{2}(f(u_j^n) + f(u_{j-1}^n) - \alpha(u_j^n - u_{j-1}^n))) \\ &= u_j^n - \frac{h_t}{2h_x}(f(u_{j+1}^n) - f(u_{j-1}^n) - \alpha(u_{j+1}^n - 2u_j^n + u_{j-1}^n)) \end{aligned}$$

for $i = 1, ..., N$. Using $u_j^n = [u_j^n[1], u_j^n[2]]$ with $u_j^n[1] := h_j^n$ and $u_j^n[2] := h_j^n v_j^n$ we get $f(u_j^n[1]) = u_j^n[2]$ and $f(u_j^n[2]) = u_j^n[2]^2 / u_j^n[1] + \frac{1}{2} g \cdot u_j^n[1]^2$. We therefore get two equations for the two values of u_j^n:

$$\begin{aligned} u_j^{n+1}[1] &= u_j^n[1] - \frac{h_t}{2h_x}(u_{j+1}^n[2] - u_{j-1}^n[2] - \alpha(u_{j+1}^n[1] - 2u_j^n[1] + u_{j-1}^n[1])) \\ u_j^{n+1}[2] &= u_j^n[2] - \frac{h_t}{2h_x}(u_{j+1}^n[2]^2 / u_{j+1}^n[1] + \frac{1}{2} g \cdot u_{j+1}^n[1]^2 - u_{j-1}^n[2]^2 / u_{j-1}^n[1] - \frac{1}{2} g \cdot u_{j-1}^n[1]^2 \\ &\quad - \alpha(u_{j+1}^n[2] - 2u_j^n[2] + u_{j-1}^n[2])) \end{aligned}$$

These equations (especially the second one) seem to be quite complicated, but we can rewrite them later in a matrix form. Before we can do that we have to implement the boundary conditions, using the ones described in the book (and also explained in the homework)

$$\begin{aligned} u_0^n[1] &= u_1^n[1] & \qquad u_{n+1}^n[1] &= u_n^n[1] \\ u_0^n[2] &= -u_1^n[2] & \qquad u_{n+1}^n[2] &= -u_n^n[2] \end{aligned}$$

This results in the following boundary equations

$$\begin{aligned} u_1^{n+1}[1] &= u_1^n[1] - \frac{h_t}{2h_x}(u_2^n[2] + u_1^n[2] - \alpha(u_2^n[1] - u_1^n[1]) \\ u_1^{n+1}[2] &= u_1^n[2] - \frac{h_t}{2h_x}(u_2^n[2]^2/u_2^n[1] + \frac{1}{2}g \cdot u_2^n[1]^2 - u_1^n[2]^2/u_1^n[1] - \frac{1}{2}g \cdot u_1^n[1]^2 - \alpha(u_2^n[2] - 3u_1^n[2]) \end{aligned}$$

$$\begin{aligned} u_N^{n+1}[1] &= u_N^n[1] - \frac{h_t}{2h_x}(-u_N^n[2] - u_{N-1}^n[2] - \alpha(-u_N^n[1] + u_{N-1}^n[1])) \\ u_N^{n+1}[2] &= u_N^n[2] - \frac{h_t}{2h_x}(u_N^n[2]^2/u_N^n[1] + \frac{1}{2}g \cdot u_N^n[1]^2 - u_{N-1}^n[2]^2/u_{N-1}^n[1] - \frac{1}{2}g \cdot u_{N-1}^n[1]^2 \\ &\quad - \alpha(-3u_N^n[2] + u_{N-1}^n[2])) \end{aligned}$$

Together we can now combine everything to get the matrix form of the equations

$$\begin{aligned} u^{n+1}[1] &= S_1 u^n[2] + S_2 u^n[1] \\ u_j^{n+1}[2] &= T_1(u^n[2]^2/u^n[1] + \frac{1}{2}g \cdot u^n[1]^2) + T_2 u^n[2] \end{aligned}$$

Note here, that $u^n[2]^2/u^n[1]$ and $u^n[1]^2$ are computed element-wise to ensure the non-linearity. The used matrices are defined below:

$$S_1 = -\frac{h_t}{2h_x}\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & \ddots & 0 & \ddots & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \end{pmatrix} \qquad S_2 = I + \frac{\alpha h_t}{2h_x}\begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

$$T_1 = -\frac{h_t}{2h_x}\begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & \ddots & 0 & \ddots & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \qquad T_2 = I + \frac{\alpha h_t}{2h_x}\begin{pmatrix} -3 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -3 \end{pmatrix}$$

2.1.2 Numerical results with constant ϵ

In the following plots one can find the solutions for $\epsilon = 0.1$ at different times. We used $n_x = 501$ for the discretisation. We only had stable solutions until $C \approx 0.3$ with this ϵ, resulting in $h_t = 0.006$.

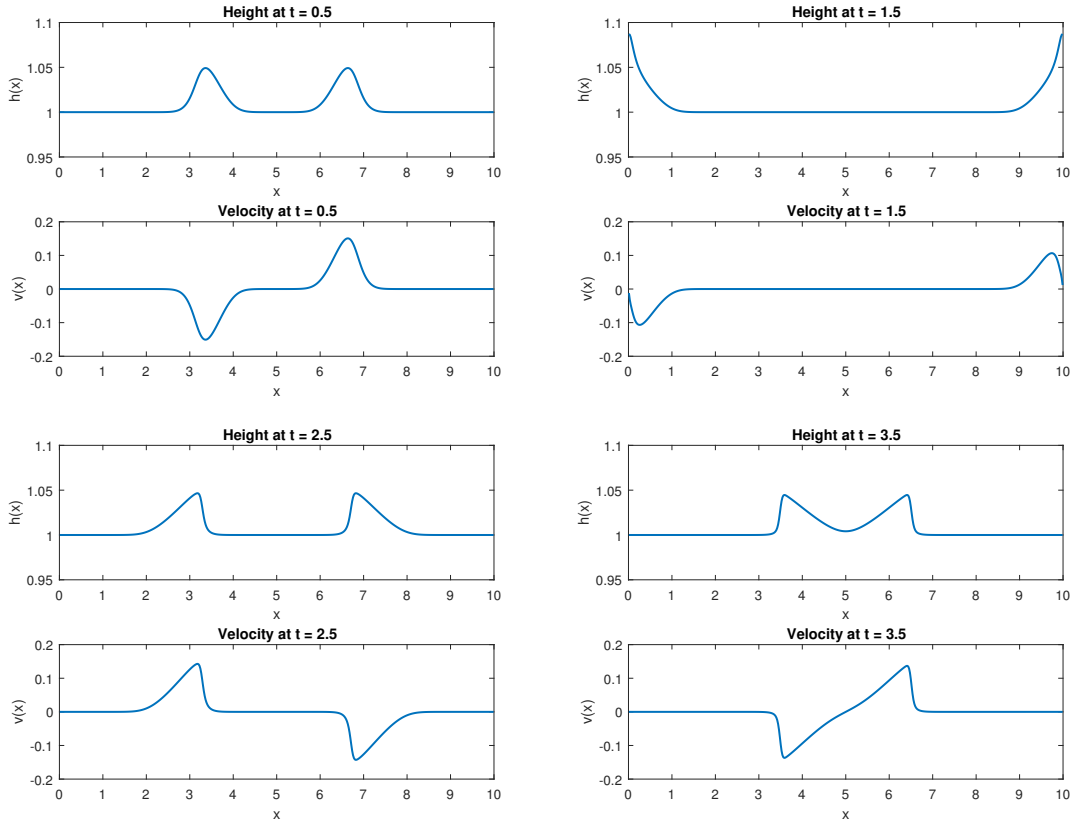

Figure 2.1.1: Computed solution at different times for $\epsilon = 0.1$, $n_x = 501$ and $C = 0.3$

We nicely observe two waves travelling in opposite directions. Over time they get more steep and get reflected at the boundaries. Choosing higher C results in the wave getting too steep, producing dispersive effects and eventually crashing the solution. Reducing C results in a quick smoothening out of the curve. A small investigation for C can be found in the following plots:

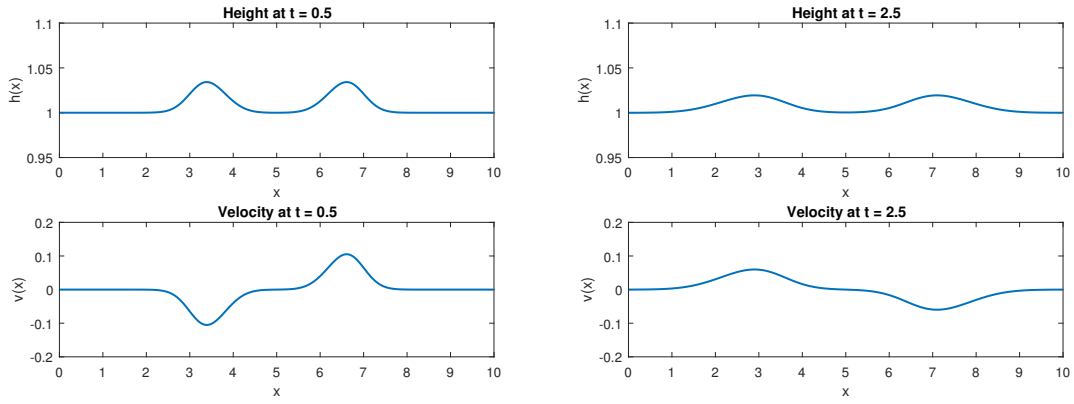

Figure 2.1.2: Computed solution at different times for $\epsilon = 0.1$, $n_x = 501$ and $C = 0.1$

In the plots from figure 2.1.2 we can nicely see the damping by choosing a C being too small.

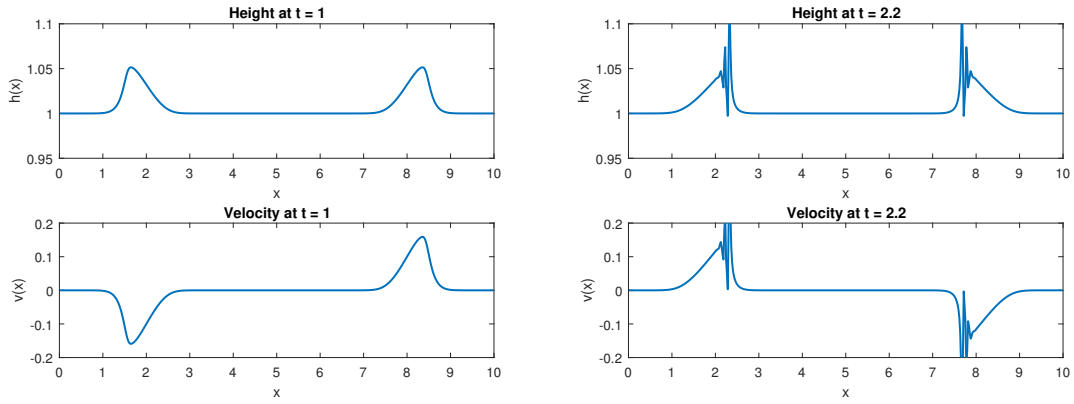

Figure 2.1.3: Computed solution at different times for $\epsilon = 0.1$, $n_x = 501$ and $C = 0.31$

In the plots from figure 2.1.3 we can observe how choosing a $C > 0.3$ results in a more steep solution creating dispersive effects after some time and eventually crashing.

2.1.3 Numerical results varying ϵ

Varying ϵ results in different changes. Most of them can be seen choosing a constant C for varying ϵ, as seen in the following plot:

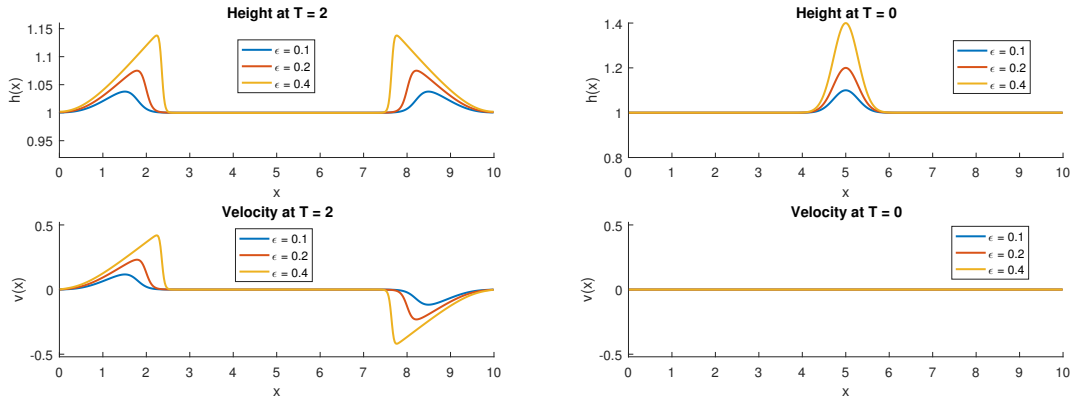

Figure 2.1.4: Computed solution with different ϵ for $T = 2$, $n_x = 501$ and $C = 0.25$ and it's initial condition

With increasing ϵ we have a higher initial condition, this leads to quicker sharpening the curve over time. Because of that we have to reduce C, and therefore reducing the time-step for larger ϵ, to ensure stability. Physically speaking this makes absolute sense, since higher parts of the waves have higher speed, resulting in a quicker breakdown if not damped enough. Besides speed, the behaviour of the solution at the boundary and while passing each other also stays the same:

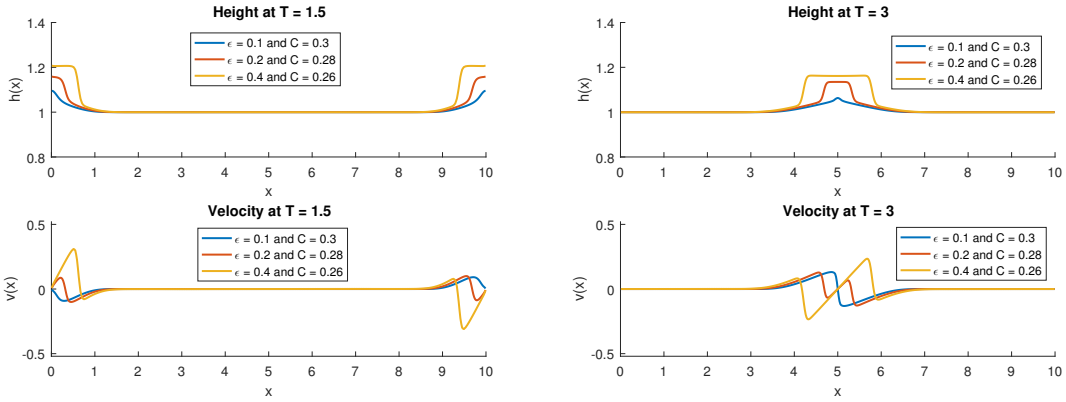

Figure 2.1.5: Computed solution with different ϵ at the boundary and while passing with $n_x = 501$ and highest C-values possible

We observe, that all solutions have a similar behaviour at the boundaries and while passing. The difference between the solution from $\epsilon = 0.1$ and the other two in the velocity in the right plot comes from the width of the solutions and the tiny difference in the speed of solutions, to which one can find plots in the following figure.

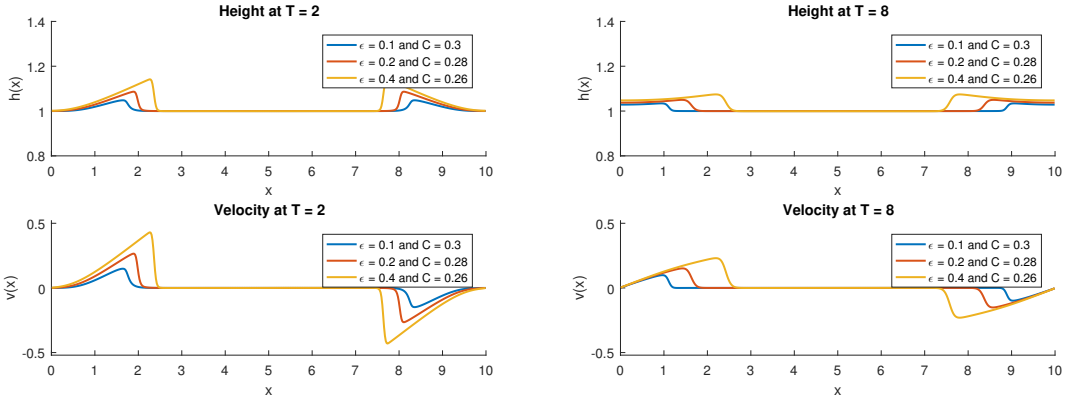

Figure 2.1.6: Computed solution with different ϵ at different times with $n_x = 501$ and highest C-values possible

Once can see, that with increasing ϵ the solution travels a bit faster. Which again makes physically sense due to the increased speed of taller waves.

2.1.4 Investigation of changes in α

In- or decreasing α results in different changes. First observations are caught in the following plots:

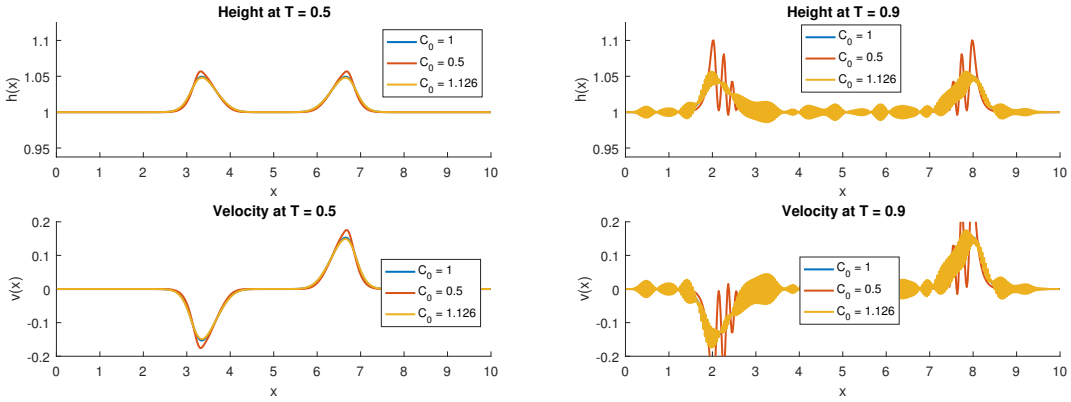

Figure 2.1.7: Computed solution with different C_0 at different times with $n_x = 501$ and $C = 0.3$

One can see, that at first all solutions look quite alike with smaller α having a higher peak, but after some time with the previous optimal C-value the other solutions break down. The solution with larger C_0 starts to oscillate while lower C_0 result in occurrence of dispersive effects. Following one can find a table with values of C_0 and maximum tested value of C with still a solution after $T = 10$ (approximately maximum stable solution with accuracy of $\Delta C = 0.05$).

C_0	maximum C
1	0.3
0.9	0.285
0.8	2.7
0.7	2.55
0.6	2.35
0.5	2.15

With decreasing C_0 we only achieve stable solutions with lower C-values. due to faster dispersive effects. This occurs since for resulting $\alpha \to 0$ our scheme becomes the Forward-Time Central-Space (FTCS) method, which is unconditionally unstable for hyperbolic problems. For $C_0 > 1$ one can observe the following:

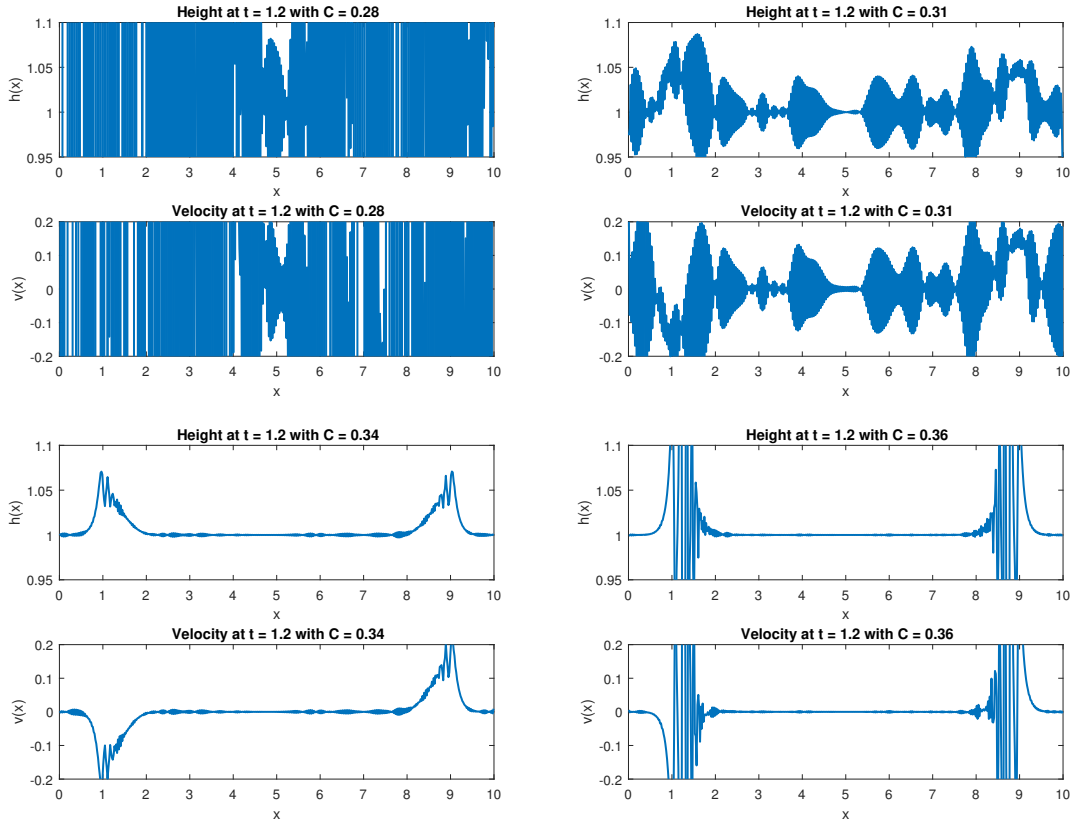

Figure 2.1.8: Computed solution with $C_0 = 1.1$ at $T = 1.2$ with $n_x = 501$ and different C-values

We see, that the oscillations start to vanish with larger C, but since we have to increase our C value over the maximum stable C-value our solution gets unstable over time (noticeable by the dispersive effects with increasing C). This means, that for all $C_0 > 1$ we get unstable results. The oscillations start to occur since for $C_0 > 1$ we violate the CFL-condition being $|a\frac{h_t}{h_x}| \leq 1$.
In total choosing $C_0 = 1$ therefore results in the best behaviour able to get.

2.2 Linearisation

2.2.1 Derivation of linearised form

Using $u = \begin{pmatrix} h \\ hv \end{pmatrix}$, one can rewrite the shallow water model in the form

$$\partial_t u + \partial_x f(u) = 0$$
$$\text{with } f(u) = \begin{pmatrix} hv \\ hv^2 + \frac{1}{2}gh^2 \end{pmatrix}$$
$$\text{and } u(x,0) = \begin{pmatrix} H \\ 0 \end{pmatrix} + \epsilon \begin{pmatrix} e^{-\frac{(x-\frac{L}{2})^2}{w^2}} \\ 0 \end{pmatrix}$$

This form can easily be linearised. Using $v_c = \begin{pmatrix} H \\ 0 \end{pmatrix}$ we can approximate

$$\begin{aligned} u(x,t) &\approx v_c + \epsilon\tilde{u}(x,t) \\ \text{with } \partial_t\tilde{u}(x,t) + f'(v_c)\partial_x\tilde{u}(x,t) &= 0 \end{aligned}$$

To achieve this form we need $f'(v_c)$:

$$\begin{aligned} f'(u) &= \begin{pmatrix} v & 1 \\ v^2 + gh & v \end{pmatrix} \\ \Leftrightarrow f'(v_c) &= \begin{pmatrix} 0 & 1 \\ gH & 0 \end{pmatrix} \end{aligned}$$

With $\tilde{u}(x,0) = \begin{pmatrix} e^{-\frac{x-\frac{L}{2}}{w^2}} \\ 0 \end{pmatrix}$ we finally get

$$\begin{aligned} u(x,t) &\approx \begin{pmatrix} H \\ 0 \end{pmatrix} + \epsilon\tilde{u}(x,t) \\ \text{with } \partial_t\tilde{u}(x,t) + &\begin{pmatrix} 0 & 1 \\ gH & 0 \end{pmatrix} \partial_x\tilde{u}(x,t) = 0 \\ \text{and } \tilde{u}(x,0) &= \begin{pmatrix} e^{-\frac{(x-\frac{L}{2})^2}{w^2}} \\ 0 \end{pmatrix} \end{aligned}$$

The eigenvalues of $f'(v_c)$ are $\lambda_{1/2} = \pm\sqrt{gH}$. Since g and H are positive, both eigenvalues are real and our problem is therefore hyperbolic. Both eigenvalues are also the wave speeds.

2.2.2 Analytical solution of the linearised problem

For the analytical problem from 2.2.1 we can derive an analytical solution. The linearised PDE $\partial_t\tilde{u}(x,t) + \begin{pmatrix} 0 & 1 \\ gH & 0 \end{pmatrix} \partial_x\tilde{u}(x,t) = 0$ has the properties

$$\Lambda = \begin{pmatrix} -\sqrt{gH} & 0 \\ 0 & \sqrt{gH} \end{pmatrix} \qquad V = \begin{pmatrix} -\frac{1}{\sqrt{gH}} & \frac{1}{\sqrt{gH}} \\ 1 & 1 \end{pmatrix} \qquad V^{-1} = \frac{1}{2}\begin{pmatrix} -\sqrt{gH} & 1 \\ \sqrt{gH} & 1 \end{pmatrix}$$

Using that we can decouple the equations using $z = V^{-1}\tilde{u}$

$$\partial_t z_p + \lambda_p \partial_x z_p = 0 \qquad \text{with } z_p(x,0) = z_p^0(x) = (V^{-1}u_0(x))_p$$

the solutions of these equations are constant along $\gamma_p(x,t) = \lambda_p t + x_0$, resulting in the general solution

$$\begin{aligned} z_p(x,t) &= z_p^0(x - \lambda_p t) \\ \tilde{u}(x,t) &= Vz(x,t) \end{aligned}$$

Calculating z^0 and $\tilde{x} := x - \lambda_p t$ gives us the analytical solution

$$\begin{aligned}
z^0(x) = V^{-1}u_0(x) &= \frac{1}{2}\begin{pmatrix} -\sqrt{gH} & 1 \\ \sqrt{gH} & 1 \end{pmatrix}\begin{pmatrix} e^{-\frac{x-\frac{L}{2}}{w^2}} \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} -\frac{\sqrt{gH}}{2} exp(-\frac{(x-\frac{L}{2})^2}{w^2}) \\ \frac{\sqrt{gH}}{2} exp(-\frac{(x-\frac{L}{2})^2}{w^2}) \end{pmatrix} \\
\tilde{x} &= \begin{pmatrix} \sqrt{gH}t + x \\ -\sqrt{gH}t + x \end{pmatrix} \\
\Leftrightarrow z(x,t) &= z^0(\tilde{x}) \\
&= \begin{pmatrix} -\frac{\sqrt{gH}}{2} exp(-\frac{(\sqrt{gH}t+x-\frac{L}{2})^2}{w^2}) \\ \frac{\sqrt{gH}}{2} exp(-\frac{(-\sqrt{gH}t+x-\frac{L}{2})^2}{w^2}) \end{pmatrix} \\
\Leftrightarrow \tilde{u}(x,t) &= Vz(x,t) \\
&= \begin{pmatrix} -\frac{1}{\sqrt{gH}} & \frac{1}{\sqrt{gH}} \\ 1 & 1 \end{pmatrix}\begin{pmatrix} -\frac{\sqrt{gH}}{2} exp(-\frac{(\sqrt{gH}t+x-\frac{L}{2})^2}{w^2}) \\ \frac{\sqrt{gH}}{2} exp(-\frac{(-\sqrt{gH}t+x-\frac{L}{2})^2}{w^2}) \end{pmatrix} \\
&= \begin{pmatrix} \frac{1}{2}(exp(-\frac{(-\sqrt{gH}t+x-\frac{L}{2})^2}{w^2}) + exp(-\frac{(\sqrt{gH}t+x-\frac{L}{2})^2}{w^2})) \\ \frac{\sqrt{gH}}{2}(exp(-\frac{(-\sqrt{gH}t+x-\frac{L}{2})^2}{w^2}) - exp(-\frac{(\sqrt{gH}t+x-\frac{L}{2})^2}{w^2})) \end{pmatrix}
\end{aligned}$$

Using this solution we can build the final analytical solution of our linearised problem

$$\begin{aligned}
u(x,t) &\approx \begin{pmatrix} H \\ 0 \end{pmatrix} + \epsilon\tilde{u}(x,t) \\
&\approx \begin{pmatrix} H \\ 0 \end{pmatrix} + \epsilon \begin{pmatrix} \frac{1}{2}(exp(-\frac{(-\sqrt{gH}t+x-\frac{L}{2})^2}{w^2}) + exp(-\frac{(\sqrt{gH}t+x-\frac{L}{2})^2}{w^2})) \\ \frac{\sqrt{gH}}{2}(exp(-\frac{(-\sqrt{gH}t+x-\frac{L}{2})^2}{w^2}) - exp(-\frac{(\sqrt{gH}t+x-\frac{L}{2})^2}{w^2})) \end{pmatrix}
\end{aligned}$$

This solution seems to be quite wild, but has an easy explanation. Our inital wavelet splits into two waves having each half of the amplitude of the initial wavelet and traveling with the same speed to the left and right direction. Computing this solution for $T = 1$ and plotting it together with the non-linear solution gives us the following figure

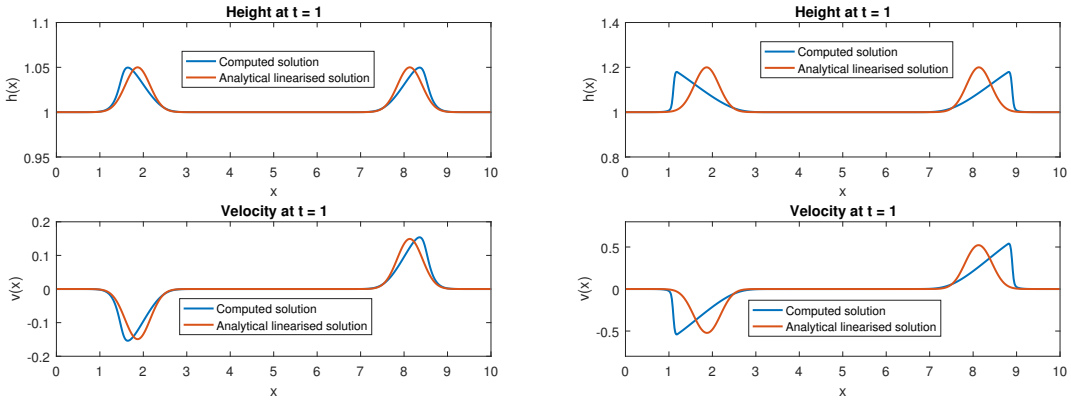

Figure 2.2.1: Analytical linearised and computed solution with $\epsilon = 0.1$ in the left and 0.4 in the right plot at $T = 1$ with $n_x = 501$ and $C = 0.3$

We can nicely see how the general behaviour of the solution is preserved. In both the linearised and the non-linearised case we have two waves traveling to the left and right at an approximately similar speed. The difference is the sharpening of the wave done by the non-linearisation which we don't have in the linearised case. Besides that the amplitude is also exact. The plots supports the claim, that for the non-linearised solution the higher parts of the wave travel faster than the smaller ones eventually resulting in the breakdown. This also explains the small difference in speed seen in the plot for $\epsilon = 0.4$.

2.2.3 Boundary conditions for the linearised case

Our linearised analytical solution behaves nicely on whole $\mathbb{R}$, but to fit our problem we need to implement boundary conditions. We have one wave traveling to the left (with speed λ_1) and one to the right (with speed λ_2). To have a well-posed problem we need to describe boundary conditions on $x = 0$ for the first and $x = L$ for the second wave. In addition we are not allowed to describe boundary conditions on the other ends for each wave. Since we have walls at each end, the velocity has to be zero there, the second component of u, being hv, therefore has to be zero at $x = 0$ and $x = L$. The first component at the boundaries is kept free. We now have to check if those conditions lead to a well-posed hyperbolic problem. To verify this we derive expressions for z_1 and z_2 at the boundaries as done in the lectures:

$$\begin{aligned}
\begin{pmatrix} * \\ 0 \end{pmatrix} = u(0,t) = Vz(0,t) = \begin{pmatrix} -\frac{1}{\sqrt{gH}} z_1 + \frac{1}{\sqrt{gH}} z_2 \\ z_1 + z_2 \end{pmatrix} \\
\Leftrightarrow 0 = z_1 + z_2 \rightarrow z_2(0,t) = -z_1(0,t) \\
\begin{pmatrix} * \\ 0 \end{pmatrix} = u(1,t) = Vz(1,t) = \begin{pmatrix} -\frac{1}{\sqrt{gH}} z_1 + \frac{1}{\sqrt{gH}} z_2 \\ z_1 + z_2 \end{pmatrix} \\
\Leftrightarrow 0 = z_1 + z_2 \rightarrow z_1(1,t) = -z_2(1,t)
\end{aligned}$$

Those derived expressions express the boundary conditions for each wave. At the left and right sides the wanted boundary condition came from the interior. The value of z_1 is reflected into z_2 at the left boundary and vice versa at the other side. Therefore our conditions cause reflections. The crucial part of the reflection will take part, when the top of our wave hits the boundary. This happens when the exponent $-\frac{(\sqrt{gH}t + x - \frac{L}{2})^2}{w^2}$ for the wave travelling to the left hits $x = 0$ and

reaches the value zero in total.

$$\begin{aligned} -\frac{(\sqrt{gH}t - \frac{L}{2})^2}{w^2} &= 0 \\ \Leftrightarrow \sqrt{gH}t &= \frac{L}{2} \\ \Leftrightarrow t &= \frac{L}{2\sqrt{gH}} \\ &\approx 1.596 \end{aligned}$$

After $T \approx 1.596$ most of the wave will be reflected and start travelling back. After the same amount of time the waves have travelled back to their origin and add each other up to form the initial condition again. Since our solution is the superposition of the two waves, nothing else than adding up happens when the waves pass each other.

2.3 Non-reflecting Boundary Conditions

For deriving the non-reflecting boundary conditions, we will get, from the first order extrapolation, the following ones:

$$\begin{aligned} u_0^n[1] &= u_1^n[1] & \qquad u_{n+1}^n[1] &= u_n^n[1] \\ u_0^n[2] &= u_1^n[2] & \qquad u_{n+1}^n[2] &= u_n^n[2] \end{aligned}$$

This results in the following boundary equations

$$u_1^{n+1}[1] = u_1^n[1] - \frac{h_t}{2h_x}(u_2^n[2] - u_1^n[2] - \alpha(u_2^n[1] - u_1^n[1])$$

$$u_1^{n+1}[2] = u_1^n[2] - \frac{h_t}{2h_x}(u_2^n[2]^2/u_2^n[1] + \frac{1}{2}g \cdot u_2^n[1]^2 - u_1^n[2]^2/u_1^n[1] - \frac{1}{2}g \cdot u_1^n[1]^2 - \alpha(u_2^n[2] - u_1^n[2])$$

$$u_N^{n+1}[1] = u_N^n[1] - \frac{h_t}{2h_x}(u_N^n[2] - u_{N-1}^n[2] - \alpha(-u_N^n[1] + u_{N-1}^n[1]))$$

$$\begin{aligned} u_N^{n+1}[2] = u_N^n[2] - \frac{h_t}{2h_x}(&u_N^n[2]^2/u_N^n[1] + \frac{1}{2}g \cdot u_N^n[1]^2 - u_{N-1}^n[2]^2/u_{N-1}^n[1] - \frac{1}{2}g \cdot u_{N-1}^n[1]^2 \\ &- \alpha(-u_N^n[2] + u_{N-1}^n[2])) \end{aligned}$$

Now the matrix form of the equations will be

$$\begin{aligned} u^{n+1}[1] &= S_1 u^n[2] + S_2 u^n[1] \\ u_j^{n+1}[2] &= T_1(u^n[2]^2/u^n[1] + \frac{1}{2}g \cdot u^n[1]^2) + T_2 u^n[2] \end{aligned}$$

with

$$S_1 = -\frac{h_t}{2h_x}\begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & \ddots & 0 & \ddots & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \qquad S_2 = I + \frac{\alpha h_t}{2h_x}\begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

$$T_1 = -\frac{h_t}{2h_x}\begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & \ddots & 0 & \ddots & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \qquad T_2 = I + \frac{\alpha h_t}{2h_x}\begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

Getting non-reflecting boundaries implies that the wave we have in the domain, whenever it reaches the limit, it will go out of the domain instead of reflecting inside the domain. As we can see on Figure 10, the new boundary conditions allows the wave to leave the domain, getting the non-reflecting desired behaviour. In contrast, the default boundary conditions keep the wave inside the domain, as we can see on Figure 11.

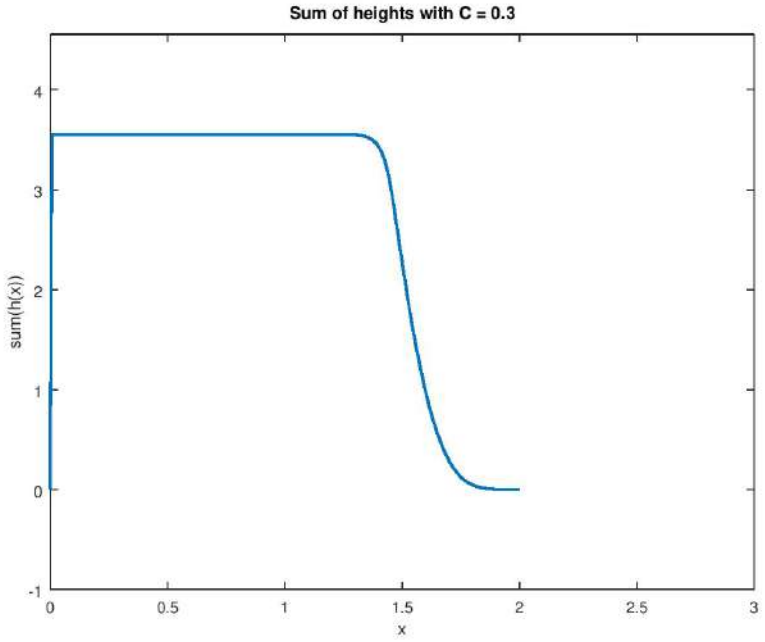

Figure 2.3.1: Sum of heights along time for non-reflecting boundaries with $\epsilon = 0.1$, $n_x = 501$ and $C = 0.3$.

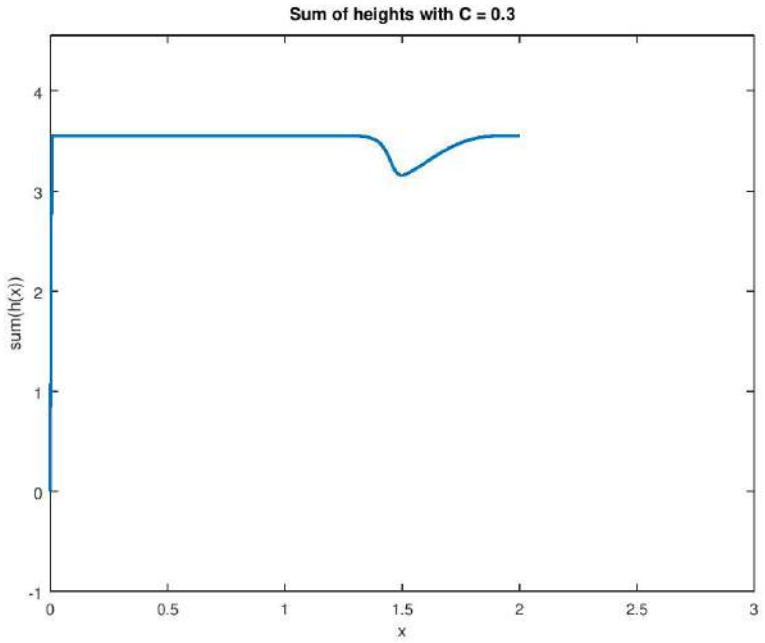

Figure 2.3.2: Sum of heights along time for default boundaries with $\epsilon = 0.1$, $n_x = 501$ and $C = 0.3$.

Numerically, the initial part of the graphs is a sum of total distortion of 3.5520, that for the non-reflecting boundaries case goes down to -0.0012071 (almost 0) what implies almost no reflection (a 0.034% of reflection), while for the reflecting boundaries case we get a final sum of total distortion of 3.5518, what implies that almost all the wave have been reflected (a 99.995% of reflection).

3 Appendix

3.1 HW2.m

```
%%% standalone script to solve the homework
clear

% variables
epsilon = 0.1; nx = 501; C = 0.3; T = 2; C0 = 1; % all important
    variables
plottime = 0; % setting to delay the plotting, only important for small
     amount of steps
beginplot = 0; % begin plot in percentage of T to skip parts

% constants
L = 10; g = 9.81; H = 1; w = 0.4;
hx = L/nx; ht = C*hx; nt = T/ht;
alpha = C0*hx/ht;
delay = plottime/nt;
```

```
% timestep at the end to reach exaclty T
htp = (nt-floor(nt))*ht; alphap = C0*hx/htp;

% discretise x-vector
x = linspace(hx/2,L-hx/2,nx);

%%% building of matrices being used
% S1
S1 = diag(-linspace(1,1,nx-1),-1) + diag(linspace(1,1,nx-1),1);
S1(1,1) = 1; S1(nx,nx) = -1;
% S2
S2 = diag(linspace(1,1,nx-1),-1) + diag(-2*linspace(1,1,nx)) + diag(
    linspace(1,1,nx-1),1);
S2(1,1) = -1; S2(nx,nx) = -1;
% T1
T1 = S1; T1(1,1) = -1; T1(nx,nx) = 1;
% T2
T2 = S2;
T2(1,1) = -3; T2(nx,nx) = -3;
% rescaling of matrices and finishing
hthx = ht/(2*hx);
S1 = -hthx * S1; T1 = -hthx * T1;
S2I = eye(nx) + hthx*alpha * S2; T2I = eye(nx) + hthx*alpha * T2;

% initial conditions
h0 = H + epsilon*exp(-(x-L/2).^2/w^2)';
v0 = x'*0; h0v0 = h0.*v0;
h = h0; v = v0; hv = h0v0;

%%% compute the timesteps
for it = 1:nt
    % compute h
    hnew = S1 * hv(:,it) + S2I * h(:,it);
    h = [h,hnew];
    % compute hv
    hvnew = T1 * (hv(:,it).^2./h(:,it) + g/2*h(:,it).^2) + T2I * hv(:,
        it);
    hv = [hv,hvnew];
    % compute v
    vnew = hvnew./hnew;
    v = [v,vnew];
end
%%% last step to get exactly to T
if htp ~= 0
    S1p = S1*htp/ht; T1p = T1*htp/ht;
    S2p = eye(nx) + htp/2/hx*alphap * S2; T2p = eye(nx) + htp/2/hx*
        alphap * T2;
    % compute h
    hnew = S1p * hv(:,it) + S2p * h(:,it);
    h = [h,hnew];
```

```
61      htest = [htest,hnew-1];
62      % compute hv
63      hvnew = T1p * (hv(:,it).^2./h(:,it) + g/2*h(:,it).^2) + T2p * hv(:,
            it);
64      hv = [hv,hvnew];
65      % compute v
66      vnew = hvnew./hnew;
67      v = [v,vnew];
68  end
69
70
71  %%% plot the data!
72  [~,SizeH] = size(h);
73  for it = floor(beginplot*(SizeH-1))+1:SizeH
74      subplot(211);
75      plot(x,h(:,it),'LineWidth',1.5);
76      axis([0,10,1- (max(h0)-1)/2,max(h0)]);
77      titlestringH = ['Height at t = ',num2str((it-1)*ht),' with C = ',
            num2str(C)];
78      if it == SizeH && htp ~= 0 % compute to exaclty T
79          titlestringH = ['Height at t = ',num2str((it-2)*ht + htp),'
                with C = ',num2str(C)];
80      end
81      title(titlestringH);
82      xlabel('x'); ylabel('h(x)');
83      subplot(212);
84      plot(x,v(:,it),'LineWidth',1.5);
85      axis([0,10,2*(1-max(h0)),2*(max(h0)-1)]);
86      xlabel('x'); ylabel('v(x)');
87      titlestringV = ['Velocity at t = ',num2str((it-1)*ht),' with C = ',
            num2str(C)];
88      if it == SizeH && htp ~= 0 % compute to exaclty T
89          titlestringV = ['Velocity at t = ',num2str((it-2)*ht + htp),'
                with C = ',num2str(C)];
90      end
91      title(titlestringV);
92
93      pause(delay);
94  end
```

3.2 investigate.m

```
1  %%% script to investigate for different values of a variable at once
2  % variables to be investigated
3  epsilon = [0.1,0.2,0.4];
4  C = [0.3,0.285,0.26];
5
6  % larger epsilons
7  % epsilon = [0.8,2,5];
8  % C = [0.22,0.163,0.11];
```

```
9
10 % other variables
11 T = 2; nx = 501; C0 = 1;
12
13 % changing factor for axis to get everything right on the plot
14 factorplotH = 3;
15 factorplotV = 1.3;
16
17 % some constants
18 L = 10; hx = L/nx; x = linspace(hx/2,L-hx/2,nx); % x for plotting
19
20 hplot = []; vplot = [];
21 subplot(211); hold on; subplot(212); hold on;
22 for ieps = 1:length(epsilon) % timeloop
23     % compute
24     [h,v] = compute_system(epsilon(ieps),nx,C(ieps),T,C0);
25     [~,SizeH] = size(h);
26     hplot = [hplot,h(:,SizeH)];
27     vplot = [vplot,v(:,SizeH)];
28
29     % plot
30     subplot(211);
31     plot(x,hplot(:,ieps),'LineWidth',1.5);
32     subplot(212);
33     plot(x,vplot(:,ieps),'LineWidth',1.5);
34 end
35 % edit plots (remove legend if less than 3 plots)
36 subplot(211);
37 axis([0,10,1- max(epsilon)/2/factorplotH,1+max(epsilon)/factorplotH]);
       xlabel('x'); ylabel('h(x)');
38 title(['Height at T = ',num2str(T)]);
39 legend(['\epsilon = ',num2str(epsilon(1)),' and C = ',num2str(C(1))],['
       \epsilon = ',num2str(epsilon(2)),' and C = ',num2str(C(2))],['\
       epsilon = ',num2str(epsilon(3)),' and C = ',num2str(C(3))]);
40 subplot(212);
41 axis([0,10,-factorplotV*max(epsilon),factorplotV*max(epsilon)]); xlabel
       ('x'); ylabel('v(x)');
42 title(['Velocity at T = ',num2str(T)]);
43 legend(['\epsilon = ',num2str(epsilon(1)),' and C = ',num2str(C(1))],['
       \epsilon = ',num2str(epsilon(2)),' and C = ',num2str(C(2))],['\
       epsilon = ',num2str(epsilon(3)),' and C = ',num2str(C(3))]);
```

3.3 linearisation.m

```
1 %%% script to compare linearised and non-linear computed solutions
2 % variables
3 epsilon = 0.4;
4 nx = 501; C = 0.255; T = 1; C0 = 1;
5 plottime = 0; % setting to delay the plotting, only important for small
       amount of steps
```

```
6  beginplot = 0; % begin plot in percentage of T to skip parts
7
8  % constants
9  L = 10; g = 9.81; H = 1; w = 0.4;
10 hx = L/nx; ht = C*hx; nt = T/ht;
11 alpha = C0*hx/ht;
12 delay = plottime/nt;
13 % timestep at the end to reach exaclty T
14 htp = (nt-floor(nt))*ht; alphap = hx/htp;
15
16 % discretise x-vector
17 x = linspace(hx/2,L-hx/2,nx);
18
19 % initial conditions needed for plotting
20 h0 = H + epsilon*exp(-(x-L/2).^2/w^2)';
21 v0 = x'*0; h0v0 = h0.*v0;
22
23 % analytical solution
24 hfun = @(x,t) H + epsilon/2 * (exp(-(sqrt(g*H)*t + x - L/2).^2/w^2) +
       exp(-(-sqrt(g*H)*t + x - L/2).^2/w^2));
25 hvfun = @(x,t) epsilon*sqrt(g*H)/2 * (exp(-(-sqrt(g*H)*t + x - L/2).^2/
       w^2) - exp(-(sqrt(g*H)*t + x - L/2).^2/w^2));
26
27 % analytical initial conditions
28 hA = hfun(x,0)';
29 hvA = hvfun(x,0)';
30 vA = hvA./hA;
31
32 % timeloop for analytical
33 for it = 1:nt
34     hA = [hA,hfun(x,(it*ht))'];
35     hvA = [hvA,hvfun(x,(it*ht))'];
36     vA = [vA,hvA(:,it+1)./hA(:,it+1)];
37 end
38 % last timesteop to reach T
39 hA = [hA,hfun(x,T)'];
40 hvA = [hvA,hvfun(x,T)'];
41 vA = [vA,hvA(:,it+1)./hA(:,it+1)];
42
43 %%% compute nonlinear case
44 [h,v] = compute_system(epsilon,nx,C,T,C0);
45
46 %%% plot the data!
47 [~,SizeH] = size(h);
48 for it = floor(beginplot*(SizeH-1))+1:SizeH
49     subplot(211);
50     hold off;
51     plot(x,h(:,it),'LineWidth',1.5);
52     hold on;
53     plot(x,hA(:,it),'LineWidth',1.5);
```

```
54      axis([0,10,1- (max(h0)-1)/2,max(h0)]);
55      titlestringH = ['Height at t = ',num2str((it-1)*ht)];
56      if it == SizeH && htp ~= 0 % compute to exaclty T
57          titlestringH = ['Height at t = ',num2str((it-2)*ht + htp)];
58      end
59      title(titlestringH);
60      xlabel('x'); ylabel('h(x)');
61      subplot(212);
62      hold off;
63      plot(x,v(:,it),'LineWidth',1.5);
64      hold on;
65      plot(x,vA(:,it),'LineWidth',1.5);
66      axis([0,10,2*(1-max(h0)),2*(max(h0)-1)]);
67      xlabel('x'); ylabel('v(x)');
68      titlestringV = ['Velocity at t = ',num2str((it-1)*ht)];
69      if it == SizeH && htp ~= 0 % compute to exaclty T
70          titlestringV = ['Velocity at t = ',num2str((it-2)*ht + htp)];
71      end
72      title(titlestringV);
73
74      pause(delay);
75  end
76  % add legends
77  subplot(211);
78  legend('Computed solution','Analytical linearised solution');
79  subplot(212);
80  legend('Computed solution','Analytical linearised solution');
```