MATH0024 PDEs Homework 2

Julien Brandoit – s200710

Academic year 2024-2025

1 Von Neumann stability analysis

The leap-frog scheme of the 1D wave equation is written as:

$$\frac{u_j^{n-1} - 2u_j^n + u_j^{n+1}}{\Delta t^2} = c^2 \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{h^2} \tag{1}$$

where the notation u_j^n stands for the approximate numerical solution at given discrete temporal and spatial points (n and j are integers):

$$u_j^n \approx u(x = jh, t = n\Delta t)$$

Given the scheme described by Eq. (1), the update equation for the explicit scheme can be written as:

$$u_j^{n+1} = \alpha^2 \left(u_{j-1}^n - 2u_j^n + u_{j+1}^n\right) + 2u_j^n - u_j^{n-1}, \quad \alpha^2 = \frac{c^2 \Delta t^2}{h^2} \tag{2}$$

The equation being linear, a similar update equation can be written for the error introduced by the discretization:

$$\varepsilon_j^n \approx \varepsilon(x = jh, t = n\Delta t)$$

A Von Neumann stability analysis can be performed by studying the behavior of a Fourier component of the error:

$$\varepsilon_j^n = \sum_\xi \hat{\varepsilon}_\xi^n \exp(i\xi jh) \xrightarrow{\text{Fourier Component}} \varepsilon_j^n(\xi) = \gamma^n(\xi) \exp(ij\xi h)$$

Where ξ is the wave number associated with the different Fourier modes and $\gamma(\xi) \in \mathbb{C}$ is the *amplification factor*. A Von Neumann stability analysis focuses on this amplification factor and seeks to express one or more sufficient conditions for it to satisfy:

$$\underline{\texttt{Von Neumann Stability Criterion :}} \quad |\gamma(\xi)| \leq 1 \Leftrightarrow |\gamma(\xi)|^2 \leq 1, \quad \forall \xi \tag{3}$$

For a linear update equation, this condition is satisfied by each Fourier mode separately, so that we have:

$$\gamma^{n+1}(\xi) e^{(ij\xi h)} = \alpha^2 \gamma^n(\xi) \left[e^{(i(j-1)\xi h)} - 2e^{(ij\xi h)} + e^{(i(j+1)\xi h)}\right] + 2\gamma^n(\xi) e^{(ij\xi h)} - \gamma^{n-1}(\xi) e^{(ij\xi h)} \tag{4}$$

$$\gamma^1(\xi) = \alpha^2 \gamma^0(\xi) \left[e^{(-i\xi h)} - 2 + e^{(i\xi h)}\right] + 2\gamma^0(\xi) - \gamma^{-1}(\xi) \tag{5}$$

$$\gamma^2(\xi) = \alpha^2 \gamma(\xi) \left[2\cos(\xi h) - 2\right] + 2\gamma(\xi) - 1 \tag{6}$$

Eq. (6) allows us to find an expression for $\gamma(\xi)$, and hence $|\gamma(\xi)|$, by solving for the roots of the quadratic polynomial:

$$P(\gamma) = \gamma^2 - 2\left[\alpha^2 \cos(\xi h) - \alpha^2 + 1\right]\gamma + 1 \tag{7}$$

The two roots of this quadratic polynomial are:

$$\gamma_{1,2} = 1 + \alpha^2 \left[\cos(\xi h) - 1\right] \pm |\alpha| \sqrt{\alpha^2 \left[\cos(\xi h) - 1\right]^2 + 2\left[\cos(\xi h) - 1\right]} \tag{8}$$

The nature of these roots (distinct and real, real and identical - multiplicity 2 -, or complex conjugates) depends on the value of $\lambda = \alpha^2 \left[\cos(\xi h) - 1\right]^2 + 2\left[\cos(\xi h) - 1\right]$. The nature of the roots is important for correctly expressing their modulus and verifying the Von Neumann criterion (Eq. 3).

Case 1, $\lambda < 0$:

For negative values of λ, the square root in Eq. (8) is purely imaginary: $i\sqrt{-\lambda}$, and the two roots are complex conjugates of equal modulus:

$$|\gamma_1| = |\gamma_2| = |\gamma|$$

Thus, we have:

$$|\gamma|^2 = 1 - 2\alpha^2 \left[\cos(\xi h) - 1\right]$$

So the Von Neumann criterion becomes:

$$-2\alpha^2 \left[\cos(\xi h) - 1\right] \geq 0$$

which is satisfied for all values of ξ and α such that $\lambda < 0$.

$\implies$ The scheme is unconditionally stable in the Von Neumann sense for $\lambda < 0$.

Case 2, $\lambda = 0$:

For $\lambda = 0$, the two roots coincide into a real root with multiplicity 2:

$$\gamma_1 = \gamma_2 = \gamma$$

We have:

$$\begin{cases} \gamma = 1 + \alpha^2 \left[\cos(\xi h) - 1\right] \\ \lambda = 0 = \alpha^2 \left[\cos(\xi h) - 1\right]^2 + 2\left[\cos(\xi h) - 1\right] \end{cases} \tag{9}$$

$$\implies \begin{cases} \gamma = -1 \\ \alpha^2 \left[\cos(\xi h) - 1\right] = -2 \end{cases} \tag{10}$$

Thus, the Von Neumann criterion is satisfied for all values of α and ξ such that $\lambda = 0$.

$\implies$ The scheme is unconditionally stable in the Von Neumann sense for $\lambda = 0$.

Case 3, $\lambda > 0$:

For positive values of λ, the roots are real and distinct. Their moduli are different.

$$\gamma_1 \neq \gamma_2, \quad \gamma_1, \gamma_2 \in \mathbb{R}$$

The Von Neumann criterion can be written as:

$$-1 \leq 1 + \alpha^2 \left[\cos(\xi h) - 1\right] \pm |\alpha| \sqrt{\alpha^2 \left[\cos(\xi h) - 1\right]^2 + 2\left[\cos(\xi h) - 1\right]} \leq 1 \tag{11}$$

Both inequalities must be satisfied. The left inequality can be written as:

$$-2 - \alpha^2 \left[\cos(\xi h) - 1\right] \leq \pm |\alpha| \sqrt{\alpha^2 \left[\cos(\xi h) - 1\right]^2 + 2\left[\cos(\xi h) - 1\right]} \tag{12}$$

In particular:

$$-2 - \alpha^2 \left[\cos(\xi h) - 1\right] \leq -|\alpha| \sqrt{\alpha^2 \left[\cos(\xi h) - 1\right]^2 + 2\left[\cos(\xi h) - 1\right]} \tag{13}$$

This cannot be satisfied for $\lambda > 0$. Indeed:

$$\lambda > 0 \tag{14}$$
$$\alpha^2 \left[\cos(\xi h) - 1\right]^2 + 2\left[\cos(\xi h) - 1\right] > 0 \tag{15}$$
$$\alpha^2 \left[\cos(\xi h) - 1\right]^2 > -2\left[\cos(\xi h) - 1\right] \tag{16}$$
$$\alpha^2 \left[\cos(\xi h) - 1\right] < -2, \quad \text{since } \left[\cos(\xi h) - 1\right] \leq 0 \tag{17}$$

Thus, the left side of Eq. 13 is always positive, while the right side is always negative. Therefore, no condition on α can satisfy the Von Neumann criterion $\forall \xi$ for $\lambda > 0$.

$\implies$ The scheme is unconditionally unstable in the Von Neumann sense for $\lambda > 0$.

Final Criterion :

Combining all three cases, we find that the scheme is stable if $\lambda \leq 0$ for all ξ. Finally, we express:

$$\lambda \leq 0 \tag{18}$$
$$\alpha^2 \left[\cos(\xi h) - 1\right]^2 + 2\left[\cos(\xi h) - 1\right] \leq 0 \tag{19}$$
$$\alpha^2 \leq \frac{2}{1 - \cos(\xi h)} \tag{20}$$
$$c^2 \frac{\Delta t^2}{h^2} \leq 1 \tag{21}$$

The final criterion is obtained by observing that $\frac{2}{1-\cos(\xi h)} \geq 1, \forall \xi$.

$\implies$ The scheme (1) is stable in the Von Neumann sense provided that $c^2 \dfrac{\Delta t^2}{h^2} \leq 1$.

2 Numerical Implementation

2.1 FEM and Classical Leap-Frog Time-Stepping

The following boundary-value problem involving the 1D wave equation can be tackled using numerical approaches:

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} - c^2 \dfrac{\partial^2 u}{\partial x^2} = 0, & (x,t) \in]0,L[\times]0,T[\\ u(x,t=0) = u_0(x), & (x,t) \in]0,L[\times\{0\} \\ \dfrac{\partial u}{\partial t}(x,t=0) = v_0(x), & (x,t) \in]0,L[\times\{0\} \\ \dfrac{\partial u}{\partial x}(x,t) = 0, & (x,t) \in \{0,L\}\times]0,T[\end{cases} \tag{22}$$

We will combine a spatial discretization using the 1D finite element method (FEM) with the *Leap-Frog* scheme.

The spatial discretization requires constructing a Galerkin formulation of the problem and choosing basis functions $\{\phi_i\}_{i=1}^N$, with N being the number of discretization nodes. This discretization allows us to formulate a semi-discrete version of the problem:

$$\begin{cases} [M]\dfrac{d^2\boldsymbol{u}}{dt^2}(t) + [K]\boldsymbol{u}(t) = 0, & t \in]0,T[\\ \boldsymbol{u}(t=0) = \boldsymbol{u}_0 \\ \dfrac{d\boldsymbol{u}}{dt}(t=0) = \boldsymbol{v}_0 \end{cases} \tag{23}$$

$$\boldsymbol{u}(t) = [u_1(t), u_2(t), \ldots, u_N(t)] \in \mathbb{R} \xrightarrow{\texttt{Finite Element Method}} u(x,t) = \sum_{k=1}^{N} u_k(t)\phi_k(x) \tag{24}$$

Where the unknowns of the problem are now the values at the nodes over time, $\boldsymbol{u}(t)$. The matrices $[M]$ and $[K]$ are respectively the mass matrix and the stiffness matrix. These matrices are constructed through an assembly procedure from the element-level matrices $[M]^e$ and $[K]^e$.

In the case of 1D elements characterized by two degrees of freedom $u_1^e(t)$ and $u_2^e(t)$ and associated with linear basis functions ϕ_i, it is possible to derive the exact expressions for the element-level matrices during the construction of the Galerkin form. We have:

$$[M] \in \mathbb{R}^{n\times n}, [K] \in \mathbb{R}^{n\times n}, [M]^e \in \mathbb{R}^{2\times 2}, [K]^e \in \mathbb{R}^{2\times 2} \tag{25}$$

$$\begin{cases} \{[M]_i^e\}_{i=1}^N \\ \{[K]_i^e\}_{i=1}^N \end{cases} \xrightarrow{\text{Assembly procedure}} \begin{cases} [M] \\ [K] \end{cases} \tag{26}$$

$$K_{i,j}^e = \int_e c^2 \frac{d\phi_i}{dx}(x)\frac{d\phi_j}{dx}(x)dx = \begin{cases} \frac{c^2}{h_e}, & \text{if } i = j \\ \frac{-c^2}{h_e}, & \text{if } i \neq j \end{cases} \tag{27}$$

$$M_{i,j}^e = \int_e \phi_i(x)\phi_j(x)dx = \begin{cases} \frac{h_e}{3}, & \text{if } i = j \\ \frac{h_e}{6}, & \text{if } i \neq j \end{cases} \tag{28}$$

The time discretization via a second-order *Leap-Frog* scheme involves approximating the time derivative as follows:

$$\begin{cases} \boldsymbol{u}(t) & \approx \boldsymbol{u}(n\Delta t) = \boldsymbol{u}_n \\ \dfrac{d\boldsymbol{u}}{dt}(t) & \approx \dfrac{\boldsymbol{u}_{n+1} - 2\boldsymbol{u}_n + \boldsymbol{u}_{n-1}}{\Delta t^2} \end{cases} \tag{29}$$

Thus, the fully discretized formulation (in time and space) is:

$$[M]\frac{\boldsymbol{u}_{n+1} - 2\boldsymbol{u}_n + \boldsymbol{u}_{n-1}}{\Delta t^2} + [K]\boldsymbol{u}_n = 0 \tag{30}$$

This scheme is implicit and requires more computational resources than a slightly different version of (30) obtained after applying the *mass-lumping* method. The Mass-Lumping approximates the non-diagonal matrix $[M]$ by $[\bar{M}]$, a diagonalized version obtained by aggregating the columns:

$$[\bar{M}] = \delta_{ij} \sum_{k=1}^{N} M_{ik} \tag{31}$$

$$\xrightarrow{\text{Mass-Lumping approximation}} \quad [\bar{M}]\frac{\boldsymbol{u}_{n+1} - 2\boldsymbol{u}_n + \boldsymbol{u}_{n-1}}{\Delta t^2} + [K]\boldsymbol{u}_n = 0 \tag{32}$$

Under the Mass-Lumping approximation, the scheme described by (32) is explicit and it can be shown to be equivalent to solving:

$$\frac{\boldsymbol{z}_{n+1} - 2\boldsymbol{z}_n + \boldsymbol{z}_{n-1}}{\Delta t^2} + [A]\boldsymbol{z}_n = 0, \quad [A] = [\bar{M}]^{\frac{-1}{2}}[K][\bar{M}]^{\frac{-1}{2}} \tag{33}$$

$$\boldsymbol{z}_n = [\bar{M}]^{\frac{1}{2}}\boldsymbol{u}_n \Longleftrightarrow \boldsymbol{u}_n = [\bar{M}]^{\frac{-1}{2}}\boldsymbol{z}_n \tag{34}$$

This time discretization requires transforming the initial conditions $u_0(x)$ and $v_0(x)$ into discrete conditions $\boldsymbol{u}_0$ and $\boldsymbol{u}_{-1}$.

2.1.1 Specific Application

In order to use this numerical method for a specific application, several aspects of the problem need to be detailed. We consider the problem described by (22) with the following parameters:

$$L = 4, \quad T = 9, \quad c = -1$$

And we want the simulation – at least initially, before boundary effects influence the solution – to correspond to:

$$u(x,t) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x - ct - \frac{L}{2})^2}{2\sigma^2}\right), \quad \sigma = 0.4 \tag{35}$$

This corresponds to a Gaussian deformation starting at the center of the domain and moving to the left with constant velocity c.

The initial condition $u_0 = u(x, t = 0)$ is straightforward to determine and transform into a discrete condition $\boldsymbol{u}_0$. We have:

$$u(x, t = 0) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x - \frac{L}{2})^2}{2\sigma^2}\right) \tag{36}$$

And we construct the vector $\boldsymbol{u}_0$ by directly evaluating (36) at the spatial positions assigned to each node $\{u_i(t) \approx u(x = x_i, t)\}_{i=1}^{N}$.

The vector $\boldsymbol{u}_{-1}$ is more complex to construct, as it requires transforming an initial condition defined on a derivative ($u_t(x, t = 0) = v_0(x)$) into a discrete condition on the solution values.

Assuming that Δt is small enough, we can approximate the value of $\boldsymbol{u}_{-1}$ by shifting the value of $u_0(x)$ with a constant velocity equivalent to the instantaneous velocity $v_0(x) = c$. We have:

$$u(x, t = -\Delta t) \approx u(x + v_0(x)\Delta t, t = 0) \tag{37}$$

We can then construct $\boldsymbol{u}_{-1}$ by directly evaluating (36) at the spatial position $x_i + v_0(x_i)\Delta t$ for each node $\{u_i(t) \approx u(x = x_i, t)\}_{i=1}^{N}$. This method is equivalent to evaluating the exact solution at the previous time step in the case of a constant initial velocity. In cases where the initial velocity is not constant, the method is an approximation that does not require prior knowledge of the exact solution.

Figure 1 illustrates the approximations $\boldsymbol{u}_0$ and $\boldsymbol{u}_{-1}$ for two different spatial discretizations. It is clear that $\boldsymbol{u}_{-1}$ is constructed so that $\boldsymbol{u}_0$ corresponds to the discretization of (36) with a constant shift proportional to $c\Delta t$.

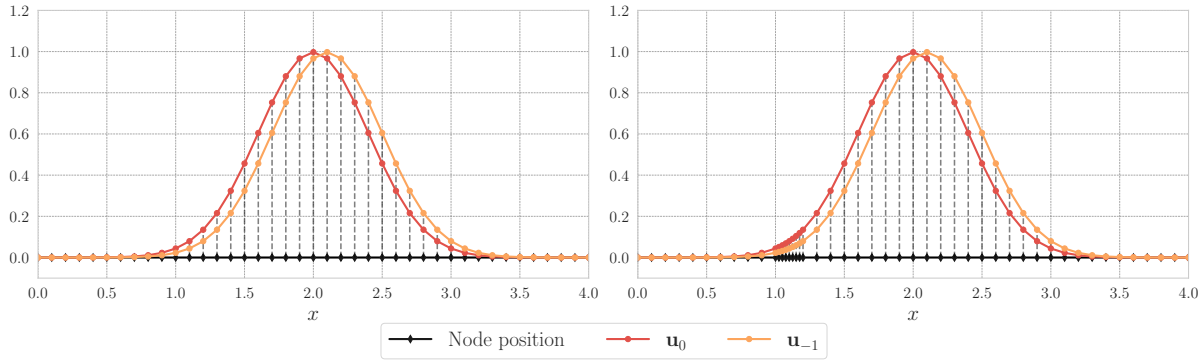

Figure 1: Initial discretization showing u_0 and u_{-1} for two meshes. Left: a regular mesh with element size h_{coarse}. Right: a refined mesh with additional resolution in the segment $[1, 1 + 2h_{\text{coarse}}]$. A relatively large value of Δt is used for visualization purposes. $h_{\text{coarse}} = 0.1, p = 4 \implies h_{\text{fine}} = 0.025$.

Numerical Study of the Stability Criterion

We can numerically study the stability criterion (21) derived in Section 1 of this work. It is proposed to work with two different meshes (spatial discretizations). The first mesh corresponds to a regular division of space into elements of length $h_{\text{coarse}} = 0.1$ (red mesh in Fig. 2). The second discretization corresponds to refining the first mesh on the segment $[1, 1 + 2h_{\text{coarse}}]$ with the two elements of this segment subdivided into $p = 4$ smaller elements of length $h_{\text{fine}} = \frac{h_{\text{coarse}}}{p}$ (blue mesh in Fig. 2).

The stability of the solution is analyzed graphically. We consider a solution to be stable if it does not appear to have 'exploded' during the simulation on $[0, T]$. In Fig. 3, we examine the solution for the first mesh (regular red mesh) for three values of the time

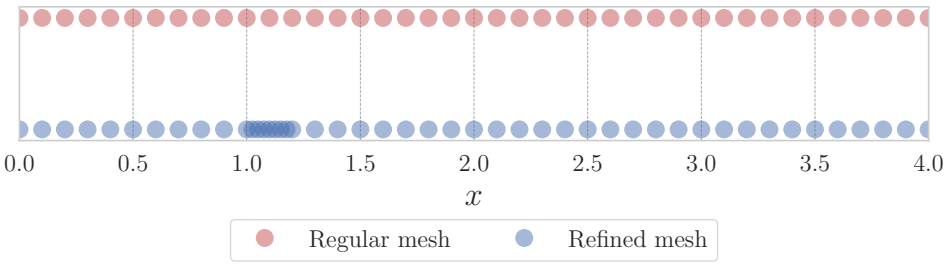

Figure 2: Illustration of the two spatial discretizations. Top (red): regular mesh with element size $h_{\text{coarse}} = 0.1$. Bottom (blue): refined mesh with a refinement factor of $p = 4$ on the segment $[1, 1 + 2h_{\text{coarse}}]$.

step: $\Delta t_{\text{reg.}} \in \{0.95\Delta t_{\text{VN},1}, \Delta t_{\text{VN},1}, 1.05\Delta t_{\text{VN},1}\}$ with

$$\Delta t_{\text{VN},1} = \frac{h_{\text{coarse}}}{|c|} = 0.1$$

In Fig. 4, we examine the solution for the second mesh (refined blue mesh) for three values of the time step: $\Delta t_{\text{ref.}} \in \{0.95\Delta t_{\text{VN},2}, \Delta t_{\text{VN},2}, 1.05\Delta t_{\text{VN},2}\}$ with

$$\Delta t_{\text{VN},2} = \frac{h_{\text{fine}}}{|c|} = 0.025$$

It is clearly observed that for both discretizations, the time steps $0.95\Delta t_{\text{VN,i}}$ and $\Delta t_{\text{VN,i}}$ yield stable solutions, whereas the simulations corresponding to $1.05\Delta t_{\text{VN,i}}$ are unstable. The numerical observations thus align well with the criterion derived analytically using the Von Neumann method. It is observed that, in the case of the irregular mesh, instability appears in the refined part of the mesh before propagating. This observation is consistent with the fact that, in the rest of the mesh, the Von Neumann criterion is respected.

These two numerical experiments highlight the strength of Von Neumann analyses, which yield practical results quite simply. It should be noted that Von Neumann stability is not always equivalent to absolute stability, but that the latter is much more complex to derive. The two experiments also highlight an important limitation of classical time-marching methods: stability is determined by the smallest element size of the discretization, imposing a very fine temporal discretization over the entire domain. To overcome this limitation, we implemented the local-time-stepping methods studied in Section 2.2 of this work.

2.1.2 Details on the Practical Numerical Implementation

This section details the techniques used to implement, in *Python*, the method presented in Section 2.1.

1D Finite Element Method

- The FEM implementation used is based on Assignment 1. In practice, it relies on an object-oriented implementation of a 1D FEM method designed to be quite general. The integrals in Eqs. (27, 28) are computed using Gaussian quadrature. The number of points

used is $n = 2$ for $[M]_e$ and $n = 1$ for $[K]_e$. This ensures exact integration [2] for our linear polynomials with constant derivatives.

- The Mass-Lumping technique is directly applied at the element level and is also considered during assembly. Thus, the matrix $[M]$ is never assembled; instead, $[\bar{M}]$ is directly constructed.

- *Sparse* matrices are used in `csr` format from the `Scipy` library [3]. The matrix $[\bar{M}]$ is assembled in the form of a vector (the diagonal) before being translated in a `csr` format.

Classical Leap-Frog

- The matrices $[A]$ and $[\bar{M}]^{-\frac{1}{2}}$ are constructed only once since they remain constant. Only these two matrices are stored in memory to optimize memory usage. A *sparse* `csr` format is used for $[A]$ and $[\bar{M}]^{-\frac{1}{2}}$. Moreover, to further optimize memory usage, the operations to obtain $[A]$ and $[\bar{M}]^{-\frac{1}{2}}$ from $[\bar{M}]$ and $[K]$ are performed *in-place*.

  For example, when computing $[A] = [\bar{M}]^{-\frac{1}{2}}[K][\bar{M}]^{-\frac{1}{2}}$, the diagonal nature of $[\bar{M}]^{-\frac{1}{2}}$ is leveraged to scale $[K]$ directly (in-place) into $[A]$:

  $$[A_{i,j}] = m_i K_{i,j} m_j$$

  This approach improves both memory usage and performance, as computing $m_i K_{i,j} m_j$ for the non-zero entries of $[K]$ is significantly more efficient than performing two full matrix multiplications.

  One can check the implementation in the `scale_in_place_csr_matrix` function. Additionally, an example of efficient memory usage is that all computations involving $[\bar{M}]^{\frac{1}{2}}$ are performed first. Then, $[\bar{M}]^{\frac{1}{2}}$ is transformed *in-place* into $[\bar{M}]^{-\frac{1}{2}}$.

- During time-stepping, only the time steps that require an output of the solution u perform the transformation

  $$\boldsymbol{u}_n = [\bar{M}]^{-\frac{1}{2}} \boldsymbol{z}_n.$$

  For other time steps, the problem is handled solely in terms of $\boldsymbol{z}$. This avoids unnecessary operations for simulations where the entire set of time steps $t_n = n\Delta t$ is not needed.

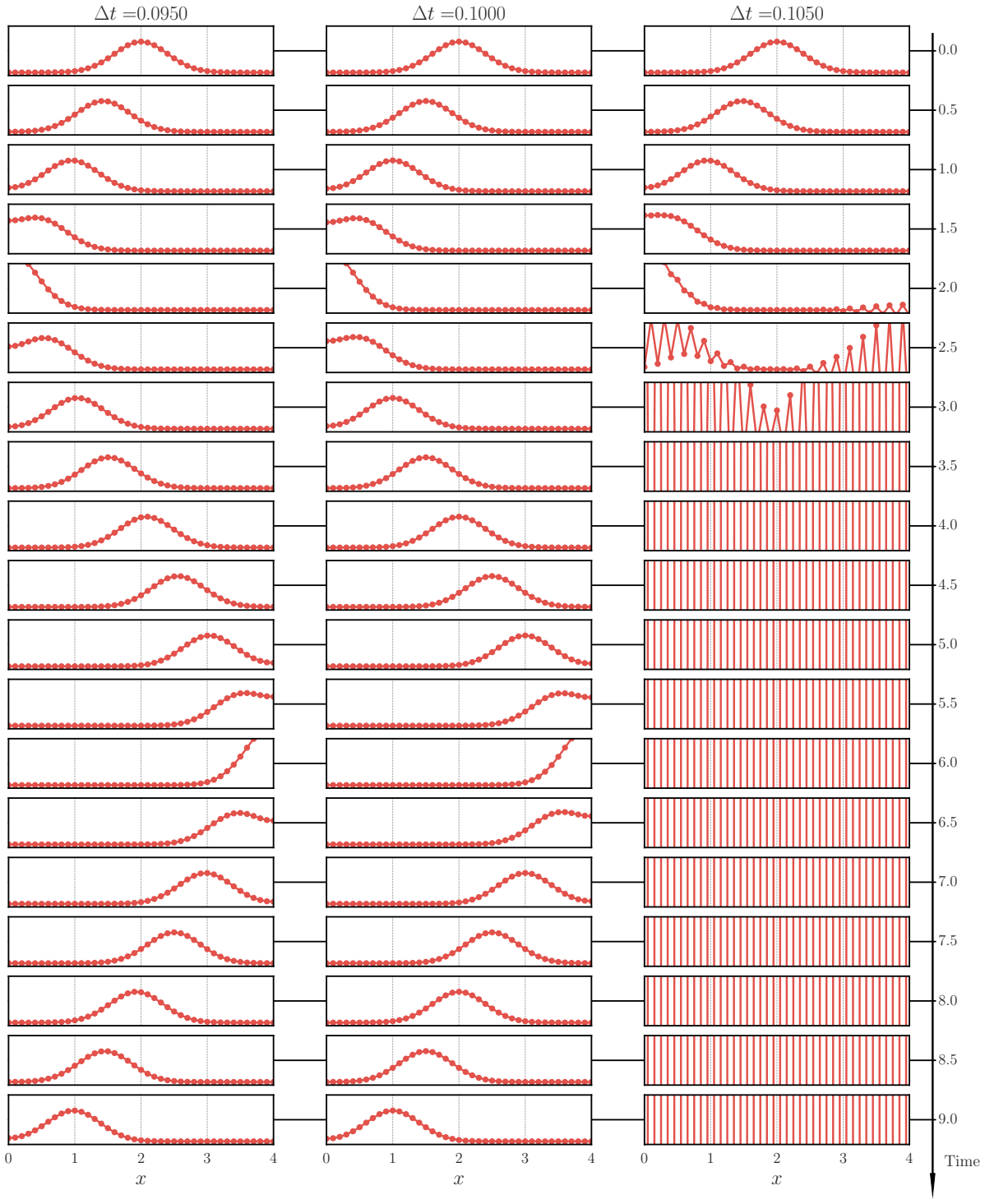

Figure 3: Time evolution of the solution for the regular mesh (red) with $h_{\text{coarse}} = 0.1$. Simulations are shown for three time steps: $\Delta t_{\text{reg.}} \in \{0.95\Delta t_{\text{VN},1}, \Delta t_{\text{VN},1}, 1.05\Delta t_{\text{VN},1}\}$. Stable solutions are observed for $\Delta t \leq \Delta t_{\text{VN},1}$, while instability occurs for $\Delta t > \Delta t_{\text{VN},1}$.

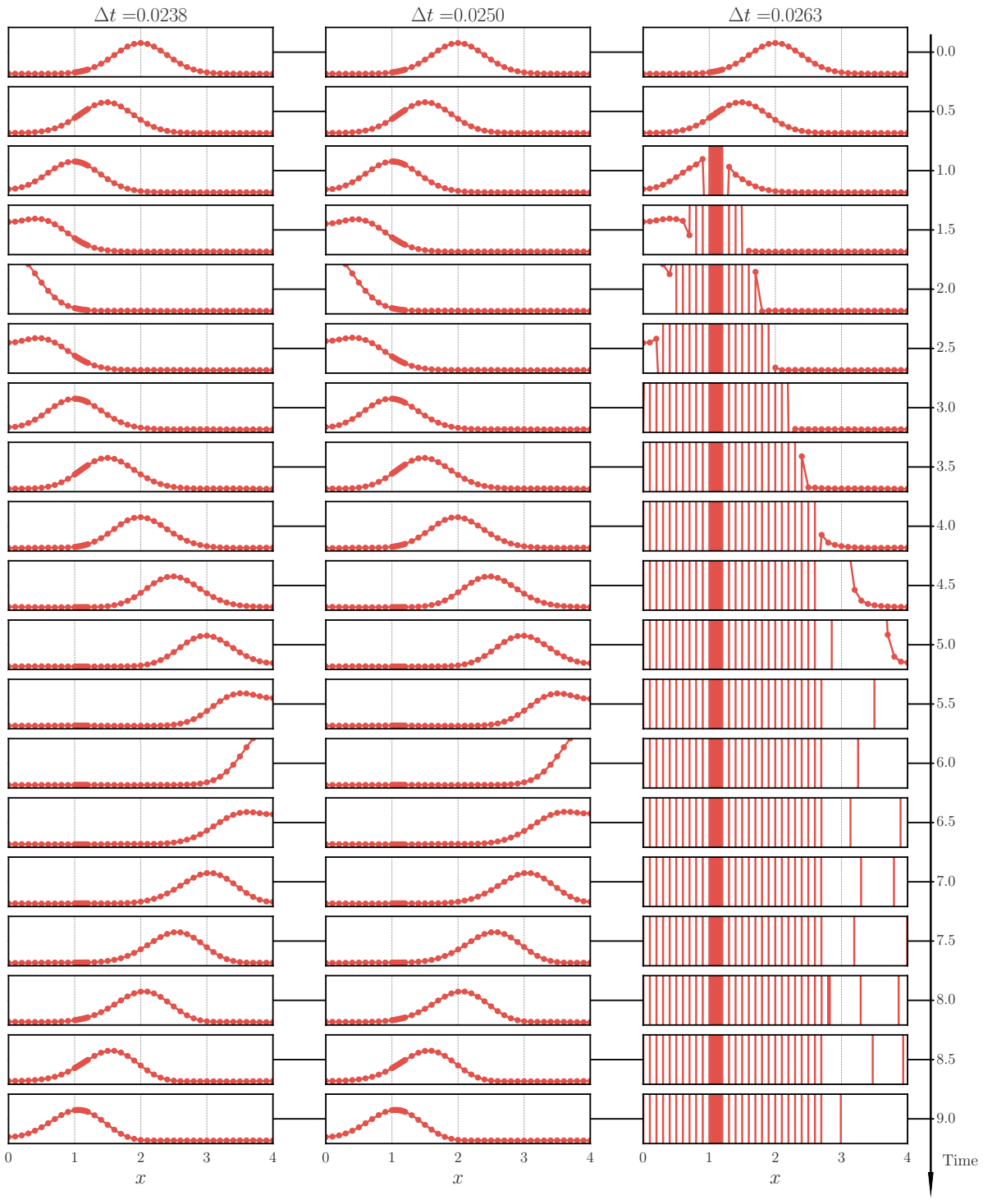

Figure 4: Time evolution of the solution for the refined mesh (blue) with additional resolution on $[1, 1 + 2h_{\text{coarse}}]$. Simulations are shown for three time steps: $\Delta t_{\text{ref.}} \in \{0.95\Delta t_{\text{VN},2}, \Delta t_{\text{VN},2}, 1.05\Delta t_{\text{VN},2}\}$. Instability is first observed in the refined region for $\Delta t > \Delta t_{\text{VN},2}$. $h_{\text{coarse}} = 0.1, p = 4 \implies h_{\text{fine}} = 0.025$.

2.2 FEM and Local-Time-Stepping Leap-Frog

In order to overcome the limitations presented in Section 2.1 – that is, the dependency of the stability criterion on the smallest element size, which imposes very fine time discretization over the entire domain – this section explores a method called *Local-Time-Stepping* (LTS) applied to a Leap-Frog scheme. The theoretical details and justifications for the method can be found in [1].

The remainder of this section provides a summary of the method, followed by a discussion on the numerical implementation carried out in practice. Finally, a numerical analysis of the implementation is presented.

2.2.1 Method Summary

The LTS method involves dividing a discretization into *coarse* and *fine* elements. The coarse elements are larger than the fine elements. The fine and coarse nodes are handled separately: for each time step Δt, p sub-steps of size $\frac{\Delta \tau}{p}$ are performed, during which the coarse nodes are considered constant, while the fine nodes are updated. This method is advantageous compared to a time discretization of $\frac{\Delta \tau}{p}$ via a classical scheme like the one presented in Section 2.1, because during the sub-steps, operations are performed involving only a part of the domain, which, if implemented wisely, allows for better stability while avoiding excessive computational cost.

Mathematically, we distinguish the unknowns associated with coarse nodes $\boldsymbol{z}_{\mathrm{C}}$ and those associated with fine nodes $\boldsymbol{z}_{\mathrm{F}}$. This distinction holds for any time step $t_n = n\Delta t$, so we do not explicitly indicate the time dependence for clarity.

$$\boldsymbol{z}_n = \begin{Bmatrix} \boldsymbol{z}_{\mathrm{C}} \in \mathbb{R}^{N_{\mathrm{C}}} \\ \boldsymbol{z}_{\mathrm{F}} \in \mathbb{R}^{N_{\mathrm{F}}} \end{Bmatrix} \in \mathbb{R}^{N}, \quad N_{\mathrm{F}} + N_{\mathrm{C}} = N \tag{38}$$

Figure 5 illustrates this separation on the refined mesh presented in Section 2.1.

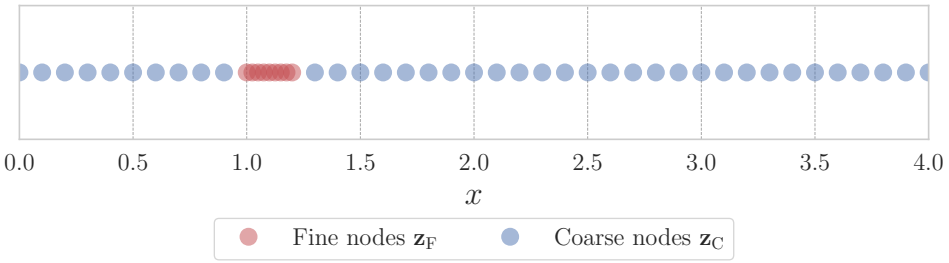

Figure 5: Illustration of the separation of coarse and fine nodes in the mesh. The coarse nodes are represented in blue, while the fine nodes, corresponding to the smaller and finer-resolution elements, are shown in red. This separation is used in the Local-Time-Stepping (LTS) method, where the coarse nodes are treated differently from the fine nodes during the time-stepping process, improving stability and computational efficiency. The refined mesh is based on the same discretization presented in Section 2.1.

From this separation, we can construct the following submatrices:

$$[A_{\mathrm{FF}}] \in \mathbb{R}^{N_{\mathrm{F}} \times N_{\mathrm{F}}}, \quad [A_{\mathrm{CF}}] \in \mathbb{R}^{N_{\mathrm{C}} \times N_{\mathrm{F}}}, \quad [A_{\mathrm{FC}}] \in \mathbb{R}^{N_{\mathrm{F}} \times N_{\mathrm{C}}}, \quad [A_{\mathrm{CC}}] \in \mathbb{R}^{N_{\mathrm{C}} \times N_{\mathrm{C}}} \tag{39}$$

as well as the "mask" matrix $[P]$, which is a diagonal projection matrix encoding the coarse/fine separation:

$$\boldsymbol{z} = ([I] - [P])\boldsymbol{z} + [P]\boldsymbol{z} = \boldsymbol{z}^{\text{coarse}} + \boldsymbol{z}^{\text{fine}} \tag{40}$$

$$\begin{Bmatrix} \boldsymbol{z}_{\text{C}} \\ \boldsymbol{z}_{\text{F}} \end{Bmatrix} = \begin{bmatrix} [I] & [0] \\ [0] & [0] \end{bmatrix} \boldsymbol{z} + \begin{bmatrix} [0] & [0] \\ [0] & [I] \end{bmatrix} \boldsymbol{z} = \begin{Bmatrix} \boldsymbol{z}_{\text{C}} \\ \{0\} \end{Bmatrix} + \begin{Bmatrix} \{0\} \\ \boldsymbol{z}_{\text{F}} \end{Bmatrix} \tag{41}$$

During a new time step, we want to update $\boldsymbol{z}_{n+1}$ based on $\boldsymbol{z}_n$ and $\boldsymbol{z}_{n-1}$. The process is as follows:

1. Apply $[A]$ to the coarse unknowns. This step is costly because it involves the largest part of the mesh. However, this operation is only performed once per time step. We compute:
$$\boldsymbol{w} = [A]\left([I] - [P]\right)\boldsymbol{z_n} \tag{42}$$

2. Perform p sub-steps of time update with step size $\Delta\tau = \frac{\Delta t}{p}$. We use the following initial conditions:
$$\tilde{\boldsymbol{z}}_{0/p} = \boldsymbol{z}_n \tag{43}$$
$$\tilde{\boldsymbol{z}}_{-1/p} = \tilde{\boldsymbol{z}}_{1/p} = \tilde{\boldsymbol{z}}_{0/p} - \frac{1}{2}\left(\frac{\Delta t}{p}\right)^2 (\boldsymbol{w} + [A][P]\tilde{\boldsymbol{z}}_{0/p}) \tag{44}$$
And the update scheme:
$$\tilde{\boldsymbol{z}}_{(m+1)/p} = 2\tilde{\boldsymbol{z}}_{m/p} - \tilde{\boldsymbol{z}}_{(m-1)/p} - \frac{1}{2}\left(\frac{\Delta t}{p}\right)^2 (\boldsymbol{w} + [A][P]\tilde{\boldsymbol{z}}_{m/p}), \quad m \in \{1, \dots, p-1\} \tag{45}$$
For theoretical justifications, refer to [1]. The computation of (45) is performed several times per time iteration. However, this operation is inexpensive because it involves only a small part of the numerical domain.

3. Finally, we compute the approximation of the solution over the entire domain via:
$$\boldsymbol{z}_{n+1} = -\boldsymbol{z}_{n-1} + 2\tilde{\boldsymbol{z}}_{p/p} \tag{46}$$

In practice, the numerical implementation does not carry out the computations as described above for optimization purposes, primarily because the matrices $[A][P]$ and $[A]\left([I] - [P]\right)$ only represent parts of $[A]$. Instead, operations are directly based on $[A_{FF}], [A_{CF}], [A_{FC}]$, and $[A_{CC}]$ as well as the explicit separation in (38). Details of the practical implementation can be found in Section 2.2.3.

2.2.2 Specific Application

The specific application is the same as the one presented in Section 2.1.1. As for the initial conditions, there is no difference with the classical method. Figure 1 (right) thus also represents the initial approximation of the solution, i.e., $\boldsymbol{u}_0$ and $\boldsymbol{u}_{-1}$ for the scheme based on the LTS method.

Only the update procedure differs between this Section and the previous one. Thus, we can directly study the numerical stability of the LTS method.

Numerical Study of the Stability Criterion

We study stability similarly to Section 2.1. We compare 6 simulations, defined around $\Delta t_{\text{VN},1}$ and $\Delta t_{\text{VN},2}$. The advantage of the LTS method is that it provides a method whose stability is determined by h_{coarse} at the cost of sub-time steps on part of the domain, which is less costly than the classical method whose stability is determined by h_{fine}. The 6 simulations presented are as follows:

$$\Delta t_{\text{LTS}} \in \{0.95\Delta t_{\text{VN},1}, \Delta t_{\text{VN},1}, 1.05\Delta t_{\text{VN},1}\} \cup \{0.95\Delta t_{\text{VN},2}, \Delta t_{\text{VN},2}, 1.05\Delta t_{\text{VN},2}\}$$

Figure 6 presents the cases $\Delta t_{\text{LTS}} \in \{0.95\Delta t_{\text{VN},1}, \Delta t_{\text{VN},1}, 1.05\Delta t_{\text{VN},1}\}$, and Figure 7 presents the cases $\Delta t_{\text{LTS}} \in \{0.95\Delta t_{\text{VN},2}, \Delta t_{\text{VN},2}, 1.05\Delta t_{\text{VN},2}\}$.

It is observed that the goal is achieved because, despite the refined mesh part, the stability criterion is no longer as strict (comparison can be made here between Figures 4 and 7). The LTS method improved the stability of the simulation. It is also observed that it is indeed h_{coarse} that determines the numerical stability, as shown in Figure 6.

2.2.3 Details on the Practical Numerical Implementation

This section aims to explain in more detail how the LTS method is implemented in practice. In particular, the equations from Section 2.2.1 are not used as presented.

The comments made in Section 2.1.2 also apply to this implementation. However, note that $[A]$ is not stored in memory but rather separated into $[A_{\text{FF}}], [A_{\text{CF}}], [A_{\text{FC}}]$ and $[A_{\text{CC}}]$, which are stored in sparse `csr` format in memory.

- The matrix $[P] \in \{0,1\}^{N\times N}$ is never constructed or used. In practice, a vector $\boldsymbol{p} \in \{0,1\}^N$ is constructed alongside the mesh, which is then translated into two index vectors: $\boldsymbol{p}_{\text{C}}$ (`coarse_idx` in the code) and $\boldsymbol{p}_{\text{F}}$ (`fine_idx` in the code). These vectors are used during the reconstruction of $\boldsymbol{u}_n$ to correctly order the two vectors $\boldsymbol{z}_{\text{C}}$ and $\boldsymbol{z}_{\text{F}}$ into $\boldsymbol{z}_n$.

- Just as $\boldsymbol{u}_n$ is only constructed when an output of the solution is required, the vector $\boldsymbol{z}_n$ is only constructed during these steps. For the calculations, we explicitly separate it into $\boldsymbol{z}_{\text{C}}$ and $\boldsymbol{z}_{\text{F}}$. We proceed similarly for $\boldsymbol{w}$, explicitly separated into $\boldsymbol{w}_{\text{C}}$ and $\boldsymbol{w}_{\text{F}}$, and for $\tilde{\boldsymbol{z}}$, separated into $\tilde{\boldsymbol{z}}_{\text{C}}$ and $\tilde{\boldsymbol{z}}_{\text{F}}$.

- The benefit of all these explicit separations lies in how we respectively perform the following operations:

$$[A][P]\tilde{\boldsymbol{z}} = \begin{Bmatrix} [A_{\text{CF}}]\{\tilde{\boldsymbol{z}}_{\text{F}}\} \\ [A_{\text{FF}}]\{\tilde{\boldsymbol{z}}_{\text{F}}\} \end{Bmatrix}, \quad [A]([I]-[P])\boldsymbol{z} = \begin{Bmatrix} [A_{\text{CC}}]\{\boldsymbol{z}_{\text{C}}\} \\ [A_{\text{FC}}]\{\boldsymbol{z}_{\text{C}}\} \end{Bmatrix} \tag{47}$$

Thus, in practice:

- Eq. (42):

$$\boldsymbol{w} = [A]\left([I]-[P]\right)\boldsymbol{z_n} \xrightarrow{\text{In practice}} \begin{Bmatrix} \{\boldsymbol{w}_{\text{C}}\} \\ \{\boldsymbol{w}_{\text{F}}\} \end{Bmatrix} = \begin{Bmatrix} [A_{\text{CC}}]\{\boldsymbol{z}_{\text{C}}\} \\ [A_{\text{FC}}]\{\boldsymbol{z}_{\text{C}}\} \end{Bmatrix} \tag{48}$$

- Eq. (43):

$$\tilde{\boldsymbol{z}}_{0/p} = \boldsymbol{z_n} \xrightarrow{\text{In practice}} \begin{Bmatrix} \{\tilde{\boldsymbol{z}}_{0/p,\text{C}}\} \\ \{\tilde{\boldsymbol{z}}_{0/p,\text{F}}\} \end{Bmatrix} = \begin{Bmatrix} \{\boldsymbol{z}_{\text{C}}\} \\ \{\boldsymbol{z}_{\text{F}}\} \end{Bmatrix} \tag{49}$$

- Eq. (44):

$$\begin{aligned}
\tilde{\boldsymbol{z}}_{1/p} &= \tilde{\boldsymbol{z}}_{0/p} - \frac{1}{2}\left(\frac{\Delta t}{p}\right)^2 (\boldsymbol{w} + [A][P]\tilde{\boldsymbol{z}}_{0/p}) \\
&\Big\downarrow \text{In practice} \\
\begin{Bmatrix} \{\tilde{\boldsymbol{z}}_{1/p,\mathrm{C}}\} \\ \{\tilde{\boldsymbol{z}}_{1/p,\mathrm{F}}\} \end{Bmatrix} &= \begin{Bmatrix} \tilde{\boldsymbol{z}}_{0/p,\mathrm{C}} - \frac{1}{2}\left(\frac{\Delta t}{p}\right)^2 (\boldsymbol{w}_{\mathrm{C}} + [A_{\mathrm{CF}}]\tilde{\boldsymbol{z}}_{0/p,\mathrm{C}}) \\ \tilde{\boldsymbol{z}}_{0/p,\mathrm{F}} - \frac{1}{2}\left(\frac{\Delta t}{p}\right)^2 (\boldsymbol{w}_{\mathrm{F}} + [A_{\mathrm{FF}}]\tilde{\boldsymbol{z}}_{0/p,\mathrm{F}}) \end{Bmatrix}
\end{aligned} \tag{50}$$

- Eq. (45):

$$\begin{aligned}
\tilde{\boldsymbol{z}}_{(m+1)/p} &= 2\tilde{\boldsymbol{z}}_{m/p} - \tilde{\boldsymbol{z}}_{(m-1)/p} - \frac{1}{2}\left(\frac{\Delta t}{p}\right)^2 (\boldsymbol{w} + [A][P]\tilde{\boldsymbol{z}}_{m/p}) \\
&\Big\downarrow \text{In practice} \\
\begin{Bmatrix} \{\tilde{\boldsymbol{z}}_{(m+1)/p,\mathrm{C}}\} \\ \{\tilde{\boldsymbol{z}}_{(m+1)/p,\mathrm{F}}\} \end{Bmatrix} &= \begin{Bmatrix} 2\tilde{\boldsymbol{z}}_{m/p,\mathrm{C}} - \tilde{\boldsymbol{z}}_{(m-1)/p,\mathrm{C}} - \frac{1}{2}\left(\frac{\Delta t}{p}\right)^2 (\boldsymbol{w}_{\mathrm{C}} + [A_{\mathrm{CF}}]\tilde{\boldsymbol{z}}_{m/p,\mathrm{C}}) \\ 2\tilde{\boldsymbol{z}}_{m/p,\mathrm{F}} - \tilde{\boldsymbol{z}}_{(m-1)/p,\mathrm{F}} - \frac{1}{2}\left(\frac{\Delta t}{p}\right)^2 (\boldsymbol{w}_{\mathrm{F}} + [A_{\mathrm{FF}}]\tilde{\boldsymbol{z}}_{m/p,\mathrm{F}}) \end{Bmatrix}
\end{aligned} \tag{51}$$

- Eq. (46):

$$\boldsymbol{z}_{n+1} = -\boldsymbol{z}_{n-1} + 2\tilde{\boldsymbol{z}}_{p/p} \xrightarrow{\text{In practice}} \begin{Bmatrix} \{\boldsymbol{z}_{\mathrm{C},n+1}\} \\ \{\boldsymbol{z}_{\mathrm{F},n+1}\} \end{Bmatrix} = \begin{Bmatrix} -\boldsymbol{z}_{\mathrm{C},n-1} + 2\tilde{\boldsymbol{z}}_{p/p,\mathrm{C}} \\ -\boldsymbol{z}_{\mathrm{F},n-1} + 2\tilde{\boldsymbol{z}}_{p/p,\mathrm{F}} \end{Bmatrix} \tag{52}$$

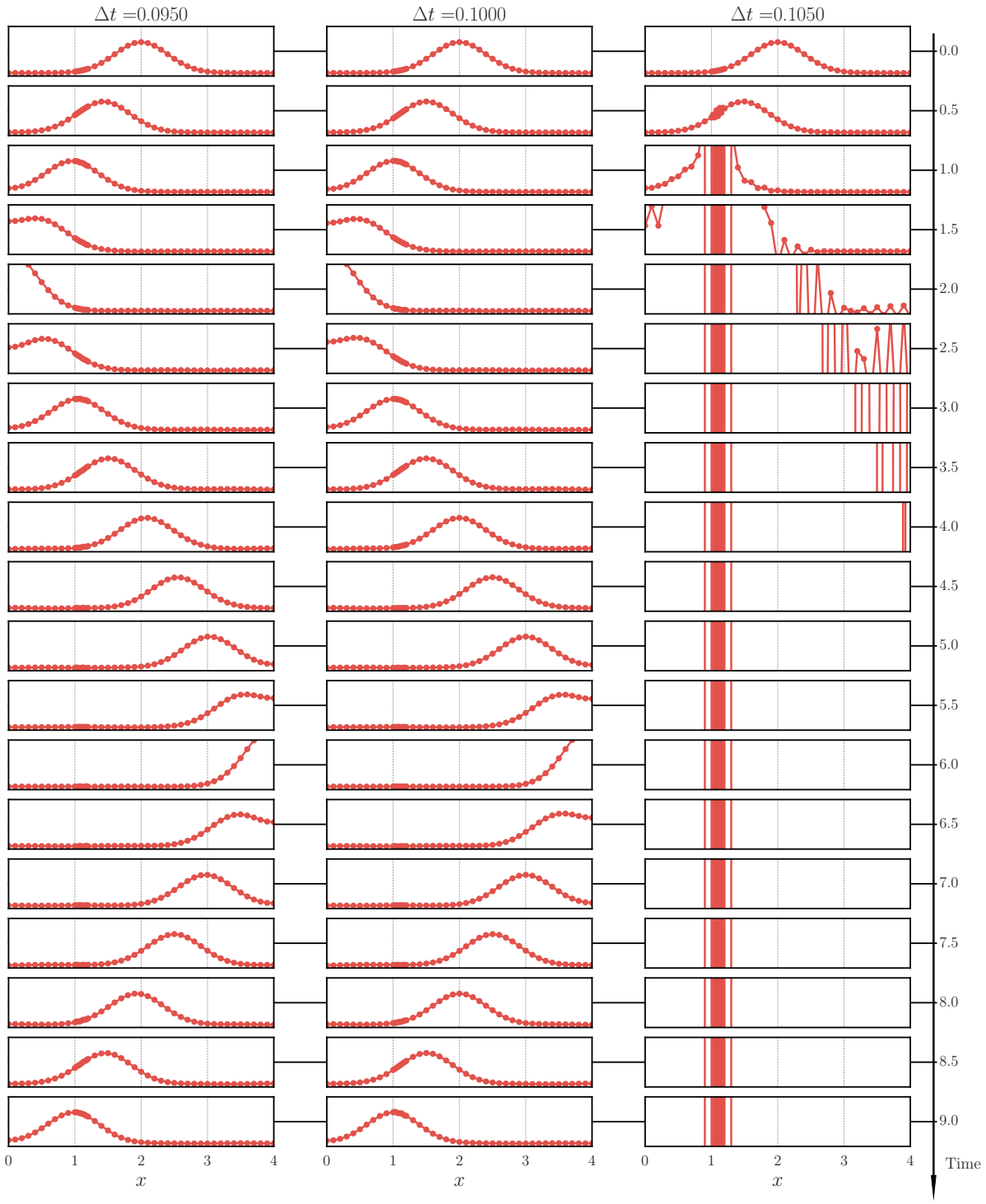

Figure 6: Time evolution of the solution for the LTS method with the refined mesh. Simulations are shown for three time steps: $\Delta t_{\text{LTS}} \in \{0.95\Delta t_{\text{VN},1}, \Delta t_{\text{VN},1}, 1.05\Delta t_{\text{VN},1}\}$. The stability of the solution is determined by the coarse mesh size, h_{coarse}, as the sub-time steps update only the fine nodes. Instability is observed only in the rightmost simulation, where $\Delta t = 1.05\Delta t_{\text{VN},1}$. $h_{\text{coarse}} = 0.1, p = 4 \implies h_{\text{fine}} = 0.025$.

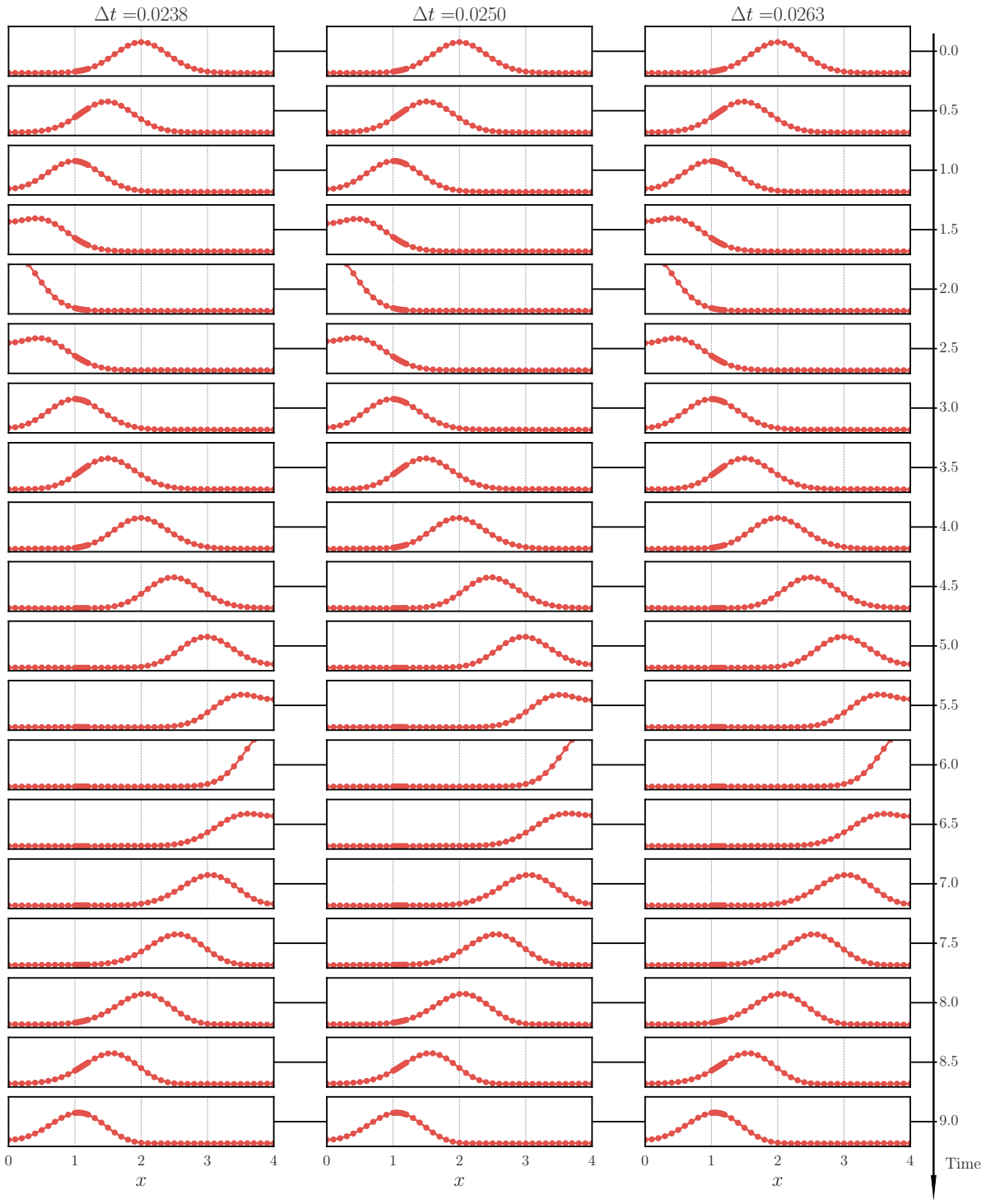

Figure 7: Time evolution of the solution for the LTS method with the refined mesh. Simulations are shown for three time steps: $\Delta t_{\text{LTS}} \in \{0.95\Delta t_{\text{VN},2}, \Delta t_{\text{VN},2}, 1.05\Delta t_{\text{VN},2}\}$. No instability is observed for any of the simulations, demonstrating that the solution remains stable even with the refined mesh and time steps around the stability criterion determined by h_{fine}. $h_{\text{coarse}} = 0.1, p = 4 \implies h_{\text{fine}} = 0.025$.

References

[1] Julien Diaz and Marcus J. Grote. "Energy Conserving Explicit Local Time Stepping for Second-Order Wave Equations". In: *SIAM Journal on Scientific Computing* 31.3 (2009), pp. 1985–2014. DOI: 10.1137/070709414. eprint: https://doi.org/10.1137/070709414. URL: https://doi.org/10.1137/070709414.

[2] Josef Stoer and Roland Bulirsch. *Introduction to Numerical Analysis*. 3rd. Springer, 2002. ISBN: 978-0-387-95452-3.

[3] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.