Acoustic FD Modelling, RTM

M D Sacchi

Inverse Problems UNLP - 2024

https://github.com/msacchi/UNLP-2024-Inversion

Codes are in **/FD_and_RTM**

Contents

- Solving the acoustic wave equation via the Finite Difference (FD) method
- Adding ABC to FD solver
- RTM

2D Acoustic Wave Equation

- Our first task is to solve the AWE to estimate the source side wavefield

$$m(\mathbf{x})\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2} - \nabla^2 u(\mathbf{x},\mathbf{t}) = s(\mathbf{x},t) \quad in \quad t \in [0,T], \quad \mathbf{x} \in \Omega$$

$$m(\mathbf{x}) = \frac{1}{c(\mathbf{x})^2}, \qquad s(\mathbf{x},t) = \delta(\mathbf{x} - \mathbf{x}_s)w(t)$$

$$\nabla^2 u(\mathbf{x},t) = \frac{\partial^2 u(\mathbf{x},t)}{\partial x^2} + \frac{\partial^2 u(\mathbf{x},t)}{\partial z^2}$$

$$u(\mathbf{x},0) = 0\,, \quad u_t(\mathbf{x},t)\,|_{t=0} = 0$$

2D Acoustic Wave Equation

- Continuous formulation:

$c(\mathbf{x})$: velocity of 2D the medium

$s(\mathbf{x}, t) = \delta(\mathbf{x} - \mathbf{x}_s)w(t)$: Source function

$\mathbf{x}_s = (x_s, z_s)$: Source coordinate

$u(\mathbf{x}, t) = u(x, z, t)$: Acoustic wavefield

$w(t)$: Temporal source signature

$\mathbf{x} = (x, z)$: Coordinates, $\quad t$: Time

2D Acoustic Wave Equation

- Discrete formulation:

c_{ij} : velocity of 2D the medium

$s_{ij}^{n} = M_{ij} w^{n}$: Discrete source function

M_{ij} : Matrix to place source

u_{ij}^{n} : Discrete acoustic wavefield

- Mesh coordinates:

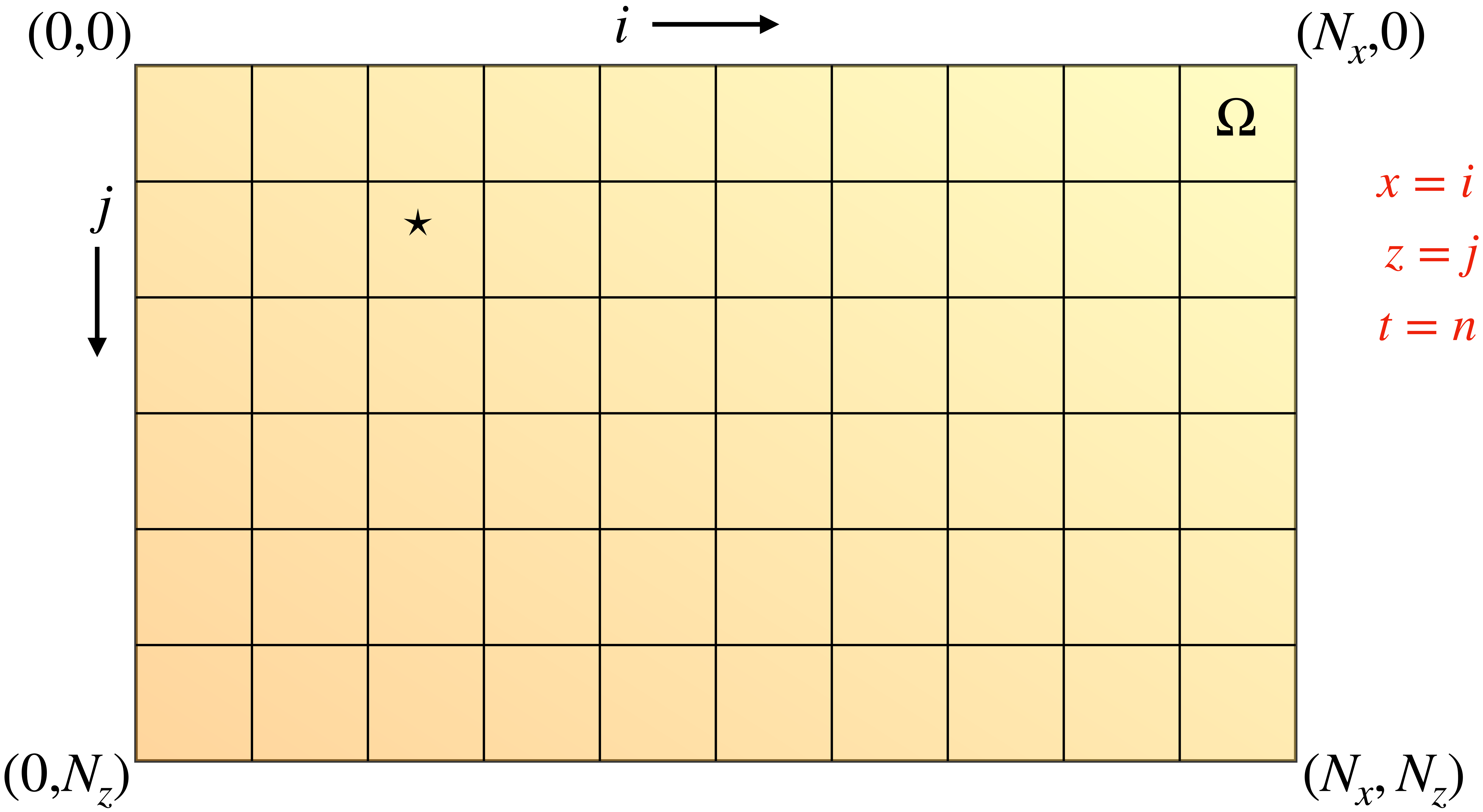
$$x = i\,\Delta x\,, i = 0 : N_x$$
$$z = j\,\Delta z\,, j = 0 : N_z$$
$$t = n\,\Delta t\,, n = 0 : N_t$$

$$\boxed{u(x,z,t) \rightarrow u_{ij}^{n}}$$

- Placement of the source:

$$M_{ij} = \begin{cases} 1 & i = i_s, j = j_s \\ 0 & \text{elsewhere} \end{cases}$$

Physical domain for wavefield



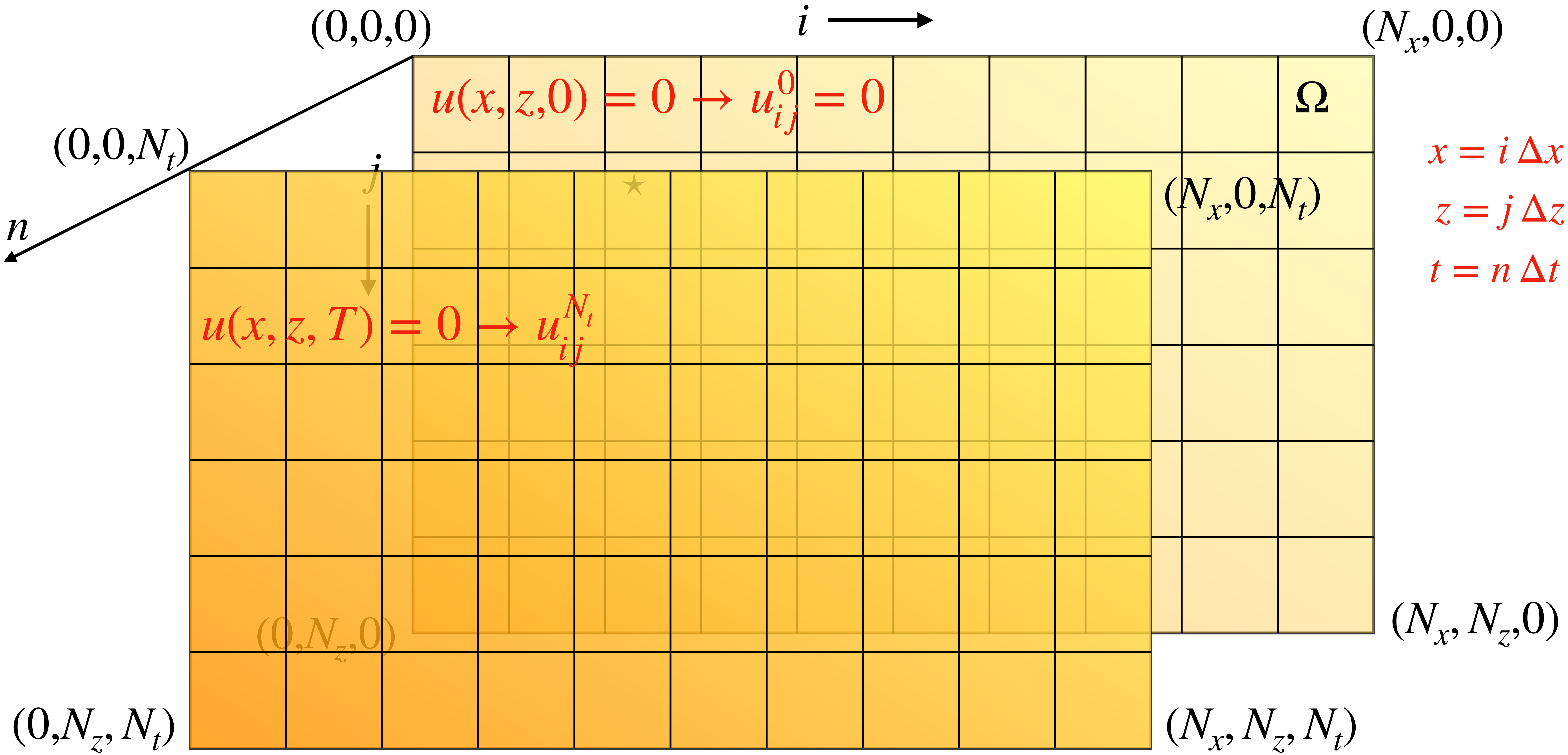
★ Source (x_s, z_s)

$x = i\,\Delta x\,, i = 0 : N_x$

$z = j\,\Delta z\,, j = 0 : N_z$

$t = n\,\Delta t\,, n = 0 : N_t$

Physical domain for the velocity field



$$x = i\,\Delta x\,, i = 0 : N_x$$
$$z = j\,\Delta z\,, j = 0 : N_z$$
$$t = n\,\Delta t\,, n = 0 : N_t$$

Discretizing the AWE

(1) $$m(\mathbf{x})\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2} - \nabla^2 u(\mathbf{x},\mathbf{t}) = s(\mathbf{x},t) \quad in \quad t \in [0,T], \quad \mathbf{x} \in \Omega$$

(2) $$\mathbf{x} = (x_i, z_j)\,, t = t_n \rightarrow \quad m(\mathbf{x}) \approx m_{i,j} = \frac{1}{c_{ij}^2}$$

(3) $$\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2} \approx \frac{u_{ij}^{n+1} - 2u_{ij}^n + u_{ij}^{n-1}}{\Delta t^2}$$

(4) $$\nabla^2 u(\mathbf{x},t) \approx \frac{u_{i+1\,j}^n - 2u_{ij}^n + u_{i-1\,j}^n}{\Delta x^2} + \frac{u_{i\,j+1}^n - 2u_{ij}^n + u_{i\,j+1}^n}{\Delta z^2}$$

(5) $$s(\mathbf{x},t) \approx M_{ij}\, w^n$$

Discretizing the AWE

- Replacing (2),(3),(4) and (5) into (1) leads to

$$u_{ij}^{n+1} = 2u_{ij}^{n} - u_{ij}^{n-1} + \Delta t^2 c_{ij}^2 (\frac{u_{i+1j}^n - 2u_{ij}^n + u_{i-1j}^n}{\Delta x^2} + \frac{u_{ij+1}^n - 2u_{ij}^n + u_{ij+1}^n}{\Delta z^2}) + \tilde{M}_{i,j} w^n$$

letting $\Delta x = \Delta z = h \quad \rightarrow$

$$u_{ij}^{n+1} = 2u_{ij}^{n}(1 - 2\alpha_{ij}) - u_{ij}^{n-1} + \alpha_{ij}(u_{i+1j}^n + u_{i-1j}^n + u_{ij+1}^n + u_{ij+1}^n) + \tilde{M}_{i,j} w^n$$

$$\alpha_{ij} = \frac{c_{ij}^2 \Delta t^2}{h^2}, \qquad \tilde{M}_{ij} = M_{ij} \Delta t^2 c_{ij}^2$$

Discretizing the AWE

- Replacing (2),(3),(4) and (5) into (1) leads to

$$u_{ij}^{n+1} = 2u_{ij}^{n} - u_{ij}^{n-1} + \boxed{L[u_{i,j}]^n} + \tilde{M}_{ij}s^n$$

Laplacian

- We could vectorized the wavefield at time $n+1, n, n-1$

$$\mathbf{u}^{n+1} = 2\mathbf{u}^{n} - \mathbf{u}^{n-1} + \mathbf{L}\mathbf{u}^{n} + \mathbf{s}^{n}$$

Discretizing the AWE

Courant–Friedrichs–Lewy (CFL) condition for second order approximation in time and space:

$$\boxed{\frac{c_{max}\Delta t}{h} < \frac{1}{\sqrt{2}}}$$

CFL is a necessary condition for convergence while solving numerically the acoustic wave equation.

- Wu, W., Lines, L.R., and Lu, H., 1996, Analysis of higher-order finite-difference schemes in 3-D reverse-time migration: Geophysics, 61, 845-856.
- Mufti, I.R., 1990, Large-scale three-dimensional seismic models and their interpretive significance: Geophysics, 55, 1166-1182.

Absorbing Boundary Conditions

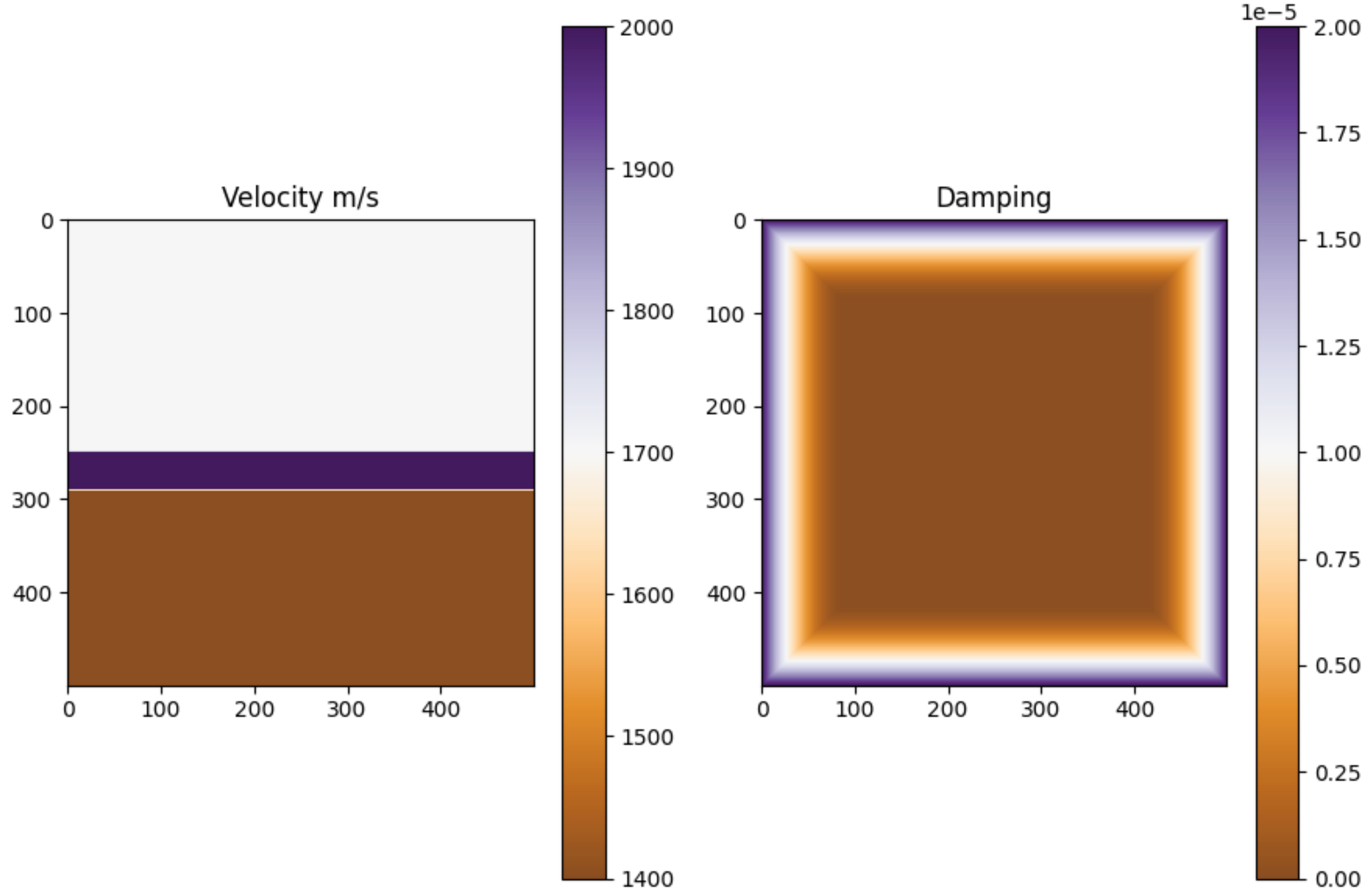
- To avoid undesired reflections from the physical borders we use absorbing boundary conditions. In this case, we modify the wave equation as follows

$$m(\mathbf{x})\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2} + a(\mathbf{x})\frac{\partial u(\mathbf{x},t)}{\partial t} - \nabla^2 u(\mathbf{x},\mathbf{t}) = s(\mathbf{x},t) \quad in \quad t \in [0,T], \quad \mathbf{x} \in \Omega$$

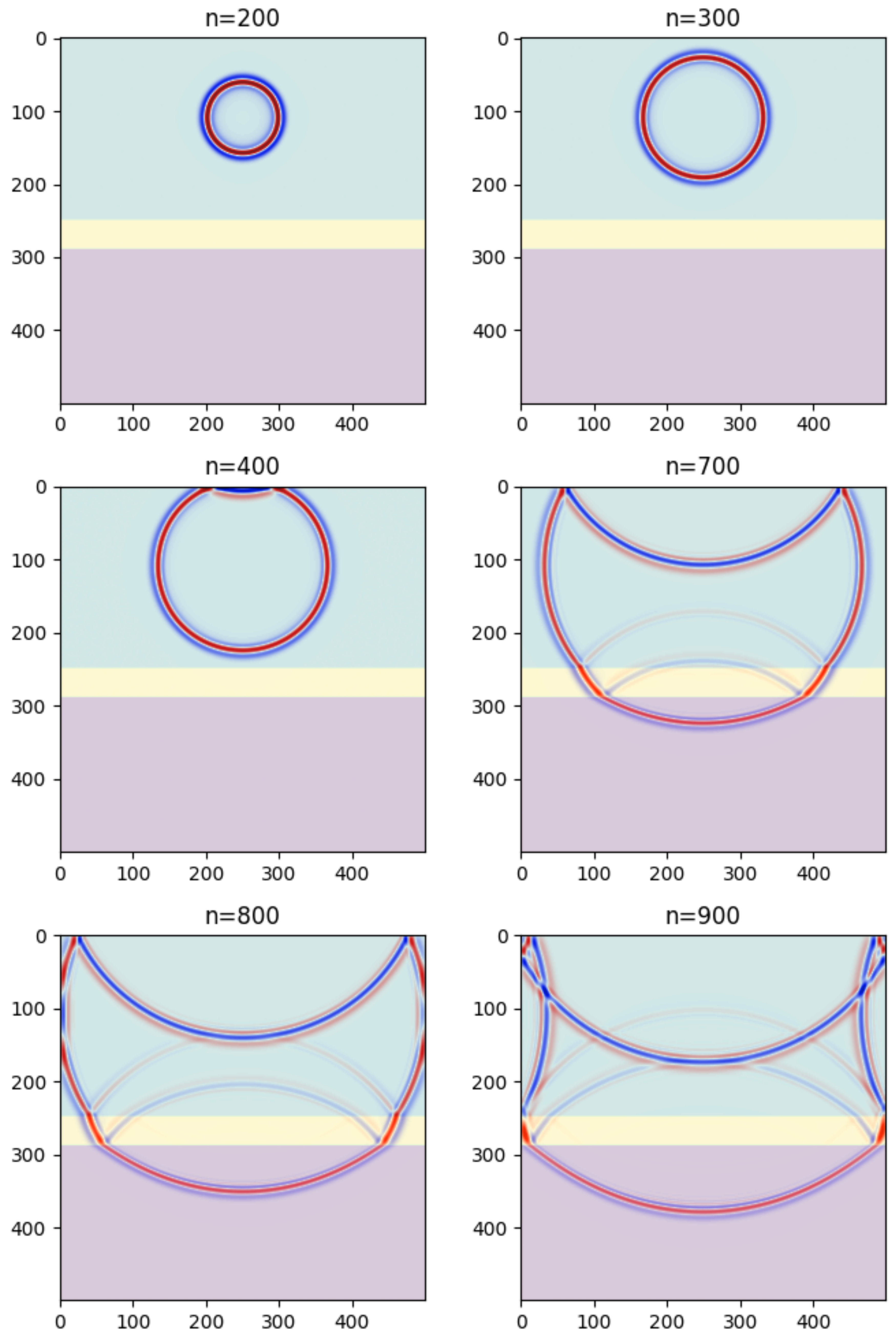
- Where $a(\mathbf{x})$ is the damping function that slowly annihilates wave propagation as the wave approaches the physical borders.

Sochacki et. Al, 1987, Absorbing Boundary conditions and surface wave. GEOPHYSICS, 52(1).

Velocity model and damping function used to simulate the propagation of an acoustic wave

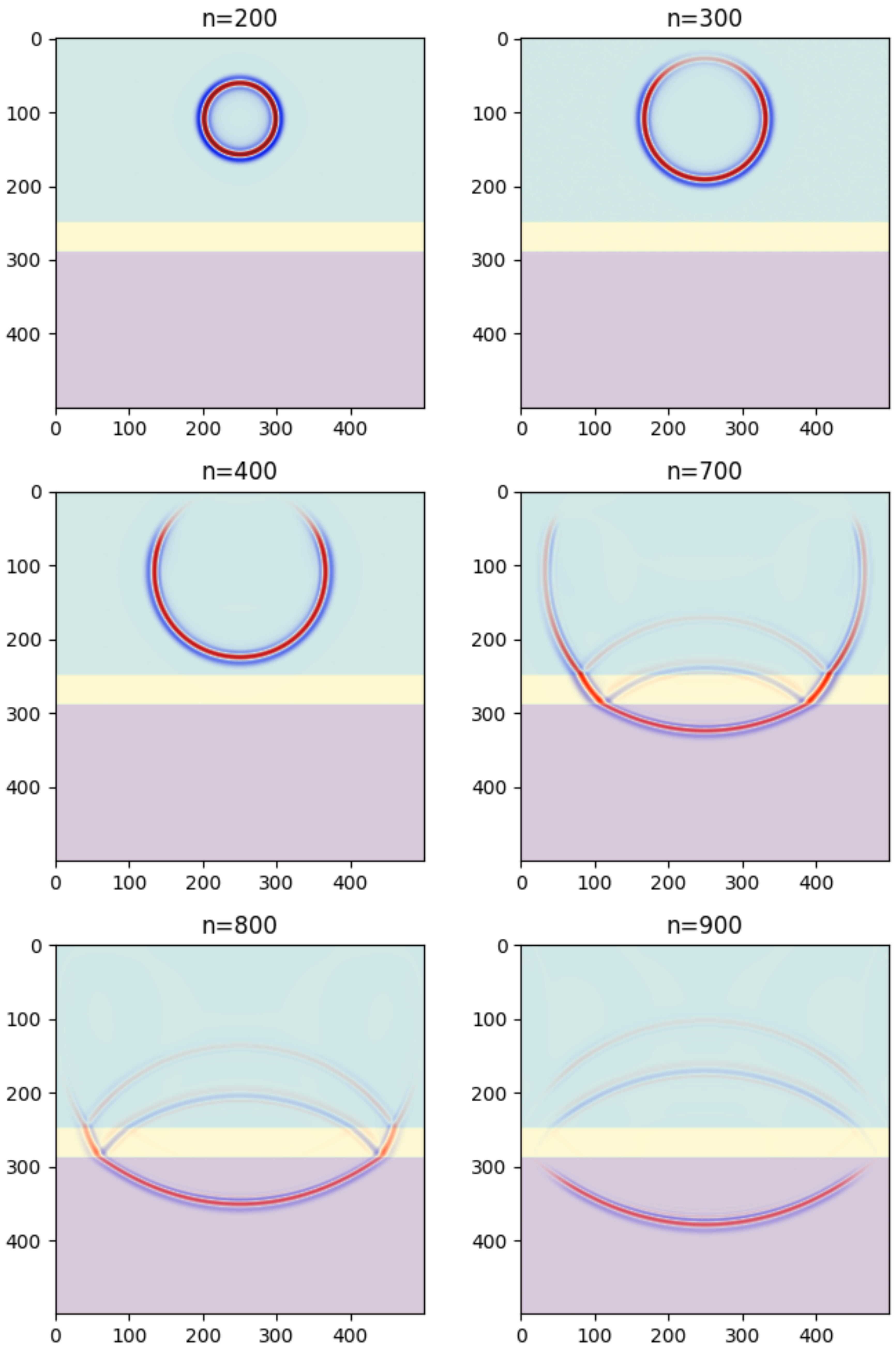


npad=50 grid points



Simulation of a propagating acoustic wave in a media with reflecting BCs.

../RTM/demo_1.jl



Simulation of a propagating acoustic wave in a media with absorbing boundary conditions (ABCs).

../RTM/demo_1.jl

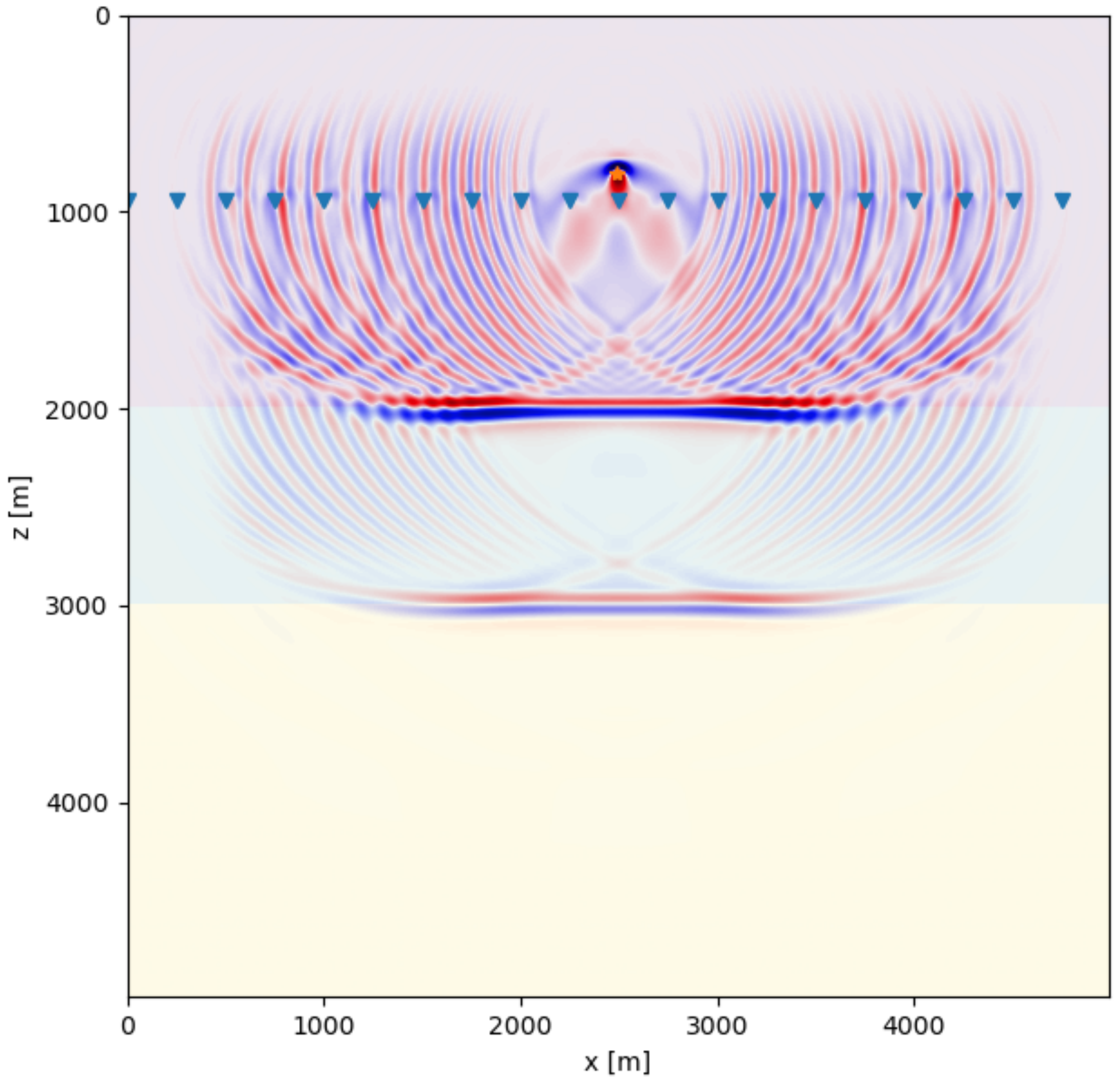
Damping function

```
function damping_2D(nx, nz, D; fact=0.000015, degree=2)
# Adjust damping function for sponge boundary conditions
    g = zeros(nx,nz)

    for i in 1:nx
        for j in 1:nz
            d_i = min(i - 1, nx - i)  # Distance from the nearest horizontal boundary
            d_j = min(j - 1, nz - j)  # Distance from the nearest vertical boundary
            d = min(d_i, d_j)         # Minimum distance to any boundary
            if d < D
                g[i, j] = (1 - d / D)
            else
                g[i, j] = 0.0
            end
        end
    end
    g = g.^degree*fact

    return g
end
```

Synthetic RTM demo_2.jl



rtm.jl

```
function rtm(nx, nz, nt, dx, dz, dt, f0, c, c0, npad, ix_s, iz_s, ix_r, iz_r)

  I = zeros(nx,nz)
 ns = length(ix_s)
 nr = length(ix_r)



 for k = 1:ns

  data = read_bin("shot_"*string(k)*".bin",nr,nt)

     S = source_2D(nx, nz, nt, dt, f0, ix_s[k], iz_s[k])
    WS =  afd_2D_f(nx, nz, nt, dx, dz, dt, c, npad, S)    # Source wavefield  WS(x,z,t)

     R = receiver_2D(nx, nz, nt, data, ix_r, iz_r)
    WR =  afd_2D_b(nx, nz, nt, dx, dz, dt, c, npad, R)    # Receiver wavefeild WR(x,z,t)


   I = I + dropdims(sum(WS.*WR,dims=3),dims=3):q


 end

   Imax = maximum(I)
   I = I/Imax

return I
end
```

I loop over shot and do the following

1. Compute source side wavelfied **af_2D_f.jl**
2. Compute receiver side wavefield **af_2D_b.jl**
3. Multiple element-wise fields and sum over time.

source_2D.jl sets the source

receiver_2D.jl injects data into receiver positions

In Julia you can include(‘lib.jl’) and then

>?source_2D.jl

Will tell you what each program in lib does