On the impact of boundary conditions in a wave equation

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Sep 1, 2014

Abstract

The purpose of this document is to demonstrate how different types of boundary conditions impacts the solution of standard, linear wave equation in one spatial dimension. Four types of conditions are treated: known value (Dirichlet condition), zero flux (Neumann condition), open boundary, and periodic boundary.

Contents

1 Mathematical model and solution method

We solve a one-dimensional, linear, constant-coefficient wave equation by an explicit finite difference method.

1.1 The wave equation problem

The standard, linear wave equation in a homogeneous one-dimensional medium reads

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad x \in (0, L),\ t \in (0, T]\,. \tag{1}$$

The unknown function u depends on space x and time t: $u = u(x, t)$.

The need for boundary conditions in the wave equation.

Four initial and boundary conditions must be specified to have a unique solution:

- Initial condition for $u(x, 0)$
- Initial condition for $u_t(x, 0)$
- Boundary condition at $x = 0$
- Boundary condition at $x = L$

1.2 Initial conditions

Most demonstrations will start with an initial profile of u,

$$u(x, 0) = I(x),$$

being at rest, i.e.,

$$\frac{\partial}{\partial t} u(x, 0) = 0\,.$$

Two initial profiles will be considered:

- a plug as to left in Figure 1

- a Gaussian peak as to the right in Figure 1

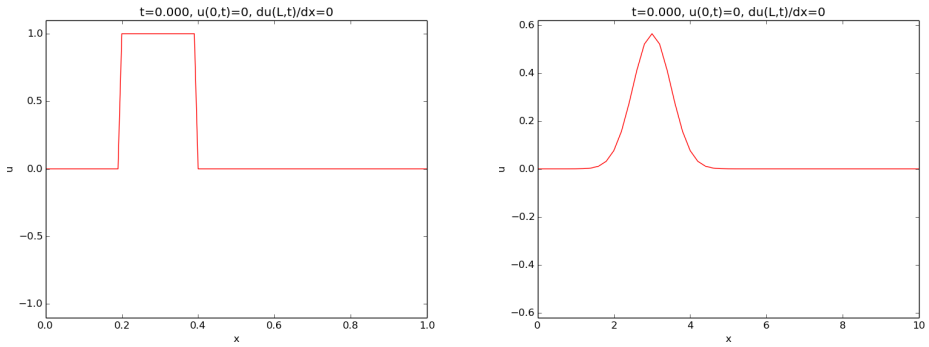


Figure 1: Examples on initial conditions.

1.3 Boundary conditions

Fixed u. At $x = 0$ we will sometimes use the condition $u = 0$, often known as a homogeneous *Dirichlet* condition. This condition will mirror the wave.

Reflecting condition. At $x = 0$ and/or $x = L$ we will apply a *reflecting* or *no-flux* condition:

$$\frac{\partial u}{\partial x} = 0\,. \tag{2}$$

This condition reflects the wave into the domain again, as a surface wave hits a vertical wave, runs up to the double amplitude, and propagates back into the domain again. This type of boundary condition is also referred to as a *Neumann* condition.

Feeding a wave from the boundary. We shall demonstrate the effect of moving u at the boundary $x = 0$ to feed the domain with an incoming wave. The boundary condition then reads

$$u(0, t) = U_0(t),$$

for some given function $U_0(t)$. A particular choice in a later demonstration is a sine function that is active in three different time intervals:

$$U_0(t) = \begin{cases} \frac{1}{4}\sin(6\pi t), & t \in T_1 \text{ or } t \in T_2 \text{ or } t \in T_3 \\ 0, & \text{otherwise} \end{cases}$$

where $T_1 = [0, \frac{1}{6}]$, $T_2 = [\frac{3}{4}, \frac{5}{6}]$, and $T_3 = [\frac{3}{2}, \frac{11}{6}]$. The movement of u at the boundary will produce a wave that is by the PDE transported to the right into the domain.

Open boundary condition. Very often one wants to let a wave travel through the boundary without being disturbed. Such a condition is called an *open* boundary condition, or a *radiation* condition, or an *artificial* boundary condition:

$$\frac{\partial u}{\partial t} - c\frac{\partial u}{\partial x} = 0, \quad x = 0, \tag{3}$$

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \quad x = L\,. \tag{4}$$

These conditions work exactly in 1D, but are challenging to generalize and implement in 2D and 3D.

Periodic boundary condition. When following a wave motion over large distances, it is desireable to let a wave travel out of the right domain and at the same time feed the wave back into the domain from the left. This approach avoids a very large domain where nothing happens in the majority of the domain. A *periodic* boundary condition at $x = 0$ can be used to feed the signal traveling out at $x = L$ into the domain:

$$u(0,t) = u(L,t)\,. \tag{5}$$

The condition at $x = L$ is then an open boundary condition (4).

1.4 Numerical solution method

The wave equation is solved by an explicit finite difference method, which is of second-order in space and time. A uniform mesh with spacing Δx and Δt is used in space and time, respectively. The no-flux or Neumann boundary conditions are implemented by modifying the computational stencil at the boundary. The open boundary conditions are implemented by forward in time, upstream in space finite differences, which exactly let the wave out of the boundary. More details are found in Appendix A. Parts of the computer code are explained in Appendix B.

2 Demonstrations

2.1 Dirichlet and Neumann conditions: reflecting and mirroring boundaries

The first two animations demonstrates the differences between a Dirichlet condition $u = 0$ at the boundary and a Neumann condition $\partial u/\partial x = 0$.

`mov-wavebc/gaussian_Nx50_C1_un0/movie.mp4`

Movie 1: Reflecting boundaries ($u_x = 0$) for a Gaussian wave.

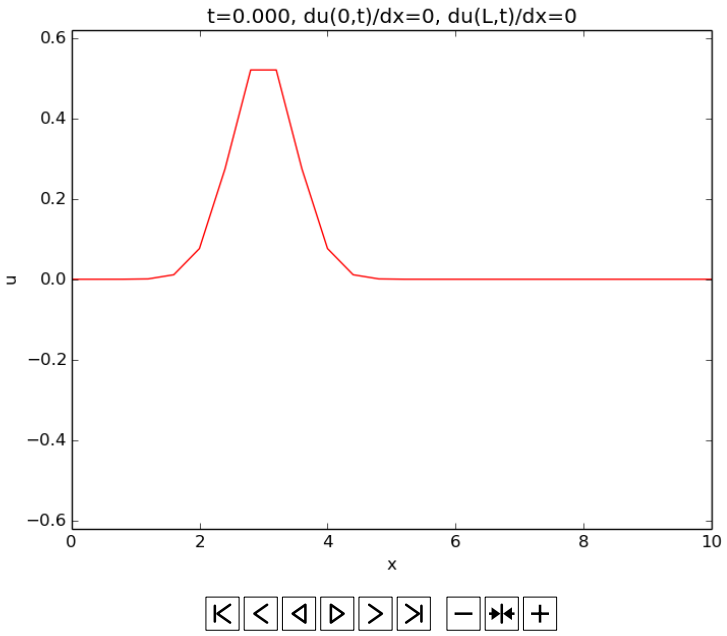
`mov-wavebc/gaussian_Nx50_C1_u0/movie.mp4`

Movie 2: Mirroring boundaries ($u = 0$) for a Gaussian wave.

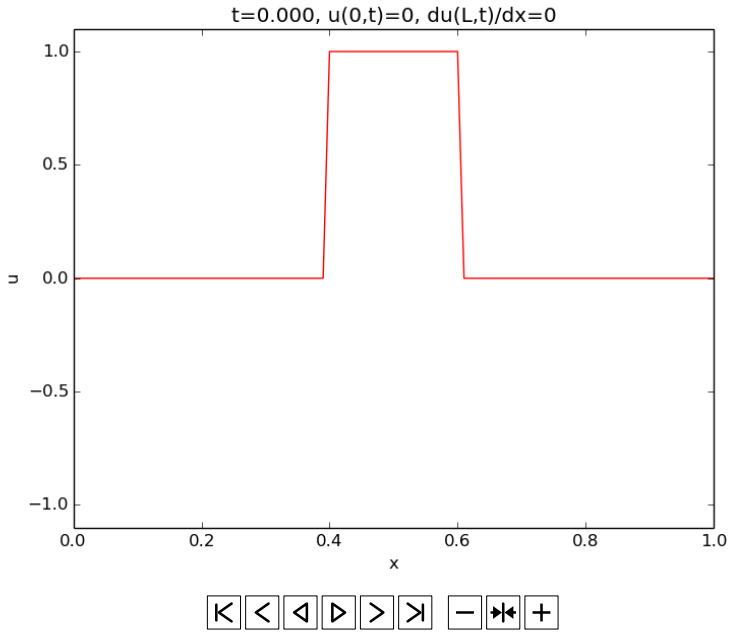
Instead of a Gaussian wave profile, we can test the geometrically simple plug profile.

`mov-wavebc/plug_Nx100_C1/movie.mp4`

Numerical noise. All of the above computations were run with unit Courant number, which means that the solutions are exact without any numerical errors. (This is a remarkable property of the numerical solution method.) For smaller Courant numbers, numerical noise may be visible, depending on the smoothness of the initial profile. Below is a smooth Gaussian profile and the almost discontinuous plug profile.



Movie 4: Reflecting boundaries ($u_x = 0$) for a Gaussian wave, computed with $C = 0.5$, which implies numerical noise on a coarse grid.



Movie 5: A plug wave generates very significant numerical noise ($C = 0.5$).

2.2 Effect of impulsive start of waves

The previous demonstrations had an initial condition with a prescribed $u = 0$ profile at rest ($u_t = 0$). Alternatively, one may start with a flat profile $u = 0$ and use an initial condition $u_t = V \neq 0$ to impulsively start the wave motion. For example, if we think of u as the displacement of a string on a string instrument, this set of initial conditions corresponds to an undeformed string that is given an impulsive start from an impact.

`mov-wavebc/gaussian_Nx50_C1_V/movie.mp4`

Movie 6: Impulsive start of a wave motion: $I = 0$, $V \neq 0$.

2.3 Feeding of waves from the boundary

We can also start with a flat profile, $u = 0$, at rest, $u_t = 0$, and create propagating signals by moving u at the boundary. That is, we have a time-varying Dirichlet condition $u(0, t) = U_0(t)$ at the left boundary. The movements lead to signals that propagate to the right. In the movie, the movements are paused to make separate signals.

https://www.youtube.com/watch?v=-oe8q83anXE

Movie 7: Feeding of waves from the boundary by a time-dependent Dirichlet condition $U_0(t)$.

2.4 Open and periodic boundary conditions

`mov-wavebc/periodic_Nx100_C1_l0/movie.mp4`

Movie 8: Demonstrations of periodic boundary condition on the left combined with an open boundary condition on the right: waves passing out of the domain enter the left end again.

`mov-wavebc/periodic_Nx100_C1_l0.3/movie.mp4`

Movie 9: Demonstrations of periodic boundary condition on the left combined with an open boundary condition on the right: waves passing out of the domain enter the left end again.

Error in the open boundary condition. In 2D and 3D...

`mov-wavebc/periodic_error_Nx100_C1_l0.3/movie.mp4`

Movie 10: Periodic boundary condition combined with a slight wrong open boundary condition at the right end (20

Summary of boundary conditions.

Condition	Formula	Effect
Dirichlet	$u(0,t) = 0$	mirror wave
Dirichlet	$u(0,t) = U_0(t)$	feed incoming wave
Neumann	$u_x(0,t) = 0$	reflect wave
Open	$u_t \pm cu_x = 0$	let wave out of the domain
Periodic	$u(0,t) = u(L,t)$	turn outgoing wave to incoming

Science is a differential equation. Religion is a boundary condition.
Alan Turing, computer scientist, 1912-1954.

A Numerical solution method

A.1 Approximating the wave equation

We introduce a uniform mesh in space and time with spacings Δx and Δt, respectively. At each mesh point

$$(x_i, t_n), \quad x_i = i\Delta x,\ i = 0, \ldots, N_x, \quad t_n = n\Delta t,\ n = 0, \ldots, N_t,$$

the wave equation is approximated by second-order finite differences,

$$\frac{\partial^2}{\partial t^2}u(x_i,t_n) \approx \frac{u_i^{n+1}-2u_i^n+u_i^{n-1}}{\Delta t^2},$$
$$\frac{\partial^2}{\partial x^2}u(x_i,t_n) \approx \frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{\Delta x^2},$$

where u_i^n is the numerical approximation to the exact solution at (x_i, t_n). These approximations give rise to an explicit scheme where a new value u_i^{n+1} is given by the formula

$$u_i^{n+1} = -u_i^{n-1} + 2u_i^n + C^2\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right), \tag{6}$$

where the parameter C,

$$C = c\frac{\Delta t}{\Delta x}, \tag{7}$$

is known as the dimensionless *Courant number*. A stable time-stepping is ensured only if $C \leq 1$. The value C governs also the accuracy of the method: $C = 1$ actually reproduces the exact solution (!), while the accuracy is reduced when decreasing C.

A.2 Approximating the initial conditions

The initial condition $u(x, 0) = I(x)$, where $I(x)$ is a prescribed mathematical function, is implemented in the numerical method by

$$u(x_i, 0) = I(x_i), \quad i = 0, \ldots, N_x\,.$$

The other initial condition,

$$\frac{\partial}{\partial t}(x, 0) = 0, \text{ or } \frac{\partial}{\partial t}(x, 0) = V(x),$$

is approximated by a centered difference over an interval $2\Delta t$. When this difference is combined with (6), we get a special formula for u_i^1. Thereafter, for $n \geq 1$, one can apply (6).

A.3 Approximation of boundary conditions

The finite difference scheme (6) is applied at all *inner points* in the spatial mesh, $i = 1, \ldots, N_x - 1$. For $i = 1$ or $i = N_x - 1$, (6) involves the boundary points u_0^n or $u_{N_x}^n$ (respectively) at the previous time step, but these are condidered known.

Dirichlet conditions. In case of $u = 0$ (Dirichlet) conditions, we just set $u_0^{n+1} = 0$ and $u_{N_x}^{n+1} = 0$. Feeding a wave in from the left is just a matter of setting u_0^{n+1} equal to the known boundary function of time: $u_0^{n+1} = U_0((n + 1)\Delta t)$.

Neumann conditions. No-flux or Neumann conditions are discretized by a centered finite difference and combined with (6), yielding a modified form of (6) at the boundary:

$$u_i^{n+1} = -u_i^{n-1} + 2u_i^n + 2C^2\left(u_{i+1}^n - u_i^n\right), \quad i = 0 \tag{8}$$

$$u_i^{n+1} = -u_i^{n-1} + 2u_i^n + 2C^2\left(u_{i-1}^n - u_i^n\right), \quad i = N_x\,. \tag{9}$$

Open boundary conditions. Radiation, artificial, or open boundary conditions of the type (3)-(4) are discretized by upstream first-order, forward in time differences, resulting in explicit updating formulas:

$$u_i^{n+1} = u_i^n + C(u_{i+1}^n - u_i^n), \quad i = 0, \tag{10}$$

$$u_i^{n+1} = u_i^n - C(u_i^n - u_{i-1}^n), \quad i = N_x\,. \tag{11}$$

Even though the underlying finite differences are of first order only, these conditions let the waves out of the domain perfectly and do not lower the accuracy of the finite difference scheme used in the interior of the domain or at no-flux (Neumann) boundaries.

Periodic conditions. The periodic condition $u(0,t) = u(L,t)$ is trivial to incorporate in the numerical method:

$$u_0^{n+1} = u_{N_x}^{n+1}\,.$$

B Computer code

The complete code used for all experiments except those involving periodic boundary conditions can be found in the file `wave1D_dn.py`. The basic solver function for problems without open boundary conditions and periodic conditions is listed below.

The key computation is the time stepping loop where (6) is used to find new u_i^{n+1} values at each time level. In addition, a special formula for the first step is needed, and boundary conditions must be incorporated at the boundary points.

```
from numpy import linspace, zeros

def solver(I, V, f, c, U_0, U_L, L, dt, C, T,
           user_action=None, version='scalar'):
    """
    Solve u_tt = c**2*u_xx for x in [0,L] and t in [0,T],
    with u(x,0)=I(x), u_t(x,t)=V(x), u(0,t)=U_0(t), u(L,t)=U_L(t),
    and time step size dt and Courant number C.
    A Neumann condition is applied by setting U_0 or  U_L to None.
    user_action(u, x, t, n) is called at every time step with
    time t[n], the solution in array u, and corresponding to x
    coordinates in array x.
```

```
"""
Nt = int(round(T/dt))          # No of time intervals
t = linspace(0, Nt*dt, Nt+1)   # Mesh points in time
dx = dt*c/float(C)             # Mesh spacing
Nx = int(round(L/dx))          # No of space intervals
x = linspace(0, L, Nx+1)       # Mesh points in space

C2 = C**2; dt2 = dt*dt         # Help variables in the scheme

u   = zeros(Nx+1)    # Solution array at new time level
u_1 = zeros(Nx+1)    # Solution at 1 time level back
u_2 = zeros(Nx+1)    # Solution at 2 time levels back

Ix = range(0, Nx+1)  # Indices for x mesh
It = range(0, Nt+1)  # Indices for t mesh

# Load initial condition into u_1
for i in Ix:
    u_1[i] = I(x[i])

if user_action is not None:
    user_action(u_1, x, t, 0)

# Special formula for the first step
for i in Ix[1:-1]:
    u[i] = u_1[i] + dt*V(x[i]) + \
           0.5*C2*(u_1[i-1] - 2*u_1[i] + u_1[i+1]) + \
           0.5*dt2*f(x[i], t[0])

i = Ix[0]
if U_0 is None:
    # Set boundary values du/dn = 0
    # x=0: i-1 -> i+1 since u[i-1]=u[i+1]
    # x=L: i+1 -> i-1 since u[i+1]=u[i-1])
    ip1 = i+1
    im1 = ip1  # i-1 -> i+1
    u[i] = u_1[i] + dt*V(x[i]) + \
           0.5*C2*(u_1[im1] - 2*u_1[i] + u_1[ip1]) + \
           0.5*dt2*f(x[i], t[0])
else:
    u[0] = U_0(dt)

i = Ix[-1]
if U_L is None:
    im1 = i-1
    ip1 = im1  # i+1 -> i-1
    u[i] = u_1[i] + dt*V(x[i]) + \
           0.5*C2*(u_1[im1] - 2*u_1[i] + u_1[ip1]) + \
           0.5*dt2*f(x[i], t[0])
else:
    u[i] = U_L(dt)

if user_action is not None:
    user_action(u, x, t, 1)

# Update data structures for next step
u_2[:], u_1[:] = u_1, u

for n in It[1:-1]:
    # Update all inner points
    if version == 'scalar':
```

```
            for i in Ix[1:-1]:
                u[i] = - u_2[i] + 2*u_1[i] + \
                       C2*(u_1[i-1] - 2*u_1[i] + u_1[i+1]) + \
                       dt2*f(x[i], t[n])

        elif version == 'vectorized':
            u[1:-1] = - u_2[1:-1] + 2*u_1[1:-1] + \
                      C2*(u_1[0:-2] - 2*u_1[1:-1] + u_1[2:]) + \
                      dt2*f(x[1:-1], t[n])
        else:
            raise ValueError('version=%s' % version)

        # Insert boundary conditions
        i = Ix[0]
        if U_0 is None:
            # Set du/dx = 0
            # x=0: i-1 -> i+1 since u[i-1]=u[i+1] when du/dn=0
            # x=L: i+1 -> i-1 since u[i+1]=u[i-1] when du/dn=0
            ip1 = i+1
            im1 = ip1
            u[i] = - u_2[i] + 2*u_1[i] + \
                   C2*(u_1[im1] - 2*u_1[i] + u_1[ip1]) + \
                   dt2*f(x[i], t[n])
        else:
            u[0] = U_0(t[n+1])

        i = Ix[-1]
        if U_L is None:
            im1 = i-1
            ip1 = im1
            u[i] = - u_2[i] + 2*u_1[i] + \
                   C2*(u_1[im1] - 2*u_1[i] + u_1[ip1]) + \
                   dt2*f(x[i], t[n])
        else:
            u[i] = U_L(t[n+1])

        if user_action is not None:
            if user_action(u, x, t, n+1):
                break

        # Update data structures for next step
        u_2[:], u_1[:] = u_1, u

    return u, x, t
```

Open boundary conditions can be handled as follows.

```
def solver(...):
    ...
    for n in range(1, Nt):
        # Update all inner points at time t[n+1]

        # Insert open boundary conditions at the end points
        i = Nx
        u[i] = u_1[i] - C*(u_1[i] - u_1[i-1])
        i = 0
        u[i] = u_1[i] + C*(u_1[i+1] - u_1[i])
```

A period condition at $x = L$, after an open conditon in the beginning of the simulation, can be coded as

```
if periodic:
    u[0] = u[Nx]
else:
    i = 0
    u[i] = u_1[i] + C*(u_1[i+1] - u_1[i])
```

where `peridoc` becomes `True` when `u[-1] > eps` for some tolerance `eps`, say `1E-4` (i.e., the outgoing wave hits the right boundary).

Index