INF5620: Compulsory excercise 2 - report

John-Anders Stende

Abstract

Bondary conditions are essential for how a wave will propogate in time. The purpose of this assignment is to compare the convergence rates for different discretizations of Neumann boundary conditions on a 1d wave with variable wave velocity. I will compute the convergence rate for four different discretizations and come to conclusions of why these are different or not.

Problem

The differential equation for a 1d wave with variable wave velocity is

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\left(q(x)\frac{\partial u}{\partial x}\right) + f(x,t)$$

or in a more compact form:

$$u_{tt} = (qu_x)_x + f(x,t)$$

We now introduce the mesh function u_i^n, which approximates the exact solution at the mesh point (x_i, t_n) for $i = 0, ..., N_x$ and $n = 0, ..., N_t$. After replacing the derivatives with centered differences, we arrive at the algebraic version of the PDE, written in operator notation

$$[D_t D_t u = D_x q^{-x} D_x u + f]_i^n$$

[where q^{-x} means that the variable coefficient is approximated by an arithmetic mean. Now it remains to solve this equation for u_i^{n+1}:

$$u_i^{n+1} = -u_i^{n-1} + 2u_i^n + \left(\frac{\Delta t}{\Delta x}\right)^2 \left(\frac{1}{2}(q_i + q_{i+1})(u_{i+1}^n - u_i^n) - \frac{1}{2}(q_i + q_{i-1})(u_i^n - u_{i-1}^n)\right) + \Delta t^2 f_i^n$$

The initial conditions are

$$u(x,0) = I(x), \quad u_t(x,0) = V(x)$$

Boundary conditions

Neumann boundary conditions for a 1D domain are

$$\frac{\partial}{\partial n}\Big|_{x=L} = \frac{\partial}{\partial x}, \quad \frac{\partial}{\partial n}\Big|_{x=0} = -\frac{\partial}{\partial x}$$

Using a centered difference at the boundaries:

$$[D_{2x} u]_i^n = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} = 0, u_{i+1}^n = u_{i-1}^n$$

leads to the following formula for computing u_i^n at $i = N_x$:

$$u_i^{n+1} = -u_i^{n-1} + 2u_i^n + \left(\frac{\Delta t}{\Delta x}\right)^2 2q_{i-\frac{1}{2}}(u_{i-1}^n - u_i^n) + \Delta t^2 f_i^n$$

where we have assumed that $dq/dx = 0$ so that $q_{i+1} = q_{i-1}$ and $q_{i+\frac{1}{2}} = q_{i-\frac{1}{2}}$. The latter, and $u_{i+1}^n = u_{i-1}^n$ implies that the formula at $i = 0$ is the same. We have also done the approximation $q_{i+\frac{1}{2}} + q_{i-\frac{1}{2}} \approx 2q_i$.

The boundary scheme is implemented in the following way in my program

```
i = Ix[0]
# Set boundary values
# 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*(q[i] + q[im1])*(u_1[im1] - u_1[i])  + \
        dt2*f(x[i], t[n])

i = Ix[-1]
im1 = i-1
ip1 = im1
u[i] = - u_2[i] + 2*u_1[i] + \
        C2*(q[i] + q[ip1])*(u_1[ip1] - u_1[i])  + \
        dt2*f(x[i], t[n])
```

Implementation

I have three main functions in my program. *solver* copmutes u_i^n for all time steps based on the initial conditions, the source term f and certain numerical parameters. *viz* is for visualization, but is in this case only used for storing the error at each time step. *viz* sends a function *user_action* to *solver* that, at each time step, summates $(u - u_i^n)^2$ for the whole mesh. This quantity is used in the third main function *converge_rates* to compute the L2 norm E of the error for all time steps:

$$E = \left(\Delta x \Delta t \sum_{n=0}^{N_t} \sum_{i=0}^{N_x} E \right)^{\frac{1}{2}}$$

In addition, the function *source_term* computes $f(x, t)$ using *sympy* and converts it to a python lambda function.

a)

I will now compute the convergence rate for the test problem $q = 1+(x-L/2)^4$, with $f(x,t)$ adapted such that the exact solution is $u(x,t) = cos(\pi x/L)cos(\omega t)$. For decreasing time steps Δt, the convergence rate can be found by comparing two consecutive experiments

$$r_{i-1} = \frac{ln(E_{i-1}/E_i)}{ln(\Delta t_{i-1}/\Delta t_i)}$$

Δt is halved for each run. r should converge to 2 for decreasing time steps because we have used centered differences in our scheme; these have errors that go like Δt^2. A convergence rate of 2 means that halving the time step reduces the error by a factor of 4.

Unfortunately, my result is not as expected. The error is small, but r doesn't behave as it should:

```
r: [1.16, 0.74, 3.29, 0.22, 0.06, 0.61, 2.01]
```

The last value is indeed almost 2, but that seems like a coincidence if you look at the preceding values. The convergence rates shouldnt oscillate like that. I have not been able to detect any errors in my program. I've also ensured that the stabiltiy criterion

$$\Delta x \leq \beta \frac{\Delta x}{max_{x\epsilon[0,L]}c(x)}$$

is satisfied.

(b)

Now, $q = cos(\pi x/L)$. The derivative of q is now zero, which was one of the assumptions we made when we derived the scheme above. This condition dit not hold in a). Again, r does not converge,

```
r: [1.11, 2.2, −0.18, 3.87, −0.26, −0.14, 1.19]
```

it even turns negative at some points, meaning that the error was larger for a smaller time step. This doesn't make sense, I therefore conclude that there must be an error in my script.

c)

The Neumann conditions can be discretized in different ways, now we'll use a one-sided difference: $u_i - u_{i-1} = 0$ at $i = N_x$ and $u_{i+1} - u_i = 0$ at $i = 0$. See *test_neumann_c.py* for implementation of these boundary conditions. Now we expect $r = 1$ because the one-sided difference has en error that goes like Δt^2, which indeed is the result:

```
r: [1.02, 1.17, 0.94, 1.04, 1.01, 1.01, 0.95]
```

d)

A third way to discretize the Neumann conditions is to use ghost cells, i.e. extend the mesh with two points. See *test_neumann_c.py* for details.