

INF5620
AUTUMN 2015

Finite difference simulation of 2D waves

John-Anders Stende

Date: October 13, 2015

Abstract

This project is a finite difference simulation of 2D waves with Neumann boundary conditions, damping and variable wave velocity. It is split into three parts: 1) Discretization of the wave equation and implementation of solver 2) Verification: Constant solution, 1d plug-wave, standing undamped waves, manufactured solution and convergence rate 3) Investigation of a specific physical problem

Discretization and implementation

To start with, I will present the complete mathematical problem and show how the equations can be discretized and implemented in a python script.

Mathematical problem

The project addresses the two-dimensional, standard, linear wave equation with damping

$$\frac{\partial^2 u}{\partial t^2} + b \frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left(q(x, y) \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(q(x, y) \frac{\partial u}{\partial y} \right) + f(x, y, t) \quad (1)$$

The associated Neumann boundary condition is

$$\frac{\partial u}{\partial n} = 0 \quad (2)$$

in a rectangular spatial domain $\Omega = [0, L_x] \times [0, L_y]$. The initial conditions are

$$u(x, y, 0) = I(x, y) \quad (3)$$

$$u_t(x, y, 0) = V(x, y) \quad (4)$$

Discretization

Using central differences for all derivatives,

$$[D_t D_t u + b D_{2t} = D_x q^{-x} D_x u + D_y q^{-y} D_y u + f]_{i,j}^n \quad (5)$$

where q^{-x} is notation for the arithmetic mean of q .

This leads to the following equation for the interior spatial mesh points,

$$u_{i,j}^{n+1} = (1 + \frac{1}{2} b \Delta t)^{-1} \left[2u_{i,j}^n + u_{i,j}^{n-1} \left(\frac{1}{2} b \Delta t - 1 \right) + \left(\frac{\Delta t}{\Delta x} \right)^2 d q d x + \left(\frac{\Delta t}{\Delta y} \right)^2 d q d y + \Delta t^2 f_{i,j}^n \right] \quad (6)$$

where

$$d q d x = \left[\frac{1}{2} (q_{i,j} + q_{i+1,j}) (u_{i+1,j}^n - u_{i,j}^n) - \frac{1}{2} (q_{i-1,j} + q_{i,j}) (u_{i,j}^n - u_{i-1,j}^n) \right] \quad (7)$$

and

$$d q d y = \left[\frac{1}{2} (q_{i,j} + q_{i,j+1}) (u_{i,j+1}^n - u_{i,j}^n) - \frac{1}{2} (q_{i,j-1} + q_{i,j}) (u_{i,j}^n - u_{i,j-1}^n) \right] \quad (8)$$

The discretized initial condition for u_t

$$D_{2t} u_{i,j} = \frac{u_{i,j}^1 - u_{i,j}^{-1}}{2 \Delta t} = V_{i,j} \quad (9)$$

yields

$$u_{i,j}^{-1} = u_{i,j}^1 - 2 \Delta t V_{i,j} \quad (10)$$

This equality is used to obtain the scheme for the first step ($n = 0$)

$$u_{i,j}^1 = u_{i,j}^0 - \frac{1}{2} \Delta t V_{i,j} (b \Delta t - 2) + \left(\frac{\Delta t}{\Delta x} \right)^2 d q d x + \left(\frac{\Delta t}{\Delta y} \right)^2 d q d y + \frac{1}{2} \Delta t^2 f_{N_x,j}^0 \quad (11)$$

We have Neumann boundary conditions all along the rectangular spatial mesh described above. Assuming $\partial q / \partial x = \partial q / \partial y = 0$ at the boundaries, the scheme along the line $i = N_x, j = [0, N_y]$ becomes

$$u_{N_x,j}^{n+1} = -u_{N_x,j}^{n-1} + 2u_{N_x,j}^n + \left(\frac{\Delta t}{\Delta x}\right)^2 2q_{N_x-\frac{1}{2},j}(u_{N_x-1,j}^n - u_{N_x,j}^n) + \quad (12)$$

$$\left(\frac{\Delta t}{\Delta y}\right)^2 2q_{N_x,j-\frac{1}{2}}(u_{N_x,j-1}^n - u_{N_x,j}^n) + \Delta t^2 f_{N_x,j}^n \quad (13)$$

The corresponding equations for the other sides of the rectangular mesh is obtained by using the boundary relations

$$u_{i-1,j}^n = u_{i+1,j}^n, \quad u_{i,j-1}^n = u_{i,j+1}^n \quad (14)$$

$$q_{i-\frac{1}{2},j} = q_{i+\frac{1}{2},j}, \quad q_{i,j-\frac{1}{2}} = q_{i,j+\frac{1}{2}} \quad (15)$$

The boundary scheme (along the line $i = N_x, j = [0, N_y]$) for the first step is obtained in the same way as for the inner spatial points above,

$$u_{N_x,j}^1 = -u_{N_x,j}^0 + \frac{1}{2}\Delta t V_{N_x,j}(b\Delta t - 2) + \left(\frac{\Delta t}{\Delta x}\right)^2 q_{N_x-\frac{1}{2},j}(u_{N_x-1,j}^n - u_{N_x,j}^n) + \quad (16)$$

$$\left(\frac{\Delta t}{\Delta y}\right)^2 q_{N_x,j-\frac{1}{2}}(u_{N_x,j-1}^n - u_{N_x,j}^n) + \Delta t^2 f_{N_x,j}^n \quad (17)$$

Implementation

I have implmented both a scalar and a vectorized version of the above schemes in my solver. The equation for the inner spatial points can be reused at the boundaries if the indicies are changed, here shown for the line $i = N_x, j = [0, N_y]$

```
i = Ix[0]
ip1 = i+1
im1 = ip1
for j in Iy[1:-1]:
    dqdx = 0.5*(q[i,j] + q[ip1,j])*(u_1[ip1,j] - u_1[i,j]) - \
            0.5*(q[i,j] + q[im1,j])*(u_1[i,j] - u_1[im1,j])
    dqdy = 0.5*(q[i,j] + q[i,j+1])*(u_1[i,j+1] - u_1[i,j]) - \
            0.5*(q[i,j] + q[i,j-1])*(u_1[i,j] - u_1[i,j-1])
    u[i,j] = (1./(1+D))*(2*u_1[i,j] + u_2[i,j]*(D-1) + \
            Cx2*dqdx + Cy2*dqdy + dt2*f(x[i], y[j], t[n]))
```

The vecorized version is

```
i = Ix[0]
ip1 = i+1
im1 = ip1
dqdx = 0.5*(q[i,1:-1] + q[ip1,1:-1])*(u_1[ip1,1:-1] - u_1[i,1:-1]) - \
        0.5*(q[i,1:-1] + q[im1,1:-1])*(u_1[i,1:-1] - u_1[im1,1:-1])
dqdy = 0.5*(q[i,1:-1] + q[i,2:])*(u_1[i,2:] - u_1[i,1:-1]) - \
        0.5*(q[i,1:-1] + q[i,:-2])*(u_1[i,1:-1] - u_1[i,:-2])
u[i,1:-1] = (1./(1+D))*(2*u_1[i,1:-1] + u_2[i,1:-1]*(D-1) + \
        Cx2*dqdx + Cy2*dqdy + dt2*f_a[i,1:-1])
```

Verification

It's important to implement methods that can verify that the code is correct. In this project I'm going to assert that the wave stays constant if the initial condition is a constant and that a wave-plug splits in two and meet again at the same spot it started from. I'm also going to find the convergence rate of the discretization for standing, undamped waves and a manufactured solution.

Constant solution

$u(x, y, t) = c$ is a solution to the PDE problem if the source term f equals zero due to the fact that the derivative of a constant with respect to any variable is zero. To construct a test case, I thus set $I(x, y) = c$ and $V(x, y) = f(x, y, t) = 0$. Any value/expression can be chosen for b and $q(x, y)$ because they are both multiplied with zero when the constant solution is inserted in the PDE. This test case is implemented in the function `test_constant_solution`. Running a `pytest` verifies that u does indeed stay constant:

```
[fenics@ubuntu64-box] ../wave_project > py.test -v 2dwave_project.py
===== test session starts =====
platform linux2 -- Python 2.7.6 -- py-1.4.20 -- pytest-2.5.2 -- /usr/bin/python
collected 2 items

2dwave_project.py:718: test_constant_solution PASSED
2dwave_project.py:745: test_plug PASSED

===== 2 passed in 0.14 seconds =====
```

Five bugs in the implementation of the mathematical formulas:

- 1) Changing the sign of $dqdx$
- 2) Changing the sign of the source term f
- 3) Changing the sign of the solution u_1 at the preceding step
- 4) Changing indices of the q array
- 5) Multiply u_2 (solution at $t - 2dt$) with a factor 10

Only bug 3) makes the pytest fail because of an assertion error. This is because all the other terms in the scheme amount to zero when $u(x, y, t) = c$.

All the finite differences in the discrete equation are zero when u is constant, i.e. $u(x, y, t) = c$ is also a solution to the discrete equations.

Exact 1d plug-wave solution in 2d

There is a *test_plug* function in my script for simulating the propagation of a plug wave, where $I(x, y)$ is constant in some region of the domain and zero elsewhere. With unit Courant number, the plug is split into two identical waves, moving in opposite direction, exactly one cell per time step. After one period the two waves should meet back at the same place.

Running a nosetest:

```
[fenics@ubuntu64-box] ../wave_project > nosetests -v 2dwave_project.py
Test that u stays constant if I(x,y) = c. ... ok
Check that plug wave splits in two and meet back at the same place. ... ok
```

```
Ran 2 tests in 0.006s
```

OK

Standing, undamped waves