INF5620

Autumn 2015

Finite difference simulation of 2D waves

John-Anders Stende

Date: October 14, 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

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

The associated Neumann boundary condition is

$$\frac{\partial u}{\partial n} = 0 \tag{2}$$

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

$$u(x,y,0) = I(x,y) \tag{3}$$
$$u_t(x,y,0) = V(x,y) \tag{4}$$

Discretization

Using central differences for all derivates,

$$[D_t D_t u + bD_{2t} = D_x q^{-x} Dxu + D_y q^{-y} Dyu + f]^n_{i,j} \tag{5}$$

where q^{-x} is notation for the arithmetic mean of q.
This leads to the following equation for the interior spatial mesh points,

$$u^{n+1}_{i,j} = (1 + \frac{1}{2}b\Delta t)^{-1}\left[2u^n_{i,j} + u^{n-1}_{i,j}\left(\frac{1}{2}b\Delta t - 1\right) + \left(\frac{\Delta t}{\Delta x}\right)^2 dqdx + \left(\frac{\Delta t}{\Delta y}\right)^2 dqdy + \Delta t^2 f^n_{i,j}\right] \tag{6}$$

where

$$dqdx = \big[\frac{1}{2}(q_{i,j} + q_{i+1,j})(u^n_{i+1,j} - u^n_{i,j}) - \frac{1}{2}(q_{i-1,j} + q_{i,j})(u^n_{i,j} - u^n_{i-1,j})\big] \tag{7}$$

and

$$dqdy = \big[\frac{1}{2}(q_{i,j} + q_{i,j+1})(u^n_{i,j+1} - u^n_{i,j}) - \frac{1}{2}(q_{i,j-1} + q_{i,j})(u^n_{i,j} - u^n_{i,j-1})\big] \tag{8}$$

The discretized initial condition for u_t

$$D_{2t}u_{i,j} = \frac{u^1_{i,j} - u^{-1}_{i,j}}{2\Delta t} = V_{i,j} \tag{9}$$

yields

$$u^{-1}_{i,j} = u^1_{i,j} - 2\Delta t V_{i,j} \tag{10}$$

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

$$u^1_{i,j} = u^0_{i,j} - \frac{1}{2}\Delta t V_{i,j}(b\Delta t - 2) + \left(\frac{\Delta t}{\Delta x}\right)^2 dqdx + \left(\frac{\Delta t}{\Delta y}\right)^2 dqdy + \frac{1}{2}\Delta t^2 f^0_{N_x,j} \tag{11}$$

We have Neumann boundary condtions 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

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

$$\left(\frac{\Delta t}{\Delta y}\right)^2 2q_{N_x,j-\frac{1}{2}}(u^n_{N_x,j-1} - u^n_{N_x,j}) + \Delta t^2 f^n_{N_x,j} \tag{13}$$

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

$$u^n_{i-1,j} = u^n_{i+1,j}, \qquad u^n_{i,j-1} = u^n_{i,j+1} \tag{14}$$

$$q_{i-\frac{1}{2},j} = q_{i+\frac{1}{2},j}, \qquad q_{i,j-\frac{1}{2}} = q_{i,j+\frac{1}{2}} \tag{15}$$

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,

$$u^1_{N_x,j} = -u^0_{N_x,j} + \frac{1}{2}\Delta t V_{N_x,j}(b\Delta t - 2) + \left(\frac{\Delta t}{\Delta x}\right)^2 q_{N_x-\frac{1}{2},j}(u^n_{N_x-1,j} - u^n_{N_x,j}) + \tag{16}$$

$$\left(\frac{\Delta t}{\Delta y}\right)^2 q_{N_x,j-\frac{1}{2}}(u^n_{N_x,j-1} - u^n_{N_x,j}) + \Delta t^2 f^n_{N_x,j} \tag{17}$$

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 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 indicies 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 som 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

With no damping b and constant wave velocity c, our wave equation without any source term admits a standing wave solution:

$$u_e(x, y, t) = Acos(k_x x)cos(k_y y)cos(\omega t), \quad k_x = \frac{m_x \pi}{L_x}, \quad \frac{m_y \pi}{L_y} \tag{18}$$

This solution can be used to test the convergence rate of the numerical method. To do this, I compute the L2 norm E of the error $e^n_{i,j} = u_e(x_i, y_j, t_n) - u^n_{i,j}$ at the last time step:

$$E = \Delta x \Delta y \left(\sum_{i=0}^{N_x} \sum_{j=0}^{N_y} e^2 \right)^{\frac{1}{2}}$$

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 this particular discretization of the wave equation. Function *standing_undamped_waves* computes r, and we see that it does indeed converge to $r = 2$:

```
r: [2.06, 2.16, 2.1, 2.06, 2.03, 2.02, 2.01]
```

Manufactured solution

The goal now is to adapt the source term $f(x, y, t)$ such that

$$u_e(x, y, t) = (Acos(\omega t) + Bsin(\omega t))e^{-ct}cos(k_x x)cos(k_y y) \tag{19}$$

is a solution to the PDE with damping and variable wave velocity. In function $manufactured\_solution$ I use sympy to calculate the $f(x, y, t)$, $V(x, y)$ and $u_e(x, y, t)$ for $c = b/2$ and $\omega = \sqrt{k_x^2 q + k_y^2 q - c^2}$ where $q = q(x, y)$ is the variable wave velocity squared. Then I calculate the convergence rates in excactly the same way as above. The results are not as expected:

```
E: [1.4393103545680261, 0.64655583544484407, 0.36979546185606887, 0.27723181301153799, 0.247
r: [0.52, 1.15, 0.81, 0.42, 0.16, 0.06]
```

Even though the error isn't that bad, r doesn't converge to a specific value. I think this has something to do with the way I've defined ω (when I animate, u_e evolves faster than u in time), but I have not been able to come up with a better expression.

Investigate a physical problem

The purpose of this part is to explore what happens to a wave that enters a medium with different wave velocity. To do this, I'm going to simulate a wave with initial form $I(x, y)$ propagating over a subsea hill $B(x, y)$. Now, $q = gH(x, y)$ where g is the acceleration of gravity and $H(x, y) = I(x, y) - B(x, y)$ is the stillwater depth.

The initial surface is taken as a smooth Gaussian function

$$I(x; I_0, I_a, I_m, I_s) = I_0 + I_a exp\left(-\left(\frac{x - I_m}{I_s}\right)^2\right) \tag{20}$$

I_m is location of peak, I_s is the width.

Three different bottom shapes shall be investigated (see code for details):
1) 2D Gaussian hill
2) Cosine hat function
3) Box

My results are presented as gifs.