INF 5620
Waveproject

Andreas Slyngstad

12. oktober 2015

1 The mathematical problem

In this project we are to take a closer look on a linear wave equation with damping

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

The spatial domain is given as $\Omega = [0, Lx] \times [0, Ly]$ in the timedomain $t \in (0, T)$ with the following initial and boundary conditions:

$$u(x,y,0) = I(x,y) \tag{2}$$
$$u_t(x,y,0) = V(x,y) \tag{3}$$
$$\frac{\partial u}{\partial n} = 0 \tag{4}$$

1.1 Discretization

For a certain position i space and time, we discretize $u(x,y,t) = u^n_{i,j}$ where i,j represents a certain point in the spatial domain Ω, while n represents a certain time within the domain t.

- To discretize the PDE probelm we have to take a closer look at the wave velocity with respect to the derivate in x and y. Introducing ϕ such that $\phi = q(x)\frac{\partial u}{\partial x}$, and using centered differentiation on this parameter we end up with

$$\Big[\frac{\partial \phi}{\partial x}\Big]^n_{i,j} \approx \frac{\phi_{i+\frac{1}{2},j} - \phi_{i-\frac{1}{2},j}}{\Delta x}$$

  Using the relation for ϕ we get

$$\phi_{i+\frac{1}{2},j} = q_{i+\frac{1}{2}}\Big[\frac{\partial u}{\partial x}\Big]^n_{i+\frac{1}{2},j} \approx q_{i+\frac{1}{2},j}\frac{u^n_{i+1,j} - u^n_{i,j}}{\Delta x}$$
$$\phi_{i-\frac{1}{2},j} = q_{i-\frac{1}{2}}\Big[\frac{\partial u}{\partial x}\Big]^n_{i-\frac{1}{2},j} \approx q_{i-\frac{1}{2},j}\frac{u^n_{i,j} - u^n_{i-1,j}}{\Delta x}$$
$$q_{i+\frac{1}{2},j} = \frac{q_{i,j} + q_{i+1,j}}{2} \qquad q_{i-\frac{1}{2},j} = \frac{q_{i,j} + q_{i-1,j}}{2}$$
$$\Big[\frac{\partial}{\partial x}\big(q(x)\frac{\partial u}{\partial x}\big)\Big]^n_{i,j} \approx \frac{1}{\Delta x^2}\Big(\frac{1}{2}(q_{i,j} + q_{i+1,j})(u_{i+1,j} - u^n_{i,j}) - \frac{1}{2}(q_{i,j} + q_{i-1,j})(u^n_{i,j} - u^n_{i-1,j})\Big)$$

  Where arithmetic mean has been used to evaluate $q_{i\pm\frac{1}{2}}$. Equivalent results can be found for the derivation with respect to y.

For the derivation with respect to time t, the discretization is more or less stright forward using centered difference.

$$\frac{\partial^2 u}{\partial t^2} \approx \frac{u_{i,j}^{n+1} - 2u_{i,j}^{n} + u_{i,j}^{n-1}}{\Delta t^2} \qquad \frac{\partial u}{\partial t} \approx \frac{u_{i,j}^{n+1} - u_{i,j}^{n-1}}{2\Delta t}$$

Finally rewriting the discretized PDE with respect to $u_{i,j}^{n+1}$ we end up with the following scheme

$$u_{i,j}^{n+1} = \frac{1}{1+b\Delta t}\Bigg[u_{i,j}^{n-1}(b\Delta t - 1) + 2u_{i,j}^{n}$$
$$\frac{1}{2}\Big(\frac{\Delta t}{\Delta x}\Big)^2\Big((q_{i,j} + q_{i+1,j})(u_{i+1,j} - u_{i,j}^{n}) - (q_{i,j} + q_{i-1,j})(u_{i,j}^{n} - u_{i-1,j}^{n}) +$$
$$\frac{1}{2}\Big(\frac{\Delta t}{\Delta y}\Big)^2\Big((q_{i,j} + q_{i,j+1})(u_{i,j+1} - u_{i,j}^{n}) - (q_{i,j} + q_{i,j-1})(u_{i,j}^{n} - u_{i,j-1}^{n}) + fi, j^{n}\Bigg]$$

- Considering the first timestep for n = 0, we face a problem calculating the $u_{i,j}^{n-1}$ term due to the fact that this point lies outside the meshgrid. This can be solved by the centered discretization of the initial velocity condition. We get the following scheme for the first step.

$$\frac{\partial u}{\partial t}(x, y, 0) = V(x, y) \qquad \frac{u_{i,j}^{1} - u_{i,j}^{-1}}{2\Delta t} = V_{i,j} \qquad u_{i,j}^{-1} = u_{i,j}^{1} - 2\Delta t V_{i,j}$$

$$u_{i,j}^{n+1} = \frac{1}{2}\Big(\frac{1}{1+b\Delta t}\Big)\Bigg[-2\Delta t V_{i,j}(b\Delta t - 1) + 2u_{i,j}^{n} +$$
$$\frac{1}{2}\Big(\frac{\Delta t}{\Delta x}\Big)^2\Big((q_{i,j} + q_{i+1,j})(u_{i+1,j} - u_{i,j}^{n}) - (q_{i,j} + q_{i-1,j})(u_{i,j}^{n} - u_{i-1,j}^{n})\Big) +$$
$$\frac{1}{2}\Big(\frac{\Delta t}{\Delta y}\Big)^2\Big((q_{i,j} + q_{i,j+1})(u_{i,j+1} - u_{i,j}^{n}) - (q_{i,j} + q_{i,j-1})(u_{i,j}^{n} - u_{i,j-1}^{n})\Big) + fi, j^{n}\Bigg]$$

- I have chosen to implement ghost points outside the region in my numerical scheme. This handy tool makes it possible to use the numerical scheme on the boundary, using a ghost node outside the original domain. This ghost node is updated at each step utilizing the centered differentiated boundary condition on the boundary, with respect to either x or y depending which boundary we are on. Say we stand on the end of boundary x, the relation becomes

$$\frac{\partial u}{\partial n} = 0 \qquad \frac{u_{Lx+1,j}^{n+1} - u_{Lx-1,j}^{n+1}}{2\Delta x} = 0 \qquad u_{Lx-1,j}^{n+1} = u_{Lx+1,j}^{n+1}$$

2 Verification

2.1 Constant Solution

To verify the numerical implementation we use a test case for constant solution $u(x, y, t) = c$. By direct insertion in our PDE we simply get $f(x, y, t) = 0, V(x, y) = 0, I(x, y) = c$, while q(x,y) and b can be chosen arbitrary. This can be verified in the numerical scheme aswell

$$\frac{\partial^2 u}{\partial t^2} \approx \frac{c^{n+1} - 2c^n + c^{n-1}}{\Delta t^2} = 0 \qquad \frac{\partial u}{\partial t} \approx \frac{c^{n+1} - c^{n-1}}{2\Delta t} = 0$$

Similar results can be shown in the space derivate aswell.

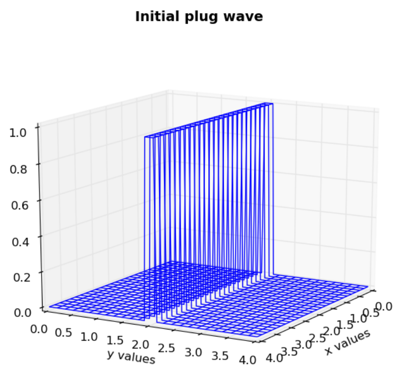
To verify the this condition I have chosen to use unittest to check these conditions. I have implemented this class to test the conditions, and raise and assertError if some condition is not met.

```
class TestMySolver(unittest.TestCase):

        def setUp(self):
                self.prog = main()
                self.version = "vectorized"
                self.plot = False

        def test_constant_solution(self):
                x,y,t= sym.symbols('x,y,t'); constant = 1
                f = sym.lambdify((x,y,t), symbolicPDE(constant,x*y)
    )
                randval=np.random.randint(10,size=3)
                x = randval[0]; y = randval[1]; t = randval[2];
                error = int(self.prog.const("vectorized",False))
                #Check f = 0
                self.assertEqual(0,f(x,y,t))
                #Check Numerical solution stays constant
                self.assertEqual(0,error)
                #Check I = c
                self.assertEqual(constant,self.prog.I(x,y))
```

2.2 Plugwave

In this part of the exercise we are to take a closer look on a 1d plug wave. This wave has an initial condition such that the wave has a constant value in some region and zero elsewhere. In this case we set $c\frac{\Delta t}{\Delta x} = 1$ We get some satisfying results, provided in the animation plugwafe.gif

Figur 1: Initial condition of a Plug wave



file provided. I also made a test using unittest, to verify some conditions plugwave must fulfill

```
        def test_plug_solution(self):
                print "Will␣raise␣stability_limit␣error"
                self.prog.plug("vectorized",False)
                dt = self.prog.dt
                dx = self.prog.dx
                dy = self.prog.dy
                q = self.prog.q
                condx = np.sqrt(q(1,1))*dt/dx
                condy = np.sqrt(q(1,1))*dt/dy
                #Check wave velocitycondition
                self.assertEqual(1,condx)
                self.assertEqual(1,condy)
```

2.3 Standing wave

Looking at the PDE problem under the assumption $b = 0$ (no damping), constant wave velocity c and no sourceterm, we have an analytical solution.

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

By putting the analytical solution into our PDE problem, we can find ω as a function of q, kx and ky

$$\omega = \sqrt{q(k_x^2 + k_y^2)}$$

The visualization of the numerical solution is provided in the animation standing.gif, where the exact solution is plotted with the numerical solution.

Since we have a analytical solution for this event, we can calculate the convergence rate of the numerical solution. Presenting a parameter h such that $\frac{dx}{Nx}$ and $dt = \frac{h}{10}$ and halfing h for each run in the convergence rates method. For certain dt we get

```
For dt = 0.1000, the convergence rate is 0.0147
For dt = 0.0500, the convergence rate is 0.5449
For dt = 0.0250, the convergence rate is 1.8945
For dt = 0.0125, the convergence rate is 1.9981
For dt = 0.0063, the convergence rate is 2.0009
```

Which are some satisfying results for the scheme, since we want this to be 2.

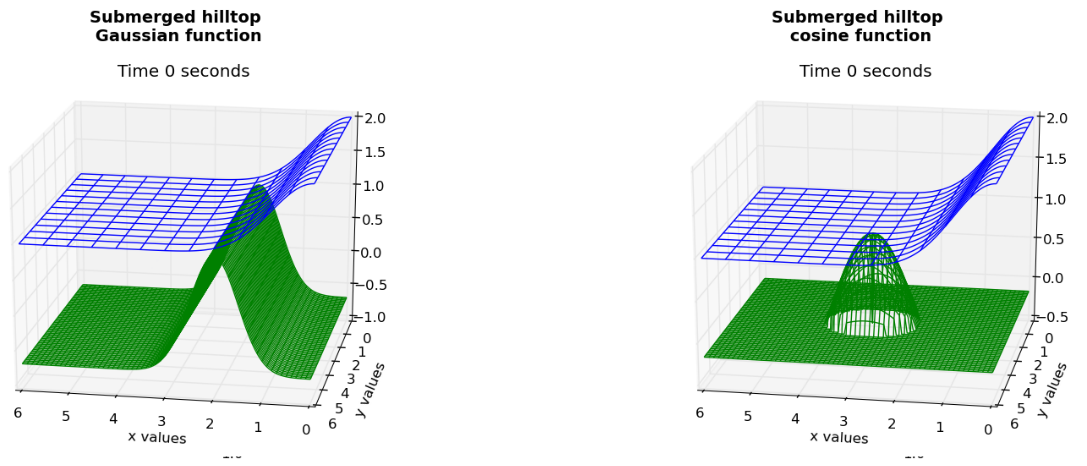
2.4 Manufactured solution

I made a example code for this, but sympy takes to long to calculate ω for it to be practical...

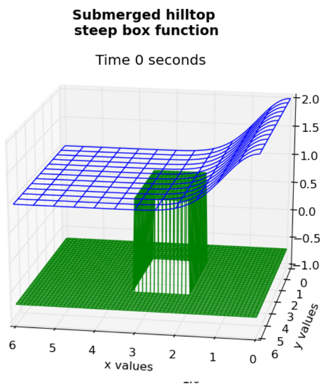
3 Investigate a physical problem

We are presented with a problem to evaluate a wave as it enters a medium with different wave velocity. We look at this problem physically as a wave in the ocean, passing a submerged geometry close to the still surface. The wave velocity is given as $q(x, y) = gH(x, y)$ where g is the acceleration of gravity, and $H(x, y)$ is the stillwater depth. We are presented with three models to represent the subsurface, a smooth Gaussian function, a cosine function and a steep constant function.

Figur 2: Initial condition of smooth Gaussian represented subsurface



Figur 3: Initial condition of box subsurface



An animation for each submurged profile is provided with the report, wavedump.gif, wavecos.gif and wavebox.gif Looking at the movies, it seems that we have some decent results. In the box-simulation we would expect some numerical scheme issues due to the sudden leap in geometry. As long as the geometry isn't to close to the surface the calculation goes well, but if the geometry is set to close we run into some trouble. Check out boxwavefails.png to get a visualisation of this.