INF5620 - Numerical methods for partial differential equations

Project 1

Even Marius Nordhagen

September 26, 2017

- For the Github repository containing programs and results, follow this link: `https://github.com/UiO-INF5620/INF5620-evenmn/tree/master/project_1`
- All the tests and the physical wave solution are run by *main.py*

1 Introduction

The aim of this project is to use the two-dimensional standard wave equation with damping,

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

to investigate a physical problem where a wave enters a medium with different wave velocity. In many cases we do not have any exact wave solution, so we need to ensure that our implementation is correct before we solve those problems. To verify the implementation we study the outcome from 4 cases where we know the exact solution. These verification methods are described in "Theory and methods". After we are convinced that the implementation is correct, we will study how a wave in a fluid is affected by the bottom shape. With this we can for instance estimate the shape of a potential tsunami wave when we know the bottom shape. We will look at a Gaussian hill-shaped bottom, a so-called "cosine-hat"-shaped bottom and a box-shaped bottom, and we expect the wave to be less smooth for less smooth bottoms. The results are presented in "Results", with a conclusion given in "Conclusion".

2 Theory and methods

To derive the discrete set of equations we use several approximations. First we use the three point formula to discretize the second derivative. Thereafter we could use the two point formula to discretize $\partial u/\partial t$, but it makes the equation quite messy, and I will therefore use the simplest (but not the best) formula for the derivative: $[\partial u/\partial t]^n_{i,j} \approx (u^n_{i,j} - u^{n-1}_{i,j})/\Delta t$. Further the spatial derivations can be approximated as

$$\left[\frac{\partial}{\partial x}\left(q(x,y)\frac{\partial u}{\partial x}\right)\right]^n_{i,j} \approx \frac{1}{\Delta x^2}\Big(q_{i+1/2,j}(u^n_{i+1,j}-u^n_{i,j}) - q_{i-1/2,j}(u^n_{i,j}-u^n_{i-1,j})\Big) \quad (2)$$

and similar in y-direction, and we end up with the general discrete equation

$$\begin{aligned} u^{n+1}_{i,j} &= 2u^n_{i,j} - u^{n-1}_{i,j} - b\Delta t(u^n_{i,j} - u^{n-1}_{i,j}) \\ &+\left(\frac{\Delta t}{\Delta x}\right)^2\Big(q_{i+1/2,j}(u^n_{i+1,j} - u^n_{i,j}) - q_{i-1/2,j}(u^n_{i,j} - u^n_{i-1,j})\Big) \\ +\left(\frac{\Delta t}{\Delta y}\right)^2&\Big(q_{i,j+1/2}(u^n_{i,j+1} - u^n_{i,j}) - q_{i,j-1/2}(u^n_{i,j} - u^n_{i,j-1})\Big) + \Delta t^2 f^n_{i,j}. \end{aligned} \quad (3)$$

We can only use this formula to find a point where we know the two previous time points, so we need a special formula for $n = 1$. For that we need the a relation from the problem description

$$\frac{u^1_{i,j} - u^{-1}_{i,j}}{2\Delta t} \approx V_{i,j}. \quad (4)$$

We can use this twice if we go back and change to the two point formula $[\partial u/\partial t]^n_{i,j} \approx (u^{n+1}_{i,j} - u^{n-1}_{i,j})/2\Delta t$ only for this special case, and we obtain

$$\begin{aligned} u^1_{i,j} = u^0_{i,j} + \left(1 - \frac{1}{2}b\Delta t\right)\Delta t V_{i,j} + \frac{1}{2}\Bigg[&\left(\frac{\Delta t}{\Delta x}\right)^2\Big(q_{i+1/2,j}(u^0_{i+1,j} - u^0_{i,j}) \\ &-q_{i-1/2,j}(u^0_{i,j} - u^0_{i-1,j})\Big) + \left(\frac{\Delta t}{\Delta y}\right)^2\Big(q_{i,j+1/2}(u^0_{i,j+1} - u^0_{i,j}) \\ &-q_{i,j-1/2}(u^0_{i,j} - u^0_{i,j-1})\Big) + \Delta t^2 f^0_{i,j}\Bigg]. \end{aligned} \quad (5)$$

For the half point steps $q_{i\pm1/2}$ we can use the arithmetic mean:

$$q_{i\pm1/2} \approx \frac{1}{2}(q_{i\pm1} + q_i). \quad (6)$$

2.1 Implementation - ghost cells

There are several ways to implement the code, but to keep the code neat and transparent, ghost cells are essential. The idea is to introduce a "layer" with non-physical cells outside the physical cells, which we can give values after all the physical cells for a fixed time n are calculated. For a two-dimensional case this is done by

$$u^n_{i-1,j} = u^n_{i+1,j} \quad \text{for } x = [0, Nx] \tag{7}$$

and similar in y-direction. The big advantage of this method is to get rid off the special formulas for the edges.

2.2 Verification methods

A test function never guarantees that the implementation is correct, but if several test function are running without any error message, it gives a good indication. We will now look at four verification tests.

2.2.1 Method 1: Constant solution

The simplest possible case is a constant wave function, $u(x, y, t) = c$, which gives $f = 0$ when inserting into equation (1). Furthermore $I = c$, $V = u'(0) = 0$, b can be an arbitrary constant and q can be an arbitrary function. Plug-in $u(x, y, t)$ into the discretized wave equation, and you will get

$$c = 2c - c + (5 \text{ terms that are zero}) + f(x, y, t) \tag{8}$$

which is true when the source term, $f(x, y, t)$, is zero. We observe that $u(x, y, t)$ should only consist of elements with value c at any time, and we can therefore implement a test which breaks if an element is not c.

2.2.2 Method 2: Exact 1D plug-wave solution in 2D

Another possible solution of the PDE is a pulse, which has a value in some region and zero value else at the beginning. The solver should split it up in two parts, and they will move in opposite directions. With an 1D-pulse at the beginning, we expect the pulse to only move in this direction, which makes an useful test. Set $b = 0$ and $q =$ constant.

2.2.3 Method 3: Standing undamped waves

The damping depends on the second term, so if we want to simulate a wave without damping we need to set $b = 0$. A simple three dimensional undamped wave can have the form

$$u(x, y, t) = A\cos(k_x x)\cos(k_y y)\cos(\omega t) \tag{9}$$

where k_x and k_y are the wave numbers in x- and y-direction respectively, A is the maximum amplitude and ω is the angular frequency. The wave numbers can be expressed by $k_i = m_i\pi/L_i$ where m_i is an arbitrary integer and L_i is the size in the current direction. The source term disappears, which reduces some potential errors. We will use this method to study the error and calculate the rate of convergence rather than making a test.

2.2.4 Method 4: Standing damped waves

A natural continuation is to study a standing damped wave. As you might guess, we need a $b \neq 0$ to find the numerical solution correctly. A standing damped wave can have the exact solution

$$u(x, y, t) = (A\cos(\omega t) + B\sin(\omega t))\exp(-ct)\cos(k_x x)\cos(k_y y) \tag{10}$$

where A, B and c are constants. In this case we need to calculate the source term, and I really recommend to use sympy for that (calculate it analytically takes forever). If you do that correctly, you will obtain an 8-term expression, which I will not express here. We also find

$$\begin{aligned} I(x, y) = u(x, y, 0) &= A\cos(k_x x)\cos(k_y y), \\ V(x, y) &= (\omega B - cA)\cos(k_x x)\cos(k_y y). \end{aligned} \tag{11}$$

We will test if the convergence rate approaches the expected limit.

2.3 Rate of convergence

Above I have mentioned the convergence rate a couple of times, but how do we compute it? The convergence rate is defined by

$$r = \frac{\log(E_{i+1}/E_i)}{\log(h_{i+1}/h_i)} \tag{12}$$

where E_i is the error of measurement i and h_i is the corresponding time step. As you might observe, we need a sequence of measurements to obtain the factor and it should approach a limit for more precise measurements (smaller time steps). In our case we should get

$$\lim_{\Delta t \to 0} r = 2. \tag{13}$$

2.4 Physical wave

We are going to study how a wave is affected by the bottom shape. The idea is that we start with an initial wave and use our program to simulate the propagation. The initial wave has the form

$$I(x) = I_0 + I_a \exp\left(-\left(\frac{x - I_m}{I_s}\right)^2 \right) \tag{14}$$

where I_0, I_a, I_m and I_s are constants. We look at bottom shapes with different perturbations; 2D Gaussian hill, Cosine-hat shape and a box shape.

2.4.1 2D Gaussian hill

The 2D Gaussian hill is a smooth perturbation without any sharp edges. It can be modelled as

$$B(x, y) = B_0 + B_a \exp\left(-\left(\frac{x - B_{mx}}{B_s}\right)^2 - \left(\frac{y - B_{my}}{bB_s}\right)^2 \right) \tag{15}$$

with B_0, B_a, B_2, B_{mx} and B_{mx} as constants a b as the scaling parameter (should not be confused with the damping constant). b controls the shape of the "ground surface" of the Gaussian perturbation: with $b = 1$ we get a circular "ground surface", and with $b \neq 1$ we get an elliptic "ground surface".

2.4.2 Cosine-Hat

Cosine hat is quite similar to the 2D Gaussian hill, but the peak is sharper. The formula is

$$B(x, y) = B_0 + B_a \cos\left(\pi \frac{x - B_{mx}}{2B_s}\right) \cos\left(\pi \frac{y - B_{my}}{2B_s}\right), \tag{16}$$

when $0 \leq \sqrt{x^2 + y^2} \leq B_s$ and $B = B_0$ outside this circle.

2.4.3 Box

The box is a perturbation with really sharp edges. It can be modelled as a non-smooth function which has value $B = B_0$ outside the box and $B = B_0 + B_a$ inside the box where the box is defined by $(B_{mx} - B_s, B_{mx} + B_s), (B_{my} - bB_s, B_{my} + bB_s)$.

3 Results

I decided to plot the waves for four times in the interval $T \in [0, 1]$, even though an animation would give a better visualization. All results are plotted with $dt = 0.0005$, $dx = dy = 0.01$ and y is constant for all of the plots ($y \approx 0.06$). We also have $I_0 = I_a = I_s = B_0 = B_a = B_s = 1.0$ and $I_m = B_{mx} = B_{my} = 0$.

3.1 2D Gaussian hill

I use $b = 1$ for the plots, i.e a circular ground surface. The plots of $T = 0.0$, $T = 0.3$, $T = 0.6$ and $T = 0.9$ are found in figure (1), (2), (3) and (4) respectively.

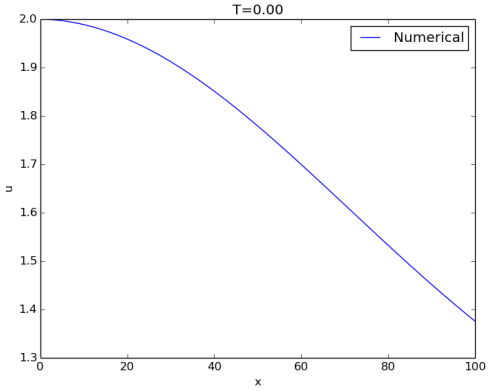

Figure 1: T=0.0

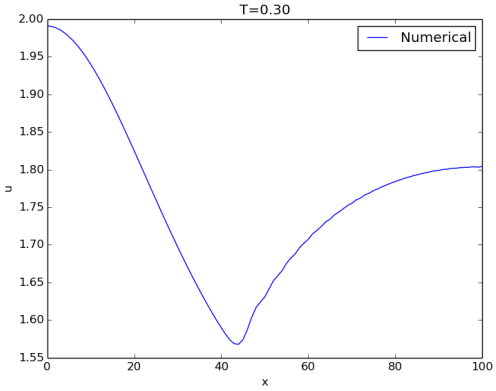

Figure 2: T=0.3

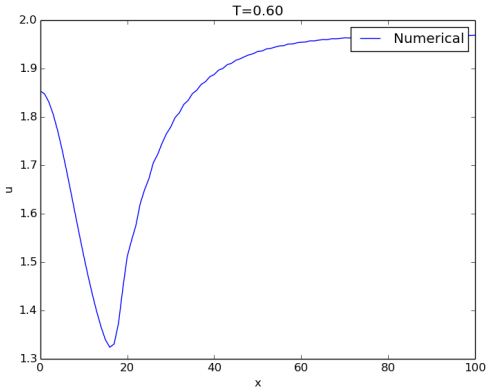

Figure 3: T=0.6

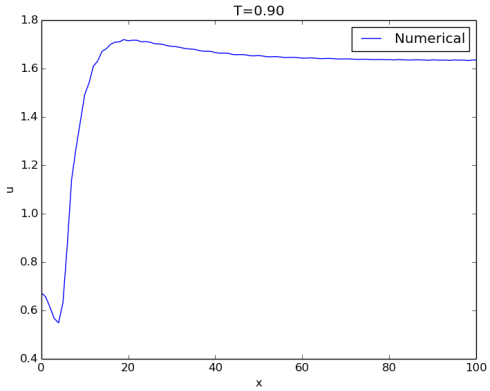

Figure 4: T=0.9

3.2 Cosine-Hat

Using the parameters presented above, we get the results found in (5), (6), (7) and (8) for $T = 0.0$, $T = 0.3$, $T = 0.6$ and $T = 0.9$ respectively.

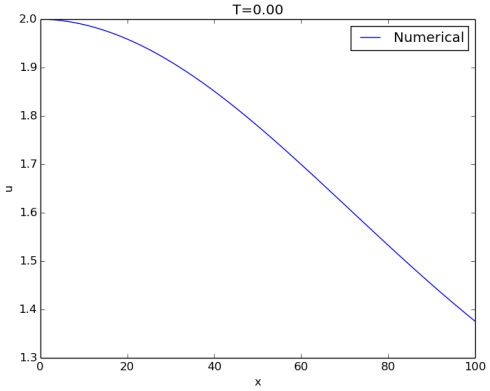

Figure 5: T=0.0

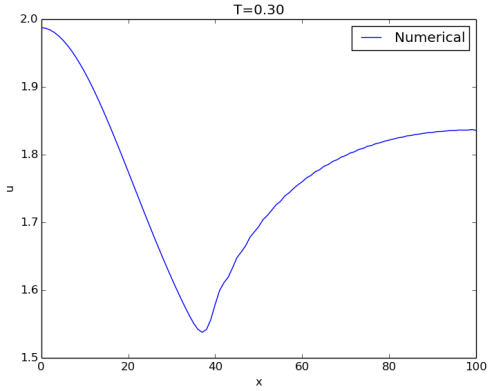

Figure 6: T=0.3

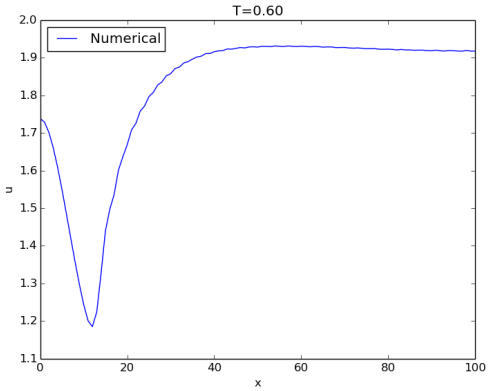

Figure 7: T=0.6

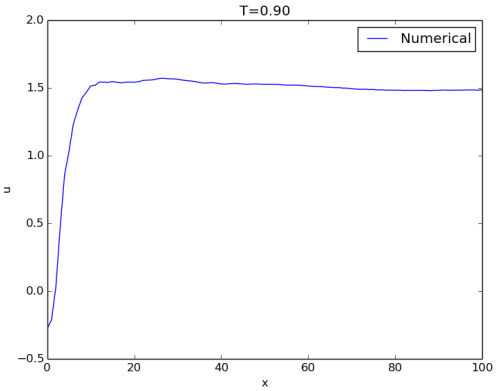

Figure 8: T=0.9

3.3 Box

For the box perturbation plots for $T = 0.0$, $T = 0.3$, $T = 0.6$ and $T = 0.9$ are found in figure (9), (10), (11) and (12) respectively.

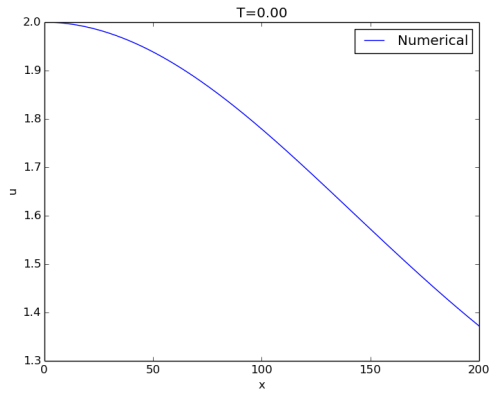

Figure 9: T=0.0

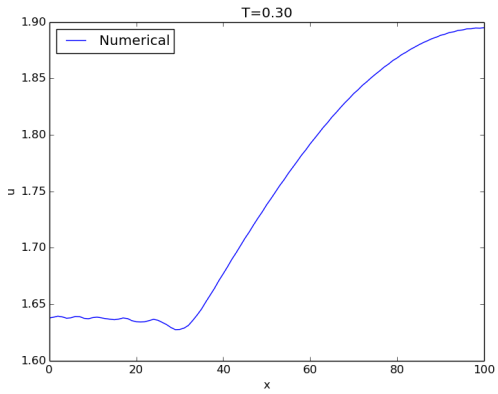

Figure 10: T=0.3

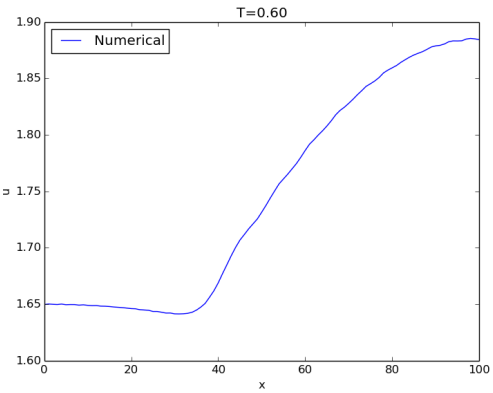

Figure 11: T=0.6

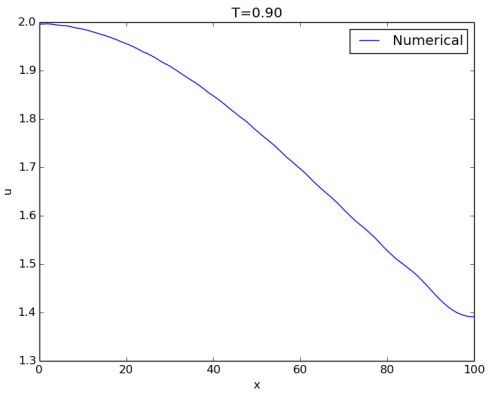

Figure 12: T=0.9

We observe that the 2D Gaussian hill and the Cosine-hat are quite similar, but not identical. The four plots for each bottom shape can give us a good picture of how the wave is changing per time for the two first perturbations. When it comes to the box perturbation, it is more difficult because there is apparently no correlation between the figures.

4 Discussion and conclusion

To make the different bottom shapes comparable, it is essential that we use the same parameters on all the cases. The 2D Gaussian hill and the Cosine-hat are similar simply because the perturbations are similar. Anyway if you

study the plots close, you will see that the Cosine-hat wave is slightly sharper than the 2D Gaussian hill wave. The reason for that is not surprising because the Cosine-hat perturbation is sharper. We have also seen that the box wave is changing pretty fast, and we need to have a lot more figures if we want to study how this wave is changing per time. This is related to the edges of the box, which make the fluid waves uneasy and unpredictable and corresponds to fast-moving waves.

What we have seen is that the sharper perturbations the faster and more aggressive waves, as expected. A wave that propagates over a small and smooth bottom object is not significant affected by the object, but with a big and sharp object the wave can totally change. This is a really interesting and important result, potentially it makes it possible to expose places where a tsunami can arise and create a tragedy.

Not everything went as it should, my convergence rates are really suspicious and converges against 1 (if it converges at all), and therefore my convergence test checks if $r \in [-1, 2]$ (I know it is wrong). The reason for that is that my numerical solution is wrong in all the cases, but especially for the standing damped wave, which tends to fluctuate much slower than it should (implies that ω is wrong). I did not succeed finding the bug, even though I spent lots of time debugging, but the error needs to be somewhere in the solver.

The present code produces a plot and a table containing the error and the convergence rate for both the standing damped wave and the standing undamped wave. One should in general not print and plot stuffs in tests, but I made an exception to show how the error degenerates. The code also produces a visualization of the physical wave (which obviously also has errors).