Second compulsory project: 2D wave equation

INF5620

2013

- Deadline: Oct 15
- Work in groups of two (or three if the amount of work is suitably extended)
- Make a directory `oblig2` in the top directory of your INF5620 repo on GitHub or Bitbucket to hold the various files of the project. Include a `README` file with a short overview of the different files and the full names of the students who did the project.
- Make reusable, flexible code.
- Write a short report summarizing the main results. LaTeX is probably the preferred format, but there are several other options[1] too.
- Note that the last part of this compulsory exericse allows different groups to go in different directions (visualization, high-performance computing, more advanced numerics).

1 Mathematical problem

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

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

with boundary condition

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

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

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

[1] `../writing_reports/index.html`

2 Discretization

Derive the discrete set of equations to be implemented in a program:

- the general scheme for computing $u_{i,j}^{n+1}$ at interior spatial mesh points,
- the modified scheme for the first step,
- the modified scheme at boundary points (first step and subsequent steps), unless you use the interior scheme also at the boundary with extra ghost cells.

3 Truncation error

Compute the truncation error of the scheme at an arbitrary interior mesh point.

4 Implementation

Implement the numerical scheme in a program. You may use `wave2D_u0.py`[2] as a starting point (this program solves the 2D wave equation with constant wave velocity and $u = 0$ on the boundary). You will need to include both scalar (pointwise) computation of the scheme for debugging and reference as well as a vectorized version for speed.

5 Verification: constant solution

Construct a test case with constant solution (not 0 or 1). Make a corresponding nose test.

Invent five types of possible bugs in the implementation of the mathematical formulas. See how many of them that lead to a wrong non-constant solution.

6 Verification: quadratic solution

Assume an exact solution on the form $u_{\mathrm{e}}(x, y, t) = X(x)Y(y)T(t)$ where X, Y, and T are polynomials of degree three or less. Construct X and Y such that the normal derivative vanishes at the four boundaries. Fit a corresponding source term $f(x, y, t)$ in the wave equation. Prove that this solution can also be an exact solution of the discretized wave equation if $b = 0$ (you need to check all the stencils involved). Implement a corresponding nose test.

7 Exact 1D plug-wave solution in 2D

The program `wave1D_dn_vc.py`[3] has a `pulse` function for simulating the propagation of a plug wave, where $I(x)$ 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. The discrete solution is then equal to the exact solution.

[2] `https://github.com/hplgit/INF5620/blob/gh-pages/src/wave/wave2D_u0/wave2D_u0.py`
[3] `https://github.com/hplgit/INF5620/blob/gh-pages/src/wave/wave1D/wave1D_dn_vc.py`

Test the 2D program using a one-dimensional plug wave in x direction with $c\Delta t/\Delta x = 1$ (the plug is constant in y direction and hence compatible with the $\partial/\partial y = 0$ boundary condition). Also propagate a one-dimensional plug wave in the y direction with $c\Delta t/\Delta y = 1$. Both test cases are essentially 1D test cases, and the results should as in the 1D case. Implement a corresponding nose test.

8 Verification: standing, undamped waves

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

$$u_{\mathrm{e}}(x,y,t) = A\cos(m_x x\pi/L_x)\cos(m_y y\pi/L_y)\cos(\omega t) \tag{5}$$

for arbitrary amplitude A, arbitrary integers m_x and m_y, and a suitable choice of ω. This solution can be used to test the convergence rate of the numerical method.

Use the truncation error analysis to set up a model for the error in terms of the discretization parameters. Note that the truncation error is not the true error $e^n_{i,j} = u_{\mathrm{e}}(x_i, y_j, t_n) - u^n_{i,j}$, but we assume that $e^n_{i,j}$ depends on the discretization parameters in the same way. We also assume that a norm of $e^n_{i,j}$, e.g.,

$$E = ||e^n_{i,j}||_{\ell^\infty} = \max_i \max_j \max_t |e^n_{i,j}|,$$

also depends on the discretization parameters in the same way. Introduce a common discretization parameter h such that Δt, Δx, and Δy are proportional to h. Show that $E = Ch^2$ for some C is the expected behavior of the error. Perform experiments with decreasing h, compute E, and verify that E/h^2 is approximately constant.

9 Verification: standing, damped waves

Try to find an analytical solution of damped waves using an ansatz of the type

$$u_{\mathrm{e}}(x,y,t) = (A\cos(\omega t) + B\sin(\omega t))\, e^{-bt}\cos(m_x x\pi/L_x)\cos(m_y y\pi/L_y)\,. \tag{6}$$

See if A, B, and ω can be adjusted such that (6) solves the PDE with no source term. Make a corresponding convergence test.

10 Verification: manufactured solution

Choose some $q(x,y) \neq 0$ and find $f(x,y,t)$ such that (wave:app:exer:standing:waves:damped) is a solution to the general 2D wave equation problem with damping and variable wave velocity. Find corresponding I and V, and make a convergence test that recovers the expected convergence rate. Make a corresponding nose test.

Hint. You may explore `sympy` for automating the analytical work:

```
>>> from sympy import *
>>> x = Symbol('x')
```

```
>>> q=x**2
>>> u=sin(x)
>>> r = diff(q*diff(u, x), x)   # Derivative: (q*u_x)_x
>>> simplify(r)
x*(-x*sin(x) + 2*cos(x))
```

11 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. A particular physical interpretation can be wave propagation of a tsunami over a subsea hill. The unknown $u(x, y, t)$ is then the elevation of the ocean surface, and the boundary condition $\partial u/\partial n = 0$ means that the waves are perfectly reflected, because of a steep hill at the shore, or the condition expresses symmetry in the solution. The wave velocity is given by $q = gH(x, y)$, where g is the acceleration of gravity and $H(x, y)$ is the stillwater depth.

It can be wise to do Problem 21[4], because a 1D program corresponding to the present 2D problem allows much faster experimentation with parameters and effects.

The initial surface is taken as a smooth Gaussian function

$$I(x; I_0, I_a, I_m, I_s) = I_0 + I_a \exp\left(-\left(\frac{x - I_m}{I_s}\right)^2\right), \tag{7}$$

with $I_m = 0$ reflecting the location of the peak of $I(x)$ and I_s being a measure of the width of the function $I(x)$ (I_s is $\sqrt{2}$ times the standard deviation of the familiar normal distribution curve).

Three different bottom shapes can be investigated. A 2D Gaussian hill can be modeled by

$$B(x; B_0, B_a, B_{mx}, B_{my}, B_s, b) = B_0 + B_a \exp\left(-\left(\frac{x - B_{mx}}{B_s}\right)^2 - \left(\frac{y - B_{my}}{bB_s}\right)^2\right), \tag{8}$$

where b is a scaling parameter: $b = 1$ gives a circular Gaussian function with circular contour lines, while $b \neq 1$ gives an elliptic shape with elliptic contour lines.

A less smooth hill is modeled by the "cosine hat" function

$$B(x; B_0, B_a, B_{mx}, B_{my}, B_s) = B_0 + B_a \cos\left(\pi \frac{x - B_{mx}}{2B_s}\right) \cos\left(\pi \frac{y - B_{my}}{2B_s}\right), \tag{9}$$

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

A more dramatic hill shape is a box:

$$B(x; B_0, B_a, B_m, B_s, b) = B_0 + B_a \tag{10}$$

[4] ../pub/sphinx_wave/._part0010_main_wave.html#problem-21-earthquake-generated-tsunami-over-a-subsea-hill

for x and y inside a rectangle

$$B_{mx} - B_s \leq x \leq B_{mx} + B_s, \quad B_{my} - bB_s \leq y \leq B_{my} + bB_s,$$

and $B = B_0$ outside this rectangle. The b parameter controls the rectangular shape of the cross section of the box.

Note that the initial condition and the listed bottom shapes are symmetric around the line $y = B_{my}$. We therefore expect the surface elevation also to be symmetric with respect to this line. This means that we can halve the computational domain by working with $[0, L_x] \times [0, B_{my}]$. Along the upper boundary, $y = B_{my}$, we must impose the symmetry condition $\partial\eta/\partial n = 0$. Such a symmetry condition ($-\eta_x = 0$) is also needed at the $x = 0$ boundary because the initial condition has a symmetry here. At the lower boundary $y = 0$ we also set a Neumann condition (which becomes $-\eta_y = 0$). The wave motion is to be simulated until the wave hits the reflecting boundaries where $\partial\eta/\partial n = \eta_x = 0$.

Investigate how different hill shapes, different sizes of the water gap above the hill, and different resolutions $\Delta x = \Delta y = h$ and Δt influence the numerical quality of the solution. One anticipates that the less smooth hill shapes will introduce more numerical noise. Presenting the results as movies of the surface elevation is effective.

12 Additional tasks

The groups can select between one or more of the following tasks.

12.1 Harmonic averaging

Harmonic means are often used if the coefficient q is non-smooth or discontinuous. Investigate if harmonic averaging of q works better than the arithmetic averging for the box-shaped subsea hill. The effect might not be big unless the water gap at the top of the hill is small. It can be wise to test the effect of harmonic averaging in 1D first.

Remark. With a small gap between the obstruction and the free surface, and with abrupt changes in the bottom shape, the model PDE does not necessarily describe the wave motion in an accurate or qualitatively correct way.

12.2 Visualization

Create some fancy 3D visualization of the water waves *and* the subsea hill. Try to make the hill transparent. Suitable tools are

- Mayavi[5]
- Paraview[6]
- OpenDX[7]
- Matplotlib[8]

[5] `http://code.enthought.com/projects/mayavi/`
[6] `http://www.paraview.org/`
[7] `http://www.opendx.org/`
[8] `http://matplotlib.org/`

12.3 Open outlet boundary: 1D condition

Implement an open boundary condition at $x = L_x$, $u_t + \sqrt{q}u_x = 0$, as suggested in Problem 20[9]. This condition is only correct in 1D, but might work satisfactorily in 2D if the wave is approximately one-dimensional when it hits the boundary. See how well this condition works in letting the tsunami pass out of the domain. The distance from the subsea hill (which disturbes the wave) and the outlet boundary $x = L_x$ is an important parameter.

12.4 Open outlet boundary: absorbing layer

Instead of using a condition $u_t + \sqrt{q}u_x = 0$, which is exact only for plane waves propagating in x direction, one can add an artificial domain $[L_x, L_x + \delta] \times [0, L_y]$ where waves are sufficiently damped and absorbed. The goal of an open boundary condition is to avoid waves being reflected back into the domain. Turn on the damping parameter b in $[L_x, L_x + \delta] \times [0, L_y]$, and test if it is wise to vary b, say in a linear or exponential fashion to have a smooth transition from $b = 0$ in the physical domain and to some significant (efficient) b value towards the artificial boundary $x = L_x + \delta$.

12.5 Compiled loops

Extend the program with compiled loops using one or more of the following techniques:

- Cython code
- Fortran code interfaced via `f2py`
- C code interfaced via Cython or `f2py`
- C/C++ code interfaced via `scipy.weave`
- C/C++ code interfaced via Instant[10]

Note that Instant comes with FEniCS (`sudo apt-get install fenics` on Ubuntu will install Instant) and it is described in the FEniCS book[11].

12.6 Parallel computing

Make a parallel version of the program using different approaches:

- Automatic OpenMP code in migrated Cython loops using `cython.parallel`[12]
- OpenMP in migrated C or Fortran loops
- MPI in migrated C or Fortran loops
- mpi4py MPI programming from Python (distribute vectorized code)

[9] `../pub/sphinx_wave/._part0010_main_wave.html#problem-20-implement-open-boundary-conditions`
[10] `https://launchpad.net/instant`
[11] `https://launchpad.net/fenics-book`
[12] `http://docs.cython.org/src/userguide/parallelism.html`

- Automatic parallelization of vectorized NumPy code via NumbaPro[13] (a license can be made available)

[13] http://docs.continuum.io/numbapro/