INF5620: Project 2

Christopher Hawkins

Problem

We wish to solve the two-dimensional, standard, linear wave equation, with damping:

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

where $u(x,y)$ is the velocity $q(x,y)$ is the , b is the damping factor and $f(x,y,t)$ the external forces.

The boundary conditions are:

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

where n is the normal vector at the boundary.

Solver

In this case we implement a finite difference scheme on a regular grid. Solving the equation is divided into two parts. First the spacial derivatives are calculated, second the equation is integrated in time.

The basic finite difference equations are: For a single derivative

$$\frac{\partial u(i,j)}{\partial x} = \frac{u(i+1,j) - u(i,j)}{\Delta x}$$

Where i, j are the grid indicies and Δx the grid spacing

For second derivative

$$\frac{\partial^2 u(i,j)}{\partial x^2} = \frac{u(i+1,j) - 2u(i,j) + u(i-1,j)}{\Delta x^2}$$

In addition we may also chose to use an isotropic finite difference derivative for greater accuracy

$$\begin{aligned}\frac{\partial u(i,j)}{\partial x} = \frac{1}{8\Delta x} & u(i+1,j+1) + 2u(i+1,j) - u(i-1,j-1) \\ & - u(i-1,j+1) - 2u(i-1,j) + u(i+1,j-1)\end{aligned}$$

There are two methods we can employ to solve the spacial derivative in the wave equation. The first method is to simply use the chain rule e.g.

$$\frac{\partial}{\partial x}(q(x,y)\frac{\partial}{\partial x}u) = q(x,y)\frac{\partial^2}{\partial x^2}u + \frac{\partial}{\partial x}q\frac{\partial}{\partial x}u$$

A simple substitution of the finite difference equivalent of $\frac{\partial}{\partial x}$ provides the simulated version. Alternatively we can calculate q at half grid points giving us the following:

$$\begin{aligned} u_d(i,j) =& q(i+1/2,j)\frac{u(i+1,j)-u(i,j)}{dx^2} - q(i-1/2,j)\frac{u(i,j)-u(i-1,j)}{\Delta x^2} \\ &+ q(i,j+1/2)\frac{u(i,j+1)-u(i,j)}{\Delta\Delta y^2} - q(i,j-1/2)\frac{u(i,j)-u(i,j-1)}{\Delta y^2} \end{aligned}$$

Where u_d is the total spacial derivative at grid point i,j. Here $q(i\pm 1/2,j) = (q(i\pm 1,j)+q(i,j))/2$ and a simular equivilent in the j direction.

In this program both methods are implemented along with a choice of either standard or isotropic derivatives to be chosen at the users leisure. The code for the program is called 'main.cpp'

Truncation error

The spacial traction error of $\frac{\partial}{\partial x}(q(x,y)\frac{\partial}{\partial x}u)$ can be written as follows

$$R([D_x q D_x u]_i) = [D_x G]_i$$

Where D_x represents the finite difference derivative, and:

$$G_i = (\frac{1}{24}u_{xxx}(i,j)q(i,j) + \frac{1}{24}u_x(i,j)q_{xx}(i,j))$$

In the temporal direction the truncation error reads as follows

$$R([D_{tt}u]_i) = \frac{1}{12}u_{tttt}(i,j)\Delta t^2$$

In the above equations R implies the truncation error. The truncation error is calculated within the main program and outputted in the data file 'truncation_error.txt' which can be viewed for the entire grid at every time step.

Test cases

Constant solution

Running the constant solution is a simple case of opening the file 'wave_equation_2d.m' this is the main file with which to run simulations with this program. This

file should be self explanatory. To run both constant $q(x, y)$ and $u(x, y)$ simply set the input files 'geometry.txt' and 'u_initial.txt' to an array of constant values by inputting the command:

make_array('geometry.txt', nx, ny, 'const 2d', round(nx/2), round(ny/2), dx, dy);

where 'const 2d' is the type of array you wish to use. The result should be a solution that is constant in time and space.

Manufactured solution

We must impliment a known solution that fulfills the boundary conditions $u(0, t) = u(L, t) = 0$. We choose the solution

$$u(x, y, t) = -x(L - x)sin(t)$$

Inserting into the wave equation and solving we gain for f

$$f = (2 - x(L - x))sin(t)$$

Here we impliment the system with initial conditions:

$$u(x, y, 0) = -x(L - x)$$

To run this test simply run the file 'manufactured_solution.m'

Cubic solution

In this test we choose to implement a wave in the form

$$u(x, y, 0) = X(x)Y(y)T(t) = (a_x x^3 + b_x x^2)(a_y y^3 + b_y y^2)T(t)$$

we choose $q(x, y)$ to be constant in which case we know the spacial derivatives will be of the form

$$\nabla^2 X(x)Y(y)T(t) = [(6a_x x+2b_x)(a_y y^3+b_y y^2)+(a_x x^3+b_x x^2)(6a_y y+2b_y)]T(t)$$

If we set

$$f(x, y, t)) = -q(x, y)[(6a_x x+2b_x)(a_y y^3+b_y y^2)+(a_x x^3+b_x x^2)(6a_y y+2b_y)]T(t)$$

then $\frac{\partial^2 u}{\partial t^2} = 0$ and hence $T(t) = ct + I$ where c is an arbitrary constant.

To impliment the boundary conditions we can choose the constants $a_x, b_x, a_y$and b_y to be set to:

$$b_x = 1/100$$
$$a_x = (-2b_x)/(3L_x)$$

And the same for a_y, b_y. Here L is the size of the simulated area

To run this in the program simply set '-test 1' and use the 2D array type as 'cubic_test'. Alternatively run the script 'cubic_test.m' where everything is preset to run

Standing undamped solution

Here we choose to implement the standing wave to which we have an exact solution, this can therefore be checked with the simulated result in order to determine the error. The standing wave takes the following form

$$u_e(x, y, t) = Acos(k_x x)cos(k_y y)cos(\omega t)$$

with $k_x = n_x\pi/L_x$ and $k_y = n_y\pi/L_y$, n_x and n_y are integers. We shall define the error as

$$E = max(u_e - u^{\Delta t}_{\Delta x, \Delta y})$$

We expect the error to scale as

$$R = \frac{1}{12}(\Delta t^2 u_{tttt} + \Delta x^2 u_{xxxx} + \Delta y^2 u_{yyyy})$$

We can rewrite this as

$$R = \frac{1}{12}(h^2 \cdot (\nabla^4 \cdot u))$$

where $h^2 = \Delta t^2 + \Delta x^2 + \Delta y^2$. If we now plot E/h^2 we could therefore expect a constant solution. The script 'standing.m' plots this relationship. It should be noted that the solution remains constant up to the point where the stability criterion is reached at which point the error increases dramatically.

Standing damped solution

The solution the the standing damped wave is

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

Inserting this into the derivative we gain

$$\begin{aligned}
&(c^2 - \omega^2)(Acos(\omega t) + Bsin(\omega t))e^{-ct}cos(k_x x)cos(k_y y) \\
&- cb(Acos(\omega t) + Bsin(\omega t))e^{-ct}cos(k_x x)cos(k_y y) \\
&+ b\omega(-Asin(\omega t) + Bcos(\omega t))e^{-ct}cos(k_x x)cos(k_y y) \\
&= -(k_x^2 + k_y^2)(Acos(\omega t) + Bsin(\omega t))e^{-ct}cos(k_x x)cos(k_y y)
\end{aligned}$$

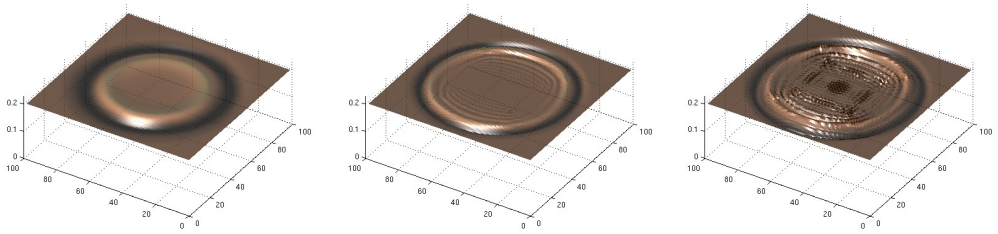

Figure 1: Velocity plot of several types of initial wave function. From left to right, Gaussian, Cosine, Sharp

This gives us the relations

$$(c^2 - \omega^2) = -(k_x^2 + k_y^2)$$
$$c = b/2$$

We can use this exact solution in the same way as for the undamped wave. The damped wave is also included in the script 'standing.m' and is called by simply using a damping coefficient $b > 0$.

Investigating a physical problem

We now choose to use the program to investigate a physical problem. Several predefined geometries are available from the 'make_array.m' function these are listed below.

const 2d	constant values
gaussian 1d	1D gaussian peak
gaussian 2d	2D gaussian peak
cos 2d	2D cosine peak
sharp 2d	sharp/square peak

these peaks can be outputted to either the geometry or initial velocity files for import. Alternatively any custom 2D array can be created at the users leisure. Below are a few of the results from the predefined functions. A series of 2D peaks of various types with constant q, notice the increase in instability of the simulation. In the following cases a 1D Gaussian passes over a specific type of predefined geometry.

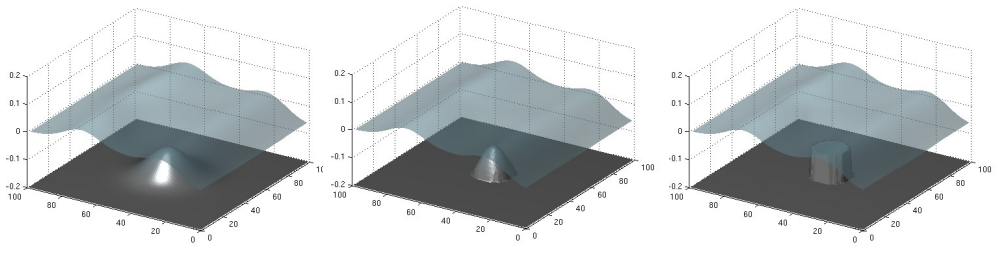

Figure 2: Velocity plot of a gaussian wave as it passes over various types of geometry. From left to right, Gaussian, Cosine, Sharp