The Wave Equation in 1D and 2D

Knut–Andreas Lie

Dept. of Informatics, University of Oslo

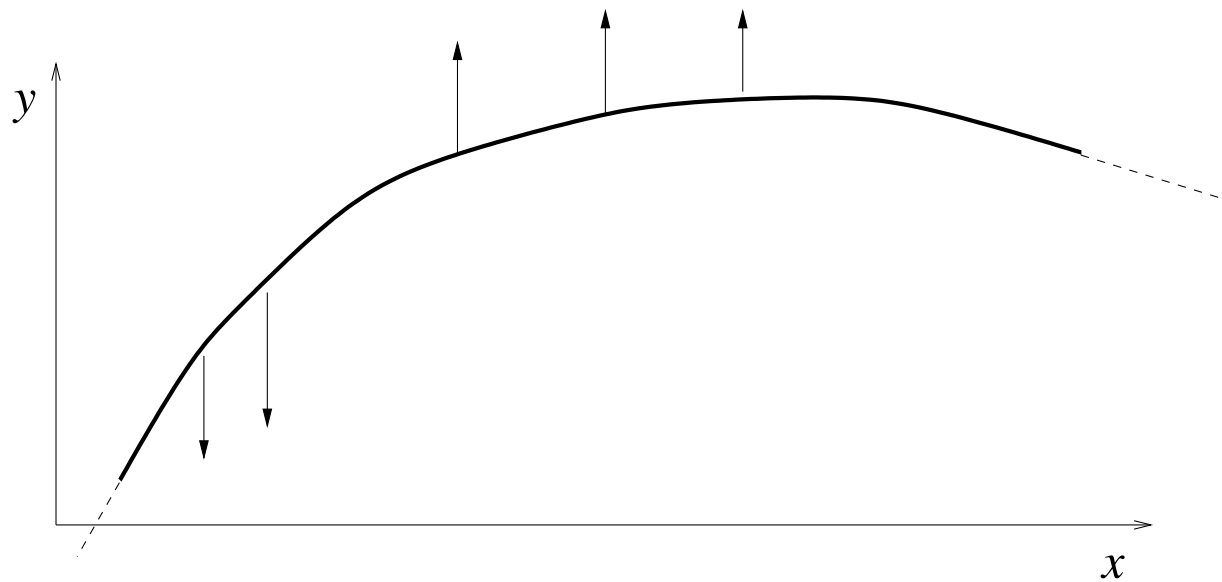
Wave Equation in 1D

- Physical phenomenon: small vibrations on a string
- Mathematical model: the *wave equation*

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \frac{\partial^2 u}{\partial x^2}, \qquad x \in (a, b)$$

- This is a time- and space-dependent problem
- We call the equation a *partial differential equation* (PDE)
- We must specify boundary conditions on u or u_x at $x = a, b$ and initial conditions on $u(x, 0)$ and $u_t(x, 0)$

Derivation of the Model



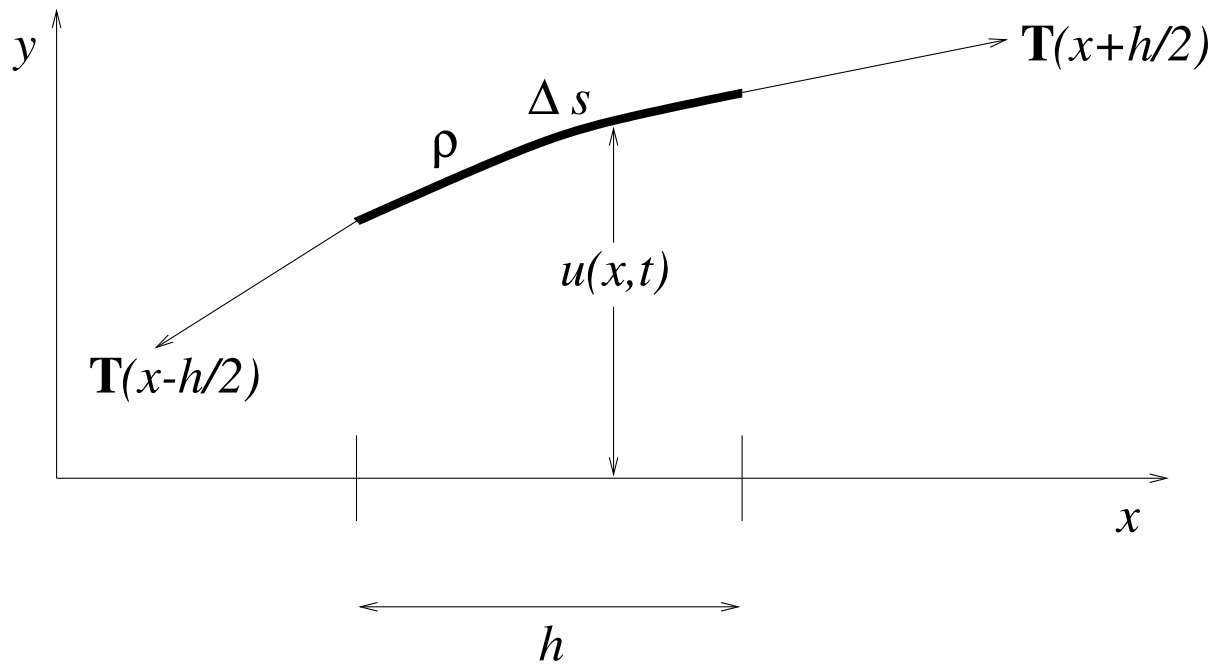
Physical assumptions:

- the string = a line in 2D space
- no gravity forces
- up-down movement (i.e., only in y-direction)

Physical quantities:

- $\mathbf{r} = x\mathbf{i} + u(x,t)\mathbf{j}$: position
- $\mathbf{T}(x)$: tension force (along the string)
- $\theta(x)$: angle with horizontal direction
- $\varrho(x)$: density

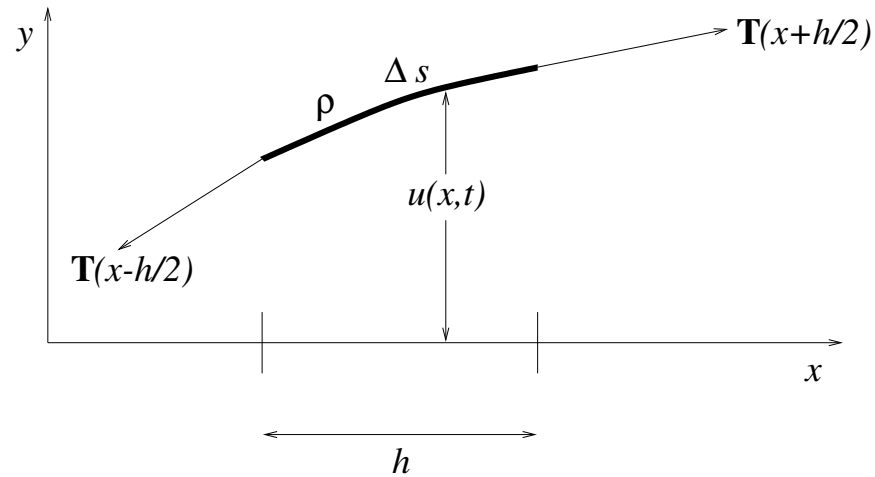
Derivation of the Model, cont'd



Physical principle, Newton's second law:

$$\text{total mass} \cdot \text{acceleration} = \text{sum of forces}$$

Derivation of the Model, cont'd



Total mass of line segment: $\varrho(x)\Delta s$

Acceleration: $$\mathbf{a} = \frac{\partial^2 \mathbf{r}}{\partial t^2} = \frac{\partial^2 u}{\partial t^2}\mathbf{j}$$

The tension is a vector (with two components):

$$\mathbf{T}(x) = T(x)\cos\theta(x)\,\mathbf{i} + T(x)\sin\theta(x)\,\mathbf{j}$$

Derivation of the Model, cont'd

Newton's law on a string element:

$$\varrho(x)\Delta s\,\frac{\partial^2 u}{\partial t^2}(x,t)\,\mathbf{j} = \mathbf{T}\Big(x+\frac{h}{2}\Big) - \mathbf{T}\Big(x-\frac{h}{2}\Big)$$

$\longrightarrow$ A vector equation with two components

Now we do some mathematical manipulations

- eliminate x-component of equation
- use geometrical considerations

and in the limit $h \longrightarrow 0$ we get:

$$\varrho\left[1+\left(\frac{\partial u}{\partial x}\right)^2\right]^{\frac{1}{2}}\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\left(T\left[1+\left(\frac{\partial u}{\partial x}\right)^2\right]^{-\frac{1}{2}}\frac{\partial u}{\partial x}\right)$$

The Linearised Equation

For small vibrations ($\partial u/\partial x \approx 0$) this simplifies to:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \qquad c^2 = T/\varrho$$

Initial and boundary conditions:

- String fixed at the ends:

$$u(a,t) = u(b,t) = 0$$

- String initially at rest:

$$u(x,0) = I(x), \qquad u_t(x,0) = 0$$

The Complete Linear Model

After a scaling, the equation becomes

$$
\begin{aligned}
\frac{\partial^2 u}{\partial t^2} &= \gamma^2 \frac{\partial^2 u}{\partial x^2}, && x \in (0,1),\ t > 0 \\
u(x,0) &= I(x), && x \in (0,1) \\
u_t(x,0) &= 0, && x \in (0,1) \\
u(0,t) &= 0, && t > 0, \\
u(1,t) &= 0, && t > 0
\end{aligned}
$$

Exercise: try to go through the derivation yourself

Finite Difference Approximation

Introduce a grid in space-time

$$
\begin{aligned}
x_i &= (i-1)\Delta x, \quad & i = 1, \ldots, n \\
t_\ell &= \ell \Delta t, & \ell = 0, 1, \ldots
\end{aligned}
$$

Central difference approximations

$$
\begin{aligned}
\frac{\partial^2 u}{\partial x^2}(x_i, t_\ell) &\approx \frac{u_{i-1}^\ell - 2u_i^\ell + u_{i+1}^\ell}{\Delta x^2}, \\
\frac{\partial^2 u}{\partial t^2}(x_i, t_\ell) &\approx \frac{u_i^{\ell-1} - 2u_i^\ell + u_i^{\ell+1}}{\Delta t^2}
\end{aligned}
$$

Finite Difference Approximation, cont'd

Inserted into the equation:

$$\frac{u_i^{\ell-1} - 2u_i^{\ell} + u_i^{\ell+1}}{\Delta t^2} = \gamma^2 \frac{u_{i-1}^{\ell} - 2u_i^{\ell} + u_{i+1}^{\ell}}{\Delta x^2}$$

Solve for $u_i^{\ell+1}$. Then the difference equation reads

$$u_i^{\ell+1} = 2u_i^{\ell} - u_i^{\ell-1} + C^2 \left(u_{i-1}^{\ell} - 2u_i^{\ell} + u_{i+1}^{\ell} \right)$$

Here $C = \gamma \frac{\Delta t}{\Delta x}$ is the *CFL number*

Initial Conditions

Two conditions at $\ell = 0$ for all i:

- $u(x,0) = I(x) \quad \longrightarrow \quad u_i^0 = I(x_i)$
- $u_t(x,0) = 0 \quad \longrightarrow \quad \frac{u_i^1 - u_i^{-1}}{\Delta t} = 0, \quad \longrightarrow \quad u_i^1 = u_i^{-1}$

The second condition inserted into the equation for $\ell = 0$

$$u_i^1 = 2u_i^0 - u_i^1 + C^2\left(u_{i-1}^0 - 2u_i^0 + u_{i+1}^0\right)$$

$$\longrightarrow u_i^1 = u_i^0 + \frac{1}{2}C^2\left(u_{i-1}^0 - 2u_i^0 + u_{i+1}^0\right)$$

Two choices: either introduce a special stencil for $\ell = 0$, or a set of fictitious values

$$u_i^{-1} = u_i^0 + \frac{1}{2}C^2\left(u_{i-1}^0 - 2u_i^0 + u_{i+1}^0\right)$$

We use the second approach in the following.

Algorithm

- Define storage u_i^+ , u_i, u_i^- for $u_i^{\ell+1}$, u_i^{ℓ}, $u_i^{\ell-1}$
- Set $t = 0$ and $C = \gamma\Delta t/\Delta x$
- Set initial conditions $u_i = I(x_i)$, $i = 1, \ldots, n$
- Define u_i^- $(i = 2, \ldots, n-1)$

$$u_i^- = u_i + \frac{1}{2}C^2(u_{i+1} - 2u_i + u_{i-1}),$$

- While $t < t_{\text{stop}}$
  - $t = t + \Delta t$
  - Update all inner points $(i = 2, \ldots, n-1)$

$$u_i^+ = 2u_i - u_i^- + C^2(u_{i+1} - 2u_i + u_{i-1})$$

  - Set boundary conditions $\quad u_1^+ = 0, \quad u_n^+ = 0$
  - Initialize for next step $\quad u_i^- = u_i, \quad u_i = u_i^+, \quad i = 1, \ldots, n$

Straightforward F77/C Implementation

```
int main (int argc, const char* argv[])
{
  cout << "Give␣number␣of␣intervals␣in␣(0,1):␣";
  int i; cin >> i;  int n = i+1;

  MyArray<double> up (n); // u at time level l+1
  MyArray<double> u (n);  // u at time level l
  MyArray<double> um (n); // u at time level l-1

  cout << "Give␣Courant␣number:␣";
  double C; cin >> C;
  cout << "Compute␣u(x,t)␣for␣t␣<=␣tstop,␣where␣tstop␣=␣";
  double tstop; cin >> tstop;

  setIC(u, um, C);
  timeLoop (up, u, um, tstop, C);
  return 0;
}
```

The timeLoop Function

```
void timeLoop (MyArray<double>& up, MyArray<double>& u,
               MyArray<double>& um, double tstop, double C)
{
  int     i, step_no=0, n = u.size();
  double h = 1.0/(n-1), dt = C*h, t=0, Csq = C*C;

  plotSolution (u, t);          // initial displacement to file
  while (t <= tstop)     {
      t += dt; step_no++;

      for (i = 2; i <= n-1; i++)   // inner points
        up(i) = 2*u(i) - um(i) + Csq * (u(i+1) - 2*u(i) + u(i-1));

      up(1) = 0;  up(n) = 0;      // update boundary points:
      um = u;  u = up;            // update data struct. for next step

      plotSolution (up, t);   // plot displacement to file
    }
}
```

The setIC Function

```
void setIC (MyArray<double>& u0, MyArray<double>& um, double C)
{
  int     i , n = u0.size ();
  double x, h = 1.0/(n-1);          // length of grid intervals
  double umax = 0.05, Csq=C*C;

  // set the initial displacement u(x,0)
  for ( i = 1; i <= n; i++) {
    x = (i-1)*h;
    if (x < 0.7)   u0(i) = (umax/0.7) * x;
    else           u0(i) = (umax/0.3) * (1 - x);
  }

  // set the help variable um:
  for ( i = 2; i <= n-1; i++)
    um(i) = u0(i) + 0.5*Csq * (u0(i+1) - 2*u0(i) + u0(i-1));
  um(1) = 0; um(n) = 0;        // dummy values, not used in the scheme
}
```

The plotSolution Function

```
void plotSolution (MyArray<double>& u, double t)
{
  int    n = u.size ();          // the number of unknowns
  double h = 1.0/(n-1);          // length of grid intervals
  char   fn[30];
  static int i=-1;

  i++; sprintf (fn,".u.dat.%03d",i);
  ofstream outfile (fn);
  for (int i = 1; i <= n; i++)
    outfile << h*(i-1) << " " << u(i) << endl;
}
```

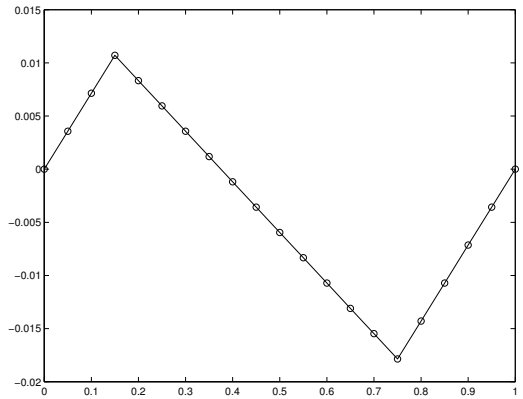
Here we have chosen to plot each time step in a separate (hidden) file with name `.u.dat.<step number>`

Animation in Matlab

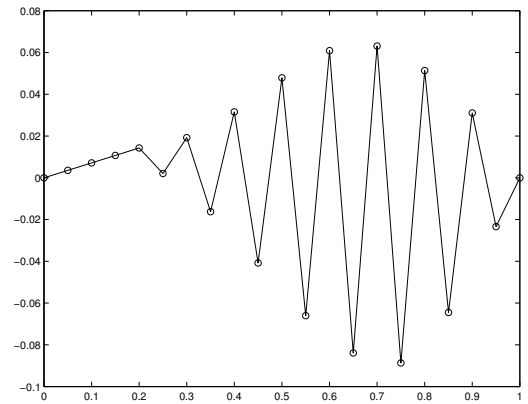
```
function myplot(nr,t,incr)
 if (nargin==2) incr=1; end;
j=0;
for i=0:incr:nr
      %
      % read simulation file number <i>
      fn=sprintf('.u.dat.%03d',i);
      fp = fopen(fn,'r'); [d,n]=fscanf(fp,'%f',[2, inf]);
      fclose(fp);
      %
      % plot the result
      plot(d(1,:), d(2,:), '-o'); axis([0 1 -0.1 0.1]);
       tittel = sprintf('Time␣t=%.3f', (t*i)/nr);
       title( tittel );
      %
      % force drawing explicitly and wait 0.2 seconds
      drawnow; pause(0.05);
 end
```

What about the Parameter C?

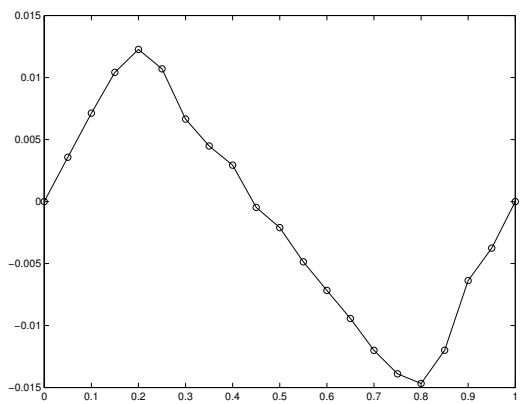
How do we choose the parameter $C = \Delta t / \Delta x$?



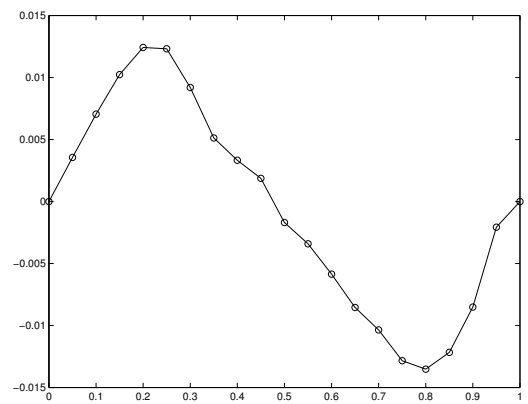
$C = 1.0$



$C = 1.05$



$C = 0.8$



$C = 0.3$

Solution at time $t = 0.5$ for $h = 1/20$

Numerical Stability and Accuracy

- We have two parameters, Δt and Δx, that are related through $C = \Delta t / \Delta x$
- How do we choose Δt and Δx?
- Too large values of Δt and Δx give
  - too large numerical errors
  - or in the worst case: unstable solutions
- Too small Δt and Δx means too much computing power
- Simplified problems can be analysed theoretically

$\Rightarrow$ Guide to choosing Δt and Δx

Large Destructive Water Waves

The wave equations may also be used to simulate large destructive waves

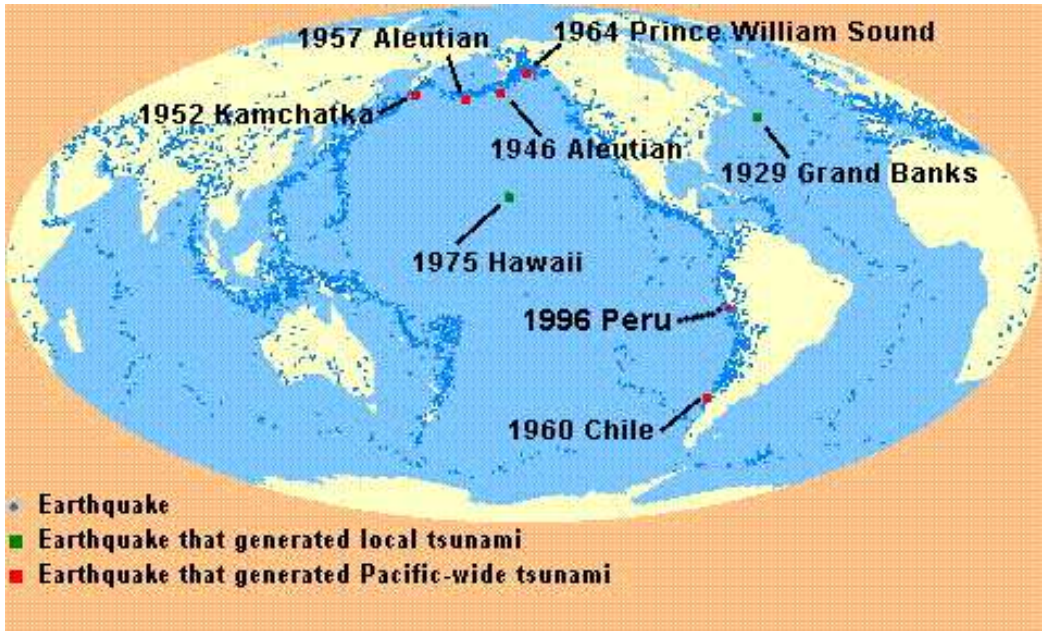
- Waves in fjords, lakes, or the ocean, generated by
  - slides
  - earthquakes
  - subsea volcanos
  - meteorittes

  Human activity, like nuclear detonations, or slides generated by oil drilling, may also generate tsunamis
- Propagation over large distances
- Wave amplitude increases near shore
- Run-up at the coasts may result in severe damage

Tsunamis (in the Pacific)

Japanese word for “large wave in harbor”. Often used as synonym for large destructive waves generated by slides, earthquakes, volcanos, etc.

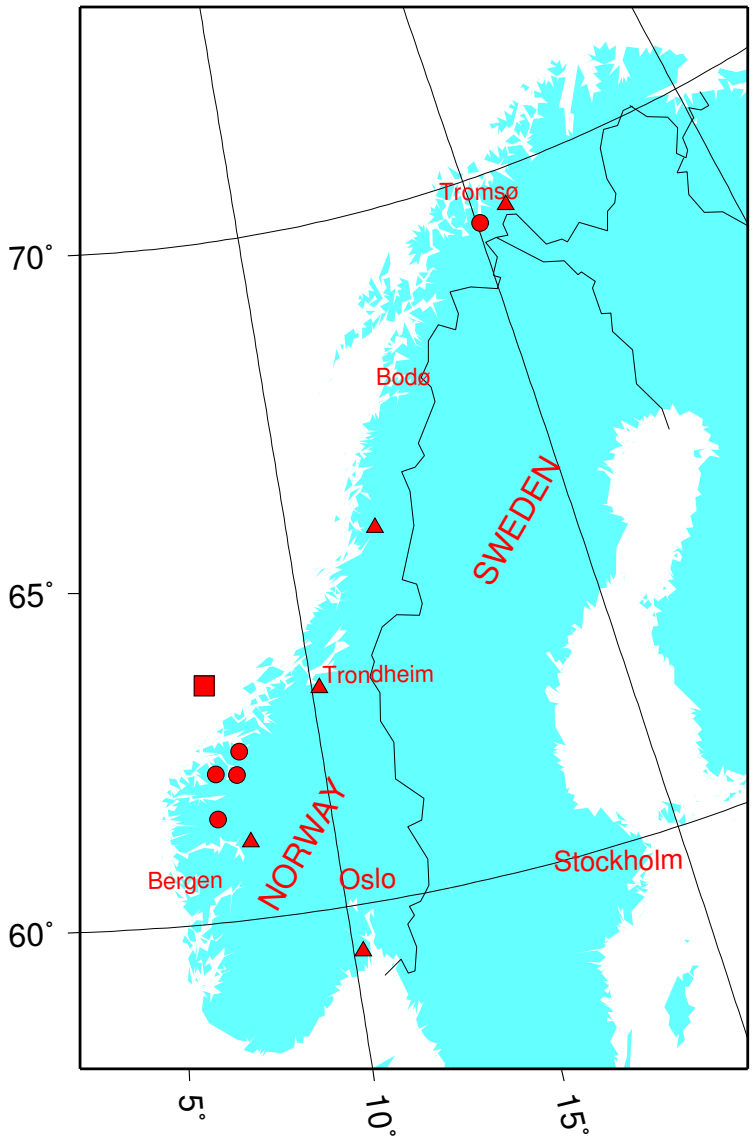
Map of older incidents:



Scenario:

Earthquake outside Chile, generates tsunami, propagating at 800 km/h accross the Pacific, run-up on densly populated coasts in Japan

Norwegian Tsunamis



Circles: Major incidents, > 10 killed

Triangles: Selected smaller incidents

Square: Storegga (5000 B.C.)

More information (e.g.,):

`math-www.uio.no/avdb/en/Research/geophys/`

`www.forskning.no/temaer/jordskjelv/`

`www.aftenposten.no/meninger/kronikker/article940524.ece`

Why Numerical Simulation?

- Increase the understanding of tsunamis
- Assist warning systems
- Assist building of harbor protection (break waters)
- Recognize critical coastal areas (e.g. move population)
- Hindcast historical tsunamis (assist geologists)

Simple Mathematical Model

The simplest model for tsunami propagation is the wave equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\left(H(x,y,t)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(H(x,y,t)\frac{\partial u}{\partial y}\right) - \frac{\partial^2 H}{\partial t^2}$$

Here $H(x,y,t)$ is the still-water depth (typically obtained from an electronic map). The t-dependence in H allows a moving bottom to model, e.g., an underwater slide or earthquake.

A common approximation of the effect of an earthquake (or volcano or faulting) is to set $H = H(x,y)$ and prescribe an initial disturbance of the sea surface.

First: the 1D Case

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\Big(H(x)\frac{\partial u}{\partial x}\Big)$$

The term $\frac{\partial}{\partial x}\big(H(x)\frac{\partial u}{\partial x}\big)$ is common for *many* models of physical phenomena

- Heat equation with spatially varying conductivity:
  $u_t = \frac{\partial}{\partial x}\Big(\lambda(x)\frac{\partial u}{\partial x}\Big)$
- Heat equation with temperature-dependent conductivity:
  $u_t = \frac{\partial}{\partial x}\Big(\lambda(u)\frac{\partial u}{\partial x}\Big)$
- Pressure distribution in a reservoir:
  $cp_t = \frac{\partial}{\partial x}\Big(K(x)\frac{\partial p}{\partial x}\Big)$
- ....

Discretisation

- Two-step discretization, first outer operator

$$\frac{\partial}{\partial x}\Big(H(x)\frac{\partial u}{\partial x}\Big)\Big|_{x=x_i} \approx \frac{1}{h}\Big(\big(H\frac{\partial u}{\partial x}\big)\Big|_{x=x_{i+1/2}} - \big(H\frac{\partial u}{\partial x}\big)\Big|_{x=x_{i-1/2}}\Big)$$

- Then inner operator

$$\big(H\frac{\partial u}{\partial x}\big)\Big|_{x=x_{i+1/2}} \approx H_{i+1/2}\frac{u_{i+1}-u_i}{h}$$

- And the overall discretization reads

$$\frac{\partial}{\partial x}\Big(H\frac{\partial u}{\partial x}\Big) \approx \frac{H_{i+1/2}(u_{i+1}-u_i) - H_{i-1/2}(u_i-u_{i-1})}{h^2}$$

Discretisation, cont'd

Often the function $H(x)$ is only given in the grid points, e.g., from measurements. Thus we need to define the value at the midpoint

- Arithmetic mean:

$$H_{i+\frac{1}{2}} = \frac{1}{2}\left(H_i + H_{i+1}\right)$$

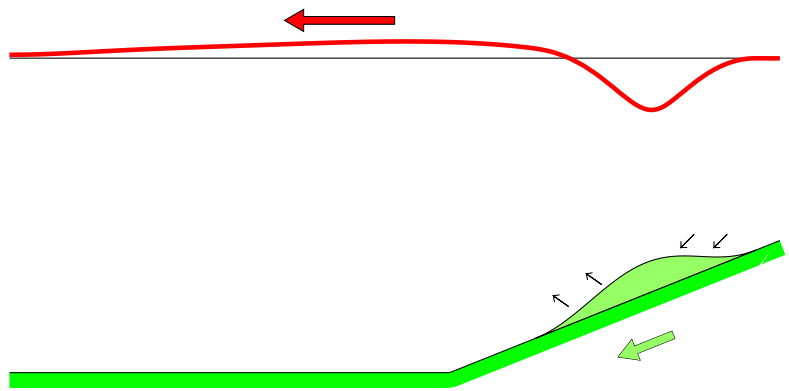
- Harmonic mean:

$$\frac{1}{H_{i+\frac{1}{2}}} = \frac{1}{2}\left(\frac{1}{H_i} + \frac{1}{H_{i+1}}\right)$$

- Geometric mean:

$$H_{i+\frac{1}{2}} = \left(H_i H_{i+1}\right)^{1/2}$$

Oblig: Tsunami Due to a Slide



We are going to study:

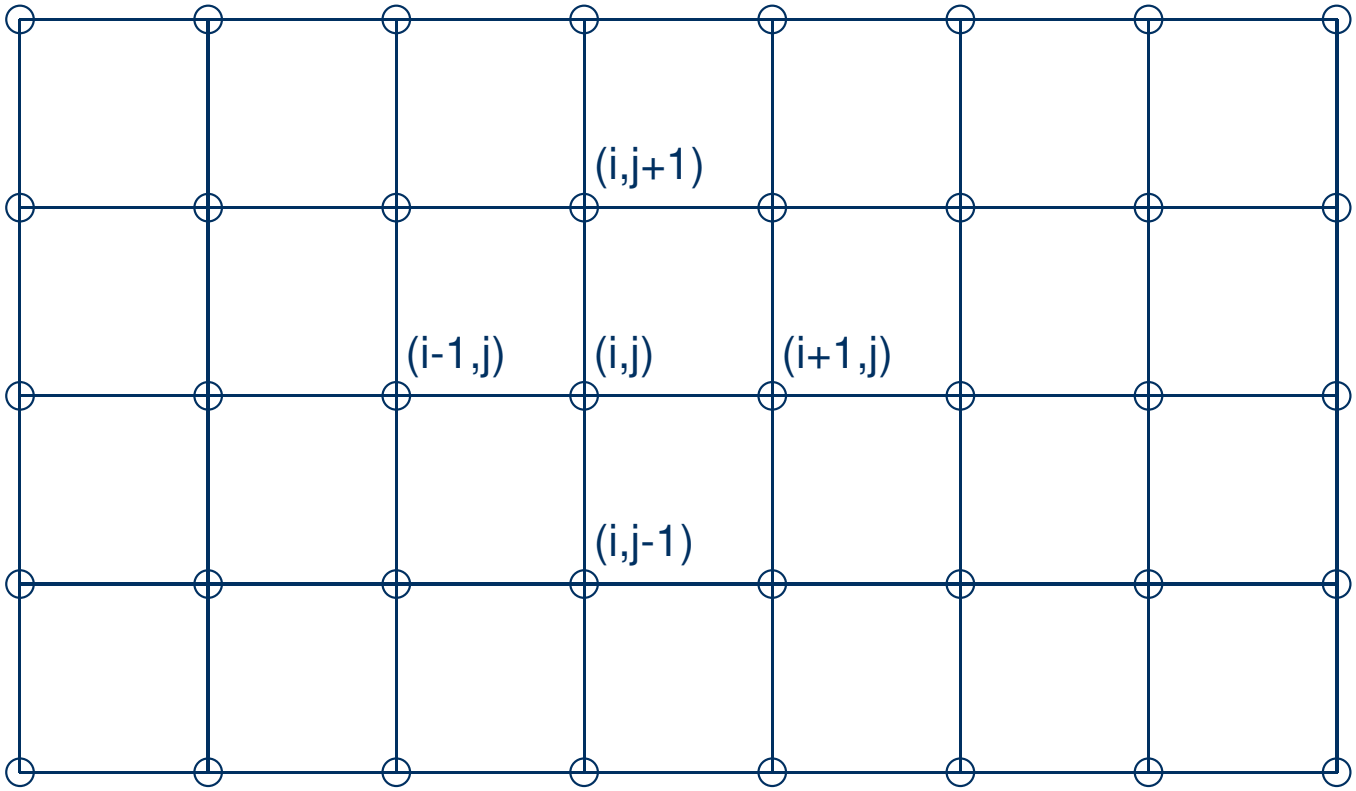
$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\left(H(x,t)\frac{\partial u}{\partial x}\right) + \frac{\partial^2 H}{\partial t^2}$$

Some physics for verification:

- Surface elevation ahead of the slide, dump behind
- Initially, negative dump propagates backwards
- The surface waves propagate faster than the slide moves

Discretisation of 2D Equation

Introduce a rectangular grid: $x_i = (i-1)\Delta x, \quad y_j = (j-1)\Delta y$



Seek approximation $u_{i,j}^{\ell}$ on the grid at discrete times $t_\ell = \ell\Delta t$

Discretisation, cont'd

Approximate derivatives by central differences

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

Similarly for the x and y derivatives.

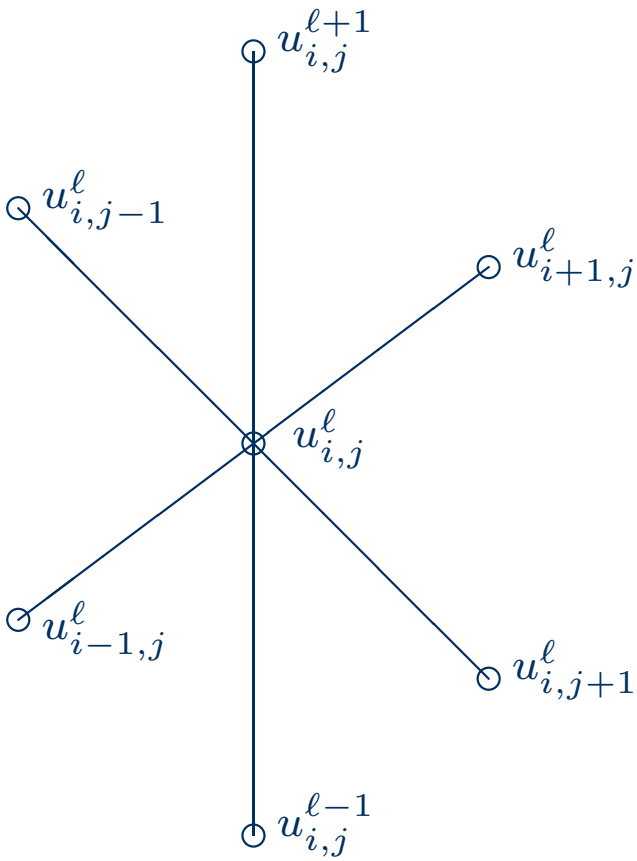
Assume for the moment that $\lambda \equiv 1$ and that $\Delta x = \Delta y$. Then

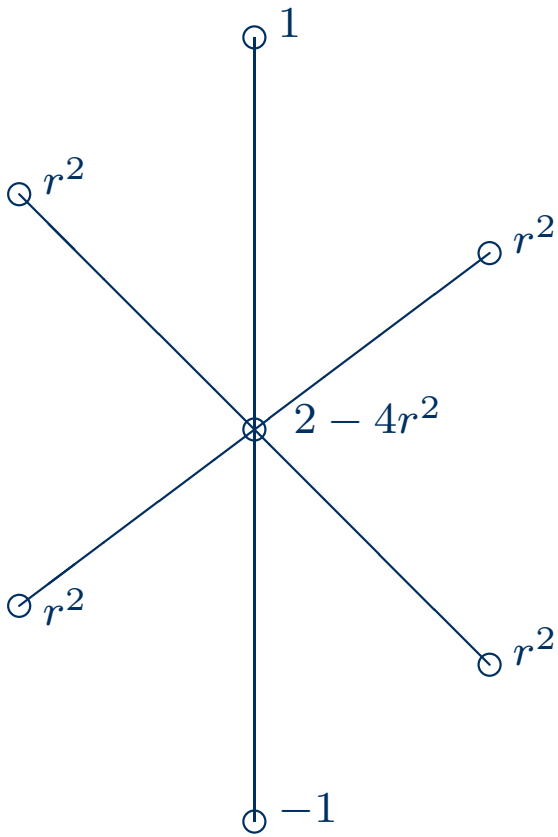
$$\frac{u_{i,j}^{\ell+1} - 2u_{i,j}^{\ell} + u_{i,j}^{\ell-1}}{\Delta t^2} = \frac{u_{i+1,j}^{\ell} - 2u_{i,j}^{\ell} + u_{i-1,j}^{\ell}}{\Delta x^2} + \frac{u_{i,j+1}^{\ell} - 2u_{i,j}^{\ell} + u_{i,j-1}^{\ell}}{\Delta y^2}$$

or (with $r = \Delta t/\Delta x$)

$$\begin{aligned} u_{i,j}^{\ell+1} &= 2u_{i,j}^{\ell} - u_{i,j}^{\ell-1} + r^2\big(u_{i+1,j}^{\ell} + u_{i-1,j}^{\ell} + u_{i,j+1}^{\ell} + u_{i,j-1}^{\ell} - 4u_{i,j}^{\ell}\big) \\ &= 2u_{i,j}^{\ell} - u_{i,j}^{\ell-1} + [\Delta u]_{i,j}^{\ell} \end{aligned}$$

Graphical Illustration





Computational molecule (stencil) in (x,y,t) space.

The Full Approximation

As we have seen earlier, a spatial term like $(\lambda u_y)_y$ takes the form

$$\frac{1}{\Delta y}\left(\lambda_{i,j+\frac{1}{2}}\left(\frac{u^\ell_{i,j+1}-u^\ell_{i,j}}{\Delta y}\right)-\lambda_{i,j-\frac{1}{2}}\left(\frac{u^\ell_{i,j}-u^\ell_{i,j-1}}{\Delta y}\right)\right)$$

Thus we derive

$$\begin{aligned}
u^{\ell+1}_{i,j} =& 2u^\ell_{i,j}-u^{\ell-1}_{i,j} \\
&+r_x^2\Big(\lambda_{i+\frac{1}{2},j}\big(u^\ell_{i+1,j}-u^\ell_{i,j}\big)-\lambda_{i-\frac{1}{2},j}\big(u^\ell_{i,j}-u^\ell_{i-1,j}\big)\Big) \\
&+r_y^2\Big(\lambda_{i,j+\frac{1}{2}}\big(u^\ell_{i,j+1}-u^\ell_{i,j}\big)-\lambda_{i,j-\frac{1}{2}}\big(u^\ell_{i,j}-u^\ell_{i,j-1}\big)\Big) \\
=& 2u^\ell_{i,j}-u^{\ell-1}_{i,j}+[\Delta u]^\ell_{i,j}
\end{aligned}$$

where $r_x=\Delta t/\Delta x$ and $r_y=\Delta t/\Delta y$.

Boundary Conditions

For the 1-D wave equation we imposed $u = 0$ at the boundary.

Now, we would like to impose full reflection of waves like in a swimming pool

$$\frac{\partial u}{\partial n} \equiv \nabla u \cdot \mathbf{n} = 0$$

Assume a rectangular domain. At the vertical ($x =$constant) boundaries the condition reads:

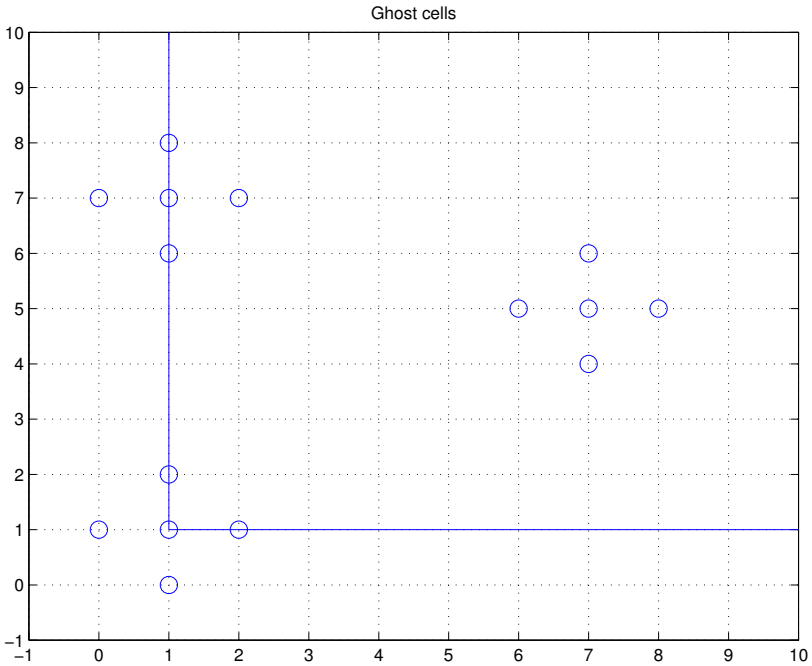
$$0 = \frac{\partial u}{\partial n} = \nabla u \cdot (\pm 1, 0) = \pm \frac{\partial u}{\partial x}$$

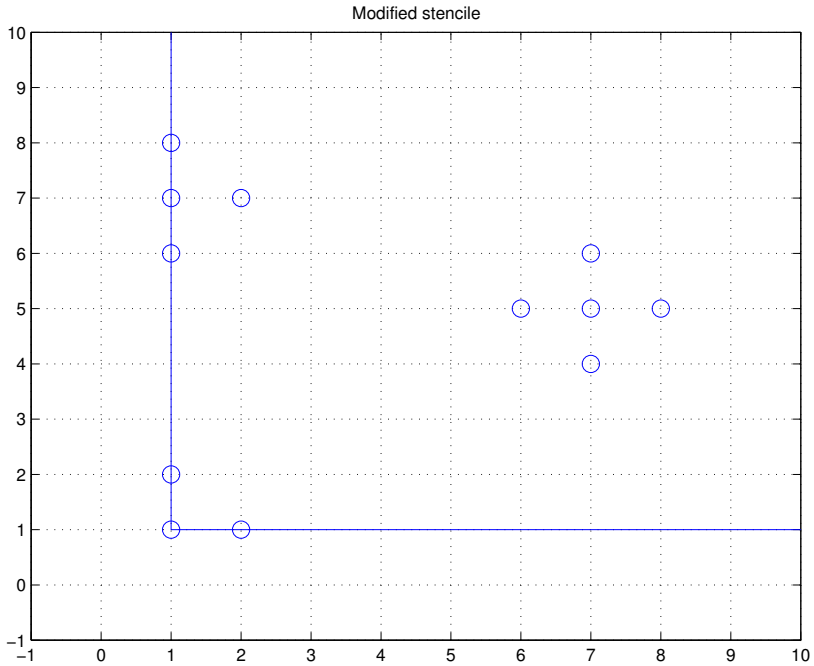
Similarly at the horizontal boundaries ($y =$constant)

$$0 = \frac{\partial u}{\partial n} = \nabla u \cdot (0, \pm 1) = \pm \frac{\partial u}{\partial y}$$

Boundary Conditions, cont'd

For the heat equation we saw that there are two ways of implementing the boundary conditions: ghost cells or modified stencils





Here we will use modified stencil to avoid the need to postprocess the data to remove ghost cells

Solution Algorithm

DEFINITIONS: storage, grid, internal points

INITIAL CONDITIONS: $u_{i,j} = I(x_i, y_j), \quad (i,j) \in \bar{\mathcal{I}}$

VARIABLE COEFFICIENT: set/get values for λ

SET ARTIFICIAL QUANTITY $u^-_{i,j}$: WAVE$(u^-, u, u^-, 0.5, 0, 0.5)$

Set $t = 0$

While $t \leq t_{\text{stop}}$

$t \leftarrow t + \Delta t$

(If λ depends on t: update λ)

update all points: WAVE$(u^+, u, u^-, 1, 1, 1)$

initialize for next step: $u^-_{i,j} = u_{i,j}, \quad u_{i,j} = u^+_{i,j}, \quad (i,j) \in \mathcal{I}$

Updating Internal and Boundary Points

WAVE(u^+, u, u^-, a, b, c)

UPDATE ALL INNER POINTS:

$$u^+_{i,j} = 2au_{i,j} - bu^-_{i,j} + c[\triangle u]_{i,j}, \quad (i,j) \in \mathcal{I}$$

UPDATE BOUNDARY POINTS:

$$\begin{aligned}
i = 1, \quad & j = 2, \ldots, n_y - 1; \\
& u^+_{i,j} = 2au_{i,j} - bu^-_{i,j} + c[\triangle u]_{i,j:i-1\rightarrow i+1}, \\
i = n_x, \quad & j = 2, \ldots, n_y - 1; \\
& u^+_{i,j} = 2au_{i,j} - bu^-_{i,j} + c[\triangle u]_{i,j:i+1\rightarrow i-1}, \\
j = 1, \quad & i = 2, \ldots, n_x - 1; \\
& u^+_{i,j} = 2au_{i,j} - bu^-_{i,j} + c[\triangle u]_{i,j:j-1\rightarrow j+1}, \\
j = n_y, \quad & i = 2, \ldots, n_x - 1; \\
& u^+_{i,j} = 2au_{i,j} - bu^-_{i,j} + c[\triangle u]_{i,j:j-1\rightarrow j+1},
\end{aligned}$$

Updating Internal and Boundary Points, cont'd

UPDATE CORNER POINTS ON THE BOUNDARY:

$$
\begin{aligned}
i = 1, \quad & j = 1; \\
& u^+_{i,j} = 2au_{i,j} - bu^-_{i,j} + c[\triangle u]_{i,j:i-1\rightarrow i+1,j-1\rightarrow j+1} \\
i = n_x, \quad & j = 1; \\
& u^+_{i,j} = 2au_{i,j} - bu^-_{i,j} + c[\triangle u]_{i,j:i+1\rightarrow i-1,j-1\rightarrow j+1} \\
i = 1, \quad & j = n_y; \\
& u^+_{i,j} = 2au_{i,j} - bu^-_{i,j} + c[\triangle u]_{i,j:i-1\rightarrow i+1,j+1\rightarrow j-1} \\
i = n_x, \quad & j = n_y; \\
& u^+_{i,j} = 2au_{i,j} - bu^-_{i,j} + c[\triangle u]_{i,j:i+1\rightarrow i-1,j+1\rightarrow j-1}
\end{aligned}
$$

Fragments of an Implementation

Suppose we have implemented `ArrayGen` for multidimensional arrays:

```
ArrayGen(real) up (nx,ny);   // u at time level l+1
ArrayGen(real) u  (nx,ny);   // u at time level l
ArrayGen(real) um (nx,ny);   // u at time level l-1
ArrayGen(real) lambda (nx,ny);  // variable coefficient
:
// Set initial data
:
// Set the artificial um
WAVE (um, u, um, 0.5, 0, 0.5, lambda, dt, dx, dy);

// Main loop
t=0; int step_no = 0;
while (t <= tstop ) {
  t += dt; step_no++;
  WAVE (up, u, um, 1, 1, 1,  lambda, dt, dx ,dy);
  um = u;  u = up;
}
```

Central Parts of WAVE

```
void WAVE(....)
{
  // update inner points according to finite difference scheme:
  for ( j=2; j<ny; j++)
    for ( i=2; i<nx; i++)
      up(i,j) = a*2*u(i,j) - b*um(i,j)
                + c*LaplaceU(i,j,i-1,i+1,j-1,j+1);

  // update boundary points (modified finite difference schemes):
  for ( i=1, j=2; j<ny; j++)
    up(i,j)=a*2*u(i,j)-b*um(i,j) +c*LaplaceU(i,j,i+1,i+1,j-1,j+1);
  for ( i=nx, j=2; j<ny; j++)
    up(i,j)=a*2*u(i,j)-b*um(i,j) + c*LaplaceU(i,j,i-1,i-1,j-1,j+1);
  :
  :
}
```

Trick: We Use Macros!

To avoid typos and increase readability we used a macro for the (long) finite difference formula corresponding to $[\Delta u]_{i,j}$:

```
#define LaplaceU(i,j,im1,ip1,jm1,jp1) \
  sqr(dt/dx)*\
  ( 0.5*(lambda(ip1,j  )+lambda(i  ,j  ))*(u(ip1,j  )-u(i  ,j  )) \
   -0.5*(lambda(i ,j  )+lambda(im1,j  ))*(u(i  ,j  )-u(im1,j  )))\
 +sqr(dt/dy)*\
  ( 0.5*(lambda(i  ,jp1)+lambda(i  ,j  ))*(u(i  ,jp1)-u(i  ,j  )) \
   -0.5*(lambda(i ,j  )+lambda(i  ,jm1))*(u(i  ,j  )-u(i  ,jm1)))
```

The macro is expanded by the C/C++ preprocessor (cpp).
Macros are handy to avoid typos and increase readability, but should be used with care...

What does the macro do? Consider the simple macro:

```
#define mac(X) q0(i,j-(X))
```

When called in the code with `mac(i+2)`, this expands to `q0(i,j-i+2)`

Efficiency Issues

Efficiency of plain loops is very important in numerics.

Two things should be considered in our case:

- Loops should be ordered such that $u(i,j)$ is traversed in the order it is stored. In our `ArrayGen` we assume that objects are stored columnwise. Therefore the loop should read:

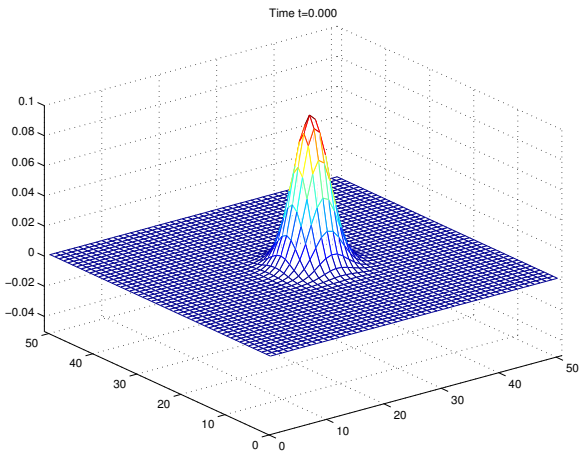
```
for ( j=1; j<ny+1; j++)
  for ( i=1; i<nx+1; i++)
    u(i,j ) = ...
```

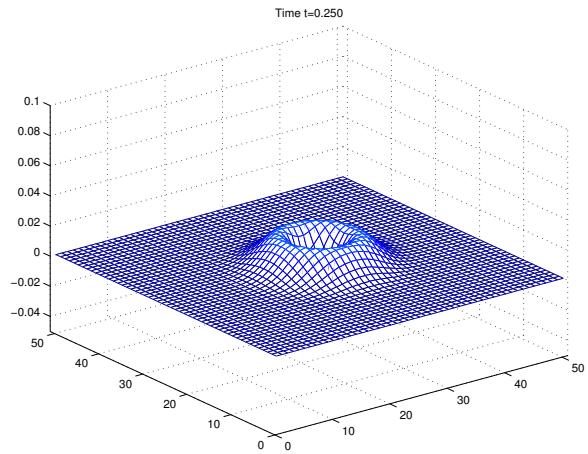
- One should avoid `if` statements in loops if possible; hence we will split the loop over all grid points separate loops over *inner* and *boundary points*.

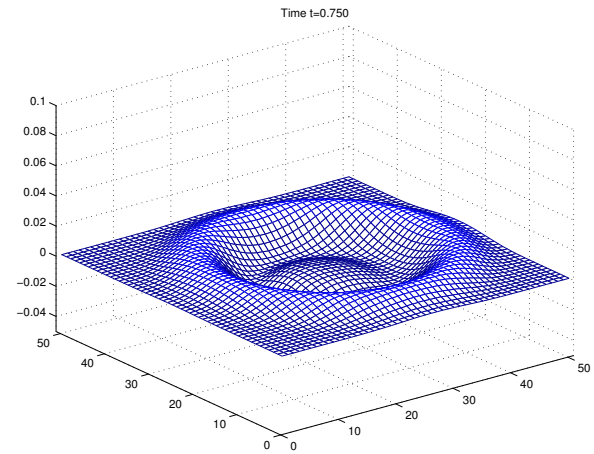
Remark I: Get the code to work before optimizing it

Remark II: Focus on a readable and maintainable code before thinking of efficiency

Visualising the Results



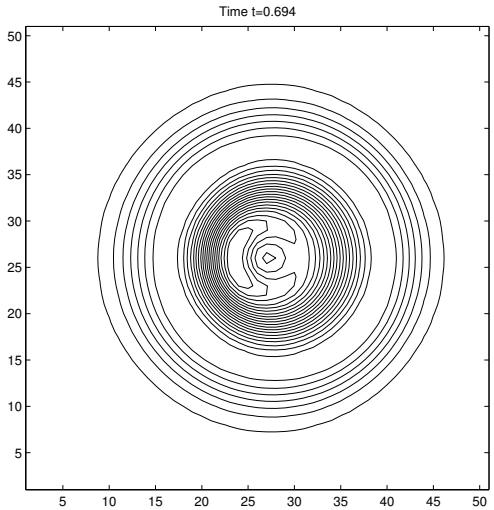




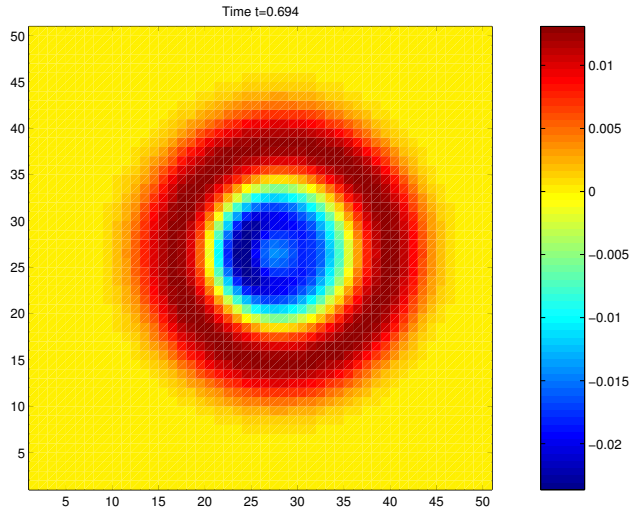
Plots generated in Matlab by the following sequence of commands:

```
>> load W.00.dat;
>> n=sqrt(length(W)); s=reshape(W,n,n);
>> mesh(s); caxis ([0 0.1]);
>> axis ([1 51 1 51 −0.05 0.1]);
>>  title ('Time␣t=0.000');
```

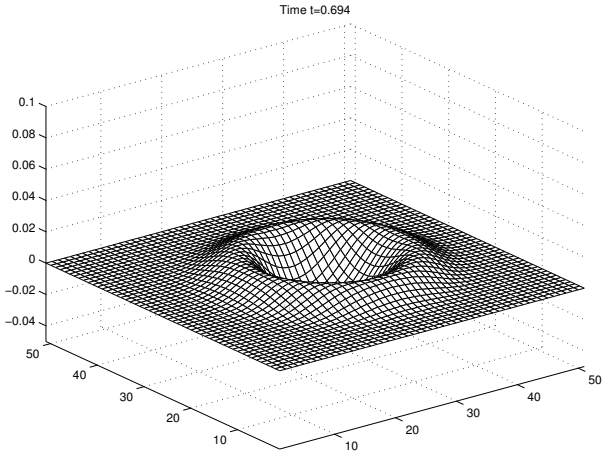
Various Ways of Visualisation



contour plot



color plot



surface mesh

lighted surface

Example: Waves Caused by Earthquake

Physical assumption: long waves in shallow water. Mathematical model

$$\frac{\partial^2 u}{\partial t^2} = \nabla \cdot \left[H(\mathbf{x}) \nabla u \right]$$

Consider a rectangular domain

$$\Omega = (s_x, s_x + w_x) \times (s_y, s_y + w_y)$$

with initial (Gaussian bell) function

$$I(x, y) = A_u \exp\left(-\frac{1}{2} \left(\frac{x - x_u^c}{\sigma_{ux}} \right)^2 - \frac{1}{2} \left(\frac{y - y_u^c}{\sigma_{uy}} \right)^2 \right)$$

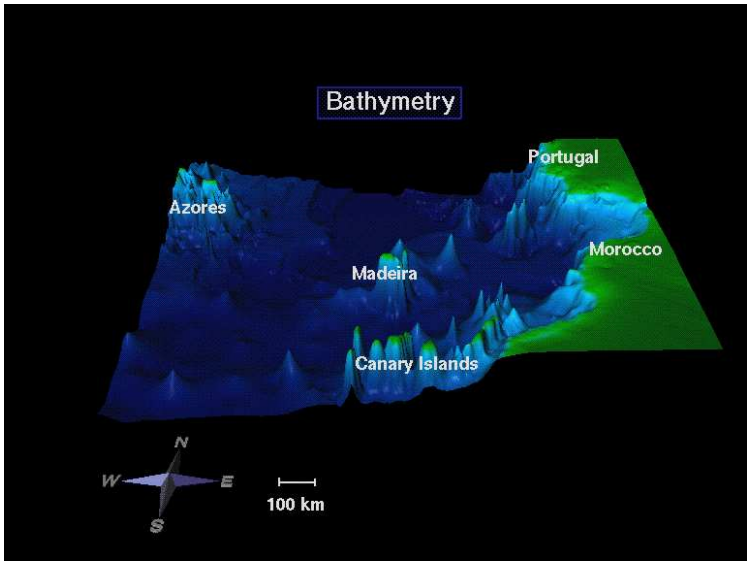
This models an initial elevation caused by an earthquake.

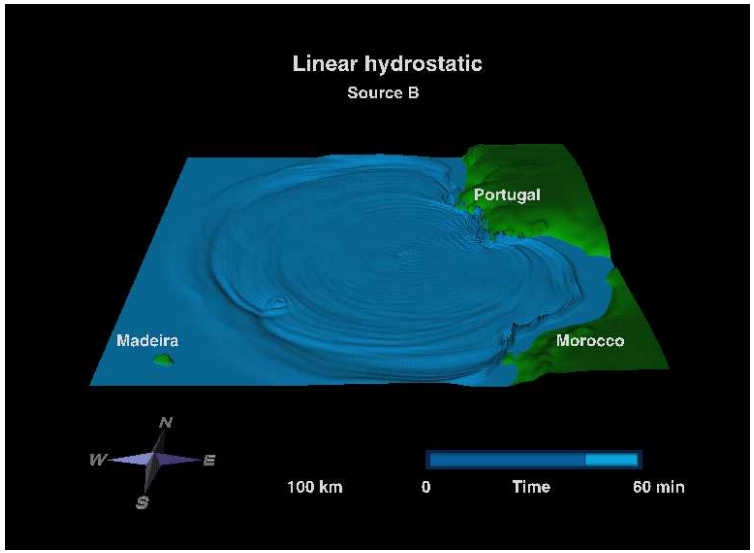
Example, cont'd

The earthquake takes place near an underwater seamount

$$H(x,y) = 1 - A_H \exp\left(-\frac{1}{2}\left(\frac{x - x_H^c}{\sigma_{Hx}}\right)^2 - \frac{1}{2}\left(\frac{y - y_H^c}{\sigma_{Hy}}\right)^2\right)$$

Simulation case inspired by the Gorringe Bank southwest of Portugal. Severe ocean waves have been generated due to earthquakes in this region.





http://www.math.uio.no/avdb/gitec/

Boundary Conditions

- waves should propagate out of the domain, without being reflected
- this is difficult to model numerically
- alternative:

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

  which gives full reflection from the boundary
- What? An unphysical boundary condition???
- This is in fact okay for a hyperbolic equation, like the wave equation; waves travel at a finite speed and the $\partial u/\partial n = 0$ condition is feasible up to the point in time where waves are reflected from the boundary

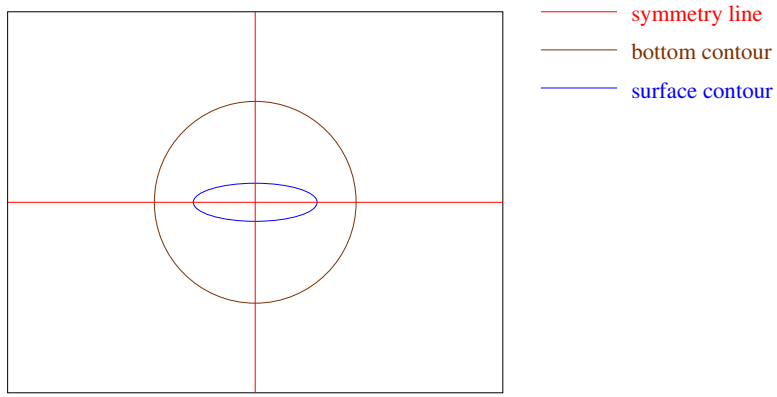
Boundary Conditions, cont'd

Use a circular bell for both I and H and set

$$y_H^c = y_u^c = x_H^c = x_u^c = 0$$

Thus we have symmetry about the lines

$$x = 0 \qquad y = 0$$



$\Rightarrow$ can reduce computational domain by a factor 4! Appropriate boundary condition at symmetry lines:

$$\frac{\partial u}{\partial n} = \nabla u \cdot \mathbf{n} = 0$$

If possible: one should always try to reduce computational domain by symmetry

Computational Results