Waves on a string can be modeled by the wave equation

\[ \frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \]

\( u(x, t) \) is the displacement of the string.

Demo of waves on a string.

The complete initial-boundary value problem

\[ \frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} , \quad x \in (0, L), \quad t \in (0, T] \]  (1)

\[ u(x, 0) = I(x) , \quad x \in [0, L] \]  (2)

\[ \frac{\partial u}{\partial t}(x, 0) = 0 , \quad x \in [0, L] \]  (3)

\[ u(0, t) = 0 , \quad t \in [0, T] \]  (4)

\[ u(L, t) = 0 , \quad t \in [0, T] \]  (5)

Input data in the problem

- Initial condition \( u(x, 0) = I(x) \): initial string shape
- Initial condition \( \partial u(x, 0)/\partial t = 0 \): string starts from rest

\( c = \sqrt{T/\rho} \): velocity of waves on the string

\( T \) is the tension in the string, \( \rho \) is density of the string.

Two boundary conditions on \( u \): \( u = 0 \) means fixed ends (no displacement).

Rule for number of initial and boundary conditions:

- \( u_{tt} \) in the PDE: two initial conditions, on \( u \) and \( u_t \)
- \( u_{xx} \) in the PDE: one boundary condition on \( u \) at each boundary point.

Demo of a vibrating string (\( C = 0.8 \))

• Our numerical method is sometimes exact (!)
• Our numerical method is sometimes subject to serious non-physical effects

Demo of a vibrating string (\( C = 1.0012 \))

Oops!
Step 1: Discretizing the domain

Mesh in time:
\[ 0 = t_0 < t_1 < t_2 < \cdots < t_{N_t} < t_{N_t} = T \]  
(6)

Mesh in space:
\[ 0 = x_0 < x_1 < x_2 < \cdots < x_{N_x} < x_{N_x} = L \]  
(7)

Uniform mesh with constant mesh spacings \( \Delta t \) and \( \Delta x \):
\[ x_i = i \Delta x, \quad i = 0, \ldots, N_x, \quad t_n = n \Delta t, \quad n = 0, \ldots, N_t \]  
(8)

Step 2: Fulfilling the equation at the mesh points

Let the PDE be satisfied at all interior mesh points:
\[ \frac{\partial^2}{\partial t^2} u(x_i, t_n) = \frac{\partial^2}{\partial x^2} u(x_i, t_n) \]  
(9)

for \( i = 1, \ldots, N_x - 1 \) and \( n = 1, \ldots, N_t - 1 \).

For \( n = 0 \) we have the initial conditions \( u = f(x) \) and \( u_t = 0 \), and at the boundaries \( i = 0, N_x \) we have the boundary condition \( u = 0 \).

Step 3: Algebraic version of the PDE

Replace derivatives by differences:
\[ \frac{u^{n+1}_i - 2u^n_i + u^{n-1}_i}{\Delta t^2} = \frac{u^{n+1}_{i+1} - 2u^n_{i+1} + u^{n-1}_{i+1}}{\Delta x^2} \]  
(10)

In operator notation:
\[ [D_t D_t u]_{n,i} = c^2 [D_x D_x u]_{n,i} \]  
(11)

Step 3: Algebraic version of the initial conditions

Need to replace the derivative in the initial condition
\[ u_t(x,0) = 0 \]  
by a finite difference approximation

The differences for \( u_0 \) and \( u_{N_x} \) have second-order accuracy

Use a centered difference for \( u_t(x,0) \):
\[ [D_t u]_0^n = 0, \quad n = 0 \quad \Rightarrow \quad u^{n+1}_0 - u^{n-1}_0 = \frac{u^{n+1}_1 - u^{n+1}_0}{\Delta t}, \quad i = 0, \ldots, N_x \]

The other initial condition \( u(x,0) = f(x) \) can be computed by
\[ u^n_i = f(x_i), \quad i = 0, \ldots, N_x \]
Step 4: Formulating a recursive algorithm

Nature of the algorithm: compute \( u \) in space at \( t = \Delta t, 2\Delta t, 3\Delta t, \ldots \)

Three time levels are involved in the general discrete equation:

\[
\begin{align*}
&n + 1, n, n - 1
&\text{are then already computed for } i = 0, \ldots, N_x, \text{ and }
\end{align*}
\]

\( u_{n+1}^i \) is the unknown quantity.

Write out \([D_t D_t u = c^2 D_x D_x]\) and solve for \( u_{n+1}^i \),

\[
\begin{align*}
&u_{n+1}^i = -u_{n-1}^i + 2u_n^i + C^2 (u_{n+1}^i - 2u_n^i + u_{n-1}^i)
\end{align*}
\]

(12)

The Courant number

\[
C = \frac{c \Delta t}{\Delta x}
\]

is known as the (dimensionless) Courant number.

Observe

There is only one parameter, \( C \), in the discrete model: \( C \) lumps mesh parameters \( \Delta t \) and \( \Delta x \) with the only physical parameter, the wave velocity \( c \). The value \( C \) and the smoothness of \( I(x) \) govern the quality of the numerical solution.

The finite difference stencil

The stencil for the first time level

Problem: the stencil for \( n = 1 \) involves \( u_{-1}^i \), but time \( t = -\Delta t \) is outside the mesh.

Remedy: use the initial condition \( u_t = 0 \) together with the stencil to eliminate \( u_{-1}^i \).

Initial condition:

\[
[2D_t u = 0] \Rightarrow \quad u_{-1}^i = u_0^i
\]

Insert it stencil \([D_t D_t u = c^2 D_x D_x]\) to get

\[
\begin{align*}
&u_1^i = u_0^i + \frac{C^2}{2} (u_{n+1}^i - 2u_n^i + u_{n-1}^i)
\end{align*}
\]

(14)

The algorithm

1. Compute \( u_0^i = I(x_i) \) for \( i = 0, \ldots, N_x \)
2. Compute \( u_1^i \) by (14) and set \( u_1^i = 0 \) for the boundary points \( i = 0 \) and \( i = N_x \) for \( n = 1, \ldots, N_t - 1 \),
3. For each time level \( n = 1, 2, \ldots, N_t - 1 \)
   a. Apply (2) to all \( u_n^i \) for \( i = 1, \ldots, N_x - 1 \)
   b. Set \( u_{n+1}^i = 0 \) for the boundary points \( i = 0, i = N_x \).

Moving finite difference stencil

web page or a movie file.
A slightly generalized model problem

Add source term and nonzero initial condition $u_0(x, 0)$:

$$
\begin{align*}
  u_1 &= c^2 u_{xx} + f(x, t), \\
  u(x, 0) &= I(x), \\
  u_t(x, 0) &= N(x), \\
  w(0, t) &= 0, \\
  w(L, t) &= 0
\end{align*}
$$

Discrete model for the generalized model problem

$$
[D, D] u = c^2 D_x^2 u + f^T
$$

Writing out and solving for the solution $u^{n+1}$:

$$
\begin{align*}
  u^{n+1} &= -u^n + 2u_t^n + C^2(u_{xx}^{n+1} - 2u^n + u_{xx}^n) + \Delta^2 f^T
\end{align*}
$$
Mo died equation for the first time level

**General difference for \( u_t(x,0) = V(x) \):**

\[
\left[ D_{tt}u - V_{tt} \right]_n = u_{n+1}^t - u_{n+2}^t - 2\Delta t V_{tt}
\]

Inserting this into the stencil (21) for \( n = 0 \) leads to

\[
u_{1}^{t} = u_{0}^{t} - \Delta t V_{t} + \frac{1}{2}\Delta^{2} (u_{0}^{x} - 2u_{1}^{x} + u_{2}^{x}) + \frac{1}{2}\Delta^{2} f_{t}^{x}
\]

Using an analytical solution of physical significance

- Standing waves occur in real life on a string
- Can be analyzed mathematically (known exact solution)

\[
u_{m}(x,t) = \tilde{A} \sin \left( \frac{m}{L} \pi x \right) \cos \left( \frac{m}{C} \pi t \right)
\]

Testing a manufactured solution

- Introduce common mesh parameter: \( h = \Delta t, \Delta x = ch/C \)
- This \( h \) keeps \( C \) and \( \Delta t/\Delta x \) constant
- Select coarse mesh: \( h/h_0 \)
- Run experiments with \( h_0 = 2^{-i} h_0 \) (halving the cell size), \( i = 0, \ldots, m \)
- Record the error \( E_i \) and \( h_i \) in each experiment
- Compute pairwise convergence rates \( r_i = \ln E_{i+1}/E_i \) \( \ln h_{i+1}/h_i \)
- Verification: \( r_i \rightarrow 2 \) as \( i \) increases

Constructing an exact solution of the discrete equations

- Manufactured solution with computation of convergence rates: much manual work
- (Simpler and more powerful: use an exact solution for \( u \), if possible so \( \tilde{u} \) and \( \tilde{f} \))
- A linear or quadratic \( u_n \) and \( \tilde{f} \) is often a good candidate
Analytical work with the PDE problem

Here, choose \( u_e \) such that \( u_e(0,t) = u_e(L,t) = 0 \):

\[ u_e(x,t) = x(L-x)(1 + \frac{1}{2} t) \]

Initial conditions:

\[ f(x) = x(L-x) \]

\[ V(x) = \frac{1}{2} x(L-x) \]

Analytical work with the discrete equations (1)

We want to show that \( u_e \) also solves the discrete equations!

Useful preliminary result:

\[ \frac{D_t^2 u_e}{\Delta t^2} = \frac{D_x^2 u_e}{\Delta x^2} \]

Initial conditions:

\[ f_i = x_i(L-x_i) \]

\[ V_i = \frac{1}{2} x_i(L-x_i) \]

Later we show that the exact solution of the discrete equations can be obtained by \( C = 1 \).

Analytical work with the discrete equations (1)

For each time level \( n \):

1. Compute \( u_0^i = I(x_i) \) for \( i = 0, \ldots, N_x \)
2. Compute \( u_1^i \) by (14) and set \( u_1^i = 0 \) for the boundary points \( i = 0, i = N_x \)
3. Set \( N_{t+1} = 2, \ldots, N_t - 1 \)
4. Apply (12) to find \( u_{t+1}^i \) for \( i = 1, \ldots, N_t - 1 \)
5. \( u_{t+1}^i = 0 \) for the boundary points \( i = 0, i = N_t \)

Testing with the exact discrete solution

\[ D_t^2 u_e = \frac{1}{\Delta t^2} \frac{D_x^2 u_e}{\Delta x^2} \]

Initial conditions:

\[ I(x) = x(L-x), V(x) = \frac{1}{2} x(L-x) \]

\[ u = x(L-x)(1 + \frac{1}{2} t) \]

\[ V = \frac{1}{2} x(L-x) \]

Let's simulate with one choice of \( c, \Delta x, \) and \( \Delta t \).

The algorithm

1. Compute \( u_i^0 = I(x_i) \) for \( i = 0, \ldots, N_x \)
2. Compute \( u_i^1 \) by (14) and set \( u_i^1 = 0 \) for the boundary points \( i = 0, i = N_x \)
3. For each time level \( t = 1, \ldots, N_t - 1 \):
   - Apply (12) to find \( u_{t+1}^i \) for \( i = 1, \ldots, N_t - 1 \)
   - Set \( u_{t+1}^i = 0 \) for the boundary points \( i = 0, i = N_t \)
Making a solver function (1)

We specify $\Delta t$ and $C$, and let the solver function compute $\Delta x = c \Delta t / C$.

```python
def solver(I, V, f, c, L, dt, C, T, user_action=None):
    ... # function definition ...
```
Making movie files

Storing spatial curve in a file, for each time level
Name files like 'something_%04d.png' % frame_counter
Combine files to a movie

Terminal> scitools movie encoder=html output_file=movie.html fps=4 frame_*.png # web page with a player
Terminal> avconv -r 4 -i frame_%04d.png -c:v flv movie.flv
Terminal> avconv -r 4 -i frame_%04d.png -c:v libtheora movie.ogg
Terminal> avconv -r 4 -i frame_%04d.png -c:v libx264 movie.mp4
Terminal> avconv -r 4 -i frame_%04d.png -c:v libpvx movie.webm

Important

Zero padding (%04d) is essential for correct sequence of frames in something_*.png (Unix alphanumeric sort)
Remove old frame_*.png files before making a new movie

Running a case

Vibrations of a guitar string
Triangular initial shape (at rest)

\[
I(x) = \begin{cases} 
\frac{ax}{x_0}, & x < x_0 \\
\frac{a(L-x)}{(L-x_0)}, & \text{otherwise} 
\end{cases}
\]

(26)

Appropriate data:

\[ L = 75 \text{ cm}, \quad x_0 = 0.8L, \quad a = 5 \text{ mm}, \quad \text{time frequency} \nu = 440 \text{ Hz} \]

Program: wave1D_u0.py

Implementation of the case

def guitar(C):
    """Triangular wave (pulled guitar string)."""
    L = 0.75
    x0 = 0.8*L
    a = 0.005
    freq = 440
    wavelength = 2*L
    c = freq/wavelength
    num_periods = 1
    T = 2*pi/omega*num_periods
    # Choose dt the same as the stability limit for Nx=50
    dt = L/50./c
    def I(x):
        return a*x/x0 if x < x0 else a/(L-x0)*(L-x)
    umin = -1.2*a; umax = -umin
cpu = viz(I, 0, 0, c, L, dt, C, T, umin, umax, animate=True, tool='scitools')

Program: wave1D_u0.py

Resulting movie for C = 0.8

Movie of vibrating string

The benefits of scaling

It is difficult to figure out all the physical parameters of a case.
And it is not necessary because of a powerful scaling.

Introduce new \( x, t, \) and \( u \) without dimension:

\[
\tilde{x} = \frac{x}{L}, \quad \tilde{t} = \frac{c}{L} t, \quad \tilde{u} = \frac{u}{a}
\]

Insert this in the PDE (with \( f = 0 \)) and dropping bars

Initial condition: set \( a = 1, \quad L = 1, \) and \( xp \in [0.1] \) it (26).

In the code: set \( \omega = 0.04\pi, \ \omega = 0.8, \) and there is no need to calculate with wave heights and frequency to estimate c!
Just one challenge: determine the period of the waves and an appropriate end time (see the text for details).

Vectorization

Problem: Python loops over long arrays are slow
One remedy: use vectorized (numpy) code instead of explicit loops
Other remedies: use Cython, port spatial loops to Fortran or C
Speedup: 100-1000 (varies with \( N_x \))

Next: vectorized loops
Operations on slices of arrays

- Introductory example: compute \( d_i = u_{i+1} - u_i \)

\[ n = u.size \]
\[ \text{for } i \in \text{range}(0, n-1): \]
\[ d[i] = u[i+1] - u[i] \]

- Note: all slice indices here are independent of each other.
- Example: \( d = (u_0, u_1, \ldots, u_{n-1}) \)
- If memory code: \( u[1:n] - u[0:n-2] \) or \( u[i+1] - u[i] \)

Vectorization of finite difference schemes (1)

Finite difference schemes basically contains differences between array elements with shifted indices. Consider the following formula:

\[ u_2[i-1] - 2u_2[i] + u_2[i+1] = \text{f}(x[i]) \]

The vectorization consists of replacing the loop by arithmetics on slices of arrays of length 2:

\[ u2[i-1] = u2[i-1] - 2u2[i] + u2[i+1] \]

Note: \( u2 \) has length 2.

If \( u2 \) is already an array of length 2, do update on "inner" elements:

\[ u2[i-1] = u2[i-1] - 2u2[i] + u2[i+1] \]

Vectorized implementation in the solver function

Scalar loop:

\[ \text{for } i \in \text{range}(1, \text{Nx}): \]
\[ u[i] = 2u_1[i] - u_2[i] + \text{f}(x[i]) \]

Vectorized loop:

\[ u[i-1] = u2[i-1] - 2u2[i] + u2[i+1] \]
\[ \text{or} \]
\[ u[i-1] = 2u_1[i] - u_2[i] + \text{f}(x[i]) \]

Program: wave1D_u0v.py

Vectorization of finite difference schemes (2)

Include a function evaluation too:

\[ \text{def f(x): return } x^2 + 1 \]

The scalar \( f \) value needs careful coding: return constant array if vectorized code, else number.

Test the understanding

Newcomers to vectorization are encouraged to choose a small array, e.g., with five elements, and simulate with pen and paper both the loop version and the vectorized version.
Efficiency measurements

- Run wave1D_u0v.py for \( N \times \) as 50, 100, 200, 400, 800 and measuring the CPU time
- Observe substantial speed-up: vectorized version is about \( N / 5 \) times faster

Much bigger improvements for 2D and 3D codes!

Generalization: reflecting boundaries

- Boundary condition \( u = 0 \): changes sign
- Boundary condition \( u_x = 0 \): wave is perfectly reflected
- How can we implement \( u_x \)? (more complicated than \( u = 0 \))

Demo of boundary conditions

Neumann boundary condition

\[
\frac{\partial u}{\partial n} \equiv n \cdot \nabla u = 0 \quad (27)
\]

For a 1D domain \([0, L]\):

\[
\frac{\partial}{\partial n} \bigg|_{x=L} = \frac{\partial}{\partial x}, \quad \frac{\partial}{\partial n} \bigg|_{x=0} = -\frac{\partial}{\partial x}
\]

Boundary condition terminology:

- \( u \) specified: Neumann condition
- \( u_x \) specified: Dirichlet condition

Discretization of derivatives at the boundary (1)

Problem: \( u^n_{n-1} \) is outside the mesh (extraneous value)

Remedy: use the stencil at the boundary to eliminate \( u^n_{n-1} \); just replace \( u^n_{n-1} \) by \( u^n_{n} \)

\[
u^n_{n+1} = -v^{n-1} + 2q^i + 2C^2 (v^{n+1}_{i+1} - v^n_i), \quad i = 0 \quad (28)
\]

Discretization of derivatives at the boundary (2)

Visualization of modified boundary stencil

Introduction: for computing \( u^3_0 \) in terms of \( u^0_0 \), \( u^1_0 \), and \( u^2_1 \):

Animation in a web page or a movie file.
Implementation of Neumann conditions

* Use the general stencil for interior points also on the boundary
* Replace \( u_{n+1}^i \) by \( u_{n-1}^i \) for \( i = 0 \)
* Replace \( u_{n+1}^i \) by \( u_{n-1}^i \) for \( i = N_x \)

\[
\begin{align*}
\text{for } i \text{ in range(0, } N_x+1): \\
\text{ip1 = } i+1 \text{ if } i < N_x \text{ else } i-1 \\
im1 = i-1 \text{ if } i > 0 \text{ else } i+1 \\
u[i] = u_1[i] + C2*(u_1[im1] - 2*u_1[i] + u_1[ip1])
\end{align*}
\]

Program wave1D_dn0.py

Moving finite difference stencil

Index set notation

* Tedious to write index sets like \( i = 0, \ldots, N_x \) and \( n = 0, \ldots, N_t \)
* Notation not valid if \( i \) or \( n \) starts at 1 instead...
* Both in math and code it is advantageous to use index sets
  \( i \in I_x \) instead of \( i = 0, \ldots, N_x \)
  \( n \in I_t \)

Definition: \( I_x = \{0, \ldots, N_x\} \)
The first index: \( i \in I_0 \)
The last index: \( i \in I_{N_x-1} \)

All interior points: \( i \in I_x^1 \)
\( I_x^1 = \{1, \ldots, N_x - 1\} \)
\( I_x^0 \) means \( \{0, \ldots, N_x\} \)
\( I_x^-1 = \{0, \ldots, N_x - 1\} \)

Index sets in action (1)

Index sets for a problem in the \( x, t \) plane:

\[
I_x = \{0, \ldots, N_x\}, \quad I_t = \{0, \ldots, N_t\}
\]

Implemented in Python as

```python
Ix = range(0, Nx+1)
It = range(0, Nt+1)
```

Program wave1D_dn.py

Index sets in action (2)

A finite difference scheme can with the index set notation be specified as

\[
\begin{align*}
\phi^{n+1} &= -\phi^n + 2\phi^{n-1} + C^2 (\phi^{n+1}_i - 2\phi^n_i + \phi^{n-1}_i), \quad i \in I_x^1, \quad n \in I_t^1 \\
\phi^n &= 0, \quad i \in I_x^0, \quad n \in I_t^1 \\
\phi^{n+1} &= 0, \quad i \in I_x^-1, \quad n \in I_t^1
\end{align*}
\]

Corresponding implementation:

```python
for n in range(1): 
    for i in range(1, N_x): 
        \n        \n        \n        \n        Program wave1D_dn.py
Alternative implementation via ghost cells

Instead of modifying the stencil at the boundary, we extend the mesh to cover
\( u_{n-1} \) and \( u_{n+1} \).
- The extra left and right cell are called ghost cells.
- The extra points are called ghost points.
- The \( u_{n-1} \) and \( u_{n+1} \) values are called ghost values.

Update ghost values as
\[
\begin{align*}
  u_{n,i-1} &= u_{n,i+1} \quad \text{for } i = 0 \\
  u_{n,i+1} &= u_{n,i-1} \quad \text{for } i = N_x
\end{align*}
\]
Then the stencil becomes right at the boundary.

Implementation of ghost cells (1)

Add ghost points:
\[
u = \text{zeros}(N_x+3)
\]
\[
u_1 = \text{zeros}(N_x+3)
\]
\[
u_2 = \text{zeros}(N_x+3)
\]
\[
x = \text{linspace}(0, L, N_x+1)
\]
# Mesh points without ghost points

A major indexing problem arises with ghost cells since Python indices must start at 0.
\[ u[-1] \] will always mean the last element in \( u \).
Math indexing: \[-1, 0, 1, 2, \ldots, N_x+1\]
Python indexing: \[0, \ldots, N_x+2\]
Remedy: use index sets

Implementation of ghost cells (2)

\[
u = \text{zeros}(N_x+3)
\]
\[
I_x = \text{range}(1, u.\text{shape}[0]-1)
\]
# Boundary values: \( u[I_x[0]] \), \( u[I_x[-1]] \)
# Set initial conditions
for \( i \) in \( I_x \):
  \[
u_1[i] = \phi(x[I_x[0]]) \quad \text{# Note } i-I_x[0]
  \]
# Loop over all physical mesh points
for \( i \) in \( I_x \):
  \[
u[i] = - u_2[i] + 2*u_1[i] + C2*(u_1[i-1] - 2*u_1[i] + u_1[i+1])
  \]
# Update ghost values
\[
i = I_x[0] \quad \text{# x=0 boundary}
\]
\[
i = I_x[-1] \quad \text{# x=L boundary}
\]
Program: wave1D_dn0_ghost.py.

Generalization: variable wave velocity

Heterogeneous media: varying \( c = c(x) \)

The model PDE with a variable coefficient

\[
\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x} \left( q(x) \frac{\partial u}{\partial x} \right) + f(x, t) \quad \text{(31)}
\]
This equation sampled at a mesh point \((x_i, t_n)\):
\[
\frac{\partial^2 u}{\partial t^2}(x_i, t_n) = \frac{\partial}{\partial x} \left( q(x_i) \frac{\partial u}{\partial x}(x_i, t_n) \right) + f(x_i, t_n)
\]

Discretizing the variable coefficient (1)

The principal idea is to first discretize the outer derivative.
Define
\[
\phi = q(x) \frac{\partial u}{\partial x}
\]
Then use a centered derivative around \( x = x_i \) for the derivative of \( \phi \)
\[
\left[ \frac{\phi}{\Delta x} \right]_i \approx \phi_{i+1} - \phi_{i-1} \frac{1}{\Delta x} = \left[ A_x \phi \right]_i
\]
Discretizing the variable coefficient (2)

Then discretize the inner operators:

\[ \phi_{i+\frac{1}{2}} - \phi_{i-\frac{1}{2}} \left( \frac{\partial}{\partial x} \right)^n_{x=j+\frac{1}{2}} \approx \frac{q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}}}{\Delta x} = [D_x q_x u]_{i+\frac{1}{2}} \]

Similarly,

\[ \phi_{i+\frac{1}{2}} - \phi_{i-\frac{1}{2}} \left( \frac{\partial}{\partial x} \right)^n_{x=j+\frac{1}{2}} \approx \frac{q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}}}{\Delta x} = [D_x q_x u]_{i+\frac{1}{2}} \]

Remark

Many are tempted to use the chain rule on the term \( \frac{\partial}{\partial x} (q(x)) \), but this is not a good idea!

Discretization of variable-coefficient wave equation in operator notation

These intermediate results are now combined to

\[ \left[ \frac{\partial}{\partial t} \left( q(x) \frac{\partial u}{\partial x} \right) \right]_{x} = \left[ \frac{1}{\Delta x^2} \left( q_{i+\frac{1}{2}} (q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}}) - q_{i-\frac{1}{2}} (q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}}) \right) \right] \]

Remark

We clearly see the type of finite differences and averaging!

I n operator notation:

\[ \left[ \frac{\partial}{\partial t} \left( q(x) \frac{\partial u}{\partial x} \right) \right]_{x} = \left[ \frac{1}{\Delta x^2} \left( q_{i+\frac{1}{2}} (q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}}) - q_{i-\frac{1}{2}} (q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}}) \right) \right] \]

Computing the coefficient between mesh points

Given \( q(x) \): compute \( q_{i+\frac{1}{2}} \) as \( q_{\frac{x}{2}} \)

Given \( x \) at the mesh points \( q_x \) use an average

\[ q_{i+\frac{1}{2}} = \frac{1}{2} \left( q_i + q_{i+1} \right) \]

(arithmetic mean) (34)

\[ q_{i+\frac{1}{2}} = \frac{1}{2} \left( \frac{1}{q_i} + \frac{1}{q_{i+1}} \right)^{\frac{1}{2}} \]

(la rynas mean) (35)

\[ q_{i+\frac{1}{2}} = \sqrt{q_i q_{i+1}} \]

(geometric mean) (36)

The arithmetic mean is (34) is by far the most used averaging technique.

Discretizing the variable coefficient (3)

These intermediate results are now combined to

\[ \left[ \frac{\partial}{\partial t} \left( q(x) \frac{\partial u}{\partial x} \right) \right]_{x} = \left[ \frac{1}{\Delta x^2} \left( q_{i+\frac{1}{2}} (q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}}) - q_{i-\frac{1}{2}} (q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}}) \right) \right] \]

Remark

Many are tempted to use the chain rule on the term \( \frac{\partial}{\partial x} (q(x) \frac{\partial u}{\partial x}) \)

Neumann condition and a variable coefficient

Consider \( \partial u / \partial x = 0 \) at \( x = L = N_x \Delta x \):

\[ q_{i+\frac{1}{2}} - q_{i-\frac{1}{2}} \left( \frac{\partial}{\partial x} \right)^n_{x=j+\frac{1}{2}} = 0 \]

There \( q_{i+\frac{1}{2}} = q_{i-\frac{1}{2}} \) in the stencil (30) for \( i = N_x \) and obtain

\[ q_{i+\frac{1}{2}} = q_{i-\frac{1}{2}} = q_{i-1} = q_{i+1} \]

(We have used \( q_{i+\frac{1}{2}} + q_{i-\frac{1}{2}} = 2 q_{i} \)).

Alternative: assume \( \partial u / \partial x = 0 \) (simpler).

Implementation of variable coefficients

Assume \( c[i] \) holds \( c_i \) the spatial mesh points:

\[ \text{for } i \in \text{range}(N_x), \text{ let:} \]

\[ q_{i+\frac{1}{2}} = q_{i+\frac{1}{2}} \left( \frac{\partial}{\partial x} \right)^n_{x=j+\frac{1}{2}} \]

In a nested format:

\[ q_{i+\frac{1}{2}} = q_{i+\frac{1}{2}} \left( \frac{\partial}{\partial x} \right)^n_{x=j+\frac{1}{2}} \]

For example:

\[ q_{i+\frac{1}{2}} = q_{i+\frac{1}{2}} \left( \frac{\partial}{\partial x} \right)^n_{x=j+\frac{1}{2}} \]

Neumann condition \( u_x = 0 \): same ideas as in 1D (modified stencil or ghost cell).
A more general model PDE with variable coefficients

\[ \rho(x)\frac{\partial^2 u}{\partial t^2} - \frac{\partial}{\partial x} \left( q(x) \frac{\partial u}{\partial x} \right) + f(x,t) \]  

(39)

A special case is

\[ \rho_{\text{eq}} \frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x} \left( q \frac{\partial u}{\partial x} \right) + f(x,t) \]  

(40)

Or

\[ \frac{\partial^2 u}{\partial t^2} - \frac{\partial}{\partial x} \left( q \frac{\partial u}{\partial x} \right) + f(x,t) \]  

(41)

No need to average \( \rho \), just sample at \( i \).

Generalization: damping

Why do waves die out?

Damping (non-elastic effects, air resistance)

2D/3D: conservation of energy makes an amplitude reduction by \( 1/\sqrt{t} \) (2D) or \( 1/t \) (3D)

Simplest damping model (for physical behavior, see demo):

\[ \rho(x)\frac{\partial^2 u}{\partial t^2} + \frac{\partial}{\partial x} \left( q(x) \frac{\partial u}{\partial x} \right) + f(x,t) \]  

(42)

\( \delta \geq 0 \): prescribed damping coefficient.

Dissipation via centered differences to ensure \( O(\Delta t^2) \) error:

\[ \frac{\partial^2 u}{\partial t^2} + \frac{\partial}{\partial x} \left( q \frac{\partial u}{\partial x} \right) + f(x,t) \]  

(43)

Need special formulas for \( s^2 \) + specialized (or ghost cells) for Neumann conditions.

Building a general 1D wave equation solver

The program `wave1D_dn_vc.py` solves a fairly general 1D wave equation:

\[
\begin{align*}
\rho(x)\frac{\partial^2 u}{\partial t^2} &= \frac{\partial}{\partial x} \left( q(x) \frac{\partial u}{\partial x} \right) + f(x,t) \\
\text{or} \\
\frac{\partial^2 u}{\partial t^2} &= \frac{\partial}{\partial x} \left( q \frac{\partial u}{\partial x} \right) + f(x,t)
\end{align*}
\]

(44)

(45)

No need to average \( \rho \), just sample at \( i \).

Collection of initial conditions

The function `pulse` in `wave1D_dn_vc.py` offers four initial conditions:

1. a rectangular pulse ("plug")
2. a Gaussian function (gaussian)
3. a "cosine hat": half a period of \( \cos(x) \), \( x \in [-1,1] \)
4. a "cosine hat": half a period of \( \cos(x) \), \( x \in [-\frac{1}{2}, \frac{1}{2}] \)

Can locate the initial pulse at \( x = 0 \) or in the middle

```
>>> import wave1D_dn_vc as w
>>> w.pulse(loc='left', pulse_tp='cosinehat', Nx=50, every_frame=10)
```

Finite difference methods for 2D and 3D wave equations

Consistent wave velocity:

\[ \frac{\partial^2 u}{\partial t^2} - c^2 \nabla^2 u \quad \text{for} \quad x \in \Omega \subset \mathbb{R}^2, \; t \in (0, T] \]  

(46)

Variable wave velocity:

\[ \frac{\partial^2 u}{\partial t^2} - \nabla \left( q \nabla u \right) + f \quad \text{for} \quad x \in \Omega \subset \mathbb{R}^2, \; t \in (0, T] \]  

(47)

Examples on wave equations written out in 2D/3D

\[
\begin{align*}
\nabla^2 u &= \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \\
\text{2D, variable:} \\
q(x,y) \frac{\partial^2 u}{\partial x^2} &= \frac{\partial}{\partial x} \left( q(x,y) \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( q(x,y) \frac{\partial u}{\partial y} \right) + f(x,y,t)
\end{align*}
\]

(51)

Compact notation:

\[
\begin{align*}
\nabla u &= (\partial_x u) + (\partial_y u) + (\partial_z u) + f \\
\nabla^2 u &= (\partial_x u)_x + (\partial_y u)_y + (\partial_z u)_z + f
\end{align*}
\]

(52)

(53)
Boundary and initial conditions

We need one boundary condition at each point on \( \partial \Omega \):
- \( u \) is prescribed \((u = 0 \) or known incoming wave)\)
- \( \partial u/\partial n - a \nabla u \) prescribed \((- \partial \) reflecting boundary)\)
- open boundary \( (\text{radiation}) \) condition: \( u + c \nabla u = 0 \) \( (\text{ke waves travel undisturbed out of the domain}) \)

PDEs with second-order time derivative need two initial conditions:
- \( u = I, \)
- \( u_t = V. \)

Discretization

\[ [(D_x D_x u) = c^2(D_x D_x u + D_y D_y u) + f_{ijk}]. \]

Write out in detail:
\[ \frac{u_{j}^{n+1} - 2u_{j}^{n} + u_{j}^{n-1}}{\Delta t^2} = c^2(u_{j+1}^{n+1} - 2u_{j}^{n+1} + u_{j-1}^{n+1}) \]
\[ + \frac{u_{j-1}^{n} - 2u_{j}^{n} + u_{j+1}^{n}}{\Delta t^2} + D_y D_y u \]

\( u_{j}^{n+1} \) and \( u_{j}^{n-1} \) are known; solve for \( u_{j}^{n+1} \):
\[ u_{j}^{n+1} = 2u_{j}^{n} - u_{j}^{n-1} + c^2 \Delta t^2(D_x D_x u + D_y D_y u) \]

Mesh

- Mesh points: \( (x, y, z, t_n) \)
- \( x \) direction: \( x_0 < x_1 < \cdots < x_N \)
- \( y \) direction: \( y_0 < y_1 < \cdots < y_M \)
- \( z \) direction: \( z_0 < z_1 < \cdots < z_S \)
- \( u_{j}^{n+1} = \phi(x, y, z, t_n) \)

Special stencil for the first time step

- The stencil for \( u_1 \) \((n=0)\) involves \( u_{j}^{1} \) which is outside the time mesh.
- For newly introduced \( u_1 \) \((v_1)\), use the discretized \( u_1 = V \) and the stencil for \( n = 0 \) to develop a special stencil (as in the 1D case).

\[ [(D_x D_x u) = v_{i}^{1} - v_{j}^{1} - 2\Delta x v_{j}] \]
\[ v_{j}^{1} = v_{j}^{0} - \Delta x v_{j} + \frac{1}{2} \Delta x^2 D_y D_y u + D_x D_y u v_{j} \]

Variable coefficients (1)

3D wave equation:
\[ \partial u_n = (\partial u_n) + (\partial x_n) + (\partial y_n) + f(x, y, z, t) \]

3D wave equation:
\[ [(D_x D_x u) = (D_x D_x u + D_y D_y u + D_z D_z u) + f_{ijk}]. \]

Need special formulas for \( v_{j}^{1} \) (use \([D_x u = V]\) and stencil for \( n = 0 \)).

Variable coefficients (2)

Write out:
\[ v_{j}^{1} = -v_{j}^{1} + 2v_{j}^{0} \]
\[ + \frac{\Delta x}{\Delta x} (\frac{1}{2} \Delta x^2 (q_{j+k} + q_{j+k} + q_{j+k} - q_{j+k})) \]
\[ + \frac{\Delta x}{\Delta x} (\frac{1}{2} \Delta x^2 (q_{j+k} + q_{j+k} + q_{j+k} - q_{j+k})) \]
\[ + \Delta x^2 (q_{j+k} + q_{j+k} + q_{j+k} - q_{j+k}). \]

\[ \Delta x^2 (q_{j+k} + q_{j+k} + q_{j+k} - q_{j+k}) + \]
Neumann boundary condition in 2D

Use ideas from 1D: Example: \[ \frac{\partial^2 u}{\partial x^2} = 0 \text{ at } x = 0, \quad \frac{\partial^2 u}{\partial y^2} = 0 \]

Boundary condition discretization:

\[ [-\partial_y u = 0] \quad \Rightarrow \quad \frac{u_{i}^{j+1} - u_{i}^{j}}{\Delta y} = 0, \quad i \in I_x \]

In 2D, \( u_{i}^{j} = u_{i}^{j-1} \) is the stencil for \( u_{i}^{j} \) so obtain a modified stencil for the boundary.

Pattern: use interior stencil also on the boundary, but replace \( j-1 \) by \( j+1 \).

Alternative: use ghost cells and ghost values

\[ \Omega = [0, L_x] \times [0, L_y] \]

Discretization:

\[ [D_t D_t u - c^2 (D_x D_x u + D_y D_y u) + f_{D_y}] \]

Implementation of 2D/3D problems

\[ u_{t} = c^2 (u_{x} + u_{y}) + f(x, y, t), \quad (x, y) \in \Omega, \quad t \in (0, T) \] (55)

\[ u(x, y, 0) = f(x, y), \quad (x, y) \in \Omega \] (56)

\[ u_{t}(x, y, 0) = v(x, y), \quad (x, y) \in \Omega \] (57)

\[ u = 0, \quad (x, y) \in \partial \Omega, \quad t \in (0, T) \] (58)

\[ \Omega = [0, L_x] \times [0, L_y] \]

Discretization:

\[ \frac{[D_t D_t u - c^2 (D_x D_x u + D_y D_y u) + f_{D_y}]}{\Delta t} \]

Scalar computations: mesh

Program: wave2D_u0.py

```python
def solver(I, V, f, c, Lx, Ly, Nx, Ny, dt, T, user_action=None, version='scalar'):
    # mesh points in x dir
    x = linspace(0, Lx, Nx+1)
    if user_action is not None:
        # mesh points in y dir
        y = linspace(0, Ly, Ny+1)
        for i in Ix:
            Iy = range(0, u.shape[0])
            for j in Iy:
                u_2 = zeros((Nx+1,Ny+1)) # solution at t-2*dt
                u_1 = zeros((Nx+1,Ny+1)) # solution at t-dt
                u = zeros((Nx+1,Ny+1)) # solution array
                u_2[i,j] = I(x[i], y[j])
                for j in Iy:
                    y = linspace(0, Ly, Ny+1)
                    for i in Ix:
                        I = range(0, u.shape[1])
                        u[I,0] = I(x[I], y[0])
                        if user_action is not None:
                            user_action(u[I,0], x[I], xv[I], y, yv[I], t, 0)
                        Arguments xv and yv, for vectorized computations
```

Scalar computations: arrays

Since \( u_{i}^{j+1}, u_{i}^{j}, \) and \( u_{i}^{j-1} \) are in three two-dimensional arrays:

\[ u = zeros((Nx, Ny)) \quad \# solution array
u_2 = zeros((Nx, Ny)) \quad \# solution at t-2*dt
u_1 = zeros((Nx, Ny)) \quad \# solution at t-dt
u_0 = zeros((Nx, Ny)) \quad \# solution at t-0*dt
\]

\( u_{i}^{j+1} \) corresponds to \( u[I,j] \), etc.
Vectorized computations: primary stencil

```python
def advance_scalar(u, u_1, u_2, f, x, y, t, n, Cx2, Cy2, dt2, V=None, ... u_1[:,i,j] = I(xv, yv)
f_a[:,i,j] = f(xv, yv, t)
```

Vectorized computations: mesh coordinates

Mesh with 30 × 30 cells: vectorization reduces the CPU time by a factor of 10 (1).

Need special coordinate arrays x and y so that f(x, y) and f(x, y, t) can be vectorized:

```python
def I(x, y):...return f(xv, yv, t)
```

Verification: quadratic solution (1)

Manufactured solution:

\[ u_e(x, y, t) = x(L_x - x)y(L_y - y)(1 + \frac{1}{2}t) \]  

Requires f = 2c^2(1 + \frac{1}{2}t)(y(L_y - y) + x(L_x - x)).

This u_e is ideal because it also solves the discrete equations!

Analysis of the difference equations

- \([D_1,D_2]^T = 0\)
- \([D_1,D_2]f = 0\)
- \([D_1,D_2]^2 = 0\)
- \(D_1, D_2\) is a linear operator:

\[
[D_1,D_2](u + bv)^T = D_1(u,D_2u)^T + b[D_1,D_2]^T
\]

\[
[D_1,D_2]^T u_1^T = [y(y - y)(1 + \frac{1}{2}t)D_1,D_2 u_1^T]_{1}\]

\[
= y(y - y)(1 + \frac{1}{2}t)\]

Similar calculations for \([D_1,D_2]^T u_1^T\) and \([D_1,D_2]^T u_1^T\) terms

Must also check the equation for \(u_1\)
Properties of the solution of the wave equation

\[
\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}
\]

Solutions:

\[
u(x, t) = g_L(x - ct) + g_R(x + ct)
\]

If \( u(x, 0) = f(x) \) and \( u_t(x, 0) = 0 \):

\[
u(x, t) = \frac{1}{2} f(x - ct) + \frac{1}{2} f(x + ct)
\]

Two waves: one traveling to the right and one to the left.

Let us change the shape of the initial condition slightly and see what happens.

A similar wave component is also a solution of the finite difference scheme (!)

Idea: a similar discrete solution (corresponding to the exact solution) solves

\[
[D, D_t u = c^2 \Delta x \Delta t u]
\]

Note: we expect numerical frequency \( \tilde{\omega} \neq \omega \)

- How accurate is \( \tilde{\omega} \) compared to \( \omega \)?
- What about the wave amplitude (can \( \tilde{\omega} \) become complex)?

Simulation of a case with variable wave velocity

A wave propagates perfectly (\( C = 1 \)) and hits a medium with 1/4 of the wave velocity (\( C = 0.25 \)). A part of the wave is reflected and the rest is transmitted.

Representation of waves as sum of sine/cosine waves

Build \( I(x) \) of wave components \( e^{ikx} = \cos(kx) + i \sin(kx) \):

\[
I(x) = \sum_{k \in K} b_k e^{ikx}
\]

- For \( b_k \) by least squares or projection method
- \( k \) is the frequency of a component (\( \lambda = 2\pi/k \) is the wave length in space)
- \( K \) is some set of all \( k \) needed to approximate \( I(x) \) well
- \( b_k \) must be computed (Fourier coefficients)

Since \( u(x, t) = \frac{1}{2} f(x - ct) + \frac{1}{2} f(x + ct) \), the exact solution is

\[
u(x, t) = \frac{1}{2} \sum_{k \in K} b_k e^{ik(x-ct)} + \frac{1}{2} \sum_{k \in K} b_k e^{ik(x+ct)}
\]

Our interest: one component \( e^{ik(x-ct)} \), \( \omega = kc \)

Preliminary results

\[
[D, D_t e^{ik \omega t}]^\Delta t = -\frac{4}{\Delta x^2} \sin^2 \left( \frac{\omega \Delta t}{2} \right) e^{ik \omega t}
\]

By \( \omega \to k \), \( t \to x \), \( n \to q \) it follows that

\[
[D, D_t e^{ikx}]^\Delta x = -\frac{4}{\Delta x^2} \sin^2 \left( \frac{k \Delta x}{2} \right) e^{ikx}
\]
The special case

Inserting the numerical wave component

Inserting a basic wave component \( u = e^{i(kx - \omega t)} \) in the scheme requires computation of

\[
[D_t D_t e^{i(kx - \omega t)}]_n = [D_t D_t e^{i(kx - \omega t)}]_n = -\frac{4}{\Delta t^2} \sin^2 \left( \frac{k \Delta x}{2} \right) e^{i(kx - \omega t)}
\]

\[
[D_t D_x e^{i(kx - \omega t)}]_n = [D_t D_x e^{i(kx - \omega t)}]_n = -\frac{4}{\Delta x^2} \sin^2 \left( \frac{k \Delta x}{2} \right) e^{i(kx - \omega t)}
\]

The equation for \( \omega \)

The complete scheme,

\[
[D_t D_t e^{i(kx - \omega t)}] = \frac{\partial^2}{\partial x^2} e^{i(kx - \omega t)}
\]

leads to an equation for \( \omega \) (which can readily be solved):

\[
\sin \left( \frac{\omega \Delta t}{2} \right) = C \sin \left( \frac{k \Delta x}{2} \right)
\]

Taking the square root:

\[
\sin \left( \frac{\omega \Delta t}{2} \right) = C \sin \left( \frac{k \Delta x}{2} \right)
\]

The numerical dispersion relation

Can easily solve for an explicit formula for \( \omega \):

\[\omega = \frac{2}{\Delta t} \sin^{-1} \left( C \sin \left( \frac{k \Delta x}{2} \right) \right)\]

Note:

- This \( \omega = \omega(k, c, \Delta x, \Delta t) \) is the numerical dispersion relation
- Inserting \( e^{i(kx - \omega t)} \) in the PDE leads to \( \omega = kc \), which is the analytical/exact dispersion relation
- Speed of waves might be easier to imagine:
  - Exact waves: \( c = \omega / k \)
  - Numerical waves: \( \tilde{c} = \omega / k \)
- We shall investigate \( \tilde{c} / c \) to see how wrong the speed of a numerical wave component is

Computing the error in wave velocity

- Introduce \( p = k \Delta x / 2 \) (the important dimensionless spatial discretization parameter)
- \( p \) measures no of mesh points in space per wave length in space
- Smallest possible wave length is mesh, \( \lambda = 2 \Delta x \), \( k = 2\pi / \lambda = \pi / \Delta x \) and \( p = k \Delta x / 2 = \pi / 2 \Rightarrow p \in [0, \pi / 2] \)
- Study error in wave velocity through \( \tilde{c} / c \) as function of \( p \)

\[
r(C, p) = \frac{\tilde{c}}{c} = \frac{2}{k \Delta x} \sin^{-1} (C \sin p) - \frac{2}{k \Delta x} \sin^{-1} \left( \frac{1}{\Delta x} \sin^2 p \right)
\]

Can plot \( r(C, p) \) for \( p \in [0, \pi / 2], C \in [0, 1] \)

Visualizing the error in wave velocity

Note: the slower waves have the larger error and vice versa.
Taylor expanding the error in wave velocity

For small $p$, Taylor expand $\tilde{\omega}$ as polynomial in $p$:

```python
>>> C, p = symbols('C p')
>>> rs = r(C, p).series(p, 0, 7)
>>> print rs
1 - p**2/6 + p**4/120 - p**6/5040 + C**2*p**2/6 - C**2*p**4/12 + 13*C**2*p**6/720 + 3*C**4*p**4/40 - C**4*p**6/16 + 5*C**6*p**6/112 + O(p**7)
```

>>>

# Drop the remainder $O(...)$ term
```python
>>> rs = rs.removeO()
```

# Factorize each term
```python
>>> rs = [factor(term) for term in rs.as_ordered_terms()]
>>> rs = sum(rs)
>>> print rs
p**6*(C - 1)*(C + 1)*(225*C**4 - 90*C**2 + 1)/5040 + p**4*(C - 1)*(C + 1)*(3*C - 1)*(3*C + 1)/120 + p**2*(C - 1)*(C + 1)/6 + 1
```

Leading error term is $1$.

Example on effect of wrong wave velocity (1)

- Smooth wave, few short waves (large $k$) in $I(x)$:
- Not so smooth wave, significant short waves (large $k$) in $I(x)$:

Stability

$$\sin \left( \frac{\Delta x}{2} \right) - C \sin \left( \frac{\Delta x}{2} \right)$$

- Exact $\omega$ is not.
- Complex $\tilde{\omega}$ will lead to exponential growth of the amplitude.
- Stability criterion: $\Re{\tilde{\omega}} < 0$.
- $\sin(\Delta x/2) \in [-1, 1]$.
- $\Delta x/2$ is always real, so right-hand side is in $[-C, C]$.
- Thus we must have $C \leq 1$.

Stability criterion:

$$C = \frac{c \Delta t}{\Delta x} \leq 1$$

Extending the analysis to 2D (and 3D)

Recall that right-hand side is in $[-C, C]$. Thus $C > 1$ means

$$\sin \left( \frac{\Delta x}{2} \right) - C \sin \left( \frac{\Delta x}{2} \right)$$

- $|\sin x| > 1$ implies complex $x$.
- Here $\tilde{\omega}$ complex: $\tilde{\omega} = \tilde{\omega}_R \pm i \tilde{\omega}_I$.
- If $\Delta x < 0$ gives $\exp\left(-i \tilde{\omega} \Delta t\right)$ and exponential growth.
- This wave component will after some time dominate the solution giving an overall exponentially increasing amplitude (non-physical).

Recall that right-hand side is in $[-C, C]$. Thus $C > 1$ means

$$\sin \left( \frac{\Delta x}{2} \right) - C \sin \left( \frac{\Delta x}{2} \right)$$

- $|\sin x| > 1$ implies complex $x$.
- Here $\tilde{\omega}$ complex: $\tilde{\omega} = \tilde{\omega}_R \pm i \tilde{\omega}_I$.
- If $\Delta x < 0$ gives $\exp\left(-i \tilde{\omega} \Delta t\right)$ and exponential growth.
- This wave component will after some time dominate the solution giving an overall exponentially increasing amplitude (non-physical).

Thus $C > 1$ leads to non-physical waves.

Why $C > 1$ leads to non-physical waves

Recall that right-hand side is in $[-C, C]$. Thus $C > 1$ means

$$\sin \left( \frac{\Delta x}{2} \right) - C \sin \left( \frac{\Delta x}{2} \right)$$

- $|\sin x| > 1$ implies complex $x$.
- Here $\tilde{\omega}$ complex: $\tilde{\omega} = \tilde{\omega}_R \pm i \tilde{\omega}_I$.
- If $\Delta x < 0$ gives $\exp\left(-i \tilde{\omega} \Delta t\right)$ and exponential growth.
- This wave component will after some time dominate the solution giving an overall exponentially increasing amplitude (non-physical).

Extending the analysis to 2D (and 3D)

Recall that right-hand side is in $[-C, C]$. Thus $C > 1$ means

$$\sin \left( \frac{\Delta x}{2} \right) - C \sin \left( \frac{\Delta x}{2} \right)$$

- $|\sin x| > 1$ implies complex $x$.
- Here $\tilde{\omega}$ complex: $\tilde{\omega} = \tilde{\omega}_R \pm i \tilde{\omega}_I$.
- If $\Delta x < 0$ gives $\exp\left(-i \tilde{\omega} \Delta t\right)$ and exponential growth.
- This wave component will after some time dominate the solution giving an overall exponentially increasing amplitude (non-physical).

Extended solutions by adding complex Fourier components of the form

$$\sum_k g_k e^{ik(x,y)} \phi_k$$

Recall that right-hand side is in $[-C, C]$. Thus $C > 1$ means

$$\sin \left( \frac{\Delta x}{2} \right) - C \sin \left( \frac{\Delta x}{2} \right)$$

- $|\sin x| > 1$ implies complex $x$.
- Here $\tilde{\omega}$ complex: $\tilde{\omega} = \tilde{\omega}_R \pm i \tilde{\omega}_I$.
- If $\Delta x < 0$ gives $\exp\left(-i \tilde{\omega} \Delta t\right)$ and exponential growth.
- This wave component will after some time dominate the solution giving an overall exponentially increasing amplitude (non-physical).

Extended solutions by adding complex Fourier components of the form

$$\sum_k g_k e^{ik(x,y)} \phi_k$$
Discrete wave components in 2D

\[ \Delta t, \Delta u = c^2(D_x^2 u + D_y^2 u) \]

This equation admits a Fourier component

\[ \tilde{q}_{\ell, \mu} = \int_\Omega q \, e^{-2\pi i (\ell x + \mu y)} \, dx \, dy \]

Inserting the Fourier component into the discrete 2D wave equation, and using formulas from the 1D analysis:

\[ \sin^2 \left( \frac{\tilde{c} \Delta t}{2} \right) = c^2 \sin^2 p_x + c^2 \sin^2 p_y \]

where

\[ c_x = \frac{c \Delta t}{\Delta x}, \quad c_y = \frac{c \Delta t}{\Delta y}, \quad \cos \theta = \frac{p_x - \frac{1}{2} k_x \Delta x, \quad \sin \theta = \frac{p_y - \frac{1}{2} k_y \Delta y}{2} \]

Numerical dispersion relation in 2D (1)

\[ \tilde{\omega} = 2 \Delta t \sin^{-1} \left( \sqrt{c_x^2 \sin^2 p_x + c_y^2 \sin^2 p_y} \right) \]

For visualization, introduce \( k = \sqrt{k_x^2 + k_y^2} \) and \( \theta \) such that

\[ k_x = k \sin \theta, \quad k_y = k \cos \theta, \quad p_x = \frac{1}{2} k \cos \theta, \quad p_y = \frac{1}{2} k \sin \theta \]

Also, \( \Delta x = \Delta y = h \). Then \( C_x = C_y = \frac{c \Delta t}{\Delta x} = C \).

Now \( \tilde{\omega} \) depends on:

- \( C \): reflecting the number cells a wave is displaced during a time step
- \( k \): reflecting the number of cells per wave length in space
- \( \theta \): expressing the direction of the wave

Stability criterion in 3D

\[ \Delta t \leq \frac{1}{\tilde{\omega}} \left( \frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2} \right)^{-1/2} \]

For \( c^2 = c^2(x) \) we must use the worst-case value

\[ c = \sqrt{\max_{x \in \Omega} c^2(x)} \]

and a safety factor \( \beta \leq 1 \):

\[ \Delta t \leq \beta \frac{1}{\tilde{\omega}} \left( \frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2} \right)^{-1/2} \]

Numerical dispersion relation in 2D (2)

\[ \tilde{\omega} = 2 \Delta t \sin^{-1} \left( \sqrt{c_x^2 \sin^2 p_x + c_y^2 \sin^2 p_y} \right) \]

Can make color contour plots of \( 1 - \tilde{\omega}/c \) in polar coordinates with \( \theta \) as the angular coordinate and \( k \) as the radial coordinate.