Study guide: Finite difference methods for vibration problems

Hans Petter Langtangen[1,2] Svein Linge[3,1]

Center for Biomedical Computing, Simula Research Laboratory[1]

Department of Informatics, University of Oslo[2]

Department of Process, Energy and Environmental Technology, University College of Southeast Norway[3]

Aug 28, 2023

Slides selected/modified by Mikael Mortensen

A simple vibration problem

$$u''(t) + \omega^2 u = 0, \quad u(0) = I,\ u'(0) = 0,\ t \in (0, T]$$

Exact solution:

$$u(t) = I \cos(\omega t)$$

$u(t)$ oscillates with constant amplitude I and (angular) frequency ω.

Period: $P = 2\pi/\omega$.

A centered finite difference scheme; step 1 and 2

Strategy: follow the "four steps" of the finite difference method.

- Step 1: Introduce a time mesh, here uniform on $[0, T]$: $t_n = n\Delta t$
- Step 2: Let the ODE be satisfied at each mesh point:

$$u''(t_n) + \omega^2 u(t_n) = 0, \quad n = 2, \ldots, N_t$$

Notice

u^0 and u^1 are obtained from initial conditions.

A centered finite difference scheme; step 3

Step 3: Approximate derivative(s) by finite difference approximation(s). Very common (standard!) formula for u'':

$$u''(t_n) \approx \frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2}$$

Use this in the ODE for $n = 1, 2, \ldots, N_t - 1$

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} = -\omega^2 u^n$$

Notice

We thus solve for $u^2, u^3, \ldots, u^{N_t}$.

A centered finite difference scheme; step 4

Step 4: Formulate the computational algorithm. Assume u^{n-1} and u^n are known, solve for unknown u^{n+1}:

$$u^{n+1} = 2u^n - u^{n-1} - \Delta t^2\omega^2 u^n$$

Nick names for this scheme: Störmer's method or Verlet integration.

Computing the first step - alternative 1

- Two initial conditions $u(0) = I, \quad u'(0) = 0$
- $u^0 = u(0) = I$ is already fixed. What about u^1? Need to use $u'(0) = 0$ somehow.

Alternative 1: Use a forward difference:

$$u'(0) = \frac{u^1 - u^0}{\Delta t} \qquad \longrightarrow u^1 = u^0 = I$$

$$u'(0) = \frac{-u^2 + 4u^1 - 3u^0}{2\Delta t} \qquad \longrightarrow u^1 = \frac{u^2 + 3u^0}{4}$$

Notice

First is merely first order accurate, second is second order, but implicit (depends on the unknown u^2.)

Computing the first step - alternative 2

Use the discrete ODE at $t = 0$ together with a central difference at $t = 0$ and a ghost cell u^{-1}. The central difference is

$$u'(0) = \frac{u^1 - u^{-1}}{2\Delta t} \quad \longrightarrow u^{-1} = u^1$$

The central ODE at $n = 0$ is:

$$u^1 = 2u^0 - u^{-1} - \Delta t^2 \omega^2 u^0$$

Insert for ghost cell u^{-1} and obtain

$$u^1 = u^0 - \frac{1}{2}\Delta t^2 \omega^2 u^0$$

Remark

Alternative 2 is favoured because the first order forward difference is inaccurate and the second order is implicit.

The computational algorithm

1. $u^0 = I$
2. compute u^1 with alternative 2
3. for $n = 1, 2, \ldots, N_t - 1$:
    - compute u^{n+1}

More precisely expressed in Python:

```
import numpy as np
t = np.linspace(0, T, Nt+1)  # mesh points in time
dt = t[1] - t[0]             # constant time step.
u = np.zeros(Nt+1)           # solution

u[0] = I
u[1] = u[0] - 0.5*dt**2*w**2*u[0]
for n in range(1, Nt):
    u[n+1] = 2*u[n] - u[n-1] - dt**2*w**2*u[n]
```

Note: `w` is used in code for ω.

Computing u'

u is often displacement/position, u' is velocity and can be computed by a second order central difference

$$u'(t_n) \approx \frac{u^{n+1} - u^{n-1}}{2\Delta t} = [D_{2t}u]^n$$

For $u'(t_0)$ and $u'(t_{N_t})$ it is possible to use forward or backwards differences, respectively. However, we know from initial conditions that $u'(t_0) = 0$.

Implementation and verification

Move to notebook

More mathematical analysis

The exact solution to the continuous vibration equation is

$$u_e(t) = I\cos(\omega t)$$

An exact discrete solution is

$$u(t_n) = I\cos(\tilde{\omega} t_n)$$

We can study the error in $\tilde{\omega}$ compared to the true ω

Find the truncation error

Insert the numerical solution $u^n = I\cos(\tilde{\omega} t_n)$ into the discrete equation

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} + \omega^2 u^n = 0$$

Quite messy, but Wolfram Alpha (or a long derivation in the book) will give you

$$\begin{aligned}
\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} &= \frac{I}{\Delta t^2}(\cos(\tilde{\omega} t_{n+1}) - 2\cos(\tilde{\omega} t_n) + \cos(\tilde{\omega} t_{n-1})) \quad (1) \\
&= \frac{2I}{\Delta t^2}(\cos(\tilde{\omega}\Delta t) - 1)\cos(\tilde{\omega} n\Delta t) \quad (2) \\
&= -\frac{4}{\Delta t^2}\sin^2(\tilde{\omega}\Delta t)\cos(\tilde{\omega} n\Delta t) \quad (3)
\end{aligned}$$

Insert into discrete equation

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} + \omega^2 u^n = 0$$

We get

$$-\frac{4}{\Delta t^2}\sin^2(\tilde{\omega}\Delta t)\cos(\tilde{\omega} n\Delta t) + \omega^2\cos(\tilde{\omega} n\Delta t) = 0$$

and thus

$$\omega^2 = \frac{4}{\Delta t^2}\sin^2\left(\frac{\tilde{\omega}\Delta t}{2}\right)$$

Solve for $\tilde{\omega}$ ..

Numerical frequency

$$\tilde{\omega} = \pm \frac{2}{\Delta t} \sin^{-1}\left(\frac{\omega \Delta t}{2}\right)$$

- Frequency error because $\tilde{\omega} \neq \omega$.
- Note: dimensionless number $p = \omega \Delta t$ is the key parameter <linebreak> (i.e., no of time intervals per period is important, not Δt itself)
- But how good is the approximation $\tilde{\omega}$ to ω?

Polynomial approximation of the frequency error

Taylor series expansion for small Δt gives a formula that is easier to understand:

```
>>> from sympy import *
>>> dt, w = symbols('dt w')
>>> w_tilde = asin(w*dt/2).series(dt, 0, 4)*2/dt
>>> print w_tilde
(dt*w + dt**3*w**3/24 + O(dt**4))/dt  # note the final "/dt"
```

$$\tilde{\omega} = \omega\left(1 + \frac{1}{24}\omega^2\Delta t^2\right) + \mathcal{O}(\Delta t^3)$$

The numerical frequency is too large (to fast oscillations).

Simple improvement of previous solver

Notice

What happens if we use $\omega = \omega(1 - \omega^2\Delta t^2/24)$?

The leading order numerical error disappears and

$$\tilde{\omega} = \omega\left(1 - \left(\frac{1}{24}\omega^2\Delta t^2\right)^2\right) + +$$

Notice

Dirty trick, and only usable when you can compute the numerical error exactly

The global error

$$u^n = I\cos\left(\tilde{\omega} n\Delta t\right), \quad \tilde{\omega} = \frac{2}{\Delta t}\sin^{-1}\left(\frac{\omega\Delta t}{2}\right)$$

The error mesh function,

$$e^n = u_e(t_n) - u^n = I\cos\left(\omega n\Delta t\right) - I\cos\left(\tilde{\omega} n\Delta t\right)$$

is ideal for verification and further analysis!

$$\begin{aligned} e^n &= I\cos\left(\omega n\Delta t\right) - I\cos\left(\tilde{\omega} n\Delta t\right) \\ &= -2I\sin\left(t\frac{1}{2}\left(\omega - \tilde{\omega}\right)\right)\sin\left(t\frac{1}{2}\left(\omega + \tilde{\omega}\right)\right) \end{aligned}$$

Convergence of the numerical scheme

Can easily show *convergence*:

$$e^n \rightarrow 0 \text{ as } \Delta t \rightarrow 0,$$

because

$$\lim_{\Delta t\rightarrow 0} \tilde\omega = \lim_{\Delta t\rightarrow 0} \frac{2}{\Delta t}\sin^{-1}\left(\frac{\omega\Delta t}{2}\right) = \omega,$$

by L'Hopital's rule or simply asking `sympy`: or WolframAlpha:

```
>>> import sympy as sym
>>> dt, w = sym.symbols('x w')
>>> sym.limit((2/dt)*sym.asin(w*dt/2), dt, 0, dir='+')
w
```

Stability

Observations:

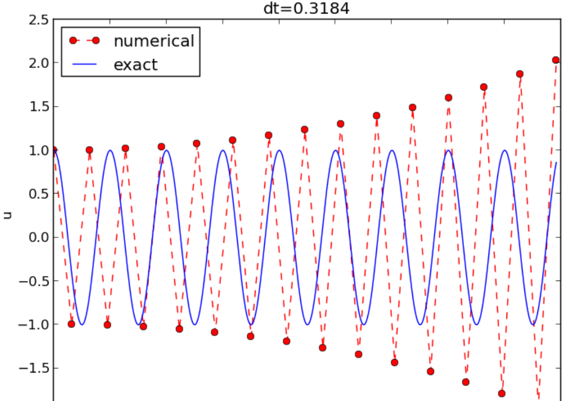
- Numerical solution has constant amplitude (desired!), but an angular frequency error
- Constant amplitude requires $\sin^{-1}(\omega \Delta t/2)$ to be real-valued $\Rightarrow |\omega \Delta t/2| \leq 1$
- $\sin^{-1}(x)$ is complex if $|x| > 1$, and then $\tilde{\omega}$ becomes complex. Can be shown that this leads to error in amplitude.

The stability criterion

Cannot tolerate growth and must therefore demand a *stability criterion*

$$\frac{\omega \Delta t}{2} \leq 1 \quad \Rightarrow \quad \Delta t \leq \frac{2}{\omega}$$

Try $\Delta t = \frac{2}{\omega} + 9.01 \cdot 10^{-5}$ (*slightly* too big!):



Summary of the analysis

We can draw three important conclusions:

1. The key parameter in the formulas is $p = \omega\Delta t$ (dimensionless)
    1. Period of oscillations: $P = 2\pi/\omega$
    2. Number of time steps per period: $N_P = P/\Delta t$
    3. $\Rightarrow \ p = \omega\Delta t = 2\pi/N_P \sim 1/N_P$
    4. The smallest possible N_P is $2 \Rightarrow p \in (0, \pi]$
2. For $p \leq 2$ the amplitude of u^n is constant (stable solution)
3. u^n has a relative frequency error $\tilde{\omega}/\omega \approx 1 + \frac{1}{24}p^2$, making numerical peaks occur too early