Computational Physics

Partial Differential Equations
Finite difference time-domain

Prototypical linear PDEs

For understanding linear ODEs it was sufficient to focus on

$$\partial_t \mathbf{y} = A\mathbf{y},$$

because every linear ODE can be brought to this form.

This is not possible for PDEs.

However, there are a few prototypical ones that represent important classes:

- Advection equation: $\partial_t u + \operatorname{div} \mathbf{j}(u) = 0$ (first order PDEs)
- Laplace equation: $\Delta u = 0$, with $\Delta = \partial_x^2 + \partial_y^2 + \partial_z^2$. (elliptic PDEs)
- Diffusion equation: $\partial_t u - \Delta u = 0$ (parabolic PDEs)
- Wave equation: $\partial_t^2 u - \Delta u = 0$ (hyperbolic PDEs)

The three classes have different types of solutions and typically require different numerical methods.

Dimensionless units

As the example for today we consider the 1d acoustic wave equation

$$\rho_0 \partial_t^2 p(x,t) - \beta^{-1} \partial_x^2 p(x,t) = 0,$$

with equilibrium mass density ρ_0, compressibility β and local pressure p.

First, we transform the problem to dimensionless units, i.e. transform time, space and/or p such that ρ and β disappear.

Absorb factor β^{-1} in the function $p(x,t)$:

$$[\beta \rho_0 \partial_t^2 - \partial_x^2] \underbrace{\beta^{-1} p(x,t)}_{\tilde{p}(x,t)} = 0,$$

Introduce new time variable τ (unit of space is still free):

$$t = \sqrt{\beta \rho_0} \tau, \quad \Rightarrow \quad \partial_\tau = \sqrt{\beta \rho_0} \partial_t.$$

$$\Rightarrow \quad [\partial_\tau^2 - \partial_x^2] \tilde{p}(x,t) = 0,$$

Simplest discretization: Finite differences

We represent the function $p(x, t)$ on a regular grid, i.e. time and space are integer multiples of mesh parameters h (our unit of space) and Δt:

$$p_i^{(j)} = p(x_i, t_j) = p(hi, \Delta tj).$$

The simplest approximation to the derivative on this are finite differences:

$$\partial_x p_i^{(j)} \approx h^{-1}[p_i^{(j)} - p_{i-1}^{(j)}] \quad \text{or} \qquad \partial_x p_i^{(j)} \approx h^{-1}[p_{i+1}^{(j)} - p_i^{(j)}].$$

This asymmetry disappears in the second derivative if we combine both formulas appropriately:

$$\begin{aligned}\partial_x^2 p_i^{(j)} \approx& h^{-1}[\partial_x p_{i+1}^{(j)} - \partial_x p_i^{(j)}] \approx h^{-2}[p_{i+1}^{(j)} - p_i^{(j)}] - h^{-2}[p_i^{(j)} - p_{i-1}^{(j)}] \\ =& h^{-2}[p_{i+1}^{(j)} - 2p_i^{(j)} + p_{i-1}^{(j)}].\end{aligned}$$

Naive time stepping

The same applies for the time derivative:

$$\begin{aligned} \partial_t^2 p_i^{(j)} &\approx \Delta t^{-2}[p_i^{(j+1)} - 2p_i^{(j)} + p_i^{(j-1)}], \\ \Rightarrow \quad p_i^{(j+1)} &\approx 2p_i^{(j)} - p_i^{(j-1)} + \Delta t^2 \partial_t^2 p_i^{(j)}, \end{aligned}$$

The second time derivative is given by the spatial derivatives through the wave equation:

$$\partial_t^2 p = \partial_x^2 p.$$

We insert in the time-stepping equation:

$$p_i^{(j+1)} \approx 2p_i^{(j)} - p_i^{(j-1)} + \Delta t^2 h^{-2}[p_{i+1}^{(j)} - 2p_i^{(j)} + p_{i-1}^{(j)}].$$

Example 1: Naive finite difference method

Implement this simple finite-difference method in MATLAB:

$$p_i^{(j+1)} \approx 2p_i^{(j)} - p_i^{(j-1)} + \Delta t^2 h^{-2}[p_{i+1}^{(j)} - 2p_i^{(j)} + p_{i-1}^{(j)}].$$

With a Gaussian pulse in the domain center as initial conditions:

$$p^{(0)}(x) = p^{(-1)}(x) = \exp(-0.01x^2),$$

Hints:

- Use $h = 1$ with 100 grid points and try different Δt, e.g. 0.1, 0.5, 0.999, 1.0, 1.001, 2. Let the program run for 300 time units.
- Use three data vectors: *p_now* for the current time step, *p_past* for the previous time step, *p_new* for the next time step,
- Use *diff(p, 2)* to evaulate the spatial derivative.
- If the Gaussian pulse works, try a shorter pulse and a box-shaped pulse.

Reduction to the advection equation

Elliptic and hyperbolic PDEs can be formally reduced to advection form. We demonstrate this for the wave equation:

$$\partial_t u = -\operatorname{div}\mathbf{j} = -\partial_x j_x - \partial_y j_y - \partial_z j_z, \qquad \partial_t^2 u - \Delta u = 0.$$

The Laplacian is related to the divergence: $\Delta = \operatorname{div}\operatorname{grad}$.

We reformulate:

$$\begin{aligned} \partial_t(\partial_t u) =& \operatorname{div}\operatorname{grad} u = \operatorname{div}\mathbf{j}, \\ \partial_t \mathbf{j} =& \partial_t \operatorname{grad} u = \operatorname{grad}(\partial_t u). \end{aligned}$$

By introducing a vector-valued state vector $\mathbf{U} = [(\partial_t u), j_x, j_y, j_z]^T$, this has the form of the advection equation:

$$\partial_t \begin{pmatrix} (\partial_t u) \\ j_x \\ j_y \\ j_z \end{pmatrix} + \partial_x \begin{pmatrix} j_x \\ (\partial_t u) \\ 0 \\ 0 \end{pmatrix} + \partial_y \begin{pmatrix} j_y \\ 0 \\ (\partial_t u) \\ 0 \end{pmatrix} + \partial_z \begin{pmatrix} j_z \\ 0 \\ 0 \\ (\partial_t u) \end{pmatrix} = 0.$$

How does that look in 1d?

In 1d, this becomes:

$$\partial_t \begin{pmatrix} v \\ w \end{pmatrix} = \partial_x \begin{pmatrix} w \\ v \end{pmatrix} \qquad \text{with} \qquad v = \partial_t u, \quad w = \partial_x u.$$

This is very reminiscent of the harmonic oscillator examples, where leapfrog integration was a good idea:

$$\partial_t \begin{pmatrix} x \\ p \end{pmatrix} = \begin{pmatrix} p \\ -x \end{pmatrix}.$$

Can we do something similar and thereby simplify the finite difference method?

How do we deal with the asymmetry of the first derivatives?

Staggered grids: Classic FDTD method

The solution is to let v and w live on *different* grids!
The grids are offset by half a grid constant h, their length differs by one.

We write this by introducing half-integer grid indices:

$$\begin{aligned}\partial_x v_{i+1/2}^{(j)} \approx& h^{-1}[v_{i+1}^{(j)} - v_i^{(j)}],\\ \partial_x w_i^{(j+1/2)} \approx& h^{-1}[w_{i+1/2}^{(j+1/2)} - w_{i-1/2}^{(j+1/2)}],\end{aligned}$$

This is just notation, in the program, both arrays are addressed via integer indices, of course.

Together with a leapfrog time integrator, we get the classic finite-difference time-domain (FDTD) method:

$$\begin{aligned}v_i^{(j+1)} =& v_i^{(j)} + h^{-1}\Delta t[w_{i+1/2}^{(j+1/2)} - w_{i-1/2}^{(j+1/2)}],\\ w_{i+3/2}^{(j+1)} =& w_{i+1/2}^{(j)} + h^{-1}\Delta t[v_{i+1}^{(j+1)} - v_i^{(j+1)}],\end{aligned}$$

Example 2: FDTD

Implement the FDTD method with staggered grids in MATLAB:

$$
\begin{aligned}
v_i^{(j+1)} =& v_i^{(j)} + h^{-1}\Delta t[w_{i+1/2}^{(j+1/2)} - w_{i-1/2}^{(j+1/2)}],\\
w_{i+1/2}^{(j+3/2)} =& w_{i+1/2}^{(j+1/2)} + h^{-1}\Delta t[v_{i+1}^{(j+1)} - v_i^{(j+1)}],
\end{aligned}
$$

With a Gaussian pulse in the domain center as initial conditions:

$$
v^{(0)}(x) = \exp(-0.01x^2), \qquad w^{(0)}(x) = 0.
$$

Hints:

- Use only one array for v and one array (shorter by one grid point) for w.
- Treat the half-integer offsets as only notation candy, i.e. write $w(ii)$ for $w_{i+1/2}$.
- Use *diff(p)* to evaulate the spatial derivative.
- As before, try shorter and boxier pulses.

Homework

The problem on the previous slide

Literature:
A. Taflove, S.C. Hagness,
"Computational Electrodynamics – The Finite-Difference Time-Domain Method"
Artech House Publishing