Fluid Dynamics — Numerical Techniques

MATH5453M Numerical Exercises 1, example

Keywords: wave equation, explicit yet mixed forward/backward Euler aka symplectic time discretization, central differences for "diffusion" term, numerical stability, verification/comparison "exact" and numerical solutions, errors, θ–scheme.

Sources: Chapter 2 of Morton and Mayers (2005/2008), Internet.

1. First, consider the *forced integro-differential diffusion equation* on $x \in [L_p, L]$ with $L > L_p$ for the unknown $\lambda(x,t)$:

$$\partial_t \lambda = \partial_{xx}\lambda - \int_{L_p}^{L} \lambda(\tilde{x},t)\,\mathrm{d}\tilde{x} - f(x) \tag{1a}$$

$$f(x) = 2 + \frac{2}{3}(L - L_p)^3 \tag{1b}$$

$$\lambda(L_p,t) = 0 \quad \text{and} \quad \partial_x\lambda|_{x=L} = 0. \tag{1c}$$

   Show that $\lambda(x) = (x - L_p)(x - 2L + L_p)$ is a steady-state solution (or fix matters). Using an explicit solver, a second-order discretization in space for the diffusion, the trapezium rule for the evaluation of the integral, and an initial condition not equal to the steady-state solution, demonstrate and visualise how the solution evolves to this steady state. E.g., use $\lambda(x,0) = (x - L_p)^2(x - 2L + L_p)^2$, i.e., it may be wise to satisfy the boundary conditions at $t = 0$. At what time is the solution in steady state and how does one deal with this challenge numerically? Show the evolution to this steady state in a clear manner in a suitable plot. If you are adventurous, also implement and test the θ–scheme for $\theta = 0, 1/4, 1/2, 3/4, 1$, say (open exploration). Take $L_p = 2, L = 4$; plot at 11 equidistant time intervals till an appropriate final time.

2. Second, consider the *linearised shallow water equations* in one horizontal dimension with spatial coordinate x and time t:

$$\partial_t \eta + \partial_x (Hu) = 0 \tag{2a}$$

$$\partial_t u + \partial_x (g\eta) = 0 \tag{2b}$$

with the free surface height $b(x) + H(x) + \eta(x,t) = H_0 + \eta(x,t)$, water depth at rest $H(x)$, bottom topography at vertical position $z = b(x)$, rest level at $z = H_0 = \text{constant}$, free-surface deviation from this rest level $\eta(x,t)$ and depth-averaged velocity $u(x,t)$. The acceleration of gravity $g \approx 9.81\text{m/s}^2$ is constant and the (varying) rest water depth $H(x)$ is a given function of x. The velocity u and free-surface deviation η as function of x and t are the unknown fields. Consider a domain $x \in [0, L]$ with $L > 0$ and with either solid-wall boundary conditions $u(0,t) = u(L,t) = 0$, periodic boundary conditions, or "open" in- and outflux boundary conditions. Initial conditions are $u(x,0) = u_0(x)$, $\eta(x,0) = \eta_0(x)$.

a) Demonstrate step-by-step that the system (2) can be rewitten as

$$\partial_{tt} \eta - \partial_x (gH \partial_x \eta) = 0. \tag{3}$$

b) Derive the (Neumann) boundary conditions for η at $x = 0, L$, concerning the solid-wall boundary conditions with $u(0,t) = u(L,t) = 0$ and also for periodic boundary conditions.

c) What should the initial conditions for η be; how many do we need and why?

d) Show that the wave equation (3) with varying medium $H(x)$, here the water depth at rest, is a hyperbolic equation.

e) Show that this wave equation is dimensionally correct as a check, using the dimensions of all variables, functions, constants and coordinates involved.

3. Discretise the wave equation (3) for the solid-wall boundary conditions using an adjoint, compact discretisation of the spatial derivatives (cf. the corresponding sections in the lecture notes and Morton and Mayers (2005/2008)). Use grid points at $\frac{1}{2}\Delta x, \frac{3}{2}\Delta x, \frac{5}{2}\Delta x, \ldots, L - \Delta x/2$ with $\Delta x = L/N_x$ in order to avoid evaluation of $H(x)$ at ghost points where $H(x)$ is unavailable. Either use a central difference discretisation in time, also addressing what to do at the initial time, or use an aide variable $p \equiv \partial_t \eta$ to derive a system of two equations whilst time stepping η with Euler forward in time and p with Euler backward in time (in the latter step one can use the update to the next time level determined in the first step), thus yielding a symplectic Euler overall time discretisation. In the latter case, one still needs to address what to impose at the initial time.

4. For the special case with constant water depth $H(x) = H_0$ such that $c^2 = gH = c_0^2 = gH_0$, derive the following exact solutions for the wave equation, either constructively or by checking the solutions via substitution in the wave equation,

$$\eta(x,t) = (A_m \cos(\omega t) + B_m \sin(\omega t)) \cos(m\pi x/L) \quad \text{and} \quad \omega = c_0 k \equiv c_0 m\pi/L \qquad (4)$$

with $m = 0, 1, 2, \ldots$ a positive integer, frequency ω as well as amplitudes A_m and B_m. This exact solution will be used as test or verfication solution of the numerical scheme and its implementation. Here, c_0 is a wave speed and $k = m\pi/L$ a quantized wave number (check this dimensionally). These are standing waves. Since the wave equation is linear one can use linear combinations of these solutions for different values of m to fit any initial condition. For testing purposes, one can take convenient combinations for a limited number of integer values of m.

5. For $B_m = 0$, $A_m = 1$, $L = 6$ and $m = 4$, say, plot the numerical and exact solutions at $t = (3, 3.25, 3.5, 3.75, 4)\, T$ with T the period of your solution, which will depend on the frequency ω. Check the periodicity in your numerical solution with the calculable

period. Also explore other cases, i.e., what is the difference when you look at the solution at $t = (9, 9.25, 3.5, 9.75, 10)\,T$ for the same resolution?

6. Demonstrate numerically that your numerical solution is second-order accurate in space by using the exact solution at $t = 4T$ and/or $t = 10T$, by calculating the L_2–error and L^∞–error for different resolutions in a table of convergence. Caculate the error but also the rate of convergence; what should this rate be? How should the error behave when you double the spatial resolution, assuming and thus taking care that the temporal resolution is not a bottleneck?

References

- K.W. Morton and D.F. Mayers 2005/2008: Numerical solution of partial differential equations. Cambridge University Press, 278 pp.
- Internet, e.g., for table of convergence, L^∞–error, et cetera.

Onno Bokhove, October 2025

A How to write a computer program

Writing a computer program, e.g., in Matlab, Python or any other programming language, consists typically of the following steps. We will use writing a code for the explicit discretization of the diffusion equation as example.

These steps are:

- Derive and write out the mathematics of the complete numerical algorithm, e.g., including discrete boundary and initial conditions, the mesh and its numbering, the equation updates for each unknown, using index notation or loops.

- Having completed this mathematical part, write a pseudo-code in words and/or block format; these blocks can in principle also become subroutines or classes or functions as this will improve the organization of the final program:

  - Introduce comments, starting with the title and goal of the program;

  - Define all parameters and variables required, clearly distinguishing these two, e.g.: $L, dx, Nx, Nt, dt, Tend, tmeasure, dtmeasure, mu = fac * dt/(dx * dx), tijd$ (domain length, spatial step, number of grid cells, time step, final time, measurement time and increment, factor fac for stability, time variable $tijd$), et cetera, and the arrays $u = u[j], unew[j]$ for $j = 1, 2, 3, \ldots, Nx + 1$ et cetera;

  - Initialize some variables and plot the initial condition.

  - Create a for or while loop over time.

  - Create a for or while loop over space, or use the vector form in matlab, e.g.: $unew[2 : Nx] = u[2 : Nx] + mu * (u[1 : Nx - 1] - 2 * u[2 : Nx] + u[3 : Nx + 1])$

  - Within these loops arange to plot at appropriate times.

  Note that it is not a good idea to store U_j^n as a matrix array as it will lead to an overflow of data. So do not store all the space–time data.

- The next step is to write a piece of code along these lines:

  - Test your code after every small stage/block by running it; deal with error messages, e.g., explicitly check the range of your arrays, et cetera;

  - Initially plot after every time step, plot the new updates of the variables, and just run your code for a few time steps to see whether matters behave appropriately or blow up;

- Verify your code against exact solutions, other people's solutions, solutions using a different method/program (as in our exercise), graphs in books, and/or high-resolution runs.

B Remarks and/or common mistakes

A table of convergence can be used to determine and/or confirm the formal order of convergence of your numerical scheme by simulations. Consider the case with a variable $u(x,t)$ and initial condition $u_0(x,0)$, and simulate to a final time $t = T_e$ with a fixed spatial grid of size Δx and (fixed) time step Δt. The easiest thing to do is to pick a time $t_c \leq T_e$ at which the solution has sufficient spatial structure when one checks the convergence in space and still has sufficient temporal structure when (also) investigating the convergence in time. Denote the numerical solution at this time t_c by U_j or $U_j^{n_c}$ and the exact or a very high resolution solution by $u(x,t_c)$. Here $x_j = (j-1)\Delta x$ is the coarse grid position one starts with $j = 1$. Denote the fine grid positions by $x_{j'} = (j'-1)\Delta x'$, i.e., we used two different meshes with two different mesh sizes.

Remarks:

- Choosing a final decayed state with $u(x, t_c = T_e) \approx 0$ is therefore not appropriate.
- Define the norms you are using clearly and in mathematical terms, not as some internal function in some programming language. That is the L^∞–error is:

$$L_\infty = max_j(|U_j^{n_c} - u(x_j, t_c)|), \tag{5}$$

which is trivial to compute by looping over all grid points. In case you only have a refined solution, either take care that the fine grid also includes the course grid position x_j or interpolate the solution between the appropriate positions $x_{j'} \leq x_j \leq x_{j'+1}$ (noting that j and j' concern indices on different grids). Make a "continuum" numerical

solution denoted by $U(x,t)$ (e.g., by interpolation). The L_2–error is, using indexing $j = 1, \ldots, N_x + 1$ and a numerical integral approximation for the integral:

$$\begin{aligned} L_2 =& \sqrt{\frac{1}{L}\int_0^L (U(x,t_c) - u(x,t_c))^2 \mathrm{d}x} \\ \approx& \frac{1}{\sqrt{L}} \sqrt{\big(\frac{1}{2}(U_0 - u(0,t_c))^2 + \frac{1}{2}(U_{N_x} - u(L,t_c))^2 + \sum_{j=2}^{N} (U_j^{n_c} - u(x_j,t_c))^2\big)\Delta x}. \end{aligned} \quad (6)$$

Even a lousy integral approximation tends to suffice to demonstrate convergence. i.e., it is usually not worth spending time on accurate integration routines to determine these norms.

- When a numerical method is second order, the error scales like $(\Delta x)^2$, i.e.

$$Error = C(\Delta x)^2 \quad (7)$$

for some constant C and for some norm. So when the resolution is doubled, the error becomes $Error = C(\Delta x)^2/4$ et cetera. *This means that the error should in principle go down by a factor of four. So for Δx smaller and smaller this convergence should be obeyed and observed by you.* If this is not the case something is wrong: your time step was not sufficiently small and one is measuring the temporal error instead, there is still a bug, the measurement time t_c is inappropriate, or the boundary conditions are implemented with another order, or something else to be determined by you.

- It is therefore easiest to use regular refinements, e.g., with mesh sizes of

$$\Delta x, \Delta x/2, \Delta x/4, \Delta x/8;$$

but do take care that when a high resolution solution is used as pseudo-exact solution, its mesh size is sufficiently smaller than the smallest mesh size of the numerical test solutions.

- When the order of the scheme is unknown one can assume $Error = C(\Delta x)^\gamma$ at first and determine the exponent γ, using logarithmic plots.

- Graphs should have labels of appropriate size and should be readable.

- As requested and required for your own sake, the mathematics of the algorithm should be worked out on paper first in detail. Code is not acceptable as explanation of the algorithm. The indexing of the grid should be clearly indicated. The treatment and adaptation of the main algorithm at the boundary points should be clearly explained in terms of mathematical formulas.

- There are alternative ways of considering the convergence without having an exact or fine numerical solution. Assume one has the numerical solutions at four resolutions $h = \Delta x, \Delta x/2, \Delta x/4, \Delta x/8$. Denote these by $u_h, u_{h/2}, u_{h/4}, u_{h/8}$. Assume an error solution Ansatz of the form $C(\Delta x)^2$ Consider the ratio

$$\frac{h^2 - (h/2)^2}{(h/4)^2 - (h/8)^2} = \frac{(||u_{h/8} - u_h|| - ||u_{h/8} - u_{h/2}||)}{(||u_{h/8} - u_{h/2}|| - ||u_{h/8} - u_{h/4}||)} \tag{8}$$

  for some norm $||\cdot||$ and analyse this using the error Ansatz. This is called Richardson's extrapolation. Do the same for an error Ansatz with unknown exponent γ.

- Something similar can be done to verify convergence in time, using refinements for of the time step Δt.

- It is advisable to make a routine such that one can plot the solutions at desired times specified before the time loop.

- Mention your name in the (Matlab) codes you submit. Also submit the codes used to calculate the tables of convergence and the plotting. These should be part of your programs.