Fluid Dynamics — Numerical Techniques

MATH5453M Numerical Exercises 1, 2024

Due date: Friday October 18th

Keywords: diffusion equation, explicit forward-Euler time discretization, central differences and diffusion, upwind discretization, numerical stability, maximum principle, convergence and errors, verification/comparison numerical solutions, θ–scheme, mandatory clear run-time instuctions (reproducibility). Sources: Chapter 2 of Morton and Mayers (2005, M&M), Internet.

Consider the non-dimensional linear advection-diffusion equation for the variable/unknown $u = u(x,t)$, with an initial condition and boundary conditions:

$$u_t - a(t)u_x - \epsilon u_{xx} = 0 \quad \text{for} \quad x \in [L_p, L] \tag{1a}$$

$$u(x,0) = u_0(x) \tag{1b}$$

$$u(L_p,t) = u(L,t) = 0, \tag{1c}$$

(small) constant diffusion ϵ and a given function $a(t)$. The boundary conditions are classical homogeneous Dirichlet conditions. The above system arose from the research on machine learning of Choi et al. (2022). In the end, we will use $L_p = -1$ and $a(t) = 1$.

1. Why is (1) a linear advection-diffusion equation. Explain all three terms used, i.e. "linear" (precisely), "advection" and "diffusion".

2. For the θ-method, prove equations (2.80) to (2.84) in M&M.

3. Write down the θ-discretisation for (1). Use a spatial (first-order) upwind discretisation for the advection term, which means that the sign of the wind $-a(t)$ determines

the two-point stencil to be used. First clearly define your space-time mesh and its indexing (number according to Python-programming convention). Explicitly deal with and explain the scheme for mesh points near the boundary. That is provide complete theoretical detail. Specify a final formulation with the unknowns on the LHS and the knowns (as vector) on the RHS.

4. Implement the explicit scheme (without matrix inversion) and reproduce Fig. 2.2 of M&M. Use the vector set-up of Python in the code. Make similar plots as in Fig. 2.2 of M&M, but display multiple time-profiles in one plot for fixed Δt. Explain the stability properties of the explicit scheme with both the results of a Fourier analysis and the maximum principle. Extend the explicit scheme to (1) above.

5. Implement the θ–scheme and use the linear algebra routines in Python to solve the matrix system, for varying spatial resolutions. Explain the stability properties of the θ–scheme with both the results of a Fourier analysis and the maximum principle for the advection and diffusion parts in separation (advanced: when you can explore the full case, e.g. by numerically displaying the expressions). Oberve that the amplitude error is zero for $CFL = 1$ when $\epsilon = 0$.

6. First explore the case $a(t) = 1$ with

$$u(x,0) = (1-x)^4(1+x) \quad \text{and then} \quad u(x,0) = (1-x)^4(1+x)(\sum_{k=0}^{3} b_k\phi_k(x) + C) \quad (2)$$

for $L_p = -1, L = 1, \epsilon = 10^{-3}, T = 1$ with $t \in [0,T]$, Legendre polynomials $\phi_k(x) = 1, x, \frac{3}{2}x^2 - \frac{1}{2}, \frac{5}{2}x^3 - \frac{3}{2}x$ and random coefficients $b_k \in [0,1)$ (uniform distribution and for $k = 0, \ldots 3$). A constant $C \geq 0$ is determined (numerically) such that $\sum_{k=0}^{3} b_k\phi_k(x) + C \geq 0$. Verify that for the appropriate θ the same numerical results are obtained as for the implementation of an explicit scheme. Display some time profiles at set times rather than set iterations for a few values of θ, μ. Use a while-time-loop with discrete time, not one using iterations. Clearly define how the stability/instability and the (violation of the) maximum principle are (numerically) determined and displayed in a μ–θ parameter

plot. Find at least three μ, θ-values illustrating the three possible cases $\theta = 0, 1/2, 1$. Interpret and discuss your findings. In then end, choose $\theta = 1$, where $CFL = 1$ works, and argue why this choice seems necessary (Choi et al., 2024). (Report your b_k's, one of my draws was $b_k = [0.22329108, 0.52316334, 0.55070146, 0.04560195, 0.36072884]$.)

7. Second, consider the case $a(t) = 1$ and initial condition (see Choi et al. 2024)

$$u(x, 0) = (1 - x)^4(1 + x)(\sum_{k=0}^{3} b_k \phi_k(x) + C) \tag{3}$$

for $L_p = -1, L = 1, \epsilon = 10^{-3}, T = 1$ with $t \in [0, T]$. Graphically confirm convergence of your simulations by refining Δx. Display some time profiles at set times (e.g., $t = 0, 0.25, 0.5, 0.75, 1$ in one plot and also zoom-in near $x = -1$) rather than set iterations for $\theta = 1$. Interpret and discuss your findings. (Report your b_k's.)

8. Third, explore the previous case for smaller values of $\epsilon, 10^{-4}, \ldots, 10^{-6}$. Report, interpret and discuss your findings (see Choi et al., 2024).

9. All results need to be reproducible, meaning that clear run-time instructions need to be given on GitHub, in an easy manner specified as choices in your codes. Please submit a clear report on GitHub with output figures saved, folded into the answers, with clear captions and explanations. Make a bespoke GitHub folder. It is not acceptable to report no output and just hand-in the codes. Reports can be handwritten with named figures as computer output. It needs to be clearly explained on GitHub and in the code how to run the codes and reproduce the figures, with what resolution, not only as emerging plots in Python but also with the graphs as named output files. Graphs should have clear labels and (hand-written) captions, etc.

References

- J. Choi, T. Yun, N. Kim, Y. Hong 2024: Spectral operator learning for parametric PDEs without data reliance *CMAME*, **420**. `https://doi.org/10.1016/j.cma.2023.116678`

- K.W. Morton and D.F. Mayers 2005: Numerical solution of partial differential equations. Cambridge University Press, 278 pp.

- Internet.

Onno Bokhove, October 2024.

A How to write a computer program

Writing a computer program, e.g., in 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.

Hence, 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 $j = 0, 1, 2, 3, \ldots, Nx$, 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.:
  $unew[1:Nx-1] = u[1:Nx-1]+mu*(u[0:Nx-2]-2*u[1:Nx-1]+u[2:Nx])$
- 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 $t = T$ with a fixed spatial grid of size Δx and (fixed) time step Δt. The easiest thing to do is to pick a time $t_c < T$ 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 $u(x, t_c = T) \approx 0$ 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:

$$L_\infty = max_j(|U_j^{n_c} - u(x_j, t_c)|), \tag{4}$$

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:

$$\begin{aligned} L_2 =& \sqrt{\frac{1}{L}\int_0^L (U(x, t_c) - u(x, t_c))^2 \mathrm{d}x} \\ \approx& \frac{1}{\sqrt{L}} \sqrt{\big(\frac{1}{2}(U_0 - u(0, t_c))^2 + \frac{1}{2}(U_{N_x} - u(L, t_c))^2 + \sum_{j=2}^{N} (U_j^{n_c} - u(x_j, t_c))^2\big)\Delta x}. \end{aligned} \tag{5}$$

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.

$$Error = C(\Delta x)^2 \tag{6}$$

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

$$\frac{h^2 - (h/2)^2}{(h/4)^2 - (h/8)^2} = \frac{(||u_{h/8} - u_h|| - ||u_{h/8} - u_{h/2}||)}{(||u_{h/8} - u_{h/2}|| - ||u_{h/8} - u_{h/4}||)} \tag{7}$$

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/on the Python codes/write-up you submit. Also submit the codes used to calculate the tables of convergence and the plotting. These should be part of your programs.