MA/CS 615. Numerical PDE: Assignment II

February 18, 2026

1 Temporal discretization of KdV

In this problem, consider to numerically solve the Korteweg de Vries (KdV) PDE

$$\partial_t u = \mathcal{K}\left[u\left(\cdot,t\right)\right] = -\partial_{xxx}u - 3\partial_x\left(u^2\right),$$

where $\mathcal{K}\left[u\left(\cdot,t\right)\right]$ denotes the functional on the right hand side, which only involves derivatives of x. Recall from the previous assignment, the KdV has traveling wave solutions in the forms of "solitons" waves

$$u_{\text{sol}}\left(x,t\right) = \frac{c}{2}\text{sech}^2\left(\frac{\sqrt{c}}{2}\left(x-ct\right)\right), \tag{1}$$

where c is the speed of the soliton moving to the right. While the soliton solution (1) is not periodic, it decays exponentially for large x so we can pretend it is periodic. Here we will consider the KdV equation on a periodic domain $x \in [-L/2, L/2]$ with $L = 60$.

We consider pseudo-spectral methods for solving this PDE, in which we represent the solution as a (truncated) Fourier series [*Recall k is not an integer here.*]

$$u\left(x,t\right) = \sum_k \hat{u}_k\left(t\right)e^{ikx}.$$

The coefficients $\hat{\mathbf{u}}\left(t\right)$ are solutions of the system of ODEs

$$\begin{aligned}\frac{d\hat{\mathbf{u}}}{dt} = \mathbf{F}\left(\hat{u}\right) &= ik^3 \odot \hat{\mathbf{u}} - 3ik \odot \mathcal{F}\left(\left(\mathcal{F}^{-1}\hat{\mathbf{u}}\right)^{\odot 2}\right)\\ &= A\hat{\mathbf{u}} + B\left(\hat{\mathbf{u}}\right)\end{aligned}$$

where $\mathcal{F}$ denotes a Fourier transform. Here the first term $A\hat{\mathbf{u}} \equiv ik^3 \odot \hat{\mathbf{u}}$ corresponding to $-\partial_{xxx}u$ is *stiff* but linear, and the second term $B\left(\hat{\mathbf{u}}\right)$ corresponding to $-3\partial_x\left(u^2\right)$ is a complicated nonlinear function (potentially involving de-aliasing features as well). Note that the mode with wavenumber $k = 0$ does not evolve with time (this is because the KdV equation is a conservation law) and it should be omitted from the code.

In this homework you will consider two different method to solve the system of ODEs

$$\frac{d\hat{\mathbf{u}}}{dt} = A\hat{\mathbf{u}} + B\left(\hat{\mathbf{u}}\right) \tag{2}$$

one an implicit-explicit multistep method, and the other an exponential Runge-Kutta integrator. Both schemes are second order accurate and handle the linear term implicitly, but treat the nonlinear term explicitly in order to avoid solving nonlinear systems of equations. If it turns out that the nonlinear term $B\left(\hat{\mathbf{u}}\right)$ does not cause stiffness, then we have a hope of having a good method to solve the KdV equation.

Note: It is preferable if you treat the unmatched mode for even number of grid points that are powers of two carefully as explained in class. If you cannot then simply *use odd grid sizes that are powers of three.* Another often-used though imperfect alternative is to zero out that mode when computing odd derivatives;

this however introduces a artificial conserved variable (the unmatched mode) so it can be dangerous if not used with some care (think about it).

You do not have to do de-aliasing for this problem, but it may improve things if you do it. A good way to test things is to employ a test function that is a sum of a small number of Fourier modes instead of a complicated nonlinear function like the soliton. Use only a small number of points (the minimal required, so that the highest Fourier mode is the unmatched one!) so that in principle the discretization should be *exact*. For the linear part of the PDE, you can write down an exact solution without ODE solvers! For the nonlinear part, just evaluate the nonlinear term in the r.h.s. for the test function and see if you can get it *exactly* correct. Doing things like handling the special mode or de-aliasing does not guarantee that you will get a more accurate answer for the soliton, but it is a way to develop a modular code where pieces are tested and correct and the solver is as accurate as possible for very smooth solutions such as a sum of a (small) finite number of Fourier components (i.e., for band-limited functions).

1.1 Absolute stability of two methods

The first ODE integration method to use is the **SBDF2** schemes. For an ODE of the form (2) a time step of duration Δt takes the form

$$\hat{\mathbf{u}}^{n+1} = \frac{4}{3}\hat{\mathbf{u}}^n - \frac{1}{3}\hat{\mathbf{u}}^{n-1} + \frac{2}{3}\Delta t A\hat{\mathbf{u}}^{n+1} + \frac{2}{3}\Delta t\left(2B\left(\hat{\mathbf{u}}^n\right) - B\left(\hat{\mathbf{u}}^{n-1}\right)\right),$$

see also Eq. (35) in the paper of Cox and Matthews (CM) where it is called the AB2BDF2 scheme.

The second scheme is the exponential time differencing RK2 method. A midpoint variant but more efficient (think about it) is to use a trapezoidal explicit RK2 formula, leading to the **ETDRK2** scheme

$$\begin{aligned}
\text{predictor}: \quad & \hat{\mathbf{u}}^{n+1,*} = e^{A\Delta t}\hat{\mathbf{u}}^n + A^{-1}\left(e^{A\Delta t} - I\right)B\left(\hat{\mathbf{u}}^n\right), \\
\text{corrector}: \quad & \hat{\mathbf{u}}^{n+1} = \hat{\mathbf{u}}^{n+1,*} + A^{-2}\left(\frac{e^{A\Delta t} - I - A\Delta t}{\Delta t}\right)\left(B\left(\hat{\mathbf{u}}^{n+1,*}\right) - B\left(\hat{\mathbf{u}}^n\right)\right),
\end{aligned}$$

see also Eqs. (20+22) in the paper by CM (hopefully now you see why you were advised to drop the $k = 0$ mode). Note that direct implementation of these formulas can suffer from roundoff errors for small Δt, more precisely, for small $|\lambda_{\min}|\,\Delta t$ where $\lambda_{\min}$ is the eigenvalue of A with smallest magnitude; make sure this is not the case or find a way to avoid catastrophic cancellation (e.g., by using Taylor series). *Make sure to write the code in a way that avoids evaluating things more than once.* For example, evaluate $B\left(\hat{\mathbf{u}}^n\right)$ only once per time step and store and reuse between the predictor and the corrector. In fact, observe that since A is a constant many things can be computed once and only once at the beginning (and that computation is super cheap if done right).

Explain why the term $A\hat{\mathbf{u}}$ is stiff when there are lots of Fourier modes. Then explain whether you think SBDF2 and ETDRK2 are good choices to solve (2) and why, focusing on the linear term $A\hat{\mathbf{u}}$ since the nonlinear part is complex to analyze (you will test things out numerically in what follows) and is treated similarly in the two methods .

1.2 Accuracy, Stability and Robustness for a Single Soliton

Take as initial condition the soliton $u\left(x, t = 0\right) = u_{\text{sol}}\left(x, 0\right)$, and then solve the KdV equation to time $T = L/c$ – the solution should be unchanged since it has traveled around the periodic domain once, i.e., $u\left(x, t = T\right) = u_{\text{sol}}x, 0$. Recall that even though the soliton wave is not a periodic solution it decays sufficiently rapidly that it will be a solution to very high accuracy even in a periodic domain. Implement both the ETDRK2 and AB2BDF2 schemes to integrate the KdV equation in time, and do each of the next tasks for each of the schemes.

1.2.1 Empirical stability

Try several grid sizes (number of Fourier modes) that are powers of two (or powers of three), say from 32 up to 256 (remember that in Assignment I that you need to keep ~ 200 Fourier modes in order to evaluate $F\left(\hat{\mathbf{u}}\right)$

to nine accurate digits for the soliton wave with $c = 1$). For each resolution, determine empirically by playing around (i.e., simply try increasing the time step size until you get unstable behavior of the solver; watching a movie of the solution can be very helpful) whether there is a stability limit on the time step size Δt, and if so, what that limit is (approximately). How does the stability limit depend on the resolution (number of grid points, or, equivalently, the number of modes)? Note that you may need to run over multiple (say 10) periods to see the instability; a method that appears stable over one period can be unstable over long times.

1.2.2 Accuracy of ODE solver

For this part choose (wisely) a certain number of points/modes to discretize the PDE in space. Solve the ODE (2) using the two schemes up to time T. Define the error from the temporal (ODE) integration

$$e(\Delta t) = \left\| \hat{\mathbf{u}}_{\Delta t}(T) - \hat{\mathbf{u}}_{\Delta t/2}(T) \right\|_2$$

to estimate the error. [*Hint: If you choose* $\Delta t \sim 2^{-k}$ *then you can compute the above ratio without doing two solves for each* Δt.] How small does Δt have to be for you to see "clear" second-order convergence? If Δt has to be too small for you to be able to perform the computation comfortably with the computing resources you have, feel free to evolve the solution over a shorter time, say one quarter of the period T.

In addition, it is much more informative to look not just at the error at the final time but to plot the error function $e(t; \Delta t) = \left\| \hat{\mathbf{u}}_{\Delta t}(t) - \hat{\mathbf{u}}_{\Delta t/2}(t) \right\|_2$. Plot this error for several values of Δt and comment on your observations, in particular, are you confident your time step sizes are in the asymptotic convergence regime?

Note: This way of testing order of convergence is sometimes called the *method of successive refinements*. Note that this does not test that the ODE solver converges to the correct solution, it simply tests that the numerical solution converges to something (potentially wrong)! In practice, one would first test the ODE solver on a simpler system of ODEs for which maybe an analytical solution can be constructed, or use something called the *method of manufactured solutions*.

1.2.3 Accuracy of PDE solver

For this next part use ~ 256 modes. This ensures that the error is dominated by the error from integrating the ODEs, that is, the temporal discretization error is much larger than the spatial discretization error. It also gives us hope (but certainly no guarantees!) that if we accurately solve the system of ODEs we will also have accurately solved the PDE.

Confirm whether the (functional) error in $u(x, T)$ is $O\left(\Delta t^2\right)$ by comparing to the exact solution of the PDE. Here you can (should) try different function norms, but also remember that plotting the error as a function is much more informative than looking at three numbers (norms). Compare the errors from the two schemes (ETDRK2 and AB2BDF2) and conclude which one is more accurate for the same time step size (but note that in practice what matters is which one is more accurate for the same overall computational cost).

If Δt has to be too small for you to be able to perform the computation comfortably with the computing resources you have, feel free to evolve the solution over a shorter time, say one quarter of the period T.

1.2.4 Robustness of PDE solver

Now gradually reduce the number of modes/points, and for each resolution set the time step size to be one half (if using grid sizes that are powers of two) or one third (if using powers of three) of the empirical stability limit you found in part 1.2.1. Plot the numerical estimate for the solution $u(x, T)$ together with the correct solution. Combine multiple curves on one plot in some intelligent and readable way (use different line styles, colors, symbols, etc., and put legends and captions). Comment on what you observe, i.e., discuss how the ways in which the solver fails (do the numerical errors make the wave move faster/slower, do they make it spread or shrink, do they cause oscillations, etc.) when the problem is *under-resolved*; this now relates to the *robustness* of the method.

Note: One lesson we will try to learn in this class is that a more accurate method is not necessarily more robust (usually the opposite!). Is one of the two ODE integration schemes clearly more robust? This is a judgment call, there is no right or wrong answer, but I hope you think about it carefully.

1.3 Collisions between two solitons

Soliton waves have the spectacular property that they can pass through each other without changing their shape – this makes the useful for optical communication, for example. Try to construct a still figure that illustrates this – this is actually challenging and an animation may be more informative. Make a movie where you start with two solitons that are well-separated and move toward each other so that eventually the two solitons collide head on, for example, one has speed c and the other has speed $-c$, or, make a movie where one faster soliton of speed $2c$ passes another of speed c. Use what you learned from the previous parts of the homework to select the grid size and time step and method of temporal integration smartly. Do not use more computational resources than necessary, in fact, try to make a "nice movie" with as few points and as large a time step size as possible. Set the length of the domain L and the initial condition so that the solitons do not overlap initially. Report what you did and the reasoning behind your choices.

If you add another small nonlinearity to the r.h.s. of the equations, e.g., $\epsilon\partial_x\left(u^3\right)$, then solitons will interact with each other and scatter from one another. This would not only make cool movies but may strain the numerical method as well. Try it if time permits.

Note: You can find lots of movies/images online, for example, check out for example solitons/kdv_ solitons.

2 Newton's Law of Motion

In this problem, you are asked to solve a system of ODEs modeling the motion of a collection of N planets around a star. The masses of the planets will be denoted with m_i and their positions with $\mathbf{r}_i\left(t\right), i = 1, \cdots, N$. We will assume that the mass of the star m_0 is much larger than the masses of the planets, so we can assume that the star remains fixed at the origin, $\mathbf{r}_0\left(t\right) \equiv \mathbf{0}$. Here you can assume that all of the planetary orbits are co-planar, $\mathbf{r}_i = \left(x_i\left(t\right), y_i\left(t\right)\right) \in \mathbb{R}^2$, although it is recommended that you write your code so that the number of dimensions (2 or 3) can be an input parameter. That way your code can be used to study the impact of a comet coming out of the plane of the solar system without any changes.

Newton's laws of motion for the planets take the form of a system of *second order ODEs*,

$$\frac{d^2\mathbf{r}_i}{dt^2} = \mathbf{F}_i\left(\mathbf{r}_1, \cdots, \mathbf{r}_N\right) = \sum_{j=0, j\neq i}^{N} Gm_j \frac{\mathbf{r}_j - \mathbf{r}_i}{\left\|\mathbf{r}_j - \mathbf{r}_i\right\|_2^3}, \quad i = 1, \cdots, N.$$

For this problem you can set the value of the gravitational constant $G = 1$, although good programming practice suggests that you should keep it is a (perhaps global) variable in the code instead of hard-wiring the unit value.

You could write your own code to solve the ODEs using the Bogacki–Shampine embedded RK3 method (BS-RK23) used in the MATLAB ode23 integrator, following the description in Wikipedia page (and the lecture notes). This explicit one-step method is third order accurate, but also has an embedded second order estimator for adaptive error control. Put effort into writing a code that can integrate any first-order system of ODEs (like ode23) and not just the specific ODEs for gravitational systems. That is the rhs of the ODE should be input to the code, not hard-wired in the code. We usually write solvers assuming that the system of ODEs is first order, unless we are using some special integrator for 2nd order PDEs, so convert the system of ODEs to a first-order system first. [*If you are familiar with symplectic 2nd order integrators you are welcome to try them and compare to ode23*]

2.1 Single planet

Consider now the case $N = 1$, and denote the position of the single planet with $\mathbf{r}\left(t\right)$ with length $\left\|\mathbf{r}\right\|_2 = r$. It is well known since Kepler that the orbit of a planet around a massive star is an ellipse with the star being one if its foci, with the period of the orbit given by the formula

$$T = \frac{2\pi}{\omega} = \left(2\pi\right) a\sqrt{\frac{a}{Gm_0}},$$

where a is the major semi-axes of the ellipse, and ω is the angular frequency. The major semi-axes is given by

$$a = -\frac{Gm_0}{2E},$$

where the energy E (which is a constant of the motion) is the sum of the gravitational (potential) and kinetic energy:

$$E = -\frac{Gm_0}{r} + \frac{r^2}{2}.$$

Note that here we set the mass of the planet to unity because it cancels out in the end and only m_0 matters for a single planet. Assume that the initial position of the planet is $\mathbf{r}(0) = (r(0), 0)$ and set the initial velocity $\mathbf{v} = d\mathbf{r}/dt = (0, v(0))$, which ensures that the initial position is either the furthest or the closest point to the star on the orbit (i.e., it is a vertex of the ellipse).

2.1.1 Circular orbit

For a circular orbit, $\mathbf{r} = (r\cos\omega t, r\sin\omega t)$, the initial speed is $v(0) = \omega r(0)$, and the centripetal and gravitational forces are balanced,

$$r\omega^2 = \frac{Gm_0}{r^2}.$$

Use the BS-RK23 method to integrate a circular orbit over a period of revolution, using a fixed time step size $\Delta t = T/M$ where M is the number of time steps. While for simplicity in this part you should set $r = 1$ and $\omega = 1$, write your code so that these parameters can be changed easily. Observe that the only dimensionless number that matters is $\Delta t/T$ and not the value of Δt on its own, so that changing the values of the parameters does not require new simulations.

Confirm that the BS-RK23 is third-order accurate as a way to test your implementation. Find the largest possible time step size Δt that ensures that the maximum error in the x and y position of the planet is no larger than $10^{-2}r$, i.e., one percent relative error or two digits of accuracy (first solution). Then compute the orbit with time step $\Delta t/2$ (second solution). Use Richardson extrapolation to obtain a more accurate solution (third solution) as a function of time. For all three solutions (least accurate, more accurate, and most accurate), plot the error

$$e^k = \left\|\mathbf{R}^k - \mathbf{r}(k\Delta t)\right\|,$$

where $\mathbf{R}^k$ is the numerical approximation and $\mathbf{r}(t)$ is the true solution, as a function of time $t = k\Delta t$ using both a linear and a logarithmic scale for the vertical axes, and comment on your observations (indicate which norm you used). How many digits of accuracy did you get with Richardson extrapolation? (Observe that you could not predict/estimate this using error estimates so you are only able to answer this question because we know the exact solution of the ODEs).

2.1.2 Adaptive integration

Write an adaptive BS-RK3 integrator that ensures that each component of the position at time T is accurate to an absolute error of $10^{-2}r(0)$. Start the orbit with a velocity 1, 2, 4 and 8 times smaller than that required to get a circular orbit, so that you get increasingly more eccentric orbits. Integrate the orbit over one period T, and compute the error in the position after that one orbit (the planet should have returned to its initial position). Check that the adaptive time step control is doing something reasonable by plotting $x(t)$ for all points along the integration.

Confirm whether the adaptive error control worked and the target error was accomplished (recall that error control is likely to be pessimistic and reduce the time step size too much). Plot the time step size as a function of time (including rejected ones, as explained in class), and comment on what you observe and whether it makes sense to you or not. Try error tolerance of $10^{-3}r(0)$ and see what that does – does the error tolerance in the input directly correspond to the actual error you get?

2.2 Multiple planets

This part is for you to play and have fun but also learn how to make scientific animations (movies). Consider the case $N > 1$, say $N = 5$, called the N-body problem. Play around with your adaptive integrator (if your adaptive strategy failed to work, just use the built-in ode23) and see if you can construct an example where something interesting happens, and report what you came up with. Make a movie showing off your example. If you can produce a standalone movie file (say AVI or MPEG) that would be great – submit that movie.

Note that because the planets interact with each other via gravitational forces, in general one cannot say much analytically about systems with more than three bodies. However, if the planets are sufficiently far from each other the gravity of the star will dominate and each planet will orbit around the star in an elliptical orbit (but that sounds a bit boring!). You can read more on Wikipedia under Stability of the Solar System.

2.3 Long-time integration

Special classes of ODE solvers have been developed for integrating Newton's laws of motion. In particular, *symplectic methods* are supposed to be much better for integrating the equations over long periods of time. The simplest second-order symplectic method is the so-called Verlet integrator, based on a direct centered second order finite difference for $d^2\mathbf{r}/dt^2$,

$$\frac{\mathbf{r}_i^{k-1} - 2\mathbf{r}_i^k + \mathbf{r}_i^{k+1}}{2\Delta t^2} = \mathbf{F}_i^k = \mathbf{F}_i\left(\mathbf{r}_1^k, \cdots, \mathbf{r}_N^k\right).$$

While in practice this method is used as a one-step method with position and velocity as two independent variables, here we have written it as a multistep method for integrating directly the second-order equations.

Write a Verlet integrator and use it integrate a circular and a somewhat elliptical orbit over many periods (define "many" as appropriate based on what you observe), using the time step size you found in part 2.1.1 but also trying larger time step sizes to see how robust the integrator is for larger time step sizes. You can generate the two starting values for the multistep method using the exact solution. Plot the orbits and see if you see any qualitative features. Compare the accuracy you get for the circular orbit to what you get using the BS-RK23 method with the same time step size, and comment on your observations.