

Homework # 2

PHY-905-005

Computational Astrophysics and Astrostatistics
Spring 2023

This assignment is due by 11:59 p.m. on Friday March 24, 2023.

Instructions: Turn in all materials via GitHub. Create a separate subdirectory for each part of the homework, named `part_1`, `part_2`, etc., which should contain the code, plots, and answers to questions that are asked in that part (with sensible and descriptive file names for everything). Your code must adhere to the class coding and commenting standards, as documented in the file `class_coding_standards.txt` from Homework #1 (and please note that adhering to these standards will constitute a significant part of your homework grade). Plots should be named sensibly, should have easily readable and logical axis labels and titles, and the source code and data used to generate the plots should be included in the subdirectories. Questions for each section should be answered in a file named `ANSWERS.md` or in a PDF document of a similar name (e.g., `ANSWERS.pdf`).

Part 1: The [Lax-Wendroff method](#) for solving hyperbolic PDEs can be derived by Taylor expansion in a similar manner to the methods done in the books and in class. By keeping terms to $\mathcal{O}(\Delta t^2)$ and using centered, second-order spatial derivatives, the update equation for the linear advection equation ($u_t + au_x = 0$) becomes:

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{2\Delta x} a [u_{i+1}^n - u_{i-1}^n] + \frac{\Delta t^2}{2\Delta x^2} a^2 [u_{i+1}^n - 2u_i^n + u_{i-1}^n] \quad (1)$$

The first two terms on the right hand side are just the FTCS update (which was shown to be unstable in a pre-class assignment) and the second term is a diffusion term. This method is stable - the explicit diffusion term counteracts the numerical oscillation that made the FTCS method unstable.

Implement this method on a cell-centered finite-volume grid. For the initial conditions, choose a Gaussian:

$$u(x, t = 0) = e^{-(x-0.5)^2/0.1^2} \quad (2)$$

with a velocity of $a = 1$ in a domain $x = [0, 1]$ and using periodic boundary conditions (this is essentially the same initial condition and domain you used in a pre-class assignment). Evolve it for one period, $T = 1/a = 1$, and demonstrate that you have implemented it correctly by verifying that you get the same solution for velocities of $a = -1$ and $a = 1$ (and show this in a plot).

We are going to measure the convergence of this method as you change the resolution of the grid. Testing convergence requires that you change your timestep as you change your grid resolution, which is handled automatically if you use the same Courant number at each resolution. To get a single number from the spatially discretized solution that represents the numerical error, we're going to use the L2 norm (as defined in Section 1.2.4 of Zingale's lecture notes). We will define our error as:

$$\epsilon_{\Delta x} = \|a(x, t = T) - a(x, T = 0)\|_2 \quad (3)$$

In other words, this is the pointwise RMS error of the final solution compared to the initial solution, normalized by Δx (so, scaling the error with the number of grid points). **Plot $\epsilon_{\Delta x}$ vs. Δx for several power-of-two grid resolutions on a log-log plot and estimate the convergence rate from the slope of the line.** Note: if you don't see a convergence rate of $\mathcal{O}(\Delta t^2)$, you almost certainly have a bug somewhere! **Hint:** You have already done almost all of the coding in class - this problem should only require writing a few additional lines of code, and then measuring errors.

Part 2: In this problem, we’re going to be using an iterative relaxation method to calculate the gravitational potential of two 3D physical setups. The file `star_array_generator.py` contains code to generate three-dimensional numpy arrays of stars. The Jupyter notebook `star_generation_visualization_example.ipynb` provides an example of how to use this routine to generate two arrays, `single_star_array` and `many_star_array`, which contain a single $1 M_{\odot}$ star at the center of the grid and a large number of stars scattered randomly about the grid, respectively. The grid has a size that is determined by a keyword argument, and is assumed by default to be a cubic volume that is 1 parsec across. Do the following:

1. Write a function that solves Poisson’s equation, $\nabla^2\phi = 4\pi G\rho$, using your choice of either Jacobi or Gauss-Seidel iteration. This function should take as inputs a user-provided cubic 3D grid of densities of arbitrary grid size (in CGS units) and a desired level of error (to determine when convergence has been achieved), as well as any other parameters that you deem necessary. The function should return an array containing the gravitational potential in CGS units, the number of iterations required to achieve convergence, and the actual error (calculated as the L2 norm of the residual for the grid, excluding the ghost zones). Assume that the grid is cell-centered (finite volume) and use Neumann boundary conditions that ensure that the derivative of the slope across the boundary of the domain (i.e., between the ghost zone and the first “real” zone in your 3D array) is identical to the slope of the cells immediately inside the boundary. This is effectively an “isolated” boundary condition. How might you implement a periodic boundary condition in this circumstance?
2. Test the function that you have written using the `single_star_array`, which by default has 101^3 cells (103^3 including ghost zones). Plot a slice of the gravitational potential through the center of the array (i.e., at the same 2D slice of cells containing the star) and verify that you get a sensible answer – in other words, that the potential falls off as $1/r$ and that it has the correct units. You should do this by making a line plot that includes calculated values of the potential from the cell containing the star to the edge of the domain, as well as the analytic solution that you would expect. You may also want to make an image of a slice of the potential array (Hint: you may wish to take the log of the absolute magnitude of a slice of the array to be able to see dynamic range!) For the default resolution, how many iterations does it take for your solution to converge to an accuracy in the residual of 10^{-1} , 10^{-2} , and 10^{-3} , respectively? How does this change if you make the grid half the size in each dimension (51^3 cells, 53^3 with ghost zones)?
3. Now that you’ve verified that your code works correctly in a simple situation, calculate the potential for `many_star_array` at the default resolution. Plot a 2D image of the potential through several different slices of the domain (again using the log of the absolute value so you can see dynamic range). Does the result look sensible? Does the number of iterations required to achieve comparable accuracy change between this part of the problem and the single-star version?
4. Congratulations! You’ve written a (very slow) gravity solver. If you wanted to turn this into a simple simulation of the evolution of a cluster of stars, how might you do that? You don’t need to write the actual code, just outline the steps you’d take!

Part 3: In this problem we’re going to make a model of the conduction of heat through the Earth’s crust using a simple 1D model. This problem has a time-dependent boundary condition – the heating of the surface of the sun throughout the seasons.

The mean temperature at the Earth’s surface can be approximated as:

$$T_0(t) = A + B \sin\left(\frac{2\pi t}{\tau}\right) \quad (4)$$

where $\tau = 365$ days, $A = 10^\circ \text{ C}$ is the average surface temperature and $B = 12^\circ \text{ C}$ captures the seasonal variation.

At a depth of 20 m the temperature of the crust is relatively constant, and is set by the heat flux coming from the interior of the Earth. We assume it to be 11° C . The thermal diffusivity of the Earth’s crust is roughly $D = 0.1 \text{ m}^2 \text{ day}^{-1}$, and we assume it to be constant throughout the domain. As an initial condition, take the temperature to be $T = 10^\circ \text{ C}$ throughout the interior zones of the domain.

Solve the diffusion equation on a domain that goes from the surface to a depth of 20 m using the [Crank-Nicolson method](#) (as discussed in class and in Zingale’s notes). Use Dirichlet boundary conditions with the time-dependent values given above. Evolve the simulation for 10 years – after a few years, the initial guess at the temperature will have relaxed out (i.e., the initial values will have disappeared and been replaced by the “true” values). Experiment with the number of zones and the timestep to get a result that is converged and runs in a sensible amount of time.

For the final year, plot $T(x)$ as a function of depth at 4 different time periods that correspond to the four seasons. By how much does the temperature of the ground at 1 m, 3 m, 5 m, 10 m, and 15 m below ground vary throughout the year?