NZ Summer School on Gravitational Waves 2023

Lab 1

Getting started: To open the Jupyter environment on your browser, go to $130.216.216.114$: 9000. Your username and password is of the form <firstname>.<lastname initial>. I.e. Chris Stevens would be chris.s You will be met with a Jupyter environment. Open a terminal, type and execute

git clone https://github.com/nullicle/GW_Workshop.git

This will clone the necessary files to your working directory. Open Lab1/g_wave_setup.ipynb and enter the cells to run the colliding gravitational wave simulation. You should see an animation at the bottom showing two waves.

Background: *Convergence* of your code *at the correct order* is a necessary and important step in many numerical investigations. This lab is designed to familiarise yourself with how to do these checks of correctness for a simple IBVP that simulates non-linear gravitational plane waves in vacuum (see https://doi.org/10.1103/PhysRevD.89.104026 for details of the system.). This system has many of the fundamental concepts that more complicated numerical relativity codes are built upon. Exact Minkowski space-time is given as the initial data and boundary conditions are used to introduce gravitational waves.

To first show that the code *converges* we need to run multiple simulations at different resolutions and save the output for comparison in post-processing. Comparison of the satisfaction of the constraints over multiple resolutions is the usual thing to look at. As the number of spatial points increases, the constraints should become closer to zero. The jump in satisfaction is the related to the *order* of your numerical methods.

Tasks:

1. Run g_wave_setup.ipynb and store the output for spatial resolutions 100, 200, 400 and 800 (you can make the display boolean False to speed things up). **Note:** The timestep is chosen by $\Delta t = CFL * \Delta z$, so as we double the number of points, the timestep is halved to maintain the same CFL.

2. Open convergence_tests.ipynb. This file post-processes your data and looks at the convergence of the constraint equations in different ways. Evaluate the cells, noting that the last three do the following respectively:

   (i) Shows $\log_{10}$ of a constraint at a fixed timeslice over the entire spatial grid. This gives specific information through the entire spatial grid, but nothing about how the constraints behave with time.

   (ii) Computes the L_2-norm of the difference of $\log_2$ of a constraint between subsequent simulations. Halving the spatial step size for a finite difference operator results in a 2^{order} increase in accuracy.

(iii) Shows $\log_{10}$ of the L_2-norm of a constraint over time. I.e. for each timeslice it sums the squares of the constraint at each spatial grid point and takes a square root of the result. This gives an overview of the constraint at one instance of time as one number, and this is then done for each timeslice.

3. Looking at the plot in (i), why are the constraint not as well satisfied close to the boundaries?

4. Looking at the numbers output from (ii), why are the numbers not around four like the order of the time integrator RK4?

5. Looking at the plot in (iii), can you explain why the constraints are so well satisfied at the start and increase so rapidly?

6. Go back to g_wave_setup.ipynb and change the diffop to be D43_Strand. (The two numbers are the order on the interior of the grid and close to the boundaries respectively). Run another batch of simulations with the same resolutions as before, making sure to name the output files something different.

7. Go back to convergence_tests.ipynb and load in the new HDF5 files. Rerun the notebook. What has changed?

8. **Challenge question:** Can you explain the shape of the plot in (iii), namely why is there a bump close to the start and a jump at around $t = 0.6$? Looking at the animation may help.

Debrief: This type of analysis is needed alongside results of running a code to help justify them by showing the code is indeed *converging* to a solution of the Einstein equations (by way of the constraints converging to zero) at (or at least close to) the predicted *order* corresponding to your numerical methods.

The convergence test on a timeslice can also help diagnose errors/bugs that are local in space, while the convergence test over time can help diagnose instabilities.