Numerical relativity exercise

This exercise will focus on some of the key numerical methods and concepts that are used to evolve the Einstein equations. Since the Einstein equations are so complicated, we will use a 1D wave equation as an example.

1. **Wave equation in 1st order form**

   The wave equation in 1 spatial dimension is

   $$\Box\psi = -\frac{\partial^2\psi(x,t)}{\partial x^2} + \frac{\partial^2\psi(x,t)}{\partial t^2} = 0\ . \tag{1}$$

   (a) Show that the wave equation can be written in first-order form

   $$\partial_t\psi = -\Pi, \tag{2}$$
   $$\partial_t\Pi + \partial_x\Phi = 0, \tag{3}$$
   $$\partial_t\Phi + \partial_x\Pi = 0, \tag{4}$$

   where $\Phi \equiv \partial_x\psi$, and $\Pi \equiv -\partial_t\psi$.

2. **Finite difference operator**

   Assume that we represent functions $f(x)$ on a set of equally-spaced grid points x_j. Then the 2nd-order *finite-difference approximation* for the derivative $f'(x)$ of the function, at the point x_j, is given by

   $$f'(x_j) = \frac{f(x_{j+1}) - f(x_{j-1})}{x_{j+1} - x_{j-1}} \tag{5}$$

   For simplicity, we will assume a periodic domain with N points, labeled 0 to $N-1$, and we will assume that the grid points are $x_j = 2\pi j/N$.

   (a) Write a python function

   ```
   def FD_Derivative(f):
     ...
   ```

   that takes a numpy array f containing $f(x_j)$ for all grid points, and returns a numpy array containing the derivative $f'(x_j)$ for all grid points. Assume that f and f' are periodic. Note that there is no grid point x_N, but the formula (5) needs this point to evaluate $f'(x_{N-1}$; therefore you should use the periodicity of the function, e.g. assume that f at the fictitious point x_N is f at x_0.

   (b) Write a python function

   ```
   def AvgErr(f1,f2):
     ...
   ```

   that computes the average difference of two functions $f(x)$ and $g(x)$ on the grid:

   $$E(f,g) = \sqrt{N^{-1}\sum_j \big(f(x_j) - g(x_j)\big)^2}, \tag{6}$$

(c) Let $f(x) = e^{-\cos(x)}$. Compute the average error $E(f'_n, f'_a)$, which is the average difference between the numerical and analytical solutions. Here $f'_n(x)$ is the result of `FD_Derivative(f)` and $f'_a(x)$ is the analytical derivative. Plot $E(f'_n, f'_a)$ as a function of N. How do the errors behave as N increases?

(d) Repeat part 2c for the function $f(x) = \text{abs}(\cos(x))$. Notice that this function is not smooth; the analytical derivative $f'_a(x)$ is undefined at the kinks. To deal with this, proceed in one of two ways:

- Set $f'_a(x) = 0$ at the kinks (i.e. choose the value that the derivative would have if the kinks were to be symmetrically smoothed out).
- Instead of $f(x) = \text{abs}(\cos(x))$, use $f(x) = \text{abs}(\cos(x + \epsilon))$, where ϵ is some small number like 10^{-8} that will ensure that you never need to evaluate the analytic derivative $f'_a(x)$ at one of the grid points x_j.

How does $E(f'_n, f'_a)$ behave as N increases? Compare with part 2c.

3. **Spectral representation:**

Now we will write $f(x)$ as a spectral expansion: Assume that

$$f(x) = \sum_{k=-N/2}^{N/2-1} f_k e^{ikx} \tag{7}$$

where N is *assumed to be even*, and f_k are complex numbers called spectral coefficients. The inverse transformation is

$$f_k = N^{-1} \sum_{j=0}^{N-1} f(x_j) e^{-ikx_j}, \tag{8}$$

where x_j are the special grid points $x_j = 2\pi j/N$.

The above transformations are called *discrete Fourier transforms*; they are discrete because the sums do not go to infinity, but are truncated at some finite N. It can be shown that if $f(x)$ is approximated as the series (7) (this is an approximation only because the N is finite), then the derivative $f(x)$ at the grid points $x_j = 2\pi j/N$ is given by

$$f'(x_j) = \sum_{k=0}^{N-1} D_{jk} f(x_k), \tag{9}$$

where the derivative matrix D_{jk} is given by

$$D_{jk} = \begin{cases} \frac{1}{2}(-1)^{k+j} \cot \frac{\pi(j-k)}{N}, & k \neq j \\ 0, & k = j \end{cases} . \tag{10}$$

(a) Write a python function

```
def SP_Derivative(f):
  ...
```

that takes an array f of length N, and returns an array containing the derivative $f'(x)$, using Eqs. (9) and (10).

(b) Plot $E(f'_n, f'_a)$ versus N for $f(x) = e^{\cos(x)}$. How does the error behave as a function of N? Compare with the finite-difference method.

(c) Repeat part 3b for the nonsmooth function $f(x) = \text{abs}(\cos(x))$. Compare with the smooth function, and with the finite-difference method.

4. **Evolving the wave equation**

(a) Write a python function

```
def WaveRHS(u,derivfunc):
  ...
```

that takes as input $u = (\psi, \Pi, \Phi)$, where ψ, Π, and Φ are numpy arrays representing the first-order wave equation variables at all grid points, and computes the time derivatives $\partial_t\psi$, $\partial_t\Pi$, and $\partial_t\Phi$ at all grid points. The 'derivfunc' argument is used to pass a function that takes derivatives, for example `FD_Derivative` or `SP_Derivative`. The return value of `WaveRHS` should be a tuple that contains $(\partial_t\psi, \partial_t\Pi, \partial_t\Phi)$.

(b) Write a python function

```
def ForwardEuler(u,Deltat,derivfunc):
  ...
```

that takes as input a tuple of numpy arrays (for example $u = (\psi, \Pi, \Phi)$) at time t, and produces as output the same vector at time $t + \Delta t$. Here 'derivfunc' is the derivative operator function. Use the Forward Euler formula for the update:

$$u(t + \Delta t) = u(t) + \Delta t R(u), \tag{11}$$

where $R(u)$ is the output of `WaveRHS`.

(c) Write a function that initially sets $\psi(x) = e^{-4(x-\pi)^2}$, $\Pi(x) = 0$, and sets $\Phi(x)$ as the derivative of $\psi(x)$, and then evolves $u = (\psi, \Pi, \Phi)$ through multiple time steps using `ForwardEuler`. To start, choose $\Delta t = 0.3$, $N = 100$, and run for about 200 steps.

(d) Visualize your results. For example, you could use `matplotlib` make a pretty plot of $\psi(x_j, t)$ at all x_j and many time levels t at once.

(e) It turns out that `ForwardEuler` is *numerically unstable*. By running for a sufficient number of time steps, you should find that the solution goes haywire. To fix this, write a python function

```
def RungeKutta4(u,Deltat,derivfunc):
  ...
```

that does the same thing as `ForwardEuler` except it updates u according to the 4th order Runge-Kutta method:

$$u(t + \Delta t) = u(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \tag{12}$$

where

$$k_1 = \Delta t R(u), \tag{13}$$
$$k_2 = \Delta t R(u + k_1/2), \tag{14}$$
$$k_3 = \Delta t R(u + k_2/2), \tag{15}$$
$$k_4 = \Delta t R(u + k_3). \tag{16}$$

It can be shown that unlike `ForwardEuler`, `RungeKutta4` can be used for stable evolutions. From now on, we will use `RungeKutta4` instead of `ForwardEuler`.

(f) The Courant-Friedrichs-Lewy (CFL) condition says that if you have too large of a timestep Δt, even with `RungeKutta4`, then the evolution will go unstable. Typically we write $\Delta t_{\max} = C_{max}\Delta x$, where Δx is the minimum grid spacing (here $2\pi/N$) and C_{max} is a number called the CFL factor. Experiment with different values of Δt to determine (approximately; one digit accuracy is fine) C_{max} for the wave equation and `RungeKutta4`, using finite differencing.

(g) Now find C_{max} for spectral methods instead of finite differencing.

(h) Given the initial data we are using, the analytical solution for $\psi(x,t)$ is always positive. For our numerical solution we will find that $\psi(x,t)$ becomes slightly negative, and we can use the negativeness as a rough measure of our error. If you evolve until $t = 10$ using finite differencing, approximately how many grid points N do you need until the minimum of the numerical $\psi(x,t)$ never dips below -10^{-2}? If you evolve until $t = 10$ using spectral methods, about how many grid points N do you need until the minimum of the numerical $\psi(x,t)$ never dips below -10^{-2}?