Homework 4

In this homework, you will implement a new class for the backward differentiation formula implicit timestepping scheme. The `timesteppers.py` file has the initialization signature for the new class, as well as classes for the timestepping schemes discussed in the lecture.

The `BackwardDifferentiationFormula` timestepping class will implement an implicit multistep timestepping method. The timestepper with solve the equation

$$\partial_t u = Lu \tag{1}$$

for a linear operator L specified via `L_op`. The user will also specify the field `u` to be evolved, and the number of steps. The scheme will be

$$\sum_{i=0}^{s} a_i u^{n-i} = Lu^n \tag{2}$$

where s is the number of steps.

First you will have to derive the coefficients a_i such that the timestepping scheme is order s accurate. As you are hopefully getting used to, you can make an order s scheme by performing a Taylor series expansion of each term, and making sure the coefficients of the first $s+1$ terms match their correct values.

As with the multistep Adams-Bashforth methods, you will have to take care when initializing the algorithm, as you will not initially have the s previous values of u necessary for equation 2. Your first few timesteps should use lower order BDF schemes until you reach step s (or something more accurate!).

The first set of tests will use constant timesteps. However, the second set of tests will have variable timesteps. When the timestep changes, so do the values of a_i. Just as you need to keep track of the previous s values of $u^{n-1}, u^{n-2}, \ldots u^{n-s}$ to use the scheme in eqn. 2, you will also need to keep track of the previous timestep sizes, as their values will determine the the correct coefficients a_i. For example, if you take 100 timesteps of size 1, and then 100 timesteps of size 2, for a 3-step scheme, you will need to calculate the coefficients a_i for timesteps (1, 1, 1), (2, 1, 1), (2, 2, 1), and (2, 2, 2), where I am listing the most recent timestep size first.