# Homework 8

In this homework you will implement a new fully-implicit timestepper, and solve two different sets of equations using fully-implicit timestepping.

- **Crank-Nicolson:** You will make a fully-implicit version of the Crank-Nicolson timestepping scheme. The linear Crank-Nicolson discretizes the equation $M \cdot \partial_t X + L \cdot X = 0$ as

$$M \cdot X^{n+1} + \frac{\Delta t}{2} L \cdot X^{n+1} = M \cdot X^n - \frac{\Delta t}{2} L \cdot X^n. \tag{1}$$

  You will implement this as the `CrankNicolsonFI` class in `timesteppers.py`. Similar to `BackwardEulerFI`, you only need to implement as `_step` method, which will be passed a timestep, $dt$ and an optional initial guess. If no initial guess is provided, use the state vector at the current time, $X^n$. Although all the tests will converge starting from $X^n$, it may be useful for testing to break the iteration loop after, e.g., 20 iterations.

- **Burgers' Equation:** You will solve the equation

$$\partial_t u - \nu \partial_x^2 u = -u \partial_x u \tag{2}$$

  will a fully-implicit timestepper. You will implement this equation as the class `BurgersFI` in `equations.py`. The equation set will be initialized with an array containing $u$, as well as the viscosity $\nu$, the desired convergence of the spatial derivatives, and the `grid`. We will solve the equations on a periodic domain. You will also need to define a `self.J` function which returns the Jacobian of the nonlinear term given a statevector. When solving this equation with a fully-implicit timestepper, you should be able to take timesteps greater than the CFL limit.

- **Reaction-Diffusion Equation with Two Species:** You will solve the equations

$$\partial_t c_1 - D \partial_x^2 c_1 = c_1(1 - c_1 - c_2), \tag{3}$$
$$\partial_t c_2 - D \partial_x^2 c_2 = r c_2(c_1 - c_2). \tag{4}$$

  These are reaction-diffusion equations for two species, $c_1$ and $c_2$. The first species $c_1$ will tend to $c_1 = 1 - c_2$ on the timescale of 1, while the second species $c_2$ will tend to $c_1$ on the timescale of $1/r$. You will solve this equation using a fully-implicit timestepper, which will allow you to take timesteps of order unity, even when $r \gg 1$.

  You will implement this equation set in the `ReactionTwoSpeciesDiffusion` class in `equations.py`. The class is instantiated with `X`, a `StateVector` containing the two fields $c_1$ and $c_2$ (in that order). It is also given the values of $D$,

$r$, the desired convergence order of the spatial derivatives, and the `grid`. You will also need to define a `self.J` function which returns the Jacobian of the nonlinear term given a statevector. The class will be used as such:

```
X = timesteppers.StateVector([c1, c2])
rd = equations.ReactionTwoSpeciesDiffusion(X, D, r, 4, grid)
```