Rules and format:

- You are welcome to discuss this exam with me (David Maxwell) to ask for hints and so forth.
- If you find a suspected typo, please contact me as soon as possible and I will communicate it to the class if needed.
- You my not discuss the exam with anyone else until after the due date/time.
- You are permitted to reference any text you would like in solving these problems.
- Each problem is weighted equally.
- The due date/time is absolutely firm.

1. Consider the problem

$$u_{xx} + \gamma u^4 = f(x)$$

on the interval $0 \le x \le 1$ with $u(0) = u(1) = 0$.

a) If $u(x) = \sin(3\pi x)$, what is the value of $f(x)$?

b) Write a code to solve this problem based on the following approach:

(a) Use centered differences to as in Section 2.2 of your text to approximate the second derivative and derive an algebraic system to solve for a vector of unknowns u_i that approximate $u(x_i)$.

(b) Implement a numerical method to solve the system (with user-supplied right-hand side f and constant γ) by applying Newton's method to approximate the solution of the algebraic system. Newton's method will be applied to a system of the form $F(u) = 0$, and iterations should stop when the residual norm $||F(u)||$ has been reduced by 10^{-9} of its original value.

(c) Show that with the verification case from part a), and separately with $\gamma = 0$, that your code exhibits $O(h^2)$ convergence.

2. Text, 5.6 (a)-(c)

3. The TR-BDF2 method is an implicit second-order Runge-Kutta method of the following form.

$$\begin{aligned} u_* &= u_n + \frac{k}{4}\left[f(u_n) + f(u_*)\right] \\ u_{n+1} &= \frac{1}{3}\left[4u_* - u_n + kf(u_{n+1}\right] \end{aligned} \tag{1}$$

a) Show that this method is L-stable.

b) Write a numerical code using TR-BDF2 as the basis for solving the heat equation $u_t = u_{xx}$ in a Method of Lines approach. Verify, using the test case of Homework 6, problem 1c, that you observe $O(h^2)$ convergence.

c) Discuss the merits of this strategy versus Backwards Euler and Crank Nicolson.

4. The aim of this problem is to implement an elementary finite volume scheme for the 1-d advection equation and to practice facility with code modularity and the method of lines. Do not worry that there is a lot of writing in the description of this problem; the aim is to provide a lot of guidance.

The problem we are solving is

$$u_t + au_x = 0$$

with $a > 0$ on the domain $[0,1]$ with periodic boundary conditions.

a) On homework 3 you wrote an Euler's method (RK1) and a RK4 ODE solver each with signature: `ode_solver(f,t0,u0,T,M)`. Modify your code (or the code in the solutions) to create an additional RK2 ODE solver with this same signature. Verify your new solver converges at the anticipated rate by testing with the same test as in homework 3.

b) Finite volume schemes for the advection equation require computation of flux at cell faces. We will break the space domain $[0,1]$ into N subintervals of equal width h and endpoints $x_i = ih$, $i = 0,\ldots,N$. Note that $x_N = 1$ is identified with $x_0 = 0$. Each cell centered at x_i has boundary points $x_{i-1/2}$ and $x_{i+1/2}$ with $x_{i\pm1/2} = x_i \pm h/2$ with the understanding that $x_{-1/2}$ is represented by $x_{N+1/2}$. Write a function `upwind_flux(u,a)` that receives a vector of values of u at points $[x_0,\ldots,x_{N-1}]$ and velocity a and returns the fluxes at $[x_{-1/2}, x_{1/2},\ldots,x_{N-1/2}]$ corresponding to the upwind scheme. You may find the slide "upwind as the **donor cell** method" from Ed's lectures to be helpful. Verify your code is correct by applying it to the case $N = 4$, $u = [1,2,3]$ and $a = 2$. Python users may find the function `np.roll` is handy. MATLAB users might like `circshift`.

c) Write an advection solver with signature
`advection(u0,a,T,N,M,ode_solver,flux_computer)` where

- `u0` is a function taking a vector of x locations as an argument and returning a vector of initial values of u at the corresponding locations.
- `a` is the (positive) wave velocity
- `T` is the final time; the solution is computed on $[0, T]$
- `N` is the number of spatial subintervals of $[0,1]$; recall 0 and 1 are identified.
- `M` is the number of time steps.
- `ode_solver` is a function with the same signature as in part **a** for solving ODEs

- `flux_computer` is a function with the same signature as in part **b** for computing fluxes

Your function should return `(x,t,u)` where `x` and `t` are vectors of x and t values at the grid points, with `x` including both 0 and 1. The matrix `u` should contain the computed value of u at the grid points (your choice of ordering is fine) but should contain values of u at both 0 and at 1.

Test your solver with your forward Euler ODE solver and your upwind flux computer. Use $u_0(x) = \sin(2\pi x)$ as your initial condition, $a = 1$, and a time interval of $T = 4$ so that the final value of u should match the initial value of u. Verify that the error at the final timestep is indeed $O(h)$ by testing with $N = [50, 100, 200, 500, 1000]$ and a suitable corresponding choice of M. Throw out any low values of N needed to detect the order of convergence. MATLAB users will want to recall that if you want to pass a function as an argument, you need to use a function handle; to pass `sin` as an argument, you would use `@sin`.

d) Write a `minmod` flux computer with the same signature and conventions as your upwind flux computer but using the minmod slope limiter. You may find the three slides starting with "slope reconstruction" from Ed's lectures to be helpful. Then repeat the convergence test from part **c**; what order of convergence to do you observe? Do not be worried if you do not like the results that you see. If you have coded your flux limiter correctly, the graph of the solution at the final time will be boxy butwithout spurious oscillations.

e) Repeat the above but using an RK2 ode solver. What order of convergence do you observe? You may again discard low values of N.

f) Repeat the above but using an RK4 ode solver. What order of convergence do you observe? Comment on your observed order of convergence.

g) Part of the point of finite volume methods is to converge at a better rate than $O(h)$ yet not introduce spurious oscillations. Discontinuous initial data is great for triggering those oscillations. Using `T=4` and `N=100` generate a plot of the numerical solution at $T = 4$ using both upwinding and the minmod slope limiter compared to the true solution for

$$u_0(x) = \begin{cases} 1 & |x - 1/2| < 1/4 \\ 0 & \text{otherwise} \end{cases}$$

A small amount of extra credit will be given for adding the results of a Lax Wendroff computation to this figure.