1. The Matlab code shown on the next page illustrates a numerical implementation for solving the 1-D heat equation in the case of:

- Periodic in space over \( x \in [-1,1] \), for time \( 0 \leq t \leq 0.2 \), with a peak-like initial condition.
- Fourier-PS discretization in space, and use of Matlab’s ode45 ODE solver for the time integration.

a. Change the space and time discretizations to FD4 and RK4 (of your own implementation) for the time stepping.

b. With the same discretization level in space as used in the demonstration code, vary the time step so that you experimentally ‘nail down’ the largest time step \( k \) you can use and still have numerical stability.

c. Analyze theoretically what the exact stability condition should be for your RK4/FD4 combination, and reconcile this with the experimental result seen in part b.

d. Returning to the provided Fourier-PS code: Modify it to simultaneously run a second independent instance of a heat equation without using any further FFTs (i.e. apply the technique of running one case in the real part and another case in the imaginary part). Verify that the code works.

2. Consider next the non-periodic case, with boundary conditions \( u(-1,t) = u(+1,t) = 0 \) for \( t \geq 0 \). Implement a Chebyshev-PS space discretization (most easily done by calling a suitable routine from J.A.C. Weideman’s “A Matlab Differentiation Matrix Suite” [http://dip.sun.ac.za/~weideman/research/differ.html](http://dip.sun.ac.za/~weideman/research/differ.html) rather than applying the FCT, etc.) and use a suitable time integrator to again produce a figure similar to the one seen on next page.
function FPS_heat_equation
% Demo code for ode45 / Fourier PS solution of
% PDE   \( u_t = u_{xx} \)
% IC    \( u(x,0) = \frac{0.05}{1.05-\cos(\pi x)} \)
% for time \( t = 0 \) to \( t = 0.2 \)

n = 32; % Set space resolution
x = linspace(-1,1,n+1); x(end) = []; % Lay out data points; skip last
u0 = (0.05./(1.05-cos(pi*x)))'; % Give IC (initial condition)

v = (-pi^2*[0:n/2, n/2-1:-1:1.^2]')'; % Vector to multiply with in Fourier space to get second derivatives
[t,u] = ode45(@dudt,0:0.01:0.2,u0,[],v); % Use ode45 to advance in time

u(:,n+1) = u(:,1); % Fill in right edge in graphics
mesh ([x,1],t,u); colormap([0 0 0]) % Plot result

%---------------------------------
function uxx = dudt(t,u,v) % Calculate uxx by Fourier PS method
uf = fft(u); % Take u to Fourier space
uf = uf.*v; % Take second derivative in Fourier space
uxx = real(ifft(uf)); % Return second derivative to physical space