Problem P37: Consider the following method for solving the advection equation $u_t + au_x$, where a is constant,

$$U_i^{n+1} = U_i^{n-1} - \frac{ak}{h}(U_{i+1}^n - U_{i-1}^n)$$

This applies centered differences to all derivatives; it is the leapfrog method.

(a) Determine the order of accuracy of the truncation error of this method. The answer will be in the form $\tau(x,t) = O(k^p + h^q)$; determine p, q.

Solution:
The truncation error for the leapfrog scheme applied to the advection equation is based on the form,

$$\frac{U_i^{n+1} - U_i^{n-1}}{k} = -a\frac{U_{i+1}^n - U_{i-1}^n}{h}.$$

Computing the truncation error centered as (x_j, t_n) we get the following.

$$\tau(x,t) = \frac{u(x,t+k) + u(x,t-k)}{k} + a\frac{u(x+h,t) + u(x-h,t)}{h}.$$

Via Taylor's Theorem we expand about $u(x,t) = u$ and we get,

$$\begin{aligned}\tau(x,t) = \frac{1}{k}\Big(&(u + ku_t + \frac{k^2}{2}u_{tt} + \frac{k^3}{6}u_{ttt} + O(k^4)) \\ &- (u - ku_t + \frac{k^2}{2}u_{tt} - \frac{k^3}{6}u_{ttt} + O(k^4))\Big) \\ +\frac{a}{h}\Big(&(u + hu_x + \frac{h^2}{2}u_{xx} + \frac{h^3}{6}u_{xxx} + O(h^4)) \\ &- (u - hu_x + \frac{h^2}{2}u_{xx} - \frac{h^3}{6}u_{xxx} + O(h^4))\Big)\end{aligned}$$

$$\begin{aligned}\tau(x,t) = \frac{1}{k}&\Big(2ku_t + \frac{k^3}{3}u_{ttt} + O(k^5)\Big) \\ +\frac{a}{h}&\Big(2hu_x + \frac{h^3}{3}u_{xxx} + O(h^5)\Big)\end{aligned}$$

$$\tau(x,t) = 2u_t + \frac{k^2}{3}u_{ttt} + O(k^4) + 2au_x + \frac{h^2}{3}au_{xxx} + O(h^4)$$

By substitution of the exact solution $u_t = -au_x$ and computing the following $u_{ttt} = -a^3u_{xxx}$ we get the following,

$$\begin{aligned}\tau(x,t) &= -2au_x + \frac{k^2}{3}(-a^3)u_{xxx} + O(k^4) + 2au_x + \frac{h^2}{3}au_{xxx} + O(h^4) \\ &= \frac{k^2}{3}(-a^3)u_{xxx} + \frac{h^2}{3}au_{xxx} + O(h^4) + O(k^4) \\ &= \frac{1}{3}au_{xxx}\left(-(a)^2k^2 + h^2\right) + O(h^4) + O(k^4) \\ &= O(k^2 + h^2).\end{aligned}$$

(b) Apply a von Neumann analysis.

Solution:

(c) State the MOL ODE system $U(t)' = AU(t)$ from which the above method comes. Assuming periodic boundary condition on the interval $x \in [0,1]$, what are the eigenvalues of A? Then derive the method by applying the midpoint ODE method to it. By looking up the stability region of the midpoint method, explain what is understood about the stability of this PDE method.

Solution:
The leapfrog method as an MOL ODE system with periodic boundary conditions is given by,

$$U_m^{n+1} = U_m^{n-1} - \frac{ak}{h}(U_{m+1}^n - U_{m-1}^n) \qquad \text{i} = 1{,}2,\dots,m+1$$

$$\frac{U^{n+1} - U^{n-1}}{2k} = -\frac{a}{2h}\begin{bmatrix} 0 & 1 & & \dots & & -1 \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ 1 & & & & -1 & 0 \end{bmatrix} U^n,$$

$$U'(t) = -\frac{a}{2h}\begin{bmatrix} 0 & 1 & & \dots & & -1 \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ 1 & & & & -1 & 0 \end{bmatrix} U(t).$$

Since our matrix is circulant it's eigenvectors are known and eigenvalues are easily computed. From our text we get that it's eigenvalues are of the form,

$$\lambda_p = -\frac{ia}{h}\sin(2\pi p h) \qquad \text{i} = 1{,}2,\dots m+1.$$

Applying the ODE midpoint method to this system we get,

$$U'(t) = -\frac{a}{2h}\begin{bmatrix} 0 & 1 & & \dots & & -1 \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ 1 & & & & -1 & 0 \end{bmatrix} U(t)$$

$$\frac{U^{n+1} - U^{n-1}}{2k} = -\frac{a}{2h}(U^n_{m+1} - U^n_{m-1}) \qquad \text{i} = 1,2,\dots,m+1$$

$$U^{n+1} = U_i^{n-1} - \frac{ak}{h}(U^n_{m+1} - U^n_{m-1}).$$

Recovering the original method as desired. The stability region of the midpoint method denoted by $z = i\alpha$ where $|\alpha| < 1$. Note that,

$$|k\lambda_p| = |k(-\frac{ia}{h}\sin(2\pi ph))| \le \left|\frac{iak}{h}\right| = \left|\frac{ak}{h}\right|.$$

Therefore in order for this method to achieve absolute stability we require,

$$\frac{|a|k}{h} \le 1.$$

(d) Implement this leapfrog method on the following periodic boundary condition problem: $x \in [0, 1]$, $a = .5$, $t_f = 10$, $u(x, 0) = \sin(6\pi x)$. To make the implementations work you will have to compute the first step by some other scheme; describe and justify what you do.

Solution:
The following code implements this leapfrog method. Forward Euler was used for the first step. Forward Euler is a consistent method so as we decrease the size of this initial step the error will also decrease. Since we are only using it for one step, the error does not compound and therefore we don't need to worry about stability (even though Forward Euler applied to this whole problem is always unstable).
Code:

```
function [tt,zz,xx] = AdvectLeapfrog(feta,t0,tf,a,m,N)
% AdvectLeapfrog  Solve
%
%    u_t = -au_x  , u(0, x) = eta
%
% for u(t, x) on the interval [t0,tf] with N steps in time
% and m+1 steps in space (extra point is the repeated boundary
% condition). Midpoint in time via method of lines.
%
% Usage: [tt,zz,xx] = AdvectLeapfrog(feta,t0,tf,a,m,N)
% Compute step size in space
% Dividing our spatical interval
% with m+1 points means m subintervals
h = 1/(m);

% Define step in time. This one stays the same
k = (tf - t0) / N;

% System factor
r = ((a*k)/(2*h));

% Generate matrix for system of IVPs in time
% Midpoint in space matrix multiplied by time spacing.
A = spdiags([-1*(ones(m,1)),zeros(m,1), ones(m,1)],[-1, 0, 1], m, m);
A(1, end) = -1;
A(end, 1) = 1;
A = -r*A;

tt = t0:k:tf;          % row vector of times
% I AM CHOOSING TO KEEP THE FIRST SPATIAL TERM
xx = linspace(0, 1, m+1); % row vector of spatial location
eta = feta(xx(1:end-1)');   % Solving for initial conditions


% Setting up solution matrix
zz = zeros(m, N+1);   % jth column is U at t_{j-1}
zz(:,1) = eta; % Initial condition

% Forward Euler with same midpoint spacial discretiation
% for the initial step.
zz(:,2) = zz(:,1) + A*zz(:,1);

for j = 2:N
    zz(:,j+1) = zz(:,j-1) + A*zz(:,j);
end
% I AM ADDING THE LAST SPATIAL TERM TO THE ARRAY
zz = [zz ; zz(1,:)];
```

(e) Noting that the final time is $t_f = 10$, the exact solution in part $u(t, x_f) = \sin(6\pi x)$; explain why. Then use $h = .1, .05, .02, .01, .005, .002$ and $k = h$ and show a log-log convergence plot using the infinity norm for the error. What $O(h^p)$ do you expect for the rate of convergence, and what do you measure?

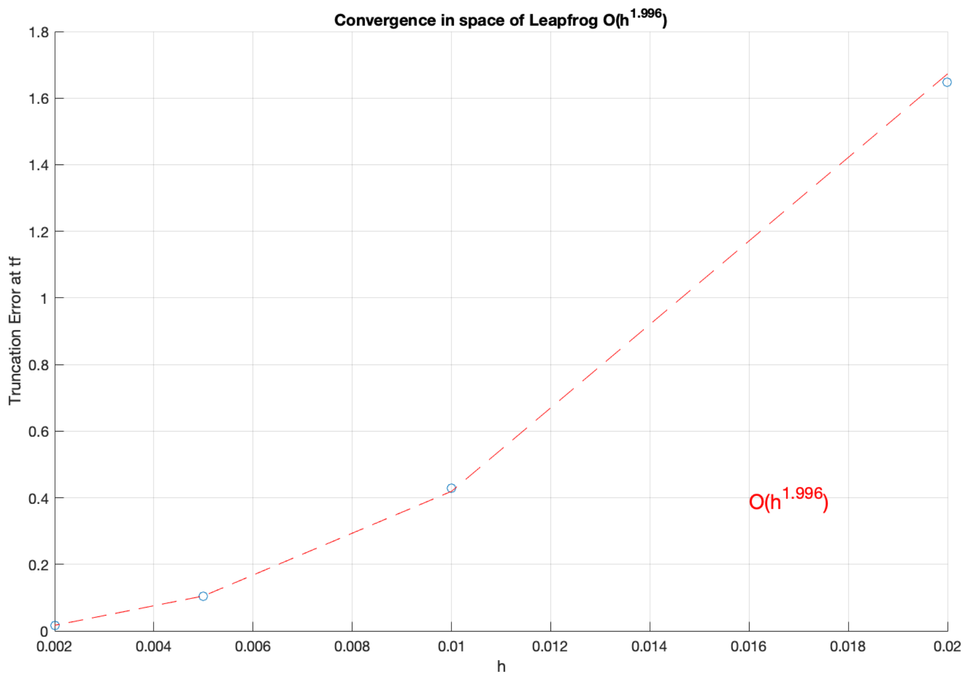
Solution:
From Appendix E, we know that via Fourier transform, given an initial value $u(x, 0) = \eta(x)$, the exact solution to the problem,

$$u_t + au_x = 0$$

is given by $u(x,t) = \eta(x - at)$. In this case our solution becomes $u(x,t) = \eta(x - .5t) = \sin(6\pi(x - .5t)) = \sin(6\pi x - 3\pi t)$. Applying our final time $t_f = 10$ and the sin difference formula we get,

$$u(x, t_f) = \sin(6\pi x - 30\pi) = \sin(6\pi x)\cos(30\pi) - \cos(6\pi x)\sin(30\pi) = \sin(6\pi x).$$

Using the given grid refinements we get the following convergence plot for the leapfrog method which shows a convergence of $\approx O(h^2)$ in space. This is as expected as we saw this same convergence in the truncation error analysis.



Code:

```
feta = @(x) sin(6*pi*x);
t0 = 0;
tf = 10;
a = .5;


uexact = @(t,x) sin(6*pi*x);
Nlist = [.1, .05, .02, .01, .005, .002];
err = [];

for k = 1:length(Nlist)
    [tt,zz,xx] = AdvectLeapfrog(feta,t0,tf,.5, 1/(Nlist(k)), 10/(Nlist(k)));
    err = [err max(abs(zz(3:end,end-1) - uexact(tf, xx(3:end))'))];
end
```

```
p = polyfit(log(Nlist(3:end)),log(err(3:end)),1);
fprintf('convergence at rate O(h^k) with k = %f\n',p(1))
hold on
grid on
loglog(Nlist(3:end),err(3:end),'o', ...
    Nlist(3:end),exp(p(2)+ p(1)*log(Nlist(3:end))),'r--')
xlabel h,  ylabel('Truncation Error at tf')
text(0.016,0.4,sprintf('O(h^{%.3f})',p(1)),'Color','r','FontSize',14)
title(sprintf('Convergence in space of Leapfrog O(h^{%.3f})',p(1)))
```

Problem P38: Consider the nonlinear Poisson equation in 2D,

$$u_{xx} + u_{yy} + \gamma u^3 = g(x, y)$$

on the unit square $(x, y) \in [0, 1] \times [0, 1]$, subject to zero Dirichlet boundary conditions.

(a) We can manufacture a solution because we are free to choose the right-hand side $g(x, y)$. Let us define,

$$u(x, y) = \sin(\pi x)\sin(2\pi y).$$

This is a smooth function which satisfies the boundary conditions. Compute $g(x, y)$ so that u is an exact solution of the differential equation. Note that the $g(x, y)$ formula will depend on γ.

Solution:
First we take the second partial derivatives of $u(x, y)$ to get u_{xx} and u_{yy}. Note that

$$u_{xx} = -\pi^2 \sin(\pi x)\sin(2\pi y),$$

$$u_{yy} = -4\pi^2 \sin(\pi x)\sin(2\pi y)$$

Then by substitution we get our desired function $g(x, y)$,

$$\begin{aligned} g(x, y) &= (-\pi^2 - 4\pi^2 + \gamma u^2)u \\ &= (-5\pi^2 + \gamma u^2)u \end{aligned}$$

(b) based on the Matlab program heat2d.m for example, write a numerical solver for the the previous problem. Use this approach:

(a) Use centered differences with spacing $h_x = h_y = 1/(m + 1)$.

(b) Approximate the PDE by a nonlinear system,

$$F(U) = 0$$

Here $U \in \mathbb{R}^N$ denotes the solution of the algebraic equations, $N = m^2$, and the function $F(V)$ is the residual for a current estimate V.

(c) Solve the algebraic equations by Newton's method. Use $U^{[0]} = 0$ as the initial iterate.

(d) Stop the Newton iteration when the residual norm

$$|F(U^{[k]})|$$

is reduced by 10^{-9} of the initial residual norm.

(e) Observe that when you set $\gamma = 0$ your code should solve the linear Poisson problem $u_{xx} + u_{yy} = g(x, y)$, by doing one Newton step.

Solution:
Modifying the heat2d.m code slightly we can apply the zero Dirichlet boundary conditions, and apply centered differences in both dimension. Then we need to modify the linear solve step, to do Newton's method to solve $F(U) = 0$. Writing out $F(U)$ we get the following where A is the block tridiagonal matrix from discretizing in two dimensions,

$$F(U) = -g(x, y) + \gamma U^3 + AU$$

Since we will be performing Newton's method on this system we'll need to compute the Jacobian. Since $-g(x, y)$ is not a function of U, it goes away when we differentiate so we get the following,

$$J = I3\gamma(U)^2 + A.$$

I know I can implement this very easily, by just adding a Newton method loop in the heat2d.m code in place of the linear solve.