Solution set 2:

Finite Difference Schemes

Exercise 2.1 *Consistency* A method is consistent if its local truncation error T_k satisfies

$$T_k(x,t) = O\left(k^p\right) + O\left(h^q\right) \qquad \text{where} \qquad p,q > 0\ . \tag{1}$$

Essentially, consistency tells us that we are approximating the solution of the correct PDE.
Convergence A method is convergent if the error E_k satisfies

$$\|E_k\left(\cdot,t\right)\| \to 0 \qquad \text{as} \quad k \to 0\ , \quad \forall t \geq 0\ . \tag{2}$$

Stability A method $U^{n+1} = H_k U^n$ is stable if for each $T > 0$ there exist constants C and k_0, such that

$$\|H_k^n\| \leq C \qquad 0 \leq nk \leq T\ , \quad 0 < k < k_0\ . \tag{3}$$

The importance of stability is made clear by the Lax equivalence theorem. In class we saw a proof of one direction of this equivalence: if a method $U^{n+1} = H_k U^n$ is consistent and stable, then it is convergent. From the proof it is clear that stability allows us, in principle, to approximate a solution at any finite time interval $[0,T]$, and to any degree of accuracy.

Exercise 2.2 (b) Let u be a smooth solution of $u_t + au_x = 0$. The local truncation error for the Leapfrog scheme is given by

$$kT_k(x,t) = u(x,t+k) - u(x,t-k) + \frac{ak}{h}\left(u\left(x+h,t\right) - u\left(x-h,t\right)\right)\ . \tag{4}$$

Next, we expand all the terms on the right hand side of the last equation about (x,t). For example, we have $u(x,t+k)$ and $u(x,t-k)$ given by

$$u(x,t+k) = u + u_t k + \frac{1}{2}u_{tt}k^2 + O\left(k^3\right) \tag{5}$$

and

$$u(x,t-k) = u - u_t k + \frac{1}{2}u_{tt}k^2 + O\left(k^3\right)\ , \tag{6}$$

where for simplicity of notation we have replaced $u\left(x,t\right)$, $u_t\left(x,t\right)$ and $u_{tt}\left(x,t\right)$ by u, u_t and u_{tt}, respectively. It is, thus, clear that

$$u(x,t+k) - u(x,t-k) = 2ku_t + O\left(k^3\right)\ . \tag{7}$$

By repeating the calculation also for the shifts in space $u(x \pm h,t)$, we get

$$kT_k\left(x,t\right) = 2ku_t + O\left(k^3\right) + \frac{ak}{h}\left(2hu_x + O\left(h^3\right)\right)\ , \tag{8}$$

which implies

$$T_k\left(x,t\right) = 2\left(u_t + au_x\right) + O\left(k^2\right) + O\left(h^2\right)\ . \tag{9}$$

Since u satisfies $u_t + au_x = 0$, the local truncation error is simply $O\left(k^2\right) + O\left(h^2\right)$.
(c) Observe that at the nth time level, to calculate U^{n+1}, the Leapfrog scheme requires the values, not only of U^n, but also of U^{n-1}. Compared to the LF and LW schemes which use only U^n to calculate U^{n+1}, this is a clear disadvantage. In computations, keeping the numerical solution at more than one time level multiplies the size of memory required for the program.

Exercise 2.3 **(a)** To show that the LF scheme is stable provided

$$\frac{|a|k}{h} \leq 1 \ , \tag{10}$$

we will show that $\|H^n\|$ is bounded for all n, where H is the operator satisfying $U^{n+1} = HU^n$. To do so, we calculate

$$\begin{aligned}
\left\|U^{n+1}\right\| = h\sum_j \left|U_j^{n+1}\right| &= h\sum_j \left|\frac{1}{2}\left(U_{j+1}^n + U_{j-1}^n\right) - \frac{ak}{2h}\left(U_{j+1}^n - U_{j-1}^n\right)\right| \\
&\leq \frac{h}{2}\sum_j \left(\left|1 - \frac{ak}{h}\right| \cdot \left|U_{j+1}^n\right| \ + \ \left|1 + \frac{ak}{h}\right| \cdot \left|U_{j-1}^n\right|\right) \\
&\leq \frac{1}{2}\left(\left|1 - \frac{ak}{h}\right| + \left|1 + \frac{ak}{h}\right|\right) \cdot \|U^n\| \ .
\end{aligned} \tag{11}$$

Owing to the calculation above, whenever (10) holds, we have

$$\|HU^n\| = \left\|U^{n+1}\right\| \leq \|U^n\| \tag{12}$$

and therefore

$$\|H^n\| \leq \|H\|^n \leq 1 \qquad\qquad \forall n = 1, 2, \dots \ . \tag{13}$$

(b), (c) The CFL condition requires that the domain of dependence of the conservation law is contained in the *numerical* domain of dependence. Therefore, the CFL condition is necessary for stability, however, as the example in **(e)** shows, it is not sufficient.
(d) Since the CFL condition is necessary for stability, and (10) ensures stability, it is clear that (10) implies that the CFL condition is satisfied. Notice that in this example, the CFL condition is in fact necessary and sufficient for stability.
(e) The CFL condition is not always sufficient for stability. For example The CFL condition of the scheme

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}\left(U_{j+1}^n - U_{j-1}^n\right) \tag{14}$$

is given by $|a|k/h \leq 1$, the same as the CFL of the LF scheme. However, we know that (14) is unconditionally unstable.

Exercise 2.4 **(a)** Matlab code for implementing the schemes to solve the advection can be found on the last two pages of this solution manual. **(b)** In Figure 1, $u\left(x, t = 0.5\right))$ is plotted as solved by the Upwind, Lax-Friedrichs, Lax-Wendroff and Beam-Warming methods. **(c)** Both Upwind and Lax-Friedrichs catch the jump, but we observe less numerical dissipation from the upwind scheme. This is because the upwind scheme exploits that information is only moving in one direction. The higher order methods Lax-Wendroff and Beam-warmng both introduce oscillations around the discontinuities. **(d,e)** See Figure 2. **(f)** Notice how, on this problem with non-smooth solutions, the rate of convergence of the first order methods is now $\mathcal{O}\left(h^{1/2}\right)$, while the rate of convergence of the second order methods are $\mathcal{O}\left(h^{2/3}\right)$ at best. Such a low order of accuracy is not practical for the solution of real problems, and in the next lecture other methods that handle discontinuities better are introduced.

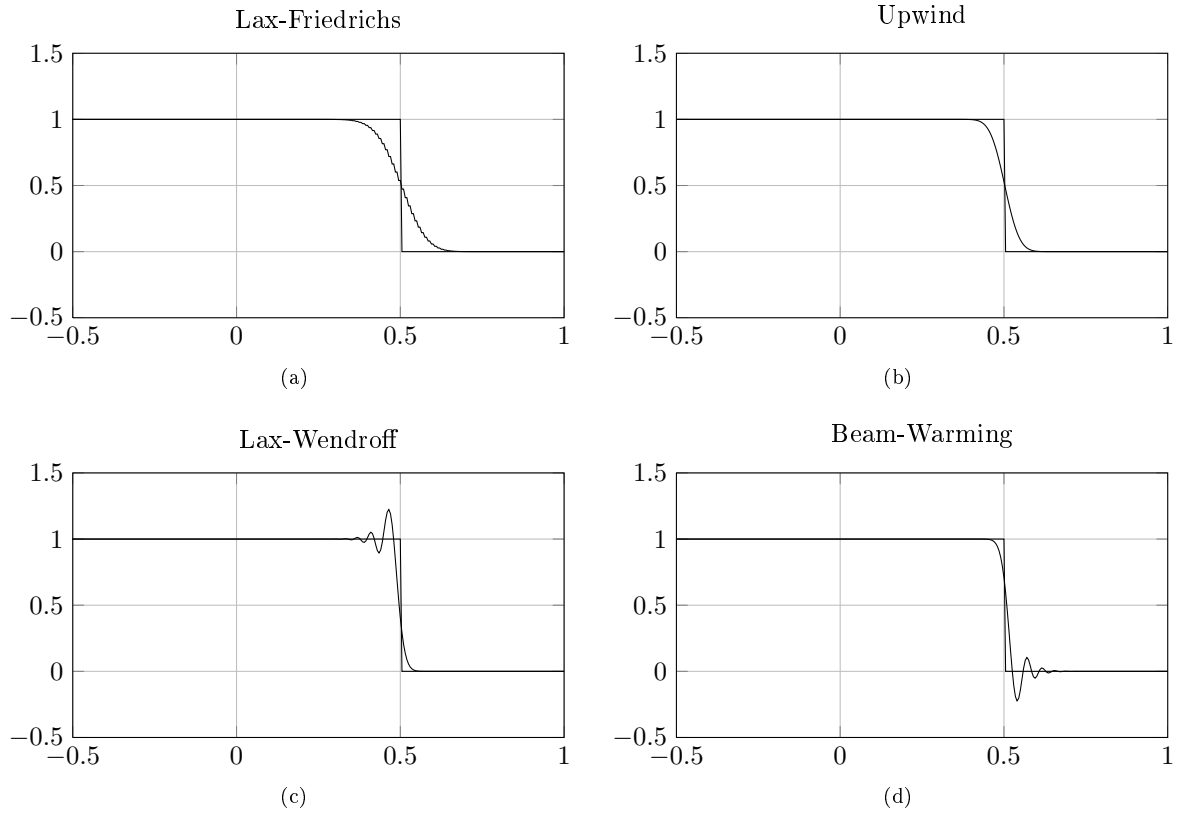

Figure 1: The analytical solution and the numerical solution $u(x, t = 0.5)$ for the advection equation on Riemann initial data with $a = 1$, $h = 0.005$, $\frac{k}{h} = 0.5$.

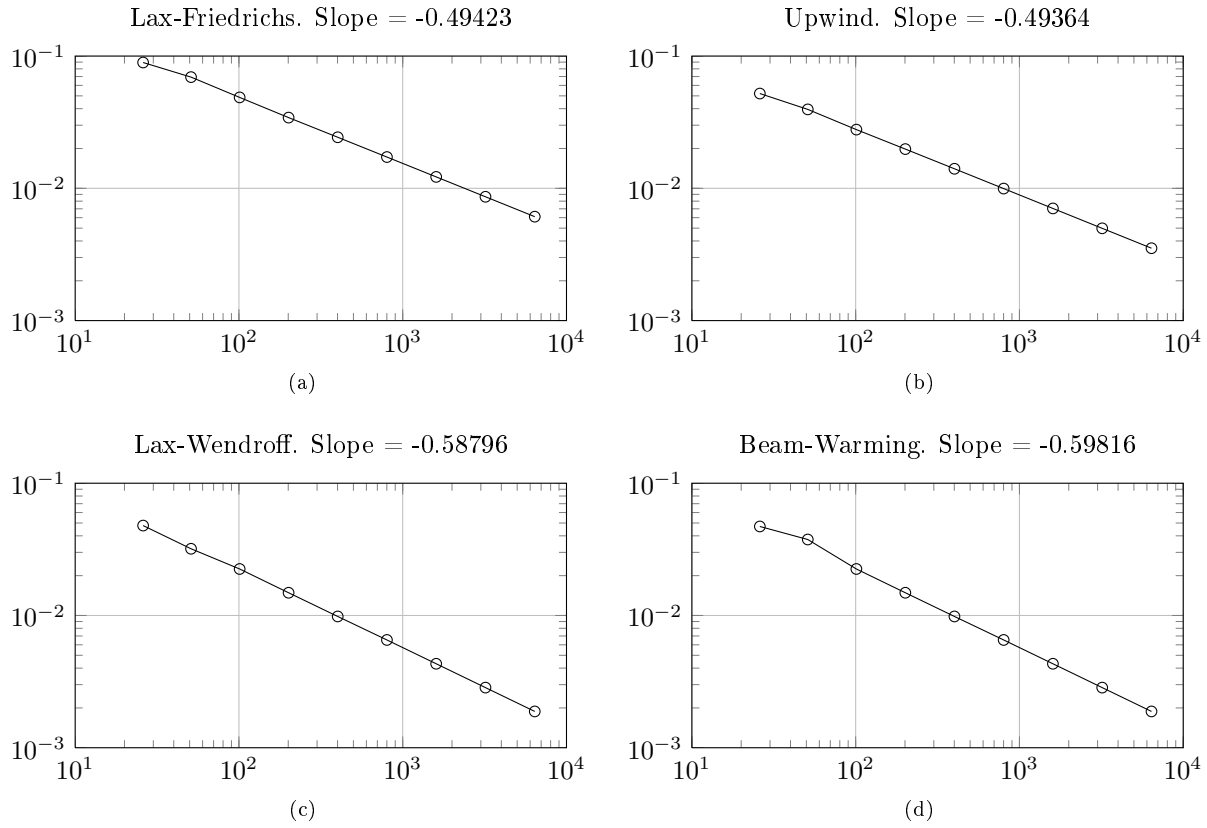

Figure 2: The error as a function of resolution. The error is measured in 1-norm, the resolution is measured as the number of discretization points in the x dimension.

```
% solution02.m clear all,clc
% This script was written for EPFL MATH459, Numerical Methods for Conservation Laws
% and tested with Octave 3.6.4. The scalar advection equation du/dx + a * du/dt = 0
% with riemann initial data is solved using a Leapfrog, Upwind, Lax-Friedirch, Lax-
% Wendroff or Beam-Warming numerical scheme.
% The solution is visualized and the accuracy of the scheme tested.

% Choose SolverNumber
% 1:    Upwind
% 2:    LaxFriedrich
% 3:    LaxWendroff
% 4:    BeamWarming

SolverNumber    = 1;
PlotSolution    = 0;
TestAccuracy    = 0;

% Function declaration
function [A] = Upwind(C,nX)
    A = diag(C*ones(nX-1,1),-1)+diag((1-C)*ones(nX,1),0);
end

function [A] = LaxFriedrich(C,nX)
    A = diag(0.5*(1+C)*ones(nX-1,1),-1)+diag(0.5*(1-C)*ones(nX-1,1),1);
end

function [A] = LaxWendroff(C,nX)
    A = diag(0.5*(C^2+C)*ones(nX-1,1),-1)+diag((1-C^2)*ones(nX,1),0)+ ...
    diag(0.5*(C^2-C)*ones(nX-1,1),1);
end

function [A] = BeamWarming(C,nX)
    A = diag((1-1.5*C+0.5*C^2)*ones(nX,1),0)+diag((2*C-C^2)*ones(nX-1,1),-1)+ ...
    diag((-0.5*C+0.5*C^2)*ones(nX-2,1),-2);
end

function [A,name] = GetMatrix(SolverNumber,C,nX)
    switch SolverNumber
    case 1
        A = Upwind(C,nX);
        name = 'Upwind';
    case 2
        A = LaxFriedrich(C,nX);
        name = 'Lax-Friedrichs';
    case 3
        A = LaxWendroff(C,nX);
        name = 'Lax-Wendroff';
    case 4
        A = BeamWarming(C,nX);
        name = 'Beam-Warming';
    end
end
```

```
% Visualization
if PlotSolution
    a = 1;
    % Discretization
    h = 0.0025;
    k = 0.5*h;
    X = -1:h:1;
    nX = numel(X);
    T = 0:k:0.5;
    nT = numel(T);
    % Generate correct solution
    u = zeros(nX,nT);
    for i = 1:nT
        u(find(X<=a*T(i)),i) = 1;
    end
    % Solve using the numerical scheme
    [A,name] = GetMatrix(SolverNumber,a*k/h,nX);
    U = zeros(nX,nT);
    U(:,1) = u(:,1);
    for i = 1:nT-1
        U(:,i+1) = A*U(:,i);
        U(1:2,i+1) = 1;
    end
    % Make a plot
    for i = 1:nT
        plot(X,U(:,i),'-r',X,u(:,i),'-b');
        ylim([-0.5 1.5]);grid on;title(name);
        drawnow
    end
end

% Accuracy test
if TestAccuracy
    H = [0.08,0.04,0.02,0.01,0.005,0.0025,0.00125];
    E = zeros(numel(H),1);
    n = zeros(numel(H),1);
    % Find Error
    for i = 1:numel(H)
        a = 1;
        % Discretization
        h = H(i);
        k = 0.5*h;
        X = -1:h:1;
        nX = numel(X);
        T = 0:k:0.5;
        nT = numel(T);
        % Generate correct solution
        u = zeros(nX,1);
        u(find(X<=a*T(end))) = 1;
        % Solve using the numerical scheme
        U = zeros(nX,1);U(find(X<=0)) = 1;
        [A,name] = GetMatrix(SolverNumber,a*k/h,nX);
        for j = 2:nT
            U = A*U;
            U(1:2) = 1;
        end
        % Measure error in 1 norm
        E(i) = sum(abs(U(:,end)-u))/nX;
        n(i) = nX;
    end
    % Make loglog plot
    p = polyfit(log(n),log(E),1);
    loglog(n,E,'-ok');grid on;
    xlabel('Resolution');ylabel('Error');
    title([name,'. Slope = ',num2str(p(1))]);
end
```