Final Exam

AM213B

Kevin Silberberg

2025-06-09

1 Problem 1

Consider the Runge-Kutta (RK) method for solving $u' = f(u,t)$:

$$k_1 = hf(u_n + 2k_1, t_n + 2h) \tag{1}$$
$$u_{n+1} = u_n + k_1 \tag{2}$$

1.1 Part 1

Derive the stability function $\phi(z)$ of the RK method.

1.1.1 Solution

Let us write out the butcher tableau for (1). This is an implicit 1-stage method.

A general 1-stage Runge-Kutta method has the form.[1]

$$k_1 = u_n + ha_{11}f(k_1, t_n + c_1 h) \tag{3}$$
$$u_{n+1} = u_n + hb_1 f(k_1, t_n + c_1 h) \tag{4}$$

thus we have that

$$a_{11} = 2 \tag{5}$$
$$b_1 = 1 \tag{6}$$
$$c_1 = 2 \tag{7}$$

and the butcher tableau for the method is

$$\begin{array}{c|c} 2 & 2 \\ \hline & 1 \end{array} \tag{8}$$

This satisfies the consistency requirements

[1] LeVeque (2007) page 126

$$\sum_{j=1}^{r} a_{ij} = c_i \tag{9}$$

$$\sum_{j=1}^{r} b_j = 1 \tag{10}$$

The stability function of an RK method is given by the following.[2]

$$\phi(z) = 1 + z\mathbf{b}^T (\mathbf{I} - z\mathbf{A})^{-1} \mathbf{1} \tag{11}$$

$$\phi(z) = 1 + \frac{z}{1-2z} \tag{12}$$

1.2 Part 2

Is the RK method A-stable? Why?

1.2.1 Solution

An RK method is A-stable if its stability function satisfies

$$|\phi(z)| \leq 1, \quad \text{whenever} \quad \text{Re}(z) \leq 0 \tag{13}$$

where $\text{Re}(z)$ denotes the real part of the complex number z.

We need to prove that the stability function (12) satisfies (13).

Proof. first let us rewrite the stability function,

$$\phi(z) = 1 + \frac{z}{1-2z} \tag{14}$$

$$= \frac{1-2z}{1-2z} + \frac{z}{1-2z} \tag{15}$$

$$= \frac{1-z}{1-2z} \tag{16}$$

Let $z = a + ib$ where $a, b \in \mathbb{R}$, $i = \sqrt{-1}$, and $a \leq 0$.

rewriting (16) in terms of a, b and then computing the modulus we have,

$$\phi(a+ib) = \frac{1-a-ib}{1-2a-i2b} \tag{17}$$

$$|\phi(a+ib)| = |\frac{1-a-ib}{1-2a-i2b}| \tag{18}$$

$$= \frac{|1-a-ib|}{|1-2a-i2b|} \tag{19}$$

$$= \frac{\sqrt{(1-a)^2 + (-b)^2}}{\sqrt{(1-2a)^2 + (-2b)^2}} \tag{20}$$

[2] Quarteroni, Sacco, and Saleri (2006) page 516

For A-stability, we simply need to show that the numerator is less than the denominator for all $a, b \in \mathbb{R}$ where $a \leq 0$.

$$\sqrt{(1-a)^2 + b^2} \leq \sqrt{(1-2a)^2 + 4b^2} \tag{21}$$

for all $a \leq 0$. Let us simplify and expand both sides.

$$1 - 2a + a^2 + b^2 \leq 1 - 4a + 4a^2 + 4b^2 \tag{22}$$
$$2a - 3a^2 \leq 3b^2 \tag{23}$$
$$a(2 - 3a) \leq 3b^2 \tag{24}$$

Notice that the RHS is always positive for all $b \in \mathbb{R}$. Additionally the LHS is always negative for all $a \leq 0$. Thus the inequality holds for all $a, b \in \mathbb{R}$ where $a \leq 0$ and the condition for A-stability is satisfied. □

We can conclude that the RK method (1) satisfies the condition for A-stability (13).

1.3 Part 3

Is the method L-stable? Why?

1.3.1 Solution

L-stability is a special case of A-stability. A method is said to be L-stable if it is both A-stable and the stability function $\phi(z) \to 0$ as $z \to \infty$.

Let us compute the limit of the stability function as $z \to \infty$ to determine if the method is L-stable.

$$\lim_{z\to\infty} \frac{1-z}{1-2z} = \lim_{z\to\infty} \frac{\frac{d}{dz}(1-z)}{\frac{d}{dz}(1-2z)} \tag{25}$$
$$= \frac{-1}{-2} \tag{26}$$
$$= \frac{1}{2} \tag{27}$$

thus the method is not L-stable.

2 Problem 2

Consider the two linear multi-step methods (LMMs) for solving $u' = f(u, t)$.

$$u_{n+2} - \frac{3}{2}u_{n+1} + \frac{1}{2}u_n = hf(u_{n+1}, t_{n+1}) \tag{28}$$

$$u_{n+2} + u_{n+1} - 2u_n = 3hf(u_{n+2}, t_{n+2}) \tag{29}$$

2.1 Part 1

For each method, find whether or not it is consistent.

2.1.1 Solution

A linear multi-step method is said to be consistent if $\tau(h) \to 0$ as $h \to 0$. Where $\tau(h) = \max_n |\tau_n(h)|$ and τ_n is the local truncation error of computing the solution from time $t_n \to t_{n+1}$. More over, if $\tau(h) = O(h^q)$, for some $q \geq 1$, then the method is said to have order q.[3]

Let us find the local truncation error (LTE) for the method (28).

We can find the LTE of the method by considering the difference between the exact solution and the approxiamte solution computed by the method.

$$\tau(h) = u(t_{n+2}) - \frac{3}{2}u(t_{n+1}) + \frac{1}{2}u(t_n) - \left(u_{n+2} - \frac{3}{2}u_{n+1} + \frac{1}{2}u_n\right) \tag{30}$$

Notice that we can center all the terms about the point t_{n+1} by offsetting the time step h such that

$$t_{n+2} \to t_{n+1} + h \tag{31}$$
$$t_n \to t_{n+1} - h \tag{32}$$

Let us taylor expand the exact solution about the point t_{n+1} .

We use the short hand notation:

$$u(t_{n+1}) = u \qquad \frac{d^m}{dt^m}(u(t_{n+1})) = u^{(n)} \qquad f(u(t_{n+1}), t_{n+1}) = f \tag{33}$$

$$u(t_{n+1} + h) = u + hu^{(1)} + \frac{h^2}{2!}u^{(2)} + O(h^3) \tag{34}$$
$$u(t_{n+1}) = u \tag{35}$$
$$u(t_{n+1} - h) = u - hu^{(1)} + \frac{h^2}{2!}u^{(2)} + O(h^3) \tag{36}$$

$$u(t_{n+2}) - \frac{3}{2}u(t_{n+1}) + \frac{1}{2}u(t_n) = \frac{h}{2}u^{(1)} + \frac{3h^2}{4}u^{(2)} + O(h^3) \tag{37}$$

Recalling that for the general ODE $u^{(n)} = f^{(n-1)}$, the LTE for (28) is thus

$$\tau(h) = \frac{h}{2}u^{(1)} + \frac{3h^2}{4}u^{(2)} - hu^{(1)} + O(h^3) \tag{38}$$
$$= -\frac{h}{2}u^{(1)} + \frac{3h^2}{4}u^{(2)} + O(h^3) \tag{39}$$
$$= O(h) \tag{40}$$

Thus the method (28) is consistent with order 1.

Let us find the (LTE) for the method (29).

We need to compute the taylor expansion of $f(u_{n+2}, t_{n+2})$ centered at t_{n+1}.

[3] Quarteroni, Sacco, and Saleri (2006) page 489

$$f(u(t_{n+1}+h), t_{n+1}+h) = f + hf^{(1)} + \frac{h^2}{2}f^{(2)} + O(h^3) \tag{41}$$

Using the same notation and results computed above the LTE is thus,

$$\tau(h) = u - 2u + u + hu^{(1)} + 2hu^{(1)} + \frac{h^2}{2}u^{(2)} - h^2u^{(2)} + O(h^3) - 3h(f + hf^{(1)} + \frac{h^2}{2}f^{(2)}) \tag{42}$$

$$= 3hu^{(1)} - \frac{h^2}{2}u^{(2)} - 3hu^{(1)} - 3h^2u^{(2)} - \frac{3h^3}{2}u^{(3)} + O(h^4) \tag{43}$$

$$= O(h^2) \tag{44}$$

Thus the method (29) is consistent with order 2.

2.2 Part 2

For each method, find whether or not it is zero-stable.

2.2.1 Solution

An LMM is zero-stable if and only if it satisfies the following root-condition,
for a numerical method of the form

$$\sum_{j=0}^{q} \alpha_j u_{n+j} = h \sum_{j=0}^{q} \beta_j f(u_{n+j}, t_{n+j}) \tag{45}$$

the zeros of the characteristic polynomial

$$\rho(z) = \sum_{j=0}^{q} \alpha_j z^j \tag{46}$$

are within the unit circle, and those of modulus one are simple.

2.2.1.1 Method LMM1

Let us find the characteristic polynomial $\rho(z)$ of the method (28).

$$\alpha_0 = \frac{1}{2} \qquad \beta_0 = 0 \tag{47}$$

$$\alpha_1 = -\frac{3}{2} \qquad \beta_1 = 1 \tag{48}$$

$$\alpha_2 = 1 \qquad \beta_2 = 0 \tag{49}$$

thus the characteristic polynomial is

$$\rho(z) = z^2 - \frac{3}{2}z + \frac{1}{2} \tag{50}$$

the roots of which are,

$$z^2 - \frac{3}{2}z + \frac{1}{2} = 0 \tag{51}$$

$$z = \frac{\frac{3}{2} \pm \sqrt{\frac{9}{4} - 2}}{2} \tag{52}$$

$$= \frac{\frac{3}{2} \pm \frac{1}{2}}{2} \tag{53}$$

$$= \frac{3}{4} \pm \frac{1}{4} \tag{54}$$

thus the roots of the characteristic polynomial are

$$z_1 = 1 \tag{55}$$

$$z_2 = \frac{1}{2} \tag{56}$$

Thus the method (28) is zero-stable.

2.2.1.2 Method LMM2

Let us find the characteristic polynomial $\rho(z)$ of the method (29).

$$\alpha_0 = -2 \qquad \beta_0 = 0 \tag{57}$$

$$\alpha_1 = 1 \qquad \beta_1 = 0 \tag{58}$$

$$\alpha_2 = 1 \qquad \beta_2 = 3 \tag{59}$$

thus the characteristic polynomial is

$$\rho(z) = z^2 + z - 2 \tag{60}$$

the roots of which are

$$(z-1)(z+2) = 0 \tag{61}$$

$$z_1 = -2 \tag{62}$$

$$z_2 = 1 \tag{63}$$

Thus, because one of the roots ($z_1 = -2$) lie outside of the unit disk, the method (29) is not zero-stable.

3 Problem 3

Consider the method for solving the PDE

$$u_t + \alpha u_x = 0 \tag{64}$$

$$u_i^{n+1} = u_i^n - \alpha r(u_i^{n+1} - u_{i-1}^{n+1}) \qquad r = \frac{\Delta t}{\Delta x} \tag{65}$$

3.1 Part 1

Find the local truncation error (LTE) of the method.

3.1.1 Solution

The local truncation error of a numerical scheme is the residual that is generated by pretending the exact solution to satisfy the numerical method itself.[4] To this end, let us put the method (65) in the form of (64).

$$u_i^{n+1} - u_i^n + \alpha \frac{\Delta t}{\Delta x}(u_i^{n+1} - u_{i-1}^{n+1}) = 0 \tag{66}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \alpha \frac{u_i^{n+1} - u_{i-1}^{n+1}}{\Delta x} = 0 \tag{67}$$

and now we formulate the truncation error at (x_j, t^n) as the following,

$$\tau_i^n = \frac{u(x_i, t^{n+1}) - u(x_i, t^n)}{\Delta t} + \alpha \frac{u(x_i, t^{n+1}) - u(x_{i-1}, t^{n+1})}{\Delta x} \tag{68}$$

Let us taylor series expand the terms at the point (x_j, t^n),

$$u(x_i, t^{n+1}) = u + \Delta t u_t + \frac{\Delta t^2}{2} u_{tt} + O(\Delta t^3) \tag{69}$$

$$u(x_{i-1}, t^{n+1}) = u - \Delta x u_x + \Delta t u_t + \frac{\Delta x^2}{2} u_{xx} - \Delta x \Delta t u_{xt} + \frac{\Delta t^2}{2} u_{tt} + O(\Delta x^3) + O(\Delta x \Delta t^2) + O(\Delta x^2 \Delta t) + O(\Delta t^3) \tag{70}$$

plugging this into (68) we have

$$\tau_i^n = \frac{1}{\Delta t}\left(u + \Delta t u_t + \frac{\Delta t^2}{2} u_{tt} + O(\Delta t^3) - u\right) \tag{71}$$

$$+ \frac{\alpha}{\Delta x}\left(u + \Delta t u_t + \frac{\Delta t^2}{2} u_{tt} - u + \Delta x u_x - \Delta t u_t - \frac{\Delta x^2}{2} u_{xx} + \Delta x \Delta t u_{xt} - \frac{\Delta t^2}{2} u_{tt} + \cdots\right) \tag{72}$$

$$= u_t + \frac{\Delta t}{2} u_{tt} + \alpha\left(u_x - \frac{\Delta x}{2} u_{xx} + \Delta t u_{xt}\right) + O(\Delta t^2) + O(\Delta x^2) \tag{73}$$

$$= u_t + \alpha u_x + \frac{\Delta t}{2} u_{tt} + \alpha \frac{\Delta x}{2} u_{xx} + \alpha \Delta t u_{xt} + O(\Delta t^2) + O(\Delta x^2) \tag{74}$$

Since u satisfies the PDE $u_t + \alpha u_x = 0$, we have:

$$\tau_i^n = \frac{\Delta t}{2} u_{tt} - \frac{\alpha \Delta x}{2} u_{xx} + \alpha \Delta t u_{xt} + O(\Delta t^2) + O(\Delta x^2) \tag{75}$$

$$= O(\Delta x + \Delta t) \tag{76}$$

Thus the method is first-order accurate in both space and time.

[4] Quarteroni, Sacco, and Saleri (2006) page 605

4 Problem 4

Consider the following Initial Boundary Value Problem

$$\begin{cases} u_t + \alpha u_x = 0 \\ u(x,0) = f(x), \quad u(-0.5,t) = g(t) \end{cases} \tag{77}$$

where $x \in (-0.5, 2.5)$, $t \geq 0$, and $\alpha = 1.5$.

The exact solution is expressed in terms of $f(x)$ and $g(t)$ as

$$u_{\text{exact}}(x,t) = \begin{cases} f(x-\alpha t), & x - \alpha t > -0.5 \\ g(t - \frac{x+0.5}{\alpha}), & x - \alpha t \leq -0.5 \end{cases} \tag{78}$$

Follow the pseudocode in lecture notes to implement the 3 methods below:

- upwind method
- lax-friedrichs method
- lax-wendroff method

The numerical grid is defined with

$$\Delta x = \frac{3}{N} \qquad x_i = -0.5 + i\Delta x \qquad 0 \leq i \leq N \tag{79}$$

(77) gives a boundary condition only at $x_0 = -0.5$. In the Lax-Friedrich and Lax-Wendroff methods, we need an artificial boundary condition at $x_N = 2.5$. In this problem, we use the exact solution.

The artificial boundary condition:

$$u_N^n = u_{\text{exact}}(x_N, t_n) \tag{80}$$

Use the exact solution to calculate the error of each method.

$$E_i^n = |u_i^n - u_{\text{exact}(x_i,t_n)}| \tag{81}$$

Use the 3 methods to solve (77) with the initial condition and boundary condition given below:

$$f(x) = -\cos(\pi x) \qquad g(t) = \sin(\alpha \pi t) \qquad \alpha = 1.5 \tag{82}$$

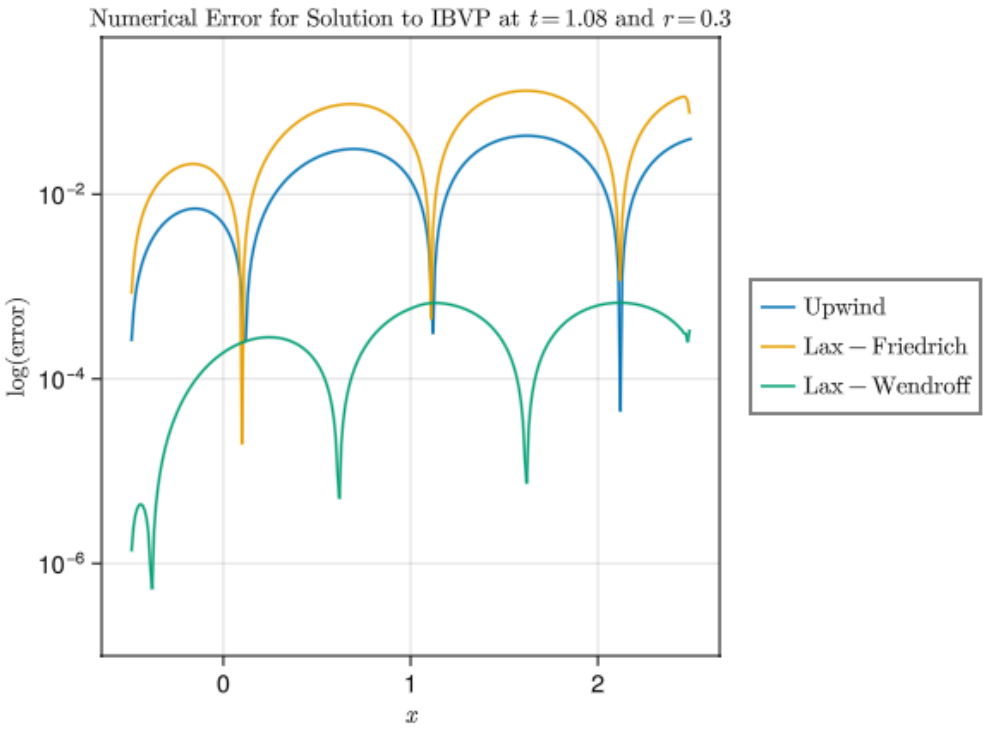
In simulations, use $N = 300$ and $\Delta t = r\Delta x$ with $r = 0.3, 0.6$.

Note: In the CFL condition r is constrained by $(ar) \leq 1$ where $\alpha = 1.5$

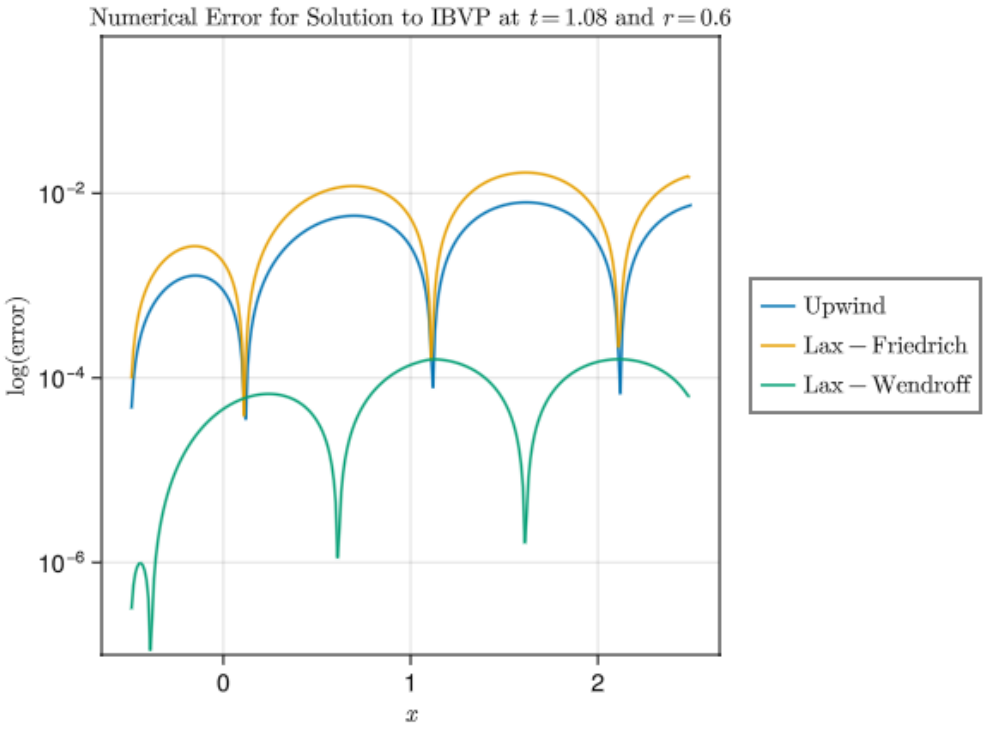
4.1 Part 1

Plot errors vs x of the 3 methods at $t = 1.08$ in one figure. Plot two figures, one for $r = 0.3$ and the other for $r = 0.6$. Use log scale for errors.

4.1.1 Solution



(a) $t = 1.08, r = 0.3$



(b) $t = 1.08, r = 0.6$

Figure 1: Plot of the log error vs x for the methods upwind (blue), lax-friedrich (yellow), and lax-wendroff (green) at the time step $t = 1.08$ for $r = 0.3$ (top) and $r = 0.6$ (bottom).

4.2 Part 2

Which method has the smallest error? Which value of r yields a smaller error?

4.2.1 Solution

The method with the smallest error is the Lax-Wendroff method. This makes sense because the method is second order accurate for space and time given that the CFL condition $|ar \leq 1|$ is met.

$r = 0.6$ produces smaller error accross all the methods (seen by a downward shift in the magnitude of the error from Figure 1a to Figure 1b).

4.3 Code

```
using GLMakie

# initial condition
f(x, t, α) = -cos(π*x)

# boundary condition
g(x, t, α) = sin(α*π*t)

# exact solution
u_exact(x, t, α) = (x-α*t) > -0.5 ? f(x-α*t, t, α) : g(x, t-(x+0.5)/α, α)

function upwind(f::Function, g::Function, u_exact::Function, params::Tuple)
    # simulation parameters
    r, α, N, x0, xf, t0, tf, Δx, Δt = params
    Nt = ceil(Int, (tf-t0)/Δt)
    # spatial and temporal grids
    xs = [x0 + i*Δx for i in 0:N]
    ts = [t0 + n*Δt for n in 0:Nt]
    # solution 2D Array (space, time)
    u = zeros(N+1, Nt+1)
    # initial condition
    u[:, 1] = f.(xs, t0, α)
    # loop in time
    for n in 1:Nt
        # enforce the left boundary condition
        u[1, n+1] = g(xs[1], ts[n+1], α)
        # loop in space
        for i in 2:N+1
            # upwind method
            u[i, n+1] = u[i, n] - α * r * (u[i, n] - u[i-1, n])
        end
    end
    return u, xs, ts
end

function lax_friedrich(f::Function, g::Function, u_exact::Function, params::Tuple)
    # simulation parameters
    r, α, N, x0, xf, t0, tf, Δx, Δt = params
    Nt = ceil(Int, (tf-t0)/Δt)
    # spatial and temporal grids
    xs = [x0 + i*Δx for i in 0:N]
    ts = [t0 + n*Δt for n in 0:Nt]
```

```
    # solution 2D Array (space, time)
    u = zeros(N+1, Nt+1)
    # initial condition
    u[:, 1] = f.(xs, t0, α)
    # loop in time
    for n in 1:Nt
        # enforce the left boundary condition
        u[1, n+1] = g(xs[1], ts[n+1], α)
        # loop in space
        for i in 2:N
            u[i, n+1] = 0.5 * (
                u[i+1, n] + u[i-1, n] -
                α*r*(u[i+1, n] - u[i-1, n])
            )
        end
        # artificial boundary condition at the right
        u[end, n+1] = u_exact(xs[end], ts[n+1], α)
    end
    return u, xs, ts
end

function lax_wendroff(f::Function, g::Function, u_exact::Function, params::Tuple)
    # simulation parameters
    r, α, N, x0, xf, t0, tf, Δx, Δt = params
    Nt = ceil(Int, (tf-t0)/Δt)
    # spatial and temporal grids
    xs = [x0 + i*Δx for i in 0:N]
    ts = [t0 + n*Δt for n in 0:Nt]
    # solution 2D Array (space, time)
    u = zeros(N+1, Nt+1)
    # initial condition
    u[:, 1] = f.(xs, t0, α)
    # loop in time
    for n in 1:Nt
        # enforce the left boundary condition
        u[1, n+1] = g(xs[1], ts[n+1], α)
        # loop in space
        for i in 2:N
            u[i, n+1] = u[i, n] +
                        0.5 * (
                            (α*r)^2 * (u[i+1, n] - 2.0*u[i, n] + u[i-1, n]) -
                            α*r * (u[i+1, n] - u[i-1, n])
                        )
        end
        # artificial boundary condition at the right
        u[end, n+1] = u_exact(xs[end], ts[n+1], α)
    end
    return u, xs, ts
end

# main driver code
function main()
    # simulation parameters
    r_values = (0.3, 0.6)
    α = 1.5
```

```
    N = 300
    x0, xf = (-0.5, 2.5)
    t0, tf = (0.0, 1.08)
    Δx = (xf - x0)/N
    # figure titles and save strings
    titles = [
        L"Numerical Error for Solution to IBVP at $t = 1.08$ and $r = 0.3$",
        L"Numerical Error for Solution to IBVP at $t = 1.08$ and $r = 0.6$"
    ]
    saves = [
        "problem4r3.png",
        "problem4r6.png"
    ]
    # loop through r values
    for i in eachindex(r_values)
        r = r_values[i]
        Δt = r*Δx
        # verify CFL conditions
        @assert α*r ≤ 1.0 "CFL condition is not met for r = $r."
        # set up figure (part1)
        fig = Figure()
        ax = Axis(
            fig[1, 1],
            title = titles[i],
            xlabel= L"x",
            ylabel= L"\log(\text{error})",
            yscale = log10,
            limits = (nothing, (1e-7, 0.5))
        )
        params = (r, α, N, x0, xf, t0, tf, Δx, Δt)
        u, xs, ts = upwind(f, g, u_exact, params)
        u_star = u_exact.(xs, ts[end], α)
        error = abs.(u[:, end] - u_star)
        lines!(ax, xs, error, label = L"\text{Upwind}")
        u, xs, ts = lax_friedrich(f, g, u_exact, params)
        u_star = u_exact.(xs, ts[end], α)
        error = abs.(u[:, end] - u_star)
        lines!(ax, xs, error, label = L"\text{Lax-Friedrich}")
        u, xs, ts = lax_wendroff(f, g, u_exact, params)
        u_star = u_exact.(xs, ts[end], α)
        error = abs.(u[:, end] - u_star)
        lines!(ax, xs, error, label = L"\text{Lax-Wendroff}")
        Legend(fig[1, 2], ax)
        save(saves[i], fig)
    end
end
main()
```

References

LeVeque, Randall J. 2007. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial; Applied Mathematics. https://doi.org/10.1137/1.9780898717839.

Quarteroni, Alfio, Riccardo Sacco, and Fausto Saleri. 2006. *Numerical Mathematics (Texts in Applied Mathematics)*. Berlin, Heidelberg: Springer-Verlag.