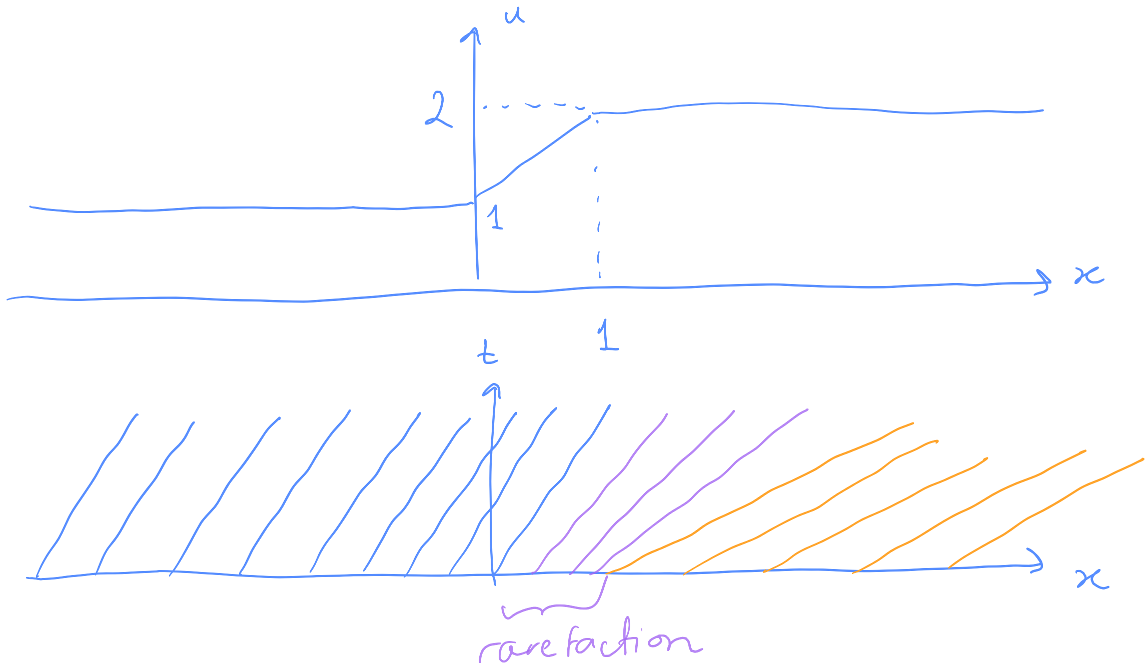
Q5)

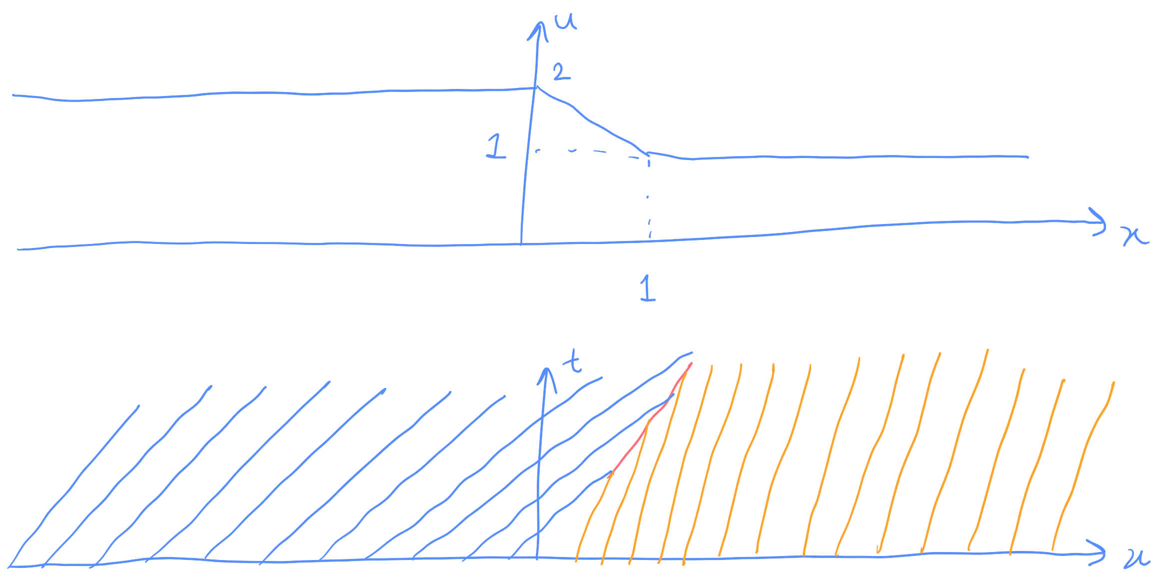
a) $$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial u^2}{\partial x} = 0$$

$$f(u) = \frac{1}{2}u^2 \qquad f'(u) = u$$

Problem 1

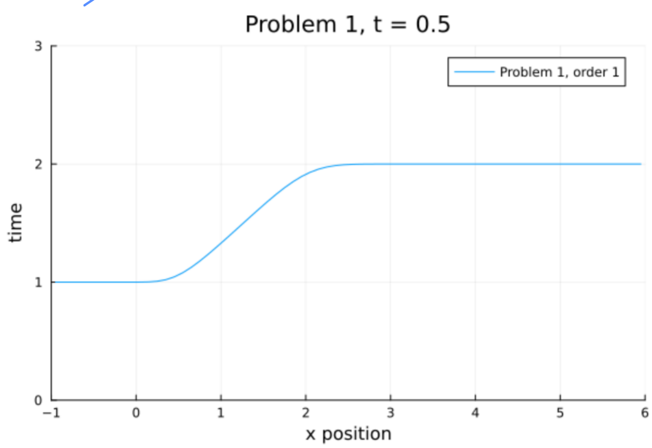


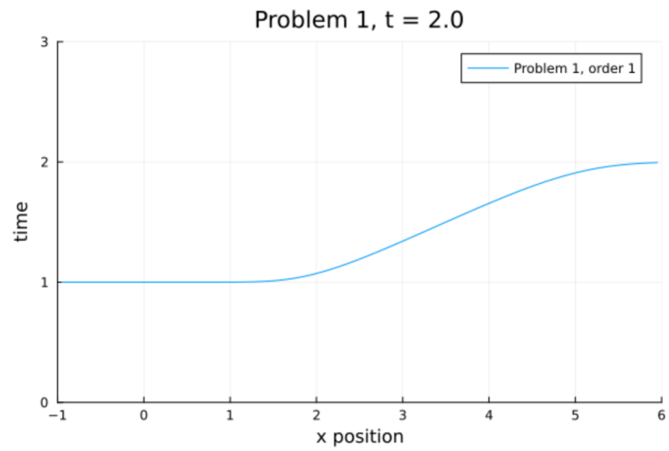
Problem 2

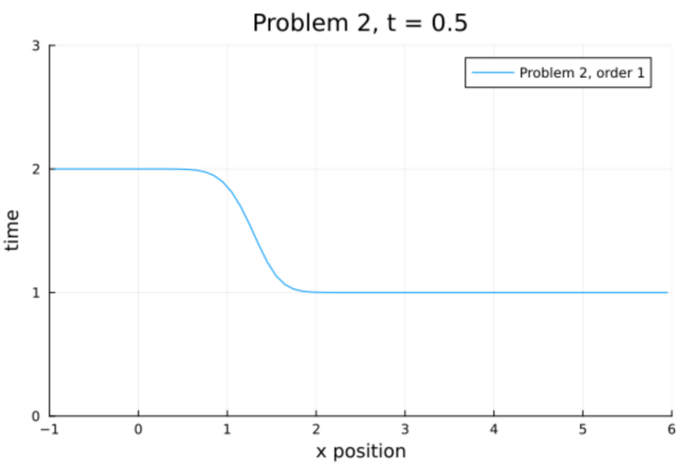


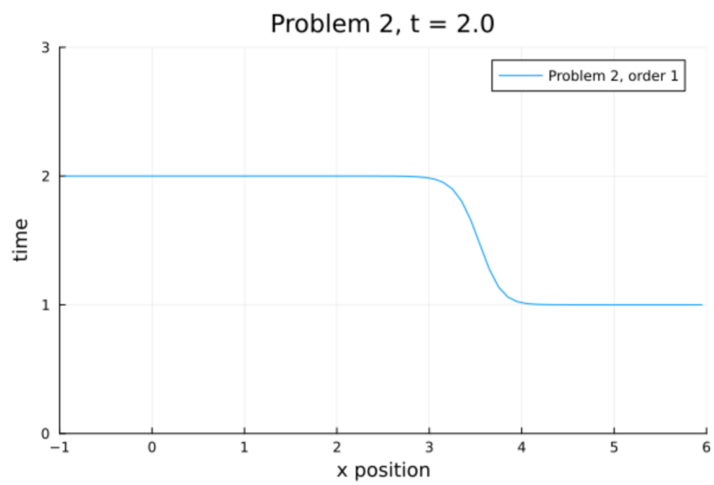
$$\frac{2 - \frac{1}{2}}{2 - 1} = \frac{3}{2} \quad \text{shock speed}$$

Q5b)

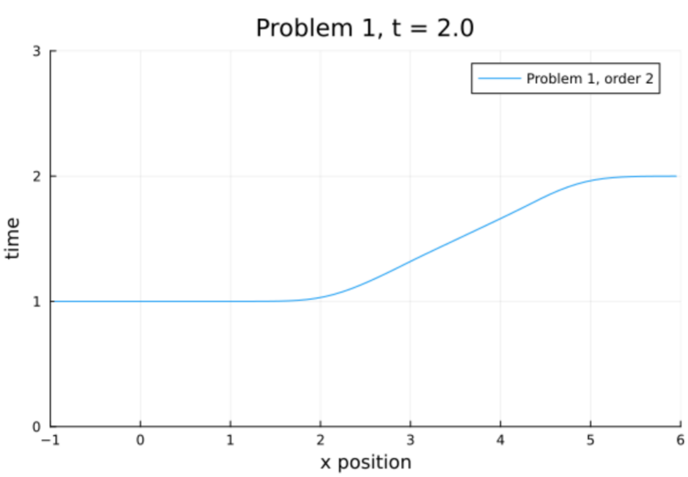

Problem 1, t = 0.5
Problem 1, order 1
time
x position

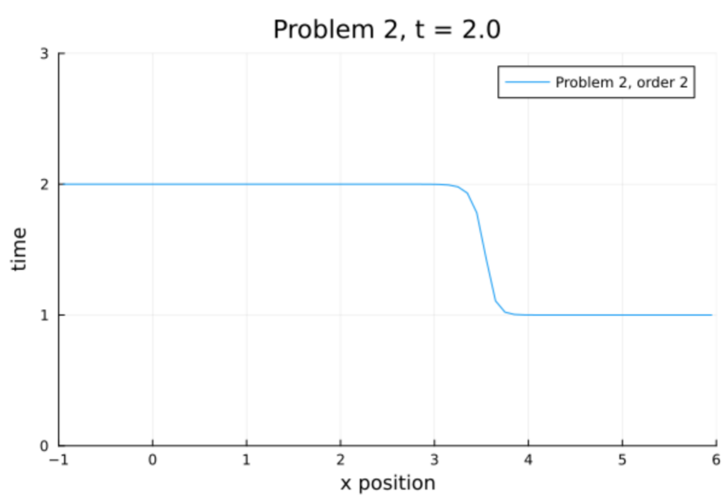

Problem 1, t = 2.0
Problem 1, order 1
time
x position


Problem 2, t = 0.5
Problem 2, order 1
time
x position

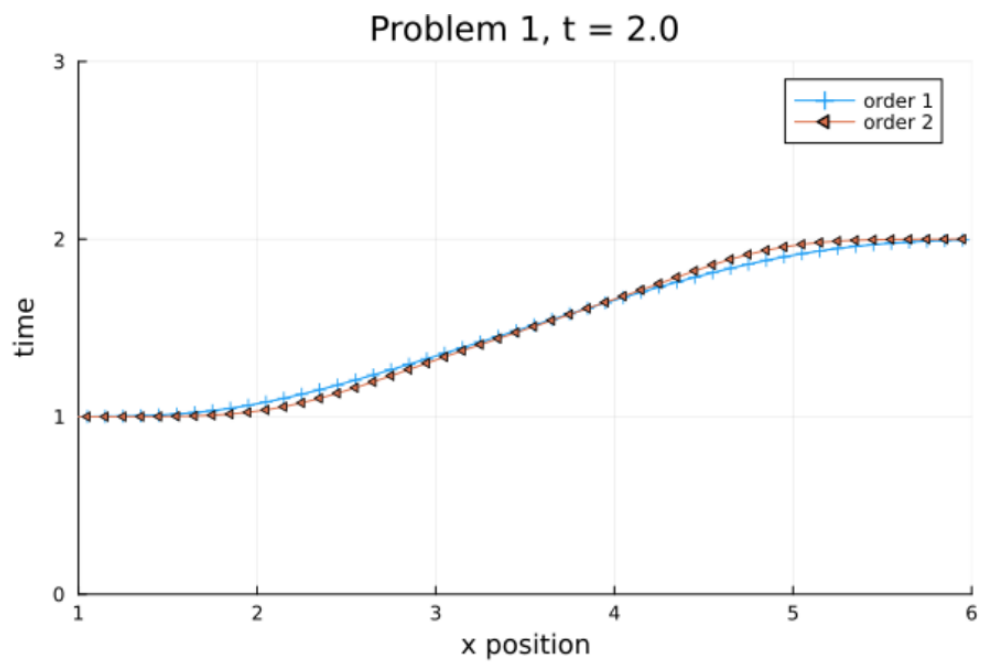

Problem 2, t = 2.0
Problem 2, order 1
time
x position

Q5c)


Problem 1, t = 2.0
Problem 1, order 2
time
x position

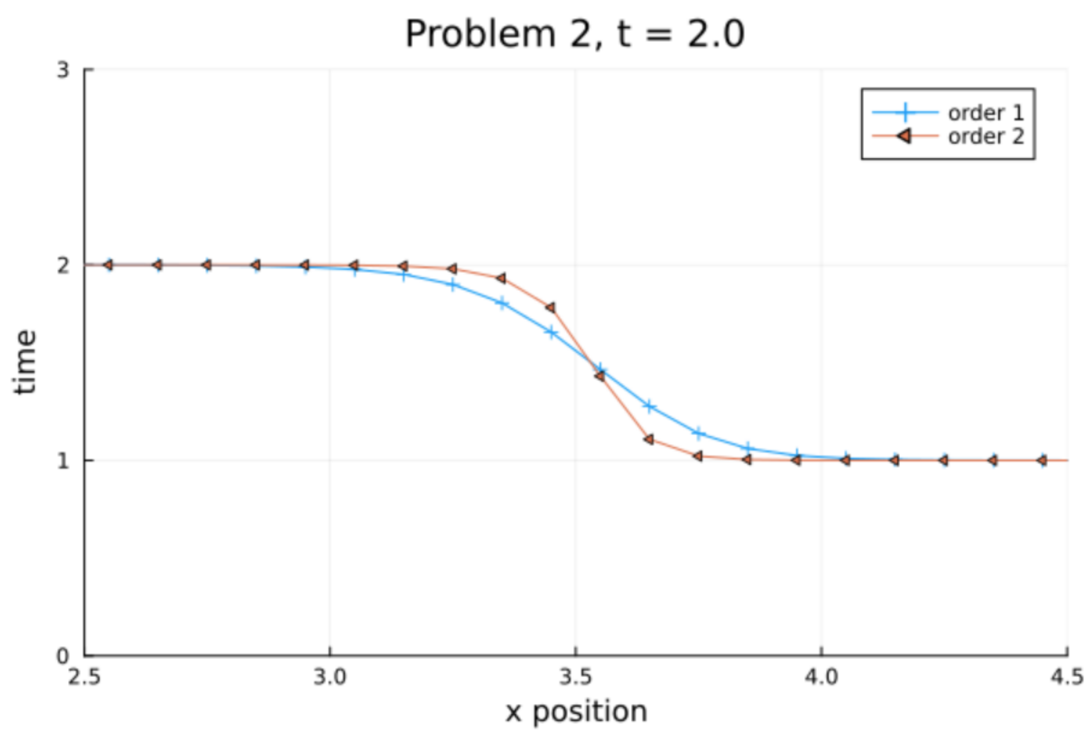

Problem 2, t = 2.0
Problem 2, order 2
time
x position

Q5d)



In this plot we see that the second order linear reconstruction has less error than the first order method. The second order method displays sharper curvature where the discontinuities should be compared to the first order method.

Q5e)



In this plot we see that the second order linear construction is better than the first order method. Strictly speaking, the minmod limiter reduces the accuracy of the second order method to first order when oscillations are detected in the vicinity of discontinuities, however, everywhere else it retains second order accuracy in space and time.

Q5 code

```julia
#===================================================================================
Question 5

Sparse Arrays using Julia: https://docs.julialang.org/en/v1/stdlib/SparseArrays/
Sparse Linear Algebra using Julia: https://docs.julialang.org/en/v1/stdlib/SuiteSparse/
===================================================================================#

using Plots, LaTeXStrings
gr()

function main(;problem=1, order=2, t_end=2)  # semi-colon to indicate keyword arguments
    N_cells = 70    # grid cells
    x_start = -1    # x lower bound      You, 3 days ago • Fist commit, A1 and A2 …
    x_end = 6       # x upper bound
    t_start = 0     # start time
    Δx = (x_end - x_start)/N_cells # Spatial discretization
    Δt = 0.5*Δx/3.  # maximum(abs.(v))
    times = t_start:Δt:t_end

    x_cell_interfaces = range(x_start, x_end, step=Δx)
    x_cell_midpoints = pairwise_average.(x_cell_interfaces[1:end-1], x_cell_interfaces[2:end])

    # Define initial condition for problem 1
    # Problem 1
    if problem == 1
        v = [xᵢ<0 ? 1. : (xᵢ<=1 ? (1+xᵢ) : 2.)  for xᵢ in x_cell_midpoints]
        # display(plot(v))
    elseif problem == 2
    # Problem 2
        # v = [xᵢ<0 ? 2. : 1.  for xᵢ in x_cell_midpoints] # To check shock speed
        # v = [xᵢ<0 ? 2. : (xᵢ<=4 ? (2-0.25*xᵢ) : 1)  for xᵢ in x_cell_midpoints] # To see shock formation
        v = [xᵢ<0 ? 2. : (xᵢ<=1 ? (2-xᵢ) : 1)  for xᵢ in x_cell_midpoints]
        # display(plot(v))
    end

    v_on_Ω = v  # collect v over the domain x -> (-1, 6) and t -> (0, 2)

    # Add ghost cells
    v = add_ghost_cells(v, problem)
    range_of_interest = 3:(length(v)-2)

    # Time marching with FV method
    anim = @animate for t = times
        if order == 1
            v = v[range_of_interest] - Δt*(LF_flux.(v[range_of_interest], v[range_of_interest.+1]) - LF_flux.(v[range_of_interest.-1], v[range_of_interest]))/Δx
            v_on_Ω = hcat(v_on_Ω, v)
            v = add_ghost_cells(v,problem)
            plot(x_cell_midpoints,v[range_of_interest], xlimits=(x_start, x_end), ylimits=(0, 3))
        elseif order == 2
            # RK2 Scheme
            v = FV_RK2(v, Δt, Δx, range_of_interest, problem)
            v_on_Ω = hcat(v_on_Ω, v[range_of_interest])
            plot(x_cell_midpoints,v[range_of_interest], xlimits=(x_start, x_end), ylimits=(0, 3))
        end
    end

    c_plot = contour(fill=true, color=:turbo,  x_cell_midpoints, vcat(times, t_end + Δt), transpose(v_on_Ω))

    gif(anim, "burgers_p$(problem)_o$(order).gif")
    savefig(c_plot, "contour_p$(problem)_o$(order).png")
    return x_cell_midpoints, v[range_of_interest]
end

# Function Definitions
pairwise_average(a,b) = (a + b)/2
flux(u) = (u^2)/2
LF_flux(u⁻, u⁺) = (flux(u⁻)+flux(u⁺))/2 - 0.5*max(abs(u⁻), abs(u⁺))*(u⁺-u⁻)
minmod(a) = max.(0.,min.(a,1.))

function add_ghost_cells(v, problem)
    if problem == 1
        v = [1;1;v;2;2]
    elseif problem == 2
        v = [2;2;v;1;1]
    end
end
```

```
function RKslope(v, Δx, range_of_interest)
    r_ll = (v[range_of_interest.-1]-v[range_of_interest.-2])./(v[range_of_interest]-v[range_of_interest.-1])
    r_ll[isnan.(r_ll)] .= 0.
    r_ll[isinf.(r_ll)] .= 0.
    v_left_of_left_cell_interface = v[range_of_interest.-1] + 0.5*minmod(r_ll).*(v[range_of_interest] - v[range_of_interest.-1])
    # v_left_of_left_cell_interface = v[range_of_interest.-1] + (v[range_of_interest.-1] - v[range_of_interest.-2])/2.      You, 32 minutes ago • Correct Q5 order 2 method

    # Avoid redundantly recomputing v at interfaces
    r_lr = vcat(r_ll[2:end], (v[range_of_interest[end]]-v[(range_of_interest.-1)[end]])./v[(range_of_interest.+1)[end]]-v[range_of_interest[end]])
    r_lr[isnan.(r_lr)] .= 0.
    r_lr[isinf.(r_lr)] .= 0.
    v_left_of_right_cell_interface = vcat(v_left_of_left_cell_interface[2:end], v[range_of_interest[end]] + 0.5*minmod(r_lr[end]).*(v[(range_of_interest.+1)[end]] - v[range_of_interest[end]]))
    # v_left_of_right_cell_interface = vcat(v_left_of_left_cell_interface[2:end], v[range_of_interest[end]] + (v[range_of_interest[end]] - v[(range_of_interest.-1)[end]])/2.)

    r_rl = (v[range_of_interest]-v[range_of_interest.-1])./(v[range_of_interest.+1]-v[range_of_interest])
    r_rl[isnan.(r_rl)] .= 0.
    r_rl[isinf.(r_rl)] .= 0.
    v_right_of_left_cell_interface = v[range_of_interest] - 0.5*minmod(r_rl).*(v[range_of_interest.+1] - v[range_of_interest])
    # v_right_of_left_cell_interface = v[range_of_interest] - (v[range_of_interest] - v[range_of_interest.-1])/2.

    # Avoid redundantly recomputing v at interfaces
    r_rr = vcat(r_rl[2:end], (v[(range_of_interest.+1)[end]]-v[range_of_interest[end]])./(v[(range_of_interest.+2)[end]]-v[(range_of_interest.+1)[end]]))
    r_rr[isnan.(r_rr)] .= 0.
    r_rr[isinf.(r_rr)] .= 0.
    v_right_of_right_cell_interface = vcat(v_right_of_left_cell_interface[2:end], v[(range_of_interest.+1)[end]] - 0.5*minmod(r_rr[end]).*(v[(range_of_interest.+2)[end]] - v[(range_of_interest.+1)[end]]))
    # v_right_of_right_cell_interface = vcat(v_right_of_left_cell_interface[2:end], v[(range_of_interest.+1)[end]] - (v[(range_of_interest.+1)[end]] - v[range_of_interest[end]]))

    Δflux = LF_flux.(v_left_of_right_cell_interface, v_right_of_right_cell_interface) - LF_flux.(v_left_of_left_cell_interface, v_right_of_left_cell_interface)
    return -Δflux/Δx
end

```

```
function FV_RK2(v, Δt, Δx, range_of_interest, problem)
    v_intermediate = v[range_of_interest] + 0.5*Δt*RKslope(v, Δx, range_of_interest)
    v_intermediate = add_ghost_cells(v_intermediate, problem)
    v_next = v[range_of_interest] + Δt*RKslope(v_intermediate, Δx, range_of_interest)
    v = add_ghost_cells(v_next, problem)
    return v
end

# Execution
_, y11_t0p5 = main(problem=1, order=1, t_end = 0.5)
_, y21_t0p5 = main(problem=2, order=1, t_end = 0.5)
x, y11 = main(problem=1, order=1)
_, y21 = main(problem=2, order=1)
_, y12 = main(problem=1, order=2)
_, y22 = main(problem=2, order=2)

```

```
Q5b1_t0p5 = plot(x, y11_t0p5, xlimits=(-1,6), ylimits=(0,3), label="Problem 1, order 1")
plot!(Q5b1_t0p5, xlabel="x position", ylabel="time", title="Problem 1, t = 0.5")
Q5b2_t0p5 = plot(x, y21_t0p5, xlimits=(-1,6), ylimits=(0,3), label="Problem 2, order 1")      You, 15 hours ago • Add plots for Q5 …
plot!(Q5b2_t0p5, xlabel="x position", ylabel="time", title="Problem 2, t = 0.5")

Q5b1_t2 = plot(x, y11, xlimits=(-1,6), ylimits=(0,3), label="Problem 1, order 1")
plot!(Q5b1_t2, xlabel="x position", ylabel="time", title="Problem 1, t = 2.0")
Q5b2_t2 = plot(x, y21, xlimits=(-1,6), ylimits=(0,3), label="Problem 2, order 1")
plot!(Q5b2_t2, xlabel="x position", ylabel="time", title="Problem 2, t = 2.0")

Q5c1 = plot(x, y12, xlimits=(-1,6), ylimits=(0,3), label="Problem 1, order 2")
plot!(Q5c1, xlabel="x position", ylabel="time", title="Problem 1, t = 2.0")
Q5c2 = plot(x, y22, xlimits=(-1,6), ylimits=(0,3), label="Problem 2, order 2")
plot!(Q5c2, xlabel="x position", ylabel="time", title="Problem 2, t = 2.0")

Q5d = plot(x, [y11, y12], xlimits=(1, 6), ylimits=(0,3), label=["order 1" "order 2"], markershape=[:cross :ltriangle])
plot!(Q5d, xlabel="x position", ylabel="time", title="Problem 1, t = 2.0")

Q5e = plot(x, [y21, y22], xlimits=(2.5, 4.5), ylimits=(0,3), label=["order 1" "order 2"], markershape=[:cross :ltriangle])
plot!(Q5e, xlabel="x position", ylabel="time", title="Problem 2, t = 2.0")


savefig(Q5b1_t0p5, "Q5b1_t0p5.png")
savefig(Q5b2_t0p5, "Q5b2_t0p5.png")
savefig(Q5b1_t2, "Q5b1_t2.png")
savefig(Q5b2_t2, "Q5b2_t2.png")
savefig(Q5c1, "Q5c1.png")
savefig(Q5c2, "Q5c2.png")

```

```
savefig(Q5d, "Q5d.png")
#=================================================================================================================
In this plot we see that the second order linear reconstruction has less error than the first order method.
The second order method displays sharper curvature where the discontinuities should be compared to the first order method.
=================================================================================================================#

savefig(Q5e, "Q5e.png")
#=================================================================================================================
In this plot we see that the second order linear construction has nearly the same slope as the first order method.
This is because the minmod limiter reduces the accuracy of the second order method to first order when oscillations due to
discontinuities are detected.
=================================================================================================================#
```