```matlab
% part 2

% properties given
gamma = 1.4 ;
R = 287 ;
T4 = 465 ;
u3 = 250 ;
n_c = 64 ;
L = 50 ;
p4 = 1*10^6 ;

ns = ns_solve(gamma, R, T4 , u3, n_c, L) ;

% extract position data for non-simple region
xx = [];
tt = [];

for i = 1:n_c
    for j = 1:n_c
        xx(i,j) = ns(i,j).x ;
        tt(i,j) = ns(i, j).t ;
    end
end

% find reflected wave simple region properties

% find uniform Jm3
a4 = sqrt(gamma*R*T4) ;
a3 = a_a4(u3, a4, gamma) ;
Jm3 = Jminus(u3, a3, gamma);

% initialize a column vector to store C+ values
ref_vec = [] ;
for i = 1:n_c
    Jp = ns(i, n_c).Jp ;
    a_x = a_Js(Jm3, Jp, gamma);
    u_x = u_Js(Jm3, Jp);
    Cp_x = slopeCp(a_x, u_x) ;
    ref_vec(i,1) = Cp_x ;
end

% now find time it takes for the head of the reflected wave to hit the wall
x_head = ns(1, n_c).x ;
t_head = ns(1,n_c).t;
t_return = t_head+ (-x_head)*ref_vec(1,1) ;

% At this time find the position of each of the other characteristics
x_final = [] ;
x_final(1,1) = 0 ;
for i = 2 : n_c
    xb = ns(i, n_c).x ;
    tb = ns(i, n_c).t ;
    sb = ref_vec(i,1) ;
    x_final(i, 1) = (t_return - tb)/sb + xb ;
end

t_final = t_return ; % common final time

% Solve a bunch of non simple regions varying n_c until a predetermined
% tolerance for change in pressure at the end wall is met

% Sounds like a while loop to me

%% ---- Converge on N using tail-only Richardson  ----

% bring variables in so I can run this stand alone
gamma = 1.4; R = 287; T4 = 465; p4 = 1e6; u3 = 250; L = 50;
a4 = sqrt(gamma*R*T4);

tol_tail  = 0.01;   % 1% max error vs Richardson asymptote on the tail
tol_end   = 0.005;  % 0.5% on the final value
tail_frac = 0.30;   % compare last 30% of the diagonal samples
tail_grid = 120;    % points for tail comparison (index space)
maxN      = 4096;   % safety cap
N0        = 16;     % starting N

% helper to get wall-pressure column vector
get_pwall = @(ns) find_p(ns, a4, p4, gamma);  % returns [t_wall, p_wall]

% helper: interpolate tail for comparison with next sample
function [s_ref, p_tail_ref, p_end] = tail_on_grid(p_wall, tail_frac, tail_grid)
    m = numel(p_wall);
    s = linspace(0,1,m).';                  % index phase s = (i-1)/(m-1)
    mask = s >= (1 - tail_frac);
    s_tail = s(mask); p_tail = p_wall(mask);
    s_ref  = linspace(1 - tail_frac, 1, tail_grid).';
    p_tail_ref = interp1(s_tail, p_tail, s_ref, 'pchip');
    p_end = p_wall(end);
end

N = N0;
while true
    if N*4 > maxN
        warning('Reached maxN=%d before convergence.', maxN);
        break
    end

    % Solve at N, 2N, 4N
    nsN   = ns_solve(gamma, R, T4, u3, N,   L);
    [~, pN]   = get_pwall(nsN);

    ns2N  = ns_solve(gamma, R, T4, u3, 2*N, L);
    [~, p2N]  = get_pwall(ns2N);

    ns4N  = ns_solve(gamma, R, T4, u3, 4*N, L);
    [~, p4N]  = get_pwall(ns4N);

    % Put the last range of samples onto a common grid
    [s_ref, pN_tail,  pN_end ]  = tail_on_grid(pN,  tail_frac, tail_grid);
    [~,     p2N_tail, p2N_end]  = tail_on_grid(p2N, tail_frac, tail_grid);
    [~,     p4N_tail, p4N_end]  = tail_on_grid(p4N, tail_frac, tail_grid);


    % p_inf ≈ p_4N + (p_4N - p_2N)/(2^1 - 1) = 2*p_4N - p_2N
    p_inf_tail = 2*p4N_tail - p2N_tail;
    p_inf_end  = 2*p4N_end  - p2N_end;

    % Estimated max error
    err_tail = max(abs(p_inf_tail - p4N_tail)) / max(abs(p_inf_tail));
    err_end  = abs(p_inf_end  - p4N_end )      / max(abs(p_inf_end));

    if err_tail < tol_tail && err_end < tol_end
        fprintf('Converged: use N = %d', 4*N);
        break
    else

    % Not converged: increase base N and try again
    N = 2*N;
    end
end
%%

% ------------ Plot control ----------------
% === Non-simple region only (decimated families) ===
% Requires: xx, tt, n_c
figure; clf; hold on;

% ---- user controls (decimation) ----
kskip      = 5;         % global stride knob
strideCp   = kskip;      % every k-th C+ family (rows)
strideCm   = kskip;      % every k-th C- family (cols)
showNodes = false;      % scatter grid nodes

% ---- colors ----
cCm   = [0.00 0.60 0.00];     % green (C- inside)
cCp   = [0.00 0.45 0.74];     % blue  (C+ inside)

lw_in = 1.2;    % line width for inside families

% ---- plot bounds from data ----
finiteMask = isfinite(xx) & isfinite(tt);
t_max = max(tt(finiteMask));
x_min = min(xx(finiteMask));
```

```
x_max = max(xx(finiteMask));

% ==========================
% INSIDE non-simple region — decimated families
%   C+ families: fix row i, connect (i,j-1)->(i,j)
%   C- families: fix col j, connect (i-1,j)->(i,j)
% ==========================

% C+ (inside): pick rows i = 1,1+strideCp,... traverse j = i+1..n_c
for i = 1:strideCp:n_c
    for j = max(i+1, 2):n_c
        if isfinite(xx(i,j)) && isfinite(tt(i,j)) && ...
           isfinite(xx(i,j-1)) && isfinite(tt(i,j-1))
            plot([xx(i,j-1) xx(i,j)], [tt(i,j-1) tt(i,j)], ...
                 'Color', cCp, 'LineWidth', lw_in);
        end
    end
end

% C- (inside): pick cols j = 1,1+strideCm,... traverse i = 2..j
for j = 1:strideCm:n_c
    for i = 2:min(j, n_c)
        if isfinite(xx(i,j)) && isfinite(tt(i,j)) && ...
           isfinite(xx(i-1,j)) && isfinite(tt(i-1,j))
            plot([xx(i-1,j) xx(i,j)], [tt(i-1,j) tt(i,j)], ...
                 'Color', cCm, 'LineWidth', lw_in);
        end
    end
end

% (optional) node markers
if showNodes
    plot(xx(finiteMask), tt(finiteMask), 'k.', 'MarkerSize', 4);
end

% axes, labels
xlim([x_min x_max]); ylim([0 1.05*t_max]);
grid on; box on;
xlabel('x (m)'); ylabel('t (s)');
title(sprintf('Non-simple region (with nc=%d, skip=%d)', n_c, kskip));

% Now plot the pressure at the end wall
figure;
[t_wall, p_wall] = find_p(ns, a4, p4, gamma);
plot(t_wall, p_wall); grid off
xlabel('t (s)'); ylabel('p_{wall} (Pa)');
title(sprintf('p(t) (with nc=%d)', n_c));

%------------- Functions -------------%
% Non simple solver
function ns = ns_solve(gamma, R, T4 , u3, n_c, L)
```

```
a4 = sqrt(gamma*R*T4) ;
a3 = a_a4(u3, a4, gamma);
Cm_head = -1/a4 ;
Cm_tail = 1/(u3-a3);
C_array = linspace(Cm_head, Cm_tail, n_c) ;


% initialize big daddy
empty = struct('x',NaN,'t',NaN,'u',NaN,'a',NaN,'Jp',NaN,'Jm',NaN);
ns_matrix = repmat(empty,n_c,n_c);
for i = 1 : n_c
    for j = 1 : n_c
        % (1,1)
        if i == 1 && j ==1
            x_a = -L ;
            t_a = C_array(1) * x_a ; % dt = dt/dx * dx, t_0 = 0
            u_a = 0 ;
            a_a = a4 ;
            Jp_a = Jplus(u_a, a_a, gamma) ;
            Jm_a = Jminus(0, a4, gamma) ;

            a.x = x_a ;
            a.u = u_a ;
            a.t = t_a ;
            a.a = a_a ;
            a.Jp = Jp_a ;
            a.Jm = Jm_a ;

            ns_matrix(i,j) = a ;

        elseif i==1 && ~(j ==1)
            %TOP ROW PROPERTIES (1, j/=1)

            s1 = slopeCp(ns_matrix(1, j-1).u, ns_matrix(1, j-1).a) ; % reflected wave C+↙
from previous entry
            x1 = ns_matrix(1, j-1).x ; %reflected wave position
            t1 = ns_matrix(1, j-1).t ;
            s2 = C_array(j) ; % incident expansion
            x2 = 0 ;
            t2 = 0;

            pos_b = pos_solve(x1, x2, t1, t2, s1, s2) ; % now I have the (x, t) of next↙
node to the right
            x_b = pos_b(1,1);
            t_b = pos_b(2,1);

            u_b = u_Cm(C_array(j), gamma, a4); % rest of properties found here
            a_b = a_a4(u_b, a4, gamma) ;
            Jp_b = Jplus(u_b, a_b, gamma) ;
            Jm_b = Jminus(u_b, a_b, gamma) ;
            b.x = x_b ;
            b.u = u_b ;
            b.t = t_b ;
            b.a = a_b ;
            b.Jp = Jp_b ;
            b.Jm = Jm_b ;
            ns_matrix(1, j) = b ;

        elseif j >= i
            % UPPER TRIANGLE
            if i == j
                % DIAGONAL
                x_d = -L ; % diagonal entries are incident with wall
                u_d = 0 ; % no motion thru wall
                Jm_d = ns_matrix(i-1,j).Jm ; % Jminus is same as point above it in matrix
                a_d = a_uJm(u_d, Jm_d, gamma);
                Jp_d = Jplus(u_d, a_d, gamma);

                % solve for time by taking intersection of C-
                % characteristic from point above and wall
                t_b = ns_matrix(i-1, j).t;
                x_b = ns_matrix(i-1, j).x;
                s_bd = slopeCm(ns_matrix(i-1, j).u, ns_matrix(i-1, j).a) ;
                t_d = t_b + (x_d - x_b)*s_bd ;

                %store d struct

                d.x = x_d ;
                d.u = u_d ;
                d.t = t_d ;
                d.a = a_d ;
                d.Jp = Jp_d ;
                d.Jm = Jm_d ;

                ns_matrix(i, j) = d ;

            else
                % OFF DIAGONAL GENERAL (e)

                Jm_e = ns_matrix(i-1,j).Jm ; % carry Jminus from above
                Jp_e = ns_matrix(i, j-1).Jp; % carry Jplus from left
                u_e = u_Js(Jm_e, Jp_e);
                a_e = a_Js(Jm_e, Jp_e, gamma);

                % now to find position, I have to average slopes using sAvg
                % function on the C+ and C- characteristics from the points
                % above and left

                % start with C+, which are points to the left
                s1 = slopeCp(ns_matrix(i, j-1).u,ns_matrix(i, j-1).a);
                s2 = slopeCp(u_e, a_e) ;
                spAvg = sAvg(s1, s2) ;
                x1 = ns_matrix(i, j-1).x ;
                t1 = ns_matrix(i, j-1).t ;

                % now minus characteristics, which are the points above
                s3 = slopeCm(ns_matrix(i-1, j).u,ns_matrix(i-1, j).a) ;
                s4 = slopeCm(u_e, a_e) ;
                smAvg = sAvg(s3, s4) ;
                x2 = ns_matrix(i-1, j).x ;
                t2 = ns_matrix(i-1, j).t ;

                % intersect those lines
                pos_e = pos_solve(x1, x2, t1, t2, spAvg, smAvg) ;
                x_e = pos_e(1,1) ;
                t_e = pos_e(2,1) ;

                %store point e
                e.x = x_e ;
                e.u = u_e ;
                e.t = t_e ;
                e.a = a_e ;
                e.Jp = Jp_e ;
                e.Jm = Jm_e ;

                ns_matrix(i, j) = e ;
            end
        end
    end
end

ns = ns_matrix ;
end

% solve system of equations

function pos = pos_solve(x1, x2, t1, t2, s1, s2)
A = [-s1, 1;
    -s2, 1] ;
b = [t1 - x1*s1;
    t2 - x2*s2] ;
pos = A \ b ; % outputs [x, t] column vector
end


% a from u in a simple region
function ai = a_a4(u, a4, gamma)
ai = a4 * (1 - (gamma-1)*0.5*(u/a4)) ;
end

% u from C- in a simple region
```

```
function u = u_Cm(Cm, gamma, a4)
u = 2/(gamma+1) * (a4 + 1/Cm) ;
end

% Define J+/- functions

function Jp = Jplus(u, a, gamma)
Jp = u + 2*a/(gamma-1);
end

function Jm = Jminus(u,a, gamma)
Jm = u - 2*a/(gamma-1);
end


% Slope functions for C+, C-, and average
function s = slopeCm(u,a), s = 1./(u - a); end   % C- line slope
function s = slopeCp(u,a), s = 1./(u + a); end   % C+ line slope

function sAvg = sAvg(s1, s2)
sAvg = tan(0.5 * (atan((s1))+ atan((s2)))) ;
end

function a_uJm = a_uJm(u, Jm, gamma)
a_uJm = (gamma-1)/2 * (u - Jm);
end


%u, a from J+, J-
function u_Js = u_Js(Jm, Jp)
u_Js = 1/2 * (Jp + Jm) ;
end

function a_Js = a_Js(Jm, Jp, gamma)
% From Jp - Jm = 4a/(γ-1) => a = (γ-1)/4 * (Jp - Jm)
a_Js = (gamma-1)/4 * (Jp - Jm);
end

% pressure from a at end wall
function p = p(c2, a, gamma)
p = c2 * a^(2*gamma/(gamma-1)) ;
end

function [t_wall, p_wall] = find_p(ns, a4, p4, gamma)
    % Isentropic constant from region 4
    C2 = p4 / (a4^(2*gamma/(gamma-1)));

    n_c    = size(ns, 1);
    t_wall = zeros(n_c,1);
    p_wall = zeros(n_c,1);
    for i = 1:n_c
        ai        = ns(i,i).a;       % sound speed at wall node
        t_wall(i) = ns(i,i).t;       % time at wall node
        p_wall(i) = C2 * ai^(2*gamma/(gamma-1));
    end

    % (optional) ensure ascending time
    [t_wall, idx] = sort(t_wall);
    p_wall = p_wall(idx);
end
```