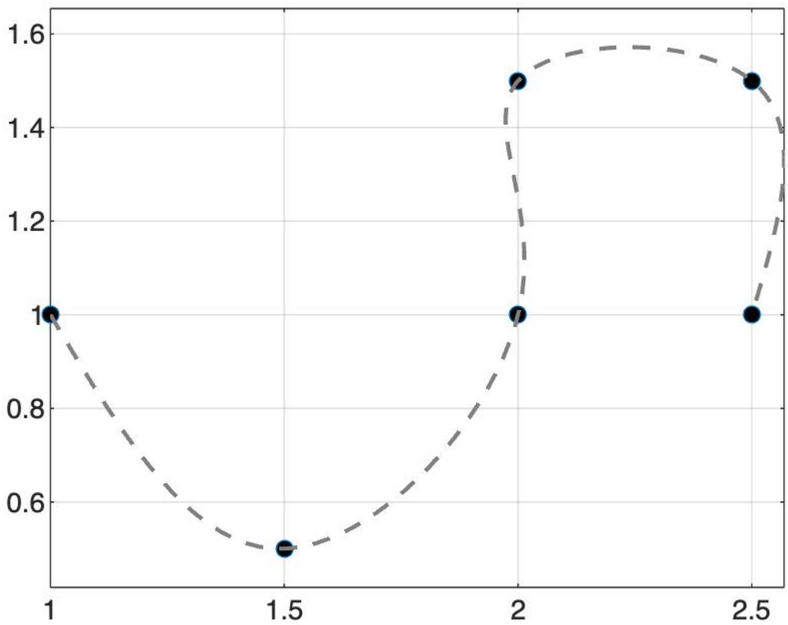
```
% Define data
t = [0, 1, 2, 3, 4, 5];
x = [1.0, 1.5, 2.0, 2.0, 2.5, 2.5];
y = [1.0, 0.5, 1.0, 1.5, 1.5, 1.0];

% Calculate the coefficients of the natural cubic spline
[b_x, c_x, d_x] = ncspline(t, x);
[b_y, c_y, d_y] = ncspline(t, y);

% Generating parameter values
xx = linspace(0, 5, 100);

% Evaluating spline functions
spline_x = splineeval(t, x, b_x, c_x, d_x, xx);
spline_y = splineeval(t, y, b_y, c_y, d_y, xx);

% plot
figure;
plot(x, y, 'o', 'MarkerSize', 6, 'MarkerFaceColor', 'k');
hold on;
plot(spline_x, spline_y, '--', 'Color', [0.5 0.5 0.5], 'LineWidth', 1.5); % spline curve
axis equal;
grid on;

% Displays the calculated coefficients
disp('Coefficients of x(t):');
disp(table(t(1:end-1)', x(1:end-1)', b_x', c_x', d_x', 'VariableNames', {'t', 'a_j', 'b_j', 'c_j', 'd_j'}))

disp('Coefficients of y(t)');
disp(table(t(1:end-1)', y(1:end-1)', b_y', c_y', d_y', 'VariableNames', {'t', 'a_j', 'b_j', 'c_j', 'd_j'}))
```

Coefficients of x(t):

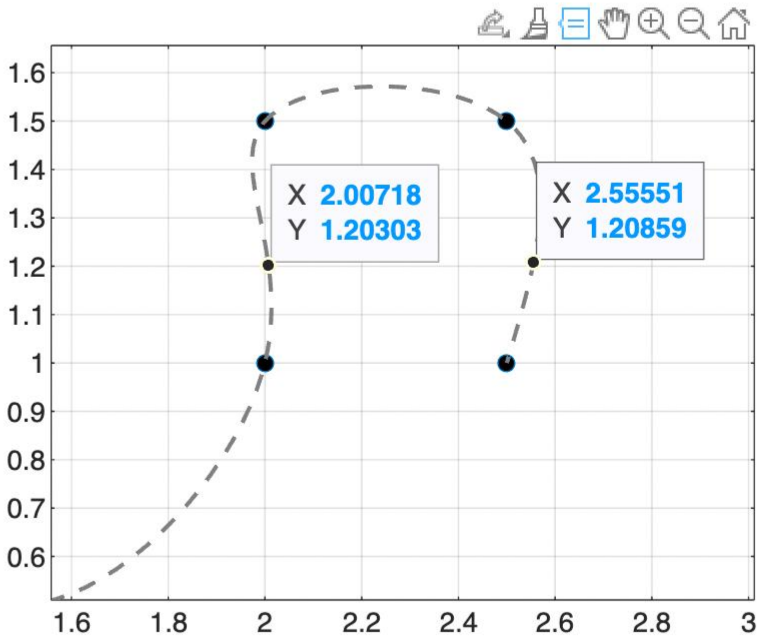
t	a_j	b_j	c_j	d_j
0	1	0.45215	0	0.047847
1	1.5	0.59569	0.14354	-0.23923
2	2	0.16507	-0.57416	0.40909
3	2	0.24402	0.65311	-0.39713
4	2.5	0.35885	-0.53828	0.17943

Coefficients of y(t)

t	a_j	b_j	c_j	d_j
0	1	-0.76077	0	0.26077
1	0.5	0.021531	0.7823	-0.30383
2	1	0.67464	-0.12919	-0.045455
3	1.5	0.2799	-0.26555	-0.014354
4	1.5	-0.29426	-0.30861	0.10287



Q2:



With y=1.2 in the picture we can tell that there are two points of intersection, one about 2.0 and the other about 2.5, so my guess is t1=2.007 and t2=2.556.

```
% Load the spline coefficients
x_vals = [0, 1, 2, 3, 4]; % Given time points

% Coefficients for y(t)
a_y = [1, 0.5, 1, 1.5, 1.5];
b_y = [-0.76077, 0.021531, 0.67464, 0.2799, -0.29426];
c_y = [0, 0.7823, -0.12919, -0.26555, -0.30861];
d_y = [0.26077, -0.30383, -0.045455, -0.014354, 0.10287];

% Define the target value
target_y = 1.2;

% Initial guess for t1 and t2 based on plot
t1 = 2.007; % Adjusted initial guess for t1
t2 = 2.556; % Adjusted initial guess for t2

% Define the function to find roots for y(t) - target_y = 0
spline_y_minus_target = @(t) splineeval(x_vals, a_y, b_y, c_y, d_y, t) - target_y;
spline_y_derivative = @(t) diffsplineeval(x_vals, a_y, b_y, c_y, d_y, t);

% Parameters for Newton's method
tol = 1e-8;
max_iter = 100;

% Newton's method for t1
for i = 1:max_iter
    f_t1 = spline_y_minus_target(t1);
    df_t1 = spline_y_derivative(t1);
    if abs(df_t1) < tol
        % If derivative is too small, slightly adjust t1
        t1 = t1 + 0.1;
        continue;
    end
    t1_next = t1 - f_t1 / df_t1;
    fprintf('Iteration %d: t1 = %.8f, f(t1) = %.8f, f''(t1) = %.8f\n', i, t1, f_t1, df_t1);
    if abs(t1_next - t1) < tol
        t1 = t1_next;
        break;
    end
```

```
    t1 = t1_next;
end

% Newton's method for t2
for i = 1:max_iter
    f_t2 = spline_y_minus_target(t2);
    df_t2 = spline_y_derivative(t2);
    if abs(df_t2) < tol
        % If derivative is too small, slightly adjust t2
        t2 = t2 + 0.1;
        continue;
    end
    t2_next = t2 - f_t2 / df_t2;
    fprintf('Iteration %d: t2 = %.8f, f(t2) = %.8f, f''(t2) = %.8f\n', i, t2, f_t2, df_t2);
    if abs(t2_next - t2) < tol
        t2 = t2_next;
        break;
    end
    t2 = t2_next;
end

% Display results with 8 significant digits
fprintf('t1 = %.8f\n', t1);
fprintf('t2 = %.8f\n', t2);
```

```
t1 = 2.31798342
t2 = 2.31798342
```

Q3:

```
>> t1 = 2.0;
t2 = 2.5;
n_values = [16, 32, 64, 128, 10000]; % Different n values

% Coefficients for x(t)
x_vals = [0, 1, 2, 3, 4]; % Given time points
a_x = [1, 1.5, 2, 2, 2.5];
b_x = [0.45215, 0.59569, 0.16507, 0.24402, 0.35885];
c_x = [0, 0.14354, -0.57416, 0.65311, -0.53828];
d_x = [0.047847, -0.23923, 0.40909, -0.39713, 0.17943];

% Coefficients for y(t)
a_y = [1, 0.5, 1, 1.5, 1.5];
b_y = [-0.76077, 0.021531, 0.67464, 0.2799, -0.29426];
c_y = [0, 0.7823, -0.12919, -0.26555, -0.30861];
d_y = [0.26077, -0.30383, -0.045455, -0.014354, 0.10287];

spline_x_derivative = @(t) diffsplineeval(x_vals, a_x, b_x, c_x, d_x, t);
spline_y_derivative = @(t) diffsplineeval(x_vals, a_y, b_y, c_y, d_y, t);

L_values = zeros(size(n_values));
h_values = zeros(size(n_values));

for k = 1:length(n_values)
    n = n_values(k);
    h = (t2 - t1) / n;
    h_values(k) = h;
    t = linspace(t1, t2, n+1);

    integrand = @(t) sqrt(spline_x_derivative(t).^2 + spline_y_derivative(t).^2);
    L = (integrand(t1) + integrand(t2)) / 2;
    for i = 2:n
        L = L + integrand(t(i));
    end
    L = L * h;
```

```
    L_values(k) = L;
end

% Highly accurate value using n = 10000
L_accurate = L_values(end);

% Compute errors
errors = abs(L_values - L_accurate);

% Plotting
loglog(h_values(1:end-1), errors(1:end-1), 'o-');
xlabel('h = (t2 - t1) / n');
ylabel('|L_n - L_{10000}|');
title('Error vs Step Size in Log-Log Plot');
grid on;

% Estimate slope
p = polyfit(log(h_values(1:end-1)), log(errors(1:end-1)), 1);
slope = p(1);
fprintf('Estimated slope: %.4f\n', slope);

% Display the results
fprintf('n\t\tL\n');
for k = 1:length(n_values)
    fprintf('%d\t\t%.8f\n', n_values(k), L_values(k));
end
Estimated slope: 1.9996
n               L
16              0.30222926
32              0.30222186
64              0.30222001
128             0.30221955
10000           0.30221939
```