```
% Define the given function x(t)
x = @(t) 10/3 .* (exp(-t./24) - exp(-t./2));

% Define y(t) as the negation of x(t)
y = @(t) -x(t);

% Call the modified threePtSearch function to compute the minimum of y(t)
% over the interval 0 <= t <= 12, with error threshold epsilon = 1e-3
interval = [0, 12];
epsilon = 1e-3;
[A1, A3] = threePtSearchModified(y, interval, epsilon);

% Compute the midpoint of A1
A2 = (A1(1) + A1(2)) / 2;

% Display the results
disp(['A1: ', num2str(A1)]);
disp(['A2: ', num2str(A2)]);
disp(['A3: ', num2str(A3)]);


% Set the initial sample points for the successive parabolic interpolation
t1 = 3;
t2 = 6;
t3 = 9;
initial_samp = [t1, t2, t3];

% Call the modified succParInt function to compute the minimum of y(t)
% using successive parabolic interpolation, with error threshold epsilon =
 1e-3
[A4, A6] = succParIntModified(y, initial_samp, epsilon);

% Compute the midpoint of A4
A5 = (A4(1) + A4(2)) / 2;

% Display the results
disp(['A4: ', num2str(A4)]);
disp(['A5: ', num2str(A5)]);
disp(['A6: ', num2str(A6)]);


% Correctly calculate the derivative of y(t)
dy_dt = @(t) (10/3) * (exp(-t/24) / 24 - exp(-t/2) / 2);

% Set the initial guess, learning rate, and Cauchy error threshold
t0 = 1;
gamma = 20; % Adjust the learning rate to a smaller value
threshold = 1e-3;

% Perform gradient descent on y(t)
A7 = gradDescent(dy_dt, t0, gamma, threshold);
```

```
% Display the result
disp(['A7: ', num2str(A7)]);
% Function threePtSearch is modified to return both the interval and the
 number of iterations
function [int, N] = threePtSearchModified(f, int, epsilon)
    a = int(1);
    b = int(2);

    x = linspace(a, b, 5);
    L = b - a;
    N = 0; % Initialize the number of iterations to zero

    while L > epsilon
        N = N + 1; % Increment the number of iterations
        y = f(x);
        min_y = min(y);
        min_ind_y = find(y == min_y);

        if length(min_ind_y) == 1
            if min_ind_y(1) == 1
                a = x(1);
                b = x(2);
            elseif min_ind_y(1) == 5
                a = x(4);
                b = x(5);
            else
                j = min_ind_y(1);
                a = x(j - 1);
                b = x(j + 1);
            end
        else
            if min(min_ind_y) == 1
                a = x(1);
                b = x(2);
            elseif max(min_ind_y) == 5
                a = x(4);
                b = x(5);
            else
                j = min(min_ind_y);
                a = x(j);
                b = x(j + 1);
            end
        end

        x = linspace(a, b, 5);
        L = b - a;
    end

    int = [a b];
end

% Function succParInt is modified to return both the interval and the number
 of iterations
function [int, N] = succParIntModified(f, samp, epsilon)
```

```
    L = samp(3) - samp(1);
    N = 0; % Initialize the number of iterations to zero

    while L > epsilon
        N = N + 1; % Increment the number of iterations
        f_samp = f(samp);
        pcoeff = polyfit(samp, f_samp, 2);
        x4 = -pcoeff(2) / (2 * pcoeff(1));

        x_new = [samp, x4];
        f_new = [f_samp, f(x4)];

        x_new(min(find(f_new == max(f_new)))) = [];
        samp = sort(x_new);
        L = samp(3) - samp(1);
    end

    int = [samp(1) samp(3)];
end

% Function gradDescent is copied from the lecture notes
function [x1] = gradDescent(df, x0, gamma, threshold)
    x1 = x0 - gamma * df(x0);
    epsilon = max(abs(x1 - x0));

    while epsilon >= threshold
        x0 = x1;
        x1 = x0 - gamma * df(x0);
        epsilon = max(abs(x1 - x0));
    end
end
```

```
A1: 5.4214      5.4221
A2: 5.4218
A3: 14
A4: 5.4215      5.4216
A5: 5.4216
A6: 8
A7: 5.4216
```

*Published with MATLAB® R2022b*