```
function [int_val, err_val] = MilneSimpson(f, a, b, epsilon)

    % Define initial values
    h_0 = ((180*epsilon)/(b - a))^0.25;
    k = floor(log2(h_0));
    h_0 = 2^k;

    % Define starting delta
    delta = (2/3)*h_0^5;

    % Prepare for iteration
    I = 0;
    E = 0;
    h = h_0;
    flag = 0;
    i = 1;
    while h ~=0

        % Compute the approximations according to Milne's rule and
        % Simpson's rule
        milne_approx = M(f, [a, a+4*h], h);
        simp_approx = S(f, [a, a+4*h], h);
        delta_MS = abs(milne_approx - simp_approx);

        % If the difference between the two approximations is less than
        % delta, save the average value and move to the next interval
        if (delta_MS <= delta)
            I = I + 0.5*(milne_approx + simp_approx);
            E = E + delta/30;
            a = a+4*h;
            h = 2*h;
            if (a + 4*h > b)
                h = (b - a)/4;
            end
            delta = 2*delta;
        % Else reduce the step size and repeat
        else
            h = h/2;
            delta = delta/2;
        end

        int_val(i) = I;
        err_val(i) = E;
        i = i + 1;

    end

end

function milne_val = M(f, interval, h)
```

```
    % Implements Milne's Rule
    milne_val = ((4*h)/3)*(2*f(interval(1) + h) - f(interval(1) + 2*h) + 2*f(interval(1)
+ 3*h));

end

function simp_val = S(f, interval, h)

    % Implements Simpson's Rule
    simp_val = ((2*h)/3)*(f(interval(1)) + 4*f(interval(1) + 2*h) + f(interval(1) +
4*h));

end
```