```
function [u, v , a  , e_abs, eta, e_cum, t, t_steps] = adapt_newmark(M,K,alpha_1,
alpha_2,p_inf,u0,v0,t_f,dt_0,nu_1,nu_2,eta_e)

    % Rayleigh damping
    C = alpha_1 * M + alpha_2 * K;

    % initial acceleration
    a0 = M \ (- C*v0 - K*u0);

    % Chung and Hulbert (1993)
    alpha_m = (2 * p_inf - 1)/(p_inf + 1);
    alpha_f = p_inf / (p_inf + 1);
    beta = 0.25 * (1 - alpha_m + alpha_f)^2;
    gamma = 0.5 - alpha_m + alpha_f;


    % create the boundary for the adpative newmark method
    lb = nu_1 * eta_e; % lower bound
    ub = nu_2 * eta_e; % upper bound


    % time stepping
    % t_f : final time
    % dt : initial time step
    % t : time vector
    % t_steps : vector of time steps (for plotting)
    t_0 = 0;
    t_current = t_0;
    dt = dt_0; % current time step
    t = t_current;
    t_steps = dt;

    % arrays for the response with the initial conditions
    u = u0;
    v = v0;
    a = a0;


    % define the container for the errors
    e_abs = 0; %absolute error
    eta = 0;  % relative error
    e_cum = 0; %cumulative error

    i = 1; % counter for the number of iterations

    while t_current <= t_f

        % effective stiffness matrix (slides pg. 63)
        K_eff = M * ((1-alpha_m)/(beta*dt^2)) + C * (gamma *(1-alpha_f)/(beta*dt)) + K *
(1 - alpha_f);
```

```
        r_eff = -K * alpha_f * u(:,i) ...
              + C * ((gamma*(1-alpha_f)/(beta * dt))* u(:,i) + ((gamma - gamma * alpha_f
- beta)/(beta))*v(:,i) + (((gamma-2*beta)*(1 - alpha_f))/(2*beta))*dt*a(:,i)) ...
              + M * (((1-alpha_m)/(beta * dt^2)) * u(:,i) + ((1-alpha_m)/(beta*dt)) * v
(:,i) + ((1-alpha_m-2*beta)/(2*beta)) * a(:,i)) ;

        % solve for u at the next time step
        %u(:,i+1) = K_eff\r_eff;
        u(:,i+1) = inv(K_eff)*r_eff;

        % update v and a -> slides pg. 60
        v(:,i+1) = (gamma/(beta * dt))*(u(:,i+1) - u(:,i)) - ((gamma - beta)/beta)*v(:,
i) - ((gamma-2*beta)/(2*beta))*dt*a(:,i);
        a(:,i+1) = (1/(beta*dt^2))*(u(:,i+1) - u(:,i)) - (1/(beta*dt)) * v(:,i) - ((1-
2*beta)/(2*beta))*a(:,i);

        %calculate our errors
        e_abs(i+1) = norm(((6 * beta - 1)/6) * (a(:,i+1) - a(:,i))*dt^2);
        eta(i+1) = (e_abs(i+1))/norm(u(:,i+1) - u(:,i));
        e_cum(i+1) = sum(e_abs);

        % check if the error is within the bounds
        if (eta(i+1) > ub || eta(i+1) < lb)
            dt = dt * sqrt(eta_e/eta(i+1));
        end
        t_current = t_current + dt;
        t(i+1) = t_current;
        t_steps(i+1) = dt;
        i = i + 1;
    end


end
```