```
%this is a reference implementation to help understand the VEDA fortran code.
%the simplifications here are
%1) there is a fixed time step which always marches forward.  in VEDA, we
%   have to accomodate an ODE solver, so sometimes we get calls out of order,
%   or get intermediate temporary points that are discarded
%2) this version is only slightly optimized for speed.  the VEDA code is highly
%   optimized for speed, which makes is fairly complicated.
%3) this version only handles one constitutive model
%4) in veda we know the indentation rate from the ODE solver.  here we
%   have to approximate with a finite difference
%5) this function has not been extensively tested, there could be some bugs
function viscoelastiticty_test

t = linspace(0,1, 300);
dt = t(2) - t(1);

alpha  = sin(pi * t);
dalpha = cdiff(alpha, dt );

R = 1;
E = 1;
eta = 0.1;

a = sqrt( R * alpha ); %contact area. pre-populate assuming the increasing case
da = cdiff( a, dt);

tm_ndx = -1;

F = zeros(length(t), 1);

res_table = zeros( length(t), 1);
bias      = zeros( length(t), 1);
num_steps = zeros( length(t), 1);

for i = 2:length(t)
    if (alpha(i) > alpha(i-1) )
        %indentation increasing.  thus contact area is increasing
        F(i) = force_i(t, a, da, i,  E, eta, R, alpha);
    else
        %indentation decreasing.  thus contact area is decreasing.
        %must find t1 by solving a convolution integral.  the most straightforward
        %way would be to compute the value of the integral for every single possible
        %value of t1 and then find the minimum.  For example, we could do this:
        %for j = 1:(i-1)
        %    res_table(j) = residual(alpha, dalpha, i, j, E, eta,t );
        %end
        %however, the calculation of this integral is very expensive.  therefore
        %we only want to calculate a handful of trial values.  we know that our
        %data will in general be smooth.  so the current t1 and the previous t1
        %are probably close, likely separated by only a few time steps.  therefore,
        %perform a linear search, starting at the previously computed t1, and
        %marching to the left or the right to minimize the residual error.

         if ( tm_ndx == -1)
             %this is the first decrease
             tm_ndx = i-1;

             %start search center two steps ago
             j = i-2;
         else
             %start search center at t1 from last point,
             j = t1_ndx(i-1);
         end

         if (j == 1)
             %can't go any further
             t1_ndx(i) = 1;
         else
             t1_ndx(i) = -1;
```

```
            res_table(j-1) = abs(residual(alpha, dalpha, i, j-1, E, eta,t ));
            res_table(j) =   abs(residual(alpha, dalpha, i, j  , E, eta,t ));
            res_table(j+1) = abs(residual(alpha, dalpha, i, j+1, E, eta,t ));
        end

        while (t1_ndx(i) == -1)
            if (( res_table(j) < res_table(j-1)) && ( res_table(j) < res_table(j+1)))
                %found a local minimum.  this is it.
                go = [];
            elseif (( res_table(j) > res_table(j-1)) && ( res_table(j) > res_table(j+1)))
                %found a local maximum.  for now assume that we go left.
                go = 'left';
            else
                %otherwise, move one step towards the smaller value
                if ( res_table(j-1) < res_table(j))
                    go = 'left';
                else
                    go = 'right';
                end
            end

            if ( strcmp(go, 'left'))
                j = j -1;
                if (j == 1)
                    %can't go any farther left. just stop here.
                    t1_ndx(i) = j;
                else
                    res_table(j-1) = abs(residual(alpha, dalpha, i, j-1 , E, eta,t ));
                end
            elseif (strcmp(go, 'right'))
                j = j + 1;
                if (j == i-1)
                    %can't go any farther right.  just stop here.
                    t1_ndx(i) = j;
                else
                    res_table(j+1) = abs(residual(alpha, dalpha, i, j+1 , E, eta,t ));
                end
            else
                t1_ndx(i) = j;
            end
        end

        %t1 has now been computed.    the rest is straight forward

        a(i) = a(t1_ndx(i));

        da(1:i) = cdiff( a(1:i), dt);

        if ( a(i) == 0)
            F(i) = 0;
        else
            F(i) = force_d(t, a, da, t1_ndx(i), i, E, eta, R);
        end

    end
end

%figure; plot( t, a, 'b', t, F, 'g', t, alpha, 'r'); legend('a', 'F','alpha')
figure; plot( alpha, F, '+-');

end

%if we have the value of t1 exactly right, the integral in here will be zero
%so any non-zero value indicates an error.
function res =  residual(alpha, dalpha, t_ndx, t1_ndx, E, eta, t )
    tau = t( t1_ndx: t_ndx);
    ps = psi(tau, t(t_ndx), E, eta);
    res =  trapz( tau, ps .* dalpha( t1_ndx:t_ndx));
end
```

```
%stress relaxation function.
function ps = psi(tau, t, E, eta)
       %three element model (for incompressible).
       E1 =  E;
       E2 =  0.5*E;
       T = eta / (E1+E2);
       ps = (E1/(E1+E2)) * ( E2 + E1 * exp((tau-t)/T));
end

%force computation when contact area is decreasing
function f = force_d(t, a, da, t1_ndx, t_ndx, E, eta, R)
   ndx = 1:t1_ndx;
   tau = t(ndx);
   f = (8 / R) * trapz(tau , psi(tau, t(t_ndx), E, eta) .* a(ndx).^2 .* da(ndx));
end

%force computation when contact area is increasing
function f = force_i(t, a, da, t_ndx, E, eta, R,alpha)
   ndx = 1:t_ndx;
   tau = t(ndx);
   f = (8 / R) * trapz(tau , psi(tau, t(t_ndx), E, eta) .* a(ndx).^2 .* da(ndx));
end
```