Supplementary Materials: Rheology of Green Plasticizer/Poly(vinyl chloride) Blends via Time–Temperature Superposition

Roya Jamarani, Hanno C. Erythropel, Daniel Burkat, James A. Nicell, Richard L. Leask and Milan Maric

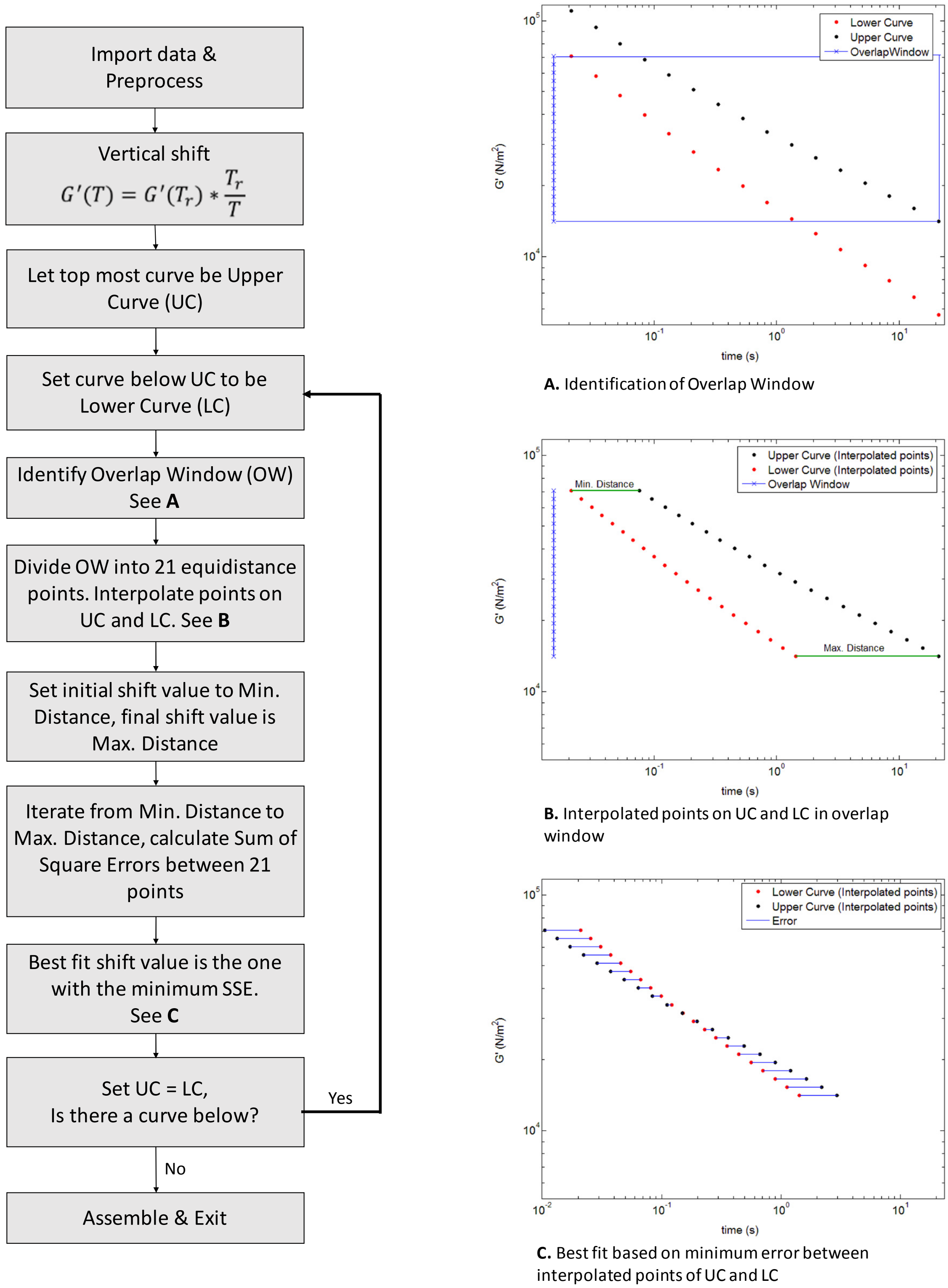

Figure S1. Time–temperature superposition algorithm, (**A**) Identification of overlap window; (**B**) Interpolated points on UC and LC in overlap window; (**C**) Best fit based on minimum error between interpolated points of UC and LC.

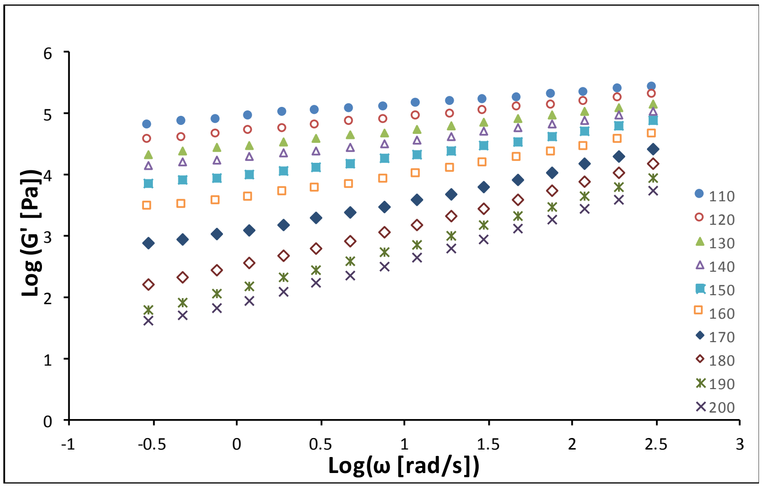

Figure S2. Dynamic frequency sweep of elastic modulus, *G*', for a 40 PHR PVC-DEHP blend from *T* = 110 °C to *T* = 200 °C.

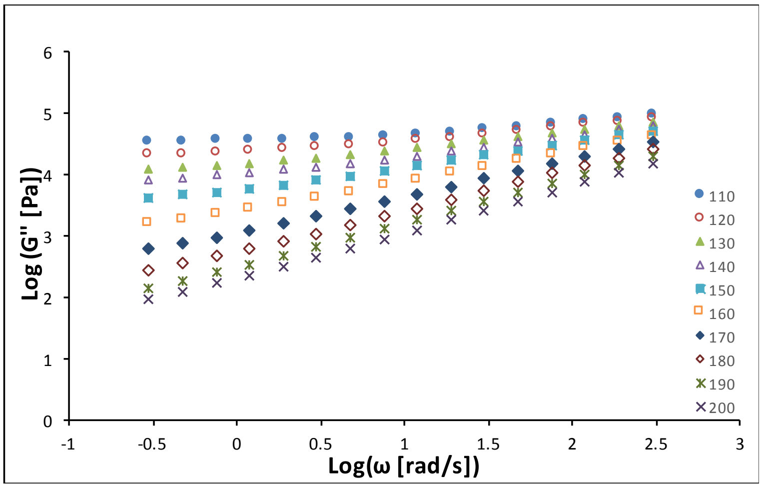

Figure S3. Dynamic frequency sweep of loss modulus, *G*'', for a 40 PHR PVC-DEHP blend from *T* = 110 °C to *T* = 200 °C.

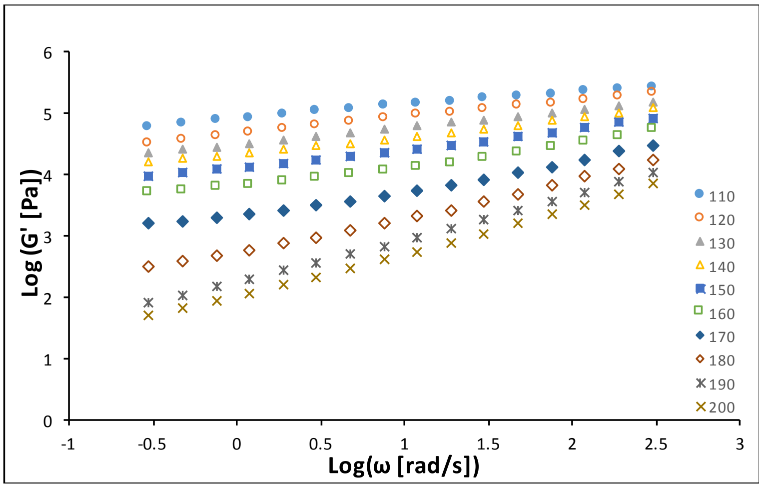

Figure S4. Dynamic frequency sweep of elastic modulus, *G*', for a 40 PHR PVC-DINCH blend from *T* = 110 °C to *T* = 200 °C.

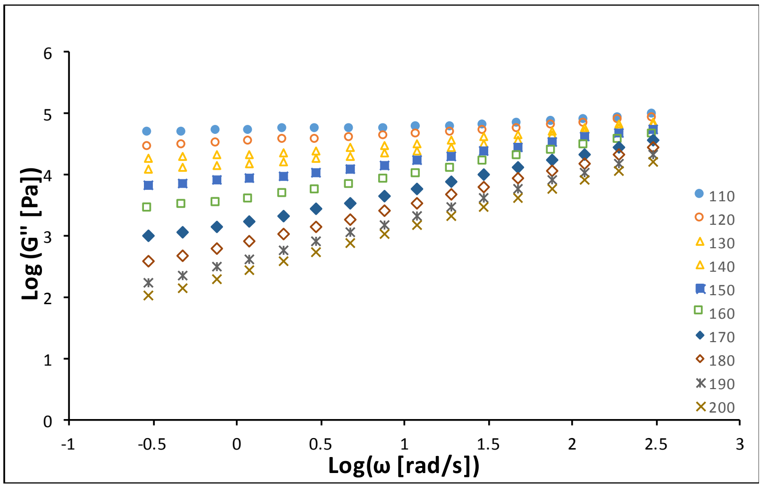

Figure S5. Dynamic frequency sweep of loss modulus, *G''*, for a 40 PHR PVC-DINCH blend from *T* = 110 °C to *T* = 200 °C.

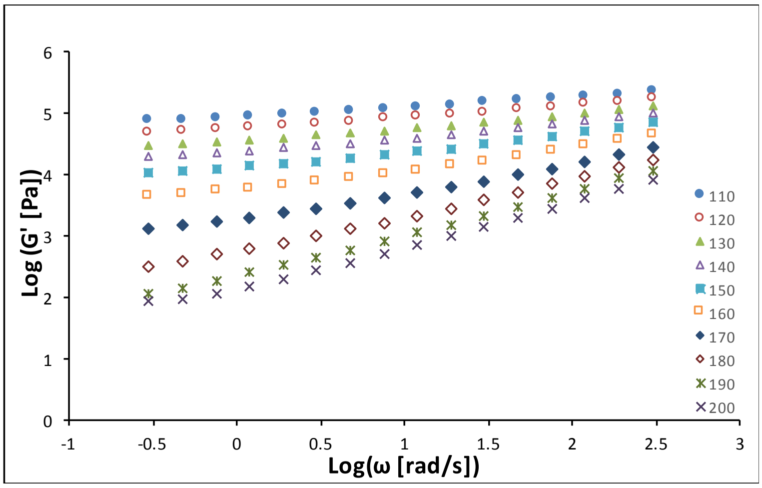

Figure S6. Dynamic frequency sweep of elastic modulus, *G'*, for a 40 PHR PVC-DOS blend from *T* = 110 °C to *T* = 200 °C.

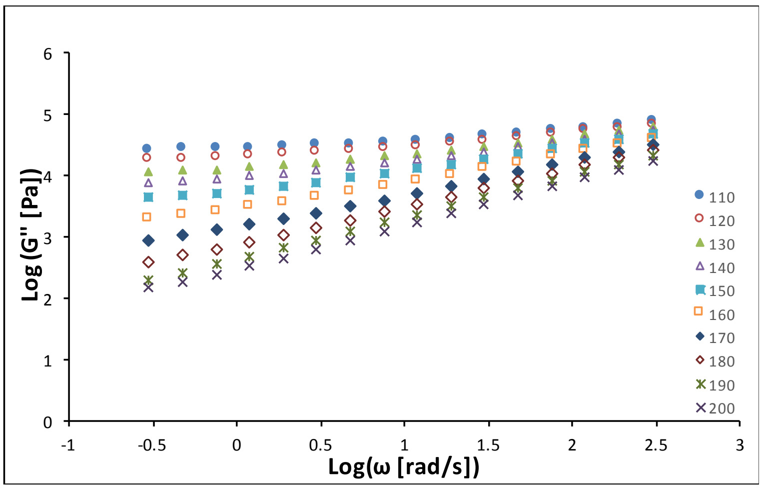

Figure S7. Dynamic frequency sweep of loss modulus, *G''*, for a 40 PHR PVC-DOS blend from *T* = 110 °C to *T* = 200 °C.

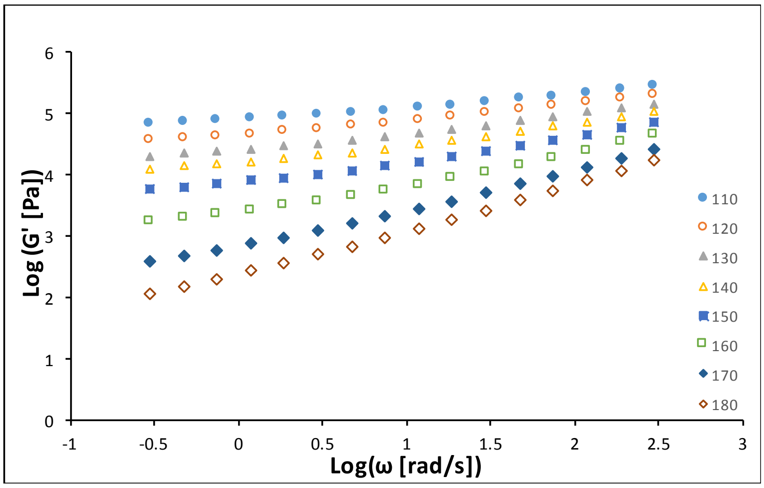

Figure S8. Dynamic frequency sweep of elastic modulus, *G*', for a 40 PHR PVC-1,4-BDDB blend from *T* = 110 °C to *T* = 180 °C.

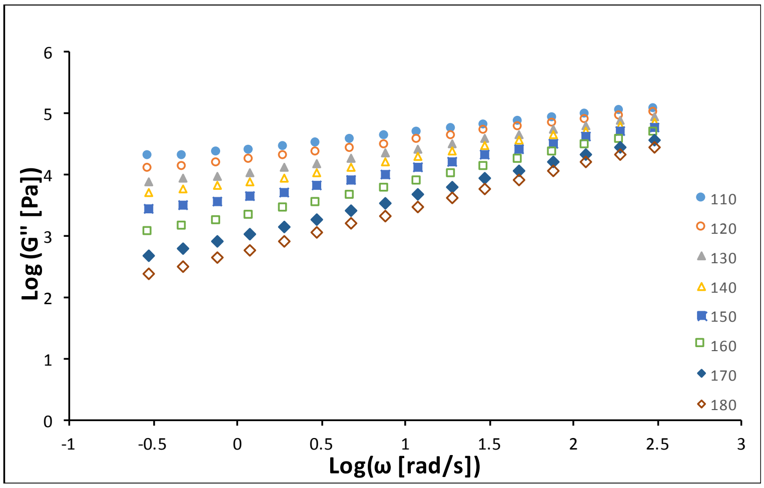

Figure S9. Dynamic frequency sweep of loss modulus, *G*'', for a 40 PHR PVC-1,4-BDDB blend from *T* = 110 °C to *T* = 180 °C.

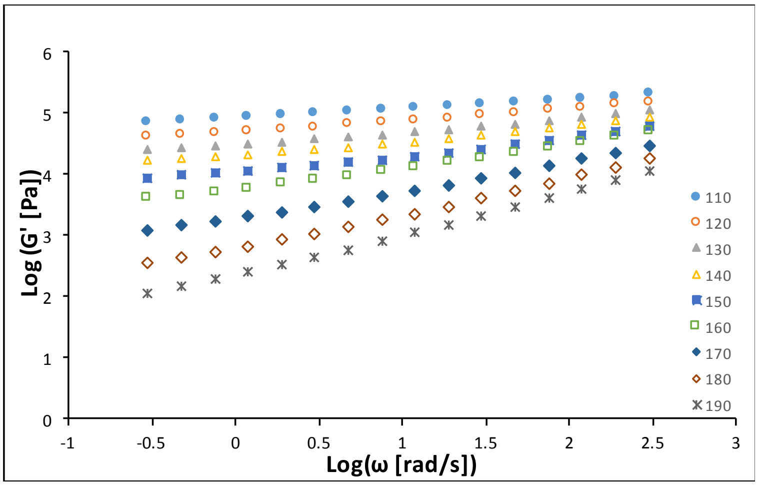

Figure S10. Dynamic frequency sweep of elastic modulus, *G*', for a 40 PHR PVC-DOM blend from *T* = 110 °C to *T* = 190 °C.

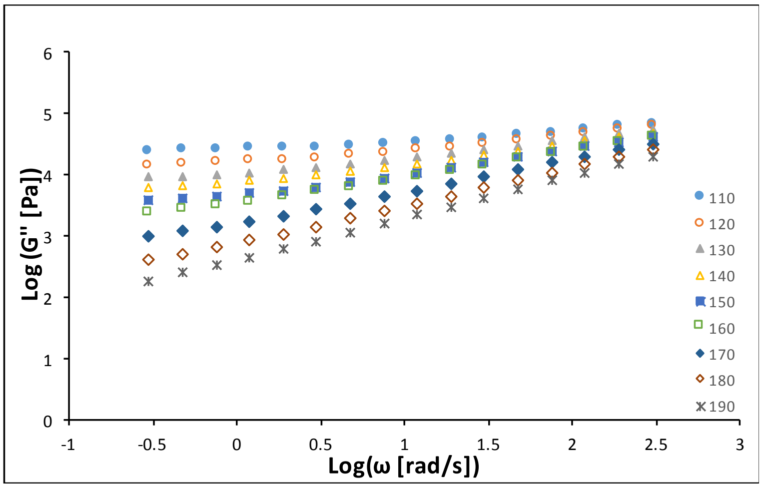

Figure S11. Dynamic frequency sweep of loss modulus, *G*'', for a 40 PHR PVC-DOM blend from T = 110 °C to T = 190 °C.

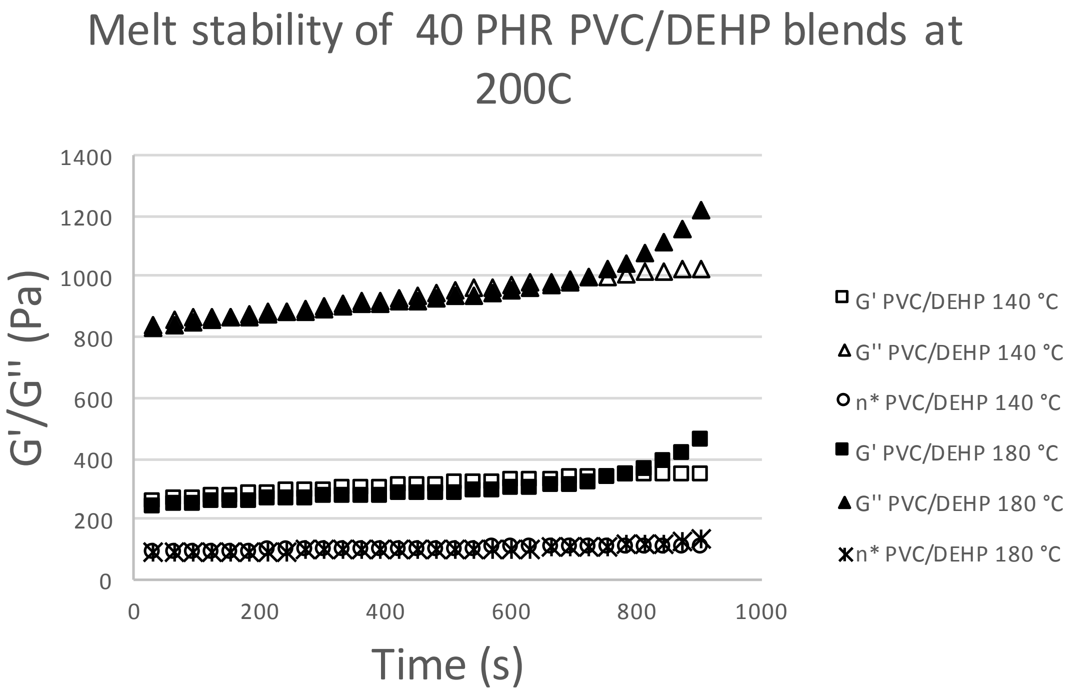

Figure S12. Melt stability of PVC/DEHP blends that were heat pressed at 140 °C and 180 °C.

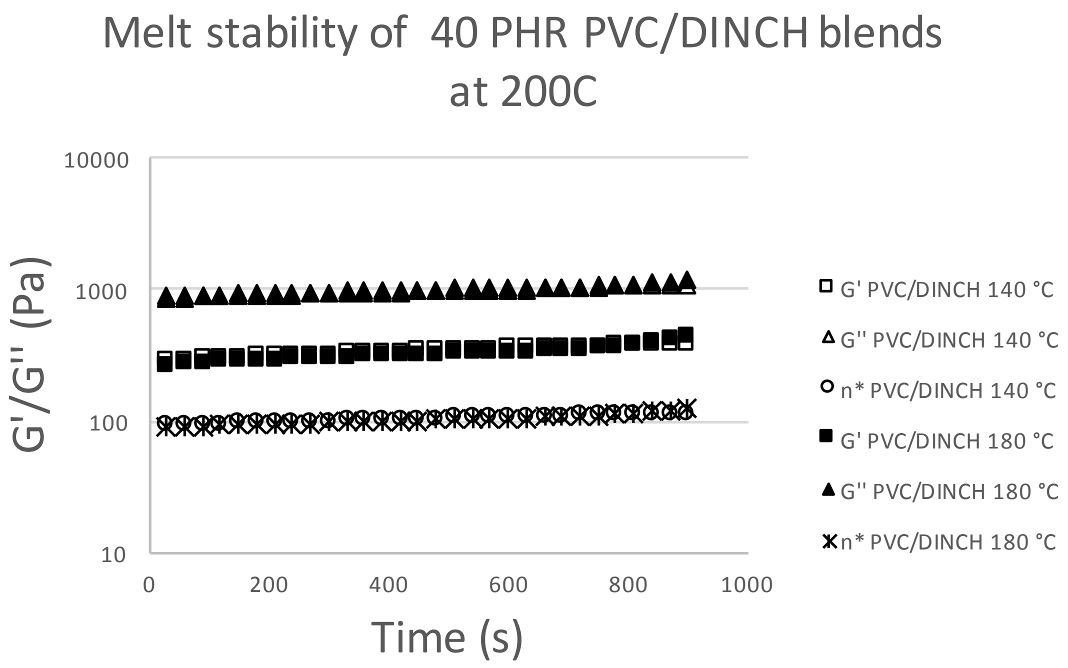

Figure S13. Melt stability of PVC/DINCH blends that were heat pressed at 140 °C and 180 °C.

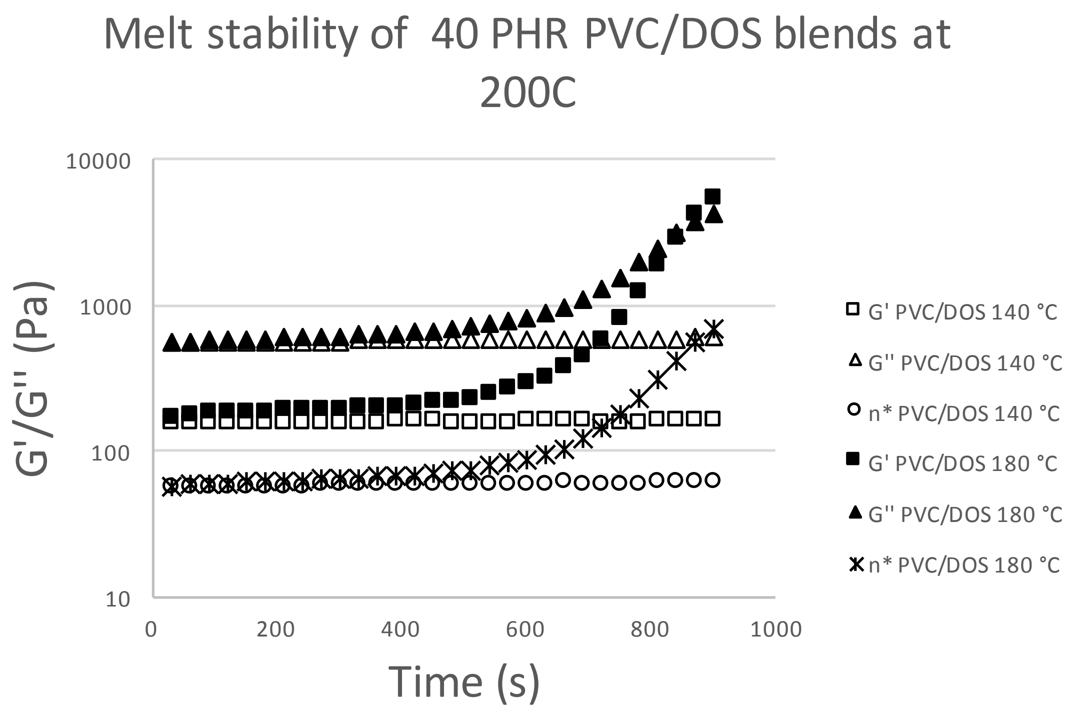

Figure S14. Melt stability of PVC/DOS blends that were heat pressed at 140 °C and 180 °C.

MATLAB code for time-temperature superposition

Main Function

```
%DATA is the matrix that contains all the raw data. It must be arranged in
%a specific order. Say the size of the DATA matrix is m rows and n column.
%Then the column 1 to n-1 contains the storage modulus or elastic modulus
%data, and column n contains the angular frequency data that is common to
%all the curves.
%COLHEAD is a 1xn cell matrix that stores the header for each column as a
%char matrix. The header for column 1 to n-1 is the temperature and the
%header for column n is the char matrix 'freq'.
%REFT is a char matrix that stores the value of the reference temperature
%that is desired.
```

```
%NAME is the char matrix that stores the name of this particular data
%series for identification purposes.
function nextDataStruct = MasterCurve_1_0(data, colhead, reft, name)

%---------------------------- setup -----------------------------------
%Program is parametrise so the row and column size of the Rheology data
%can change.
a = size(data);
row = a(1,1);
col = a(1,2);

%initialize the structure that will hold all the data linked to one
%master curve
nextDataStruct = struct('info',{}, 'header', {}, 'xraw', {}, 'yraw', {},...
    'xShifted', {}, 'yShifted', {}, 'modelStruct',{},...
    'checkMatrix', {}, 'MasterCurve', {});

%The dependent variables (storage modulus or elasticity modulus) are stored
%in the first COL-1 columns.
%The independent variable is stored in the last column of the DATA matrix
y = zeros(row,col-1);
y(:,1:col-1) = data(:,1:col-1);
x = data(:,col);

nextDataStruct(1).info = {name, reft};
nextDataStruct(1).header = colhead;
nextDataStruct(1).xraw = x;
nextDataStruct(1).yraw = y;

refCol = reft;
disp(['ref temperature is', colhead(1,refCol)]);
%---------------------------- END setup ------------------------------


%------------------ vertical and horizontal shifting --------------------
%Apply vertical shifting for the y-axis values of each curve, see
%subfunctions
yShifted = verticalShifter(y, colhead, refCol);
%run the least square error fitting algorithm, see subfunctions
xShifted = shifter(x, yShifted, col, row, refCol);

%store the shifted data into the data structure that is returned
nextDataStruct(1).yShifted = yShifted;
nextDataStruct(1).xShifted = xShifted;
nextDataStruct(1).modelStruct = getArrheniusModel(refCol, colhead, xShifted);
%----------------- END: vertical and horizontal shifting ----------------


%----------------------- trimming of curves ----------------------------
%The points are shifted, there is need to "trim" the master curve to obtian
%the final result. Store points to include in a checkMatrix, a 1 represents
%the point is appart of the Master Curve.

%The TOL is used to decided if a point is included in the trimmed master
%curve.
tol = 1/6;

ileading = col-1;
icurx = 1;
checkMatrix = zeros(row,col);

%This while loop moves along the curves and finds which points to keep so
%that the trimmed curve is smoother.
while true
    changedLeading = false;
    x1 = xShifted(icurx, ileading);
    x2 = xShifted(icurx+1, ileading);
    y1 = yShifted(icurx, ileading);
    y2 = yShifted(icurx+1, ileading);
    coords = getBracketed(x1, x2, xShifted);

    %------------------ What points to keep for the MasterCurve ----------
    d = size(coords);
    includedCoords = [];
    for k = 1:d(1,1)

        m = coords(k,1);
        n = coords(k,2);

        if (n == ileading || n == ileading-1)
            v = [xShifted(m,n) - x1; yShifted(m,n) - y1];
            s = [x2 - x1; y2 - y1];

            v2 = (dot(v,s)/dot(s,s))*s;

            diff = norm(v - v2);

            %When the below statement is true, the po
```

```
            if(diff < (tol*(x2-x1)) )
                includedCoords = [includedCoords ; coords(k,:)];
            end
        end
    end
    %------------- END: What points to keep for the MasterCurve ----------


    %Check if there is a point from ILEADING-1 to change leading lines
    %Check if point was already included in the trimmed master curve
    d = size(includedCoords);
    for k = 1:d(1,1)
        %Changes the leading curve when the below statement is true.
        if includedCoords(k,2) == ileading - 1
            ileading = ileading - 1;
            icurx = includedCoords(k,1);
            changedLeading = true;
        end

        %Adds to the master curve points that are either on ILEADING or
        %ILEADING-1. When a point from ILEADING-1 is included do not
        %include the second point from ILEADING
        %Makes sure not to include a point twice
        m = includedCoords(k,1);
        n = includedCoords(k,2);
        if (checkMatrix(m,n) == 0 && ~(changedLeading==true && k == d(1,1)))
            checkMatrix(m,n) = 1;
        end

    end

    if changedLeading == false
        icurx = icurx +1;
    end

    %Exit the loop when the last point is reached
    if(ileading == 1 && icurx == row)
        break;
    end


end %ends the big while loop

nextDataStruct(1).checkMatrix = checkMatrix;
nextDataStruct(1).MasterCurve = assembleMC(yShifted,xShifted,checkMatrix);
%--------------------- END: trimming of curves --------------------------
end
```

Subfunctions

```
%---------------------------- SHIFTER ------------------------------------
%This is the main part of the code. This function loopes over each set of
%adjacent curves and finds the values by which they need to be shifted that
%minimizes the squared error.
%X is a matrix with one column that has the common frequency data for each
%curve.
%Y is a matrix, each column of Y is the storage or elasticity modulus data
%for a distinct curve.
%CURVES is the number of curves, equals the number of columns of y.
%ROW is the number of data points for each curve.
%REFCURVE indicates which curve is the reference curve.
function xShifted = shifter(x, y, curves, row, refCurve)

%store the best k values in matrix KBEST
kBest = zeros(1, curves-2);

%Loop from the top curve to the bottom curve.
for i = 1:curves-2

    %N = 100 means that the algorithm will try 101 fits of the curve and
    %the best one will be chosen.
    n = 100;

    %UC stands for Upper Curve
    %LC stands for Lower Curve
    UCy = y(:,i);
    UCx = x(:,1);
    LCy = y(:,i+1);
    LCx = x(:,1);

    %OW stands for Overlap Window
    %Find the OW, the OWLOW is the first y value of the upper curve, OWHIGH
    %is the last y value of the lower curve.
    OWlow = UCy(1,1);
    OWhigh = LCy(row,1);
```

```
    %Cut segment into 20 intervals (so 21 points), need to find the
    %respective interpolated x values for the UC and LC
    %The interpolated values are stored in UCXINTERP and LCXINTERP
    points = 21;
    yPoints = linspace(OWlow, OWhigh, points);
    UCxInterp = zeros(points,1);
    LCxInterp = zeros(points,1);

    %Calculate the interpolated values. Subfunction getBracket() returns
    %the index of the first value to use in a 3 point Lagrange
    %interpolation. The subfunction LagrangeInterp() returns the matrix of
    %interpolated x values that match the YPOINTS.
    for j = 1:points
        u1 = getBracket(yPoints(j), UCy);
        l1 = getBracket(yPoints(j), LCy);
        UCxInterp(j,1) = LagrangeInterp(u1, yPoints(1,j), UCy, UCx);
        LCxInterp(j,1) = LagrangeInterp(l1, yPoints(1,j), LCy, LCx);
    end


    %Find an initial shift value (KSTART) and a final shift value (KLAST)
    %to avoid unnecessary calculations KSTART and KLAST
    distances = abs(UCxInterp - LCxInterp);
    kStart = min(distances) -0.1;
    kLast  = max(distances) + 0.1;
    step = (kLast - kStart)/n;
    kSSE = zeros(101,2);
    kIndex = 1;

    %Iterate over 101 trail shift values between KSTART and KLAST
    for k = kStart:step:kLast
        %Check the sum of square errors (SSE) for the K value and store it
        UCxShifted = UCxInterp + k;
        E = LCxInterp - UCxShifted;
        SSE = sum(E.^2);
        kSSE(kIndex,1) = k;
        kSSE(kIndex,2 ) = SSE;
        kIndex = kIndex + 1;
    end

    %Choose the shift value that lead to the min SSE and store it in KBEST
    [C, G] = min(kSSE(:,2));
    kBest(1,i) = kSSE(G,1);

end

%Finally shift the curves according to the reference temperature using the
%best shift values that were found.
%Start from reference temperature curve and go one direction, then in
%the other direction.

xShifted = repmat(x,1,curves-1);

%Next for loops are to displace each curve to the best position.
%shift to the right
for i = refCurve:-1:2
    k = kBest(i-1);
    shiftBy = k;
    for j = i-1:-1:1
        xShifted(:,j) = xShifted(:,j) + shiftBy;
    end
end
%shift to the left
for i = refCurve:curves-2
    k = kBest(i);
    shiftBy = k;
    for j = i+1:curves-1
        xShifted(:,j) = xShifted(:,j) - shiftBy;
    end
end

%DONE
end


%--------------------------- Vertical shifter ----------------------------
%Apply the vertical shift factor
function yshifted = verticalShifter(y, colhead, refCol)

a = size(y);
maxCol = a(1,2);
yshifted = zeros(a(1,1), a(1,2));
yshifted(:,refCol) = y(:,refCol);
Tref = str2double(colhead(1, refCol));

for k = refCol+1:maxCol
    yshifted(:,k) = log10((Tref/str2double(colhead(1,k)))) + y(:,k);
```

```
end

for k = refCol-1:-1:1
    yshifted(:,k) = log10((Tref/str2double(colhead(1,k)))) + y(:,k);
end

end


%Returns a matrix with the coordinates of points that fall within a
%bracketed region. Include the points x1 and x2 in the coords matrix.
function y = getBracketed(x1, x2, x)
[m,n] = size(x);
y = [];
for i = 1:n
    for j = 1:m
        if(x1 <= x(j,i) && x(j,i) <= x2)
            coords = [j,i];
            y = [y; coords];
        end
    end
end

end


%Returns the y value for a value of x using a 3 point Lagrange polynomial
%interpolation.
function yinterp = LagrangeInterp(i, xp, x, y)
p1 = y(i);
p2 = y(i+1);
p3 = y(i+2);
x1 = x(i);
x2 = x(i+1);
x3 = x(i+2);

p12 = p1*(xp - x2)/(x1-x2) + p2*(x1-xp)/(x1-x2);
p23 = p2*(xp-x3)/(x2-x3) + p3*(x2-xp)/(x2-x3);
yinterp = p12*(xp-x3)/(x1-x3) + p23*(x1-xp)/(x1-x3);

end


%Returns the index value of left most point to be used in the interpolation
function y1 = getBracket(xPoint, x)

%first find the two point bracket of xPoint
for index = 1:length(x)-1
    if (x(index) <= xPoint && xPoint < x(index+1) )
        y1 = index;
        break
    end
end
if xPoint == x(length(x))
    y1 = length(x)-1;
end

%determine the third point that will be used in the Lagrange interp.
if(y1+2 > length(x))
    y1 = y1-1;

end

end


%Calculate the log(a_T) value, so the amount by which the curves are
%shifted to the left and right of the curve at the reference temperature.
function logAT = getLogAT(refColumn, xShifted)
    logOmegaT = xShifted(1,:);
    logOmegaTref = xShifted(1,refColumn);
    logAT = logOmegaT - logOmegaTref;

end


%Compare the shifted data to a well accepted theoretical model to validate
%the superpostition generated
function modelStruct = getArrheniusModel(refCol, colhead, xShifted)
modelStruct = struct('info', {}, 'fitParam', {}, 'logAt', {}, 'temp', {}, 'refT', {},
'modeledLogAt',{});
logOmegaT = xShifted(1,:);
logOmegaTref = xShifted(1, refCol);
colheadSize = size(colhead);
%in the last column of this cell array there is the string for frequency,
%remove it.
colheadSize = colheadSize(1,2) - 1;
colheadTempCells = colhead(1, 1:colheadSize);
```

```
modelStruct(1).info = 'Arrhenius';
modelStruct(1).logAt = logOmegaT - logOmegaTref;
modelStruct(1).temp = str2double(colheadTempCells);
modelStruct(1).refT = str2double(colhead(1, refCol));

arrheniusModel = @(Ea, temp) (Ea/8.314).*(1./(temp + 273.15) - 1/(modelStruct(1).refT +
273.15));
Ea0 = 50;

[fitParam, resnorm] = lsqcurvefit(arrheniusModel, Ea0, modelStruct(1).temp,
modelStruct(1).logAt)

modelStruct(1).fitParam = fitParam;
modelStruct(1).modeledLogAt = arrheniusModel(fitParam, modelStruct(1).temp);

assignin('base', 'modelStruct', modelStruct);

end


%returnes an assembled MasterCurve by using the check matrix.
%this function was placed here to make code above cleaner.
function y = assembleMC(yShifted,xShifted, checkMatrix)
d = size(yShifted);
row = d(1,1);
col = d(1,2);

length = sum(sum(checkMatrix));
MasterCurve = zeros(length,2);
k = 1;

for j = 1:col
    for i = 1:row
        if(checkMatrix(i,j) == 1)
            MasterCurve(k,1) = xShifted(i,j);
            MasterCurve(k,2) = yShifted(i,j);
            k = k + 1;
        end
    end
end
y = sortrows(MasterCurve);

end
```