```
%     ____________________________________________________________
%     |   Approximate the lighting channel by a graph using LMA pulse  |
%     |   locations as vertices in the graph. Edges are generated by   |
%     |   connecting closest neighboring vertices.                     |
%     |                                                                |
%     |               Version 1.1  (April 2, 2006)                     |
%     |                                                                |
%     |                  William W. Hager                              |
%     |                  hager@math.ufl.edu                            |
%     |               http://www.math.ufl.edu/~hager                   |
%     |                Department of Mathematics                       |
%     |                  University of Florida                         |
%     |               Gainesville, Florida 32611 USA                   |
%     |                  352-392-0281 x 244                            |
%     |                                                                |
%     |               Copyright by William W. Hager                    |
%     |                                                                |
%     |This program is free software; you can redistribute it and/or   |
%     |modify it under the terms of the GNU General Public License as  |
%     |published by the Free Software Foundation; either version 2 of |
%     |the License, or (at your option) any later version.             |
%     |This program is distributed in the hope that it will be useful, |
%     |but WITHOUT ANY WARRANTY; without even the implied warranty of  |
%     |MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the   |
%     |GNU General Public License for more details.                    |
%     |                                                                |
%     |You should have received a copy of the GNU General Public       |
%     |License along with this program; if not, write to the Free      |
%     |Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, |
%     |MA  02110-1301  USA                                             |
%     |________________________________________________________________|
%
% Input:
%    LMA = rectangular matrix with 3 rows, each column gives the coordinates
%        of an LMA pulse. The coordinates should be in meters.
%
% Output:
%    EdgeList = Rectangular matrix with 2 rows, each column contains 2
%            integers which are the pulse number of two LMA pulses.
%            These two pulses should be connected by a line segment and
%            the union of these line segments approximate the lightning
%            channel.
%
% We remove pulses which are are isolated from all other pulses by a distance
```

```
% given in the parameter node_connect. We grow the channel by first
% connecting together neighboring nodes to form channel segments.
% And then channel segments are connected together when
% they are near other channel segments. The parameter channel_connect
% gives the maximum separation distance that two channel segments can have
% and be connected together.
%
% We include an option to remove dangling nodes from the channel and to remove
% dangling channel segments. Set RemoveDangle = 1 to remove the dangling
% nodes and segments. A node n1 is dangling if it is connected to only one other
% node, say n2, and this other node n2 is connected to at least 2 other nodes:
%             n1  (dangling node)
%             ^
%             |
%             v
%       n3 <--> n2 <--> n4
%
% A channel segment is considered dangling if the number of nodes in the
% channel is <= the value given in the variable DanglingChannel.

% Default parameter values:
%    node_connect    = 1000  (ignore nodes separated by 1 km from other nodes)
%    channel_connect = 2000  (connect 2 channel segments if separation <= 2 km)
%    RemoveDangle    = 1     (remove dangling nodes and channel segments)
%    DanglingChannel = 15    (any channel segment with fewer than 15 nodes is
%                            ignored when RemoveDangle = 1)

function [EdgeList] = PulseGraph (LMA)

node_connect = 1000 ;
channel_connect = 2000 ;
RemoveDangle = 0 ;
DanglingChannel = 15 ;

nLMA = size (LMA, 2) ;
fprintf (1, 'Initial number LMA pulses: %i\n', nLMA) ;
nisolated = 0 ;
LMA_ones = ones (1, nLMA) ;

% initial pass, find all the isolated nodes
for j = 1:nLMA
   R = LMA - LMA (:, j)*LMA_ones ;
   Rnorm = sum (R.^2) ;
   Rnorm (j) = inf ;
```

```
   [rmin k] = min (Rnorm) ;
   if ( rmin > node_connect^2 ) % skip this node
      LMA_ones (j) = 0 ;
      nisolated = nisolated + 1 ;
   end
end
fprintf (1, 'Number of  isolated nodes: %i\n', nisolated) ;

% remove all isolated nodes

LMA_keep = find (LMA_ones) ;
LMA = LMA (:, LMA_keep) ;
nLMA = size (LMA, 2) ;
fprintf (1, 'Number of LMA pulses (after removing isolates): %i\n', nLMA) ;
LMA_ones = ones (1, nLMA) ;
EdgeList = zeros (2, nLMA) ;
Sets = zeros (nLMA, 1) ;
Label = -ones (nLMA, 1) ; % initialize labels to -1
Next = zeros (nLMA, 1) ;
Prev = zeros (nLMA, 1) ;
nsets = 0 ;
nlabels = 0 ;
nedges = 0 ;

% pair up nodes that are adjacent to each other
for j = 1:nLMA
   if ( Label (j) == -1 )
      R = LMA - LMA (:, j)*LMA_ones ;
      Rnorm = sqrt (sum (R.^2)) ;
      Rnorm (j) = inf ;
      [rmin k] = min (Rnorm) ;
      if ( Label (k) == -1 ) % start a new set
         nsets = nsets + 1 ;
         Sets (nsets) = j ;
         Next (j) = k ;
         Prev (j) = k ;
         Next (k) = j ;
         Prev (k) = j ;
         Label (j) = nsets ;
         Label (k) = nsets ;
      else                  % add j to set Label (k) ;
         l = Prev (k) ;
         Prev (j) = l ;
         Next (l) = j ;
```

```
            Next (j) = k ;
            Prev (k) = j ;
            Label (j) = Label (k) ;
        end
        nedges = nedges + 1 ;
        EdgeList (1, nedges) = j ;
        EdgeList (2, nedges) = k ;
    end
end
fprintf (1, 'Initial number of edges: %i\n', nedges) ;
fprintf (1, 'Initial number of sets: %i\n', nsets) ;

ActiveSet = ones (nsets, 1) ;
nactive = nsets ;
AliveNodes = ones (nLMA, 1) ;
while ( nactive > 1 ) % add edges while no sets are isolated
%   fprintf (1, 'begin loop, nactive: %i\n', nactive) ;
    i = 0 ;
    while ( i <= nsets )
        i = i + 1 ;
%       fprintf (1, 'set # %i nsets: %i\n', i, nsets) ;
        if ( ActiveSet (i) ) % the set is not isolated from the rest
%           fprintf (1, '    active\n') ;
            j = Sets (i) ;
            jstart = j ;
            LMA_ones = AliveNodes ;
            while ( 1 )
                j = Next (j) ;
                LMA_ones (j) = 0 ;
                if ( j == jstart )
                    break ;
                end
            end
            Comp = find (LMA_ones) ; % find the nodes in the complement
            lenComp = length (Comp) ;
%           fprintf (1, 'size of complement: %i\n', lenComp) ;
            LMAComp = LMA (:, Comp) ;
            BestDist = inf ;
            ones_Comp = ones (1, lenComp) ;
            while ( 1 )
                j = Next (j) ;
                R = LMAComp - LMA (:, j)*ones_Comp ;
                Rnorm = sqrt (sum (R.^2)) ;
                [rmin k] = min (Rnorm) ;
```

```
        if ( rmin < BestDist )
          BestDist = rmin ;
          BestK = Comp (k) ;
          BestJ = j ;
        end
        if ( j == jstart )
          break ;
        end
      end
%         fprintf (1, 'dist to adjacent set: %e\n', BestDist) ;
      if ( BestDist <= channel_connect ) % connect two channel segments
        nactive = nactive - 1 ;
        nedges = nedges + 1 ;
        EdgeList (1, nedges) = BestJ ;
        EdgeList (2, nedges) = BestK ;
        l = Label (BestK) ;
        if ( l < nsets )    % relabel the last set as l
          m = Sets (nsets) ;
          Sets (l) = m ;
          ActiveSet (l) = ActiveSet (nsets) ;
          n = m ;
          while ( 1 )
            n = Next (n) ;
            Label (n) = l ;
            if ( n == m )
              break ;
            end
          end
        end
        nsets = nsets - 1 ;

        % update the labels
        j = BestK ;
        while ( 1 )
          j = Next (j) ;
          Label (j) = Label (BestJ) ;
          if ( j == BestK )
            break ;
          end
        end

        % update the links
        m = Prev (BestK) ;
        n = Next (BestJ) ;
```

```
            Next (BestJ) = BestK ;
            Prev (BestK) = BestJ ;
            Next (m) = n ;
            Prev (n) = m ;
         else                  % this set is separated from the rest
%             fprintf (1, '    kill\n') ;
            ActiveSet (i) = 0 ;  % this set is dead
            nactive = nactive - 1 ;
            j = jstart ;
            while ( 1 )
               j = Next (j) ;
               AliveNodes (j) = 0 ;
               if ( j == jstart )
                  break ;
               end
            end
         end
      end
   end
end

% remove dangling nodes and their associated edge

if ( RemoveDangle )
   degree = zeros (nLMA, 1) ;
   adj = zeros (nLMA, 1) ;
   edge = zeros (nLMA, 1) ;
   for j = 1:nedges
      k = EdgeList (1, j) ;
      l = EdgeList (2, j) ;
      adj (k) = l ;
      adj (l) = k ;
      edge (k) = j ;
      edge (l) = j ;
      degree (k) = degree (k) + 1 ;
      degree (l) = degree (l) + 1 ;
   end
   J = find (degree == 1) ;
   EdgeFlag = ones (nedges, 1) ;
   ndangling = 0 ;
   for j = J'
      k = adj (j) ;
      if ( degree (k) > 2 )
         degree (k) = degree (k) - 1 ;
```

```
      EdgeFlag (edge (j)) = 0 ;
      ndangling = ndangling + 1 ;
    end
end
fprintf (1, 'number of dangling nodes: %i\n', ndangling) ;
K = find (EdgeFlag) ;
EdgeList = EdgeList (:, K) ;
nedges = size (EdgeList, 2) ;

% remove dangling sets and their associated edge
ndrop = 0 ;
DropList = zeros (nsets, 1) ;
for i = 1:nsets
    j = Sets (i) ;
    jstart = j ;
    n = 0 ;
    while ( 1 )
      j = Next (j) ;
      n = n + 1 ;
      if ( j == jstart )
        break ;
      end
    end
    if ( n < DanglingChannel ) % drop the set
      ndrop = ndrop + 1 ;
      DropList (ndrop) = i ;
    end
end
fprintf (1, 'number of dangling sets: %i\n', ndrop) ;
EdgeLabels = zeros (2, nedges) ;
for k = 1:nedges
    EdgeLabel (1, k) = Label (EdgeList (1, k)) ;
    EdgeLabel (2, k) = Label (EdgeList (2, k)) ;
end
EdgeFlag = ones (nedges, 1) ;
for i = 1:ndrop
    j = DropList (i) ; % drop this set
    J = find (EdgeLabel (1, :) == j) ;
    EdgeFlag (J) = 0 ;
    J = find (EdgeLabel (2, :) == j) ;
    EdgeFlag (J) = 0 ;
end
K = find (EdgeFlag) ;
EdgeList = EdgeList (:, K) ;
```

```
   nedges = size (EdgeList, 2) ;
else
   EdgeList = EdgeList (:, 1:nedges) ;
end

fprintf (1, 'final number of edges: %i\n', nedges) ;
xmax = -inf ;
xmin =  inf ;
ymax = -inf ;
ymin =  inf ;

LMA = LMA/1000 ;
plot the channel
figure ;
hold ;
for j = 1:nedges
   k = EdgeList (1, j) ;
   l = EdgeList (2, j) ;
   x1 = LMA (1, k) ;
   xmax = max (xmax, x1) ;
   xmin = min (xmin, x1) ;
   x2 = LMA (1, l) ;
   xmax = max (xmax, x2) ;
   xmin = min (xmin, x2) ;
   X = [ x1 x2 ] ;
   y1 = LMA (2, k) ;
   ymax = max (ymax, y1) ;
   ymin = min (ymin, y1) ;
   y2 = LMA (2, l) ;
   ymax = max (ymax, y2) ;
   ymin = min (ymin, y2) ;
   Y = [ y1 y2 ] ;
   plot (X, Y) ;
end
axis equal
axis ([xmin xmax ymin ymax]) ;

hold ;

% map back to original nodes
for j = 1:nedges
   EdgeList (1, j) = LMA_keep (EdgeList (1, j)) ;
   EdgeList (2, j) = LMA_keep (EdgeList (2, j)) ;
end
```