```
function [DCdir midOrdDist midOrdX midOrdY midOrdZ chordAz] = findMidOrdDistance (chordLength,xStatic
%
% findMidOrdDistance.m - returns distance between a chord's mid ordinate
%      and a series of line segments defined by data points.
%
%   Input: search radius, chord and mid-ordinate X & Y coordinates,
%        easting, northing of a set of ordered data points
%
%   Output: distance between mid ordinate of a chord and the intersection
%         orthagonal with the chord at the mid ordinate, midOpt coords.
%
% Syntax:
%
% ***********************************************************************
% Other m-files required: dist2pts.m, midOrd.m, pts2eqn.m,
%                           perp2line.m, intsct2lines.m
%
% Subfunctions:
%
% MAT-files required: none
%
% See also: Route Location and Design, 5th ed. Thomas Hickerson, 1964
%       chapter 14: "String-Lining Railroad Curves",pp 346-355
% ***********************************************************************
% Author: Peter J Dailey, inspired by Doug Hull's (Doug.Hull@mathworks.com)
%   Matlab Video Tutorial: Intersecting a circle with a line series.
% email: daileypj@mac.com
% Doug's website posting: http://blogs.mathworks.com/videos/2008/02/19/...
%     practical-example-intersecting-a-circle-with-a-line-series/
% Last revision: 11-Aug-2009
% ***********************************************************************
%
% Copyright 2010 by Peter J Dailey
%
% Permission to use, copy, modify, distribute, and sell this software and its
% documentation for any purpose is hereby granted without fee, provided that
% the above copyright notice appear in all copies and that both that
% copyright notice and this permission notice appear in supporting
% documentation, and that the name of Peter J Dailey not be used in
% advertising or publicity pertaining to distribution of the software without
% specific, written prior permission.
% Peter J Dailey makes no representations about the
% suitability of this software for any purpose.  It is provided "as is"
% without express or implied warranty.
% ***********************************************************************

% find endpoint of chord, first is station, find second
[xChord yChord zChord] = getIntLinesCircle(chordLength,xStation,yStation,zStation,x,y,z,lastXYZIndex)

% direction of chord
chordAz = az2pts (xStation,yStation,xChord,yChord);

% middle of chord coordinates
% Calculate middle of chord coordinate given end point coordinates
dX            = abs(xStation - xChord);
dY            = abs(yStation - yChord);
dZ            = abs(zStation - zChord);
xMidOrd       = min(xStation,xChord)+(dX/2);
yMidOrd       = min(yStation,yChord)+(dY/2);
zMidOrd       = min(zStation,zChord)+(dZ/2); %elev between chord end points

% find closest point to the mid-ordinate
[first]       = closestPoint(xMidOrd,yMidOrd,zMidOrd,x,y,z);

% definition of the 2D chord line
[aChord bChord cChord] = pts2eqn (xStation,yStation,xChord,yChord);
```

```
67
68 % definition of the line orthagonal to the chord passing through the mid ordinate
69 [aPerChord bPerChord cPerChord] = perp2line (xMidOrd,yMidOrd,aChord,bChord,cChord);
70
71 %
72 for i = 1:3 % three data points define two lines closest to MidOrdinate
73    % definition of each line
74    point           = first-1;
75    xx(i)           = x(point+i); % (the point one before closest - 1)
76    yy(i)           = y(point+i);
77 end
78
79 [ a(1) b(1) c(1) ] = pts2eqn (xx(1),yy(1),xx(2),yy(2));
80 [ a(2) b(2) c(2) ] = pts2eqn (xx(2),yy(2),xx(3),yy(3));
81
82 % coordinates of the two intersections with the mid ordinate perpendicular
83 [xInt(1) yInt(1)]  = intsct2lines (a(1),b(1),c(1),aPerChord,bPerChord,cPerChord);
84 [xInt(2) yInt(2)]  = intsct2lines (a(2),b(2),c(2),aPerChord,bPerChord,cPerChord);
85
86 % distance from mid ordinate to each intersection
87 d = dist2pts (xMidOrd,yMidOrd,xInt,yInt);
88
89 % find the minimum distance to the two intersections
90 minD     = min(d);
91 % index of intersection, use outside of loop
92 foundIdx = find(d == minD);
93 % If they are the same value then find has two index entries.
94 % Use the first one.
95 if numel(foundIdx) > 1
96      index    = foundIdx(1);
97 else
98      index    = foundIdx;
99 end
100
101 % ********************************************************************
102 % This this was the original method to determine mid ordinate
103 % Assign the closest intersection to the return value
104 % Return values in midOrdX and midOrdY
105 closeMidOrdDist    = d(index);
106 midOrdX            = xInt(index);
107 midOrdY            = yInt(index);
108 % ********************************************************************
109
110 % ********************************************************************
111 % This method determines the mid ordinate from the mean
112 % of the far and near intersections.
113 % Find the coordinates between the two intersections,
114 % Return values in midOrdX and midOrdYs
115 xMidMean           = mean(xInt);
116 midOrdX            = xMidMean;
117 yMidMean           = mean(yInt);
118 midOrdY            = yMidMean;
119 meanMidOrdDist     = dist2pts(xMidOrd,yMidOrd,xMidMean,yMidMean);
120 % ********************************************************************
121 %midOrdDist         = closeMidOrdDist;
122 midOrdDist         = meanMidOrdDist;
123 % ********************************************************************
124
125 % calculate elevation at intersection
126 dist2Int           = dist2pts (x(first),y(first),xInt(index),yInt(index));
127
128 if index           == 1
129    Xpt1            = x(first-1);
130    Ypt1            = y(first-1);
131    Zpt1            = z(first-1);
132
```

```
   Xpt2            = x(first);
   Ypt2            = y(first);
   Zpt2            = z(first);
else
   Xpt1            = x(first);
   Ypt1            = y(first);
   Zpt1            = z(first);

   Xpt2            = x(first+1);
   Ypt2            = y(first+1);
   Zpt2            = z(first+1);
end
%
midOrdZ            = getStaElev(dist2Int,Xpt1,Ypt1,Zpt1,Xpt2,Ypt2,Zpt2);
% ****************************************************************
% find the direction from midordinate to intersection, (+ right) (- left)
moAz               = az2pts(xMidOrd,yMidOrd,xInt(index),yInt(index));
DCdir              = round(moAz - chordAz);
DCdirStr           = num2str(DCdir);
switch DCdirStr
     % point to the right of the midordinate
     case '90'
          direction    = -1;
     case '-270'
          direction    = -1;

     % point to the left of the midordinate
     case '-90'
          direction    = 1;
     case '270'
          direction    = 1;
end
try
midOrdDist = midOrdDist * direction; % adj result left or right
catch
     fprintf('MidOAz: %4.1f DCdir: %4.1f DCdirStr:%s\n',...
          moAz,DCdir,DCdirStr);
end
end %function

% subfunction to find the point before the closest point to the midordinate
function [first] = closestPoint(xMidOrd,yMidOrd,zMidOrd,x,y,z)
% Input: data set X & Y values; circle center x & y.
% Output: index of all points within radius
deltaX = x - xMidOrd; % all X coord - circle origin X coord, Cx
deltaY = y - yMidOrd; % all Y coord - circle origin Y coord, Cy
deltaZ = z - zMidOrd; % all Z coord - circle origin Z coord, Cz

% distance = pythagorus from center to point(s)
distance           = sqrt(deltaX.^2 + deltaY.^2 + deltaZ.^2);

% find the closest point
closest            = min(distance);
% get index of closest point
index = find(distance == closest);
% return the index before the closest point
first = index-1;
end

```