```
% Linear stationing and MidOrdinate distance calculation from absolute
% coordinate values of a GNSS track survey.
%
% 1) Read a text file with a particular data format produced by Trimble Geomatic
% Office Software.
% 2) Read a file containing mile post locations with a particular format.
% 3) Read a file containing track survey data with a particular format. Survey
% data must begin before the starting mile post and end after the closing mile
% post.
% 4) Station points along a series of sequential lines at a fixed distance
%      from the previous station. Default = 15.5 feet.
% 5) Determine the mid ordinate distance of a chord of a fixed length. Default
% chord length = 62 feet (FRA reg)
% 6) Determine the degree of curvature at the mid ordinate distance of each
% chord.
%
% Other m-files required: path2DataFile.m, readTGOoutput.m, dist2pts.m,
%   getIntLinesCircle.m, insertrows.m, findClosest.m, az2pts.m,
%   findMidOrdDistance.m
% Related m-files:
% Subfunctions:
% MAT-files required: none
% Data files required: comma delimited (.CSV) file
%
% See also:
% ************************************************************************
% Author: Peter J Dailey
% 140 Sunset Drive Charleston WV 25301
% email: daileypj@mac.com
% Website: http://www.daileyplanet.us
% Last revision: 19-Aug-2009 / 11-Feb-2010
% ************************************************************************
%
% 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. This license includes software written
% for Matlab or any other programming language.
% 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.
% ************************************************************************
%%
% Clear all variables, screen, and close any open figures. Start timer.
% clear;
clc;
close all;
tic;

% Accept defaults?
s             = input('Accept defaults? Y/[N]','s');
   if isempty(s)
      s       = 'no';
   end
defaults      = strncmpi(s, 'n', 1);

if not(defaults)
     % Default values
     stationDelta  = 15.5;
     chordLength   = 62;
     trackID       = 'default';
```

```
     smoothFactor  = 0.03;
     delay         = 0;
     dispCalc      = 1;
     logFlag       = 0;
     useDefault    = 1; % default file set
     useStats      = 0;
     alpha         = 0;
else

%% Get user input for stationing distance, default is 15.5 feet.
s                  = input('\nStation distance [15.5 feet]? ','s');
   if isempty(s)
      s            = '15.5';
   end
stationDelta       = str2double(s);

% Get user input for chord distance, default is 62 feet.
s                  = input('Chord length [62 feet]? ','s');
   if isempty(s)
      s               = '62';
   end
chordLength        = str2double(s);

 % Get user input for track number.
s                  = input('Track number or ID? ','s');
   if isempty(s)
      s               = '';
   end
trackID            = s;

% Get user input for maximum DegOfCrv.
s                  = input('Maximum expected Dc [8]? ','s');
   if isempty(s)
      s               = '8';
   end
maxDc              = str2double(s);

% Get user input for smoothing factor.
s                  = input('Smoothing coefficient [0.03]? ','s');
   if isempty(s)
      s               = '0.03';
   end
smoothFactor       = str2double(s);

% Get user input for pausing between miles, default is 0 seconds delay.
s                  = input('Display delay [0 seconds]? ','s');
   if isempty(s)
      s            = '0';
   end
delay              = str2double(s);

% Get user input for displaying calculation information, default ON.
s                  = input('Display calculations? Yes [N]o ','s');
   if isempty(s)
      s            = 'no';
      dispCalc     = 0;
   end
dispCalc       = strncmpi(s, 'yes', 1);

% Get user input for logging I/O, default is off.
s                  = input('Log Yes [N]o ? ','s');
   if isempty(s)
      s            = 'no';
      logFlag      = 0;
   end
logFlag            = strncmpi(s, 'yes', 1);
```

```
    if (logFlag)
       diary on
    end

% Read file or use default?
useDefault         = 1;
s                  = input('\nUse default survey files [N]o Yes? ','s');
   if isempty(s)
      s            = 'No';
      useDefault   = strncmpi(s, 'no', 1);
   else
      useDefault   = 1;
   end

% Calculate stats?
s                  = input('\nCalculate stats [N]o Yes? ','s');
   if isempty(s)
      s            = 'No';
   end
useStats           = strncmpi(s, 'yes', 1);

% If stats, then choose alpha, else provide default dummy to send to plotDC as a
% flag that indicates not to calc or display stats.
if useStats
     s                  = input('Alpha [0.01], change? ','s');
        if isempty(s)
           s            = '0.01';
        end
     alpha              = str2num(s);
else
     alpha              = 0;
end
%
end
% ********************************************************************
%% Initialize
noData             = NaN;
numRec             = 0;

%% Read mile post location, TGO output, of a particular format

% Read mile post file. Data file in PJD-2 format, expecting Unix end of line
% character - not Windows CR/LF
if (useDefault)
     startPath     = strcat([pwd '/']);
     windowText         = 'Pick mile post file';
     [mpFileName,filePath]= path2DataFile(startPath, windowText);
else
     mpFileName         = '495-523_mp.CSV';
     startPath          = '/Users/peteD/Documents/Engineering/Data/';
end
%% Read the mile post file
[ mpID mpfCode mpX mpY mpZ antHtMP hPrecMP vPrecMP filePath mpFileName ] =...
     readTGOoutput ( 'Mile Post Locations',filePath,mpFileName);
fprintf('\nRead mile post file: %s\n',strcat([filePath mpFileName]));

%% Extract mile post number from mile post ID in mile post data
% extract point number using a regular expression
milePostNum        = (regexp(mpID(:),'[0-9]\d*','match'));
mpNum              = str2num(char([milePostNum{:,1}]));

%% Read survey data file. Data file from TGO format PJD-1 or 2.
%    expecting Unix end of line character - not Windows CR/LF

if (useDefault)
     windowText         = 'Pick survey data file';
```

```
     [dataFileName,filePath]= path2DataFile(filePath, windowText);
else
     dataFileName       = 'mp499-500.CSV';
     filePath           = '/Users/peteD/Documents/Engineering/Data/';
end
[ ptID fCode x y z antHt hPrec vPrec filePath dataFileName ] =...
     readTGOoutput ('Survey',filePath,dataFileName);
fprintf('Read survey file: %s\n',strcat([filePath dataFileName]));

pointNum           = (regexp(ptID(:),'[0-9]\d*','match'));
ptNum              = str2num(char([pointNum{:,1}]));
%pointID            = str2num(ptID);

%% Start the waitbar
%h=waitbar(0,'Processing, please wait...');

% Step value is based on the direction of data processing,
%   increasing mile posts is positive, decreasing mile posts is negative
stepValue          = sign(mpNum(1)-mpNum(numel(mpNum)))*-1;

% Determine index of mile post within a chordLength of start of data
startFound         = false;
%% Determine the closest mile post to the first point to start, then insure
% the starting data point is within a chordLength of the mile post.
mpIdxStart         = 1;
numOfmp            = numel(mpNum);
mpIdxEnd           = numOfmp;

while not(startFound)
     [index minD]       = findClosest(mpX(mpIdxStart),mpY(mpIdxStart),x,y);
     startMP            = mpNum(mpIdxStart);
     if minD < chordLength
          startFound    = true;
     fprintf('Nearest mile post to data start is MP%i at %3.2f feet.\n',startMP,minD);
     else
          if mpIdxStart    == numel(mpX)
               mpIdxStart  = mpIdxStart - 1;
          else
          mpIdxStart       = mpIdxStart + 1;
          end
     end
end

%% Determine index of mile post within a chordLength of the end of the data
endFound           = false;
% Determine the closest mile post to the last data point, then insure
% the ending data point is within a chordLenth of the mile post.
while not(endFound)
     [index minD]     = findClosest(mpX(mpIdxEnd),mpY(mpIdxEnd),x,y);
     endMP            = mpNum(mpIdxEnd);
     if minD < chordLength
          endFound    = true;
     fprintf('Nearest mile post to data end is MP%i at %3.2f feet.\n',endMP,minD);
     else
          fprintf('No data near end, MP%i at %3.2f feet.\n',endMP,minD);
          mpIdxEnd       = mpIdxEnd - 1;
     end
end
%%
% Length of data, in mile post numbers from start to end of data
mp2mp              = abs(mpNum(mpIdxStart)-mpNum(mpIdxEnd));
fprintf('Data length from MP%3.0f to MP%3.0f is %3.0f miles.\n',startMP,endMP,mp2mp);

%% Make a dataset of nearset point to each mile post
closestFound       = false;
ptIdx = 1;
```

```
for i = mpIdxStart:mpIdxEnd
     [index minD]     = findClosest(mpX(i),mpY(i),x,y);
     nearest(i,1)     = mpNum(i);     % the mile post number
     nearest(i,2)     = index;        % the index in the x,y dataset
     nearest(i,3)     = minD;         % the distance to the MP
     nearest(i,4)     = ptNum(index); % the data point number
     fprintf('Nearest point to MP%4.0f is point %6.0f PtID %6.0f at %3.2f feet.\n',...
        nearest(i,1),nearest(i,2),nearest(i,4),nearest(i,3));
end
%% Open output file for mile or mile series
% fileNameRoot       = regexp(dataFileName,'[^.CSV]','match');
% fileNameRoot       = strcat(fileNameRoot{:});
outFile            = strcat([num2str(startMP) '-' num2str(endMP) '_' num2str(chordLength) '.CSV']);
outFileName        = strcat([filePath outFile]);
% Create new file for writing. Discard existing contents, if any.
fid                = fopen (outFileName,'w');
% Entry heading
fprintf(fid,'mp,dir,offset,length,mpRef,profLft62ft,profRt62ft,alignLeft,alignRight,dc\n');
%% Process the data between mile posts
for i = startMP:stepValue:(endMP-stepValue)

%% open output file for single mile
% fileNameRoot       = regexp(dataFileName,'[^.CSV]','match');
% fileNameRoot       = strcat(fileNameRoot{:});
outFile            = strcat([num2str(startMP) '-' num2str(endMP) '_' num2str(chordLength) '.CSV']);
outFileName        = strcat([filePath outFile]);
% Open file for writing. Append data to the end of the file.
fid                = fopen (outFileName,'a');
%
%% Initial conditions for mile
     % waitbar(i/numMiles,h);drawnow
   fprintf('\nBegining to work on MP%3.0f to MP%3.0f.\n',i,i+stepValue);
   gapCount        = 0;
   numRecThisMile  = 0;
   go2NextMile     = false; % reset skip flag
   mpRefMO         = i;
   mpRefBegin      = i;
   mpRefEnd        = mpNum(find(mpNum==i+stepValue));
   mpIdxBegin      = find(mpNum==i);
   mpIdxEnd        = find(mpNum == i + stepValue);

% find XY of survey point nearest MP
   [datStartIdx minD]    = findClosest(mpX(mpIdxBegin),mpY(mpIdxBegin),x,y);
% determine the direction of travel from the first survey data points
if datStartIdx     == 1;
   azStart             = az2pts (x(datStartIdx),y(datStartIdx),x(datStartIdx+1),y(datStartIdx+1));
else
   azStart             = az2pts (x(datStartIdx-1),y(datStartIdx-1),x(datStartIdx),y(datStartIdx));
end
   tol                 = 5; %degrees
% determine if closest data point to the mile post is after the mile post
   DOTcheck            = inDoT(mpX(mpIdxBegin),...
    mpY(mpIdxBegin),x(datStartIdx),y(datStartIdx),azStart,tol);
% If the direction of travel to the data point is 'behind' the MP, then increase
% the datStartIdx until the first point is in 'front' of the MP.
   while DOTcheck  == false
    datStartIdx = datStartIdx + 1;
    DOTcheck   = inDoT(mpX(mpIdxBegin),mpY(mpIdxBegin),x(datStartIdx),y(datStartIdx),azStart,tol);
   end
% find data point nearest next MP (end of this mile)
   [datEndIdx endD]     = findClosest(mpX(mpIdxEnd),mpY(mpIdxEnd),x,y);
% determine the direction of travel from the second closest survey data point at the
% next mile post to the closest survey data point
   azEnd               = az2pts (x(datEndIdx-1),y(datEndIdx-1),x(datEndIdx),y(datEndIdx));
   tol                 = 5; %degrees
% determine if closest data point to the mile post is after the mile post
```

```
   DOTcheck            = inDoT(x(datEndIdx),y(datEndIdx),mpX(mpIdxEnd),mpY(mpIdxEnd),azEnd,tol);
% If the direction of travel to the data point is 'behind' the MP, then increase
% the datStartIdx until the first point is in 'front' of the MP.
   while DOTcheck  == false
     datEndIdx     = datEndIdx - 1;
     DOTcheck      = inDoT(x(datEndIdx),y(datEndIdx),mpX(mpIdxEnd),mpY(mpIdxEnd),azEnd,tol);
   end

% ****************************************************************
%% Build the data set for this mile in the form:
% 1. before MP point > 2. startMP > 3. survey data > 4. endMP > 5. after MP point
% Use XYZ coordinats: easting, northing, elevation
clear tempA tempB
% Point before the begining mile post, if it exists
if datStartIdx       == 1
     datStartIdx     = 2;
else
     tempA(1,:)      = [x(datStartIdx-1) y(datStartIdx-1) z(datStartIdx-1)];
end
% Begining mile post
     tempA(2,:)      = [mpX(mpIdxBegin) mpY(mpIdxBegin) mpZ(mpIdxBegin)];
% Ending mile post
     tempA(3,:)      = [mpX(mpIdxEnd) mpY(mpIdxEnd) mpZ(mpIdxEnd)];

% Add a half chordLength of point(s) after the ending mile post to allow
%  for crossing the mile post boundry.
extraData            = false;
datCount             = 0;
while not(extraData)
     datCount                = datCount + 1;
     endDatIdx               = datEndIdx+datCount;
     afterMP                 = dist2pts(mpX(mpIdxEnd),mpY(mpIdxEnd),mpZ(mpIdxEnd),...
          x(endDatIdx),y(endDatIdx),z(endDatIdx));
     if afterMP < chordLength
          tempA(3+datCount,:)= [x(endDatIdx) y(endDatIdx) z(endDatIdx)];
     else
          extraData          = true; % got enough quit
     end
end
%% Build a temporary array of all data between the start and end MP.
%  Account for mile posts in reverse direction of travel order when
%   compared with survey data direction of travel.
if datStartIdx < datEndIdx
     tempB          = [x(datStartIdx:datEndIdx) y(datStartIdx:datEndIdx) z(datStartIdx:datEndIdx)];
else
     tempB          = [x(datEndIdx:datStartIdx) y(datEndIdx:datStartIdx) z(datEndIdx:datStartIdx)];
end
% Insert the tempArray between rows 2 and 3 of the tempA
     mileDat = insertrows(tempA,tempB,2);
% ****************************************************************

% calculate slope distance between surveyed points
distance           = sqrt(diff(mileDat(:,1)).^2 + diff(mileDat(:,2)).^2 + diff(mileDat(:,3)).^2);
meanDist           = mean(distance);

% Determine the sum of the distances between mile posts -
%   from the 2nd entry in the mileData index value of the mile data
%   that matches X coordinates with the mile post listing.
cumDistPts         = cumsum(distance(2:numel(distance))); % linear measure from first point
mileLength         = cumDistPts(numel(cumDistPts));
% calculate mile length by summing the differences between mile posts
   fprintf('MP%3.0f to MP%3.0f is %7.2f feet, mean D = %5.1f.\n',...
        i,i+stepValue,mileLength,meanDist);
   pause(delay);

% Set initial station location for the first mile post from MP coordinates,
```

```
if i     == startMP % first mile start at the MP.
%    chordBegin        = mileDat(2,:);
   chordBegin(1)     = mileDat(2,1); %easting
   chordBegin(2)     = mileDat(2,2); %northing
   chordBegin(3)     = mileDat(2,3); %elevation
   lastPt            = 2; %last point is the start of the MP
else
% Subsequent miles cross the MP boundry, begin at last station.
%      chordBegin        = chordEnd;
   chordBegin(1)     = chordEnd(1);
   chordBegin(2)     = chordEnd(2);
   chordBegin(3)     = chordEnd(3);
end

% clear the previous mile's stationing
   clear ftOffSta1 ftDeltaMO station;
   clear midOrd dc mpRef prof cumFtOffMO sumStaD dOT;

% Set the "done processing each station in the mile" flag to false
   done              = false;
% Reset stations per mile counter
   j                 = 0;
% Reset gap counter
   gapCount          = 0;

%%
% Begin processing stations in this mile until finished
   while done == false;

      % Increase next station counter
      j              = j + 1;

      if (dispCalc)
         fprintf('\nMile %4.0f to %4.0f, %5.0fft. Station: %4.0f to %4.0f\n'...
              ,i,i+stepValue,mileLength,j,j+1);
      end

% If it is the first chord endpoint of a new mile, start at the mile post
      if j == 1 % start of mile
        ftOffSta1(1,1)       = 0;
        cumFtOff             = 0;
        station(1,1)         = chordBegin(1);
        station(1,2)         = chordBegin(2);
        station(1,3)         = chordBegin(3);
        dist2end             = mileLength;

% In a data gap, the lastPt becomes empty, save the last point processed.
        if isempty(lastPt)
          lastMileDat        = lastMileDat;
        else
          lastMileDat        = lastPt;
        end

% Stations after the MP is 'stationDelta' distance from last station.
%  Determine next station coordinates.
        [chordBegin(1) chordBegin(2) chordBegin(3) lastPt staFlagOut gap] =...
             getIntLinesCircle(stationDelta,station(j,1),station(j,2),station(j,3),...
             mileDat(:,1),mileDat(:,2),mileDat(:,3),lastMileDat);
      else
% Accumulate the station distance
         ftOffSta1(j,1)      = dist2pts(chordBegin(1),chordBegin(2),station(j-1,1),station(j-1,2));
% Cumulative sum of the station differences
         cumFtOff            = cumsum(ftOffSta1);
         station(j,1)        = chordBegin(1);
         station(j,2)        = chordBegin(2);
         station(j,3)        = chordBegin(3);
```

```
% Determine the distance to go by subtracting the accumulated station
%  Determine distance to the end from mileLength.
         cumStaD        = cumsum(ftOffSta1); % linear measure from first point
         dist2end       = mileLength - cumStaD(j-1); %
% Keep track of the last data point processed, a call to getIntLinesCircle
%  can return an empty value after a number of data gaps.
        if isempty(lastPt)
            lastMileDat      = lastMileDat;
        else
            lastMileDat      = lastPt;
        end
      end % handling station 1

% Next station is 'stationDelta' distance from the station to a point on a
%   the set of ordered line segments.
     [chordBegin(1) chordBegin(2) chordBegin(3) lastPt staFlagOut gap] =...
          getIntLinesCircle(stationDelta,station(j,1),station(j,2),station(j,3),...
          mileDat(:,1),mileDat(:,2),mileDat(:,3),lastMileDat);

% reset lastMileDat to lastPt if lastPt not empty
     if (not(isempty(lastPt)))
          lastMileDat        = lastPt;
     end

%% ******* Data gap handler ********
% If the getIntLinesCircle function returns with a gap flag, then count it
%  and check to see if if the gap is greater a chordLength. If the gap is
% greater than a chordLength, display a gap warning and gap length.
   if staFlagOut          == true
        % Count the number of passes through the gap handler
        gapCount         = gapCount + 1;
        % flag if gap is equal to or greater than the chord length
        if gap           >= chordLength
           fprintf('\n***** Data Gap ******');
           fprintf('%8.2f ft gap %u to %u count:%u\nLast data point:%u last in:%u\n\n',...
                gap,j-1,j,gapCount,lastMileDat,lastPt);
        end
   end
%% ******* end handler ******
% Conditions that signal the end of the mile:
%  1. Previous MidOrd MP reference is not between mile posts.
%  2. Less than 1/2 a chord length to the ending mile post.
%  3. Less than a station distance to the ending mile post.
% ******************

% Determine the direct distance from the station to the mile post.
     direct2MP     = dist2pts(station(j,1),station(j,2),station(j,3),...
          mpX(mpIdxEnd),mpY(mpIdxEnd),mpZ(mpIdxEnd));
     if direct2MP  < chordLength/2
% Transition to next mile, set flags.
          done          = true;
          go2NextMile   = true;
%          pause;
     end
% Check for the last MidOrdOffset is in the next mile. !!!!
% Determine two cases for increasing or decreasing mile post
%  depends on the sign of stepValue!!!!
     switch stepValue
          % Increasing mp numbers
          case (1)
               if (mpRefMO > mpRefEnd) %check in next mile
                    done          = true;
                    go2NextMile   = true;
% Determine the direct distance from the station start to the
%  last data point in the mile set. The last data point is past the last mile
%  post.
```

```
          lastDatPt     = numel(mileDat(:,1));
          direct2LastPt = dist2pts(station(j,1),station(j,2),station(j,3),...
               mileDat(lastDatPt,1),mileDat(lastDatPt,2),mileDat(lastDatPt,3));
               end
          case (-1)
               if (mpRefMO < mpRefEnd) %check in next mile
                    done          = true;
                    go2NextMile   = true;
% Determine the direct distance from the station start to the
%  last data point in the mile set. The last data point is past the last mile
%  post.
          lastDatPt     = numel(mileDat(:,1));
          direct2LastPt = dist2pts(station(j,1),station(j,2),station(j,3),...
               mileDat(lastDatPt,1),mileDat(lastDatPt,2),mileDat(lastDatPt,3));
               end
     end
%%
% Calculate the chord ending coordinates, use the last mileDat
%           [chordEnd(1) chordEnd(2) chordEnd(3) lastPt chordFlagOut gap] =...
%             getIntLinesCircle(chordLength,station(j,1),station(j,2),station(j,3),...
%             mileDat(lastDatPt,1),mileDat(lastDatPt,2),mileDat(lastDatPt,3),las
%             tMileDat);
%%
if not(done)
%%
% Calculate the chord ending coordinates.
     if staFlagOut == true
          %skip it
     else
       [chordEnd(1) chordEnd(2) chordEnd(3) lastPt chordFlagOut gap] =...
       getIntLinesCircle(chordLength,station(j,1),station(j,2),station(j,3),...
       mileDat(:,1),mileDat(:,2),mileDat(:,3),lastMileDat);
     end
% Calculate a mile post reference for the station (chord beginning) coordinates
       mpRefSta1    = calcMilePostRef ( i, stepValue, cumFtOff(j), mileLength);
       if (dispCalc);
          fprintf('Station %4.0f (E/N/El) : %12.2f  %12.2f  %7.2f mp %10.6f\n',...
               j,station(j,1),station(j,2),station(j,3),mpRefSta1);
          fprintf('Station %4.0f (E/N/El) : %12.2f  %12.2f  %7.2f\n',...
               j+1,chordEnd(1),chordEnd(2),chordEnd(3));
       end

%% If the chordFlagOut is true, MOD & degOcrv are NaN.
% Otherwise determine the degree of curvature and the direction of travel.
       if chordFlagOut  == true;
          moDist        = noData;
          dOt(j,1)      = noData;
          degOcrv       = noData;
          midOrd(j,3)   = noData;
          % hold last mid ordinate elevation
          mOz           = mOz;
       else
% Determine distance from chord mid ordinate to track
       [moDir moDist mOx mOy mOz az]= findMidOrdDistance (chordLength,...
            station(j,1),station(j,2),station(j,3),mileDat(:,1),mileDat(:,2), ...
            mileDat(:,3),lastMileDat);
          dOt(j,1)      = az;
          degOcrv       = degreeOfCurvature(moDist,chordLength);

%% Filter Dc and elevation outliers. Dc will NEVER > 16. Rarely > 8.
%
     if (degOcrv > maxDc) || (degOcrv < -maxDc)
          degOcrv       = noData;
% If the Dc is out, so is the elevation
          mOz           = noData;
     end
```

```
     end %chordFlagOut check
       midOrd(j,1)      = mOx;
       midOrd(j,2)      = mOy;
       midOrd(j,3)      = mOz(1);%(1);

%% Determine the mile post reference of the mid ordinate.
%  If it is the first point in the mile set, the first MidOrd is half a chord
%  length from the first station. Otherwise, accumulate station deltas.
       if j == 1
          ftDeltaMO(j,1) = dist2pts(station(1,1),station(1,2),midOrd(j,1),midOrd(j,2));
       else
          ftDeltaMO(j,1) = dist2pts(midOrd(j-1,1),midOrd(j-1,2),midOrd(j,1),midOrd(j,2));
       end

       cumFtOffMO        = cumsum(ftDeltaMO);
       mpRefMO           = calcMilePostRef ( i, stepValue, cumFtOffMO(j), mileLength);

%% save calculated values to an output file
    fprintf(fid,'%4.0d,%+1.0d,%5.0f,%5.0f,%7.5f,%7.5f,%9.5f,%7.5f,%7.5f,%8.5f\n',...
    i,stepValue,cumFtOff(j),mileLength,mpRefMO,noData,mOz(1),noData,noData,degOcrv);

%% set counters
    numRec               = numRec +1;
    numRecThisMile       = numRecThisMile + 1;
    mpRef(numRecThisMile)= mpRefMO;
    dc(numRecThisMile)   = degOcrv;
    prof(numRecThisMile) = mOz(1);  % 2 elements ???
    figName              = strcat('MP',num2str(i,'%i'),'-',num2str(i+stepValue,'%i'));
%%
       % Display mid ordinate calcs if display flag is true.
       %  Calc display doubles processing time.
       if (dispCalc);
          fprintf('Midordinate %u mp:%10.5f LPI:%i  DtoEnd:%6.1f\n',j,mpRefMO,lastMileDat,dist2end);
          fprintf('Midordinate (E/N/El)  : %12.2f  %12.2f  %7.2f\n',mOx(1),mOy(1),mOz(1));
          fprintf('MOD:%5.3f MODir:%4.1f Dc@%u:%5.3f DoT:%4.2f\n'...
               ,moDist,moDir,chordLength,degOcrv,dOt(j,1));
       end
    end % not done with mile
%%
   end % while/done flag
   fprintf('\n************************************************************');
   fprintf('\n------------------------------------------------------------');
   fprintf('\nEnd mile %5.0f to %5.0f, %5.0f ft long, %4.0f stations.\n',i,i+stepValue, ...
        mileLength,j);
   fprintf('%5.0f total stations, %6.0f data points, mean pt sep %5.1f.\n',numRec, ...
        (datEndIdx-datStartIdx),meanDist);
   fprintf('%i stations w/o MOD due to data gaps, mile length is %7.2f feet',gapCount,mileLength);
   fprintf('\n------------------------------------------------------------');
   fprintf('\n************************************************************\n');
   pause(delay);
%% finished processing mile
toc
% close output file
fclose(fid);

% Pass mile post locations to plan view plot
MP(1,1)  = mpX(mpIdxBegin);
MP(1,2)  = mpY(mpIdxBegin);
MP(1,3)  = mpRefBegin;
MP(2,1)  = mpX(mpIdxEnd);
MP(2,2)  = mpY(mpIdxEnd);
MP(2,3)  = mpRefEnd;

%% Visualization

% Plan view, Northing vs Easting
```

```
%plotMapView(mileDat(:,1),mileDat(:,2),MP,figName,filePath)
mpScaleFactor      = mileLength/5280;
% Degree of curvature & profile vs mile post ref
plotDOC(stepValue,mpRef,dc,smoothFactor,stationDelta,chordLength,prof,mileDat(:,1), ...
    mileDat(:,2),MP,figName,filePath,dataFileName,outFile,alpha,trackID,mpScaleFactor);

% 3D Northing, Easting, Elevation
hold on;
figure('Name',figName,'NumberTitle','off');
plot3(mileDat(:,1),mileDat(:,2),mileDat(:,3));grid on;
text(MP(1,1),MP(1,2),MP(1,3),num2str(mpRefBegin));
text(MP(2,1),MP(2,2),MP(2,3),num2str(mpRefEnd));
hold off;
%% Track degree of curvature stats (usefull for tangent track)
if useStats
     [mu,sigma,muci,sigmaci, med] = docStat( filePath,outFile,alpha,figName,chordLength )
     fprintf('\nMean:%5.4f CI:%5.4f %5.4f StdDev:%5.4f CI:%5.4f %5.4f median:%5.4f\n',...
        mu,muci(1),muci(2),sigma,sigmaci(1),sigmaci(2),med);
end
%%
end

diary off;
```