```
% 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 Geo
% 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. Sur
% data must begin before the starting mile post and end after the closing m
% 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. Defa
% 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 i
% documentation for any purpose is hereby granted without fee, provided tha
% 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 witho
% specific, written prior permission. This license includes software writte
% 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);
```

```
61
62 if not(defaults)
63      % Default values
64      stationDelta  = 15.5;
65      chordLength   = 62;
66      trackID       = 'default';
67      smoothFactor  = 0.03;
68      delay         = 0;
69      dispCalc      = 1;
70      logFlag       = 0;
71      useDefault    = 1; % default file set
72      useStats      = 0;
73      alpha         = 0;
74 else
75
76 %% Get user input for stationing distance, default is 15.5 feet.
77 s                  = input('\nStation distance [15.5 feet]? ','s');
78    if isempty(s)
79       s            = '15.5';
80    end
81 stationDelta       = str2double(s);
82
83 % Get user input for chord distance, default is 62 feet.
84 s                  = input('Chord length [62 feet]? ','s');
85    if isempty(s)
86       s              = '62';
87    end
88 chordLength        = str2double(s);
89
90  % Get user input for track number.
91 s                  = input('Track number or ID? ','s');
92    if isempty(s)
93       s              = '';
94    end
95 trackID            = s;
96
97 % Get user input for maximum DegOfCrv.
98 s                  = input('Maximum expected Dc [8]? ','s');
99    if isempty(s)
100      s              = '8';
101   end
102 maxDc              = str2double(s);
103
104 % Get user input for smoothing factor.
105 s                  = input('Smoothing coefficient [0.03]? ','s');
106    if isempty(s)
107      s              = '0.03';
108   end
109 smoothFactor       = str2double(s);
110
111 % Get user input for pausing between miles, default is 0 seconds delay.
112 s                  = input('Display delay [0 seconds]? ','s');
113    if isempty(s)
114       s            = '0';
115    end
116 delay              = str2double(s);
117
118 % Get user input for displaying calculation information, default ON.
119 s                  = input('Display calculations? Yes [N]o ','s');
120    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
% 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 lin
% character - not Windows CR/LF
if (useDefault)
     startPath      = strcat([pwd '/']);
     windowText          = 'Pick mile post file';
     [mpFileName,filePath]= path2DataFile(startPath, windowText);
else
```

```
181      mpFileName          = '495-523_mp.CSV';
182      startPath           = '/Users/peteD/Documents/Engineering/Data/';
183 end
184 %% Read the mile post file
185 [ mpID mpfCode mpX mpY mpZ antHtMP hPrecMP vPrecMP filePath mpFileName ] =.
186      readTGOoutput ( 'Mile Post Locations',filePath,mpFileName);
187 fprintf('\nRead mile post file: %s\n',strcat([filePath mpFileName]));
188
189 %% Extract mile post number from mile post ID in mile post data
190 % extract point number using a regular expression
191 milePostNum         = (regexp(mpID(:),'[0-9]\d*','match'));
192 mpNum               = str2num(char([milePostNum{:,1}]));
193
194 %% Read survey data file. Data file from TGO format PJD-1 or 2.
195 %    expecting Unix end of line character - not Windows CR/LF
196
197 if (useDefault)
198      windowText          = 'Pick survey data file';
199      [dataFileName,filePath]= path2DataFile(filePath, windowText);
200 else
201      dataFileName        = 'mp499-500.CSV';
202      filePath            = '/Users/peteD/Documents/Engineering/Data/';
203 end
204 [ ptID fCode x y z antHt hPrec vPrec filePath dataFileName ] =...
205      readTGOoutput ('Survey',filePath,dataFileName);
206 fprintf('Read survey file: %s\n',strcat([filePath dataFileName]));
207
208 pointNum            = (regexp(ptID(:),'[0-9]\d*','match'));
209 ptNum               = str2num(char([pointNum{:,1}]));
210 %pointID             = str2num(ptID);
211
212 %% Start the waitbar
213 %h=waitbar(0,'Processing, please wait...');
214
215 % Step value is based on the direction of data processing,
216 %   increasing mile posts is positive, decreasing mile posts is negative
217 stepValue           = sign(mpNum(1)-mpNum(numel(mpNum)))*-1;
218
219 % Determine index of mile post within a chordLength of start of data
220 startFound          = false;
221 %% Determine the closest mile post to the first point to start, then insure
222 % the starting data point is within a chordLength of the mile post.
223 mpIdxStart          = 1;
224 numOfmp             = numel(mpNum);
225 mpIdxEnd            = numOfmp;
226
227 while not(startFound)
228      [index minD]        = findClosest(mpX(mpIdxStart),mpY(mpIdxStart),x,y);
229      startMP             = mpNum(mpIdxStart);
230      if minD < chordLength
231           startFound     = true;
232      fprintf('Nearest mile post to data start is MP%i at %3.2f feet.\n',sta
233      else
234           if mpIdxStart     == numel(mpX)
235                mpIdxStart   = mpIdxStart - 1;
236           else
237           mpIdxStart        = mpIdxStart + 1;
238           end
239      end
240 end
```

```
241
242 %% Determine index of mile post within a chordLength of the end of the data
243 endFound           = false;
244 % Determine the closest mile post to the last data point, then insure
245 % the ending data point is within a chordLenth of the mile post.
246 while not(endFound)
247      [index minD]     = findClosest(mpX(mpIdxEnd),mpY(mpIdxEnd),x,y);
248      endMP            = mpNum(mpIdxEnd);
249      if minD < chordLength
250           endFound    = true;
251      fprintf('Nearest mile post to data end is MP%i at %3.2f feet.\n',endMP
252      else
253           fprintf('No data near end, MP%i at %3.2f feet.\n',endMP,minD);
254           mpIdxEnd       = mpIdxEnd - 1;
255      end
256 end
257 %%
258 % Length of data, in mile post numbers from start to end of data
259 mp2mp              = abs(mpNum(mpIdxStart)-mpNum(mpIdxEnd));
260 fprintf('Data length from MP%3.0f to MP%3.0f is %3.0f miles.\n',startMP,end
261
262 %% Make a dataset of nearset point to each mile post
263 closestFound       = false;
264 ptIdx = 1;
265 for i = mpIdxStart:mpIdxEnd
266      [index minD]     = findClosest(mpX(i),mpY(i),x,y);
267      nearest(i,1)     = mpNum(i);     % the mile post number
268      nearest(i,2)     = index;        % the index in the x,y dataset
269      nearest(i,3)     = minD;         % the distance to the MP
270      nearest(i,4)     = ptNum(index); % the data point number
271      fprintf('Nearest point to MP%4.0f is point %6.0f PtID %6.0f at %3.2f f
272 end
273 %% Open output file for mile or mile series
274 % fileNameRoot       = regexp(dataFileName,'[^.CSV]','match');
275 % fileNameRoot       = strcat(fileNameRoot{:});
276 outFile            = strcat([num2str(startMP) '-' num2str(endMP) '_' num2st
277 outFileName        = strcat([filePath outFile]);
278 % Create new file for writing. Discard existing contents, if any.
279 fid                = fopen (outFileName,'w');
280 % Entry heading
281 fprintf(fid,'mp,dir,offset,length,mpRef,profLft62ft,profRt62ft,alignLeft,al
282 %% Process the data between mile posts
283 for i = startMP:stepValue:(endMP-stepValue)
284
285 %% open output file for single mile
286 % fileNameRoot       = regexp(dataFileName,'[^.CSV]','match');
287 % fileNameRoot       = strcat(fileNameRoot{:});
288 outFile            = strcat([num2str(startMP) '-' num2str(endMP) '_' num2st
289 outFileName        = strcat([filePath outFile]);
290 % Open file for writing. Append data to the end of the file.
291 fid                = fopen (outFileName,'a');
292 %
293 %% Initial conditions for mile
294      % waitbar(i/numMiles,h);drawnow
295    fprintf('\nBegining to work on MP%3.0f to MP%3.0f.\n',i,i+stepValue);
296    gapCount        = 0;
297    numRecThisMile  = 0;
298    go2NextMile     = false; % reset skip flag
299    mpRefMO         = i;
300    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(datStart
else
   azStart             = az2pts (x(datStartIdx-1),y(datStartIdx-1),x(datS
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(datStart
% If the direction of travel to the data point is 'behind' the MP, then in
% 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(da
   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 po
% next mile post to the closest survey data point
   azEnd               = az2pts (x(datEndIdx-1),y(datEndIdx-1),x(datEndId
   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),mp
% If the direction of travel to the data point is 'behind' the MP, then in
% 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(mpI
   end

% ********************************************************************
%% Build the data set for this mile in the form:
% 1. before MP point > 2. startMP > 3. survey data > 4. endMP > 5. after M
% 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(mp
```

```
          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
else
     tempB          = [x(datEndIdx:datStartIdx) y(datEndIdx:datStartIdx) z
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 +
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
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+stepV
   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
```

```
421    while done == false;
422
423       % Increase next station counter
424       j            = j + 1;
425
426       if (dispCalc)
427          fprintf('\nMile %4.0f to %4.0f, %5.0fft. Station: %4.0f to %4.0f\
428               ,i,i+stepValue,mileLength,j,j+1);
429       end
430
431 % If it is the first chord endpoint of a new mile, start at the mile post
432       if j == 1 % start of mile
433         ftOffSta1(1,1)       = 0;
434         cumFtOff             = 0;
435         station(1,1)         = chordBegin(1);
436         station(1,2)         = chordBegin(2);
437         station(1,3)         = chordBegin(3);
438         dist2end             = mileLength;
439
440 % In a data gap, the lastPt becomes empty, save the last point processed.
441         if isempty(lastPt)
442           lastMileDat       = lastMileDat;
443         else
444           lastMileDat       = lastPt;
445         end
446
447 % Stations after the MP is 'stationDelta' distance from last station.
448 %  Determine next station coordinates.
449         [chordBegin(1) chordBegin(2) chordBegin(3) lastPt staFlagOut gap]
450              getIntLinesCircle(stationDelta,station(j,1),station(j,2),stat
451              mileDat(:,1),mileDat(:,2),mileDat(:,3),lastMileDat);
452       else
453 % Accumulate the station distance
454          ftOffSta1(j,1)      = dist2pts(chordBegin(1),chordBegin(2),statio
455 % Cumulative sum of the station differences
456          cumFtOff            = cumsum(ftOffSta1);
457          station(j,1)        = chordBegin(1);
458          station(j,2)        = chordBegin(2);
459          station(j,3)        = chordBegin(3);
460 % Determine the distance to go by subtracting the accumulated station
461 %  Determine distance to the end from mileLength.
462          cumStaD        = cumsum(ftOffSta1); % linear measure from first p
463          dist2end       = mileLength - cumStaD(j-1); %
464 % Keep track of the last data point processed, a call to getIntLinesCircle
465 %  can return an empty value after a number of data gaps.
466          if isempty(lastPt)
467              lastMileDat      = lastMileDat;
468          else
469              lastMileDat      = lastPt;
470          end
471       end % handling station 1
472
473 % Next station is 'stationDelta' distance from the station to a point on a
474 %   the set of ordered line segments.
475       [chordBegin(1) chordBegin(2) chordBegin(3) lastPt staFlagOut gap] =..
476            getIntLinesCircle(stationDelta,station(j,1),station(j,2),station
477            mileDat(:,1),mileDat(:,2),mileDat(:,3),lastMileDat);
478
479 % reset lastMileDat to lastPt if lastPt not empty
480       if (not(isempty(lastPt)))
```

```
481           lastMileDat        = lastPt;
482      end
483
484 %% ******* Data gap handler ********
485 % If the getIntLinesCircle function returns with a gap flag, then count it
486 %  and check to see if if the gap is greater a chordLength. If the gap is
487 % greater than a chordLength, display a gap warning and gap length.
488    if staFlagOut          == true
489         % Count the number of passes through the gap handler
490         gapCount         = gapCount + 1;
491         % flag if gap is equal to or greater than the chord length
492         if gap           >= chordLength
493            fprintf('\n***** Data Gap ******');
494            fprintf('%8.2f ft gap %u to %u count:%u\nLast data point:%u las
495         end
496    end
497 %% ******* end handler ******
498 % Conditions that signal the end of the mile:
499 %  1. Previous MidOrd MP reference is not between mile posts.
500 %  2. Less than 1/2 a chord length to the ending mile post.
501 %  3. Less than a station distance to the ending mile post.
502 % ******************
503
504 % Determine the direct distance from the station to the mile post.
505      direct2MP     = dist2pts(station(j,1),station(j,2),station(j,3),...
506           mpX(mpIdxEnd),mpY(mpIdxEnd),mpZ(mpIdxEnd));
507      if direct2MP  < chordLength/2
508 % Transition to next mile, set flags.
509           done          = true;
510           go2NextMile   = true;
511 %          pause;
512      end
513 % Check for the last MidOrdOffset is in the next mile. !!!!
514 % Determine two cases for increasing or decreasing mile post
515 %  depends on the sign of stepValue!!!!
516      switch stepValue
517           % Increasing mp numbers
518           case (1)
519                if (mpRefMO > mpRefEnd) %check in next mile
520                     done          = true;
521                     go2NextMile   = true;
522 % Determine the direct distance from the station start to the
523 %  last data point in the mile set. The last data point is past the last m
524 %  post.
525           lastDatPt     = numel(mileDat(:,1));
526           direct2LastPt = dist2pts(station(j,1),station(j,2),station(j,3),
527                mileDat(lastDatPt,1),mileDat(lastDatPt,2),mileDat(lastDatPt
528                end
529           case (-1)
530                if (mpRefMO < mpRefEnd) %check in next mile
531                     done          = true;
532                     go2NextMile   = true;
533 % Determine the direct distance from the station start to the
534 %  last data point in the mile set. The last data point is past the last m
535 %  post.
536           lastDatPt     = numel(mileDat(:,1));
537           direct2LastPt = dist2pts(station(j,1),station(j,2),station(j,3),
538                mileDat(lastDatPt,1),mileDat(lastDatPt,2),mileDat(lastDatPt
539                end
540      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),stat
%             mileDat(lastDatPt,1),mileDat(lastDatPt,2),mileDat(lastDatPt,
%             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) coordi
       mpRefSta1    = calcMilePostRef ( i, stepValue, cumFtOff(j), mileLen
       if (dispCalc);
          fprintf('Station %4.0f (E/N/El) : %12.2f  %12.2f  %7.2f mp %10.6
               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(:,
          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 ch
%  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),
       else
```

```
601           ftDeltaMO(j,1) = dist2pts(midOrd(j-1,1),midOrd(j-1,2),midOrd(j,1
602       end
603
604       cumFtOffMO         = cumsum(ftDeltaMO);
605       mpRefMO            = calcMilePostRef ( i, stepValue, cumFtOffMO(j),
606
607 %% save calculated values to an output file
608    fprintf(fid,'%4.0d,%+1.0d,%5.0f,%5.0f,%7.5f,%7.5f,%9.5f,%7.5f,%7.5f,%8
609    i,stepValue,cumFtOff(j),mileLength,mpRefMO,noData,mOz(1),noData,noData
610
611 %% set counters
612    numRec             = numRec +1;
613    numRecThisMile     = numRecThisMile + 1;
614    mpRef(numRecThisMile)= mpRefMO;
615    dc(numRecThisMile)   = degOcrv;
616    prof(numRecThisMile) = mOz(1);  % 2 elements ???
617    figName            = strcat('MP',num2str(i,'%i'),'-',num2str(i+stepV
618 %%
619       % Display mid ordinate calcs if display flag is true.
620       %  Calc display doubles processing time.
621       if (dispCalc);
622          fprintf('Midordinate %u mp:%10.5f LPI:%i  DtoEnd:%6.1f\n',j,mpRe
623          fprintf('Midordinate (E/N/El)  : %12.2f  %12.2f  %7.2f\n',mOx(1)
624          fprintf('MOD:%5.3f MODir:%4.1f Dc@%u:%5.3f DoT:%4.2f\n'...
625               ,moDist,moDir,chordLength,degOcrv,dOt(j,1));
626       end
627    end % not done with mile
628 %%
629   end % while/done flag
630   fprintf('\n*************************************************************'
631   fprintf('\n-------------------------------------------------------------'
632   fprintf('\nEnd mile %5.0f to %5.0f, %5.0f ft long, %4.0f stations.\n',i
633   fprintf('%5.0f total stations, %6.0f data points, mean pt sep %5.1f.\n'
634   fprintf('%i stations w/o MOD due to data gaps, mile length is %7.2f fee
635   fprintf('\n-------------------------------------------------------------'
636   fprintf('\n*************************************************************\
637   pause(delay);
638 %% finished processing mile
639 toc
640 % close output file
641 fclose(fid);
642
643 % Pass mile post locations to plan view plot
644 MP(1,1)  = mpX(mpIdxBegin);
645 MP(1,2)  = mpY(mpIdxBegin);
646 MP(1,3)  = mpRefBegin;
647 MP(2,1)  = mpX(mpIdxEnd);
648 MP(2,2)  = mpY(mpIdxEnd);
649 MP(2,3)  = mpRefEnd;
650
651 %% Visualization
652
653 % Plan view, Northing vs Easting
654 %plotMapView(mileDat(:,1),mileDat(:,2),MP,figName,filePath)
655 mpScaleFactor      = mileLength/5280;
656 % Degree of curvature & profile vs mile post ref
657 plotDOC(stepValue,mpRef,dc,smoothFactor,stationDelta,chordLength,prof,mile
658
659 % 3D Northing, Easting, Elevation
660 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,figNam
     fprintf('\nMean:%5.4f CI:%5.4f %5.4f StdDev:%5.4f CI:%5.4f %5.4f medi
end
%%
end

diary off;
```