```
function [xInt yInt zInt lastIn flagOut gap] = getIntLinesCircle (r,cx,cy,c
% getIntLinesCircle.m - returns coordinates of the intersection of a
%      circle (given origin & radius) with an ordered series of lines
%      defined by their x&y coordinate pairs.
%   Input:
%    search radius
%    center point coordinates
%    x,y,z end points of an ordered series of line segments
%
%   Output:
%    x & y coordinates of the intersection with the ordered line series
%    index of the last point inside the search radius
%    flag indicating no point inside the search radius indication a gap
%         a data gap beyond the the radius
%    distance to the point outside the search radius
%
% Syntax:[flag x y Z] = getIntLinesCircle (radius, circle origin,
%                  x,y,z coords, *index of last point processed* optional)
%
% ************************************************************************
% Other m-files required:
%
% Subfunctions: findLastPointWithinRadius, findFirstPointOutsideRadius,
%         findIntersect.
%
% MAT-files required: none
%
% See also: Survey Theory & Practice, 7th ed. J.Anderson, E.Mikhail
%  pp. 1076-1077 A.26
% ************************************************************************
% 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-August-2009
% ************************************************************************
%
% 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.
% 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.
% ************************************************************************

% Determine the furthest point from the origin inside the search radius
[lastIn gap]       = findLastPointWithinRadius(r,cx,cy,cz,x,y,z);

% If lastIn is empty there is no next point inside the radius
%   which means there is a data gap.
%   Look for the next point outside the search radius.
if isempty(lastIn)
% Determine the next data point beyond the cap, and the distance between
```

```
%   the search origin and the next data point
   [firstPtOut gap] = findFirstPointOutsideRadius(r,cx,cy,cz,x,y,z,lastInde
% Set a flag that indicates the point is outside the search radius
   flagOut           = true; % next point is outside chord
% Define the line segment from last data point and the next data point
   lineSegX        = x(lastIndex:lastIndex+1);
   lineSegY        = y(lastIndex:lastIndex+1);
   lineSegZ        = z(lastIndex:lastIndex+1);

% If lastIn is not empty,  and if the next point is
%   inside the search radius, and more data points exist beyond the search,
%   define the line segment and the next intersection
elseif (lastIn+1 <= numel(x));
% No gap, set flags and gap distance to false
   flagOut         = false;
   gap             = 0;
   % define the intersecting line segment
   lineSegX        = x(lastIn:lastIn+1);
   lineSegY        = y(lastIn:lastIn+1);
   lineSegZ        = z(lastIn:lastIn+1);

end %if
% Find the intersection of circle & line
[xInt yInt zInt]= findIntersect(r,cx,cy,cz,lineSegX,lineSegY,lineSegZ);

end % end main function

function [lastPointIn gap] = findLastPointWithinRadius(r,cx,cy,cz,x,y,z)

% Input: data set X & Y values; circle center x & y.
% Output: index of the last point within radius

deltaX = x - cx; % all X coord - circle origin X coord, Cx
deltaY = y - cy; % all Y coord - circle origin Y coord, Cy
deltaZ = z - cz; % all Z coord - circle origin Z coord, Cz

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

% flag distance index with logical when distance <= to the
%  distance to the line end point.
flagInPoints = (distance <= r); % true is within circle radius

% return the index of the last point that was flagged
lastPointIn     = find(flagInPoints, 1, 'last');

gap               = 0; % set the gap distance to zero
end

function [firstPtOut gap] = findFirstPointOutsideRadius(r,cx,cy,cz,x,y,z,la
% This function searches for the next point outside the search radius

% Input: data set X & Y values; circle center x & y.
% Output: index of the closest point outside the radius

deltaX = x - cx; % all X coord - circle origin X coord, Cx
deltaY = y - cy; % all Y coord - circle origin Y coord, Cy
deltaZ = z - cz; % all Z coord - circle origin Z coord, Cz

% distance = pythagorus from center to all points
```

```
distanceOut        = sqrt(deltaX.^2 + deltaY.^2 + deltaZ.^2);

% flag distance index with logical when distance <= to the
%  distance to the line end point.
flagOutPoints      = (distanceOut > r); % true is within circle radius

flagOutPoints(1:lastIdx)    = 0; %zero all dist to previous points

% return the index of the first point that was flagged
firstPtOut              = find(flagOutPoints,1,'first');
% gap is the distance beyond the chord end
gap                     = distanceOut(firstPtOut);

end

function [xInt yInt zInt]=findIntersect(r,cx,cy,cz,lineX,lineY,lineZ)
% Input: circle origin (station coordinates), radius, and line end point co
% Output: x & y coordinates of intersection
%   Intersection of a line starting at the circle origin to a point outside
%     circle. Only one intersection.

% Weisstein, Eric W. "Circle-Line Intersection." From MathWorld--A Wolfram
% Resource. http://mathworld.wolfram.com/Circle-LineIntersection.html
%
% d_x   =   x_2-x_1, => diff(lineX)
% d_y   =   y_2-y_1. => diff(lineY)
% d_r   =   sqrt(d_x^2 + d_y^2)
% D =   |x_1 x_2; y_1 y_2|= (x_1 * y_2) - (x_2 * y_1)
%
% x =   (Dd_y  +/- sgn *(d_y) * d_x * sqrt(r^2 * d_r^2 - D^2))/(d_r^2)
% sgn(x)= -1 for x <0; 1 otherwise. No negative coordinates, so sgn(x) = 1
% y =   (-Dd_x +/- |d_y|sqrt(r^2 * d_r^2 - D^2))/(d_r^2)
%
% The discriminant, Delta = r^2 * d_r^2 - D^2
% Delta    incidence
% Delta<0     no intersection
% Delta=0     tangent
% Delta>0     intersection
%
a = diff(lineX)^2 + diff(lineY)^2 + diff(lineZ)^2; %
b = 2 * (diff(lineX)*(lineX(1) - cx) + diff(lineY)*(lineY(1) - cy) + diff(l
c = (cx - lineX(1))^2 + (cy - lineY(1))^2 + (cz - lineZ(1))^2 - r^2; %
% Use the quadratic equation to find the intersection X1
u = (-b + sqrt(b^2 - (4*a*c)))/(2*a);
% Parametric form of the line segment, x intercept & y intercept
xInt(1) = lineX(1) + u*diff(lineX);
yInt(1) = lineY(1) + u*diff(lineY);
zInt(1) = lineZ(1) + u*diff(lineZ);
end
```