BOUNDARY ELEMENT CODE: TWODD (19)

I Main topics

A Comparison of FEM, BEM, FD methods

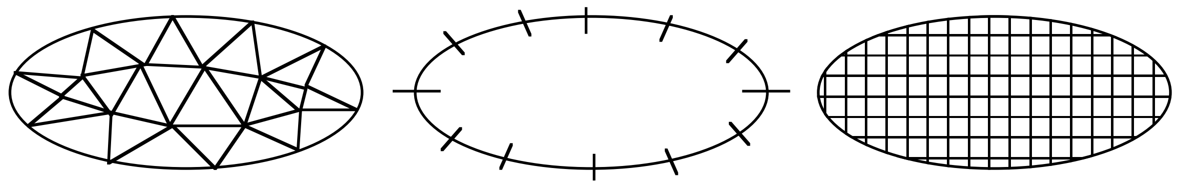
B Organization of boundary element code twodd

Modified from Crouch and Starfield (1983)

C Listing of boundary element code twodd.m and associated scripts

II Comparison of FEM, BEM, FD methods

Finte element (FEM) method | Boundary element (BEM) method | Finite difference (FD) method



Domain elements	Boundary elements	Internal cells
Governing differential equations relate behavior of adjacent elements	Governing equations are integral equations	Governing differential equations relate behavior of adjacent elements; written in finite difference form
System of simultaneous equations solved	System of simultaneous equations solved	System of simultaneous equations solved
<u>Main Advantages</u> Widely tested approach Commercial availability Flexibility Handles many geometries Accuracy Handles stress concentrations Good for nonlinear problems	<u>Main Advantages</u> Reduces problem dimension Low memory demands Less unnecessary information Focus on the body boundary Good for incompressible materials Easy to define and vary boundary elements Accuracy Good for stress concentrations **(fracture)**	<u>Main Advantages</u> Conceptual simplicity Mathematical simplicity Ease of programming
<u>Main Disadvantages</u> Generation/variation of mesh Large matrices Memory intensive Generates "waste" information Poor for incompressible materials	<u>Main Disadvantages</u> Unfamiliar mathematics Not efficient for nonlinear problems Not good for thin shells Fully populated matrices (but matrices are small)	<u>Main Disadvantages</u> Not well suited for many geometries (i.e., curves) Does not handle stress concentrations well Memory intensive

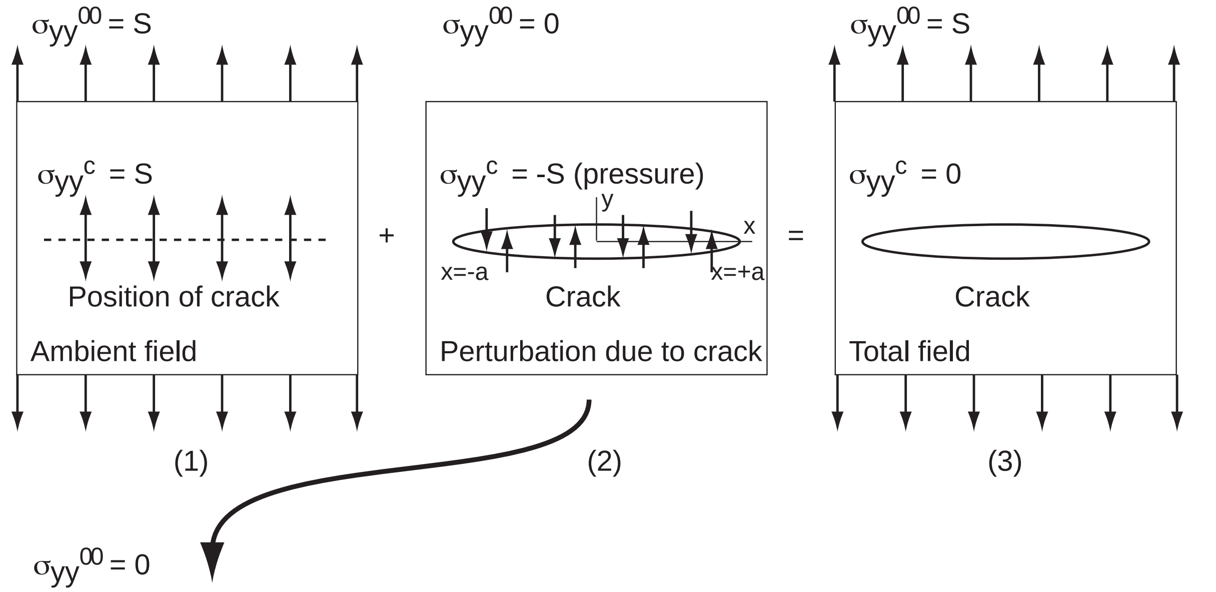
III Organization of boundary element code twoddm

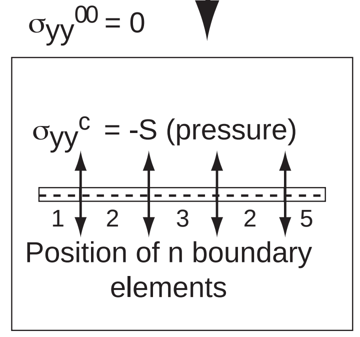
- A Read elastic constants, remote stress field data, and boundary data
- B Define new elastic constants to be used internally
- C Define locations, lengths, orientations, and boundary conditions for boundary elements
- D Adjust stress boundary conditions (B) to account for remote stresses
- E Compute influence coefficients (A) between boundary elements
- F Solve system of algebraic equations to get the displacement discontinuities (i.e., solve AX = B for X)
- G Compute displacements and stresses ("B") on boundary elements
- H Calculate influence coefficients at gridpoints (A*)
- I Calculate stresses and displacements (B*) at grid points (i.e., solve for B* in A*X = B*) due to relative motion of fracture elements
- J Add back the remote field to obtain the total stress field

IV Listing of boundary element code twodd.m and associated scripts

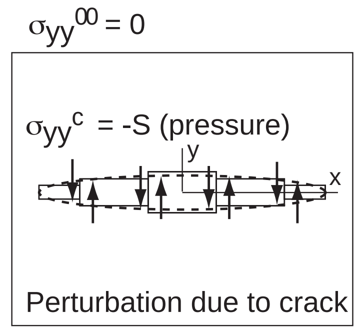
Main Steps in the Displacement Discontinuity Method

Define the boundary conditions ("B") for the perturbation problem. For 2-D, B is a 2nx1 array, where "n" is the number of boundary elements.

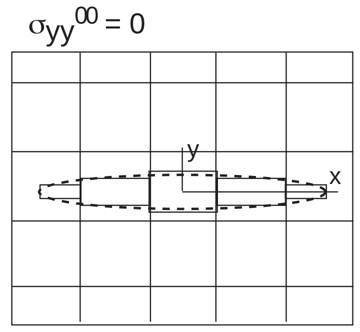



$\sigma_{yy}^{00} = 0$
$\sigma_{yy}^{c} = -S$ (pressure)
1 2 3 2 5
Position of n boundary elements

Find influence coefficients ("A"):
Calculate effect of a unit opening on element "i" on the stresses at element "j". Do this for all element-element combinations.
If "n" is the number of elements, 2nx2n will be the size of "A" for a 2-D problem.
For a 3-D problem, "A" will be a 3nx3n matrix.



Solve for the displacement discontinuities ("X) on all the elements such that the superposed induced stresses from all the elements yield the normal and shear stresses (i.e., the boundary conditions) on all the elements for the modified problem.
Do this for all the element-element combinations.
In other words, sove for "X" is AX = B.
For 2-D, X is a 2nx1 array.



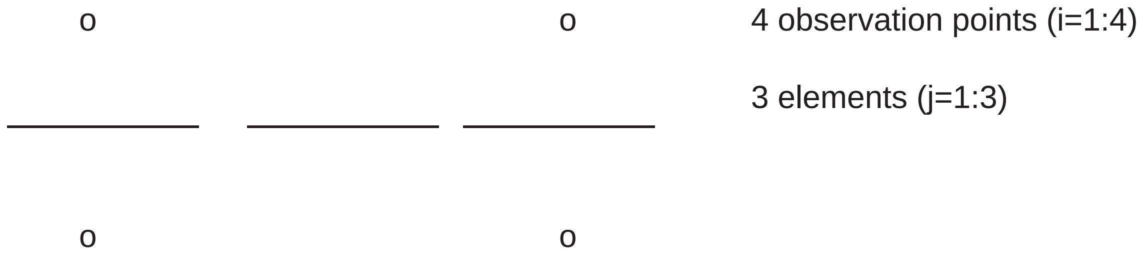
Find a new set of influence coefficients ("A*"):
Calculate effect of a unit opening on element "i" on the stresses at gridpoint "j". Do this for all element-gridpoint combinations.

Then, using A* and the known displacement discontintuities (X), solve for the stresses and displacements ("B*") at the grid points. This solves the stress perturbation problem (2). To solve for the total stress field (3), back the ambient field (1).

Methodology in function twoddm_coeff

A(i,j)	d(j)	b(i)
Influence (effect) on observation pt i of unit cause at pt j	Cause of unknown strength at pt j	Prescribed effect at observation pt i

Consider the following case with m=4 observation points and n=3 elements



The mxn influence coefficient matrix A, the nx1 array d, and the mx1 array b.
The mxn influence coefficient matrix A relates each observation point i to each element j.
Each element of A is a particular displacement component or a particular stress component.

$$\begin{bmatrix} A_{11} & A_{12} & A_{1n} \\ A_{21} & A_{22} & A_{2n} \\ A_{31} & A_{32} & A_{3n} \\ A_{m1} & A_{m2} & A_{mn} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_m \end{bmatrix}$$

m can equal n, or m can equal pxq

Organization of m observation points into n columns. Each column has m points/rows: i = 1:m

$$\begin{bmatrix} X_1 & X_1 & X_1 \\ X_2 & X_2 & X_2 \\ X_3 & X_3 & X_3 \\ X_m & X_m & X_m \end{bmatrix}$$

Organization of n elements into m rows. Each row has n elements/columns: j = 1:n

$$\begin{bmatrix} x_1 & x_2 & x_n \\ x_1 & x_2 & x_n \\ x_1 & x_2 & x_n \\ x_1 & x_2 & x_n \end{bmatrix}$$

Influence coefficients can be converted from one reference frame to another. For example, consider influence coefficient matrices C and D that consist of displacement components in the xi- and yj-directions, respectively, where the xj and yj directions can vary from element to element. Now suppose we want influence coefficient matrices A and B composed of displacement components that act in the X- and Y-directions, respectively. Direction cosine matrices relate the influence coefficient matrices in the different reference frames. The arrangement of direction cosine matrices below makes this work. Note that .* multiplication is used here rather than * multiplication to relate the mxn matrices C and N to the mxn matrices A and B (and to relate coefficient ij in matrix A to coefficient ij in matrix B). This requires the direction cosine matrices to be mxn matrices, with each row of a particular direction cosine matrix matching all the other rows. I do not see how to make this conversion with * multiplication because that would require relating coefficient ij in matrix A to all the elements in row i in a direction cosine matrix and all the coefficients in column j of matrix B, and that won't yield the desired conversion.

$$\begin{bmatrix} A_{11} & A_{12} & A_{1n} \\ A_{21} & A_{22} & A_{2n} \\ A_{31} & A_{32} & A_{3n} \\ A_{m1} & A_{m2} & A_{mn} \end{bmatrix} = \begin{bmatrix} aXx_1 & aXx_2 & aXx_n \\ aXx_1 & aXx_2 & aXx_n \\ aXx_1 & aXx_2 & aXx_n \\ aXx_1 & aXx_2 & aXx_n \end{bmatrix} .* \begin{bmatrix} C_{11} & C_{12} & C_{1n} \\ C_{21} & C_{22} & C_{2n} \\ C_{31} & C_{32} & C_{3n} \\ C_{m1} & C_{m2} & C_{mn} \end{bmatrix} + \begin{bmatrix} aXy_1 & aXy_2 & aXy_n \\ aXy_1 & aXy_2 & aXy_n \\ aXy_1 & aXy_2 & aXy_n \\ aXy_1 & aXy_2 & aXy_n \end{bmatrix} .* \begin{bmatrix} D_{11} & D_{12} & D_{1n} \\ D_{21} & D_{22} & D_{2n} \\ D_{31} & D_{32} & D_{3n} \\ D_{m1} & D_{m2} & D_{mn} \end{bmatrix}$$

$$\begin{bmatrix} B_{11} & B_{12} & B_{1n} \\ B_{21} & B_{22} & B_{2n} \\ B_{31} & B_{32} & B_{3n} \\ B_{m1} & B_{m2} & B_{mn} \end{bmatrix} = \begin{bmatrix} aYx_1 & aYx_2 & aYx_n \\ aYx_1 & aYx_2 & aYx_n \\ aYx_1 & aYx_2 & aYx_n \\ aYx_1 & aYx_2 & aYx_n \end{bmatrix} .* \begin{bmatrix} C_{11} & C_{12} & C_{1n} \\ C_{21} & C_{22} & C_{2n} \\ C_{31} & C_{32} & C_{3n} \\ C_{m1} & C_{m2} & C_{mn} \end{bmatrix} + \begin{bmatrix} aYy_1 & aYy_2 & aYy_n \\ aYy_1 & aYy_2 & aYy_n \\ aYy_1 & aYy_2 & aYy_n \\ aYy_1 & aYy_2 & aYy_n \end{bmatrix} .* \begin{bmatrix} D_{11} & D_{12} & D_{1n} \\ D_{21} & D_{22} & D_{2n} \\ D_{31} & D_{32} & D_{3n} \\ D_{m1} & D_{m2} & D_{mn} \end{bmatrix}$$

or

[mxn]= [mxn].*[mxn]+ [mxn].*[mxn] [mxn]= [mxn].*[mxn]+ [mxn].*[mxn]
[Aij] = [aXx].*[Cij] + [aXy].*[Dij] [Bij] = [aYx].*[Cij] + [aYy].*[Dij]

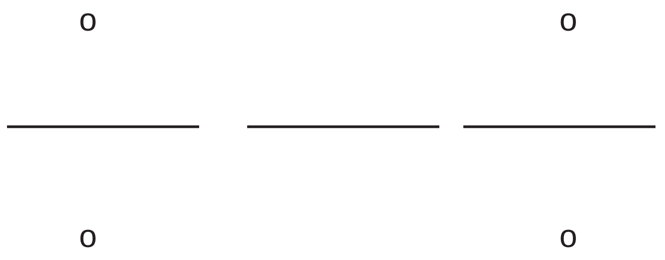
An analogous approach can be used to transform the stress influence coefficient matrices, except that two sets of direction cone matrices would be needed everywhere a single direction cosine matrix appears above. For example, if [A] represents the σXX effect, C the σxx effect, D the σxy effect, E the σyx effect, and F the σyy effect, then

[Aij] = [aXx].*[aXx].*[Cij] + [aXx].*[aXy].*[Dij] +
[aXy].*[aXx].*[Eij] + [aXy].*[aXy].*[Fij]

Methodology in function twoddm

A(i,j)	d(j)	b(i)
Influence (effect) on observation pt i of unit cause at pt j	Cause of unknown strength at pt j	Prescribed effect at observation pt i

Consider the following case



4 observation points (i=1:4)

3 elements (j=1:3)

$[\sin\beta_1 \quad \sin\beta_2 \quad \sin\beta_n]$ Organization of sines of element angles, i = 1:n

$[\cos\beta_1 \quad \cos\beta_2 \quad \cos\beta_n]$ Organization of cosines of element angles, i = 1:n

$[anX_1 \quad anX_2 \quad anX_n]$ Organization of direction cosines, i = 1:n

$$[AnX] = \begin{bmatrix} anX_1 & anX_2 & anX_n \\ anX_1 & anX_2 & anX_n \\ anX_1 & anX_2 & anX_n \end{bmatrix}$$ Direction cosine matrix AnX - rows are copes of anX.

ELEMENTS

Convert xy coefficients to ns reference frame of the "i" elements

Stress influence coefficients

$$\begin{bmatrix} sns_{11} & sns_{12} & sns_{1n} \\ sns_{21} & sns_{22} & sns_{2n} \\ sns_{n1} & sns_{n2} & sns_{nn} \end{bmatrix} = \begin{bmatrix} anx_1 & anx_1 & anx_1 \\ anx_2 & anx_2 & anx_2 \\ anx_n & anx_n & anx_n \end{bmatrix} .* \begin{bmatrix} asx_1 & asx_1 & asx_1 \\ asx_2 & asx_2 & asx_2 \\ asx_n & asx_n & asx_n \end{bmatrix} .* \begin{bmatrix} sxx_{11} & sxx_{12} & sxx_{1n} \\ sxx_{21} & sxx_{22} & sxx_{2n} \\ sxx_{n1} & sxx_{n2} & sxx_{nn} \end{bmatrix}$$
$$+ \begin{bmatrix} anx_1 & anx_1 & anx_1 \\ anx_2 & anx_2 & anx_2 \\ anx_n & anx_n & anx_n \end{bmatrix} .* \begin{bmatrix} asy_1 & asy_1 & asy_1 \\ asy_2 & asy_2 & asy_2 \\ asy_n & asy_n & asy_n \end{bmatrix} .* \begin{bmatrix} sxy_{11} & sxy_{12} & sxy_{1n} \\ sxy_{21} & sxy_{22} & sxy_{2n} \\ sxy_{n1} & sxy_{n2} & sxy_{nn} \end{bmatrix}$$
$$+ \begin{bmatrix} any_1 & any_1 & any_1 \\ any_2 & any_2 & any_2 \\ any_n & any_n & any_n \end{bmatrix} .* \begin{bmatrix} asx_1 & asx_1 & asx_1 \\ asx_2 & asx_2 & asx_2 \\ asx_n & asx_n & asx_n \end{bmatrix} .* \begin{bmatrix} syx_{11} & syx_{12} & syx_{1n} \\ syx_{21} & syx_{22} & syx_{2n} \\ syx_{n1} & syx_{n2} & syx_{nn} \end{bmatrix}$$
$$+ \begin{bmatrix} any_1 & any_1 & any_1 \\ any_2 & any_2 & any_2 \\ any_n & any_n & any_n \end{bmatrix} .* \begin{bmatrix} asy_1 & asy_1 & asy_1 \\ asy_2 & asy_2 & asy_2 \\ asy_n & asy_n & asy_n \end{bmatrix} .* \begin{bmatrix} syy_{11} & syy_{12} & syy_{1n} \\ syy_{21} & syy_{22} & syy_{2n} \\ syy_{n1} & syy_{n2} & syy_{nn} \end{bmatrix}$$

$$[sns] = [Anx].*[Asx].*[sxx] + [Anx].*[Asy].*[sxy] + [Any].*[Asx].*[syx] + [Any].*[Asy].*[syy]$$

Displacement influence coefficients

$$\begin{bmatrix} usi\_sj_{11} & usi\_sj_{12} & usi\_sj_{1n} \\ usi\_sj_{21} & usi\_sj_{22} & usi\_sj_{2n} \\ usi\_sj_{n1} & usi\_sj_{n2} & usi\_sj_{nn} \end{bmatrix} = \begin{bmatrix} asx_1 & asx_1 & asx_1 \\ asx_2 & asx_2 & asx_2 \\ asx_n & asx_n & asx_n \end{bmatrix} .* \begin{bmatrix} uxs_{11} & uxs_{12} & uxs_{1n} \\ uxs_{21} & uxs_{22} & uxs_{2n} \\ uxs_{n1} & uxs_{n2} & uxs_{nn} \end{bmatrix}$$
$$+ \begin{bmatrix} asy_1 & asy_1 & asy_1 \\ asy_2 & asy_2 & asy_2 \\ asy_n & asy_n & asy_n \end{bmatrix} .* \begin{bmatrix} uys_{11} & uys_{12} & uys_{1n} \\ uys_{21} & uys_{22} & uys_{2n} \\ uys_{n1} & uys_{n2} & uys_{nn} \end{bmatrix}$$

$$[usi\_sj] = [Asx].*[uxs] + [Asy].*[uys]$$

Listing of Matlab code easy_anti_input1.m

```
% easy_anti_input1
% For testing and running easy_anti_coeff
%xb = -1:0.5:0.5;
%xe = -0.5:0.5:1;
xb1 = -2.00:0.05:0.25;
xe1 = -1.95:0.05:0.30;
yb1 = 0.1*ones(size(xe1));
ye1 = 0.1*ones(size(xe1));
xb2 =  -0.30:0.05:1.95;
xe2 =  -0.25:0.05:2.00;
yb2 = -0.1*ones(size(xe2));
ye2 = -0.1*ones(size(xe2));
xb =[xb1 xb2];
xe =[xe1 xe2];
yb =[yb1 yb2];
ye =[ye1 ye2];
pyz = 0;
PXZ = 0;
PYZ = 1;
X=[];
Y=[];
[X,Y] = meshgrid(-1.95:0.1:1.95);

xi = 0.5*(xe+xb);
xj = xi;
yi = 0.5*(ye+yb);
yj = ye;
a = (xe-xb)/2;
aj = a;

G = 1;

[d,u_neg,u_pos,syz,UZ,SXZ] = easy_anti(G,pyz,xb,yb,xe,ye,PXZ,PYZ,X,Y);
figure(2)
hold on
plot(xb,yb,'r')
figure(3)
hold on
plot(xb,yb,'r')
figure(4)
hold on
plot(xb,yb,'r')
```

Listing of Matlab code easy_anti.m

```
function [d,u_neg,u_pos,syz,UZ,SXZ] = easy_anti(G,pyz,xb,yb,xe,ye,PXZ,PYZ,X,Y)
%  Two-dimensional displacement discontinuity boundary element script
%  For anti-plane strain elements parallel to x-axis with stress boundary conditions
%  Reference frame: x = vertical, y = horizontal, z = "into page"
% G = shear modulus                                      constant
%  pyz = shear traction on boundary elements,      [1xn] row vector or a constant
%  xb = starting element endpoint x-coordinate,    [1xn] row vector
%  yb = starting element endpoint y-coordinate,    [1xn] row vector
%  xe = ending element endpoint x-coordinate,      [1xn] row vector
%  ye = ending element endpoint y-coordinate,      [1xn] row vector
%  PXZ, PYZ = far-field (ambient) shear stress,    [1xn] row vector or a constant
%  X,Y = gridpoints (found using "meshgrid"),      [pxq] array
%  Last revised on 2/22/03

% FORMAT BOUNDARY ELEMENT ARRAYS AS ROW VECTORS (IN CASE THEY WEREN'T)
    xb = (xb(:))';    yb = (yb(:))';xe = (xe(:))'; ye = (ye(:))';
% CALCULATE BOUNDARY ELEMENT MIDPOINTS AND HALF-LENGTHS (lower case)
    xm       = (xb + xe)/2;      % element midpoints, [1xn] row vector
    ym       = (yb + ye)/2;      % element midpoints, [1xn] row vector
    a   =   xe-xm;            %  half-lengths,              [1xn] row vector
% ADJUST STRESS (TRACTION) BOUNDARY CONDITIONS TO ACCOUNT FOR AMBIENT FIELD
% (The remote stress will be added back at the end of the solution)
    bc = (pyz - PYZ)*ones(size(xb(:)));             % bc = [nx1] column vector
% COMPUTE INFLUENCE COEFFICIENT ARRAYS FOR ELEMENT-ELEMENT INTERACTONS
%  A(i,j) = effect at obs pt i due to a unit load at element j, A = [nxn] arrays
    [a_uz,a_sxz,a_syz] = easy_anti_coeff(G,xm,ym,xm,ym,a);
% Set self-influence displacement coefficients (on main diagonal of a_uz) to -0.5
    a_uz(find(eye(length(a_uz)))) = -0.5;
%  SOLVE [a_syz][d] = [bc] FOR DISPLACEMENT DISCONTINUITIES [d], [nxn]*[nx1]=[nx1]
    d = a_syz\bc;
% FIND TRACTIONS AND DISPLACEMENTS AT THE BOUNDARY ELEMENTS
    syz = a_syz * d + PYZ;    % Note that remote stress is added back in;
    u_neg = a_uz * d;         % Displacement on negative side of elements;
    u_pos = u_neg + d;                % Displacement on positive side of elements;
%  PLOT SLIP FIGURE
    figure(1);plot(xm,d,'r',xm,u_pos,'g',xm,u_neg,'b');    legend('Slip','u+','u-');
    xlabel('x');      ylabel('Slip or displacement');           title('Slip and
    Displacements');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if isempty(X), UZ = [], SXZ = [], return, end
% COMPUTE DISPLACEMENTS AND STRESSES AT SPECIFIED POINTS IN IN BODY (CAPS)
%  Calculate influence coefficient arrays for gridpoint-element pairs, A = [pxq,n]
%  Gridpoint arrays (X,Y) are sent to anti_coeff as column vectors.  X(:) = [pxq,1] = [m,1]
    [p,q] = size(X);
    [A_UZ,A_SXZ,A_SYZ] = easy_anti_coeff(G,X(:),Y(:),xm,ym,a);
%  Calculate the displacements and stress perturbation at gridpoints
    UZ  = A_UZ * d;                     %   [pxq,n]*[n,1]=[pxq,1]
    SXZ = A_SXZ * d + PXZ;          %   [pxq,n]*[n,1]=[pxq,1]
    SYZ = A_SYZ * d + PYZ;          %   [pxq,n]*[n,1]=[pxq,1]
%  Reshape the arrays into the matrices the size of X (i.e., pxq)
    UZ = reshape(UZ,p,q);
    SXZ = reshape(SXZ,p,q);
    SYZ = reshape(SYZ,p,q);
%  PLOT FIGURES FOR OBSERVATION POINTS
    figure(2);c1=contour (X,Y,UZ);    clabel(c1);  xlabel('x');  ylabel('y');
        axis('equal'); title  ('UZ');
    figure(3);c2=contour(X,Y,SXZ);    clabel(c2);  xlabel('x');  ylabel('y');
        axis('equal'); title('SXZ');
    figure(4);c3=contour(X,Y,SYZ);    clabel(c3);  xlabel('x');  ylabel('y');
        axis('equal'); title('SYZ');
```

Listing of Matlab code easy_anti_coeff.m

```
function [a_uz,a_Sxz,a_Syz] = easy_anti_coeff(G,xi,yi,xj,yj,aj)
% Calculates displacement and stress influence coefficients for the
% anti-plane strain displacement discontinuity script easy_anti.
% Displacement discontinuities have a unit magnitude slip = b.
% For anti-plane strain elements parallel to x-axis with stress boundary conditions
% Reference frame: x = horizontal, y = vertical, z = "out of page"
% A(i,j) = influence on observation pt i due to an effect at element j
% G = shear modulus
% xi = x-coordinate of point feeling the influence
% yi = x-coordinate of point feeling the influence
% xj = x-coordinate of midpoint of element causing the influence
% yj = y-coordinate of midpoint of element causing the influence
% Last revised on 2/18/03

b = 1;
b_2pi  =  b/(2*pi);
Gb_2pi       =  G*b/(2*pi);

% Set up intial matrices used to find influence coefficient matrices
% [Aij] [dj] = [bi],          [mxn]*[nx1]  =  [mx1]
% The influence coefficient matrices are mxn matrices, where
% m (the number of rows) is the # of observation points
% n (the number of columns) is the # of boundary elements
% The number of columns in A is the number of rows in [d]
% The number of rows in A is the number of rows in [b]
% Distances are from the jth element to the ith element
   m  =  length(xi);                % m = # of observation points
   n  =  length(xj);                % n = # of boundary elements
% All the matrices below have dimensions of mxn
xi  =  repmat(xi(:),1,n);           % Row i = xi (i = 1:m)
yi  =  repmat(yi(:),1,n);           % Row i = yi (i = 1:m)
xj  =  repmat((xj(:))',m,1);        % Column j = xj (j = 1:n)
yj  =  repmat((yj(:))',m,1);        % Column j = yj (j = 1:n)
aj  =  repmat((aj(:))',m,1);        % Column j = aj (j = 1:n)
% Contribution from "positive" end ("p")
xp     =  xi  -  (xj+aj);
yp     =  yi  -  yj;
rp2    =  xp.*xp  +  yp.*yp;
Syzp   =    Gb_2pi.*(xp./rp2);
Sxzp   =   -Gb_2pi.*(yp./rp2);
uzp    =  b_2pi*atan2(yp,xp);
% Contribution from "negative" end ("n")
xn     =  xi  -  (xj-aj);
yn     =  yi  -  yj;
rn2    =  xn.*xn  +  yn.*yn;
Syzn   =    Gb_2pi*(xn./rn2);
Sxzn   =   -Gb_2pi*(yn./rn2);
uzn    =  b_2pi*atan2(yn,xn);
% Total contribution for uz(y=0+) - uz(y=0+) = 1
% Note that -(Syzn - Syzp) = Syzp - Syzn
a_Syz  = Syzp - Syzn;
a_Sxz  = Sxzp - Sxzn;
a_uz   = uzp - uzn;
```

Listing of Matlab code twoddm.m, modified from Crouch and Starfield (1983)

```
function [d,snn,sns,us_neg,un_neg,us_pos,un_pos,UX,UY,SXX,SXY,SYY,S1,S2,tau,mean,theta] = ...
twoddm(k,G,PR,bvn,bvs,xb,yb,xe,ye,PXX,PXY,PYY,X,Y)
%  Two-dimensional boundary element MATLAB script with stress boundary conditions
%  This program is modified from Appendix B of the book
%  "Boundary element methods in Solid Mechanics"
%  by S.L. Crouch and A.M. Starfield, 1983
%  k = kappa = 3-4*PR for plane strain, or (3-PR)/(1+PR) for plane stress
%  G = shear modulus
%  PR = Poisson's ratio
%  bvn = normal traction on boundary elements,        [1xn] row vector or a constant
%  bvs = shear traction on boundary elements,         [1xn] row vector or a constant
%  xb = starting element endpoint x-coordinate,       [1xn] row vector
%  yb = starting element endpoint y-coordinate,       [1xn] row vector
%  xe = ending element endpoint x-coordinate,         [1xn] row vector
%  ye = ending element endpoint y-coordinate,         [1xn] row vector
%  PXX,PXY,PYY = far-field (ambient) stresses,        [1xn] row vector or a constant
%  X,Y = gridpoints (found using "meshgrid"),         [pxq] array, or []
%  Reference frames: Global Cartesian (XY), local Cartesian (xy or ns), and
%  local polar (rt).
%  Displacements and stresses at ELEMENTS designated by u and s, respectively.
%  Displacements and stresses at GRID PTS designated by U and S, respectively.
%  Pay careful attention to the bookkeeping and coordinate transformations
%  Last revised on 3/08/03
%  This approach can handle mutliple cracks and boundaries.  To do so it would
%  be convenient to arrange the input quantities in a matrix, with each row
%  corresponding to a particular crack or boundary condition type.

% FORMAT BOUNDARY ELEMENT ARRAYS AS [1xn] ROW VECTORS (IN CASE THEY WEREN'T)
    [k,l] = size(xb);
    xb = xb(:)';yb = yb(:)';    xe = xe(:)';    ye = ye(:)';
    bvs = bvs(:)';      bvn = bvn(:)';  PXX = PXX(:)'; PXY = PXY(:)'; PYY = PYY(:)';
% CALCULATE BOUNDARY ELEMENT MIDPOINTS, HALF-LENGTHS, AND ORIENTATIONS (lower case)
    n = length(xb);                           % # of elements
    xm = (xb + xe)/2;               % element midpoints,  [1xn] row vector
    ym = (yb + ye)/2;               % element midpoints,  [1xn] row vector
    xd = xe-xm;                     % delta x,            [1xn] row vector
    yd = ye-ym;                     % delta y,            [1xn] row vector
    a = sqrt(xd.^2 +yd.^2);         % half-lengths,       [1xn] row vector
%  Set up direction cosine matrices for reference frame transformations
%  Below, b = beta = thetaXx = orientation of element relative to global X-axis
%  The local x- and s-axes are parallel, and the local y- and n-axes are parallel
    sinb = yd./a;       % sine beta             % [1xn] row vector
    cosb = xd./a;       % cosine beta           % [1xn] row vector
    anX = -sinb;                                % [1xn] row vector
    asX = cosb;                               % [1xn] row vector
    anY = cosb;                               % [1xn] row vector
    asY = sinb;                               % [1xn] row vector

% ADJUST STRESS (TRACTION) BOUNDARY CONDITIONS TO ACCOUNT FOR AMBIENT FIELD
%  First transform the XY components of the remote stress field
%  to resolve the normal stress (Snn0) and shear stress (Sns0) ON each element
    PYX = PXY;
```

```
%     [1xn]  =  [1xn].*[1xn].*[1x1]  +  ...
    snn0 = anX.*anX.*PXX + anX.*anY.*PXY + anY.*anX.*PYX + anY.*anY.*PYY;
    sns0 = anX.*asX.*PXX + anX.*asY.*PXY + anY.*asX.*PYX + anY.*asY.*PYY;
%   Now subtract the resolved remote stress from the boundary stresses.
%   The remote stress will be added back at the end of the solution.
    bvs = bvs - sns0;                           % [1xn] row vector
    bvn = bvn - snn0;                           % [1xn] row vector

%  COMPUTE INFLUENCE COEFFICIENT (c) ARRAYS FOR ELEMENT-ELEMENT INTERACTONS
%   c(i,j) = effect at obs pt i due to a unit load at element j, c = [nxn] array
%   Find influence coefficients in the xy reference frame of the "j" elements
    anX=  repmat(anX,n,1);              % All n rows of anX are copies of each other
    asX=  repmat(asX,n,1);              % All n rows of anX are copies of each other
    anY=  repmat(anY,n,1);              % All n rows of anX are copies of each other
    asY=  repmat(asY,n,1);              % All n rows of anX are copies of each other
[uXs,uYs,uXn,uYn,sXXs,sXYs,sYYs,sXXn,sXYn,sYYn] = ...
    twoddm_coeff(G,k,PR,xm,ym,xm,ym,a,cosb,sinb);
   sYXs = sXYs;        sYXn = sXYn;
%   Convert XY coefficients to the ns reference frame of the "i" elements
%    Stress influence coefficients:   [nxn] = [nxn].*[nxn].*[nxn] + ...
    snsi_sj = anX.*asX.*sXXs + anX.*asY.*sXYs + anY.*asX.*sYXs + anY.*asY.*sYYs;
    snni_sj = anX.*anX.*sXXs + anX.*anY.*sXYs + anY.*anX.*sYXs + anY.*anY.*sYYs;
    snsi_nj = anX.*asX.*sXXn + anX.*asY.*sXYn + anY.*asX.*sYXn + anY.*asY.*sYYn;
    snni_nj = anX.*anX.*sXXn + anX.*anY.*sXYn + anY.*anX.*sYXn + anY.*anY.*sYYn;
%   Displacement influence coefficients:   [nxn] = [nxn].*[nxn].*[nxn] + ...
    usi_sj = asX.*uXs + asY.*uYs;       % [nxn] = [nxn]*[nxn] + [nxn]*[nxn]
    uni_sj = anX.*uXs + anY.*uYs;       % [nxn] = [nxn]*[nxn] + [nxn]*[nxn]
    usi_nj = asX.*uXn + asY.*uYn;       % [nxn] = [nxn]*[nxn] + [nxn]*[nxn]
    uni_nj = anX.*uXn + anY.*uYn;       % [nxn] = [nxn]*[nxn] + [nxn]*[nxn]

    clear anx asX anY asY;
%   Re-set self-influence ns displacement coefficients (on main diagonals)
    usi_sj(find(eye(n)))  =  -0.5;
    uni_sj(find(eye(n)))  =  0;
    usi_nj(find(eye(n)))  =  0;
    uni_nj(find(eye(n)))  =  -0.5;

%  SOLVE [c][d] = [b] FOR DISPLACEMENT DISCONTINUITIES [d]
%   First group the influence coefficient and boundary condition submatrices as shown:
%   |   [snsi_sj] [snsi_nj]  |      | [ds]    |        =        | [bs]    |
        [2nx2n]*[2nx1]=[2nx1]
%   |   [snni_sj] [snni_nj]  |      | [dn]    |                 | [bn]    |
    c  = [snsi_sj,  snsi_nj;  snni_sj,  snni_nj];         % c = influence coefficients;
    b  = [bvs(:);  bvn(:)];                               % b= Stress boundary conditions;
    d=  c\b;                                              clear c b;
%   Separate the vector d into subvectors ds and dn
    ds  = d(1:n);                % ds values are in upper half of d,      [nx1]
    dn  =  d(n+1:2*n);           % dn values are in lower half of d,      [nx1]

%  FIND ns TRACTIONS (STRESSES) AND DISPLACEMENTS AT THE BOUNDARY ELEMENTS
%   The ns tractions (stresses) should match the original boundary conditions
%   Solve for the ns displacements on the NEGATIVE sides of the elements
    us_neg  =  usi_sj*ds  +  usi_nj*dn;   %  [nx1]  =  [nxn]*[nx1]  +  [nxn].*[nx1]
    un_neg  =  uni_sj*ds  +  uni_nj*dn;   %  [nx1]  =  [nxn]*[nx1]  +  [nxn].*[nx1]
```

```
% Add the displacement discontinuity at each element to get
% the displacement components on the positive side of each element
    us_pos = us_neg + ds;              % [nx1] = [nx1] + [nx1]
    un_pos = un_neg + dn;              % [nx1] = [nx1] + [nx1]
% Find the tractions (stresses) in an ns reference frame, and add far-field back
%  [nx1] = [nxn]*[nx1] + [nxn]*[nx1] + [nx1]
    sns      = snsi_sj*ds + snsi_nj*dn + sns0';
    snn      = snni_sj*ds + snni_nj*dn + snn0';

% PLOT FIGURES FOR BOUNDARY ELEMENTS
    figure(1); plot(xm,dn,xm,un_pos,xm,un_neg,xm,ds,xm,us_pos,xm,us_neg);
    legend('dn','un+','un-','ds','us+','us-');
    xlabel('x'); ylabel('Slip or displacement');      title('Slip and Displacements');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
UX=[]; UY=[]; SXX=[]; SXY=[]; SYY=[]; S1=[]; S2=[]; tau=[]; mean=[]; theta=[];
if isempty(X), return, end
% COMPUTE DISPLACEMENTS AND STRESSES AT GRID POINTS (CAPS)
% Find influence coefficient arrays for gridpoint-element pairs, C = [pxq,n]
% Gridpoint arrays (X,Y) are sent as column vectors, e.g., X(:) = [pxq,1] = [m,1]
    [p,q] = size(X);
[UXs,UYs,UXn,UYn,SXXs,SXYs,SYYs,SXXn,SXYn,SYYn] =
    twoddm_coeff(G,k,PR,X(:),Y(:),xm,ym,a,cosb,sinb);
% Calculate the XY displacements and stress perturbation at gridpoints
% by multiplying the influence coefficients by the displacement discontinuities
    UX = UXs*ds + UXn*dn;                 % [pxq,n]*[n,1] + [pxq,n]*[n,1] = [pxq,1]
    UY = UYs*ds + UYn*dn;                 % [pxq,n]*[n,1] + [pxq,n]*[n,1] = [pxq,1]
    SXX      = SXXs*ds + SXXn*dn;         % [pxq,n]*[n,1] + [pxq,n]*[n,1] = [pxq,1]
    SXY      = SXYs*ds + SXYn*dn;         % [pxq,n]*[n,1] + [pxq,n]*[n,1] = [pxq,1]
    SYY      = SYYs*ds + SYYn*dn;         % [pxq,n]*[n,1] + [pxq,n]*[n,1] = [pxq,1]
% Add back the far-field stress (this would need to be modified for gravity)
    SXX      = SXX + PXX;                 % [pxq,1]
    SXY      = SXY + PXY;                 % [pxq,1]
    SYY      = SYY + PYY;                 % [pxq,1]
% Reshape the [pxq,1] vector arrays into matrices the size of X (i.e., pxq)
    UX = reshape(UX,p,q);
    UY = reshape(UY,p,q);
    SXX = reshape(SXX,p,q);
    SXY = reshape(SXY,p,q);
    SYY = reshape(SYY,p,q);
% Calculate quantities for principal stresses
    S1       = sig1(SXX,SYY,SXY);
    S2       = sig2(SXX,SYY,SXY);
    TAU      = taumax(SXX,SYY,SXY);
    MEAN     = ave(SXX,SYY,SXY);
    THETA    = angp(SXX,SYY,SXY); cosT = cos(THETA);        sinT = sin(THETA);
% NOTE: No check is made to find observation points within an element length
% of element midpoints. Such points could be found, but if those points are set
% to NaN before the stress trajectories are plotted, then the stress trajectory
% plotting routine will not run.
% PLOT FIGURES FOR OBSERVATION POINTS
    figure(2);  quiver(X,Y,UX,UY);
        xlabel('x');    ylabel('y');    axis('equal');  title  ('U');
    figure(3);  c3=contour(X,Y,SXX);  clabel(c3);
        xlabel('x');    ylabel('y');    axis('equal');  title('SXX');
    figure(4);  c4=contour(X,Y,SXY);  clabel(c4);
        xlabel('x');    ylabel('y');    axis('equal');  title('SXY');
    figure(5);  c5=contour(X,Y,SYY);  clabel(c5);
        xlabel('x');    ylabel('y');    axis('equal');  title('SYY');
    figure(6);  c6=contour(X,Y,S1);           clabel(c6);
        xlabel('x');    ylabel('y');    axis('equal');  title('S1');
    figure(7);  c7=contour(X,Y,S2);           clabel(c7);
        xlabel('x');    ylabel('y');    axis('equal');  title('S2');
    figure(8);  c8=contour(X,Y,TAU);  clabel(c8);
        xlabel('x');    ylabel('y');    axis('equal');  title('TAU');
    figure(9);  c9=contour(X,Y,MEAN);         clabel(c9);
        xlabel('x');    ylabel('y');    axis('equal');  title('MEAN');
    figure(10); traj2(X,Y,-sinT,cosT);
        xlabel('x');ylabel('y');  axis('equal');  title('S2  Trajectories');
```

Listing of Matlab function twoddm_coeff

```
function [UXs,UYs,UXn,UYn,SXXs,SXYs,SYYs,SXXn,SXYn,SYYn] = ...
twoddm_coeff(G,k,PR,Xi,Yi,Xj,Yj,aj,cosbj,sinbj)
%  This is the MATLAB function version of SUBROUTINE COEFF of TWODD,
%  the two-dimensional boundary element code of Crouch and Starfield.
%  It returns displacement (U__) and stress (S___) components
%  at points (Xi,Yi) in a global XY observation point array
%  due to unit positive shear and unit normal displacement discontinuities
%  at element j centered at (Xj,Yj).
%  Parameters G, k, and PR are elastic parameters (see Barber, 1992)
%  Parameters cosbj and sinbj give the orientation of the displacement
%  discontintuities in a GLOBAL XY reference frame.
%  The solution is based on equations of Tables 8.1 and 9.1 of Barber (1992)
%  for the following POLAR stress functions:
%   phi1 = r*logr*cost (mode  I dislocation)
%   phi2 = r*logr*sint (mode II dislocation)
%  IMPORTANT: In Barber's solutions, a dislocation (d) cut extends
%  from the origin to the right along the x-aXis (not the left).  So to form
%  a displacement discontintuity (dd) from x = -a to x = +a, dd = dn - dp, where
%  dn is the dislocation that extends right from the negative end of "dd",
%  and dp is the dislocation that extends right from the positive end of "dd".
%  As of 3/1/03, this seems to yield good solutions for mode I and mode II cracks.

% Set up intial matrices used to find influence coefficient matrices [A]
% [Aij] [dj] = [bi],           [mxn]*[nx1]  =  [mx1]
% The influence coefficient matrices [A] are mxn matrices, where
% m (the number of rows) is the # of observation points
% n (the number of columns) is the # of boundary elements
% The number of columns in A is the number of rows in [d]
% The number of rows in A is the number of rows in [b]
% Distances are from the jth element to the ith element
m      =  length(Xi);                 % m = # of observation points
n      =  length(Xj);                 % n = # of boundary elements
% Each matrix on the left side of the equations below has dimensions mxn
Xi     =  repmat(Xi(:),1,n);    % All n columns of Xi are copies of each other
Yi     =  repmat(Yi(:),1,n);    % All n columns of Yi are copies of each other
Xj     =  repmat(Xj(:)',m,1);          % All m rows of Xj are copies of each other
Yj     =  repmat(Yj(:)',m,1);          % All m rows of Yj are copies of each other
a      =  repmat(aj(:)',m,1);          % All m rows of a are copies of each other
% Set direction cosine matrices
axX=  repmat(cosbj(:)',m,1);                % All m rows of axX are copies of each other
axY=  repmat(sinbj(:)',m,1);                % All m rows of axY are copies of each other
ayX    = -axY;                              % All m rows of ayX are copies of each other
ayY    = axX;                               % All m rows of ayY are copies of each other

% Calculate coordinates of observation points in local xy reference frame
% centered at the element and oriented along the element.
% The transformation requires a translation and a rotation
% x(i,j) = x-distance of observation point Xi from element at point Xj
x  =  axX.*(Xi-Xj)  +  axY.*(Yi-Yj);  %  [mxn]=[mxn].*[mxn]  +  [mxn].*[mxn]
y  =  ayX.*(Xi-Xj)  +  ayY.*(Yi-Yj);  %  [mxn]=[mxn].*[mxn]  +  [mxn].*[mxn]
```

```
% Find geometric terms from positive (p) and negative (n) element ends to obs. pts.
xp      =  x-a;                          xn      = x+a;
rp      =  sqrt(xp.^2  +  y.^2);         rn      =  sqrt(xn.^2  +  y.^2);
costp   =  xp./rp;                       costn   =  xn./rn;
sintp   =  y./rp;                        sintn   =  y./rn;
tp      =  atan2(y,xp);                  tn      =  atan2(y,xn);
logrp   =  log(rp);                      logrn   =  log(rn);
axrp    =  xp./rp;                       axrn    =  xn./rn;
ayrp    =  y./rp;                        ayrn    =  y./rn;
axtp    =  -ayrp;                        axtn    =  -ayrn;
aytp    =  axrp;                         aytn    =  axrn;

%  Calculate the lead coefficients C3 and C4 (see Barber (1992), p. 169
%  These are defined to yield ux(+) - ux(-) >0 and uy(+) - uy(-) >0
C3u   =   -1/(pi*(k+1));            % Mode I dislocation displacement coefficient
C4u   =   +1/(pi*(k+1));            % Mode II dislocation displacement coefficient
C3s   =   -(2*G)/(pi*(k+1));        % Mode I dislocation stress coefficient
C4s   =   +(2*G)/(pi*(k+1));        % Mode II dislocation stress coefficient

%  Calculate polar DISPLACEMENT components in an element-end-based rt
%  reference frame due to unit SHEAR disp. discontinuity
%  for dislocations extending right from x = -a and x = +a.
%  NOTE:  A uniform shift does not need to be added to the displacements here
%  to "center" the disp. discontinuity because the contributions from the
%  positive and negative ends would cancel each other out
Ursp   =   C4u*0.5*(  -(k+1).*tp.*costp  -  sintp  +  (k-1).*logrp.*sintp  );
Ursn   =   C4u*0.5*(  -(k+1).*tn.*costn  -  sintn  +  (k-1).*logrn.*sintn  );
Utsp   =   C4u*0.5*(    (k+1).*tp.*sintp  +  costp  +  (k-1).*logrp.*costp  );
Utsn   =   C4u*0.5*(    (k+1).*tn.*sintn  +  costn  +  (k-1).*logrn.*costn  );
%  Convert displacements from rt coordinates to xy coordinates and superpose
Uxs    =  (axrn.*Ursn  +  axtn.*Utsn)  -  (axrp.*Ursp  +  axtp.*Utsp);
Uys    =  (ayrn.*Ursn  +  aytn.*Utsn)  -  (ayrp.*Ursp  +  aytp.*Utsp);

%  Calculate polar DISPLACEMENT components in an element-end-based rt
%  reference frame due to unit NORMAL disp. discontinuity
%  for dislocations extending right from x = -a and x = +a.
Urnp   =   C3u*0.5*(    (k+1).*tp.*sintp  -  costp  +  (k-1).*logrp.*costp);
Urnn   =   C3u*0.5*(    (k+1).*tn.*sintn  -  costn  +  (k-1).*logrn.*costn);
Utnp   =   C3u*0.5*(    (k+1).*tp.*costp  -  sintp  -  (k-1).*logrp.*sintp);
Utnn   =   C3u*0.5*(    (k+1).*tn.*costn  -  sintn  -  (k-1).*logrn.*sintn);
%  Convert displacements from rt coordinates to xy coordinates and superpose
Uxn    =  (axrn.*Urnn  +  axtn.*Utnn)  -  (axrp.*Urnp  +  axtp.*Utnp);
Uyn    =  (ayrn.*Urnn  +  aytn.*Utnn)  -  (ayrp.*Urnp  +  aytp.*Utnp);
```

```
%  Calculate polar STRESS components due to unit SHEAR disp. discontinuity
%       Positive end (x = +a)                   Negative end (x = -a)
        Srrsp  =  +C4s*sintp./rp;               Srrsn  =  +C4s*sintn./rn;
        Srtsp  =  -C4s*costp./rp;               Srtsn  =  -C4s*costn./rn;
        Strsp  = Srtsp;                         Strsn  = Srtsn;
        Sttsp  =  Srrsp;                        Sttsn  =  Srrsn;
%  Convert stresses from local rt coordinates to local xy coordinates
% Shear-mode dislocation, end at x = +a
[Sxxsp,Sxysp,Syxsp,Syysp]  =  stress_trans(Srrsp,Srtsp,Strsp,Sttsp,axrp,axtp,ayrp,aytp);
% Shear-mode dislocation, end at x = -a
[Sxxsn,Sxysn,Syxsn,Syysn]  =  stress_trans(Srrsn,Srtsn,Strsn,Sttsn,axrn,axtn,ayrn,aytn);
% Superpose dislocations to form displacement discontinuity from x = -a and x = +a.
Sxxs = Sxxsn - Sxxsp;        Sxys = Sxysn - Sxysp;    Syxs = Sxys;        Syys = Syysn -
Syysp;

%  Calculate polar rt STRESS components due to unit NORMAL displacement discontinuity
%       Positive end (x = +a)                   Negative end (x = -a)
        Srrnp  =  +C3s*costp./rp;               Srrnn  =  +C3s*costn./rn;
        Srtnp  =  -C3s*sintp./rp;               Srtnn  =  -C3s*sintn./rn;
        Strnp  = Srtnp;                                 Strnn  = Srtnn;
        Sttnp  =  Srrnp;                                Sttnn  =  Srrnn;
%  Convert stresses from local rt coordinates to local xy coordinates
% Normal-mode dislocation, end at x = +a
[Sxxnp,Sxynp,Syxnp,Syynp]  =  stress_trans(Srrnp,Srtnp,Strnp,Sttnp,axrp,axtp,ayrp,aytp);
% Normal-mode dislocation, end at x = -a
[Sxxnn,Sxynn,Syxnn,Syynn]  =  stress_trans(Srrnn,Srtnn,Strnn,Sttnn,axrn,axtn,ayrn,aytn);
% Superpose dislocations to form displacement discontinuity from x = -a and x = +a.
Sxxn = Sxxnn - Sxxnp;    Sxyn = Sxynn - Sxynp;    Syxn = Sxyn;  Syyn = Syynn - Syynp;

% Convert the displacements and Stresses from the xy frame to the XY frame
% These are the influence coefficients
% Rename the direction cosines to keep the mathematics and bookkeeping clear
aXx = axX; clear axX; aXy = ayX; clear ayX; aYx = axY; clear axY; aYy = ayY; clear ayY;
UXs = aXx.*Uxs + aXy.*Uys;                  UYs = aYx.*Uxs + aYy.*Uys;
UXn = aXx.*Uxn + aXy.*Uyn;                  UYn = aYx.*Uxn + aYy.*Uyn;
[SXXs,SXYs,SYXs,SYYs] = stress_trans(Sxxs,Sxys,Syxs,Syys,aXx,aXy,aYx,aYy);
[SXXn,SXYn,SYXn,SYYn] = stress_trans(Sxxn,Sxyn,Syxn,Syyn,aXx,aXy,aYx,aYy);
```

Listing of Matlab function sig1

```
function S1=sig1(SIGxx,SIGyy,SIGxy)
% function sig1.  Calculates one 2-D principal stress magnitude
S1 = (SIGxx+SIGyy)/2 + sqrt( ((SIGxx-SIGyy)/2).^2 + SIGxy.^2);
```

Listing of Matlab function sig2

```
function S2=sig2(SIGxx,SIGyy,SIGxy)
% function sig2.  Calculates the other 2-D principal stress magnitude
S2 = (SIGxx+SIGyy)/2 - sqrt( ((SIGxx-SIGyy)/2).^2 + SIGxy.^2);
```

Listing of Matlab function taumax

```
function tau=taumax(SIGxx,SIGyy,SIGxy)
tau = sqrt( ((SIGxx-SIGyy)/2).^2 + SIGxy.^2);
```

Listing of Matlab function ave

```
function mean=ave(SIGxx,SIGyy,SIGxy)
mean = (SIGxx+SIGyy)/2;
```

Listing of Matlab function angp

```
function theta=angp(SIGxx,SIGyy,SIGxy)
% function angp.  Calculates a principal stress orientation
theta = 0.5*atan2(SIGxy,(SIGxx-SIGyy)/2);
```

Listing of Matlab function angp

```
function [sxx,sxy,syx,syy] = stress_trans(srr,srt,str,stt,axr,axt,ayr,ayt)
% 2-D stress transformation function
sxx = axr.*axr.*srr + axr.*axt.*srt + axt.*axr.*str + axt.*axt.*stt;
sxy = axr.*ayr.*srr + axr.*ayt.*srt + axt.*ayr.*str + axt.*ayt.*stt;
syx = ayr.*axr.*srr + ayr.*axt.*srt + ayt.*axr.*str + ayt.*axt.*stt;
syy = ayr.*ayr.*srr + ayr.*ayt.*srt + ayt.*ayr.*str + ayt.*ayt.*stt;
```