WARP3D Reference Note Documentation

Yukun Li

Jul 22, 2018

Contents:

CHAPTER 1

didriv.f: Domain Integral DRIVer

1.1 Description

domain driver: drive the computation of j-integral values using the domain integral technique.

or:

drive the computation of mixed-mode stress intensity factors and t-stress using the interaction integral.

1.2 Calling Tree

```
c ***************************************************************
c *                                                             *
c *    didriv.f                                                 *
c *       -dickdm                                               *
c *       -diheadr                                              *
c *       -distup.f                                             *
c *       -di_cf_elem                                           *
c *       -dimrot.f                                             *
c *            -dimrot_coord_displ                              *
c *                 -di_trans_nodvals                           *
c *            -dimrott                                         *
c *                 -di1dsf.f                                   *
c *                 -difrts                                     *
c *       -di_calc_e33                                          *
c *       -di_calc_curvature                                    *
c *            -di_calc_coefficients                            *
c *       -difrar.f (di_front_q_area)                           *
c *            -di1dsf.f                                        *
c *       -di_node_props_setup                                  *
c *            -di_node_props                                   *
c *                 -di_fgm_alphas                              *
```

(continues on next page)

(continued from previous page)

```
c *                  -di_constant_alphas                          *
c *                  -di_seg_alpha_e_nu                           *
c *        -di_fgm_setup                                          *
c *             -di_nod_vals                                      *
c *                  -di_extrap_to_nodes                          *
c *                       -ndpts1.f                               *
c *                       -oulg1.f (oulgf)                        *
c *        -diexp4.f                                              *
c *             -diexp4                                           *
c *                  -diadit                                      *
c *                  -dibmck                                      *
c *             -diexp13                                          *
c *                  -diadit                                      *
c *                  -dibmck                                      *
c *                  -digete                                      *
c *        -dicmj.f                                               *
c *             -di_trans_nodvals                                 *
c *             -vec_ops (zero_vector.f)                          *
c *             -dielem                                           *
c *                  -dielem_a.f                                  *
c *                  -dielem_b.f                                  *
c *                  -dielem_c.f                                  *
c *        -di_write_std_out                                      *
c *        -di_write_packets                                      *
c *                                                               *
c ***************************************************************
```

1.3 Procedure

1. check validity of domain before starting computations.
2. perform exhaustive check on consistency of 1) number of front nodes, 2) front interpolation order, and 3) domain type.

call `dickdm`:

```
c ***************************************************************
c *                                                             *
c * domain_check  - exhaustive testing of domain defintiion for *
c *                 consistency                                 *
c *                                                             *
c ***************************************************************
```

3. allocate space for a q-value at each structure node. allocate space for expanded lists of coincident front nodes at each front location.
4. output header information for domain

call `diheadr`:

```
c ***************************************************************
c *                                                             *
c * domain_header - output info at start of domain computations *
c *                                                             *
c ***************************************************************
```

5. when q-values are given by the user, expand the compressed list of q-values into a list of length number of structure nodes.
6. set up the domain. for automatic domains, set q-values at front nodes. then find coincident front nodes and set their q-values.

call `distup.f`:

```
c ****************************************************************
c *                                                              *
c * set_up_domain  -- set up the domain for processing           *
c *                   find coincident front nodes and set their  *
c *                   q-values                                   *
c *                                                              *
c ****************************************************************
```

Output:

`j_data.q_values` REAL (:) ALLOCATABLE SAVE

7. allocate a vector of logicals and assign .true. for each element connected to a crack front node.

call `di_cf_elem`:

```
c ********************************************************************
c *                                                                  *
c * di_cf_elem - create a logical vector whose entries are .true. for  *
c *              elements incident on the crack tip, and .false. for   *
c *              those that are not. dicmj will use this info to set a *
c *              flag for each element that is analyzed by dielem. if  *
c *              a user includes the domain integral command           *
c *              'omit crack front elements for fgms yes', the flag    *
c *              will cause terms7 and 8 to be set to zero.            *
c *                                                                  *
c ********************************************************************
```

Output:

`j_data.crack_front_elem` LOGICAL (:) ALLOCATABLE SAVE

8. at point on front where integral is being computed, build the global->crack rotation matrix. gather coordinates and displacements of crack-front nodes, and rotate them to local crack-front system.

call `dimrot.f`:

```
c ********************************************************************
c *                                                                  *
c * dimrot - compute the 3x3 global -> crack front local rotation    *
c *                                                                  *
c ********************************************************************
```

Output:

`j_data.domain_origin` INTEGER `j_data.domain_rot(3,3)` DOUBLE PRECISION (3,3)

8c. calculate strain e33 at node at domain origin. this is for T-stress calculations using the interaction integral

call `di_calc_e33`:

```
c ****************************************************************
c *                                                              *
c *   calculate strain e33 at domain origin for T-stress calcs.  *
```

(continues on next page)

(continued from previous page)

```
c *   calculate strain e33 as the difference between the            *
c *   deformed and undeformed crack-front lengths delta_L / L       *
c *                                                                 *
c *******************************************************************
```

8c. calculate properties of a curve passing through the front nodes. these will be used to compute distance 'r' from integration points to a curved crack front.

call `di_calc_curvature` from

```
c *******************************************************************
c *                                                                 *
c *   calculate coefficients of curve described by crack front      *
c *   nodes.                                                        *
c *                                                                 *
c *******************************************************************
```

9. compute area under the q-function over that part of crack front for this domain. the area must be >0 else fatal error in domain (user forgot to set q-values on front nodes)

call `difrar.f`:

```
c *******************************************************************
c *                                                                 *
c * di_front_q_area - compute area under q-function along front for *
c *                   this domain                                   *
c *                                                                 *
c *******************************************************************
```

10. set logical flags to indicate if the nodal velocities and accelerations are all zero for this load step. if so, some later computations can be skipped.

11. Build the node average value of thermal expansion coefficient. for temperature-dependent material properties, also build the node average value of young's modulus and poisson's ratio. for temperature-independent material properties, values of e and nu are obtained within dicmj.f. nodal properties are needed for domain integral computations to compute spatial derivatives within the domain.

call `di_node_props_setup`:

```
c *******************************************************************
c *                                                                 *
c * di_node_props_setup - obtain alpha values at nodes. for         *
c *                       temperature-dependent properties, also    *
c *                       compute e and nu values at nodes. this    *
c *                       routine replaces di_expan_coeff_setup,    *
c *                       and the routine it calls, di_node_props,  *
c *                       replaces di_node_expan_coeff.             *
c *******************************************************************
```

12. Build the nodal averages of strain energy density (stress work density) and strains. These terms are used to calculate the derivative of the strain energy density, which appears in the domain integral when material properties vary spatially (e.g. fgms). The nodal values calculated in di_fgm_setup will be used to compute their spatial derivatives at integration points.

call `di_fgm_setup`:

```
c ***************************************************************
c *                                                             *
c * di_fgm_setup - allocate data structures for two terms used  *
c * in the calculation of the derivative of the stress work density.  *
c * these are: nodal values of stress work density and strain.  *
c *                                                             *
c ***************************************************************
```

12b. at point on front where integral is being computed, collect young's modulus and poisson's ratio. this assumes that all elements connected to this crack-front node have identical, homogeneous material properties, or that fgm material properties have been assigned to the model. for homogeneous material, "props" contains material data. for fgms, read data from "fgm_node_values." for temperature-dependent properties, segmental data arrays contain the properties.

13. if q-values given by user, we compute domain integral right now. otherwise, we set up a loop to generate q-values for automatic domains and their computation.

14. user wants automatic construction of domains.

14a. get last ring at which output will be printed. domains are always generated starting at ring 1 but j-values may not be computed for every domain.

14b. allocate arrays needed to support construction/definition of the domains. for type 4, we need only a nodal bit map. for types 1-3, we need two sets of nodal bit maps. each set has 3 maps of length to record all structure nodes. element list stored in the common vector.

14c. set up to accumulate statistics for computed domain values.

14d. loop over all domains. construct definition of the domain (q-values, element list). call driver to actually calculate value for domain.

call `diexp4.f`:

```
c ***************************************************************
c *                                                             *
c * domain expand 4 - expand type 4 automatic domain            *
c * domain expand 13 - expand type 1-3 automatic domain         *
c *                                                             *
c ***************************************************************
```

call `dicmj.f`:

```
c ***************************************************************
c *                                                             *
c * domain_compute - drive execution of element routine to      *
c *                  compute j and i-integrals for a single     *
c *                  domain                                     *
c *                                                             *
c ***************************************************************
```

14e. release allocatable arrays for automatic domains

15a. write j-integral and i-integral data to standard output

15b. write j-integral and i-integral data to packets

16. release arrays used for both user defined and automatic domains

CHAPTER 2

`distup.f`: Domain Integral SeT UP

2.1 Description

call `distup( c, crdmap, nonode, debug_driver, out )`

set_up_domain:

set up the domain for processing find coincident front nodes and set their q-values

2.2 Procedure

1. for automatic domain definitions, set the q-values at user specified front nodes based on the domain type and the user specified front interpolation order.
2. if linear interpolation of q-values over the domain is on, we also need to interpolate q-values along front for quadratic elements.
3. look at first 1 or 2 nodes specified to be on the crack front. compute a distance to be used for locating all other nodes conincident with each specified crack front node. set the "box" size for proximity checking as a fraction of the distance measure.
4. loop over each specified front node. locate all other nodes in model that are conincident with the front node. (use the box test). set the q-value of each coincident node to be same as user specified value for the specified front node. dump verification list if user requested it.

CHAPTER 3

`dimrot.f`: Domain Integral ROTation

3.1 Description

call `dimrot( c, crdmap, u, dstmap, debug_driver, out )`

compute the 3x3 global -> crack front local rotation

Output:

`j_data.domain_origin` INTEGER

`j_data.domain_rot(3,3)` DOUBLE PRECISION (3,3)

3.2 Calling Tree

```
c ****************************************************************
c *                                                              *
c *    dimrot.f                                                  *
c *        -dimrot_coord_displ                                   *
c *              -di_trans_nodvals                               *
c *        -dimrott                                              *
c *              -di1dsf.f                                       *
c *              -difrts                                         *
c *                                                              *
c ****************************************************************
```

3.3 Procedure

1. gather coordinates and displacements of crack-front nodes

2.1 if only one node on the crack front. we have the simple 2-D case. rotation matrix determined only by direction cosines. global-z and crack-z are assumed to coincide.

call `dimrot_coord_displ`

2.2 if more than one node on front specified. we have a 3-D situation. get the position of the front-node to be used as the domain origin. for domain types 'a', 'b' and 'c', (types 1, 2 and 3, respectively), take the domain origin as the location where the crack-front advance is maximum (where the q-value equals 1.0). for domain type 'd' (uniform crack advance) take the domain origin as the first front node.

3. based on the number of front nodes, interpolation order, and q-function type construct a unit tangent vector to the front that lies in the crack plane (crack-z direction). crack-y is given by direction cosines of normal to crack plane. construct crack-x and final 3x3 structure-to-crack rotation by cross-product.

3a. for q-function type 'd' (=4), we just use the first two nodes on the front to find the tangent vector. this case is for a uniform crack advance along the full front.

alternatively, the user may have defined components of the crack front tangent vector rather than using the computed value. this feature helps, for example, when the mesh has the crack front intersecting a symmetry plane at other than 90-degrees. A correct user-defined tangent vector restores domain independent J-values.

call `dimrott`

4. get crack front (unit) tangent vector (crack-z). compute crack-x unit vector from z and y. fill in global-to-crack rotation matrix.

call `dimrot_coord_displ`

3.4 Call `dimrot_coord_displ`

rotate coordinates of crack-front nodes to local system.

if necessary, transform nodal displacement values from constraint-compatible coordinates to global coordinates. then rotate displacements from global to local coordinates

call `di_trans_nodvals`

> subroutine to transform a 3x1 vector of constraint-compatible-coordinate-system values to global-coordinate-system values. the routine premultiplies the incoming constraint-compatible values by the transpose of the stored global-to-constraint coordinate system rotation matrix.

3.5 Call `dimrott`

unit tangent vector to front directed along crack local-Z direction

Output:

```
tvec(3)
```

```
c              compute or load unit tangent vector to the crack
c              front according to the situation indicated by the
c              case number or the user-defined flag.
c              for cases 2,5,6 we average two tangents
c              computed for adjacent line segments. for cases 1,3,4 we
c              compute a single tangent at start or end of line segment.
c              fnodes is list of structure nodes defining the segment(s).
c
```

(continues on next page)

(continued from previous page)

```
c              tangent_vector_defined = .T. --> the user specified the
c                                               tangent vector to use.
c                                               components are in
c                                               crack_front_tangent
c                                               skip computations.
c
c
c              case no.           situation
c              --------           ---------
c                 1               linear; 2-nodes on front
c                 2               linear; 3-nodes on front
c                 3               quadratic; 3-nodes on front
c                 4               cubic; 4-nodes on front
c                 5               quadratic; 5-nodes on front
c                 6               cubic; 7-nodes on front
c
c              fnctyp: 1-4, used for cases 3,4 to determine if
c                      tangent is needed for first or last node
c                      on front. = 1 (first node), =3 (last node).
c
c              status: returned value = 0 (ok) = 1 (bad data)
c
c              case 1,3,4: 2, 3 or four nodes on front but only a single
c                          element (linear, quadratic or cubic). compute
c                          tangent at first or last node (2,3 or 4). use
c                          utility routine to get derivatives of the 1-D
c                          isoparametric shape functions.
c
c              case 2,5,6: two linear, quadratic or cubic segments
c                          connecting 3, 5 or 6 front nodes.
c                          compute average tangent at center (common)
c                          node. compute two unit tangents, average
c                          components, then restore unit length.
```

3.6 Call `di1dsf.f`

calculates shape function values and derivatives for 1-dimension

Output:

`sf(3)` - shape function values

`dsf(3)` - derivatives of shape function

CHAPTER 4

di_calc_curvature: Domain Integral CALculate CURVATURE

4.1 Description

Domain Integral CALculate CURVATURE: calculate coefficients of curve described by crack front nodes.

```
c             determine if crack front is curved.
c             find coefficients of a quadratic expression defined by
c             three crack front nodes: x = az^2 + bz + c
c             if 'a' equals zero +/- tolerance, consider the front
c             nodes to be colinear, and exit.
c
c             for a circular crack, find center and radius of a
c             circle defined by the first three crack-front nodes.
```

Output:

j_data.crack_curvature DOUBLE PRECISION (7)

```
c             contents of array 'crack_curvature':
c
c             crack_curvature(1) = 0 for straight front
c                                = 1 for curved front
c             crack_curvature(2) = 0 for unknown curvature
c                                = 1 for for a circular curve
c             crack_curvature(3) = local x coordinate for circle
c                                = coefficient 'a' for polynomial
c             crack_curvature(4) = local z coordinate for circle
c                                = coefficient 'b' for polynomial
c             crack_curvature(5) = circle radius for circle
c                                = coefficient 'c' for polynomial
c             crack_curvature(6) = coefficient 'd' for polynomial
c             crack_curvature(7) = coefficient 'e' for polynomial
```

4.2 Calling Tree

```
c ****************************************************************
c *                                                              *
c *    di_calc_curvature                                         *
c *        -di_calc_coefficients                                 *
c *                                                              *
c ****************************************************************
```

curvature:

q = (x3 - x2) / ((z3 - z2) * (z3 - z1)) & - (x1 - x2) / ((z1 - z2) * (z3 - z1))

circular curve:

x_center = ((x1*x1 + z1*z1) * (z3 - z2) & + (x2*x2 + z2*z2) * (z1 - z3) & + (x3*x3 + z3*z3) * (z2 - z1)) & / (two * (x1 * (z3 - z2) & + x2 * (z1 - z3) & + x3 * (z2 - z1)))

z_center = ((x1*x1 + z1*z1) * (x3 - x2) & + (x2*x2 + z2*z2) * (x1 - x3) & + (x3*x3 + z3*z3) * (x2 - x1)) & / (two * (z1 * (x3 - x2) & + z2 * (x1 - x3) & + z3 * (x2 - x1)))

circle_radius = ((x1 - x_center)two** & + (z1 - z_center)**two)**half

3-node curve:

a = (x3 - x2) / ((z3 - z2) * (z3 - z1)) & - (x1 - x2) / ((z1 - z2) * (z3 - z1))

b = (x1 - x2 + a * (z2*z2 - z1*z1)) / (z1 - z2)

c = x1 - a * z1*z1 - b * z1

5-node curve:

call least-squares regression subroutine `di_calc_coefficients` to compute coefficients of polynomial.

4.3 Call `di_calc_coefficients`

subroutine to compute coefficients of a fourth-order polynomial by a linear least-squares regression of the five data points of the five crack-front nodes.

or:

solve a linear set of simultaneous equations using gauss elimination with partial pivoting. one rhs is permitted. the coefficient matrix may be non-symmetric. the coefficient matrix is replaced by it's triangulated form.

solves: [a] {x} = {b}:

```
c *        a       -- coefficient matrix of                      *
c *        x       -- vector to receive computed solution        *
c *        b       -- vector containing the right hand side      *
x = [e d c b a]
```

CHAPTER 5

`difrar.f`: Domain Integral FRont ARea

5.1 Description

di_front_q_area:

compute area under q-function along front for this domain

Output:

`j_data.front_q_area` DOUBLE PRECISION

`j_data.front_length` DOUBLE PRECISION

CHAPTER 6

di_node_props_setup: Domain Integral NODE PROPertieS SETUP

6.1 Description

```
call di_node_props_setup (1, nonode, nelblk )
```

di_node_props_setup:

obtain alpha values at nodes. for temperature-dependent properties, also compute e and nu values at nodes. this routine replaces di_expan_coeff_setup, and the routine it calls, di_node_props, replaces di_node_expan_coeff.

Output:

`j_data.count_alpha` INTEGER (nonode) ALLOCATABLE SAVE

`j_data.snode_alpha_ij` REAL (nonode,6) ALLOCATABLE SAVE

`j_data.seg_snode_e` REAL (nonode) ALLOCATABLE SAVE

`j_data.seg_snode_nu` REAL (nonode) ALLOCATABLE SAVE

`j_data.block_seg_curves` LOGICAL (nellk) ALLOCATABLE SAVE

`j_data.process_temperatures` LOGICAL

6.2 Calling Tree

```
c ***********************************************************************
c *                                                                     *
c *       -di_node_props_setup                                          *
c *            -di_node_props                                           *
c *                 -di_fgm_alphas                                      *
c *                 -di_constant_alphas                                 *
c *                 -di_seg_alpha_e_nu                                  *
```

(continues on next page)

(continued from previous page)

```
c *                                                                  *
c ********************************************************************
```

6.3 Call `di_node_props`

di_node_props:

> build average nodal values of material properties at each node in the model. all values are single-precision reals because double-precision accuracy is unnecessary.

loop through element blocks. check type of property assignment in each block.

```
property assignment: element nodes (fgm)    = 1
                     constant in element    = 2
                     temperature dependent = 3
```

1. check for fgm alphas. the identifier 'fgm_mark' for fgms assigned in inmat.f is -99.0. elprp.f then assigns this value to props(9,*), props(13,*) and props(34,*)

```
call di_fgm_alphas( span, first_elem, count_alpha, snode_alpha_ij )
c ********************************************************************
c *                                                                  *
c * di_fgm_alphas - retrieve nodal values of alpha assigned through  *
c *                 'fgm' input.                                     *
c *                                                                  *
c ********************************************************************
```

2. check for alphas that are constant throughout element. (this is check #2 because fgm assignment of alpha places '-99' in the props array.)

```
call di_constant_alphas( span, first_elem, count_alpha, snode_alpha_ij )
c ********************************************************************
c *                                                                  *
c * di_constant_alphas - retrieve element values of alpha assigned   *
c *                      through element definitions, and add values *
c *                      to nodal sums. the average nodal value is   *
c *                      then computed in the calling routine.       *
c *                                                                  *
c ********************************************************************
```

3. check for temperature-dependent material properties assigned using segmental curves. curve_set_type

```
curve_set_type =0, temperature-independent properties
               =1, temperature-dependent properties
               =2, strain-rate dependent properties
```

```
call di_seg_alpha_e_nu( curve_set, num_curves_in_set,
    &                   first_curve, curve_set_type, span, first_elem,
    &                   count_alpha, snode_alpha_ij,
    &                   seg_snode_e, seg_snode_nu )
c ********************************************************************
c *                                                                  *
c * di_seg_alpha_e_nu - retrieve nodal values of alpha, e and nu     *
c *                     assigned through temperature-dependent       *
```

(continues on next page)

(continued from previous page)

```
c *                     segmental curves. to simplify logic, nodal     *
c *                     alpha values are added to sum and averaged     *
c *                     in calling routine. e and nu values at nodes   *
c *                     are not averaged.                              *
c *                                                                    *
c ********************************************************************
```

CHAPTER 7

di_fgm_setup: Domain Integral FGM SETUP

7.1 Description

```
call di_fgm_setup ( 1, nonode, out, temperatures )
```

di_fgm_setup:

allocate data structures for two terms used in the calculation of the derivative of the stress work density. these are: nodal values of stress work density and strain.

Output:

`j_data.extrap_counts` INTEGER (nonode) ALLOCATABLE SAVE

`j_data.swd_at_nodes` (nonode) ALLOCATABLE SAVE

`j_data.strain_at_nodes` DOUBLE PRECISION (6,nonode) ALLOCATABLE SAVE

`j_data.fgm_e` LOGICAL

`j_data.fgm_nu` LOGICAL

7.2 Calling Tree

```
c ***************************************************************
c *                                                             *
c *       -di_fgm_setup                                         *
c *            -di_nod_vals                                     *
c *                 -di_extrap_to_nodes                         *
c *                      -ndpts1.f                              *
c *                      -oulg1.f (oulgf)                       *
c *                                                             *
c ***************************************************************
```

7.3 Call `di_nod_vals`

`call di_nod_vals( extrap_counts, swd_at_nodes, strain_at_nodes )`

build average nodal values of the strain and stress work density for all nodes in the model.

```
c                build average nodal values of the stress work density
c                (swd) and strains in the model. for each element,
c                shape functions will be used to extrapolate integration-
c                point values to each of the element's nodes where they
c                are then averaged.
```

7.4 Call `di_extrap_to_nodes`

subroutine to extrapolate strain and stress work density values from integration points to nodes for one element.

```
c                        loop over element nodes. for 8 and 20-noded
c                        hex elements using 2x2x2 integration, we
c                        extrapolate to the nodes using the lagrangian
c                        polynomials. otherwise we just average the
c                        integration-point values and use that value
c                        at every node. warp3d uses the following
c                        arguments to describe elements and integration
c                        order:
c
c                        etype = 1: 20-noded brick
c                        etype = 2:  8-noded brick
c                        etype = 3: 12-noded brick
c                        etype = 4: 15-noded brick
c                        etype = 5:  9-noded brick
c
c                        etype = 1, int_order = 1: 27 point rule (not used)
c                        etype = 1, int_order = 8:  8 point rule
c                        etype = 1, int_order = 9: 14 point rule
c                        etype = 2, int_order = 1:  8 point rule
c                        etype = 2, int_order = 2:  6 point rule (not used)
```

CHAPTER 8

diexp4.f: Domain Integral EXPand

8.1 Calling Tree

```
c ***************************************************************
c *                                                             *
c *    diexp4.f                                                 *
c *       -diexp4                                               *
c *            -diadit                                          *
c *            -dibmck                                          *
c *       -diexp13                                              *
c *            -diadit                                          *
c *            -dibmck                                          *
c *            -digete                                          *
c *                                                             *
c ***************************************************************
```

```
c ***********************************************************************
c *                                                                     *
c *   diadit -- add to bit map from specified item                      *
c *                                                                     *
c *   dibmck -- test for entry presence in a bit map                    *
c *                                                                     *
c *   digete -- get element edge data                                   *
c *                                                                     *
c ***********************************************************************
```

8.2 diexp4: Domain Integral EXPand domain_type 4

```
c ***************************************************************
c *                                                             *
c * domain expand 4 - expand type 4 automatic domain            *
```

(continues on next page)

(continued from previous page)

```
c *                                                                  *
c ********************************************************************
```

8.3 `diexp13`: Domain Integral EXPand domain_type 1-3

```
call diexp13( ring, nonode, noelem, incmap, iprops, incid,
 &                    out, bits, q_new_map, q_old_map, q_map_len )
c ********************************************************************
c *                                                                  *
c * domain expand 13 - expand type 1-3 automatic domain              *
c *                                                                  *
c ********************************************************************
```

convert list of coincident nodes at each corner node on the front to a bit map. there are 2 or three corner nodes. for 2 corner nodes we just leave one of the maps all zero which makes code logic simpler in later steps. set which positions in the list of front nodes are the corner nodes.

loop over all edges of this element. get the two structure nodes corresponding to the two corner nodes for the edge. classify the edge based on it's connectivity to nodes listed in the three node maps. build 3 logical flags for each corner node, marking it's appearance in the 3 node maps. based on connectivity of edge nodes on the 3 nodes lists, add a corner node to the lists.

```
logic tree:
0 = both corner nodes appear in the 3 node lists
0 = neither corner node appears in the 3 node lists
if corner node 1 only appears in the 3 lists,
    add corner node 2 to the node list in which corner
    node 1 appears.
if corner node 2 only appears in the 3 lists,
    add corner node 1 to the node list in which corner
    node 2 appears.
```

CHAPTER 9

dicmj.f: Domain Integral CoMpute J

9.1 Description

domain_compute:

drive execution of element routine to compute j and i-integrals for a single domain

9.2 Calling Tree

```
c ***************************************************************
c *                                                             *
c *       -dicmj.f                                              *
c *            -di_trans_nodvals                                *
c *            -vec_ops (zero_vector.f)                         *
c *            -dielem                                          *
c *                 -dielem_a.f                                 *
c *                 -dielem_b.f                                 *
c *                 -dielem_c.f                                 *
c *                                                             *
c ***************************************************************
```

9.3 Procedure

1. loop over all structure elements. if element is involved in integral computations, call element dependent routine.

- build copy of element coordinates, displacements, velocities, accelerations, q-values from global data. zero load terms that correspond to constrained dof. if we have temperature loadings, get temps for element nodes (node values + element uniform value). rotate displacements, velocities and accelerations from constraint-compatible to global coordinates. obtain element nodal values of stress work density and strain from structure node-value arrays.

  - `e_coord(3,num_enodes)`: element nodal coordinates
  - `snodes(num_enodes)`: element nodal numbers
  - `q_element(num_enodes)`: element nodal q-values
  - `e_displ(3,num_enodes)`: element nodal displacements
  - `e_vel(3,num_enodes)`: element nodal velocities
  - `e_accel(3,num_enodes)`: element nodal accelerations
  - `e(num_enodes)`: element nodal Young's modulus
  - `nu(num_enodes)`: element nodal Poisson's ratio
  - `e_force(num_enodes)`: element nodal force
  - `e_node_temps(num_enodes)`: element nodal temperatures
  - `e_alpha_ij(6,num_enodes)`: element nodal thermal expansion coefficients
  - `elem_nod_swd(num_enodes)`: element nodal values of stress work density
  - `elem_nod_strains(6,num_enodes)`: element nodal strains

- if necessary, transform nodal values from constraint-compatible coordinates to global coordinates.

```
call di_trans_nodvals( e_displ(1,enode), trnmat(snode)%mat(1,1))
call di_trans_nodvals( e_vel(1,enode), trnmat(snode)%mat(1,1))
call di_trans_nodvals( e_accel(1,enode), trnmat(snode)%mat(1,1))

c*****************************************************************
c                                                                *
c subroutine to transform a 3x1 vector of constraint-           *
c compatible-coordinate-system values to global-coordinate-     *
c system values. the routine premultiplies the incoming         *
c constraint-compatible values by the transpose of the stored   *
c global-to-constraint coordinate system rotation matrix.       *
c                                                                *
c*****************************************************************
```

- for problems with temperature loads, pull out temperatures of element nodes and set the thermal expansion coefficients at element nodes. remove reference temperatures of model nodes.
- crack-face tractions. the contribution to J arising from crack-face tractions is calculated element-by-element using the equivalent nodal loads on each element face computed during setup of load step for applied pressures, surface tractions and body forces. here we just get the element equiv. nodal forces. they are passed to element J routine which sorts out the loaded face. a similar procedure is used to compute the contribution to I.

```
call vec_ops( e_force,
 &              eq_node_forces(eq_node_force_indexes(elemno)),
 &              dummy, 3*num_enodes, 5 )

c     ****************************************************************
c     *                                                              *
c     *     perform simple vector-vector operations on single or     *
c     *     double precision vectors (based on the port)             *
c     *                                                              *
c     ****************************************************************
subroutine vec_ops( veca, vecb, vecc, n, opcode )
c             opcode v:   a = b
```

- determine if we really need to process element based on q-values, nodal velocities, accelerations and specified temperature loadings.
- gather element stresses, histories, properties and 3x3 rotation matrices from polar decompositions at each gauss point of element for geonl
  - `e_stress(nstrs * num_gpts)`: element stresses at gauss points
  - `elem_hist(hist_size * num_gpts)`: element histories at gauss points
  - `e_props(mxelpr)`: element properties
  - `e_rots(9*num_gpts)`: rotation matrices
- gather element strains at integration points. and store in tensor form. convert engineering (total) strains of off-diagonal terms to tensor strains.
  - `e_strain(9,num_gpts)`: element strains at gauss points in tensor form
- for crack-face loading, make a copy of the tractions input by the user in the domain definition. if none was input, dielem_c.f automatically uses the equivalent nodal loads used to solve the boundary value problem.

```
cf_load(1) = cf_tractions(1)
cf_load(2) = cf_tractions(2)
cf_load(3) = cf_tractions(3)
```

- call the element routine to compute contribution to the j-integral and or i-integral for domain

```
call dielem( e_coord, q_element, e_displ, e_vel, e_accel,
 &                 e_force, e_stress, elem_hist, e_props, e_props,
 &                 e_props, e_rots, hist_size, domain_rot, iout,
 &                 e_jresults, e_node_temps, elem_temps, e_alpha_ij,
 &                 ierr, element, debug_elements, one_point_rule,
 &                 geonl, numrow_sig, snodes, elem_nod_swd,
 &                 elem_nod_strains, omit_ct_elem, fgm_e, fgm_nu,
 &                 e_strain, e, e_front, nu, nu_front, front_nodes,
 &                 num_front_nodes, front_coords, domain_origin,
 &                 domain_type, front_order, e_iresults, comput_i,
 &                 comput_j, cf_traction_flags, cf_load,
 &                 front_elem_flag, expanded_front_nodes,
 &                 myid, numprocs, crack_curvature, face_loading,
 &                 seg_curves_flag, process_temperatures,
 &                 max_exp_front )
```

2. print values as required based on user specified flags.

2a. J-integral results: calculate stress intensity factor K, from J

2b. interaction-integral results: calculate K from relationship with interaction integral

```
c              the 8 terms for I-integral results are stored
c              in array e_iresults(8,7) as follows:
c
c                                value    auxiliary field
c
c                   iiterms(i,1): KI       plane stress
c                   iiterms(i,2): KI       plane stress
c                   iiterms(i,3): KII      plane stress
c                   iiterms(i,4): KII      plane stress
c                   iiterms(i,5): KIII     anti-plane shear
c                   iiterms(i,6): T11,T33  plane stress
```

(continues on next page)

(continued from previous page)

```
c                  iiterms(i,7): T11,T33  plane strain
c                  iiterms(i,8): T13      anti-plane strain
```

2c. compute J-values from K-values

3. print sum of values from all elements in domain, and J, I for domain

CHAPTER 10

dielem: Domain Integral ELEMent

```
c   defined in dielem_a.f
subroutine dielem ( coord, qvals, edispl, evel, eaccel,
 &                    feload, estress, history,
 &                    props, iprops, lprops, erots, nrow_hist,
 &                    rotate, iout, e_jresults, e_node_temps,
 &                    temperatures, enode_alpha_ij, ierr, elemno,
 &                    gdebug, one_point, geonl, numrows_stress,
 &                    snodes, elem_nod_swd, elem_nod_strains,
 &                    omit_ct_elem, fgm_e, fgm_nu, e_strain,
 &                    ym_nodes, ym_front_node, nu_nodes,
 &                    nu_front_node, front_nodes, num_front_nodes,
 &                    front_coords, domain_origin, domain_type,
 &                    front_order, e_iresults, comput_i, comput_j,
 &                    cf_traction_flags, cf_tractions,
 &                    front_elem_flag, expanded_front_nodes,
 &                    myid, numprocs, crack_curvature, face_loading,
 &                    seg_curves_flag, process_temperatures,
 &                    max_exp_front )

c   called by dicmj
call dielem( e_coord, q_element, e_displ, e_vel, e_accel,
 &                e_force, e_stress, elem_hist, e_props, e_props,
 &                e_props, e_rots, hist_size, domain_rot, iout,
 &                e_jresults, e_node_temps, elem_temps, e_alpha_ij,
 &                ierr, element, debug_elements, one_point_rule,
 &                geonl, numrow_sig, snodes, elem_nod_swd,
 &                elem_nod_strains, omit_ct_elem, fgm_e, fgm_nu,
 &                e_strain, e, e_front, nu, nu_front, front_nodes,
 &                num_front_nodes, front_coords, domain_origin,
 &                domain_type, front_order, e_iresults, comput_i,
 &                comput_j, cf_traction_flags, cf_load,
 &                front_elem_flag, expanded_front_nodes,
 &                myid, numprocs, crack_curvature, face_loading,
 &                seg_curves_flag, process_temperatures,
```

(continues on next page)

(continued from previous page)

```
&                 max_exp_front )
```

10.1 Description

- **domain integral for 3-d isoparametric elements. supports:**
  - finite strains-rotations
  - body forces (including inertia)
  - crack-face tractions
  - temperature loads
  - kinetic energy terms
  - anisotropic thermal expansion coefficients
  - nonhomogeneous material properties
- **interaction integral for 3-d isoparametric elements. supports:**
  - linear-elastic material behavior
  - crack-face tractions
  - nonhomogeneous material properties

```
c ****************************************************************
c *                                                              *
c *         element name      type no.         description       *
c *         ------------      --------         -----------       *
c *                                                              *
c *         q3disop              1          20 node brick(*)     *
c *         l3dsiop              2           8 node brick        *
c *         ts12isiop            3          12 node brick        *
c *         ts15isiop            4          15 node brick        *
c *         ts9isiop             5           9 node brick        *
c *                                                              *
c ****************************************************************
```

```
c ****************************************************************
c *                                                              *
c *           strain-stress ordering in warp3d vectors:          *
c *           ----------------------------------------           *
c *                                                              *
c *              eps-x, eps-y, eps-z, gam-xy, gam-yz, gam-xz     *
c *              sig-x, sig-y, sig-z, tau-xy, tau-yz, tau-xz     *
c *                                                              *
c ****************************************************************
```

10.2 Inputs

- `coord`: `e_coord(3,num_enodes)`: element nodal coordinates
- `qvals`: `q_element(num_enodes)`: element nodal q-values

- edispl: e_displ(3,num_enodes): element nodal displacements
- evel: e_vel(3,num_enodes): element nodal velocities
- eaccel: e_accel(3,num_enodes): element nodal accelerations
- feload: e_force(num_enodes): element nodal force
- estress: e_stress(nstrs * num_gpts): element stresses at gauss points
- history: elem_hist(hist_size * num_gpts): element histories at gauss points
- props: e_props(mxelpr) (=props(mxelpr,elemno)): element properties
- iprops: e_props(mxelpr) (=props(mxelpr,elemno)): element properties
- lprops: e_props(mxelpr) (=props(mxelpr,elemno)): element properties
- erots: e_rots(9*num_gpts): 3x3 rotation matrices from polar decompositions at each gauss point of element for geonl
- nrow_hist: hist_size: history size at each gauss point
- rotate: domain_rot(3,3): global -> crack front local rotation matrix
- iout: iout: output file handle
- e_node_temps: e_node_temps element nodal temperatures
- temperatures: elem_temps element temperatures
- enode_alpha_ij: e_alpha_ij(6,num_enodes): element nodal thermal expansion coefficients
- ierr: ierr error code
- elemno: element current element number
- gdebug: debug_elements: debug flags for domain integral computations
- one_point: one_point_rule: use one point rule flag

  .true. for use 1 point integration for 8-node elements
- geonl: geonl (=lprops(18,elemno)) geometric nonlinearity flag
- numrows_stress: numrow_sig (=nstrs=9) number of stress values per strain point in arrays passed by warp
- snodes: snodes(num_enodes): element nodal numbers
- elem_nod_swd: elem_nod_swd(num_enodes): element nodal values of stress work density
- elem_nod_strains: elem_nod_strains(6,num_enodes): element nodal strains
- omit_ct_elem: omit_ct_elem omit current element flag
- fgm_e: fgm_e 'fgm' values of e flag

  .true. for 'fgm' values of e have been assigned at the nodes
- fgm_nu: fgm_nu 'fgm' values of nu flag

  .true. for 'fgm' values of nu have been assigned at the nodes
- e_strain: e_strain(9,num_gpts): element strains at gauss points in tensor form
- ym_nodes: e(num_enodes): element nodal Young's modulus
- ym_front_node: e_front young's modulus at point on front for homogeneous material
- nu_nodes: nu(num_enodes): element nodal Poisson's ratio

- nu_front_node: nu_front poisson's ratio at point on front for homogeneous material
- front_nodes: front_nodes(num_front_nodes) numbers of front nodes
- num_front_nodes: num_front_nodes numbers of front nodes
- front_coords: front_coords(3,num_front_nodes) coordinates of crack-front nodes
- domain_origin: domain_origin position of domain origin in the list front_nodes
- domain_type: domain_type: domain type of q functions a-d
- front_order: front_order: order of crack front interpolation
- comput_i: comput_i: compute interaction inetgral flag

  .true. for compute interaction inetgral
- comput_j: comput_j: compute j inetgral flag

  .true. for compute j inetgral
- cf_traction_flags: cf_traction_flags: crack face traction flag

  .true. for compute crack face load terms
- cf_tractions: cf_load(3): crack face load
- front_elem_flag: front_elem_flag: front element flag

  .true. for current element is front element
- expanded_front_nodes: expanded_front_nodes:
- myid: myid:
- numprocs: numprocs:
- crack_curvature: crack_curvature(7): coefficients of curve described by crack front nodes. (see subroutine di_calc_curvature)
- face_loading: face_loading: flag
- seg_curves_flag: seg_curves_flag: temperature loading flag

  .true. for have temperature loading
- process_temperatures: process_temperatures: process temperatures flag

  .true. for temperature depended process
- max_exp_front: max_exp_front (=200): maximum number of front nodes list

10.3 Outputs

- e_jresults: e_jresults(8)
- e_iresults: e_iresults(8,8)

10.4 Calling Tree

```
c ****************************************************************
c *                                                              *
c *    (dielem_a.f)                                              *
c *       -dielem                                                *
c *           -digetr                                            *
c *           -diqmp1                                            *
c *           -getgpts (getgpts.f)                               *
c *           -derivs (derivs.f)                                 *
c *           -dielcj                                            *
c *           -dieler                                            *
c *           -digrad                                            *
c *           -dielrv                                            *
c *           -dieliq                                            *
c *           -dielrt                                            *
c *           -dippie                                            *
c *           -shapef (shapef.f)                                 *
c *           -dielav                                            *
c *           -di_calc_j_terms                                   *
c *                                                              *
c *    (dielem_b.f)                                              *
c *           -di_calc_r_theta                                   *
c *               -di_calc_distance                              *
c *           -di_calc_constitutive                              *
c *           -di_calc_aux_fields_k                              *
c *           -di_calc_aux_fields_t                              *
c *           -di_calc_i_terms                                   *
c *                                                              *
c *    (dielem_c.f)                                              *
c *           -di_calc_surface_integrals                         *
c *               -dielwf                                        *
c *               -dielrl                                        *
c *               -di_calc_surface_j                             *
c *               -di_calc_surface_i                             *
c *                   -di_reorder_nodes                          *
c *                                                              *
c ****************************************************************
```

10.5 Procedure

- set up basic element properties. load all six thermal expansion coefficients from material associated with the element. if they are all zero, we skip processing of element for temperature loading. all temperature terms in domain involve derivatives of temperature at gauss points and expansion coefficients at gauss points. if specified coefficients for the element are zero, we skip temperature processing of elements.
- for 'fgm' alpha values, elem_alpha(1)-elem_alpha(3) will receive a value of -99.0. in the loop over integration points, subroutine dielav changes elem_alpha to the value of alpha at the integration point by interpolating nodal alpha values. to be consistent with the solution of the boundary-value problem, for linear-displacement elements, dielav will assign to elem_alpha the average of nodal alpha values.
- set integration order info for volume integrals.
- set number of stress values per strain point in arrays passed by warp
- set up data needed at all integration points and nodes needed for the volume integrals and crack face traction integrals:

1. transform unrotated Cauchy stresses to Cauchy stresses for geonl formulation. estress on input contains unrotated stresses. first convert to Cauchy stresses then to 1 st PK stresses. these are in global coordinates. note: 1 PK stresses are non-symmetric so we keep the 3x3 tensor stored as a vector. position 10 is for the work density. the work density is scaled by det F to refer to t=0 configuration.

2. for small-displacement formulation, copy the input (symmetric) stresses (6x1) into 3x3 tensor form.

```
do ptno = 1, ngpts
    loc = ( ptno-1 )  * numrow
    stresses(1,ptno)  = estress(loc+1)
    stresses(2,ptno)  = estress(loc+4)
    stresses(3,ptno)  = estress(loc+6)
    stresses(4,ptno)  = estress(loc+4)
    stresses(5,ptno)  = estress(loc+2)
    stresses(6,ptno)  = estress(loc+5)
    stresses(7,ptno)  = estress(loc+6)
    stresses(8,ptno)  = estress(loc+5)
    stresses(9,ptno)  = estress(loc+3)
    stresses(10,ptno) = estress(loc+7)
end do
```

3. copy engineering strains into 3x3 tensor form. change gamma strains to tensor strains for off-diagonal terms. this formulation is not correct for large strains.

```
do enode = 1, nnode
    strains(1,enode) = elem_nod_strains(1,enode)
    strains(2,enode) = elem_nod_strains(4,enode)/2.0
    strains(3,enode) = elem_nod_strains(6,enode)/2.0
    strains(4,enode) = elem_nod_strains(4,enode)/2.0
    strains(5,enode) = elem_nod_strains(2,enode)
    strains(6,enode) = elem_nod_strains(5,enode)/2.0
    strains(7,enode) = elem_nod_strains(6,enode)/2.0
    strains(8,enode) = elem_nod_strains(5,enode)/2.0
    strains(9,enode) = elem_nod_strains(3,enode)
end do
```

4. rotate nodal coordinates, displacements, velocities and accelerations to crack front normal coordinate system.

call `dielrv`: Domain Integral ELement Rotate Vector

```
call dielrv( coord, cdispl, cvel, caccel, edispl, evel, eaccel,
 &             rotate, nnode, iout, debug )
c ****************************************************************
c *                                                              *
c *   rotate nodal vector values to crack x-y-z                  *
c *                                                              *
c ****************************************************************
```

- `cdispl(3,20)` nodal displacements at crack x-y-z
- `cvel(3,20)` nodal velocities at crack x-y-z
- `caccel(3,20)` nodal accelerations at crack x-y-z

5. modify the nodal q-values to reflect linear interpolations for 20, 15, 12-node elements.

call `dieliq`: Domain Integral ELement Interpolate Q-values

```
call dieliq( qvals, debug, iout, nnode, etype, elemno, snodes,
 &             coord, front_elem_flag, num_front_nodes, front_nodes,
```

(continues on next page)

(continued from previous page)

```
 &              front_coords, expanded_front_nodes, domain_type,
 &              domain_origin, front_order, qp_node, crack_curvature,
 &              max_exp_front)
c ******************************************************************
c *                                                                *
c *          interpolate q-values at side nodes                    *
c *                                                                *
c *          if any 1/4-point node is detected, special integration *
c *          for crack-face traction on crack-front element faces  *
c *          is not used.                                          *
c *                                                                *
c ******************************************************************
```

6. compute coordinate jacobian at all gauss points and the center point of element. these are now in cracked coordinates.

6a. isoparametric coordinates of gauss point

```
call getgpts( etype, order, ptno, xsi, eta, zeta, weight )
c     ****************************************************************
c     *                                                              *
c     *     given the element type, order of integration and the     *
c     *     integration point number, return the parametric          *
c     *     coordinates for the point and the weight value for       *
c     *     integration                                              *
c     *                                                              *
c     ****************************************************************

call derivs( etype, xsi, eta, zeta, dsf(1,1,ptno),
 &               dsf(1,2,ptno), dsf(1,3,ptno) )
c     ****************************************************************
c     *                                                              *
c     *     given the element type, integration point coordinates    *
c     *     return the parametric derivatives for each node          *
c     *     shape function                                           *
c     *                                                              *
c     ****************************************************************
```

6b. coordinate jacobian, it's inverse, and determinant.

call `dielcj`: Domain Integral ELement Compute Jacobian

```
call dielcj( dsf(1,1,ptno), coord, nnode, jacob(1,1,ptno),
 &               jacobi(1,1,ptno), detvol(ptno), ierr, iout, debug )
c ******************************************************************
c *                                                                *
c *    compute coordinate jacobian, its determinate, and inverse   *
c *                                                                *
c ******************************************************************
```

- `jacob(3,3,28)` coordinate jacobian matrix at integration points
- `jacobi(3,3,28)` coordinate inverse jacobian matrix at integration points
- `detvol(28)` determinant of jacobian matrix at integration points

7. rotate stresses and strains to crack-front coordinates at all gauss points. get the stress work density from warp results.

call `dielrt`: Domain Integral ELement Rotate Tensor

```
call dielrt( ptno, rotate, stresses(1,ptno), csig(1,ptno),
 &               1, iout, debug )
call dielrt( ptno, rotate, e_strain(1,ptno), ceps_gp(1,ptno),
 &               2, iout, debug )
c ********************************************************************
c *                                                                  *
c *  rotate stresses or strains from global->crack x-y-z             *
c *                                                                  *
c ********************************************************************
```

- `csig(10,27)`: stress in crack-front coordinates at gauss points
- `ceps_gp(9,27)`: strain in crack-front coordinates at gauss points

8. rotate strains to crack-front coordinates at all nodes.

```
call dielrt( enode, rotate, strains(1,enode), ceps_n(1,enode),
 &        3, iout, debug )
```

- `ceps_n(9,20)`: nodal strain in crack-front coordinates

9. process temperature (inital) strains if required. rotate the thermal expansion coefficients from from global->crack coordinates in case they are anisotropic. elem_alpha(1:6) in global is replaced by elem_alpha(1:6) in crack. rotate values that are constant over the element (elem_alpha) and values at each node of the element (enode_alpha_ij). (nodal values enable computation of the x1 derivative of the alpha_ij term in J.) for isotropic CTEs this is some unnecessary work.

```
call dippie( rotate, iout, debug, elem_alpha )
c ********************************************************************
c *                                                                  *
c *  set up to process temperature effects for domain integral       *
c *                                                                  *
c ********************************************************************
```

10. for single point integration of bricks, compute average stresses, energy density, and strain at center of element. set number of gauss points to one to control subsequent loop.

- loop over all gauss points and compute each term of the domain integral for j, and the interaction integral for i.

```
c              jterm(1) :  work density
c              jterm(2) :  traction - displacement gradient
c              jterm(3) :  kinetic energy denisty
c              jterm(4) :  accelerations
c              jterm(5) :  crack face loading (handled separately)
c              jterm(6) :  thermal strain (2 parts. set to zero for an fgm)
c              jterm(7) :  stress times partial of strain wrt x1 (for fgms)
c              jterm(8) :  partial of stress work density wrt x1 (for fgms)
c              (jterms 7 & 8 are used to replace the explicit partial
c               derivative of stress work density wrt x1 (for fgms))
c
c              iterm(1) :  stress * derivative of aux displacement
c              iterm(2) :  aux stress * derivative of displacement
c              iterm(3) :  mixed strain energy density (aux stress * strain)
c              iterm(4) :  first term of incompatibility (stress * 2nd deriv
c                          of aux displacement
c              iterm(5) :  second term of incompatibility (stress * deriv
c                          of aux strain)
```

(continues on next page)

(continued from previous page)

```
c              iterm(6) :  deriv of constitutive tensor * strain * aux strain
c              iterm(7) :  (not yet verified)
c                          aux stress * deriv of cte * relative change in temp
c                        + aux stress * cte * deriv of relative change in temp
c              iterm(8) :  crack face traction * aux disp. derivative
```

1. isoparametric coordinates of gauss point and weight.

```
call getgpts( etype, order, ptno, xsi, eta, zeta, weight )
```

2. evaluate shape functions of all nodes at this point. used to find value of q function at integration point, the total velocity at the point, values of acceleration at the point, and alpha values at the point.

```
call shapef( etype, xsi, eta, zeta, sf )
```

call `dielav`: Domain Integral ELement gAuss point Values

```
call dielav( sf, evel, eaccel, point_velocity, point_accel_x,
 &               point_accel_y, point_accel_z, qvals, point_q,
 &               nnode, point_temp, e_node_temps, elem_alpha,
 &               enode_alpha_ij, linear_displ, fgm_alphas, elemno,
 &               ym_nodes, nu_nodes, point_ym, point_nu, fgm_e,
 &               fgm_nu, coord, point_x, point_y, point_z,
 &               seg_curves_flag, iout )
c ********************************************************************
c *                                                                  *
c *   get q, velocity, accelerations, temperature and alpha at       *
c *   gauss point.                                                   *
c *                                                                  *
c ********************************************************************
```

- `point_q`: q-valuse at gauss point
- `point_velocity`: total velocity at gauss point
- `kin_energy`: kinetic energy at gauss point
- `point_accel_x, point_accel_y, point_accel_z`: accelerations at gauss point
- `point_temp`: temperature at gauss point
- `elem_alpha(6)`: alpha at gauss point
- `point_ym`: young's modulus at gauss point
- `point_nu`: poisson's ratio at gauss point
- `point_x, point_y, point_z`: coordinates of gauss point in the local crack-front system

3. for this integration point, compute displacement derivatives, q-function derivatives and temperature derivative in crack coordinate system. strains will be used in the order eps11, eps22, eps33, eps12, eps23, eps13

- `dux, dvx, dwx`: displacement derivatives with respect to x in crack coordinate system
- `dqx, dqy, dqz`: q-function derivatives in crack coordinate system
- `dtx`: temperature derivative with respect to x in crack coordinate system
- `dalpha_x1(6)`: alpha derivative with respect to x in crack coordinate system
- `dswd_x1`: stress work density derivative with respect to x in crack coordinate system

- `dstrain_x1(9)`: nodal strain derivative with respect to x in crack coordinate system
- `csig_dstrain_x1`:
- `de_x1`: young's modulus derivative with respect to x in crack coordinate system
- `dnu_x1`: poisson's ratio derivative with respect to x in crack coordinate system

4. call routines to compute domain-integral terms for current integration point.

4a. compute j terms

```
call di_calc_j_terms(weight, evol, jterm, ptno, dqx,
 &                       dqy, dqz, dux, dvx, dwx, dtx, csig,
 &                       kin_energy, rho, point_q, point_accel_x,
 &                       point_accel_y, point_accel_z, point_temp,
 &                       process_temperatures, elem_alpha,
 &                       dalpha_x1, csig_dstrain_x1, dstrain_x1,
 &                       dswd_x1, omit_ct_elem, fgm_e, fgm_nu,
 &                       elemno, myid, numprocs, seg_curves_flag,
 &                       debug, iout)
```

4b. compute i terms

- perform edi evaluations for traction loaded faces for jterm(5) and then for iterm(8).

```
call di_calc_surface_integrals( elemno, etype, nnode, snodes,
 &                              feload, cf_traction_flags,
 &                              cf_tractions, rotate, dsf, jacobi,
 &                              cdispl, qvals, coord, front_nodes,
 &                              front_coords, domain_type,
 &                              domain_origin, num_front_nodes,
 &                              front_order, ym_front_node,
 &                              nu_front_node, comput_j,
 &                              comput_i, jterm, iterm,
 &                              front_elem_flag, qp_node,
 &                              crack_curvature, face_loading, iout,
 &                              debug)
c*********************************************************************
c                                                                    *
c    subroutine to drive computation of surface-traction             *
c    integrals                                                       *
c*********************************************************************
```

- save edi values in result vectors

10.6 Compute i terms

1. call `di_calc_r_theta`: Domain Integral CALCulate Radius and angle THETA

    - for straight element edges:

    compute distance from integration point to the closest line that connects two adjacent front nodes.

    compute angle between integration point, line connecting front nodes, and projection of integration point onto crack plane.

    - for curved element edges:

compute distance from integration point to the curve fitted through adjacent front nodes.

compute angle between integration point, curve, and projection of integration point onto crack plane.

```
call di_calc_r_theta( 2, front_nodes, num_front_nodes,
 &                       front_coords, domain_type, domain_origin,
 &                       front_order, point_x, point_y, point_z,
 &                       elemno, ptno, r, t, crack_curvature,
 &                       debug, iout )
c ****************************************************************
c                                                                *
c subroutine to calculate radius "r", and angle "theta" of       *
c a single point, measured in polar coordinates in the           *
c local crack-front system.                                      *
c                                                                *
c ****************************************************************
```

- `r`: radius “r” in polar coordinates in the local crack-front system
- `t`: angle “theta” in polar coordinates in the local crack-front system

2. call `di_calc_constitutive`: Domain Integral CALCulate CONSTITUTIVE tensor components

```
call di_calc_constitutive( dcijkl_x1, sijkl, dsijkl_x1,
 &                           point_ym, point_nu, de_x1, dnu_x1,
 &                           elemno, debug, iout )
c***************************************************************
c                                                               *
c    subroutine to calculate constitutive tensors for           *
c    interaction integral                                       *
c                                                               *
c***************************************************************
```

- `dcijkl_x1(3)`: constitutive tensor derivative
- `sijkl(3)`: compliance tensor
- `dsijkl_x1(3)`: compliance tensor derivative

```
c              the three non-zero components of cijkl are:
c                 (1) lambda + 2*mu = e(1-nu)/((1+nu)(1-2nu))
c                     d(1)/de  = (1-nu)/((1+nu)(1-2nu))
c                     d(1)/dnu = 2e*nu(2-nu)/((1+nu)^2*(1-2nu)^2)
c                 (2) lambda = e*nu/((1+nu)(1-2nu))
c                     d(2)/de  = nu/((1+nu)(1-2nu))
c                     d(2)/dnu = e(1+2nu^2)/((1+nu)^2*(1-2nu)^2)
c                 (3) 2*mu = e/(1+nu)
c                     d(3)/de  = 1/(1+nu)
c                     d(3)/dnu = -e/(1+nu)^2
c
c              the three non-zero components of sijkl are:
c                 (1) d(1) = 1/e
c                     d(1)/de = -1/e^2       d(1)/dnu = 0
c                 (2) d(2) = -nu/e
c                     d(2)/de = nu/e^2       d(2)/dnu = -1/e
c                 (3) d(3) = (1+nu)/e
c                     d(3)/de = -(1+nu)/e^2  d(3)/dnu = 1/e
```

3. call `di_calc_aux_fields_k`: Domain Integral CALCulate AUXiliary FIELDS for stress intensity factors K

```
call di_calc_aux_fields_k( elemno, ptno, r, t, ym_front_node,
 &                            nu_front_node, dcijkl_x1, sijkl,
 &                            dsijkl_x1, aux_stress,
 &                            aux_strain, daux_strain_x1,
 &                            du11_aux,  du21_aux,  du31_aux,
 &                            du111_aux, du112_aux, du113_aux,
 &                            du211_aux, du212_aux, du213_aux,
 &                            du311_aux, du312_aux, du313_aux,
 &                            iout )
c*****************************************************************
c                                                                *
c subroutine to calculate auxiliary stresses and displacements *
c at element integration points using the expressions obtained *
c by Williams: On the stress distribution at the base of a     *
c stationary crack. Journal of Applied Mechanics 24, 109-114,  *
c 1957.                                                        *
c                                                                *
c*****************************************************************
```

- `aux_stress(9,8)`: auxiliary stresses at integration point
- `aux_strain(9,8)`: auxiliary strains at integration point
- `daux_strain_x1(9,8)`: auxiliary strains derivatives
- `du111_aux(8)`,`du112_aux(8)`,`du113_aux(8)`,`du211_aux(8)`, `du212_aux(8)`, `du213_aux(8)`,`du311_aux(8)`,`du312_aux(8)`,`du313_aux(8)`: second derivatives of displacement (uj,1i)

4. call `di_calc_aux_fields_t`: Domain Integral CALCulate AUXiliary FIELDS for T-stresses

```
call di_calc_aux_fields_t( elemno, ptno, r, t, ym_front_node,
 &                            nu_front_node, dcijkl_x1, sijkl,
 &                            dsijkl_x1, aux_stress,
 &                            aux_strain, daux_strain_x1,
 &                            du11_aux,  du21_aux,  du31_aux,
 &                            du111_aux, du112_aux, du113_aux,
 &                            du211_aux, du212_aux, du213_aux,
 &                            du311_aux, du312_aux, du313_aux,
 &                            iout )
c*****************************************************************
c                                                                *
c subroutine to calculate auxiliary stresses and displacements *
c at integration points for t-stresses using the expressions   *
c obtained by Michell for a point load on a straight crack:    *
c (see Timoshenko and Goodier, Theory of Elasticity p. 110.,   *
c  eq. 72 for stress sigma_r.)                                 *
c                                                                *
c*****************************************************************
```

5. call `di_calc_i_terms`: Domain Integral CALCulate Interaction integral terms for stress intensity factors

```
call di_calc_i_terms( ptno, dqx, dqy, dqz, dux, dvx, dwx,
 &                        dtx, csig, aux_stress, ceps_gp,
 &                        aux_strain, dstrain_x1,
 &                        daux_strain_x1, dcijkl_x1,
 &                        du11_aux,  du21_aux,  du31_aux,
 &                        du111_aux, du211_aux, du311_aux,
 &                        du112_aux, du212_aux, du312_aux,
```

(continues on next page)

(continued from previous page)

```
     &                        du113_aux, du213_aux, du313_aux,
     &                        process_temperatures, elem_alpha,
     &                        dalpha_x1, point_temp, point_q, weight,
     &                        elemno, fgm_e, fgm_nu, iterm,
     &                        iout, debug)
      c*****************************************************************
      c                                                                *
      c    subroutine to calculate terms of i-integral                 *
      c                                                                *
      c*****************************************************************
```

Reference:

- warp3d.net : WARP3D User Manual
- WARP3D - GitHub