FEM formulation for hydrostatic stress gradient

* 1. Abaqus continuum coupled temperature elements

In Abaqus, the element selection is crucial to the simulation workflow. Meshing in FEA software usually employ the hexahedral or tetrahedral shape element, and this thesis prioritizes using the hexahedral element shape. This is because hexahedral elements tend to perform better than tetrahedral elements for a wide range of simulation conditions, contact and BCs [1].

Additionally, element types are available for diverse analyses, such as 3D stress, coupled temperature displacement, thermal, and cohesive, etc. For coupled mechanical loading – heat transfer analysis, the element must be of type coupled temperature displacement. Since the thesis aims to benchmark the 2D and 3D case studies, both 2D plane strain and 3D solid brick temperature-displacement elements for this analysis are covered in Table 5.

Table 5. Different 2D and 3D coupled temperature displacement elements

|  |  |  |
| --- | --- | --- |
|  | 2D plane strain elements | 3D solid brick elements |
| Linear (1st order),  full integration | CPE4T (4 NPs, 4 IPs) | C3D8T (8 NPs, 1 IPs) |
| Linear (1st order),  reduced integration | CPE4RT (4 NPs, 1 IP) | C3D8RT (8 NPs, 8 IPs) |
| Quadratic (2nd order),  full integration | CPE8T (8 NPs, 4 IPs) | C3D20T (20 NPs, 8 IPs) |
| Quadratic (2nd order),  reduced integration | CPE8RT (8 NPs, 9 IPs) | C3D20RT (20 NPs, 27 IPs) |

In FEM, reduced integration means this element is using fewer IPs than required for fully integrating the stiffness matrix, which reduces computational cost and can help avoid numerical locking in certain problems. However, reduced integration can lead to a phenomenon called hourglassing, where spurious zero-energy deformation modes occur that makes the element bend unrealistically in a shape like an hourglass, which significantly reduces the accuracy of the numerical solution [2]. Full integration usually bypasses all these problems, except that linear full integration elements may suffer from volumetric locking. More information about this problem shall be discussed in the Annex E in the Appendix for interested readers.

Additionally, it is needed to define the terms “linear” and “quadratic”. For linear elements, they have only NPs at their vertices, and their interpolation functions are linear. This implies that the edges of these elements are straight. They are among the most popular element types as it has simple shape functions and are less computationally exhaustive. They also converge faster due to few DOFs than their quadratic counterparts. However, they may be less accurate for complex geometries or stress gradients that require finer meshes to achieve comparable precision.

For quadratic elements, they are the exact opposite of linear elements. These elements include additional NPs along their edges, resulting in quadratic interpolating shape functions. This implies that the edges of these elements can curve and is preferred for problems with high gradients or curved geometries.

They tend to have higher accuracy with fewer elements compared to linear elements, but it consumes much more computational resources and storage disk than linear elements.

In light of these knowledge, there are three constraints of the chosen element type for each dimension. These constraints are:

1. It should have acceptable small numerical errors and no numerical problems with hourglassing. This will immediately rule out linear first order reduced integration elements (CPE4RT, C3D8RT).
2. It should not have too many IPs, because Abaqus will store all of them as SDVs in the ODB file, which makes this file extremely heavy, rendering postprocessing tasks and storage highly challenging. This will rule out second order full integration elements (CPE8T, C3D20T).
3. It should preferably have quadratic order to ensure non-constant hydrostatic stress gradient. When we say “preferably”, it means that linear elements can also be used, but they would need more work on processing than quadratic ones before simulation begins.

To sum up, for 2D case studies, this thesis uses CPE8RT element, and for 3D case studies and simulations for CP1000 steel, this thesis uses C3D8T element. For each case, hydrostatic stress gradient is formulated to illustrate the difference between the strategies of linear and quadratic elements.

* 1. Calculation of hydrostatic stress gradient for CPE8RT

The formulation of the hydrostatic stress gradient for the element CPE8RT has been documented by Emilio et al. [3]. Basically, the gradient is calculated based on the derivatives of the shape functions of IPs that maps physical coordinates to local coordinates. The entire calculation process is based only on IPs. Nonetheless, the explanation is not in-depth, so this section shall describe this process step by step from basic theories. The CPE8RT element is illustrated in Figure 12 below.

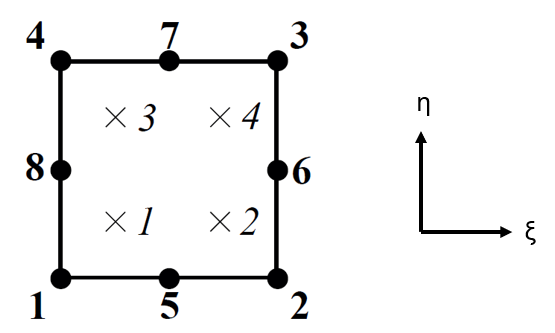


Figure 12. CPE8R/CPE8RT - eight NP, four IP quadrilateral

second-order plane strain element [3]

From the section *ABAQUS UMAT subroutine*, is computed at each IP for all elements in the model. This implies two ideas: is not a DOF, and must be obtained at all IPs. There are in total 4 steps of calculating the term for hydrogen diffusion model as follows.

**Step 1**: Find the shape functions that extrapolates from the current IP (kIP) onto any other IPs (iIP) within the elements. The shape function is bilinear with an average coefficient of ¼ for 4 IPs.

where and are the local coordinates of the current IP and and are the local coordinates of the other IP (including its own). Normally, the local coordinates of and lie at by convention. Based on Figure 12, the specific coordinates of the IPs are listed in Table 6.

Table 6. Local coordinates of IPs in CPE8RT element

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| /iIP | 1 | 2 | 3 | 4 |
| / |  | + |  |  |
|  |  |  |  |  |

Additionally, the shape function in FEA must satisfy these 4 conditions:

1. The shape function associated with a specific point must have a value of 1 at that node and 0 at all other points.
2. Partition of unity, which means the sum of all shape functions for an must be 1 at any point inside the element.
3. Shape function must be continuous with one differentiable form for linear shape functions and twice differentiable with quadratic ones.
4. Shape function must be nonnegative.

Finally, it is convenient to define the derivative of the shape functions with respect to and as follows:

**Step 2**: We need to find the rate of change of physical and coordinates of the other IP with respect to their local coordinates, and . We need to access the physical coordinates of those IPs ( and ) as inputs, which are already provided by UMAT’s parameter COORDS.

These four components are used to form the Jacobian matrix that transforms local IP coordinates and to physical IP coordinates and

**Step 3**: We obtain the inverse of Jacobian matrix , which transforms the physical IP coordinates and to local IP coordinates and

Now we use the inverse of the Jacobian matrix to compute the derivatives of the shape function ​with respect to physical coordinates (, ). This formula is computed for all

**Step 4**: Finally, the gradient of the hydrostatic stress at each IP is obtained from the sum product of the rate of change of shape function with respect to global coordinates ( and ) and the hydrostatic stress at each IP. It is noted that has already been computed inside UMAT. This formula is then computed for all .

* 1. Calculation of hydrostatic stress gradient for C3D8T

Before discussing the computation of for 3D case, it is necessary to make clear the relationship of interpolation and extrapolation between NPs and IPs. In Abaqus, values at DOFs at the current increment are passed into each element to solve the incremental values of the DOFs for the next increment. This behaviour is unseen in UMAT/UMATHT since these two subroutines operate at the IP level, but they are available as the variables *u* and *du* in the UEL subroutine (Annex F) because this subroutine is at the element level. Then, values of the DOFs at the IPs shall be interpolated from the NPs using shape functions, such as the internal thermal energy per unit mass *u* passed into UMATHT at the begin of the increment. This is the standard procedure of interpolation from NP to calculate of physical properties at IPs.

After analysis, all values of physical properties stored at IPs are illustrated in Abaqus/Viewer in the postprocessing stage. Values of these physical properties at NPs shall be extrapolated from the IPs onto the NPs. The contour plot displays the average output parameter value over a specific element, where the relative contribution difference onto a NP is less than or equal to 75%. Therefore, during simulation analysis, interpolation of NPs to IPs is required for passing physical properties to user subroutines for integration, while extrapolation of IPs to NPs is required in visualization and calculate the values at NPs for non-DOFs physical properties for extraction.

The formulation of the hydrostatic stress gradient for the element C3D8T is significantly more complex than that in the 2D case. This time, we would also use NPs together with IPs and conduct a series of numerical manipulation to achieve the correct behaviour of . Firstly, the C3D8 element is illustrated in Figure 13 below.

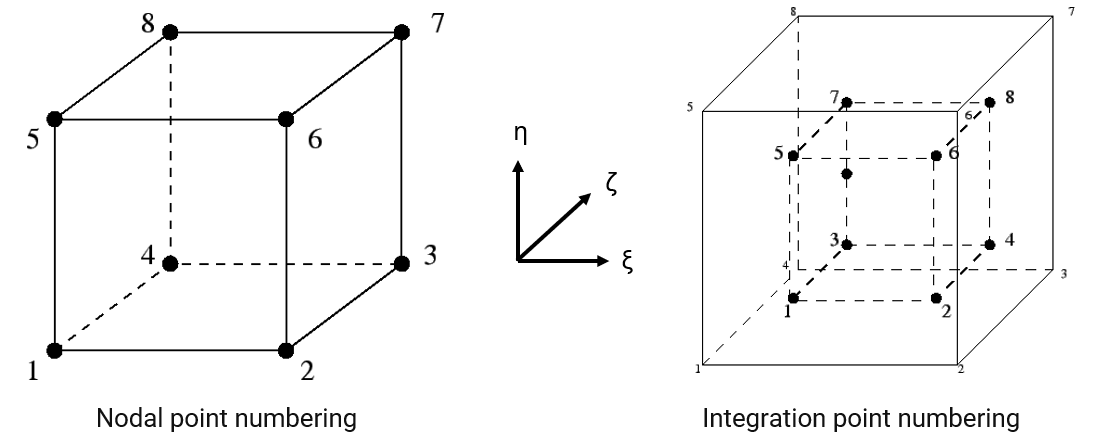


Figure 13. C3D8/C3D8T - eight NPs, eight IPs linear

first-order brick element [4]

The process of calculating consists of three separate works, and the procedure of assembling the works will be described in the next section *ABAQUS UEXTERNALDB and UFIELD subroutine*

**Work #1**: Find the shape functions that interpolates from the current NP (iNP) into the IPs (kIP) within the elements. The shape function is trilinear with an average coefficient of 1/8 for 8 NPs.

where , , and are the local coordinates of the IPs and , , and are the local coordinates of the NPs. Normally, the local coordinates of , , and lie at by convention, and the local coordinates , , and lie at . The value of originates from the Gaussian quadrature method, which is used for numerical integration in FEA. Gaussian quadrature selects IPs as the roots of Legendre polynomials within the reference interval to ensure the exact evaluation of integrals for polynomials up to a certain degree [5]. For a 2-point Gaussian quadrature rule, the integration points are derived from the roots of the second-order Legendre polynomial, given by:

The roots of this equation would be , hence the local coordinates of IPs. To verify if the local coordinates of C3D8T are actually (approx. 0.57735), they can be queried from Abaqus output database (ODB). Figure 14 has indeed confirmed the correct local coordinates of IPs.



Figure 14. Local coordinates of IPs of C3D8T element in Abaqus

Therefore, the local coordinates for the NPs and IPs for the interpolation version from NPs into IPs are recorded in Table 7 and Table 8.

Table 7. Local coordinates of NPs in C3D8T element (interpolation)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  | + |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Table 8. Local coordinates of IPs in C3D8T element (interpolation)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Additionally, we can obtain the derivatives of the with respect to three local coordinates for all IPs

However, there is another convention which states that the source points containing values for extrapolation should be scaled onto the range. Therefore, in the case of extrapolation from IPs to NPs, the IPs’ actual local coordinates would lie at , and scaling by the factor of , the actual local coordinates of NPs become . Therefore, we find the shape functions that extrapolates from the current IP (kIP) onto the NPs (iNP) within the elements. The shape function is trilinear with an average coefficient of 1/8 for 8 IPs.

From Figure 13, it is possible to define the local coordinates for extrapolation from IPs to NPs in Table 8 and Table 9 below.

Table 9. Local coordinates of NPs in C3D8T element (extrapolation)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  | + |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Table 10. Local coordinates of IPs in C3D8T element (extrapolation)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Figure 15. Local coordinates of IP inside C3D8T element in Abaqus

Therefore, we can observe that shape function can behave in both ways: interpolating at IPs from NPs, or extrapolate at NPs from IPs, depending on the order of integration of the source points. Finally, the weight of contributions of all extrapolating shape functions are all for both versions.

**Work #2:** To calculate the gradient of any physical quantities with respect to physical coordinates of IP for C3D8T element, it is required to define three matrices to keep track of the information in the mesh. For illustration purpose, a simple 3D FE model is built with 8 elements and 27 NPs. In Figure 17, the brown number is the element ID, and the purple number is the NP’s ID. There are two hidden NPs, which is ID 14 at the middle centre inside the mesh, and ID 15, which is the centre of the bottom face.

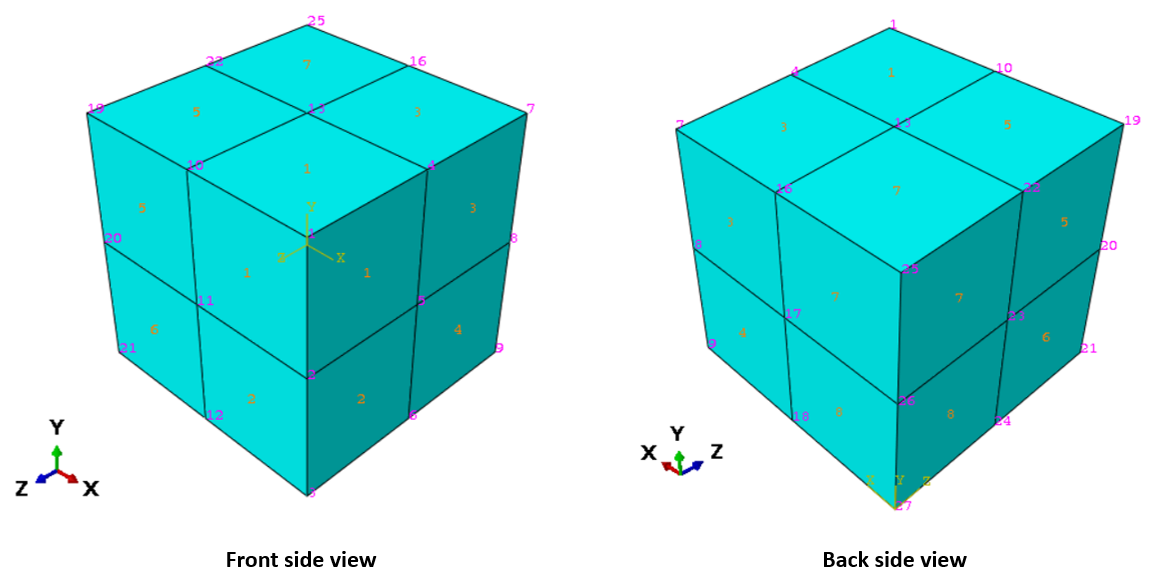


Figure 17. A simple 3D FE model comprised of C3D8T elements

The three matrices are defined as below together with their shape.

* : Element connectivity matrix

This matrix can be obtained directly from Abaqus .inp file where the connectivity of the FE mesh is defined below the element type declaration. It keeps track of the node IDs that form the current queried element ID. The order by which these nodes are inserted into the array is based on the NP counting order convention for C3D8T element. In this matrix, the element ID is the same as the index number. because Fortran array starts counting from 1. For illustration, this matrix should look like below.

Element ID 1:

Element ID 2:

Element ID 3: 13, 14, 17, 16, 4, 5, 8, 7

Element ID 4:

Element ID 5:

Element ID 6:

Element ID 7:

Element ID 8:

* : Element counting matrix for NP.

This matrix keeps track of how many elements that contains the current queried NP. Similar as before, the node ID to be queried from this matrix is the same as the index number. For illustration, this matrix should look like below for the 3D FE model.

NP of ID 1, 7, 19, 25, 3, 9, 21 and 27:

NP of ID 4, 10, 16, 22, 2, 8, 20, 26, 6, 12, 18, and 24:

NP of ID 5, 11, 13, 17, 23, 15:

NP of ID 14:

* : NP connectivity matrix

This matrix can be thought of as an inverse mapping of . It keeps track of the element IDs that forms the current queried NP. There is a point to consider from the matrix above that each NP in the mesh may have different number of elements containing them. By empirical investigation of the mesh, it is almost certain that the maximum number of elements sharing the same NP is 10 for C3D8T element, which is the value of **nmax\_elems**. Finally, the third dimension contains two values. The first value tells the element ID that contains this NP, while the second value tells the position of this queried NP in this element. Like previous matrices, the queried IP of NP is the same as the index number.

On careful observation, if the NP does not have **nmax\_elems** elements containing it, the rest entries will be padded with zeros for both element ID and the NP position in this element. The order by which these elements are inserted into the matrix is first encountered – first added fashion when scanning the connectivity matrix of the FE mesh. When this matrix is used in practice such as in the for-loop, the total number of elements containing the queried NP is obtained from to save users from checking while-loop condition until it meets an entry of padded zeros.

For illustration, this matrix should look like below for the 3D FE model. Since there are 27 NPs, four examples are given where the NP have 1, 2, 4 and 8 elements containing it.

NP of ID 3 (1 element containing it):

(padded with 0s)

NP of ID 12 (2 elements containing it):

= (padded with 0s)

NP of ID 11 (4 elements containing it):

=

NP of ID 14 (8 elements containing it):

**Work #3:** Calculate the at all NPs after each increment. To do this work, it is important to be aware that a single NP can belong to many elements as mentioned in Work #2, and is defined at IPs. By first intuition, is obtained at NP from all IPs within the element containing such NP by using the extrapolating shape functions. However, this process is computable for all elements that contains that NP. The question arises is which neighbouring element one should use to calculate at the NP. If the mesh is homogeneous and all elements have roughly the same size, then averaging out calculated at all neighbouring elements suffice.

Nonetheless, the mesh in reality has a wide variety of size and shape. Therefore, the approach used in this thesis is averaging out the values of at the NP among neighbouring elements by the weighted volume contribution of that NP to each element using the Jacobian matrix . The term represents the Jacobian determinant for a specific NP in a specific element (EL). It describes the transformation between the local coordinates (ξ, η, ζ) and the physical coordinates (x, y, z) at that node within the given element.

The Jacobian roughly satisfies the condition of volume integration:

In order to calculate , we first define the shape functions that interpolates from the current NP (kNP) into other NPs (iNP) (including themselves) within the elements The shape function is trilinear with an average coefficient of 1/8 for 8 NPs.

In this version, all local coordinates lie at , and their node order and their respective coordinates are listed in Table 7. Then, we compute gradients of shape functions with respect to local coordinates:

Then, is calculated using the dot product of the gradient of the shape function and the physical coordinates of the NP , and .

Finally, for each NP of index kNP, we calculate ​ using contributions from obtain at that NP of index kNP from all elements containing it, average out by the sum of the Jacobians.

* 1. ABAQUS UEXTERNALDB and UFIELD subroutine

For simplicity of the documentation, the pseudo-algorithm of calculation is only reported for the 3D case (C3D8T). In Abaqus/Standard, the subroutine UEXTERNALDB is called at various moments throughout the analysis process. It allows communication between other user subroutines, opens, and closes external files for other subroutines, calculates or reads history information at the start of each increment, and writes the current values of the user-calculated history information to external files [4]. The interface of this subroutine is shown in Pseudocode 3.

Pseudocode 3. The UEXTERNALDB subroutine interface

|  |  |  |
| --- | --- | --- |
| **1** | subroutine **UEXTERNALDB**(lop, lrestart, time, dtime, kstep, kinc) | |
| **2** |  | include ‘aba\_param.inc’ |
| **3** |  | dimension time(2) |

Parameters:

* kstep: Current step number. For lop=0 or lop=4, kstep is 0.
* kinc: Current increment number. For lop=0 or lop=4, kinc is 0.
* lop is an integer indicating which moment in the analysis is called. Particularly, there are special seven moments during an analysis, each is associated with a lop number.

• LOP=0: UEXTERNALDB is called at the start of the analysis.

• LOP=1: UEXTERNALDB is called at the start of the current increment.

• LOP=2: UEXTERNALDB is called at the end of the current increment.

• LOP=3: UEXTERNALDB is called at the end of the analysis.

• LOP=4: UEXTERNALDB is called at the start of a restart analysis.

• LOP=5: UEXTERNALDB is called at the start of a step.

• LOP=6: UEXTERNALDB is called at the end of a step.

We proceed to create these matrices to keep track of the information for calculation of throughout the simulation analysis.

* : Maps element IDs to their corresponding NPs, as outlined in Work #2
* : Counts the total number of elements connected to each NP, as outlined in Work #2
* : Stores the element IDs and local node indices associated with each nodal point, as outlined in Work #2
* : hydrostatic stress at IP for each element.
* : Represents the hydrostatic stress at NP for each element, extrapolated from its own IPs.
* : Stores the physical coordinates of all NPs
* : Represents the determinant of the Jacobian at NPs for all elements, as outlined in Work #3
* : The averaged hydrostatic stress at each NP using the method outlined in Work #3
* : Hydrostatic stress gradient at all IPs.
* : Shape function extrapolating from IPs to NPs, as outlined in Work #1
* : Shape function interpolating from NPs to IPs, as outlined in Work #1
* : Shape function gradient of with respect to IP’s local coordinates, as outlined in Work #1
* : Shape function gradient interpolating from current NPs to other NPs, as outlined in Work #3

In the workflow, at LOP = 0 (start of the simulation analysis), the tasks are

* Open the Abaqus .inp file that contains the mesh information and populate the three matrices , and .
* Calculating , and . These three matrices are always fixed throughout the simulation. For example, to obtain the shape function values at the kIP interpolated from all NPs, the matrix can be queried as kIP,1:nnode)

During each increment, UMAT is called for all IPs, and is populated at each call by , where noel is the element ID and npt is kIP. At the end of the increment, this matrix is fully populated. In the workflow, at LOP = 4 (the end of each increment), the tasks are:

* Populating by using

to extrapolate values of stored in IPs from IPs to all NPs of each element.

Pseudocode 4. Calculation of hydrostatic stress at NPs for each element

|  |  |
| --- | --- |
| **1** | do = 1, total\_elems |
| **2** | do kNP = 1, nnode |
| **3** | ! Initialize hydrostatic stress for the current node to zero |
| **4** |  |
| **6** | ! Compute the hydrostatic stress at the nodal point |
| **7** | do kIP = 1, ninpt |
| **8** |  |
| **9** | end do |
| **10** | end do |
| **11** | end do |

* Populating as outlined in Work #3. Now we can use the shape function gradient

Pseudocode 5. Calculation of Jacobian determinant at NPs for each element

|  |  |
| --- | --- |
| **1** | ! Initialize temporary arrays to store the physical coordinates |
| **2** | , , of dimension (nnode), J of dimension (ndim, ndim) |
| **3** | do = 1, total\_elems |
| **4** | do kNP = 1, nnode |
| **5** | ! Retrieve the node ID of the current node in this element |
| **6** |  |
| **7** | ! Extract the current physical coordinates of this NP |
| **8** |  |
| **9** |  |
| **10** |  |
| **11** | end do |
| **12** | ! Loop over each node in the current element |
| **13** | do kNP = 1, nnode |
| **14** | ! Initialize Jacobian matrix as zeros |
| **15** | J = 0.0 |
| **16** | ! Calculate the Jacobian matrix of the current node kNP |
| **17** | do iNP = 1, nnode |
| **18** | do jdim = 1,ndim |
| **19** | J(jdim,1) += |
| **20** | J(jdim,2) += |
| **21** | J(jdim,3) += ) |
| **22** | end do |
| **23** | end do |
| **23** | ! Compute determinant of Jacobian matrix and store into |
| **24** |  |
| **25** |  |
| **26** | end do |
| **27** | end do |

* Populating by using the weighted volume average of and as outlined in Work #3.

Pseudocode 6. Calculation of hydrostatic stress at all NPs

|  |  |
| --- | --- |
| **1** | = 0.0 |
| **2** | ! Loop over all nodes in the mesh |
| **3** | do = 1, total\_nodes |
| **4** | ! Find the number of elements containing the node ID |
| **5** | = () |
| **6** | ! Initialize temporary variables for summing and detJ |
| **7** | = 0.0 |
| **8** | = 0.0d0 |
| **9** | ! Loop over all elements that contain the current node |
| **10** | do kEL = 1, |
| **11** | ! Get the element ID and the node order number inside the element |
| **12** | = (, kEL, 1) |
| **13** | kNP\_order = (, kEL, 2) |
| **14** | ! Retrieve for the current element and node |
| **15** | = (, kNP\_order) |
| **16** | ! Retrieve detJ for the current node inside the current element |
| **17** | = (, kNP\_order) |
| **18** | ! Accumulate the weighted sum of and the sum of detJ |
| **19** | += |
| **20** | += |
| **21** | end do |
| **22** | ! Compute the weighted average of for the current node |
| **23** | if ( > 0.0) then |
| **24** | () = / |
| **25** | else |
| **26** | () = 0.0 |
| **27** | end if |
| **28** | end do |

However, between consecutive increments, it is certain that there has not been access to the physical coordinates of the NPs. Fortunately, Abaqus/Standard has a subroutine called UFIELD, which can manipulate the user predefined fields between each increment, as shown in Pseudocode 7.

Pseudocode 7. The UFIELD subroutine interface

|  |  |  |
| --- | --- | --- |
| **1** | subroutine **UFIELD** (field,kfield,nsecpt,kstep,kinc,time,node, & | |
| **2** |  | coords,temp,dtemp,nfield) |
| **3** |  | include ‘aba\_param.inc’ |
| **4** |  | dimension field(nsecpt,nfield), time(2), coords(3), & |
| **5** |  | temp(nsecpt), dtemp(nsecpt) |

Inside UFIELD, we can update the used in Pseudocode 5 by a simple for loop as shown in Pseudocode 8 below. The coords parameter in UFIELD now refers to physical coordinates of NPs, and node parameter refers to .

Pseudocode 8. Updating physical coordinates of all NPs

|  |  |
| --- | --- |
| **1** | = coords(1) |
| **2** | = coords(2) |
| **3** | = coords(3) |

Finally, we must calculate the hydrostatic stress gradient for all IPs. It could be computed at the start of UMAT calls or also inside USDFLD subroutine. This thesis chooses the former approach, as shown in Pseudocode 9.

Pseudocode 9. Calculation of hydrostatic stress gradient at all IPs

|  |  |
| --- | --- |
| **1** | INPUT: current and current integration point kIP |
| **2** | Initialize (ndim, ndim), , |
| **3** | do kNP = 1, nnode |
| **4** | = (, kNP) |
| **5** | (kNP) = () |
| **6** | end do |
| **7** | J = 0.0 |
| **8** | do kNP = 1, nnode |
| **9** | do idim = 1, ndim |
| **10** | do jdim = 1, ndim |
| **11** | = (, kNP) |
| **12** | J(jdim, idim) += (, idim) |
| **13** | end do |
| **14** | end do |
| **15** | end do |
| **16** | = inv() |
| **17** | = matrix multiplication(, ) |
| **19** | (, kIP, 1:ndim) = matrix multiplication (, ) |