SANDIA REPORT

SAND2009-0625
Unlimited Release
Printed January 2009

J-Integral Modeling and Validation for GTS Reservoirs

Arthur A. Brown, Alex J. Lindblad, Yuki Ohashi
Multi-Physics Modeling & Simulation Department

Bonnie R. Antoun, Kevin Connelly, Soonsong Hong, Edwin M. Huestis, Jonathan A. Zimmerman
Mechanics of Materials Department

Kevin A. Nibur, Brian P. Somerday
Hydrogen & Metallurgy Science Department

Stephen B. Margolis, Monica L. Martinez-Canales
Advanced Software R&D Department

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2009-0625
Unlimited Release
Printed January 2009

J-Integral Modeling and Validation for GTS Reservoirs

Arthur A. Brown, Alex J. Lindblad, Yuki Ohashi
Multi-Physics Modeling & Simulation Department

Bonnie R. Antoun, Kevin Connelly, Soonsong Hong,
Edwin M. Huestis, Jonathan A. Zimmerman
Mechanics of Materials Department

Kevin A. Nibur, Brian P. Somerday
Hydrogen & Metallurgy Science Department

Stephen B. Margolis, Monica L. Martinez-Canales
Advanced Software R&D Department

Sandia National Laboratories
Livermore, California 94551-0969

Abstract

Non-destructive detection methods can reliably certify that gas transfer system (GTS) reservoirs do not have cracks larger than 5%-10% of the wall thickness. To determine the acceptability of a reservoir design, analysis must show that short cracks will not adversely affect the reservoir behavior. This is commonly done via calculation of the J-Integral, which represents the energetic driving force acting to propagate an existing crack in a continuous medium. J is then compared against a material's fracture toughness (J_c) to determine whether crack propagation will occur. While the quantification of the J-Integral is well established for long cracks, its validity for short cracks is uncertain.

This report presents the results from a Sandia National Laboratories' project to evaluate a methodology for performing J-Integral evaluations in conjunction with its finite element analysis capabilities. Simulations were performed to verify the operation of a post-processing code (J3D) and to assess the accuracy of this code and our analysis tools against companion fracture experiments for 2- and 3-dimensional geometry specimens. Evaluation is done for specimens composed of 21-6-9 stainless steel, some of which were exposed to a hydrogen environment, for both long and short cracks.

Acknowledgements

The authors would like to acknowledge helpful discussions and input from Dorian K. Balch, Douglas J. Bammann, James W. Foulk III, and Christopher W. San Marchi, and assistance from Michael Burger with meshing our simulation geometries and from Nathan Spencer with curve fitting. The authors would also like to acknowledge support of this project from the Readiness in Technical Base & Facilities (RTBF) program and the Gas Transfer Systems Department.

Contents

List of Figures

List of Tables

This page intentionally left blank.

1 Introduction

1.1 Technical Problem and Project Goals

Current non-destructive detection methods can reliably certify that gas transfer system (GTS) reservoirs do not have cracks larger than 5%-10% of the wall thickness. Hence, in order to determine the acceptability of a reservoir design, analysis must show that short cracks will not adversely affect the reservoir behavior. The current GTS Design Standard, DG10215, is undergoing revision to better represent and define the margins against crack extension. This revision is part of an effort to demonstrate damage tolerance in reservoirs and to meet the requirements of the Price-Anderson Amendment Act. Finite element analysis is the conventional computational method by which the stress fields within components are predicted given mechanical loading and/or displacements at the component's external boundaries. It has been long established that these stress fields, along with strain energy density fields and displacement field gradients, can be used to calculate the J-Integral, a path independent integral that evaluates the energetic driving force that acts to propagate an existing crack in a continuous medium. The quantity "J" is then compared against a critical value (J_c) representing the material's resistance to fracture to determine whether crack propagation will occur and the manner of that propagation (i.e. stable or unstable crack growth). J_c is often referred to as a material's fracture toughness. While the quantification of the J-Integral is well documented in both textbooks and experimental standards for long cracks, what has not been as rigorously determined are the length limits for short cracks at which the J-Integral expression loses its validity.

This project's primary goal is to establish and evaluate a methodology for performing J-Integral evaluations in conjunction with Sandia's finite element analysis capabilities. Simulations were performed to verify the operation of a post-processing code (J3D) for calculating the J-Integral and to assess the accuracy of this code and our analysis tools against companion fracture experiments for 2- and 3-dimensional geometry specimens. These specimens were composed of 21-6-9 stainless steel, and half of the specimens were exposed to a hydrogen environment resulting in an atomic concentration of hydrogen of 1%. This evaluation is done for both long and short cracks, the latter of which is a concern in quantifying safety margins for designs of gas transfer system (GTS) reservoirs. Through this report, we will provide a rigorous assessment of our ability to use the J-Integral to predict fracture in components such as GTS reservoirs and provide a recommendation regarding their use in the design of future reservoirs as documented in the GTS design standard DG10215.

Another goal of this project is to quantify the margins and uncertainties (QMU) for the modeling and simulation activities discussed above and throughout this report. The information obtained through these efforts is needed to revise the gas transfer systems (GTS) reservoir design standard to better represent and define the margins against crack extension. Project tasks have focused on addressing the issues of

verification and validation of the elastic-plastic constitutive model used and the algorithm for calculating the J-Integral using standard output from the analysis code, on quantifying uncertainties in the fitted parameters of the constitutive model and gauging their effect on the subsequent FEA and J-Integral analysis, and on assessing the requirements that affect the calculation of the J-Integral and the requirements that affect experimental measurement of the fracture toughness – the criteria against which the J-Integral is evaluated.

1.2 Background Information on J-Integral

The J-Integral, developed by Rice [1] and based on the work by Eshelby [2], is a path independent integral that evaluates the energetic driving force that acts to propagate an existing crack in a continuous medium. The J-Integral is a path integral defined with respect to a plane that intersects the front of a crack tip, as shown below in Figure 1.

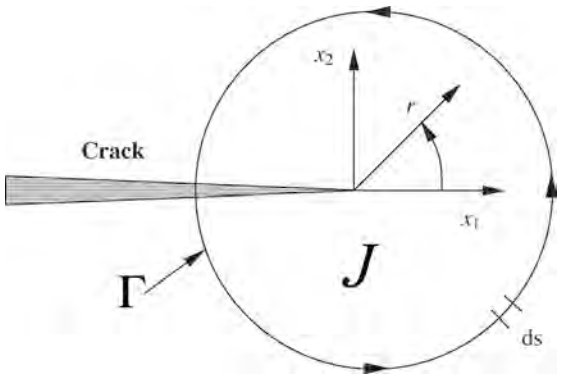


Figure 1 The J-Integral path (from Rao and Rahman, 2004[3])

The conventional expression for J is

$$J = \int_{\Gamma} \left(W n_1 - \sigma_{ij} n_j \frac{\partial u_i}{\partial x_1} \right) ds,$$

where *W* is the strain energy density, **n** is the vector normal to the path defined by Γ and pointing away from the crack tip (i.e. outward normal), σ_{ij} is the Cauchy stress field and **u** is the displacement field. The power of the J-Integral lies in its path-independence; evaluation of the integral along a path far from the crack tip yields the correct driving force even though the tip may be surrounded by a zone in which complex physical mechanisms such as plastic deformation (for ductile materials), fiber pullout (for composite materials), or granular rotation and separation (for brittle polycrystalline materials) may be occurring. This driving force is related to the geometry of the body containing the crack, as well as the external loads applied to

the body. Propagation commences when the value of the J-Integral reaches a critical level, the fracture toughness J_c.

The form of J as a path integral in a 2-dimensional body immediately presents a challenge for how to define J in a 3-dimensional body containing a variable-shaped crack front. Papers by Blackburn[4], Kishimoto et al.[5], Amestoy et al.[6] and Batte et al.[7] define path independent integrals that include volume terms to take into account fracture process zones, post-yield behavior and 3-dimensionality. For example, for a 3-dimesional body or a material that exhibits inhomogeneous or non-elastic behavior, Blackburn defines J as

$$J = \oint_{\Gamma} \left(W n_1 - \mathbf{T} \cdot \frac{\partial \mathbf{u}}{\partial x_1} \right) d\Gamma - \operatorname*{Lim}_{\rho \to 0} \left[\int_S \frac{\partial}{\partial x_3} \left((\boldsymbol{\sigma} \cdot e_3) \cdot \frac{\partial \mathbf{u}}{\partial x_1} \right) dS \right].$$

J in the above expression consists of the normal path integral and the limit of an integral over the area enclosed by the path as the area size (ρ) approaches zero. J is evaluated at a point along the crack front; thus, a discretization of the crack front and evaluation of J at these discrete locations would be necessary to model driving force along the entire crack front. Kishimoto et al.[5] proposed a somewhat different expression for a path independent integral, and for a 3-dimensional body evaluates a fracture process region or "plate" characterized by inner and outer contours, shown in Figure 2.

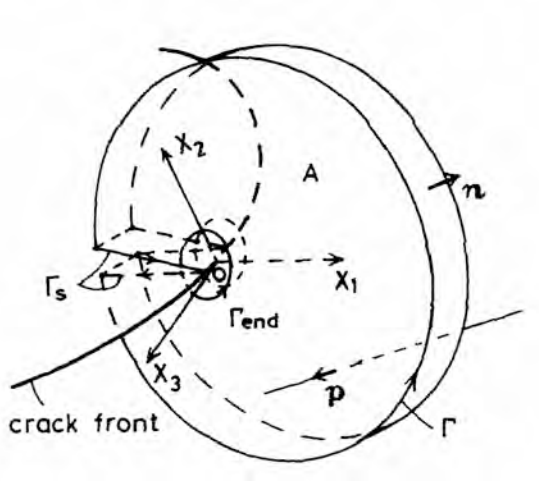


Figure 2 A 3-dimensional crack front, fracture process plate for J-Integral calculation (from Kishimoto et al., 1980[5])

An alternative approach developed by Li et al.[8] and Shih et al.[9] was to define surface and volume integral expressions for J using weight functions that vary from zero to unity between inner and outer contour paths or surfaces, respectively. An example volume and defining surfaces are shown in Figure 3 for an axisymmetric

body. Similar methods and expressions were developed and used in papers by Chiarelli and Frediani[10], Meith and Hill[11], Ericksson[12], and Nguyen et al.[13]

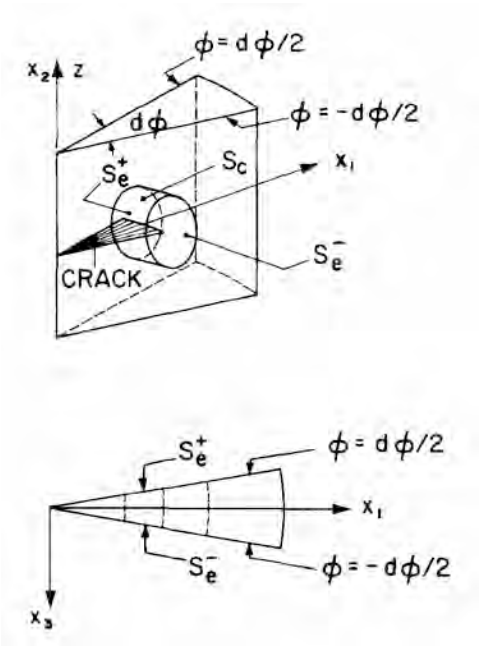


Figure 3 Three Dimensional Surface for J-Integral calculation in axisymmetric body (from Shih et al, 1986)

At Sandia, Wellman developed post-processing codes J2D and J3D to calculate the J-Integral expression by Amestoy et al.[6] using results from finite element analyses[14]. The expression used,

$$J_{3D} = \oint_{\Gamma} \left(W n_1 - \mathbf{T} \cdot \frac{\partial \mathbf{u}}{\partial x_1} \right) d\Gamma - \int_S \frac{\partial}{\partial x_3} \left((\boldsymbol{\sigma} \cdot e_3) \cdot \frac{\partial \mathbf{u}}{\partial x_1} \right) dS,$$

is similar to the one discussed earlier by Blackburn. Wellman's method for evaluating J in 3-dimensional bodies is as follows:

1. First, a plane is defined that intersects the crack front.
2. Next, at least four approximately concentric paths that enclose the crack tip are defined on that plane.
3. The path integral portion of the above expression is evaluated and a different value of "J" is recorded for each path. Also recorded is the 2-dimensional area that each path defines, "A".
4. It can be shown that the surface integral in the Amestoy expression scales is proportional to A^2. Hence, a least squares regression analysis is performed to fit the unknown coefficients in the relation: $J = C_0 + C_1 A^2$. Comparing this relation with the Amestoy formula, one realizes that the coefficient C_0, i.e. the

zero area limit of the relation, yields the true J (J_{3D}). This is consistent with the expression by Blackburn.

Another concern regarding the J-Integral is its validity for short cracks. Most textbooks[15] and mechanical testing standards require a minimum crack length, relative to specimen dimensions, to define and quantify J. The rationale behind this requirement has to do with the effect of geometric constraint on the stress state at the crack tip. Mercer and Nicholas[16] noted that standard expressions for crack tip stress states, used in the conventional and 3-dimensional J-Integral expressions discussed above, do not predict fatigue fracture behavior for bodies containing short cracks. A more comprehensive theoretical investigation was conducted by O'Dowd and Shih[17, 18], who argued that J alone is insufficient for short cracks where crack tip stress triaxiality constraint is lost. They defined a parameter Q to describe the stress distribution and maximum stress, and designated that J sets the size scale for large stresses and strains through the expression

$$\frac{\sigma_{ij}}{\sigma_Y} \sim \left(\frac{J/\sigma_Y}{r} \right)^{\frac{1}{n+1}} f(\theta, n) + Q \left(\frac{r}{J/\sigma_Y} \right)^{q} g(\theta, n).$$

Hence, J (the driving force for crack propagation) is not sufficient to predict the fracture of short cracks. Rather, a combination of J and Q must be used. Put another way, the fracture toughness J_c can now be considered a function of the crack tip stress distribution, i.e. Q. This conclusion was shown in later analysis by Ainsworth and O'Dowd[19] and is shown in Figure 4.

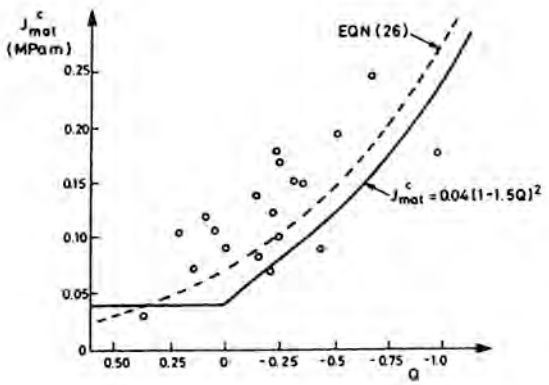


Figure 4 Fracture toughness as a function of crack tip constraint factor Q for a body with a short crack (from Ainsworth and O'Dowd[19]). Circles represent experimental measurements while the solid and dashed lines represent fits to analytic expressions.

From this perspective, Matvieko and Morozov developed expressions for a 2-parameter fracture criterion for short cracks[20]. A goal of this analysis project was to confirm or refute that a single value of fracture toughness is insufficient to predict crack propagation.

1.3 Approach for Technical Work

As stated earlier, this project is focused on establishing a methodology for performing computational J-Integral evaluations. Our exploration of scientific and engineering literature has identified several approaches for calculating the J-Integral for 3-dimensional geometries. We've also obtained several references that examine the usefulness of the J-Integral when crack-tip constraint is lost or has not been established. Several articles suggest the use of multiple criteria to predict the onset of crack propagation, however the necessity of this use has not been firmly established. As such, our project breaks-down into the following tasks:

1. Develop accurate material models for the elastic-plastic deformation behavior of 21-6-9 stainless steel in both its annealed state and upon exposure to a hydrogen atmosphere resulting in the material containing 1 atomic % hydrogen. Parameters for these models are fit from data collected during uniaxial tension experiments performed by a separately funded C6 project[21]. Verification and optimization of these parameters is performed using notched-tension experiments performed by the same research group.
2. Verify the performance of our finite element analysis (FEA) simulation capabilities and the J3D code using standard fracture test geometries and comparison with semi-analytic solutions found in the ASTM Standard for Measurement of Fracture Toughness[22].
3. Validate the performance of our FEA capabilities and the J3D code through comparison of simulation and J calculations with fracture experiments performed on a 2-dimensional, Compact Tension (CT) geometry. Experiments were performed by the separately funded C6 project mentioned above, and details about these experiments can be found in [21].
4. Validate the performance of our FEA capabilities and the J3D code through comparison of simulation and J calculations with fracture experiments performed on a 3-dimensional, axisymmetric geometry. Experiments were performed by the separately funded C6 project mentioned above, and details about these experiments can be found in [21]. Unlike the experiments for the 2-dimensional geometry, techniques for their execution were developed here at Sandia, and a method to estimate the J-Integral was taken from the scientific literature instead of the aforementioned ASTM standard[22]. Experiments and simulations are performed for various crack lengths to examine both the 3-dimensional and short crack issues discussed previously.
5. Perform a similar analysis and comparison with experiment for a 3-dimensional asymmetric geometry that contains a continuum of crack lengths at different points along the crack front. While comparison of the load-displacement response is conceptually straight forward, the calculation of the J-Integral is less so.

2 Fitting of Material Model

As already mentioned, our project chose its focus material to be 21-6-9 stainless steel. This choice was made to maximize the probability of observing classic, brittle fracture behavior in a stainless steel alloy. This probability was further increased by including material exposed to a hydrogen atmosphere and "charged" to contain 1 atomic % hydrogen. 21-6-9 exhibits a combined elastic-plastic deformation response. While several material models are adequate to represent this response, the model originally developed by Bammann, Chiesa and Johnson[23, 24] was selected for use in a majority of our modeling and simulation activities (an exception to this selection will be noted in the next chapter). The "BCJ" model, as it was originally named, has undergone a series of extensions and modifications since its original publication and is currently referred to as the Evolving Microstructural Model of Inelasticity, or EMMI[25]. EMMI was selected for use in this project due to our project team's familiarity with the model, and because it has been implemented within the FEA code ADAGIO used for the fracture simulations discussed in subsequent chapters.

The focus of our project required our fitted version of EMMI to accurately model only the elastic-plastic deformation response of 21-6-9 at a single, slow loading rate, at room temperature and with no unloading (for details, see [21]). As such, a simplified version of the EMMI model is used:

$$\dot{\varepsilon}^p = f\left(\sinh\left(\left\langle \frac{\sigma_{eq}}{\kappa + Y} - 1 \right\rangle\right)\right)^n$$

$$\dot{\kappa} = \left(H - R_d \kappa\right)\left|\dot{\varepsilon}^p\right|$$

The first relation governs the rate at which plastic strain (ε^p) develops, while the second relation governs the rate at which kinematic hardness (κ) evolves. The parameters f, Y, n, H and R_d (defined in reference [25]) require fitting, as do the elastic properties of Young's modulus (E) and Poisson's ratio (ν). Fitting of these seven parameters is done using the computer code BFIT[26] in conjunction with uniaxial tension test data collected by the experimental project.

2.1 Uncharged 21-6-9 stainless steel

Parameters were first fit using the hardening portion of the tension test data for the annealed, uncharged (no hydrogen added) material. The uniaxial stress-strain curve for both the experimental data and fitted material model are shown in Figure 5. The fitted parameters are listed in Table 1.

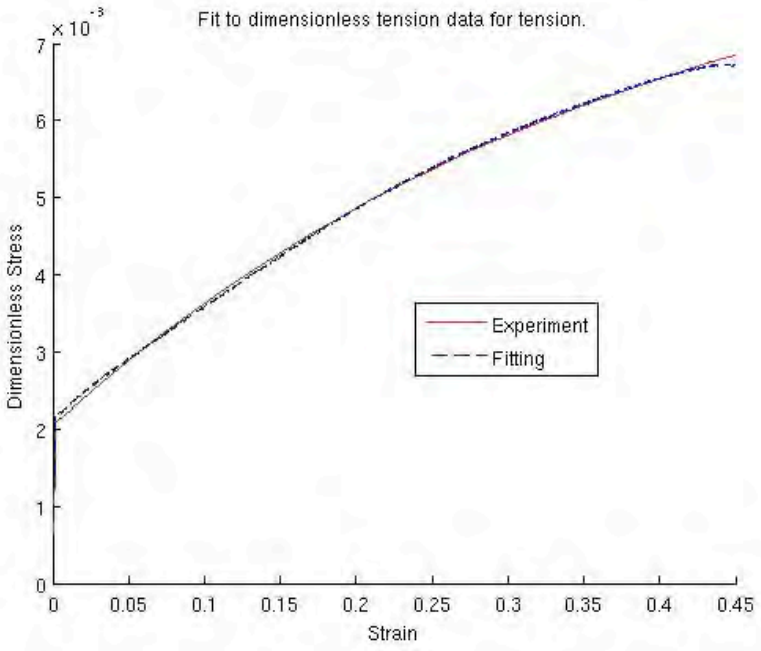


Figure 5 Fit of EMMI model parameters to tensile test data for uncharged 21-6-9

Table 1 EMMI parameter values for uncharged 21-6-9

Parameter	Units	Value
E	psi	30,500,000
ν	-	0.3
f	s^{-1}	0.0547043749985
n	-	2,000.10895362
Y	psi	27,016.0102624
H	psi	220,553.505061
R_d	-	2.57892421411

Figure 5 clearly shows that the EMMI model with fitted parameters does an excellent job at representing the elastic-plastic deformation of this material. The “Fitting” curve shown in Figure 5 is a numerical estimation assuming a uniaxial stress-strain state. The performance of the fitted model was verified at a basic level by constructing a multi-element mesh of the tension test specimen and simulating the test loading using FEA as implemented in ADAGIO. Results for this simulation are shown in Figure 6 and again show excellent agreement with the original experimental curve. This figure also shows that since damage-related aspects of EMMI were not included in the parameterization and fitting, our FEA model does not display the necking behavior observed in the experimental curve.

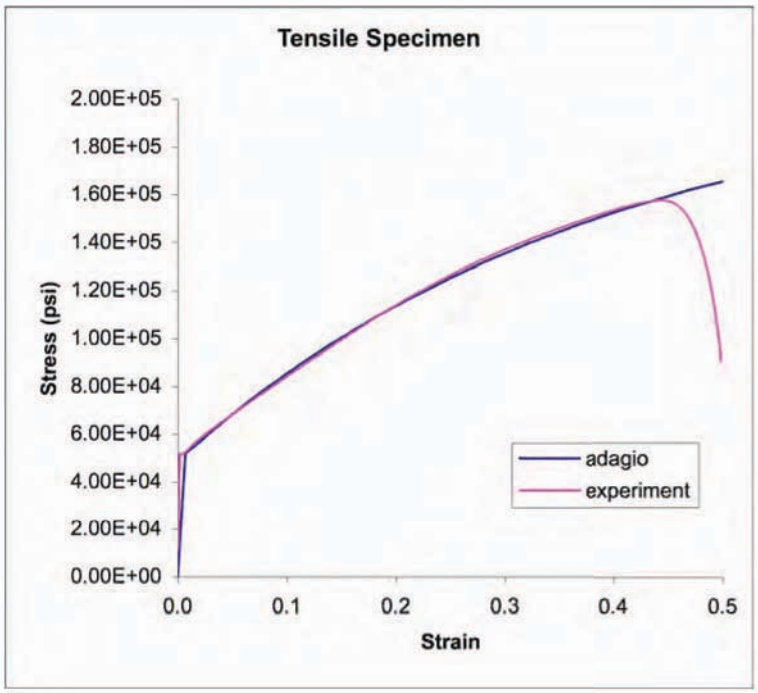


Figure 6 Comparison of FE analysis using fitted material model with experimental data for uncharged 21-6-9

2.2 Hydrogen charged 21-6-9 stainless steel

Parameters were next fit using tension test data for the annealed, hydrogen charged material. The fitted parameters are listed in Table 2, and the uniaxial stress-strain curve for both the experimental data and fitted material model (as demonstrated through an FEA simulation of the tension test) are shown in Figure 7. As with the uncharged material, we observe excellent agreement of the fitted model with experiment. This agreement does not validate our model, as the parameters were determined using the experimental data. However, the curves do show that our fitting process is consistent with expectations.

Table 2 EMMI parameter values for Hydrogen charged 21-6-9

Parameter	Units	Value
E	psi	30,500,000
ν	-	0.3
f	s^{-1}	0.088921065
n	-	6,664.715946
Y	psi	36,303.98519
H	psi	264,799.7199
R_d	-	1.679662456

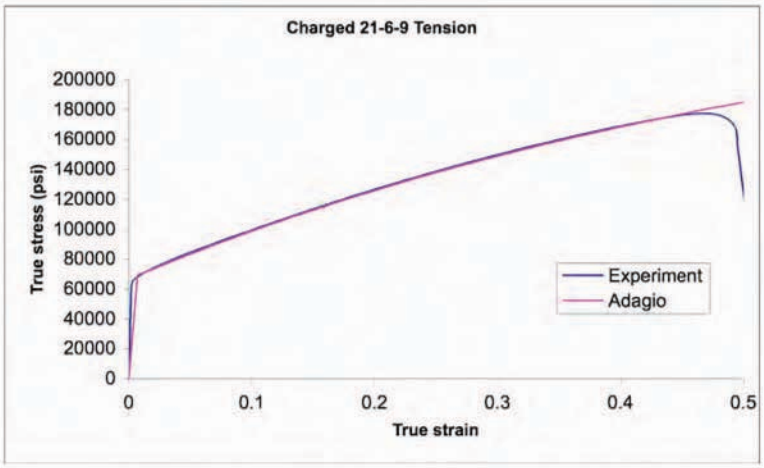


Figure 7 Comparison of FE analysis using fitted material model with experimental data for Hydrogen charged 21-6-9

A comparison between the stress-strain responses of the uncharged and hydrogen charged materials was made and is shown in Figure 8. Although this Figure was constructed using the FEA-produced curves, a comparison of the experimental curves would be identical prior to load reduction.

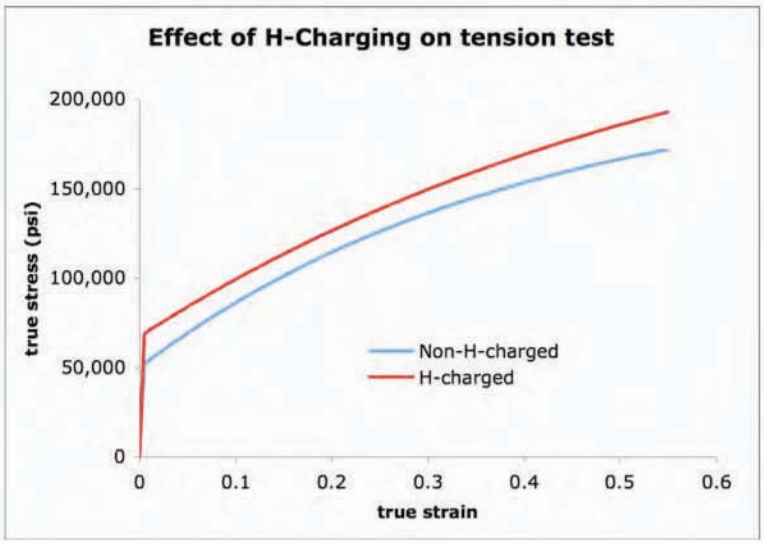


Figure 8 Comparison of true stress-strain curves for FE analyses of tension test for uncharged (blue) and hydrogen charged (red) 21-6-9

We observe that the two materials have identical elastic responses, and the plastic response, while qualitatively similar, differs somewhat quantitatively. Specifically, the hydrogen charged material exhibits a higher yield stress and a larger degree of kinematic hardening. This is also observed through calculation of the steady-state kinematic hardness:

$$\kappa_{ss} \equiv \frac{H}{R_d} = \begin{cases} 85{,}522\,\text{psi} & \text{for uncharged} \\ 157{,}651\text{psi} & \text{for charged} \end{cases}.$$

Thus, not only does the hydrogen charged material yield at a higher stress than the uncharged material, this offset in stress changes as deformation occurs, achieving a maximum value in the steady-state.

2.3 Verification and validation of material model

The ADAGIO calculations discussed in the previous section do verify the consistency of our fitted material models but do not validate them. Validation of the material models was done by using the fitted parameters with ADAGIO to analyze notched tension test specimens and comparing the results against the experimental data measured by the C6 project group. Details regarding these experiments can be found in [21].

We first consider the uncharged 21-6-9 and four notched specimens of radii 0.039", 0.078", 0.156" and 0.390". Our analysis model consists of 1/8th of the specimen, using appropriate symmetry boundary conditions at the side and bottom surfaces. Figure 9 and Figure 10 show the meshed geometry analyzed for each radius, along with the load-displacement responses for both the simulation and the experiment.

Figure 9 and Figure 10 show that agreement between the experiment and our fitted model is very good for displacements up to (and in some instances beyond) the peak load reached. For the notched tension specimen of largest radii, we notice that the analysis under-predicts experiment past a certain amount of displacement just prior to the amount coinciding with the peak load. This discrepancy caused us to pause and consider two questions: At what displacement should our model no longer be valid? Does optimization of model parameters improve the agreement between analysis and experiment?

To answer the first question, we examined the analysis results for the notched tension test of the largest radius geometry, 0.390", and focused on the evolution of the Von Mises (or equivalent) stress and plastic strain in the element undergoing the largest amount of deformation. Our analysis showed that the point at which the load-displacement discrepancy becomes noticeable (a displacement of approximately 0.08") coincides with the plastic strain going beyond the range over which the EMMI parameters were fitted, approximately 42%. Our analysis also shows that stress within the element never decreases, but rather asymptotically approaches the limit of $Y + \kappa_{ss}$, as predicted by the underlying theory to EMMI. Hence, we attribute the drop off in load seen in Figure 10 to a localization of deformation in the meshed model corresponding to specimen thinning. Our material model indeed remains valid over the range of plastic strain it was fitted to.

(a)

(b)

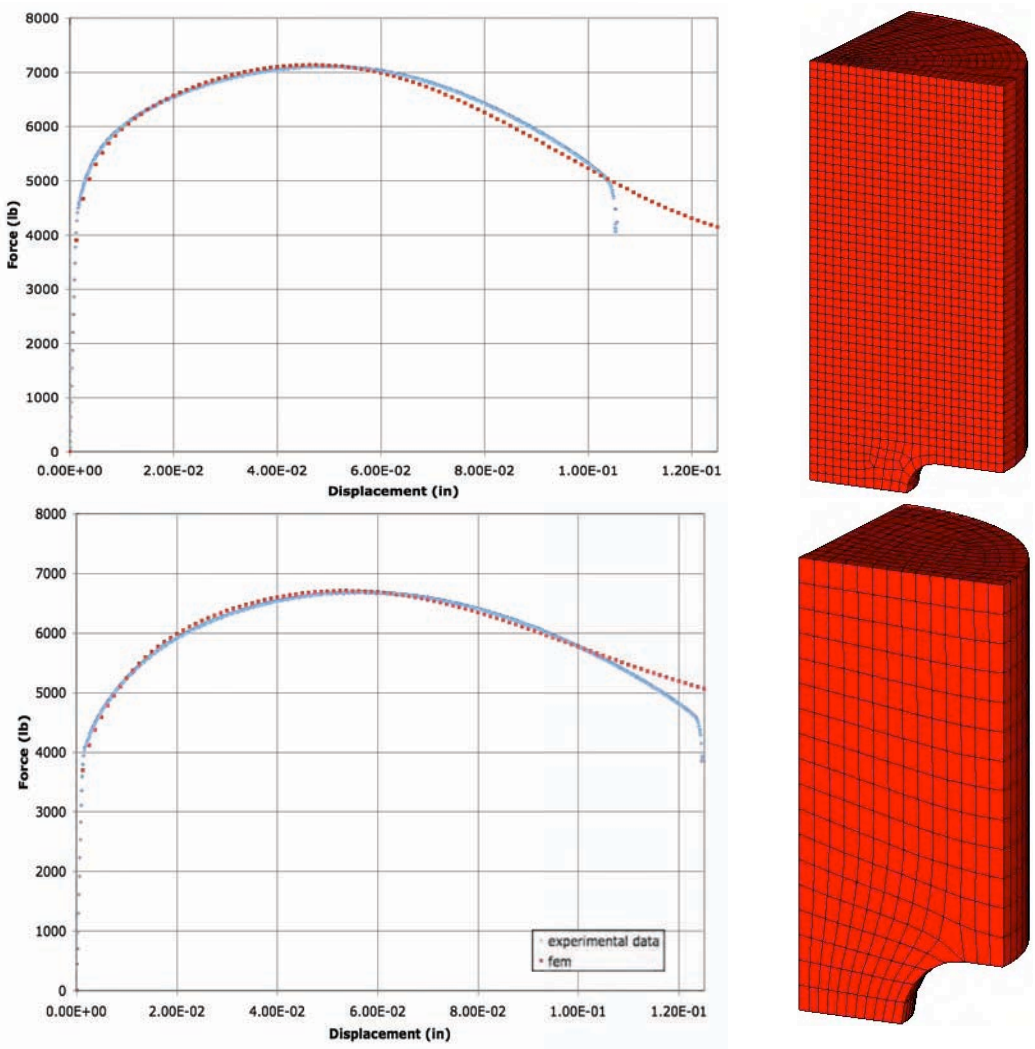


Figure 9 Comparison of FEA using fitted material model for uncharged 21-6-9 with notched tension test data. (a) Notch radius = 0.039", (b) Notched radius = 0.078"

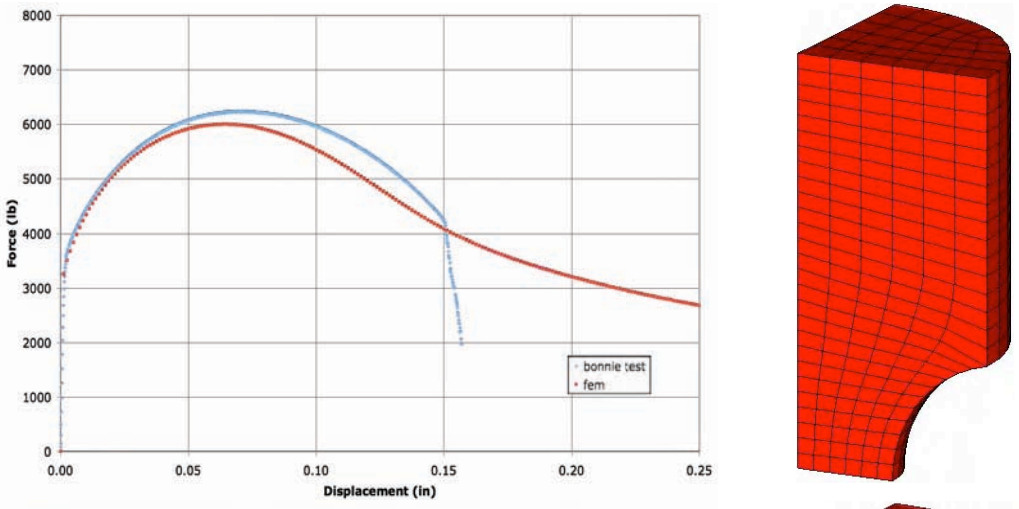
Mesh refinement and analysis was also done for each of the two larger notch radii in order to assess how the fidelity of the mesh affects the load-displacement behavior. This refinement was accomplished by dividing each element edge in half, thereby increasing the number of elements in each mesh by a factor of eight. No difference was seen in the pre-peak portion of the load-displacement curve. While mesh refinement did change the post-peak behavior, it is again noted that this occurs in a region outside that used to fit our material model and is due to localization, which is known to be mesh dependent.

The same notched test experiments and analyses were performed for the hydrogen charged material. Figure 11 shows the resulting load-displacement curves for all four notch radii. It is observed that the fitted model over-predicts the load-displacement curve for smaller notch radii but under-predicts it for larger radii. Figure 12 shows the elemental stress-strain response observed in all four simulations, and it is indeed the

case that the expected response (that of the tension test the material model was fit to) occurs in all specimens analyzed.

These observations answer our first question; our models are valid over displacement ranges corresponding to strain ranges over which the parameters were fitted. To answer the second question, in the next section we consider optimizing the EMMI parameters of the uncharged material and a statistical sampling study to characterize the sensitivity of the parameters of the hydrogen charged material.

(a)



(b)

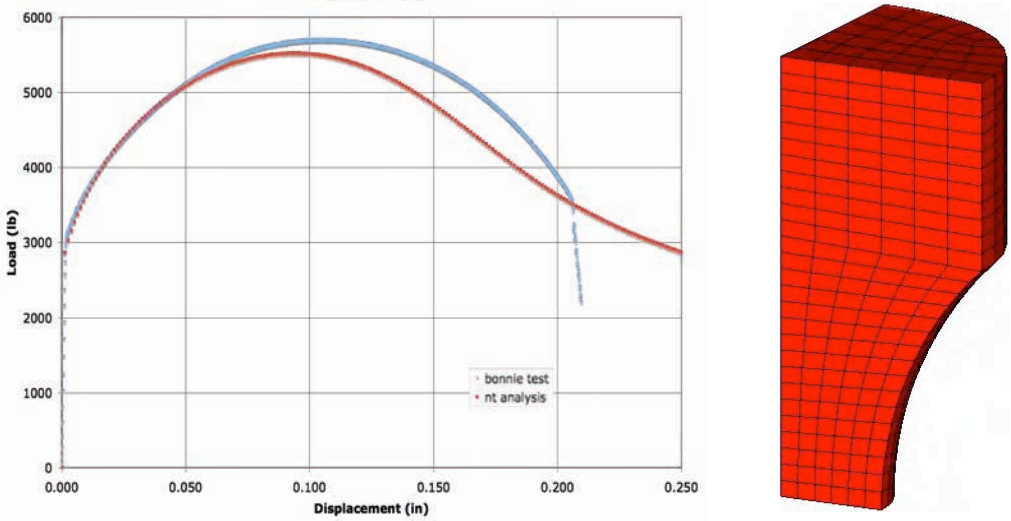


Figure 10 Comparison of FEA using fitted material model for uncharged 21-6-9 with notched tension test data. (a) Notch radius = 0.156", (b) Notched radius = 0.390"

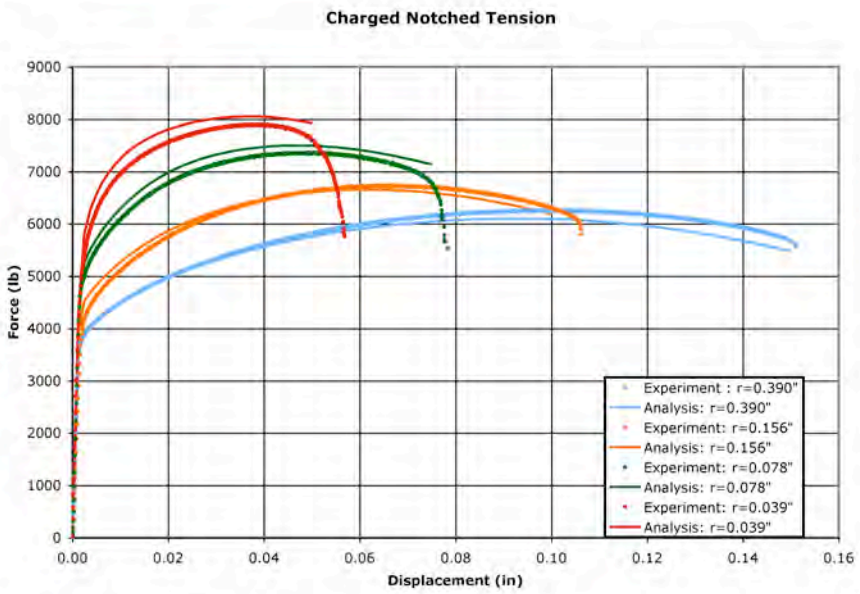


Figure 11 Comparison of FEA using fitted material model for Hydrogen charged 21-6-9 with notched tension test data.

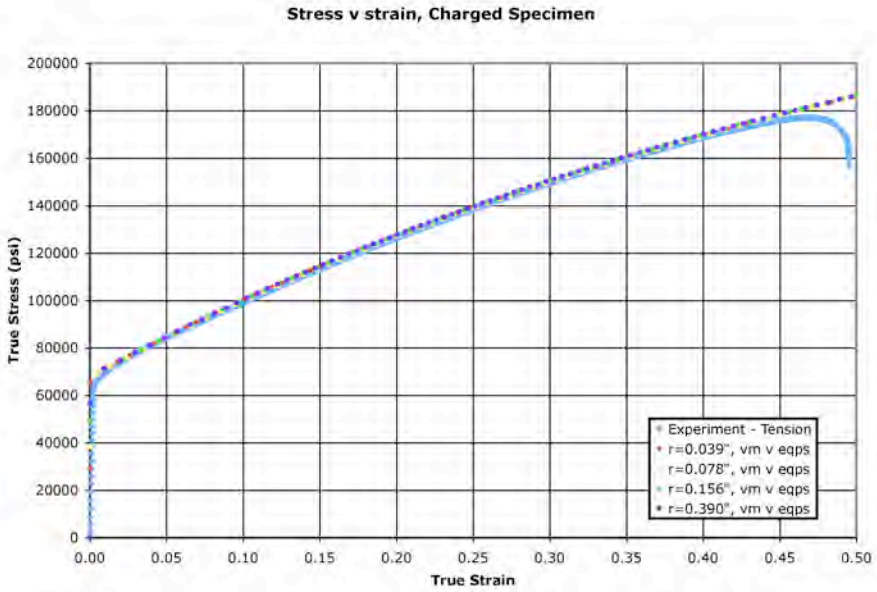


Figure 12 Comparison of elemental stress-strain response for FEA of notched tension tests with tensile test data.

2.4 Optimization and statistical sampling of material models

2.4.1 Approach

As already discussed, the analysis work thus far uses 5-parameter versions of the EMMI constitutive model for materials exhibiting elastic-plastic mechanical behavior. Simulations were performed using the ADAGIO finite element code, a component of the SIERRA framework. Material model parameters were originally fit to tension test data for 21-6-9 stainless steel. Two sets of parameters were obtained: one for annealed material and another for annealed material that underwent exposure to a hydrogen atmosphere (hydrogen charging), resulting in the steel containing 1 atomic % hydrogen. In addition to the tension tests, notched tension experiments were also performed for both uncharged and charged specimens. Four notch radii were used: 0.039", 0.078", 0.156" and 0.390". This data was used to perform statistical sampling study for uncertainty quantification of the charged material model and optimization of model parameters for the uncharged material.

The quantification of uncertainty in the predicted load (as a function of displacement) due to uncertainty in material parameters is determined from a statistical sampling procedure. Specifically, in order to reduce the n-dimensional sampling space associated with n uncertain parameters to a tractable number of samples, we employ Latin hypercube sampling (LHS)[27]. In this procedure, the range of each of the n parameters (here, $n = 5$) is divided into m contiguous intervals of equal probability (e.g., equal intervals if the uncertainty is deemed uniform across a given range). A set of m sample points, each corresponding to specific values of each of the n parameters, is then chosen according to a randomized process that guarantees good coverage of the sample space. In particular, values are chosen in a specific fashion such that no two sample points have values of any parameter from the same interval (i.e., no two points lie in the same "row" or "column" of the hypercube). We note that because a full quadratic least-squares fit of the sample data would require a minimum of n^2+n+1 sample points, this serves as a reasonable lower limit for the number m of sample points. However, a somewhat larger number (e.g., a modest multiple of this value) is desirable if one wishes to produce relatively smooth histograms of the output data. In our sample calculations described below, we specified $m = 300$ sample points in the LHS sampling algorithm.

In the actual calculations, we used the DAKOTA software package[28] to generate the LHS points, run the simulations (one for each point in the sample space), and tabulate the output. This entailed the composition of a DAKOTA input file that specified the sampling method (LHS), set bounds and distribution properties of the uncertain parameters (uniform within a range of ±20% of nominally optimal values), and called a user-composed simulation script to launch the (parallel) simulations (four per sample point, corresponding to the four geometries) and post-process the results. The latter required the composition of a response function based on the simulation data. In this particular case, a response function for each of the charged notched tension geometries was defined as a root-mean-square error of the calculated load

relative to experimental values, evaluated at ten regularly spaced displacements between zero and the maximum in the load-displacement curve. This response function can be expressed as

$$F = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(L_n - L_n^{\exp}\right)^2}$$

where the L's are the loads obtained from the simulations and the $L^{\exp}$'s are the loads measured from experiments, both obtained at each of the N displacements. A single composite response function was then computed as the average of the four response functions for each of the four geometries, which thus constituted the response output arising from a single sample point. The final primary output from DAKOTA was a table of m rows, the n+1 columns of which listed the n parameter values corresponding to each of the m sample points plus the corresponding composite response function. This table was then imported into a statistical software package (JMP) to facilitate the compilation of the preliminary uncertainty-quantification results described below.

In addition to sampling, optimization of the material model for the non-hydrogen-charged material was also performed using the SGOPT pattern search method available within the DAKOTA software package. The same response function as defined above was used to perform this optimization, where N=10 displacements of values D = .01, .02, … .10" were used. We minimized this objective F by varying the 5 model parameters: c1, c3, c5, c13, and c15. These parameters correspond to combinations of the material model parameters f, Y, n, H and R_d. Iterative use of the ADAGIO analysis code within DAKOTA achieved this minimization. This optimization process was first applied separately to the 0.039" and 0.390" data, to determine optimal parameters for these particular notched tension tests, and also to the combined set of data from the tests for four notch radii.

For clarity, we reiterate that the response function F (for either optimization or sampling) is constructed only for data within the pre-peak region of the load-displacement curve. This limits the impact that material damage has on these analyses, an aspect not considered during the fitting of our material model.

2.4.2 Optimization of EMMI parameters for uncharged material

In conducting sample studies such as that described above, it is necessary to specify, at the very least, nominal values of the uncertain parameters to center the parameter distribution (such as the ±20% uniform uncertainty range in the calculation above). In order to aid in this specification, we also used DAKOTA to carry out preliminary optimization studies in which the previously defined response function is regarded as an objective function and parameter values are sought so as to minimize this objective. In this application, we include in the DAKOTA input file the specification of the (global) optimization method, the initial guesses and ranges for the parameters to be optimized, and a call to another user-composed simulation

script to launch the (parallel) simulations and post-process the results. In this case, the final outputs from DAKOTA are the values of the optimized parameters. The results for the uncharged case are shown in Table 3-5, where the first two tables are optimizations for two particular geometries, and the last is an optimization for all four geometries using a composite objective as described in the LHS study. In this example, the initial guesses based on preliminary simulations were pretty good, but the optimization procedure was still able to reduce the objective somewhat and provide improved nominal estimates for the parameters. We note that it is generally possible to reduce the objective by a greater percentage when restricting the optimization to a single geometry (Table 3, 4) as compared to an optimization over all 4 geometries (Table 5).

Table 3 Optimized values of EMMI parameters for uncharged 21-6-9 based on notched tension test data for radius = 0.039”

Parameter **R=.039”**	**Initial Values** **RMS Load Error = 61.12255**	**RMS Optimized Values** **RMS Error = 47.00786 (#659)**
C1	2.00000000e+03	1.97590887e+03
C3	4.42090000e+04	4.48753487e+04
C5	5.59399985e−02	5.68622541e−02
C13	3.66160011e+00	3.69427593e+00
C15	2.85920000e+05	2.80498426e+05

Table 4 Optimized values of EMMI parameters for uncharged 21-6-9 based on notched tension test data for radius = 0.390”

Parameter **R=.390”**	**Initial Values** **RMS Load Error = 94.77993**	**RMS Optimized Values** **RMS Error = 34.68147 (#323)**
C1	2.00000000e+03	2.15418764e+03
C3	4.42090000e+04	4.34179863e+04
C5	5.59399985e−02	5.48375186e−02
C13	3.66160011e+00	3.57702633e+00
C15	2.85920000e+05	2.94789301e+05

Table 5 Optimized values of EMMI parameters for uncharged 21-6-9 based on notched tension test data for all radii

Parameter ***All R: .039, .078, .156, .390***	**Initial Values** **RMS Load Error = 102.9047** ***Individual RMS Errors: 61.12, 51.12, 164.39, 94.78***	**RMS Optimized Values** **RMS Error = 89.42621 (#115)** ***Individual RMS Errors: (93.38, 101.12, 103.28, 48.76)***
C1	2.00000000e+03	2.04816124e+03
C3	4.42090000e+04	4.40571908e+04
C5	5.59399985e−02	5.65821483e−02
C13	3.66160011e+00	3.62171598e+00

C15	2.85920000e+05	2.90736124e+05

The improvement in the predicted load-displacement response for the geometry and optimized parameters given in Table 4, along with the corresponding prediction using the combined optimized values given in Table 5, is shown in Figure 13.

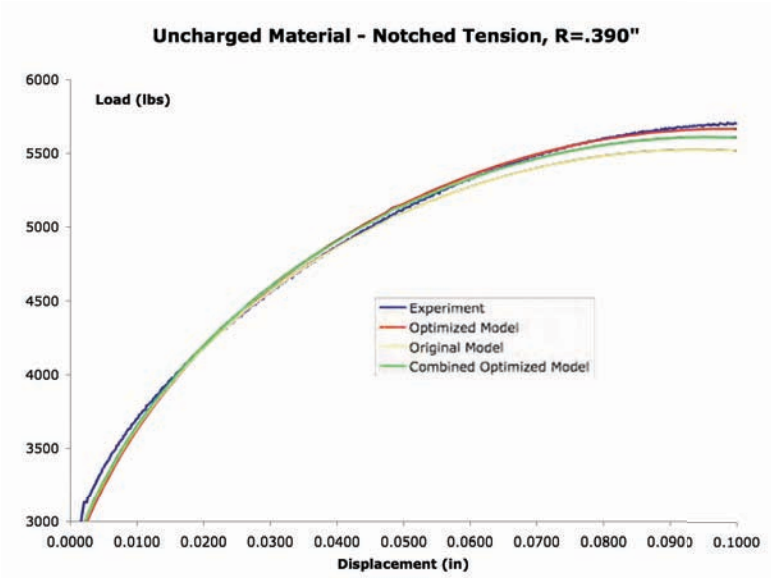


Figure 13 Comparison of FEA analyses using original and optimized EMMI material parameters for notched tension test for radius = 0.390”

Although the greatest improvement comes from using optimized parameters particular to the given geometry, both sets of values lead to modest improvements over the nominal values of the parameters. However, we note from the heading of the last column of Table 5 that the combined optimization produces, in this case, a poorer fit for the smaller notch radii relative to even the given nominal parameter values.

2.4.3 Statistical sampling study of EMMI parameters for hydrogen-charged material

Figure 14 displays a scatter plot of the composite response function, expressed as a decimal (rather than as a percent), for each sample point of the 300-point LHS study. Figure 15(a) and (b) exhibit a graphic and several fundamental statistical measures of the response distribution. In particular, the response mean is approximately 0.1125, which may be interpreted as a mean predictive error of (only) 11.25% arising from the uniform ±20% uncertainty in the five constitutive parameter inputs. Similarly the sample standard deviation is 0.0591, or 5.91%, which represents a measure of the broadness of the error distribution.

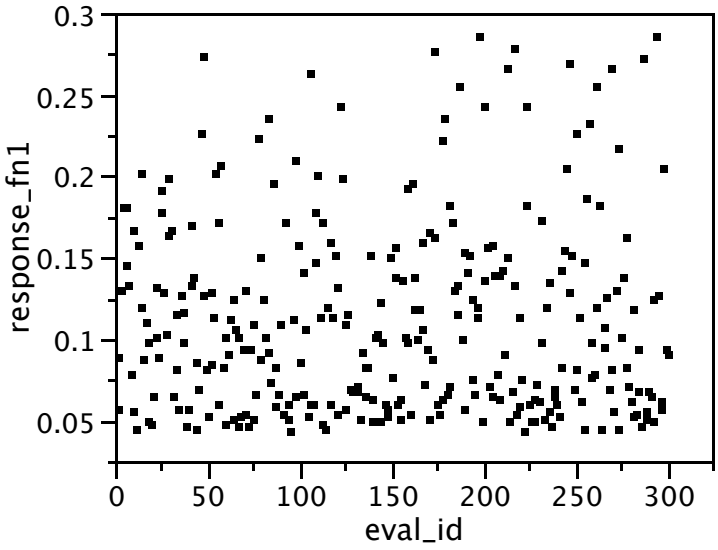
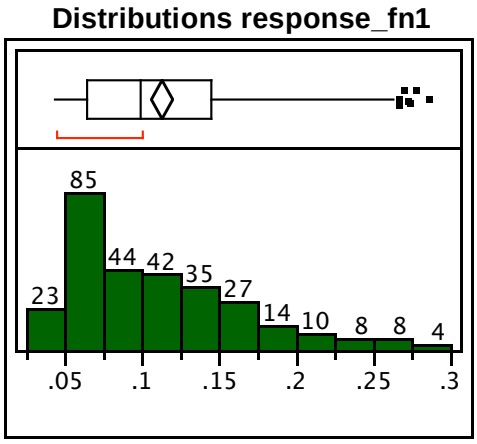


Figure 14 Response-function distribution (300 sample points)



(a)

Quantiles

100.0%	maximum	0.28659
99.5%		0.28657
97.5%		0.26790
90.0%		0.20015
75.0%	quartile	0.14441
50.0%	median	0.09896
25.0%	quartile	0.06321
10.0%		0.05156
2.5%		0.04558
0.5%		0.04376
0.0%	minimum	0.04360

Moments

Mean	0.1125343
Std Dev	0.0591043
Std Err Mean	0.0034124
upper 95% Mean	0.1192497
lower 95% Mean	0.105819
N	300

(b)

Figure 15 (a) Histogram and quantile box plot for the distribution in Figure 14. (b) Fundamental statistical analysis for the distribution in Figure 14.

The graphic itself presents the response data shown in Figure 14 in the form of a histogram, where the box-and-diamond above the histogram bars graphically summarizes the key statistical attributes of the distribution. Specifically, the two upper and lower diamond points spanning the box lie at the sample mean, and the range

between the other two diamond points within the box defines the 95% confidence interval of the mean (i.e., the probability, based on the statistical sample, that the response mean lies outside that interval is 5%). The box itself spans the interquartile range between the 25^{th} and 75^{th} quartiles, respectively (i.e., 25% of the points lie beyond each end of the box, and 50% lie within the box), and the horizontal lines (whiskers) extending beyond the box contain all points that lie within 1.5 times the interquartile range. Points beyond the whiskers represent possible outliers (i.e., points corresponding to extreme values). The vertical line across the middle of the box denotes the sample median, and the bracket along the edge of the box encompasses the shortest interval containing 50% of the response values. Figure 15(a) and (b) thus give a preliminary uncertainty quantification of the response error based on the type and range of uncertainty in the input parameters.

Depending on the need, additional characterizations of the response might include calculating more detailed statistical attributes of the output, fitting a probability distribution to the histogram data, computing a least-squares polynomial fit of the response data, and examining correlation data to determine the degree to which parameters are interconnected (i.e., correlated). For example, in this particular case, the sample correlation coefficients associated with any two parameters are all small (Table 6), strongly suggesting that the five constitutive parameters considered here are statistically independent (i.e., uncorrelated). We remark that further refinement of these and other measures of the predictive uncertainty could be achieved if it were possible to better characterize the uncertainty in the input parameters by more accurately specifying the nature of their statistical distributions (e.g., normal rather than uniform).

Table 6 Sample multivariate correlation matrix for the data in Figure 14.

	c1	c3	c5	c13	c15
c1	1.0000	-0.0149	0.0251	-0.0071	-0.0186
c3	-0.0149	1.0000	0.0021	0.0168	-0.0399
c5	0.0251	0.0021	1.0000	0.0011	0.0098
c13	-0.0071	0.0168	0.0011	1.0000	0.0304
c15	-0.0186	-0.0399	0.0098	0.0304	1.0000

3 J3D – information and verification

3.1 J3D code for calculating the J-Integral from FEA

As was mentioned in Chapter 1, Wellman developed the post-processing code J3D to calculate the J-Integral expression by Amestoy et al.[6],

$$J_{3D} = \oint_{\Gamma} \left(Wn_1 - \mathbf{T} \cdot \frac{\partial \mathbf{u}}{\partial x_1} \right) d\Gamma - \int_S \frac{\partial}{\partial x_3} \left((\boldsymbol{\sigma} \cdot e_3) \cdot \frac{\partial \mathbf{u}}{\partial x_1} \right) dS,$$

using results from finite element analysis[14]. Wellman's method for evaluating J in 3-dimensional bodies is as follows:

1. First, a plane is defined that intersects the crack front.
2. Next, at least four approximately concentric paths that enclose the crack tip are defined on that plane.
3. The path integral portion of the above expression is evaluated and a different value of "J" is recorded for each path. Also recorded is the 2-dimensional area that each path defines, "A".
4. It can be shown that the surface integral in the Amestoy expression scales like A^2. Hence, a least squares regression analysis is performed to fit the unknown coefficients in the relation: $J = C_0 + C_1 A^2$. Comparing this relation with the Amestoy formula, one realizes that the coefficient C_0, i.e. the zero area limit of the relation, yields the true J (J_{3D}).

A version of the J3D code is stored on Sandia's SHASTA institutional computing cluster, and instructions on how to compile and run the code appear in Appendix A. In addition to the J3D code, a separate Perl script was developed to define concentric element paths enclosing a crack tip for a given mesh and provide them to the J3D code. The Perl script, j3d_general_paths.pl, along with the necessary modules is included in Appendix B for reference. j3d_general_paths.pl takes six, optionally seven, inputs:

- The input exodus file. `<exodus_file>`
- The node number of the crack tip on the plane of symmetry. `<inner_crack_node>`
- The direction from the crack tip node to look for the first element in the plane. Possible inputs are: +x, -x, +y and –y where these are strings. `<crack_plane_dir>`
- The direction perpendicular to the first plane and in to the model. Possible inputs are +z and –z where these are strings. `<next_plane_dir>`
- The number of paths per plane. `<num_paths>`
- The number of planes. `<num_planes>`
- The first path offset. This is the number of elements away from the first element connected to the crack tip node on the plane of symmetry in which the

first path should start. This input is optional with a default of 1.
`<path_offset>`

Based on the input, j3d_general_paths.pl performs the following steps to determine the elements along a circular path around the crack tip (Figure 16 gives a graphical representation of the vectors in the following algorithm):

- Find the element connected to `<inner_crack_node>` in the direction `<crack_plane_dir>`
- Move in the direction, `<crack_plane_dir>`, `<path_offset>` number of elements.
- Loop over the number of planes
  - Until the path search hits a boundary, do the following:
    - Calculate `ray_vector` for the current element in the path, orthogonal to search direction.
    - Calculate search `direction_vector` for the current element in the path.
    - Find the two nodes whose `node_direction` are most orthogonal to the `direction_vector`.
    - Find the element for whose `element_vector` dotted with the `direction_vector` is maximum.
    - If no dot product is larger than a given threshold (0.75 worked well), terminate the path loop
  - From the first element in the path, find the next element in the direction `<next_plane_dir>`.

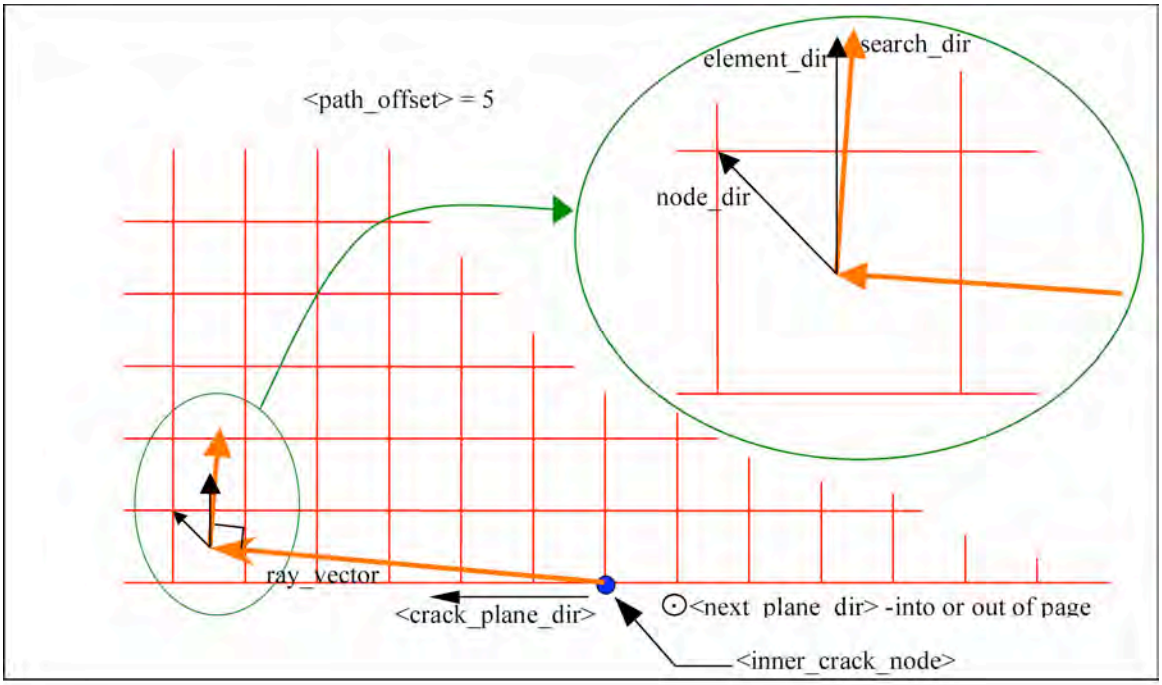


Figure 16 Graphical representation of path generation algorithm.

3.2 Performance of J3D code: Single Edge Notch Bend example

Before using the J3D code to analyze simulations of the fracture experiments, we need to verify its performance and ensure that a reliable calculation of the J-Integral will be obtained. This task was done by performing an analysis of a single edge notch bend (SENB) fracture specimen and comparing both the load-displacement and the J-load responses with semi-analytical expressions used in the ASTM E-1820 standard[22]. These expressions are developed using the Ramberg-Osgood elastic-plastic deformation model

$$\frac{\varepsilon}{\varepsilon_0} = \frac{\sigma}{\sigma_0} + \alpha\left(\frac{\sigma}{\sigma_0}\right)^n ,$$

where σ is the Von Mises equivalent stress, ε is the Von Mises equivalent strain, σ_0 is the yield strength of the material, $\varepsilon_0 = \sigma_0/E$, α is a dimensionless constant and n is the strain hardening exponent.

The ASTM standard [22] uses the EPRI J-estimation procedure[15] where the following relationships are used:

$$J = J_{\mathrm{el}} + J_{\mathrm{pl}}$$

$$J_{\mathrm{el}} = \frac{K_I^2}{E'}$$

$$K_{\mathrm{I}} = \frac{P}{B\sqrt{W}}\, f\left(\frac{a_{eff}}{W}, \frac{S}{W}\right)$$

$$\Delta = \Delta_{\mathrm{el}} + \Delta_{\mathrm{pl}}$$

$$\Delta_{\mathrm{el}} = \frac{P}{BE'} Z_{LL}\left(\frac{a_{eff}}{W}, \frac{S}{W}\right)$$

In these relations, K_I is the mode I stress intensity factor, E' is the effective modulus ($E' = E$ for plane stress and $E' = E/(1-\nu^2)$ for plane strain), P is the applied load, B is the specimen thickness, W is the specimen width, a is crack length, a_{eff} is an effective crack length, Δ is displacement at the loading line (comprised of elastic, Δ_{el}, and plastic, Δ_{pl}, portions), S is the span of the SENB geometry, and $f(\mathrm{x})$ and $Z_{LL}(\mathrm{x})$ are semi-analytical functions that depend on the specimen geometry. For the Ramberg-Osgood material model,

$$a_{eff} = a + \frac{1}{1+\left(P/P_0\right)^2}\frac{1}{\beta\pi}\left(\frac{n-1}{n+1}\right)\left(\frac{K_1(a)}{\sigma_0}\right)^2$$

$$J_{\text{pl}} = \alpha\varepsilon_0\sigma_0 b h_1\left(\frac{P}{P_0}\right)^{n+1}$$

$$\Delta_{\text{pl}} = \alpha\varepsilon_0 a b h_3\left(\frac{P}{P_0}\right)^{n}$$

where $b = W - a$, $P_0 = 1.455Bb^2\sigma_0/S$, and h_1 and h_3 are factors that depend on both geometry and the strain hardening exponent.

For our analysis, we chose the values of W = 1 in, S = 4 in, B = 0.5 in, a = 0.125 in, E = 29.85 x 10^6 psi, ν = 0.3, σ_0 = 30 x 10^3 psi, $\alpha\sigma_0/E$ = 0.002, and n = 5. For the SENB geometry and this choice of n, h_1 = 0.687 and h_3 = 15. Our choice of B also dictates a plane strain analysis. The mesh of the SENB geometry appears below in Figure 17. Only half of the SENB geometry needs to be meshed, as symmetry boundary conditions are used on the left side face. Fixed boundary conditions are used on the front and back faces to emulate the plane strain condition. Also, two meshes were analyzed: one using normal hexahedral elements everywhere and another using special "collapsed" hexahedral elements for the region adjacent to the crack tip. In his report[14], Wellman notes that such collapsed elements are necessary for accurate computation of the J-Integral.

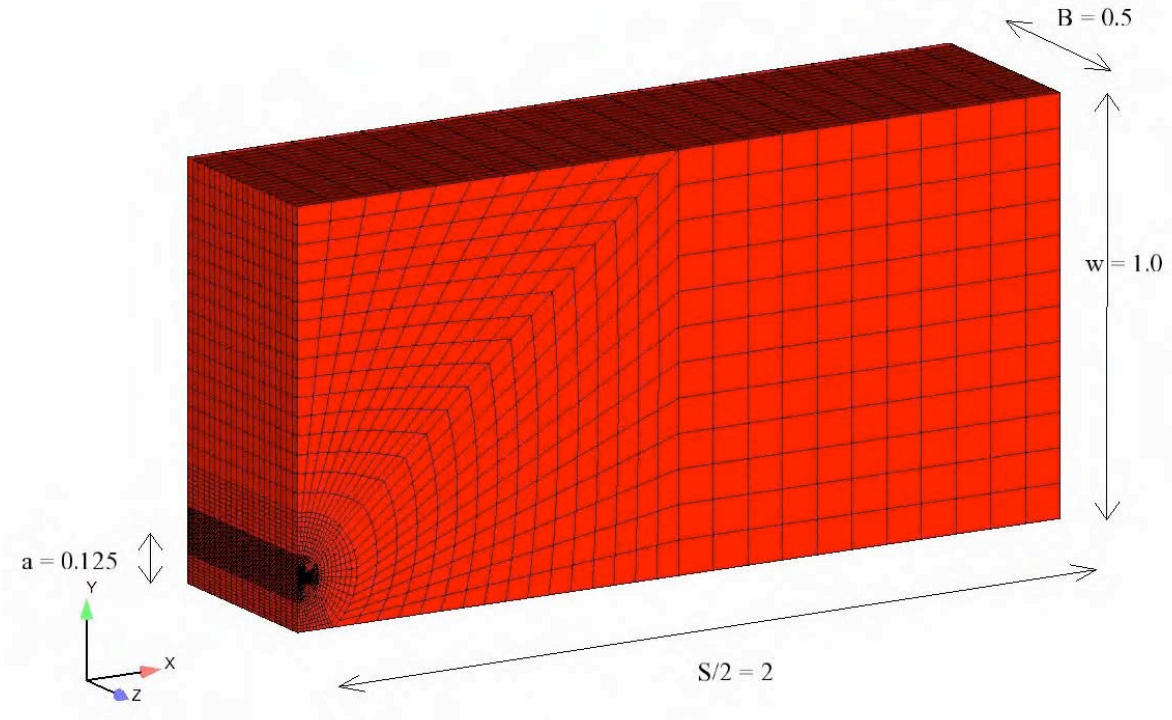


Figure 17 Mesh of SENB specimen.

In Figure 17, we notice that 20 elements were used across the thickness of our geometry. We also examined the use of a single element across the entire thickness, as no variations of stress, displacement or strain should exist across the thickness of our specimen. The ADAGIO finite element code is used to simulate the loading of this fracture geometry. Displacements are applied along the left side face of the beam in increments of 0.0005" and the resulting stress and deformation fields for the FE system is determined at each increment. The calculation is performed using 16 processors.

Figure 18 shows the collapsed, hexahedral meshes colored by values of Von Mises effective stress. The 20 element mesh is shown on the left, while the single element mesh is shown on the right. While the maximum values of stress appear in the expected locations, i.e. at the crack tip and at the positions of applied loads and restraints, it is interesting to note that higher values of stress are apparent for the mesh that uses a single element across its thickness, as compared with the 20 element mesh. However, both meshes do not display any noticeable variations across the thickness. This is more clearly observed Figure 19, which shows close-up views of the crack tip regions.

Figure 20 shows the collapsed, hexahedral meshes colored by values of equivalent plastic strain. Again, we notice that the variation of plastic strain is quite different for the single element mesh as compared with the 20 element mesh, with higher values obtained near the applied loads/restraints and lower values obtained near the crack tip.

In Figure 18 and Figure 20, we also notice large deformations in several regions of the SENB mesh. It should be noted that while collapsed elements provide the correct stress singularity for small-strain analysis, the use of such elements for large deformations should be done with caution.

(a)

(b)

(c)

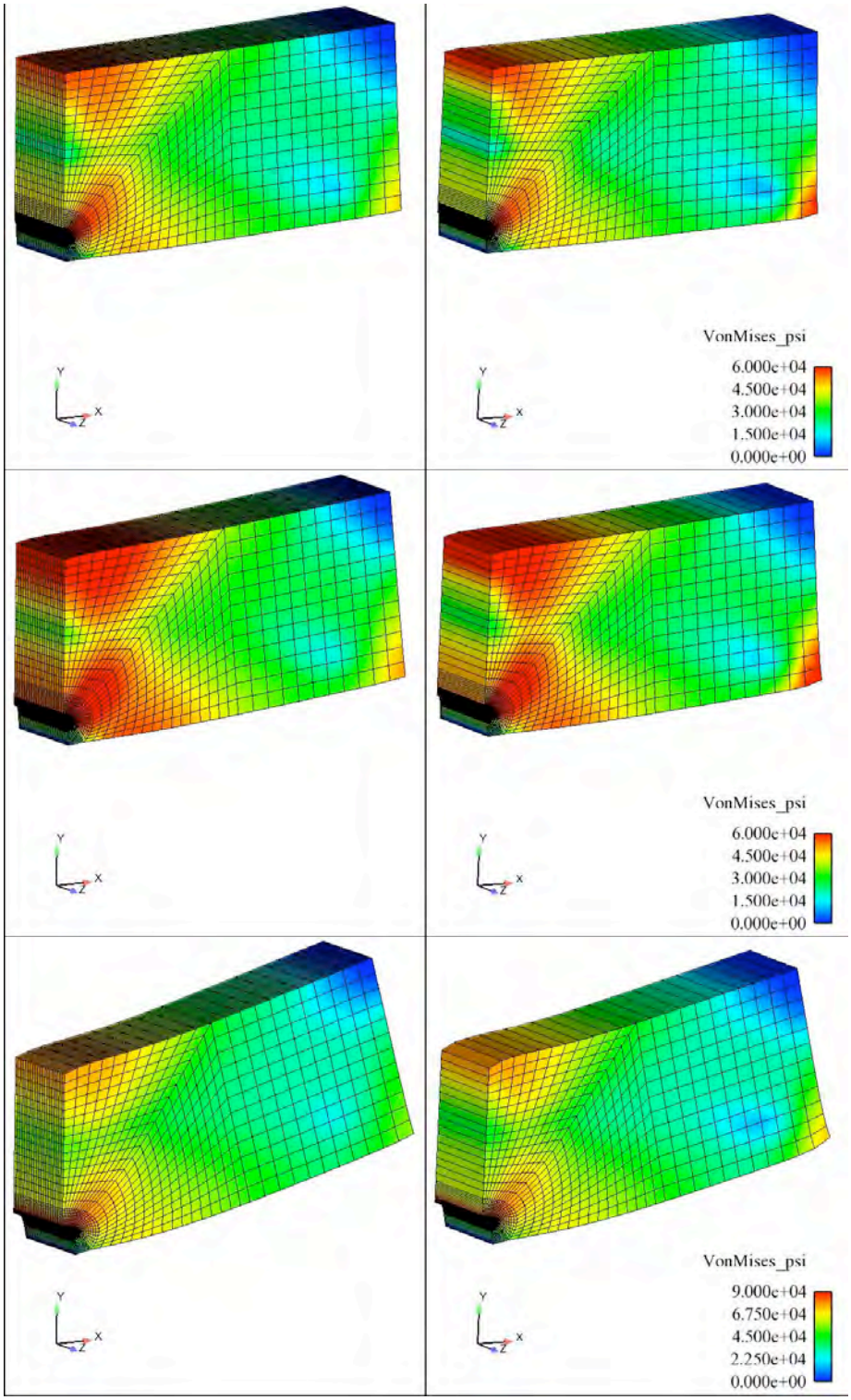


Figure 18 Deformed meshes of SENB geometry at three increments of load line displacement (a: 0.1 in, b: 0.2 in, c: 0.5 in) colored by elemental values of Von Mises stress.

(a)

(b)

(c)

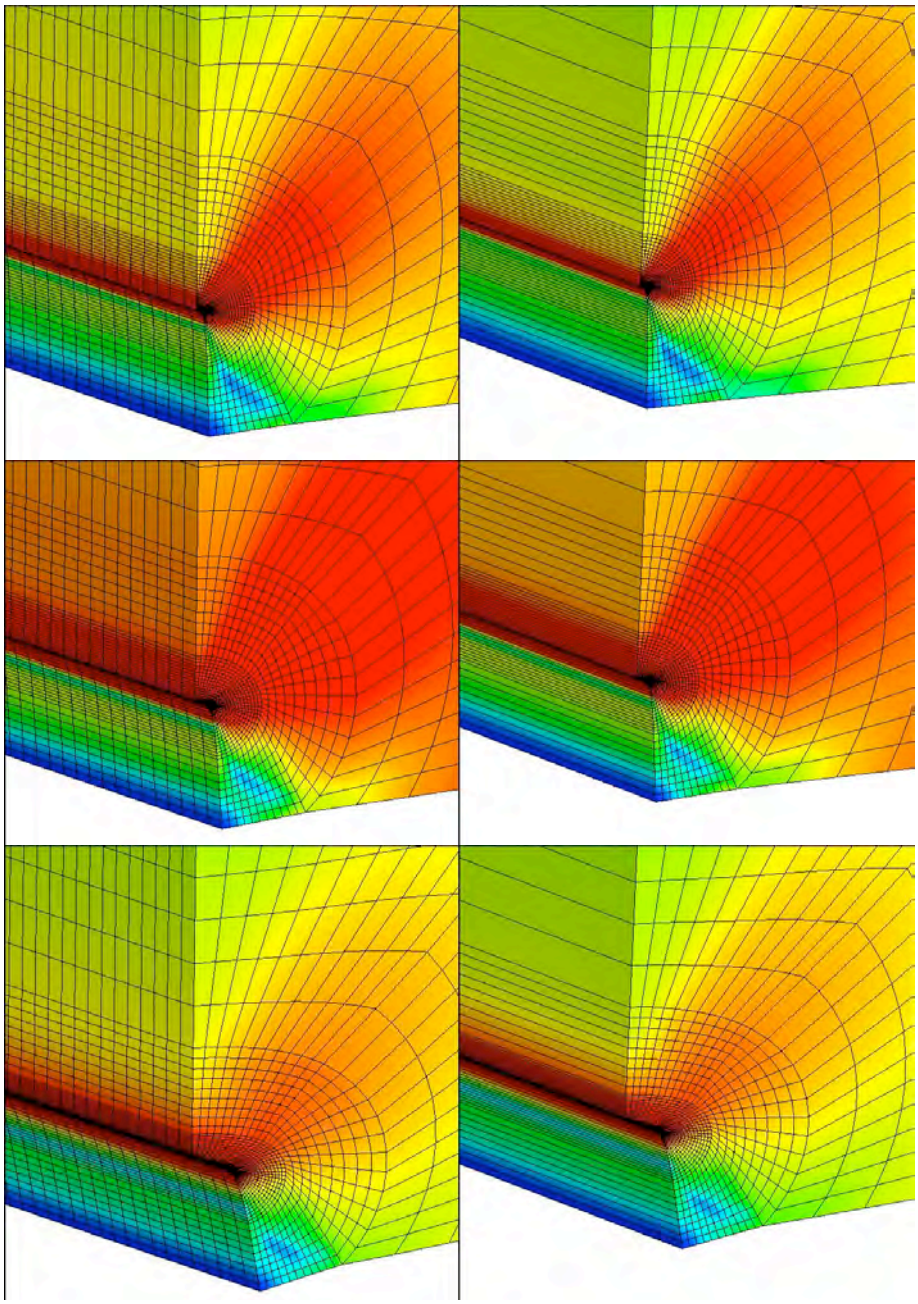


Figure 19 Close-up views of the pictures shown in Figure 18.

(a)

(b)

(c)

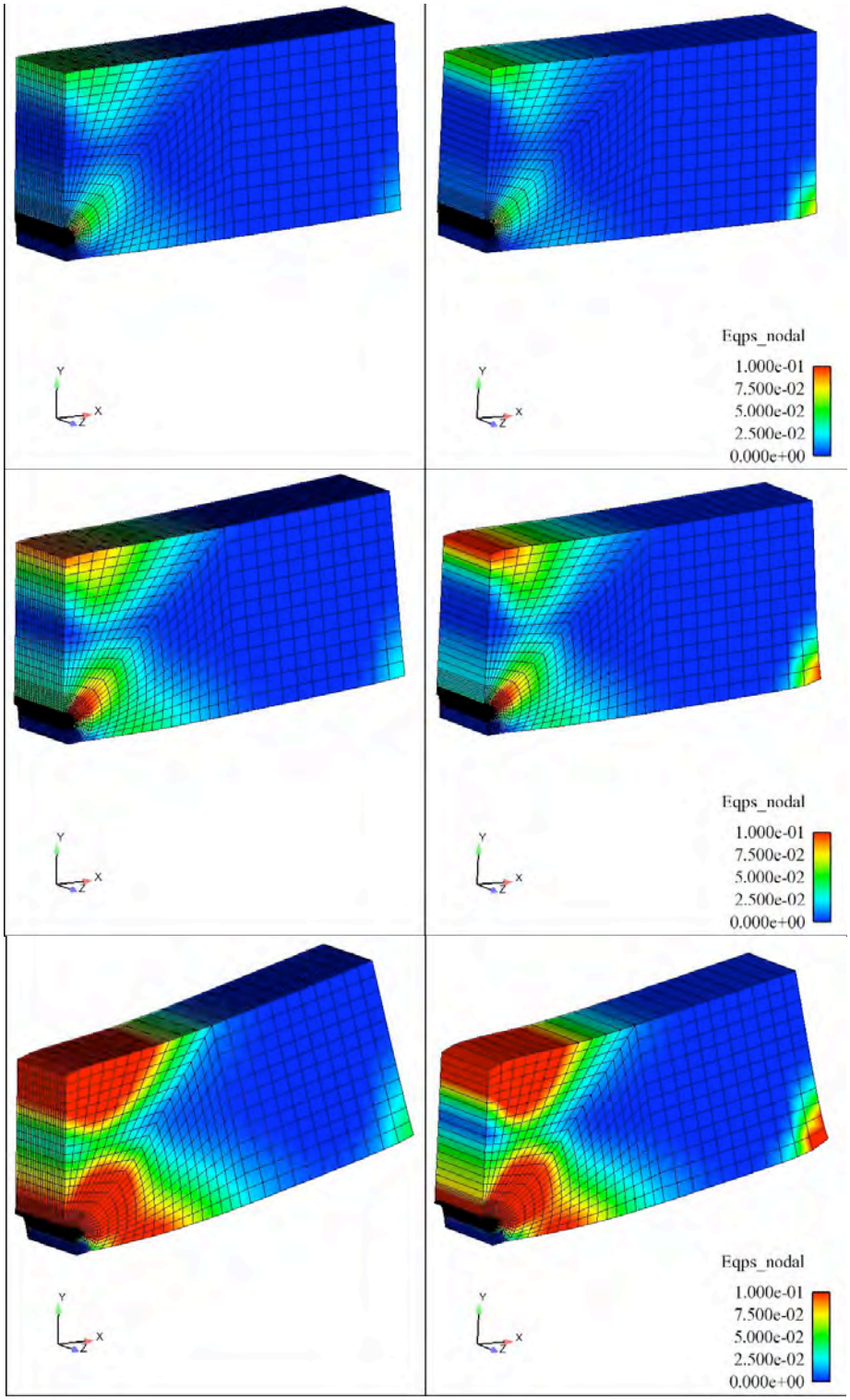


Figure 20 Deformed meshes of SENB geometry at three increments of load line displacement (a: 0.1 in, b: 0.2 in, c: 0.5 in) colored by elemental values of equivalent plastic strain.

Figure 21 shows the load-displacement curve for the analyses of the collapsed element meshes. It is observed that the FEA solution for the 20 element mesh is in

complete agreement with the semi-analytical EPRI solution, whereas the single element mesh exhibits some disagreement, i.e. a lower load is required for the same load line displacement. This behavior was discussed with ADAGIO code developers and it was recommended that higher order (shape function) elements be used for the single element thickness analysis. However, such higher order elements were not available for the elastic-plastic model used in these analyses and further investigation on this issue is recommended for future work.

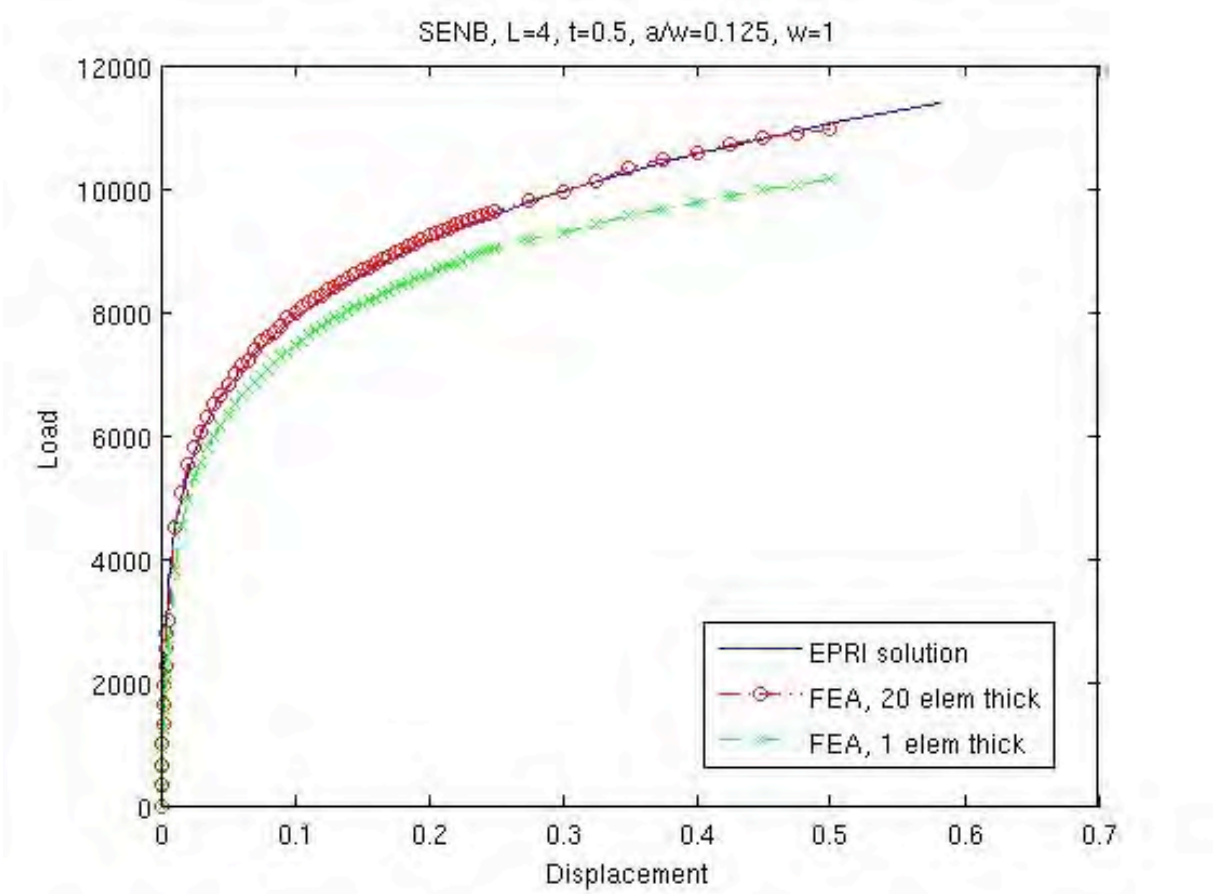


Figure 21 Load-displacement curves for the SENB fracture specimen. Displacement units are inches and load units are lbs.

Figure 22 shows the resulting J-load curves calculated using the J3D code. The curve corresponding to the 20 element mesh shows good agreement with the EPRI solution over the range of loads up to 11,000 lbs. A more direct comparison is shown in Figure 23, which displays the fractional difference between the FEA and EPRI estimates of J as a function of load. It is observed that the J3D code under-predicts the EPRI estimate of J for the range of loads in the elastic regime and at the onset of plastic deformation, and then over-predicts J for regimes of extensive plastic deformation. The error in J ranges from -40% to +40%, as seen in the Figure.

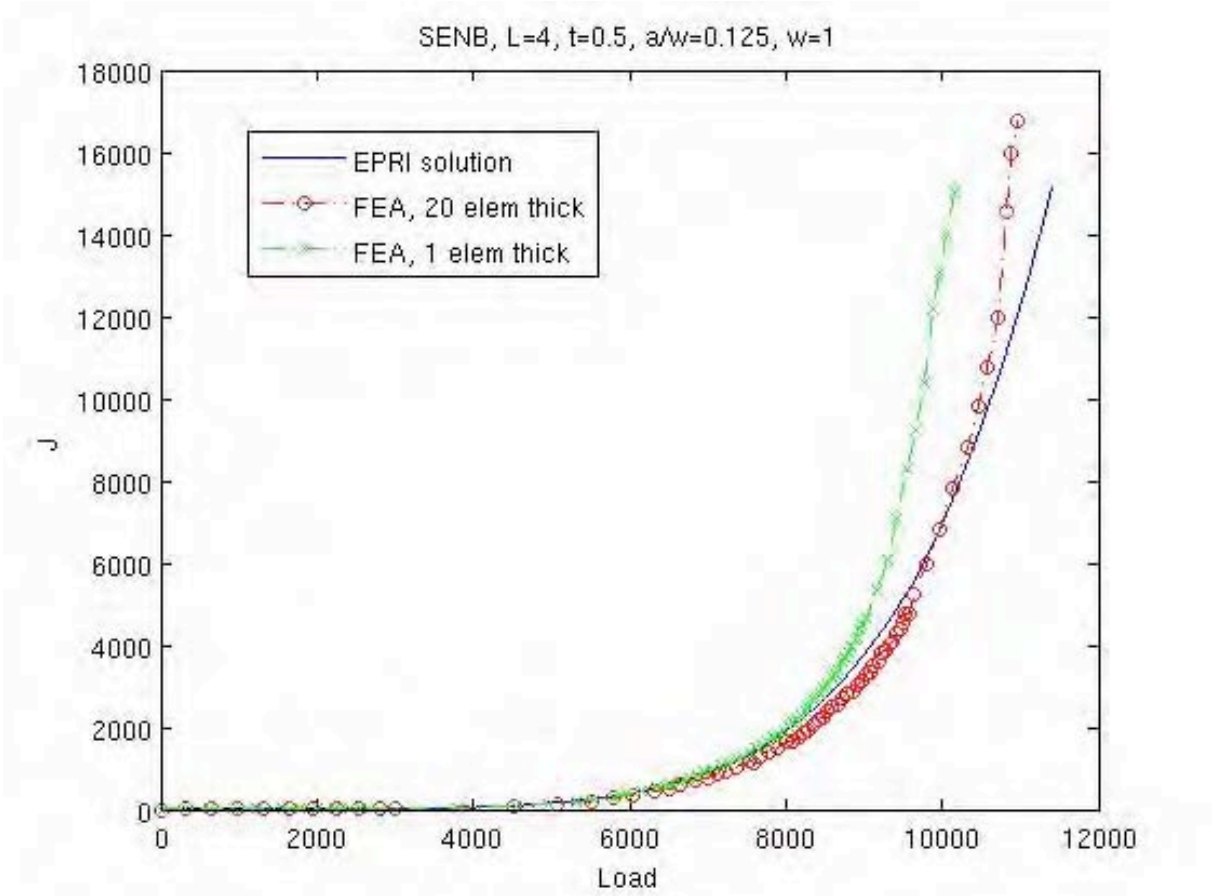


Figure 22 J-load curves for the SENB fracture specimen. Load units are lbs and J units are in-lbs/in^2

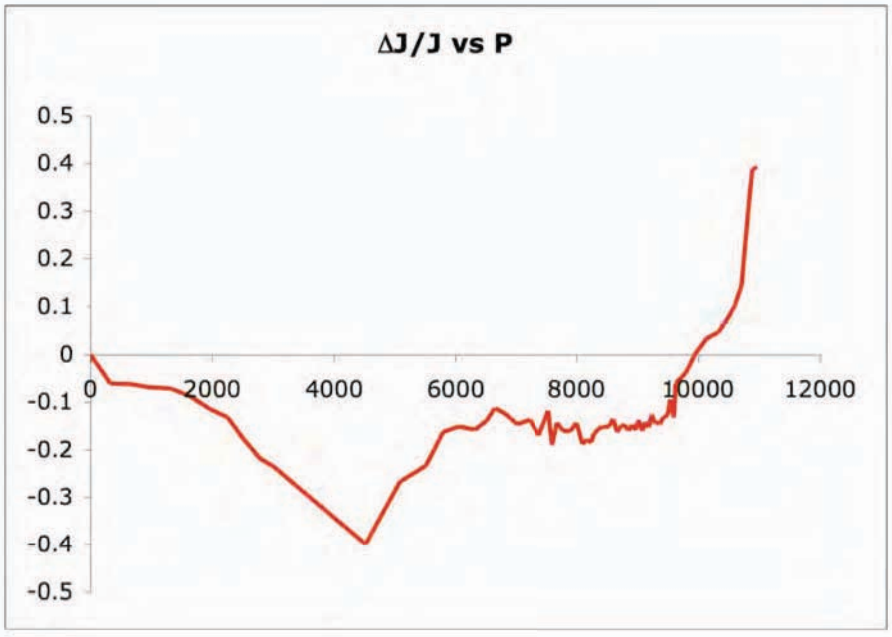


Figure 23 ΔJ/J-load curve for the 20 element mesh of the SENB fracture specimen.

For completeness, we present and briefly discuss our analysis results using the conventional hexahedral elements. Figure 24 shows the load-displacement curves for both coarse and refined hex meshes as compared with the curves for the collapsed hex meshes from Figure 21.

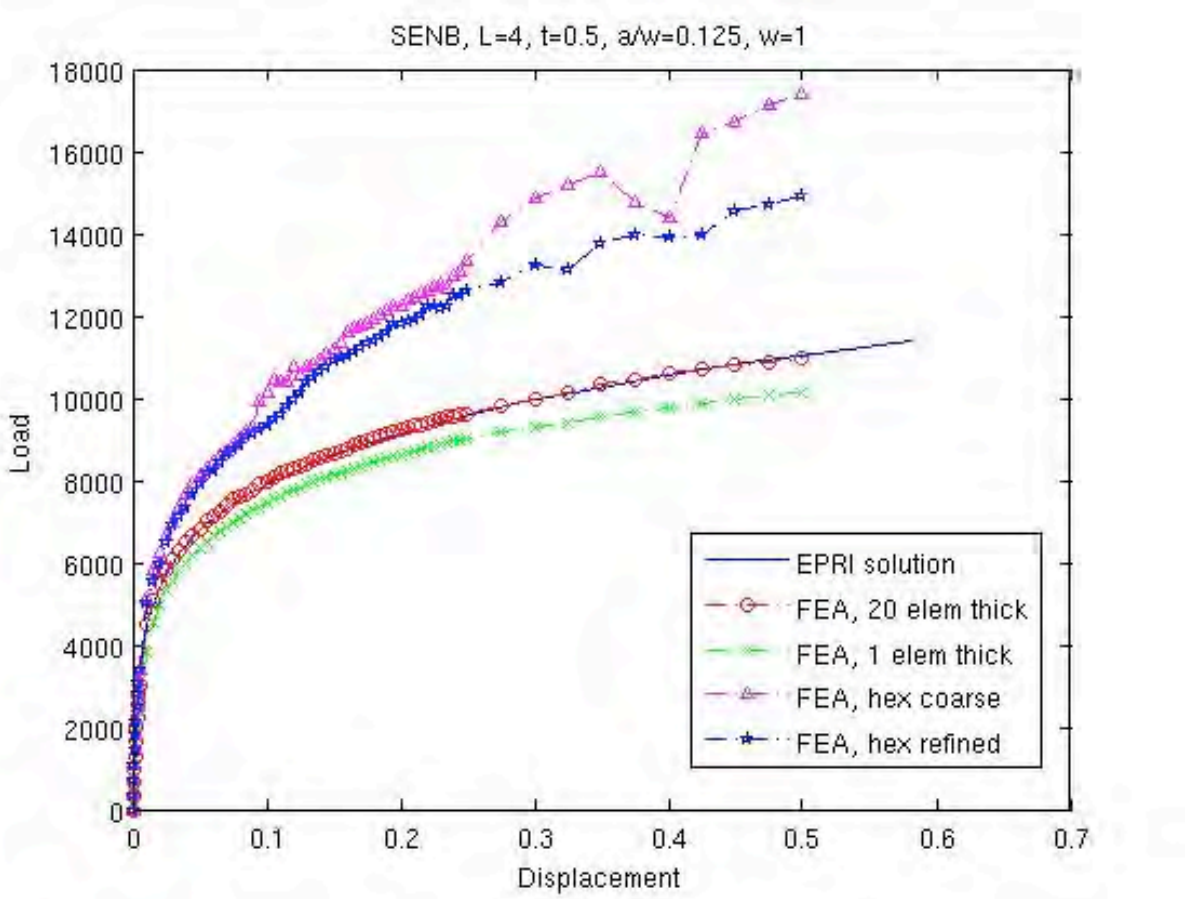


Figure 24 Load-displacement curves for the SENB fracture specimen. Displacement units are inches and load units are lbs.

This figure clearly shows a much different response of the hex mesh from the collapsed hex mesh. A close-up examination of the crack tip region shows radically different displacements and a dramatic difference in the crack tip blunting that occurs. The hex mesh also exhibits a deformed shape for the whole SENB geometry that contains non-physical features such as perturbations along the top edge of the specimen. While it is not apparent what causes these differences and irregularities, it is clear that Wellman's advice to use collapsed elements is sound.

Figure 25 shows the J-load curve for the hex and collapsed hex meshes. The hex mesh vastly under-predicts the EPRI solution for the entire range of load values. Given the results shown in Figure 24, this behavior is anticipated.

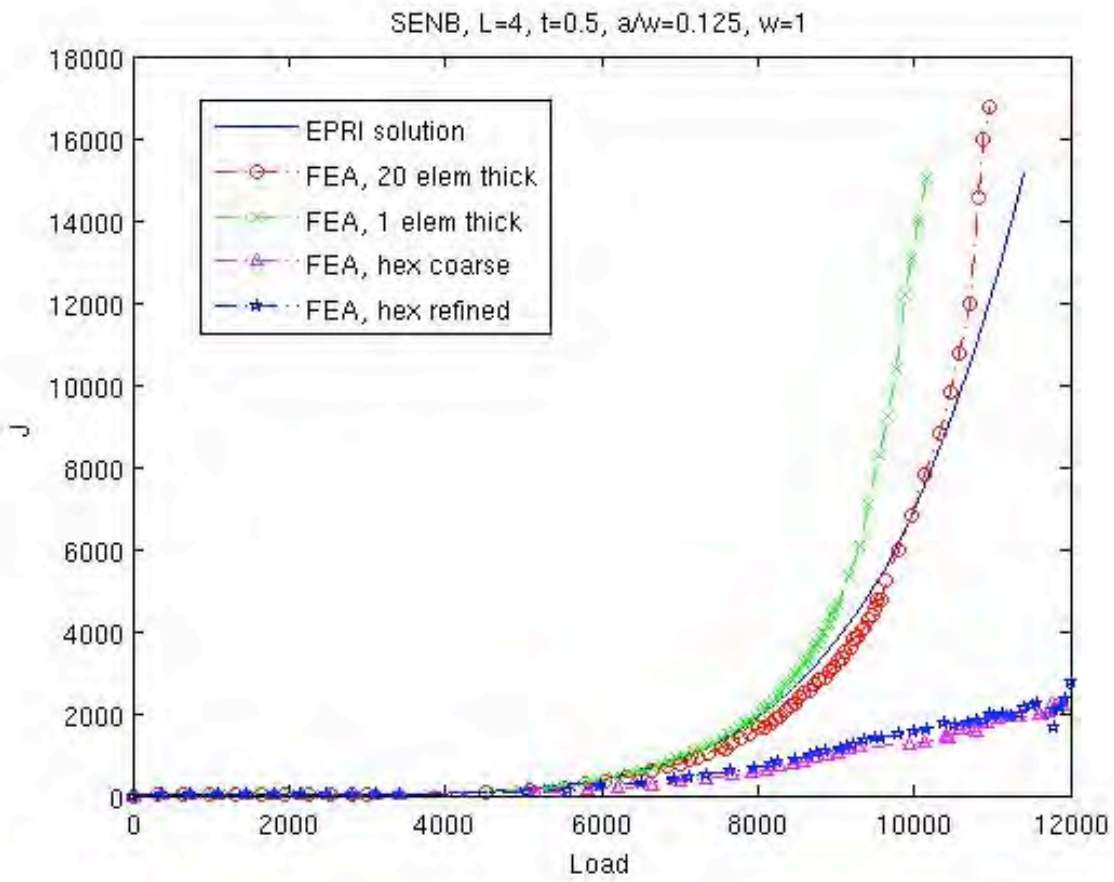


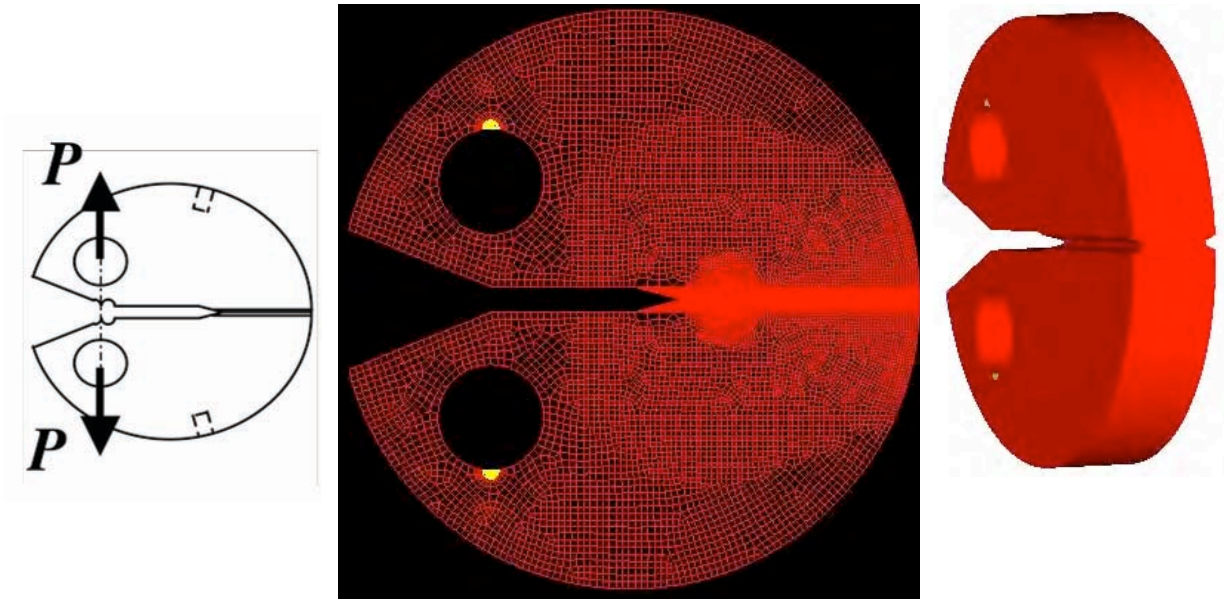
Figure 25 J-load curves for the SENB fracture specimen. Displacement units are inches and load units are lbs.

4 Two Dimensional Compact Tension Fracture Experiments and Simulations

In previous chapters, we have validated our elastic-plastic material model and have verified the performance of both our finite element code ADAGIO and the J3D code for calculating the J-Integral in a 3-dimensional body. We now attempt to validate the performance of our FEA capabilities and the J3D code through comparison of simulation and J calculations with fracture experiments performed on a 2-dimensional, Compact Tension (CT) geometry. Experiments were performed by the separately funded C6 project mentioned above, and details about these experiments can be found in [21].

4.1 Analysis methods

Figure 26 shows the geometry of the disk-shaped Compact Tension (CT) fracture specimen. The diameter of the specimen is 2.5", with a distance of 1.85" between the load line and the far end of the disk, i.e. W. For the analysis of the uncharged 21-6-9, a crack length of 0.93345" is used (a/W = 0.50457), which is a mean of values from the two experiments for which data was collected (0.9322" and 0.9347"). For the analysis of the hydrogen charged 21-6-9, a crack length of 0.92275" is used, a mean of values from the two experiments for which data was collected (0.9185" and 0.927"). Our mesh, like its real-world counterpart, contains side grooves along the expected crack propagation plane. This is done to enforce the plane strain condition at the crack tip and ensure uniform crack driving force along the crack front.



(a) (b) (c)

Figure 26 Disk-shaped Compact Tension Fracture Specimen: (a) loading configuration, (b) 3D FEA mesh containing ~ 300,000 elements, (c) off-diagonal solid rendering

The mesh analyzed is actually ¼ of the geometry shown in Figures 25(b) and (c). Symmetry boundary conditions are used along the crack plane and mid-way through the thickness. This mesh contains a total of 306,004 hexahedral elements (2,538 for the elastic loading-pin region and 303,466 for the elastic-plastic region) and 322,680 nodes. To assess the mesh dependency of our results, a refined mesh was also analyzed that contains 968,817 hex elements (8,280 for the elastic loading-pin region and 960,537 for the elastic-plastic region) and 1,003,892 nodes. The typical element size near the crack tip was on the order of 0.003".

The ADAGIO finite element code is used to simulate the loading of these fracture geometries. The loading-pin regions are displaced in increments of 0.0003" and the resulting stress and deformation fields for the FE system is determined at each increment. These computations are performed using between 32 and 40 nodes on the SHASTA institutional computer cluster, each of which is configured with dual 3.06 GHz Intel Xeon processors and 2GB RAM. Analyses required between 12 and 36 hours of compute time, depending on the size of the mesh analyzed and the number of processors used.

4.2 Uncharged 21-6-9

Figure 27 shows the load-displacement curve for the CT specimen composed of uncharged 21-6-9 stainless steel. Our analysis was performed using the EMMI parameters originally determined (listed in Table 1), and then was redone using the optimized parameters (listed in Table 5). This figure includes the curves from our analyses along with the corresponding curves from the experiments as report in [21].

It is observed that our analyses very closely agree with the loading curves measured in experiment. In particular, the two sets of curves show quantitatively similar values in both load and displacement in the region where plastic deformation becomes dominant, i.e. the "bend" in the load-displacement curve. Disagreement between analysis and experiment at high levels of displacement (deformation) is understandable as localized material unloading is probably occurring in the real material but is not allowed in the material model chosen for this analysis.

It is interesting to note that use of the optimized model parameters negligibly affects the resulting load-displacement curve. The two analysis curves are only noticeably different at very high levels of displacement, where the validity of the analysis model is already in question due to the issue of deformation localization.

Figure 28 shows estimated error percentages in load between the analysis using the original parameters and the two experimental curves. From this figure, it is apparent that a large level of disagreement exists in the displacement range dominated by elastic deformation. The cause of this disagreement was unable to be determined. Several explanations were pursued to explain this disagreement, and the culprit is believed to be a difference in material orientation between the tensile test specimens

used to fit the material model and the CT specimens. This difference was not anticipated to lead to the amount of error observed, but other potential causes of the error were investigated, including dimensions of the side grooves and rate effects, but was not found to account for the observed error. Future investigations to isolate the effect of material orientation are warranted.

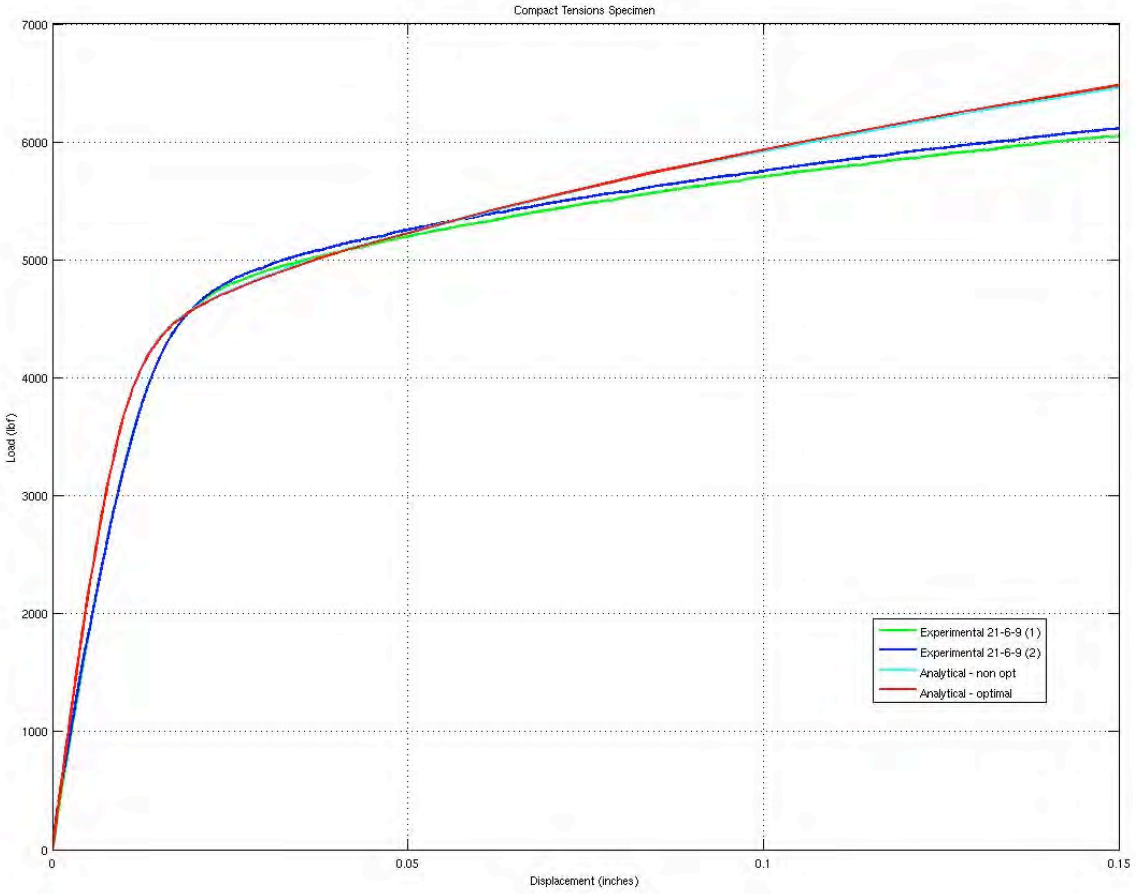


Figure 27 Load-displacement curve for CT fracture experiment and analysis of uncharged 21-6-9.

Figure 28 also shows that agreement between analysis and experiment is quite strong as plastic deformation begins to be dominant over elastic deformation, i.e. at displacements of 0.02" and higher. In this regime, the error essentially stays within the margins of ±5%.

One of the goals of this project is to characterize the effect that variations in crack length have on the deformation response of a material. Figure 29 shows the load-displacement curve for an FEA simulation of a CT specimen with a 9% longer crack, a = 1.0175" (a/W = 0.55). There is no particular significance to this choice of a 9% crack length increase; rather, the results presented here were generated for a mesh created early-on in the project and that was later deduced to possess a longer crack as compared with the experimental specimens. The elastic deformation is very similar to the experimental results from the specimen with the shorter crack, showing only a slight decrease in stiffness of the initial deformation response. Also, the shape

of the curve at high levels of deformation is very similar to both of the experimental curves; the 3 curves appear essentially parallel to one another. However, a dramatic reduction in the load at the onset of plastic deformation is observed with the load at bend reducing from a value of 4,500 lbs to 3,500 lbs, a change of over 22%. This example clearly shows that uncertainties in crack dimensions may result in non-linear changes of the deformation response, justifying the use of large safety margins to prevent catastrophic failure.

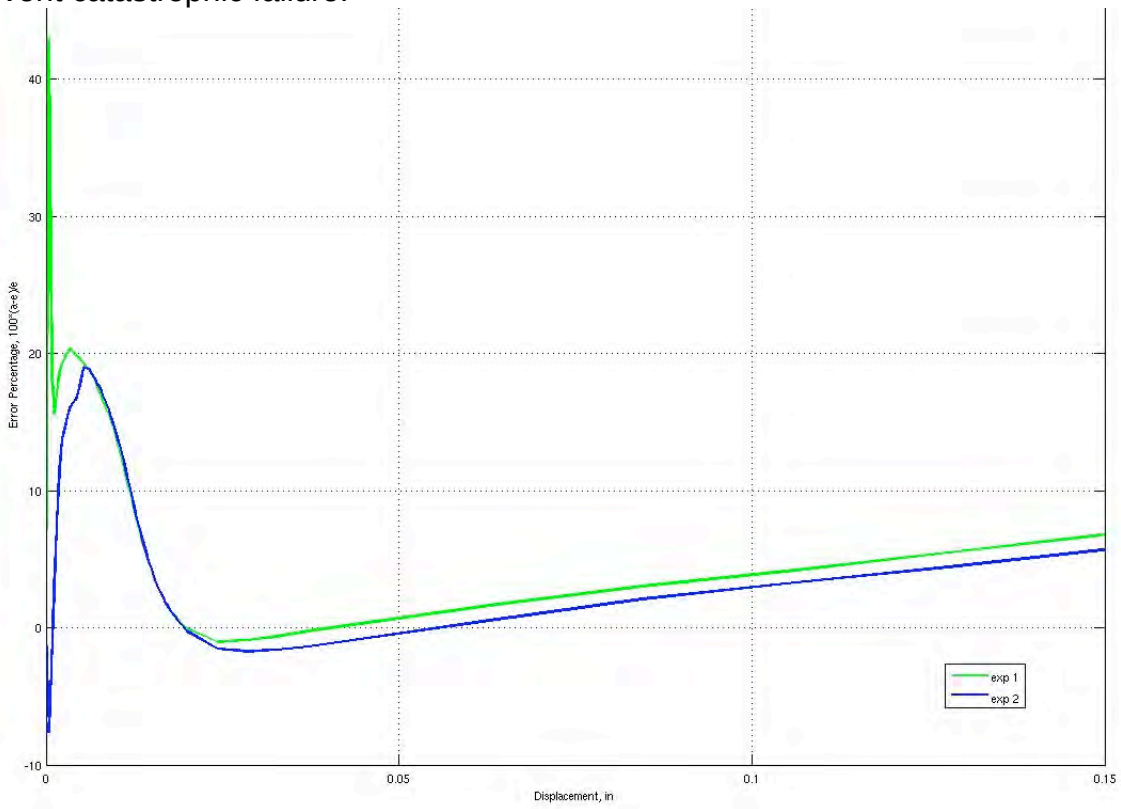


Figure 28 Error analysis of FEA results as compared with experimental data for uncharged 21-6-9.

4.3 Hydrogen charged 21-6-9

Figure 30 shows the load-displacement curve for the CT specimen composed of hydrogen charged 21-6-9 stainless steel. Our analysis was performed using the EMMI parameters originally determined and listed in Table 2. This figure includes the curves from our analyses along with the corresponding curves from the experiments as report in [21].

It is observed that our analysis approximately agrees with the loading curves measured in experiment, although the agreement does not appear to be as good as for the uncharged material. Similar trends are noticed for the elastic dominant regime, the load “bend” and the plastic dominant regime. Again, disagreement between analysis and experiment at high levels of displacement (deformation) is understandable as localized material unloading is probably occurring in the real material but is not allowed in the material model chosen for this analysis.

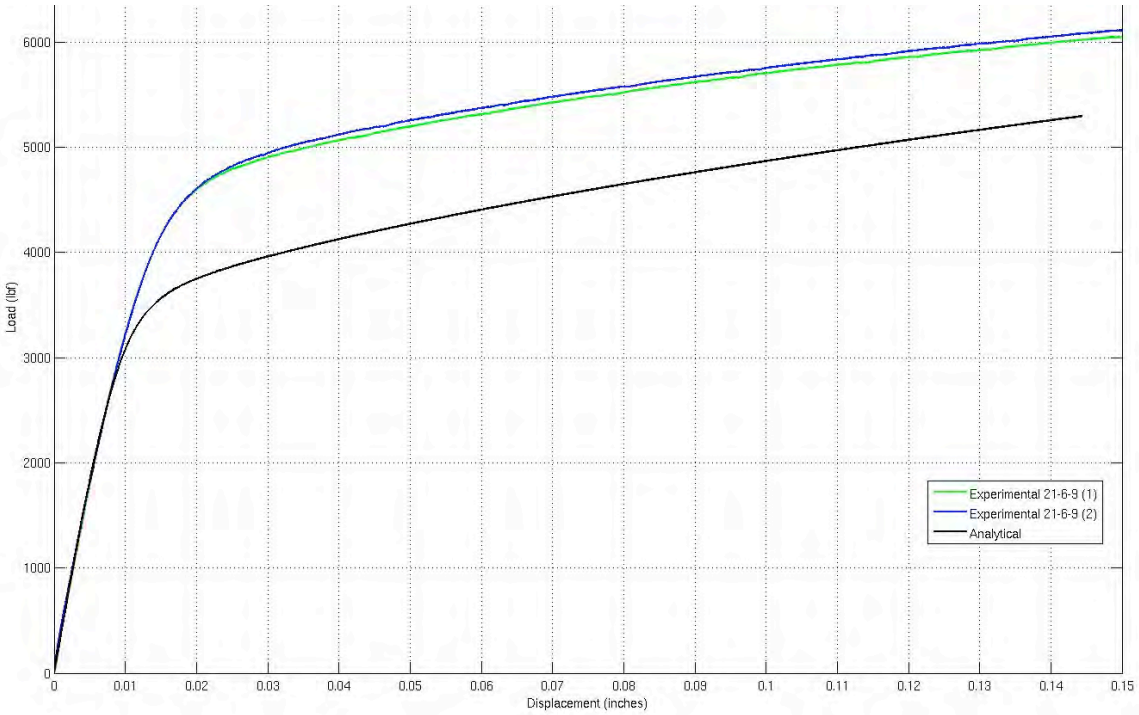


Figure 29 Load-displacement curve for analysis of CT fracture specimen with 9% longer crack than shown in Figure 27.

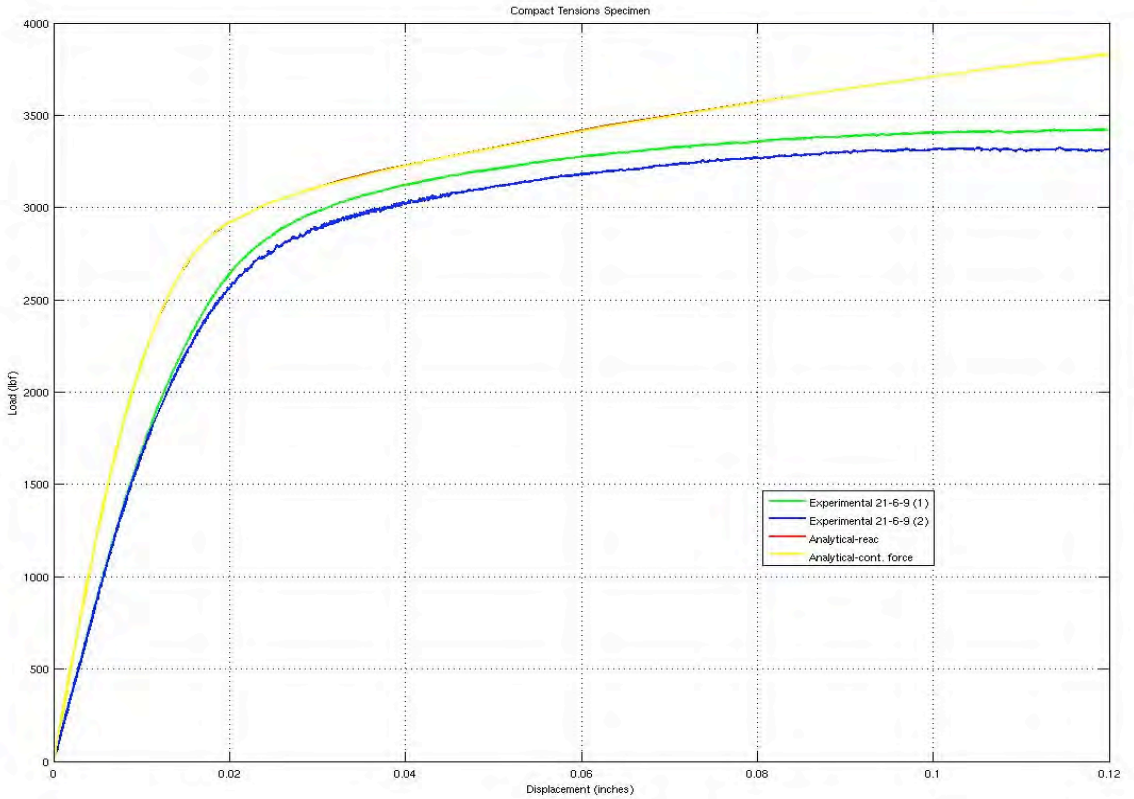


Figure 30 Load-displacement curve for CT fracture experiment and analysis of hydrogen charged 21-6-9.

Figure 30 also shows that while some disagreement between analysis and experiment exists with regard to the displacement at which the load bend occurs, the magnitude of the load at bend is approximately equal to 2,900 lbs for both. The error between analysis and experiment is more clearly shown in Figure 31. In this figure, the observed error is very high (almost 45%) initially, but rapidly decreases to between 10 and 15% at the displacement corresponding to the load bend. Upon further deformation, this error decreases more, and remains under 15% for the range of displacement simulated. As for the uncharged case, we are uncertain as to why our analysis overpredicts the stiffness in the elastic regime of the load-displacment curve.

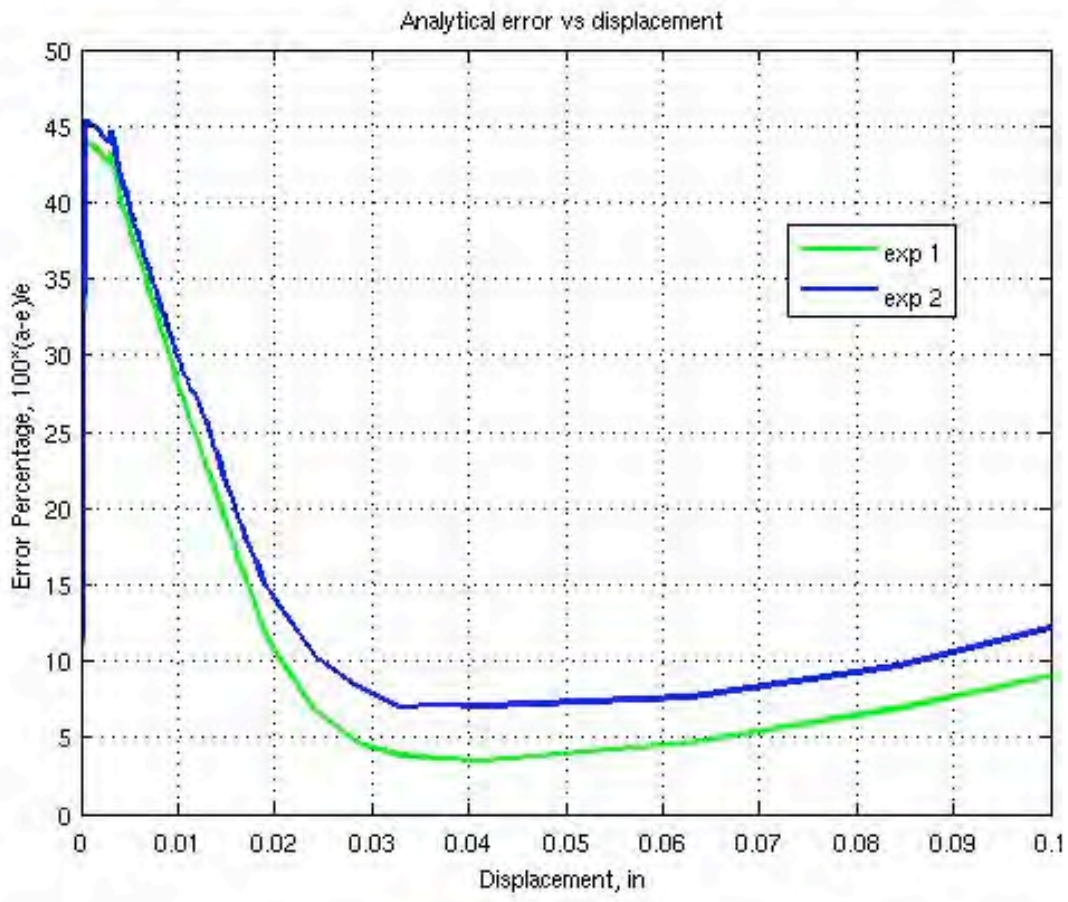


Figure 31 Error analysis of FEA results as compared with experimental data for hydrogen charged 21-6-9.

As with the uncharged material, we examined an analysis variation containing a slightly longer crack. Figure 32 shows the load-displacement curve for an FEA simulation of a CT specimen with a 16% longer crack, a = 1.0715" (a/W = 0.5792). Again, there is no particular significance to the choice of a 16% crack length increase; rather, the results presented here were generated for a mesh created early-on in the project and that was later deduced to possess a longer crack as compared with the experimental specimens. As before, the elastic deformation is similar to the

specimen with the shorter crack and exhibits a decrease in stiffness of the initial deformation response. Also, a dramatic reduction in the onset of plastic deformation is observed with the load at bend reducing from a value of 2,900 lbs to 1,900 lbs, a change of under 35%. This calculation again clearly shows that uncertainties in crack dimensions manifest in non-proportional changes in the deformation response.

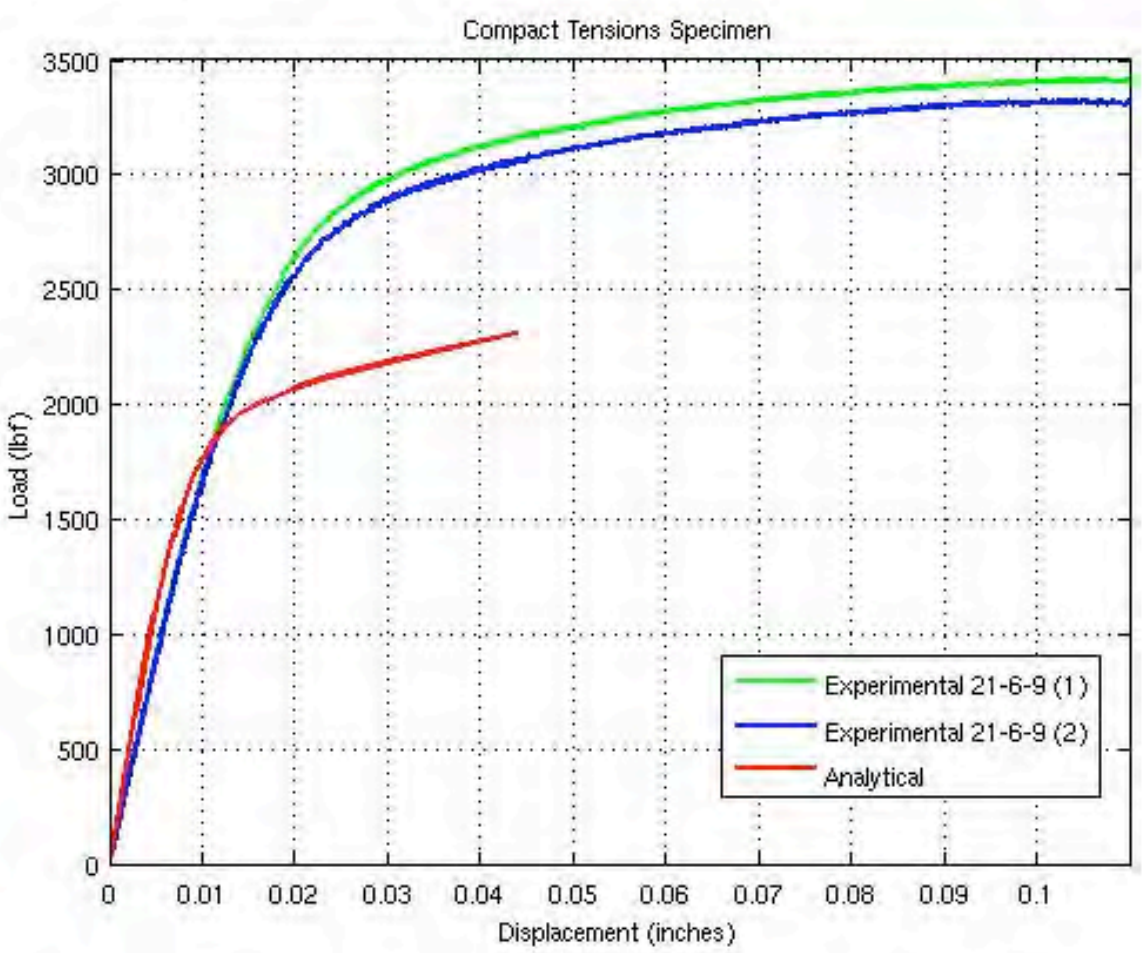


Figure 32 Load-displacement curve for analysis of CT fracture specimen with 16% longer crack than shown in Figure 30.

4.4 J-Integral analysis

The J3D code was used to estimate the J-Integral as a function of load using the load-displacement calculations shown in Figure 27 and Figure 30, along with the corresponding stress-strain results for those ADAGIO analyses. We first examine the case of the hydrogen charged material since the corresponding experiment resulted in brittle fracture and a value of fracture toughness (J_c) was measured to be approximately 1,900 in-lbs/in^2. Figure 33 shows the J-load curve calculated from our analysis on two separate planes: one at the mid-plane of the meshed geometry and the other plane 10 planes away from the mid-plane (a distance of approximately

0.03”). For both planes, 8 paths are used to calculate the value of J, as discussed in section 3.1. These curves are compared with curves calculated from the experimental data using the method detailed in ASTM Standard E-1820[22]. Details on the experiments can be found in [21].

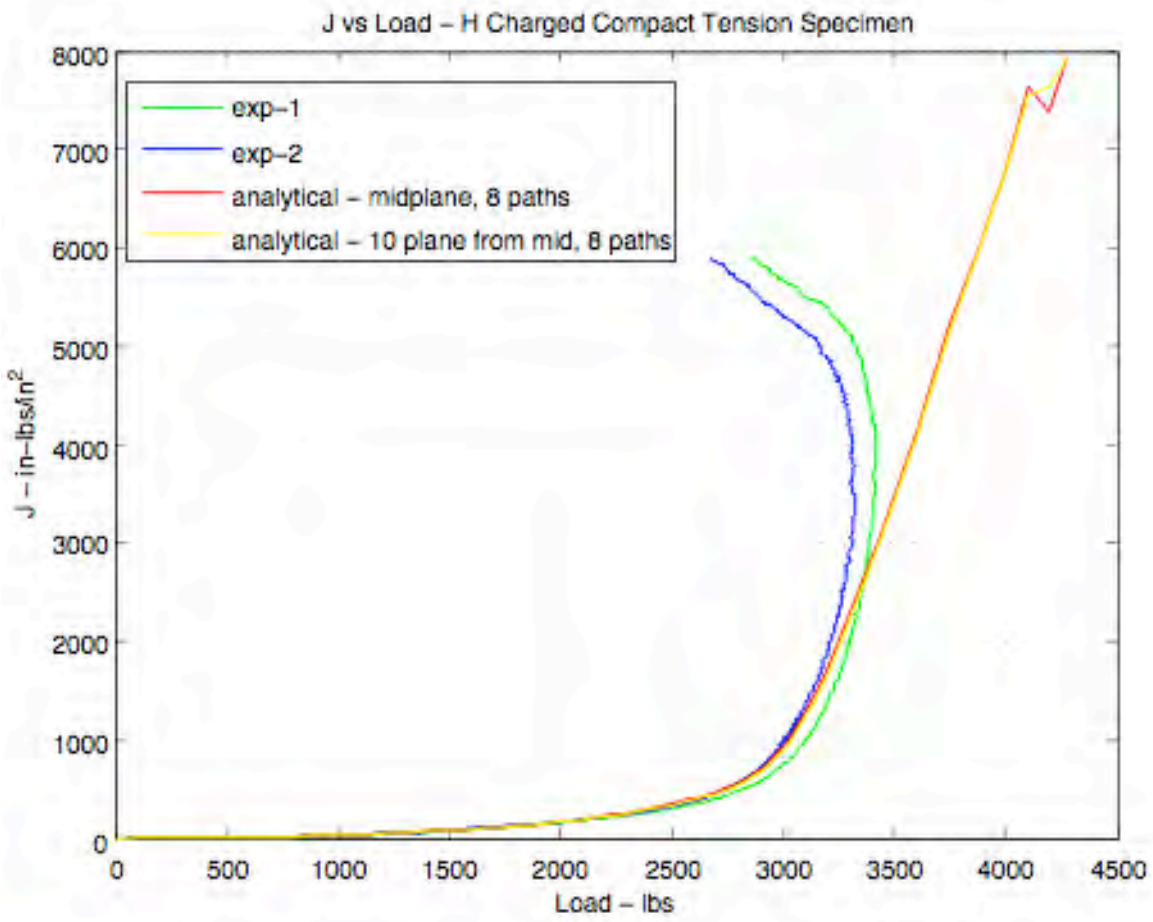


Figure 33 J versus load curve for CT fracture experiment and analysis of hydrogen charged 21-6-9 (J_c ~ 1900 in-lbs/in^2).

It is observed that despite the disagreement noted in the load-displacement diagram (Figure 30), the J3D code very closely follows the J-load quantified in experiment. This agreement is excellent up to and including the value of fracture toughness (1,900 in-lbs/in^2). Beyond this value of J and load, the analysis and experimental curves differ considerably, an expected observation since fracture has occurred in the experiment but cannot be replicated in our finite element model.

Figure 34 shows the J-load curves computed by J3D for the analysis of the uncharged material. For this case, no value of fracture toughness was quantified in experiment as the crack blunted and plastically deformed before any brittle-like propagation was observed. Comparing the analysis curves with the experimental ones, we note that the J3D code over-predicts the value of J for the load range

between 4,000 and 5,700 lbs. The load range corresponds to the deformation just prior to well beyond the load bend observed in Figure 27. Hence, for this range the J3D code conservatively overestimates the crack driving force as compared with the driving force measured in experiment. The discrepancy appears to be quite large, roughly 100% at a load of 5,000 lbs., although the value at which the discrepancy is most relevant is not clear since a fracture toughness value has not been quantified for the uncharged material.

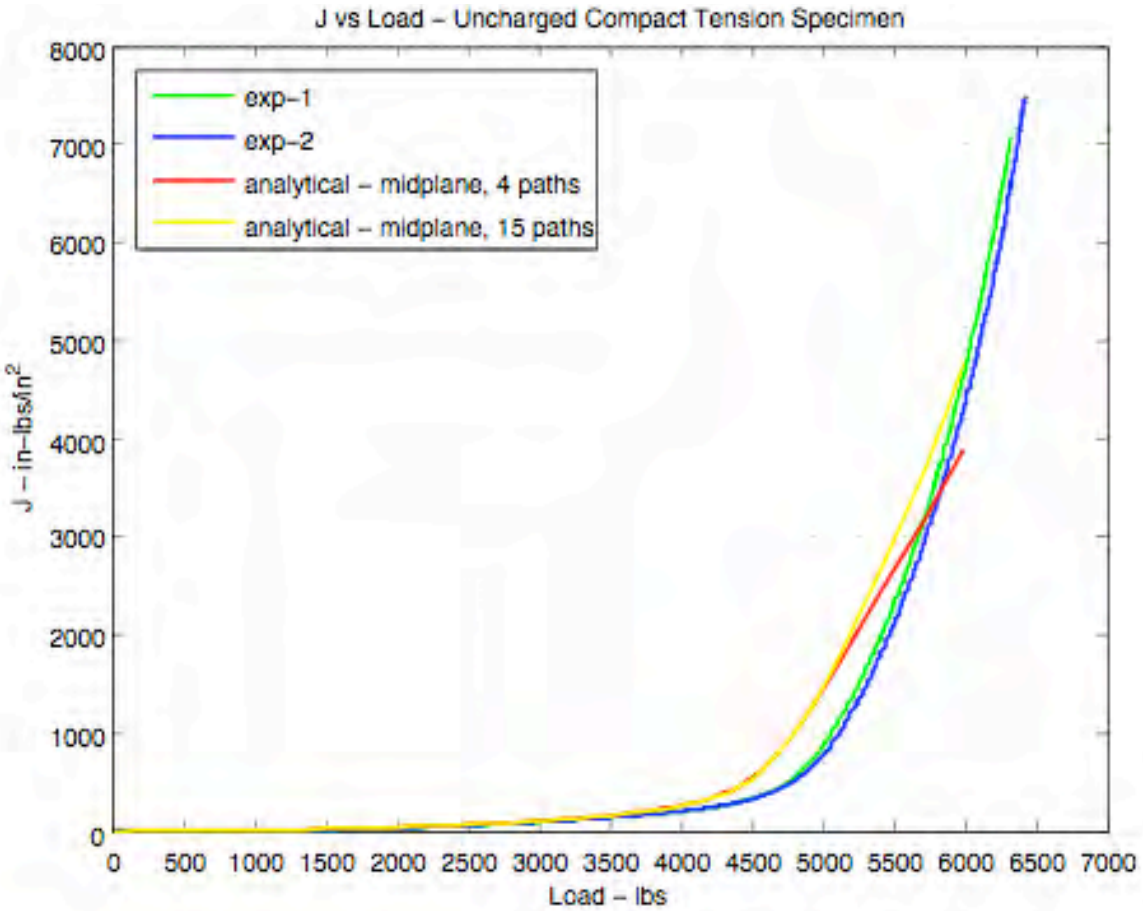


Figure 34 J versus load curve for CT fracture experiment and analysis of uncharged 21-6-9 (J_c unknown).

This page intentionally left blank.

5 Three Dimensional Fracture Experiments and Simulations

Three dimensional fracture specimens were modeled and analyzed in an attempt to further validate the use of finite element analysis and the J3d code to determine the feasibility of using this method to qualify GTS reservoirs. The three dimensional geometries chosen are more similar to the shape and loading of a real system than the compact tension specimens. As with the two dimensional specimens, both annealed and hydrogen charged specimens were studied. All test specimens — notched tension, compact tension, and cracked round bar — were created from the same bar stock to avoid variations in properties due to lot differences.

The symmetric specimens were circumferentially cracked round bars (CRB) with axi-symmetric pre-cracks of varying crack ratios. The results from the varying crack ratio tests and analyses will allow comparison of the accuracy of the evaluation methods based on crack lengths. These specimens yielded load versus displacement data, and the J-integral value was calculated using analytical methods from this data to compare against the finite element analysis and J3d results. These experimental and analytical methods and results will be documented in a separate report[21].

The asymmetric specimens were similar to the symmetric CRB specimens, but side notches were cut perpendicular to the cylinder from one or both sides of the bar prior to pre-cracking. Pre-crack methods caused oval or sometimes completely asymmetric initiation profiles. The most regular samples were chosen for analysis comparison of the load versus displacement relationship.

A summary of the three dimensional specimens that were modeled and analyzed is shown in Table 7.

Table 7 Definition and naming scheme for 3-d cracked round bar specimens.

	Symmetric		Asymmetric	
	Test #	**a/r**	**Test #**	**a/r***
Annealed	a6	0.208	b1	0.30/0.75
	a8	0.507		
	a9	0.605		
Hydrogen -charged	a4	0.19 - 0.25	b2	0.23/0.69
	a7	0.485		
	a1	0.600		

*Crack ratios for the asymmetric specimens are given in the major axes of the pre-cracked cross section for comparison purposes.

It should be noted that the initial crack dimensions are calculated from measurements taken after the samples are loaded to failure, under the assumption that the crack ratio remained constant throughout the test. The crack measurements are scaled for

the original specimen diameter, using a linear relationship. Some of the discrepancies between the experimental results and the analysis results may be attributed to this assumption, particularly in the annealed specimens, which have high ductility. As discussed in the compact tension specimen section, initial pre-crack assumptions have a large impact on bulk deformation behavior of the specimen.

Meshes with standard hexes at the crack tip were created using Cubit. Cubit currently does not have an automated method for creating collapsed elements; meshes with collapsed elements at the crack tip were created in TrueGrid. All analyses were run using the sierra code ADAGIO on the CA institutional computing cluster Shasta. Depending on the availability of resources and size of the model, 24-48 processors were used. The finite element models ranged from 100,000-300,000 elements.

(a)

(b)

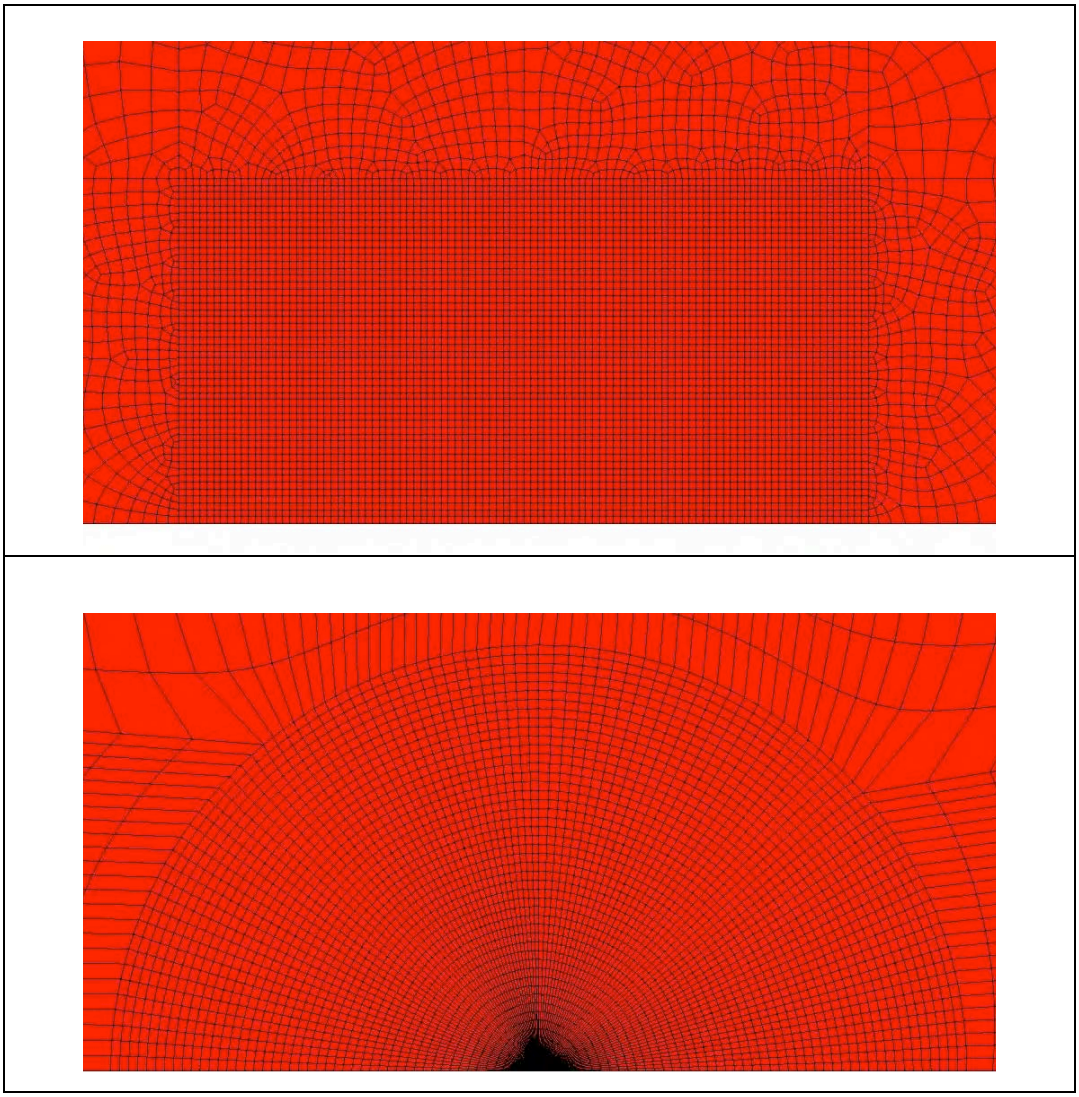


Figure 35 Close up mesh of crack tip with (a) standard hex elements, and (b) collapsed hex elements.

The symmetric specimens were modeled with both standard, 8-noded hexes throughout and also with the collapsed hexes at the crack tip. Crack tip meshes of with standard hexes and collapsed hexes are shown in Figure 35. The region surrounding the crack tip with a fine, regular grid has a radius of approximately 0.031 inches for both the collapsed and standard hex meshes. Regions beyond the original regular grid have elements quickly increasing in size. Detailed discussion of the collapsed elements versus conventional hex elements can be found in the SENB specimen section. For our 3-dimensional geometry analyses, we observe that bulk load versus displacement behavior of the specimen are only slightly affected by the elements at the crack tip; however, J versus load behavior much more closely matches expected values when using the collapsed elements at the crack tip.

It should be noted that mesh spacing in the radial direction is slightly (≤ 5%) smaller for the regular hex mesh than for the one using collapsed elements. However, because the number of elements in the theta direction is constant for the collapsed element mesh, the mesh density in that direction increases closer to the crack tip with elements of siginficantly larger aspect ratio than used in the regular hex mesh.

For each of the cracked round bar specimens, load versus displacement and J-integral versus load results from the finite element analysis, as well as the errors from the experimental results, will be presented. Because both curves contain servere bends associated with the load plateau, one section of the plot will be nearly "vertical", yielding very high errors on any small deviation from the experimental curve. These excessively large errors do not provide any useful comparisons of our data, and we are most interested in the plastic regime. Thus, we will present load errors as a function of displacement and load errors as a function of J. The second may seem counterintuitive as J is calculated from the stresses in the model, but calculating errors in this manner provides more useful information than calculating J errors as a function of load.

5.1 Symmetric, Circumferentially Cracked Round Bar Specimens

In order to adequately refine the mesh at the crack tip to capture the behavior in this region without allowing the model to become unmanageably large, the symmetric CRB specimens (A1, A4, A6-A9) were modeled as 30-degree sectors with cylindrical boundary conditions applied at the 0 and 30 degree faces. Symmetry was also used across the crack plane, yielding an overall 1/24 model of the system. The analysis model was meshed very finely at the crack tip, and elements further away from the crack tip were increasingly larger.

All samples were created from 0.75-inch diameter bar stock. The outer diameter for the medium and long cracked specimens was 0.75 in. Specimens A6 and A4, with the short cracks, had additional machining performed after the initial pre-crack, which yielded an initial specimen diameter of 0.5 inches.

The finite element model and loading conditions for the short cracked specimens are shown in Figure 36. The machined axi-symmetric groove was removed during the second machining process, yielding a cylindrical specimen with radius 0.25 inches, pre-cracked by approximately 20% of the specimen radius.

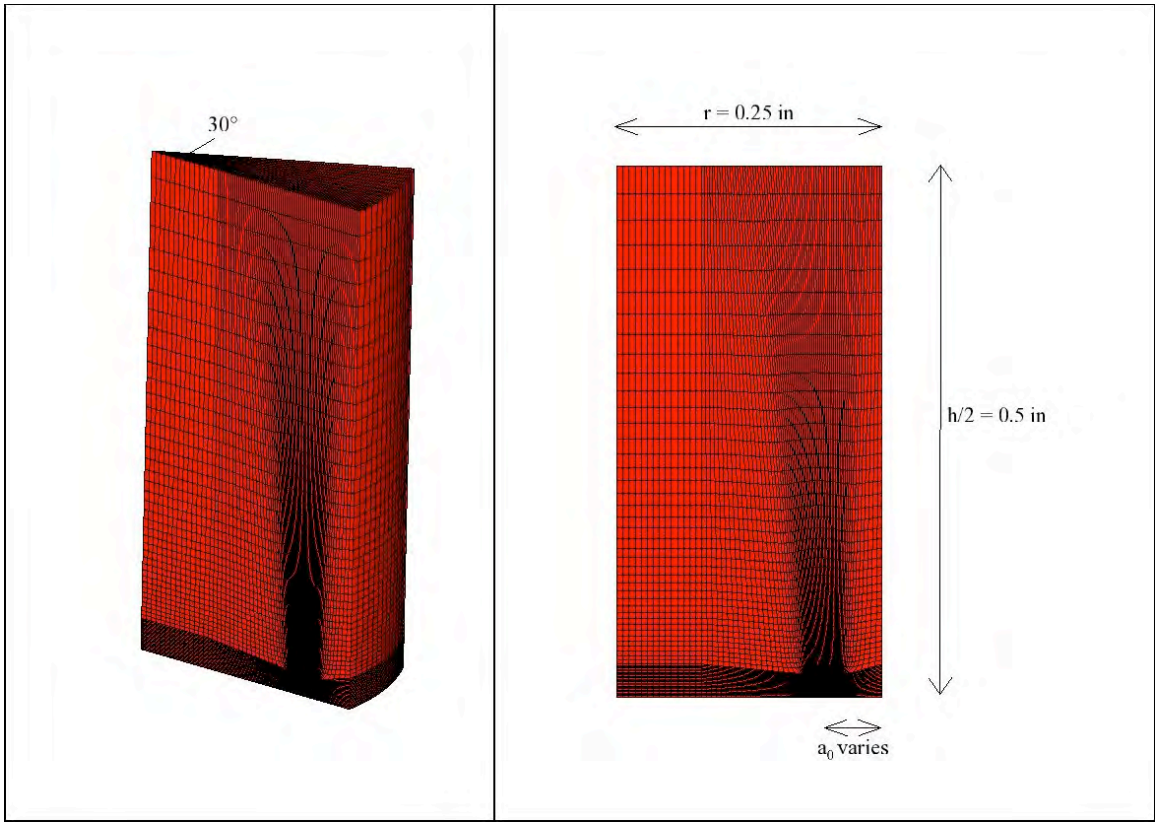


Figure 36 Finite element mesh of a symmetric, circumferentially cracked round bar with a short crack (specimens A6 and A4).

A sample mesh of the medium and long cracked CRB specimens (A1, A7-A9) is shown below in Figure 37. The length of the initial pre-crack differs between these remaining specimens, but all other measurements remain constant.

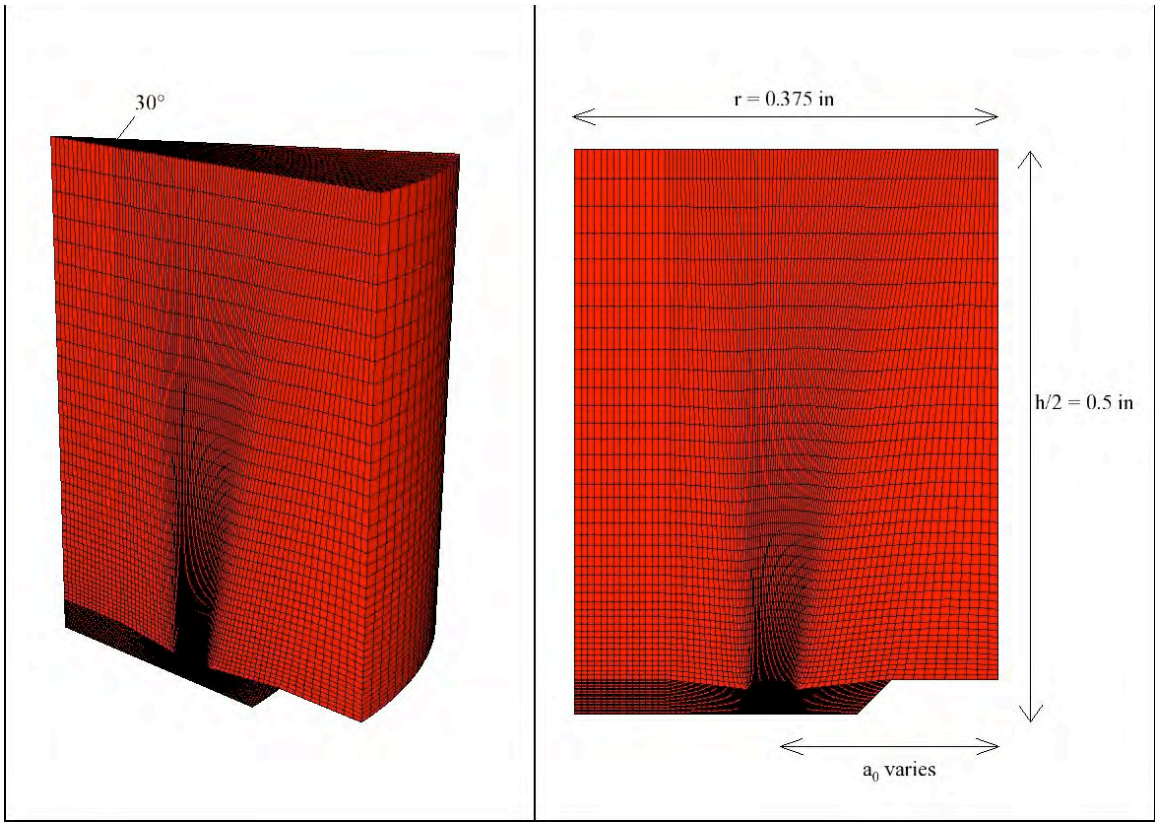


Figure 37 Finite element mesh of a symmetric, circumferentially cracked round bar with medium or long crack (specimens A1, A7-A9).

There is no explicit maximum number of elements per path stated in J3d manual [14]; however, it was determined through trial and error that there is a maximum number of elements per path allowed by the code. Initial mesh refinement yielded 192 elements per path, which caused J3d to fail catastrophically with no meaningful results. Halving the number of elements per path to 96 allowed J3d to complete calculations without problem.

5.1.1 Uncharged Symmetric 3D Specimen: Short Crack (A6)

The uncharged short crack specimen had an a/r ratio of 0.208 and an initial radius of 0.5 inches. Figure 38 shows the stress state just beyond yielding and at the peak load. The yield stress of the material is 1.6e5 psi.

(a)

(b)

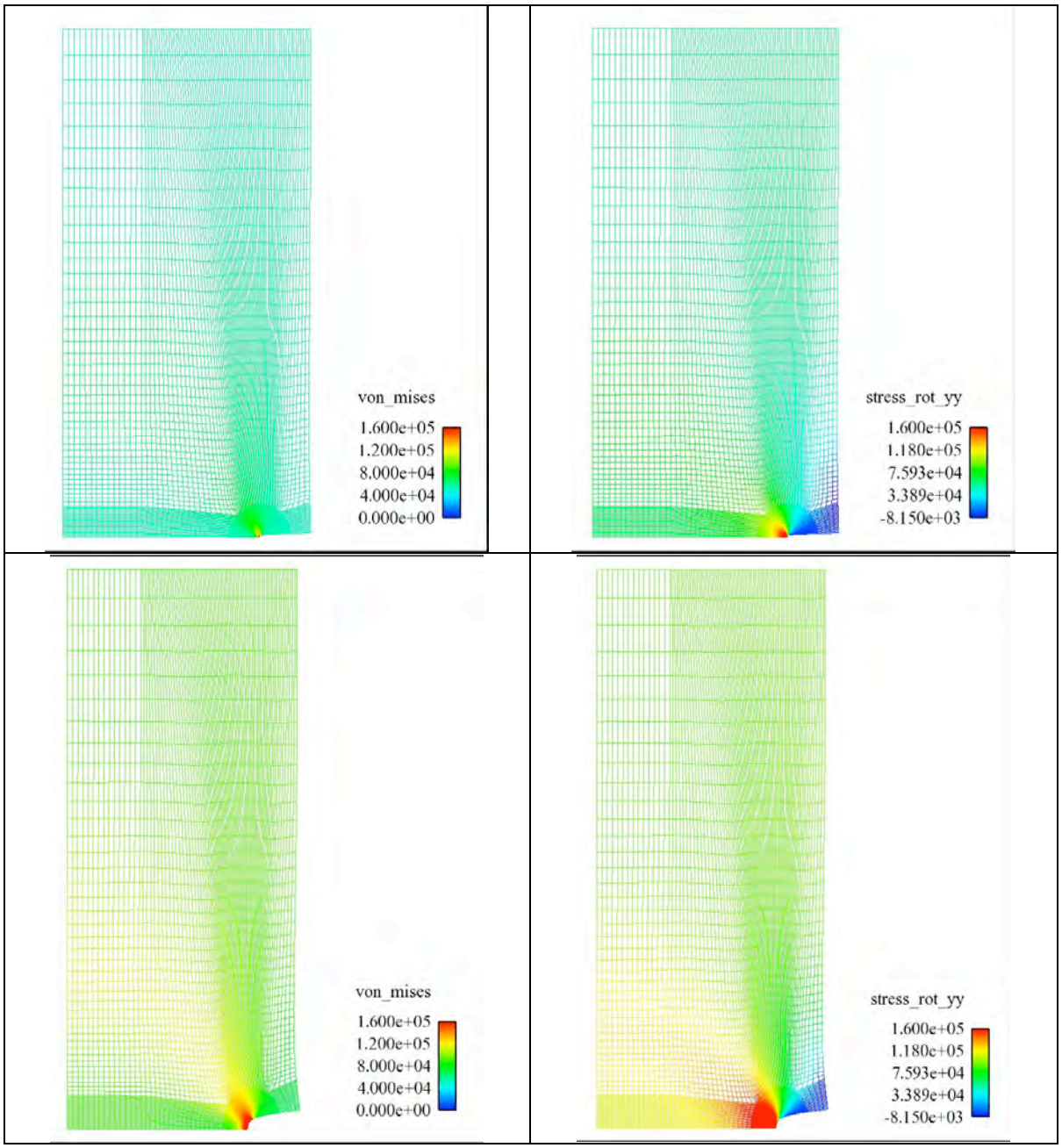


Figure 38 Von Mises effective stress and axial stress states of CRB specimen with a short initial crack, A6, at (a) displacement = 0.025 inches and (b) displacement = 0.150 inches. Regions in red meet or exceed the yield stress value of 1.6e5 psi.

The load versus displacement and percent error on displacement plots are shown in Figure 39 and Figure 40, respectively. As seen in these plots, the bulk displacement behavior of the short crack annealed specimen was well predicted using finite element models, both with the recommended collapsed elements and with the standard but not recommended hexes. Both element types predict the load-displacement behavior to within 4% of the testing. The first few points of all of the

error plots can be ignored, as small deviations will yield excessively large errors because the plots are nearly vertical before yielding occurs.

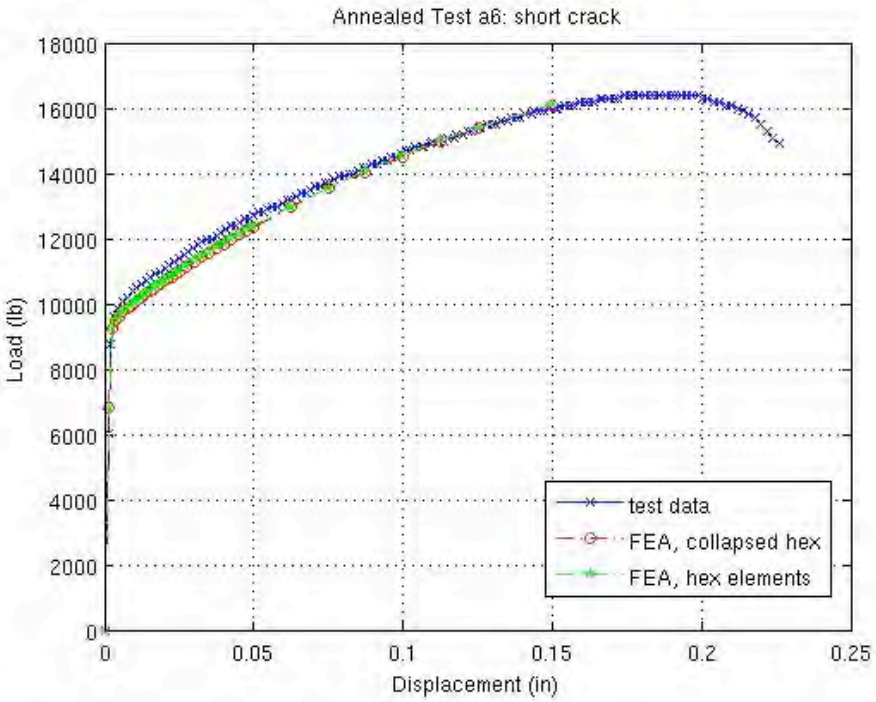


Figure 39 Load-displacement curve of uncharged CRB specimen with short crack, A6.

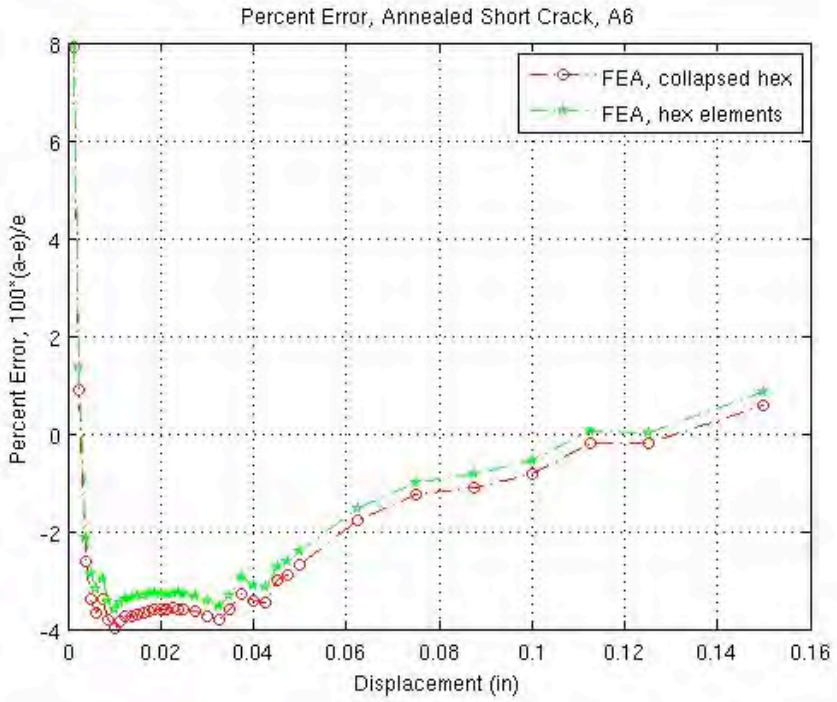


Figure 40 Error analysis of load versus displacement behavior of FEA results as compared with experimental data for uncharged CRB specimen with short crack, A6.

The J-integral values calculated by J3d are compared against the J-integral calculated from the test specimens, and the comparison is shown in Figure 41 and Figure 42. While the analysis slightly under-predicts the load at a given displacement, the J3d code significantly over-predicts the load at a given J-integral value. Alternatively stated, at a given load the J3d code under-predicts J as compared with the experimental estimate. Also of interest is that while both models fairly accurately predicted the load versus displacement behavior, the standard hex mesh yielded significantly more errant J-integral results than the mesh with the collapsed hexes at the crack tip. Neither mesh predicted J with any confidence for the uncharged short-crack specimen. A 25% error on J versus load is unacceptable given a 4% error on load versus displacement.

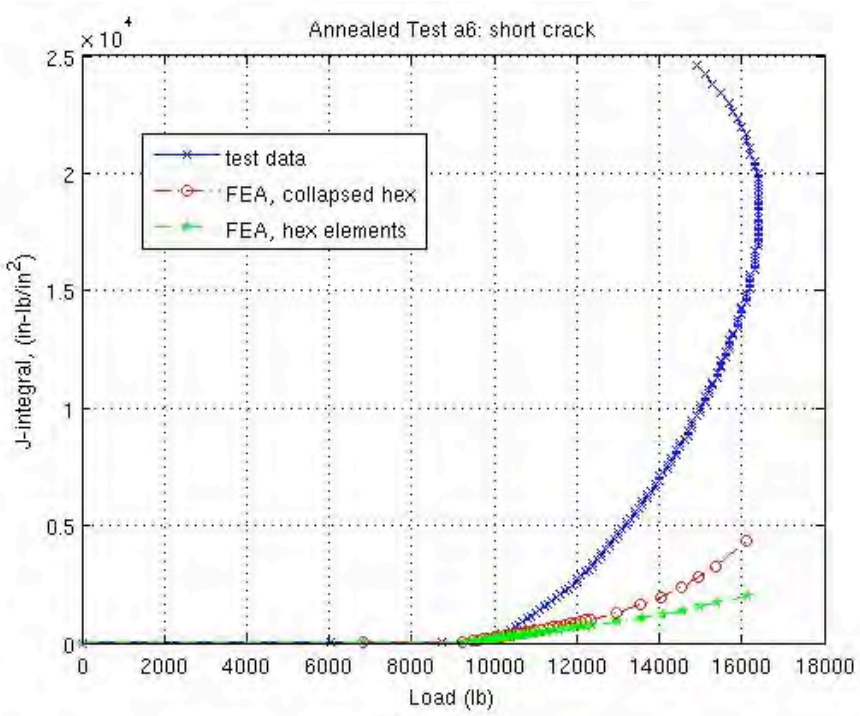


Figure 41 J versus load of uncharged CRB specimen with short crack, A6.

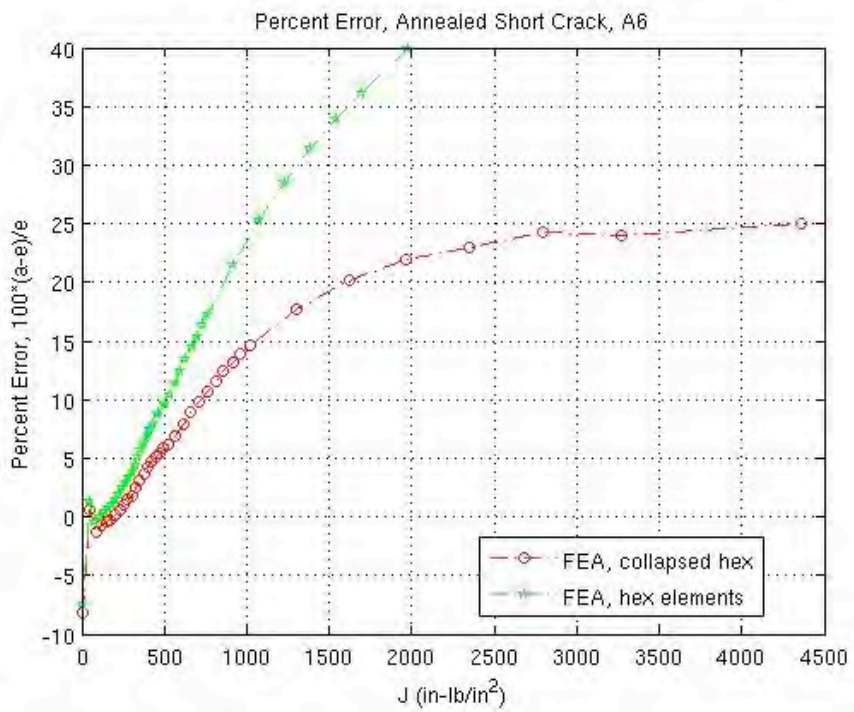


Figure 42 Error analysis of J versus load behavior of FEA results as compared with experimentally derived J values for uncharged CRB specimen with short crack, A6.

5.1.2 Uncharged Symmetric 3D Specimen: Medium Crack (A8)

The medium crack specimen, A8, had an a/r ratio of 0.508 and an initial specimen radius of 0.375 inches. Figure 43 shows the stress state just beyond yielding and at the peak load. The yield stress of the material is 1.6e5 psi.

(a)

(b)

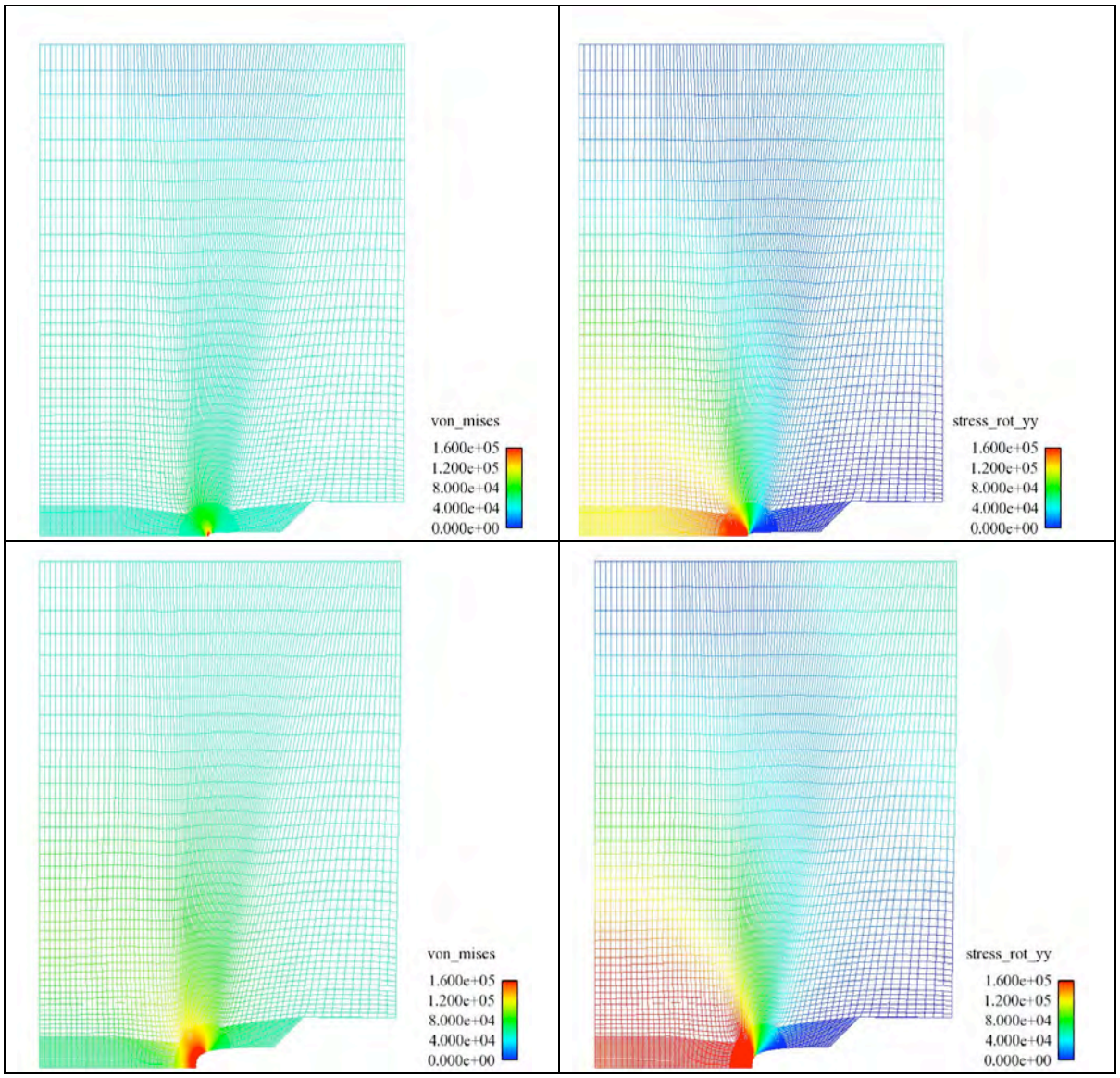


Figure 43 Von Mises effective stress and axial stress states of CRB specimen with a medium initial crack, A8, at (a) displacement = 0.010 inches and (b) displacement = 0.050 inches. Regions in red meet or exceed the yield stress value of 1.6e5 psi.

As seen in the load versus displacement and error plots in Figure 44 and Figure 45, respectively, the analysis predicts the specimen behavior within 11% of the test results. Like the short crack specimen, the analysis of the mesh with the standard hexes at the crack tip yielded marginally better predictions of load versus displacement than the collapsed element mesh.

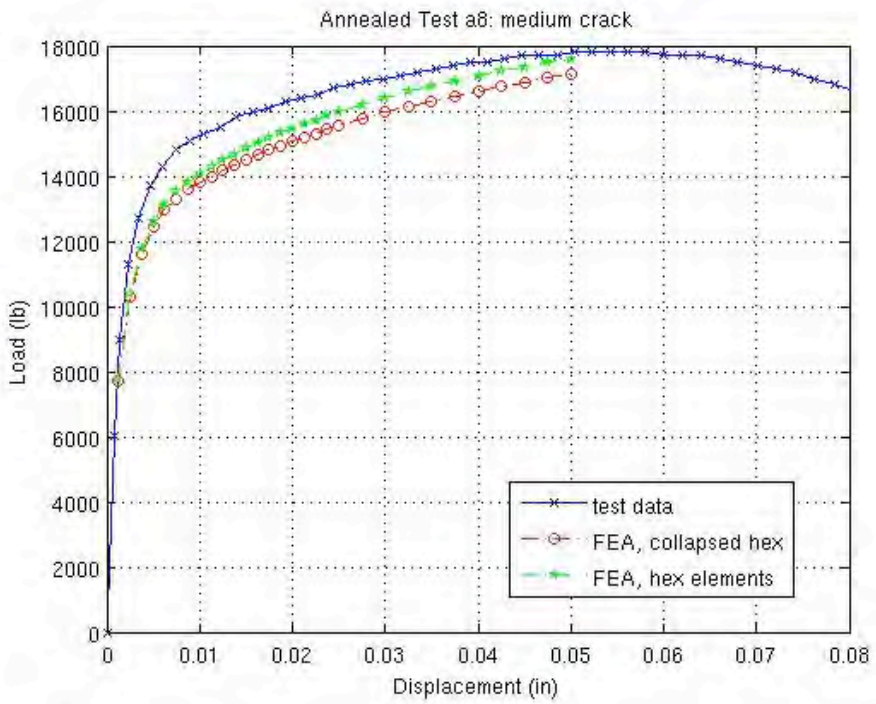


Figure 44 Load versus displacement of uncharged CRB specimen with medium crack, A8.

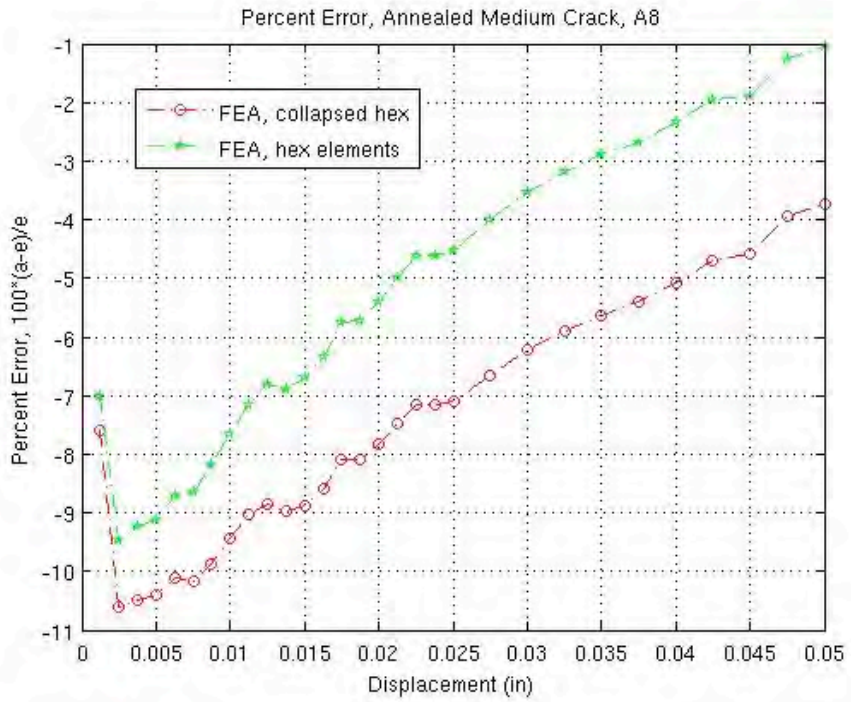


Figure 45 Error analysis of load versus displacement behavior of FEA results as compared with experimental data for uncharged CRB specimen with medium crack, A8.

The J versus load behavior, shown in Figure 46 and Figure 47, was significantly more accurate using the collapsed hexes than the standard hexes. Since the stress states from the analysis are input into the J3d code, it would be reasonable to assume that the errors in the load-displacement response would propagate through to the J-integral calculations. As seen in these plots, this assumption is supported, with the load is under-predicted at approximately 11% of the test value towards the beginning of the displacement process to under-predicting by approximately 4% as the applied load reaches the peak load. The errors in the J-load behavior follow the same trend are approximately the same values, supporting the theory that the errors in the J-load curve are caused by the input, and not the J3d code itself.

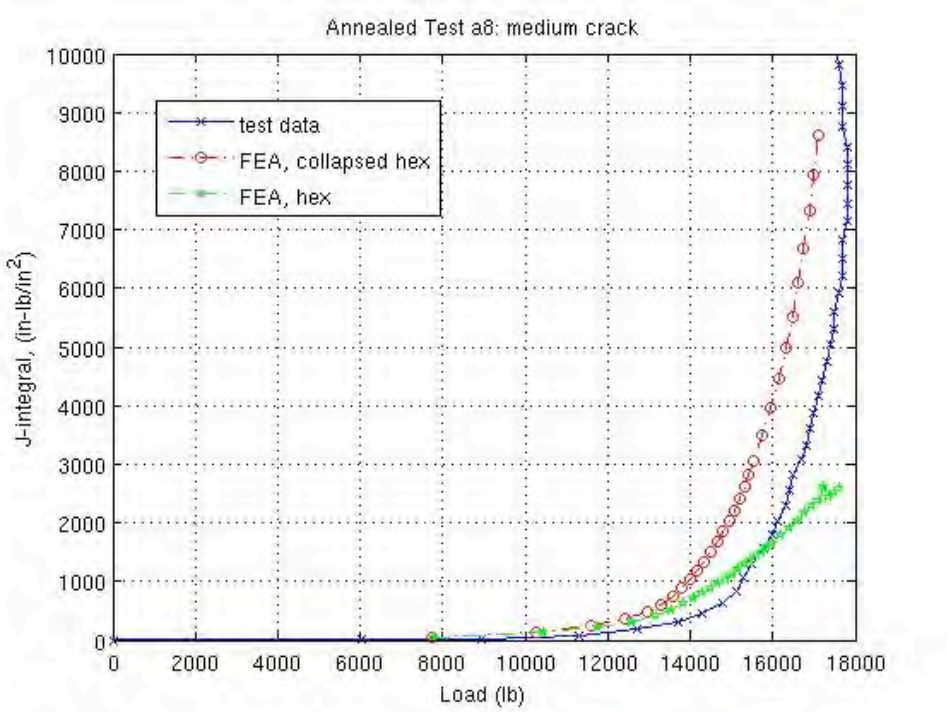


Figure 46 J versus load of uncharged CRB specimen with medium crack, A8.

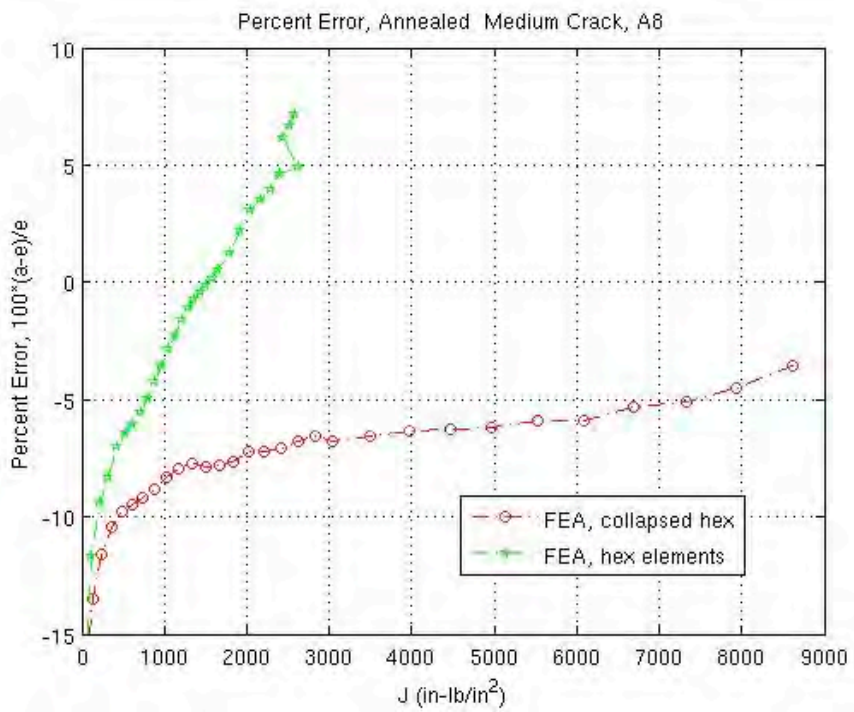


Figure 47 Error analysis of J versus load behavior of FEA results as compared with experimentally derived J values for uncharged CRB specimen with medium crack, A8.

5.1.3 Uncharged Symmetric 3D Specimen: Long Crack (A9)

The annealed long cracked specimen, A9, had an a/r ratio of 0.605 and an initial specimen radius of 0.375 inches. Figure 48 shows the stress state just beyond yielding and at the peak load. The yield stress of the material is 1.6e5 psi.

(a)

(b)

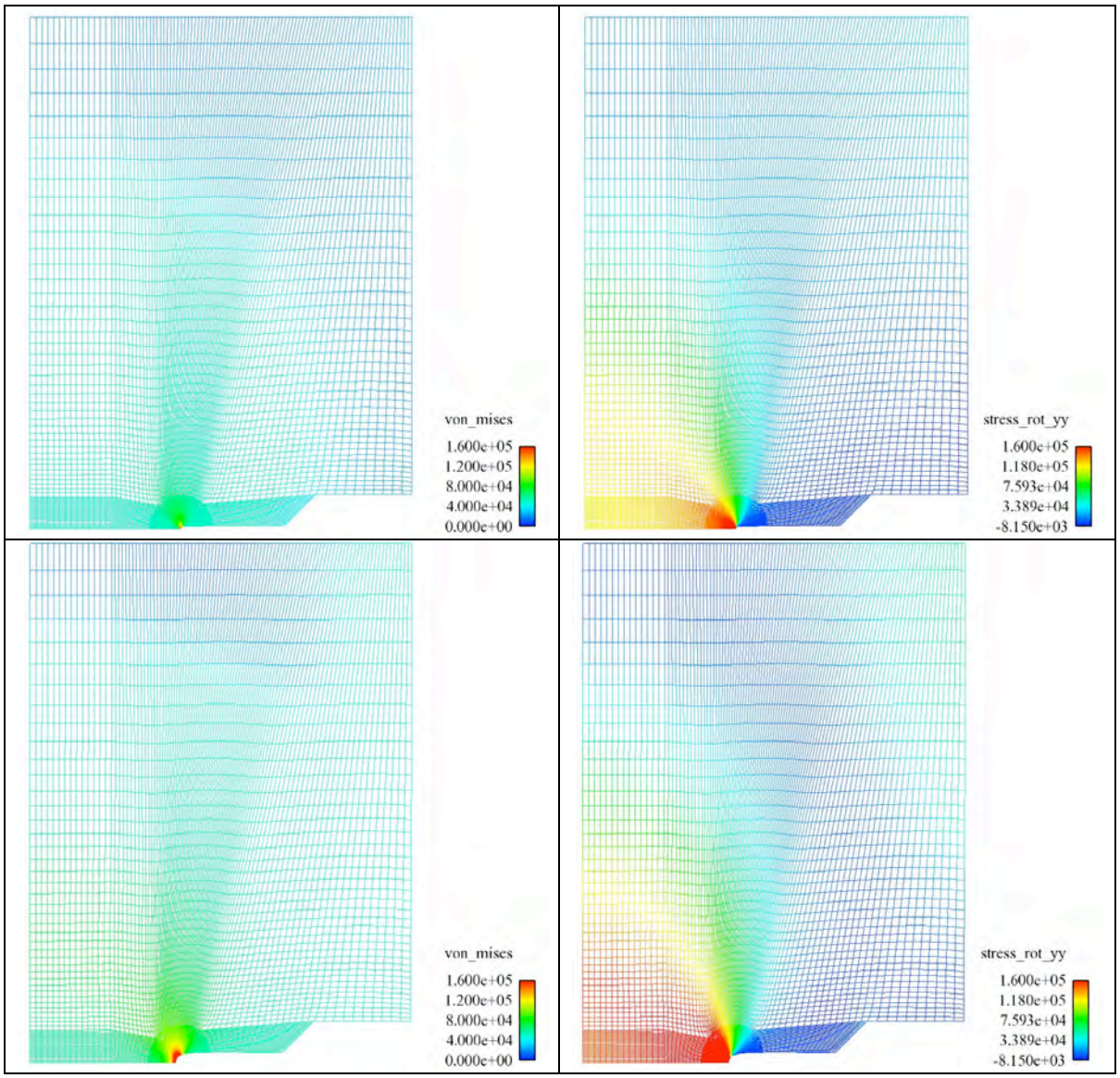


Figure 48 Von Mises effective stress and axial stress states of CRB specimen with a long initial crack, A9, at (a) displacement = 0.005 inches and (b) displacement = 0.020 inches. Regions in red meet or exceed the yield stress value of 1.6e5 psi.

Similar to the behavior seen in the short and medium cracked uncharged specimens, the standard hex mesh and the collapsed hex mesh predict the load versus displacement response equally well. Load versus displacement and error analysis is shown in Figure 49 and Figure 50. The long crack specimens results were more disappointing and had an error of almost 20% near yielding; although the behavior was slightly better towards the peak load, the errors were minimum 16%.

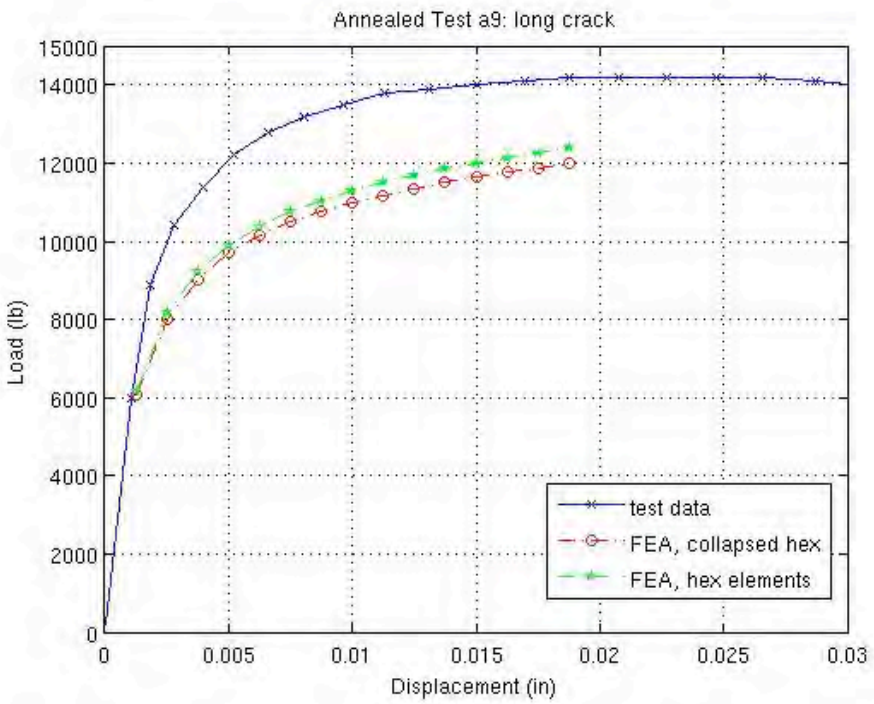


Figure 49 Load versus displacement of uncharged CRB specimen with long crack, A9.

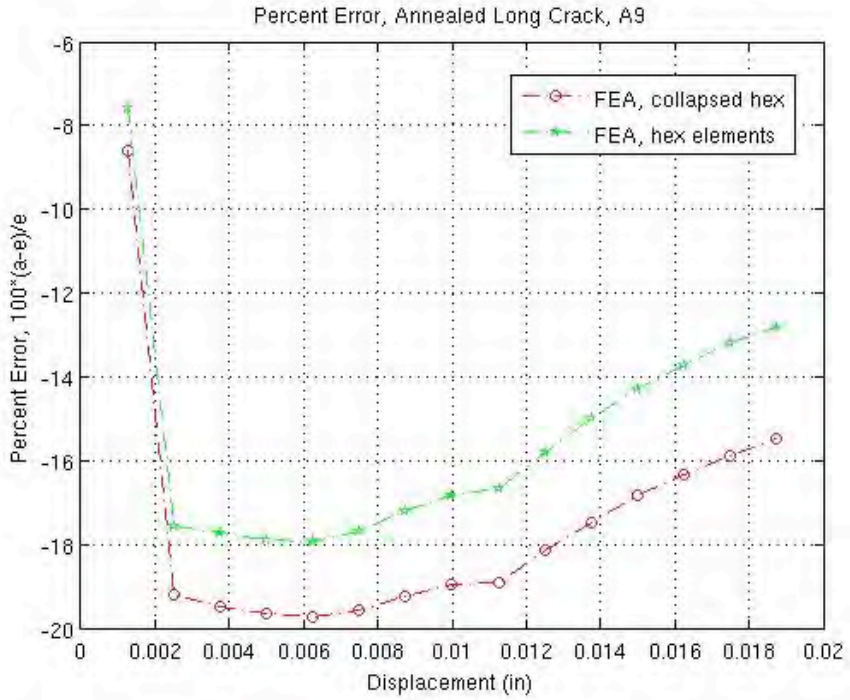


Figure 50 Error analysis of load versus displacement behavior of FEA results as compared with experimental data for uncharged CRB specimen with long crack, A9.

Discounting the results from the standard hex model, the J-integral values calculated from the stresses output from the analysis yield errors around 12%, as shown in Figure 51 and Figure 52. This is slightly lower than would be expected from the

errors calculated in the load-displacement plot; however, the values are within reason and follow the expected trend of under-predicting both the load-displacement and the J-load curves.

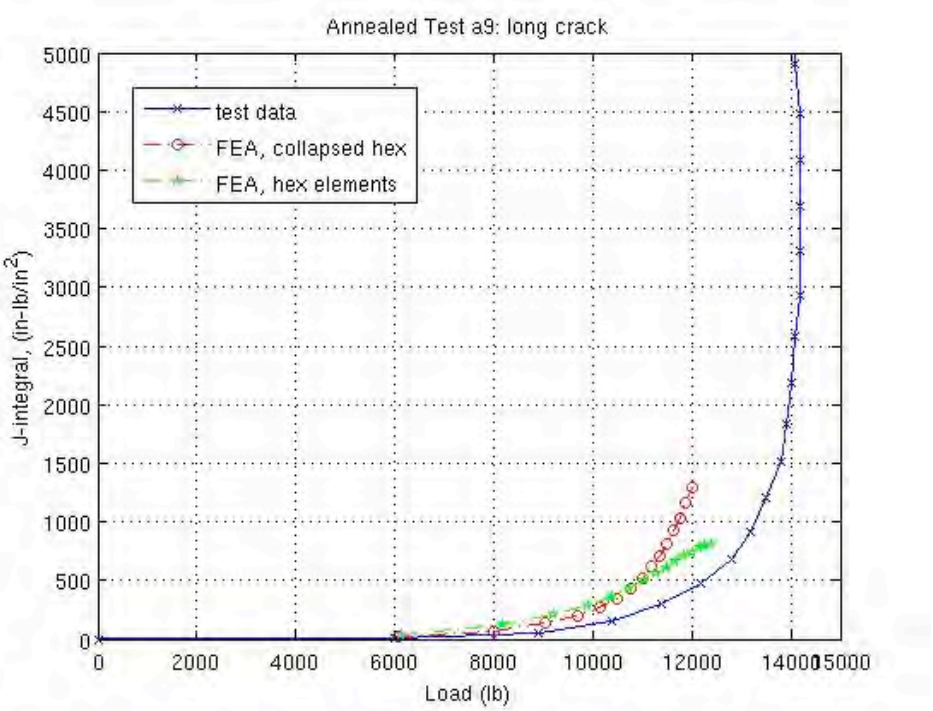


Figure 51 J versus load of uncharged CRB specimen with long crack, A9.

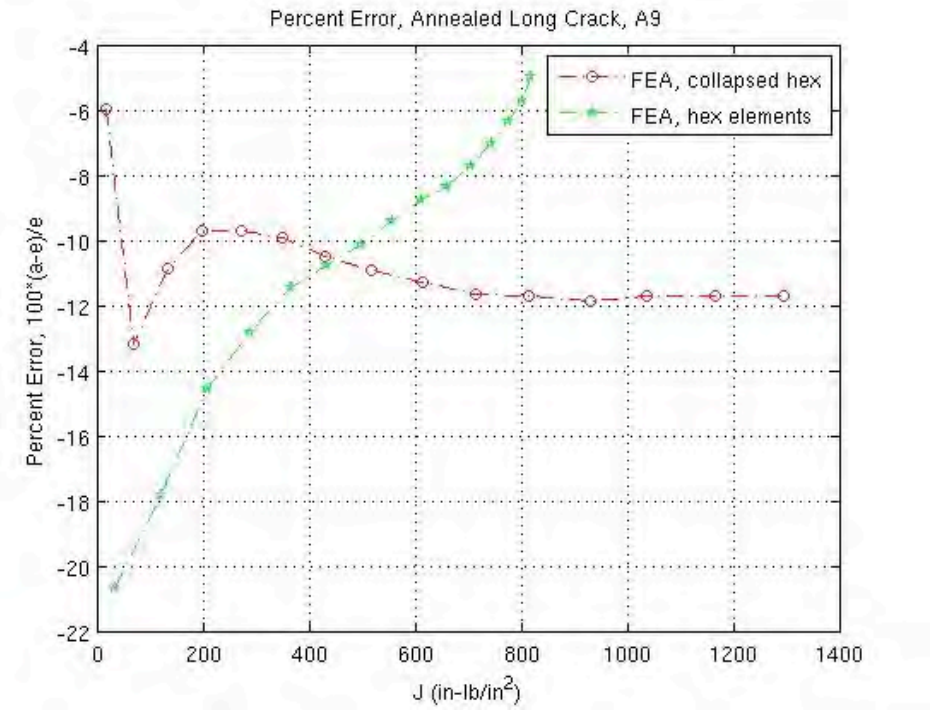


Figure 52 Error analysis of J versus load behavior of FEA results as compared with experimentally derived J values for uncharged CRB specimen with long crack, A9.

5.1.4 Hydrogen Charged Symmetric 3D Specimen: Short Crack (A4)

The annealed analysis sufficiently supported the recommendation by the J3d code to use collapsed elements at the crack tip instead of standard hex elements; thus, only results from collapsed element meshes will be shown henceforth.

The hydrogen charged short crack specimen, A4, had a pre-crack profile that was not concentric with the outer diameter of the specimen, yielding a non-symmetric test specimen. Figure 53 shows the cross section of A4 at the crack plane.

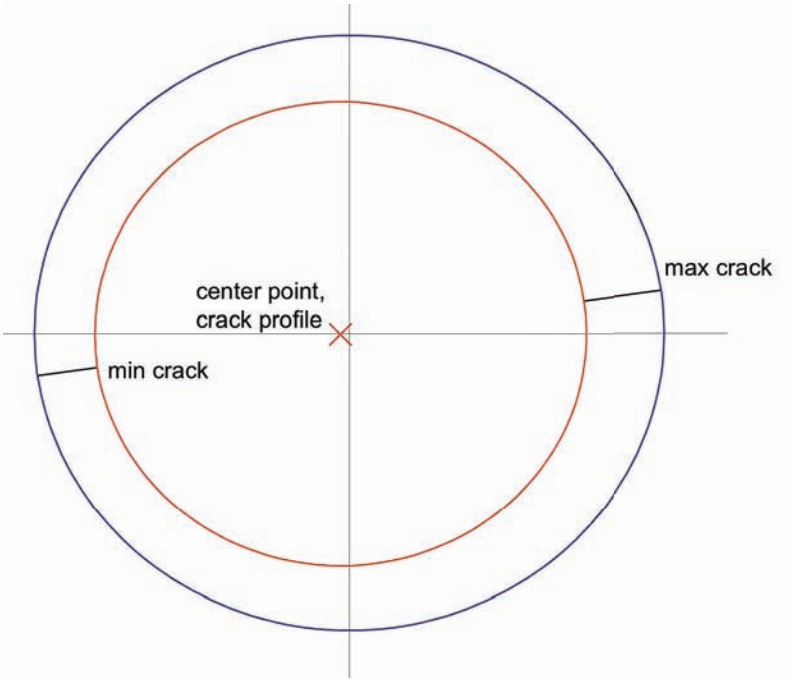


Figure 53 Cross-section through the crack plane of specimen A4. The outer circle represents the specimen profile, and the inner circle represents the initial crack. The red "x" represents the center of the pre-crack profile.

The maximum a/r, at approximately 8° from the horizontal axis in figure 53, was 0.25. The minimum a/r, 180° from the maximum, was 0.19. The average between the minimum and maximum was a/r = 0.22. To simplify modeling efforts, symmetric models similar to the other CRB models were created with the average, minimum and maximum crack ratios. Results from each of the models will be discussed.

A4 is the charged specimen roughly corresponding to the annealed short cracked specimen, A6. Figure 54 shows the stress state just beyond yielding and at the peak load for the model with the average crack ratio. The minimum and maximum cracks ratio models had similar stress states at these displacements. The yield stress of the material is 1.6e5 psi.

(a)

(b)

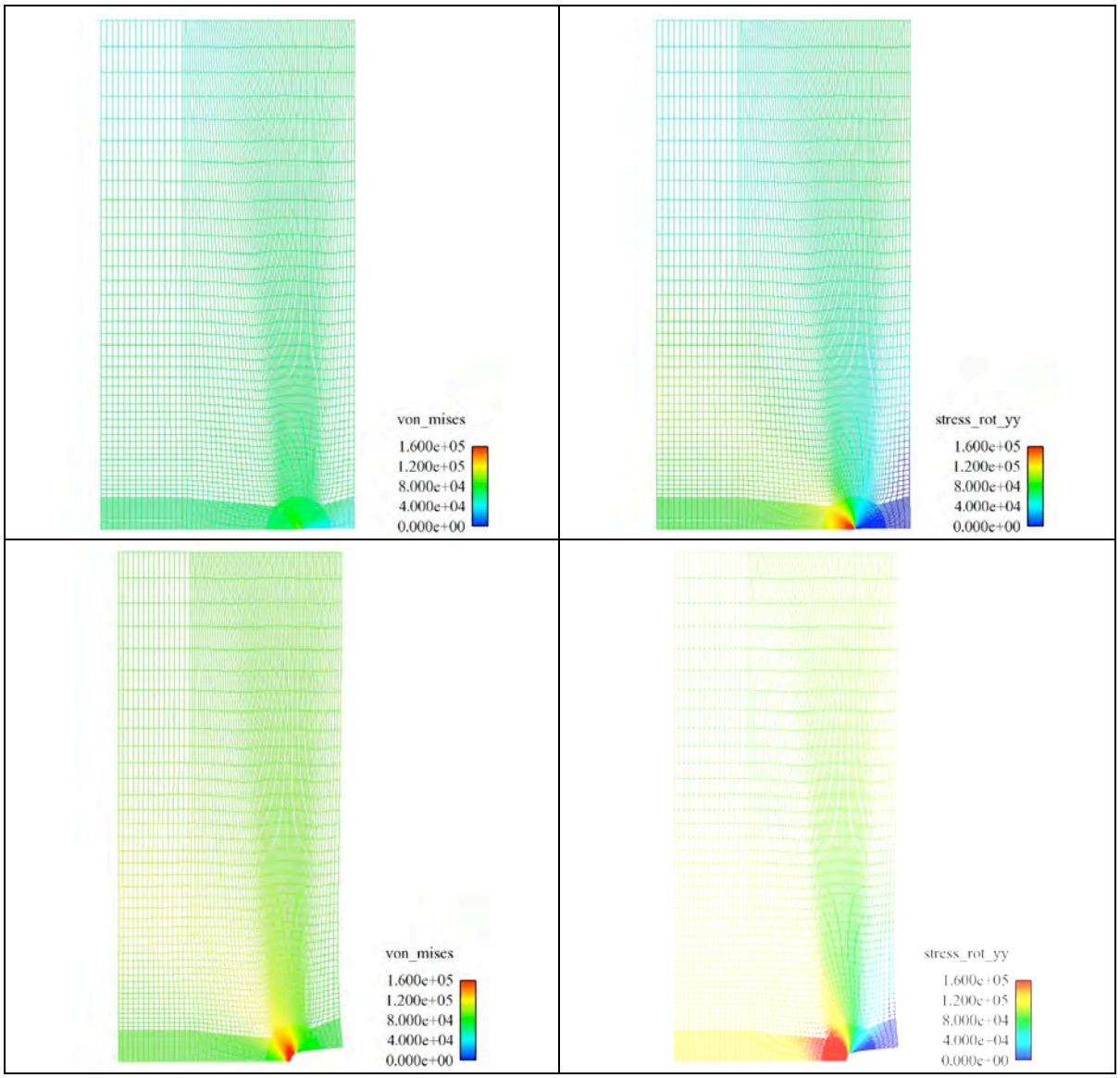


Figure 54 Von Mises effective stress and axial stress states of a hydrogen charged CRB specimen with a short initial crack, A4, at (a) displacement = 0.005 inches and (b) displacement = 0.1 inches. Regions in red meet or exceed the yield stress value of 1.6e5 ps

Load versus displacement behavior calculated from the analyses is shown in Figure 55 and Figure 56. Finite element analysis of the hydrogen charged CRB with a short crack predicted load versus displacement behavior within +/- 4% of the experimental values in the regions of interest.

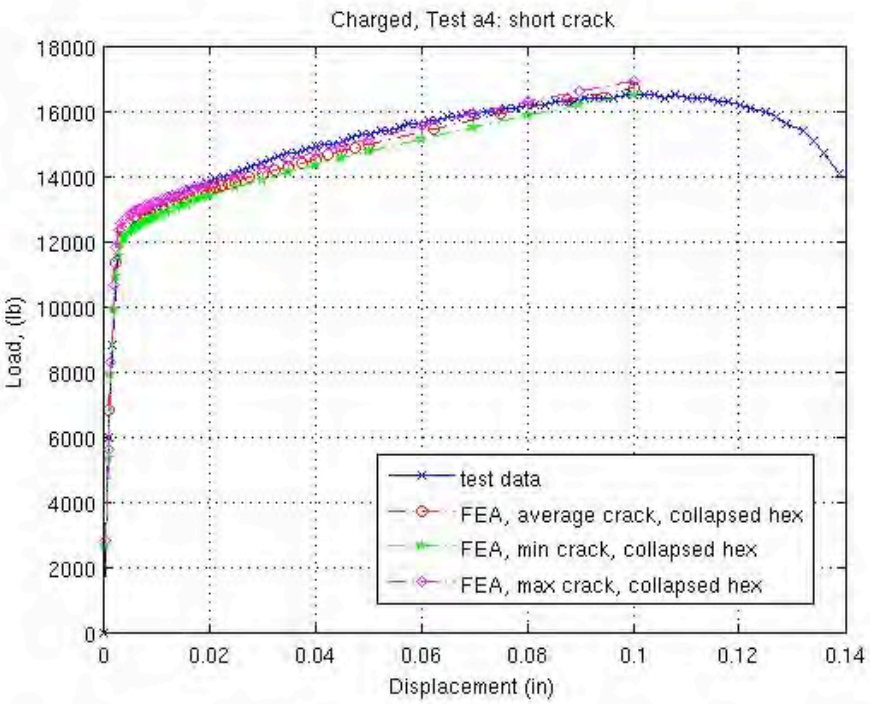


Figure 55 Load versus displacement of charged CRB specimen with short crack, A4. Results from models with the average, maximum and minimum crack lengths are shown. Specimen A4 can be compared with the uncharged specimen with short crack, A6.

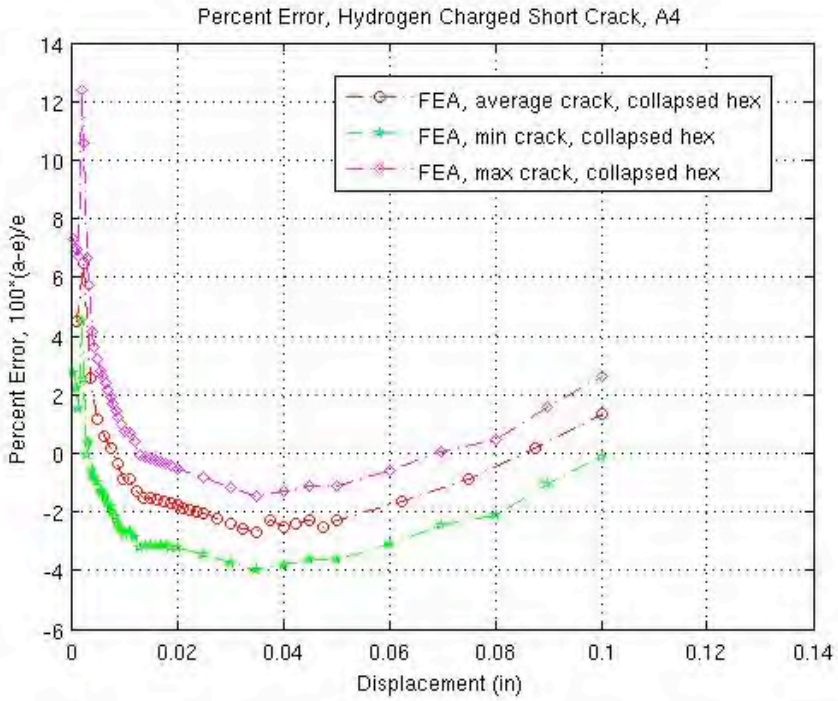


Figure 56 Error analysis of load versus displacement behavior of FEA results as compared with experimental data for charged CRB specimen with short crack, A4.

While the load versus displacement behavior is fairly well predicted in the case of the hydrogen charged short crack CRB, the J versus load predictions yield much higher errors than expected, as seen in Figure 57 and Figure 58. Not only are the error values higher than expected, but the trend of slightly under-predicting the load for smaller displacements and over-predicting the load at larger displacements is not reflected in the J-integral calculations.

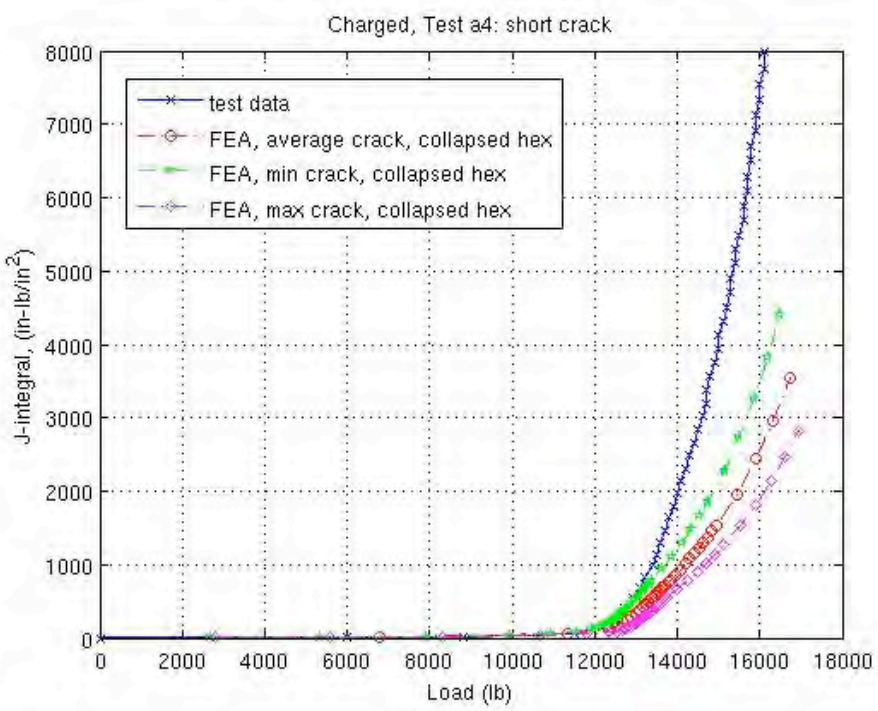


Figure 57 J versus load of hydrogen charged CRB specimen with short crack, A4.

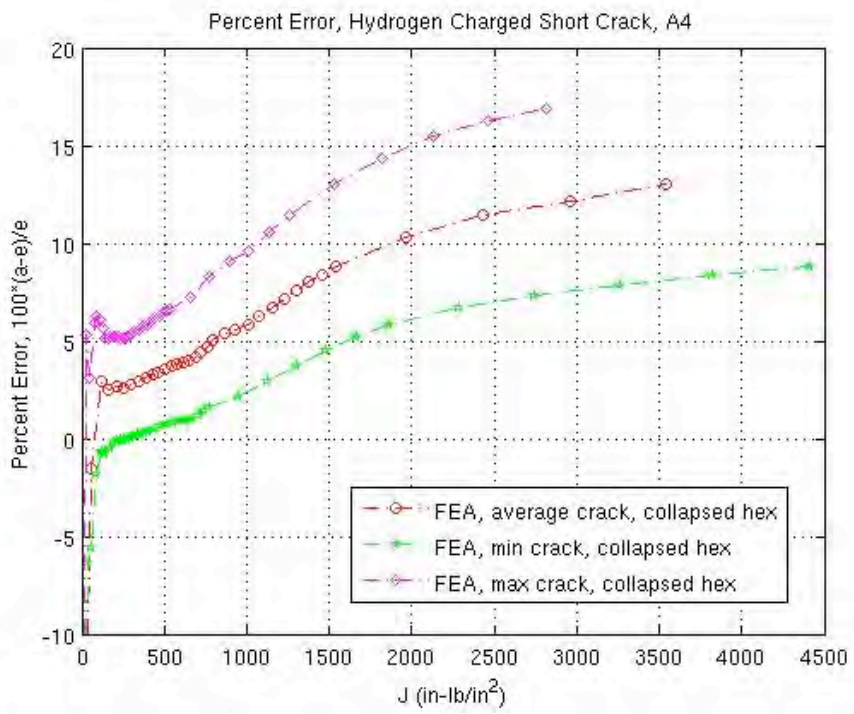


Figure 58 Error analysis of J versus load behavior of FEA results as compared with experimentally derived J values for charged CRB specimen with short crack, A4.

While the hydrogen charged short crack specimen yielded slightly more accurate J-integral values than the annealed short crack specimen, finite element analysis and the J3d code did not predict J-integral values with confidence for either specimen.

5.1.5 Hydrogen Charged Symmetric 3D Specimen: Medium Crack (A7)

The hydrogen charged medium crack specimen, A7, had an a/r ratio of 0.485 and an initial specimen radius of 0.375. A7 is the charged specimen roughly corresponding to the annealed medium cracked specimen, A8. Figure 59 shows the stress state just beyond yielding and at the peak load. The yield stress of the material is 1.6e5 psi.

(a)

(b)

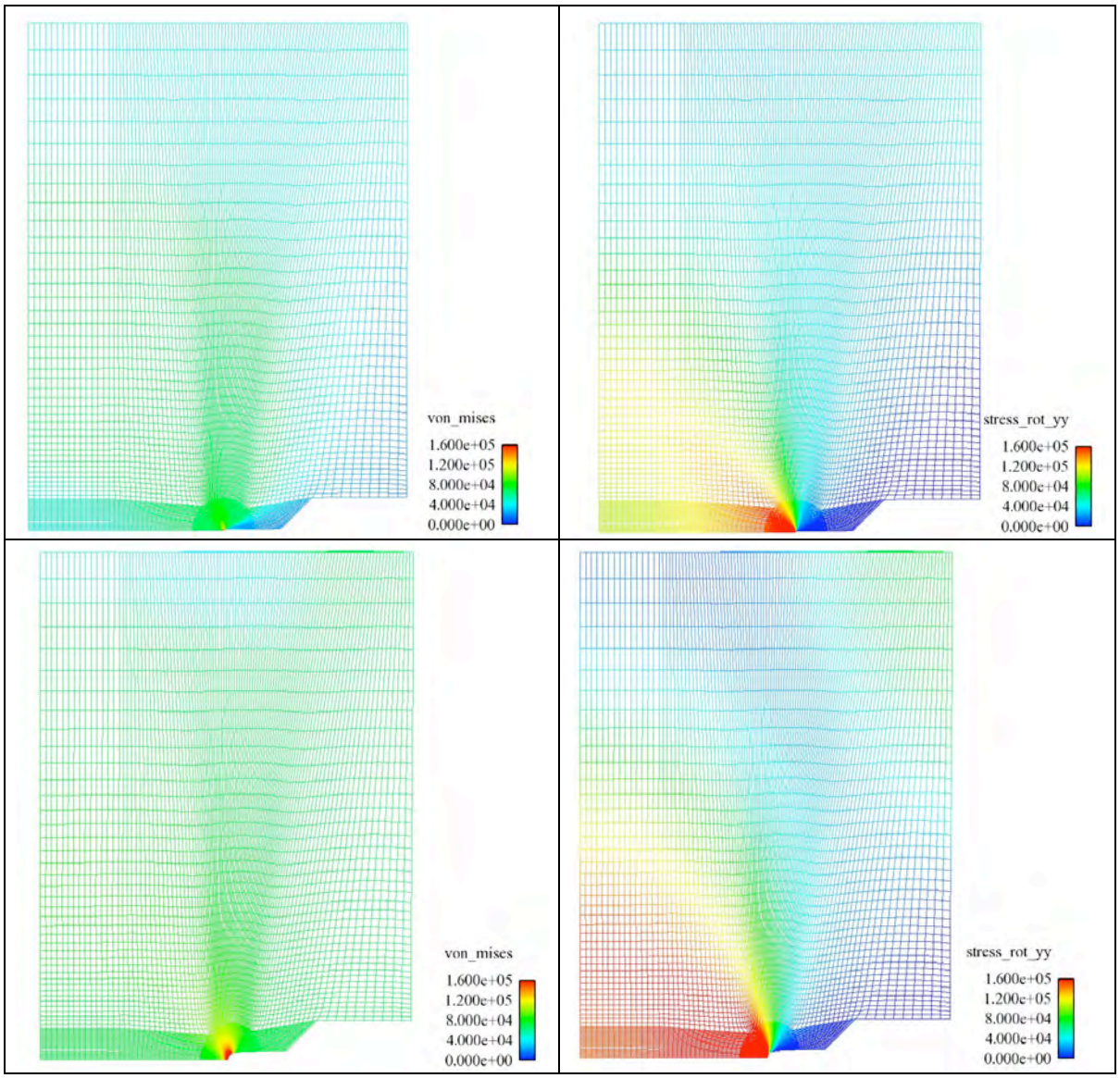


Figure 59 Von Mises effective stress and axial stress states of a hydrogen charged CRB specimen with a medium initial crack, A7, at (a) displacement = 0.004 inches and (b) displacement = 0.020 inches. Regions in red meet or exceed the yield stress value of 1.6e5

Load versus displacement behavior calculated from the analysis is shown in Figure 60 and Figure 61. Finite element analysis of the hydrogen charged CRB with a medium crack predicted load versus displacement behavior fairly well, with a maximum of 9% error at the peak load.

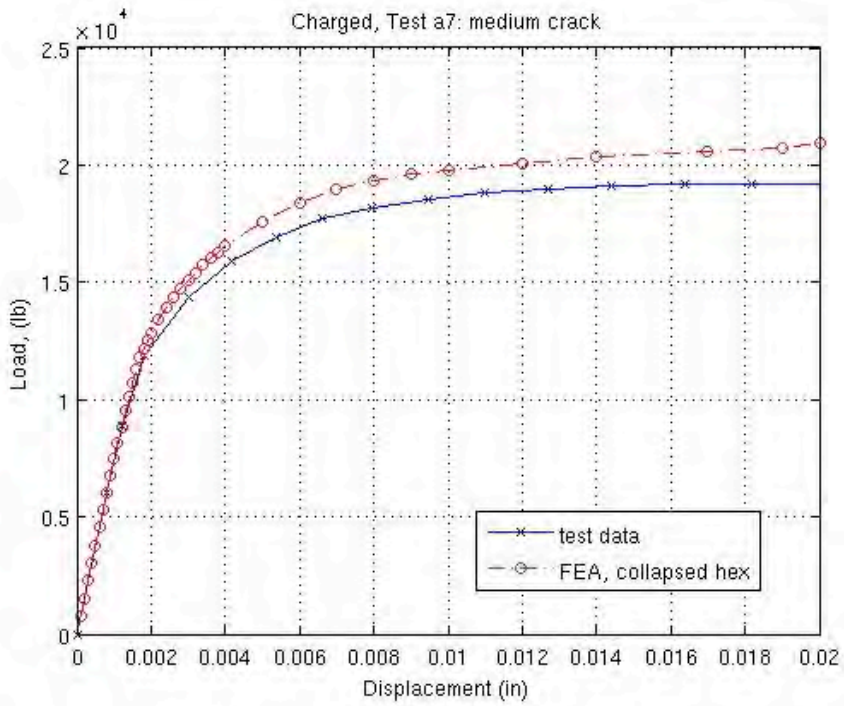


Figure 60 Load versus displacement of charged CRB specimen with medium crack, A7. Specimen A7 can be compared with the uncharged specimen with medium crack, A8.

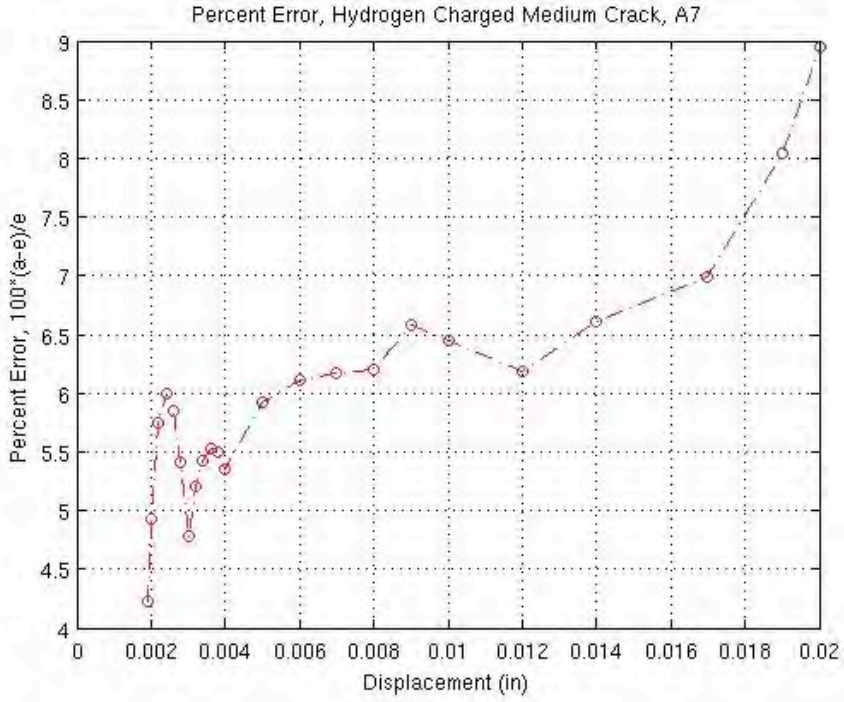


Figure 61 Error analysis of load versus displacement behavior of FEA results as compared with experimental data for charged CRB specimen with medium crack, A7.

Errors in the load versus displacement behavior seem to be reflected in J-integral calculations, with errors around 7% for much of the curve and a maximum of 9%, as seen in Figure 62 and Figure 63. These errors are in line with the errors calculated in the load-displacement curves.

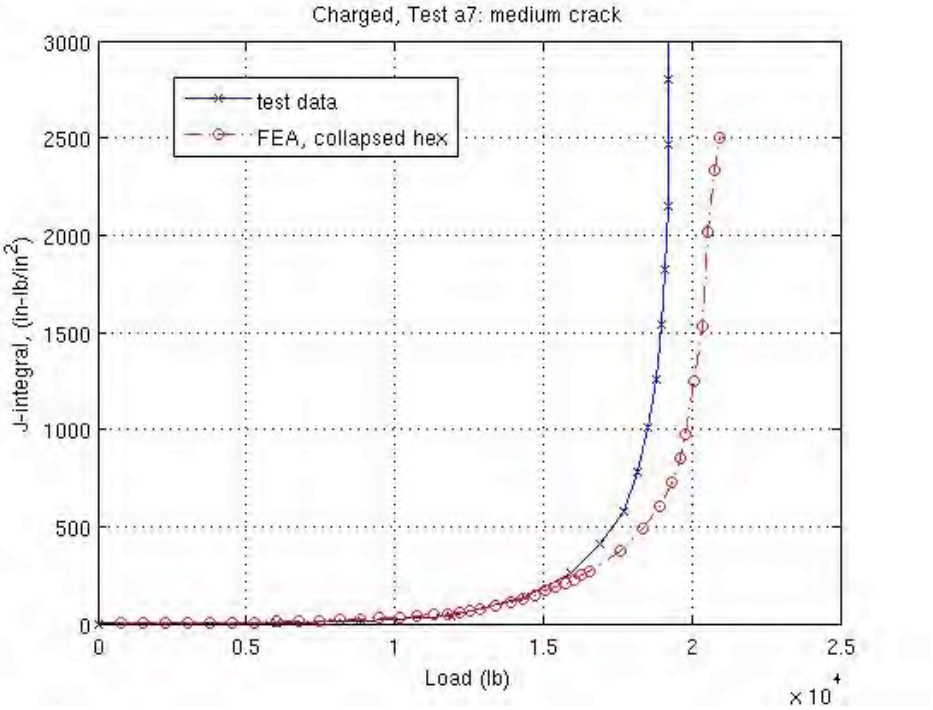


Figure 62 J versus load of hydrogen charged CRB specimen with medium crack, A7.

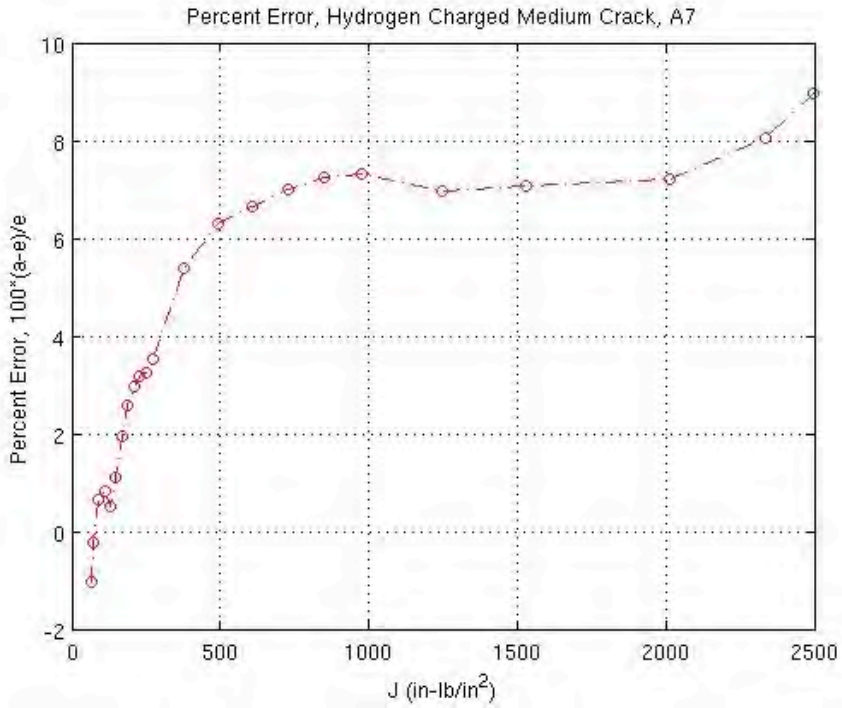


Figure 63 Error analysis of J versus load behavior of FEA results as compared with experimentally derived J values for charged CRB specimen with medium crack, A7.

5.1.6 Hydrogen Charged Symmetric 3D Specimen: Long Crack (A1)

The hydrogen charged medium crack specimen, A1, had an a/r ratio of 0.600 and an initial specimen radius of 0.375. A1 is the charged specimen roughly corresponding to the annealed medium cracked specimen, A9. Figure 64 shows the stress state just beyond yielding and at the peak load. The yield stress of the material is 1.6e5 psi.

(a)

(b)

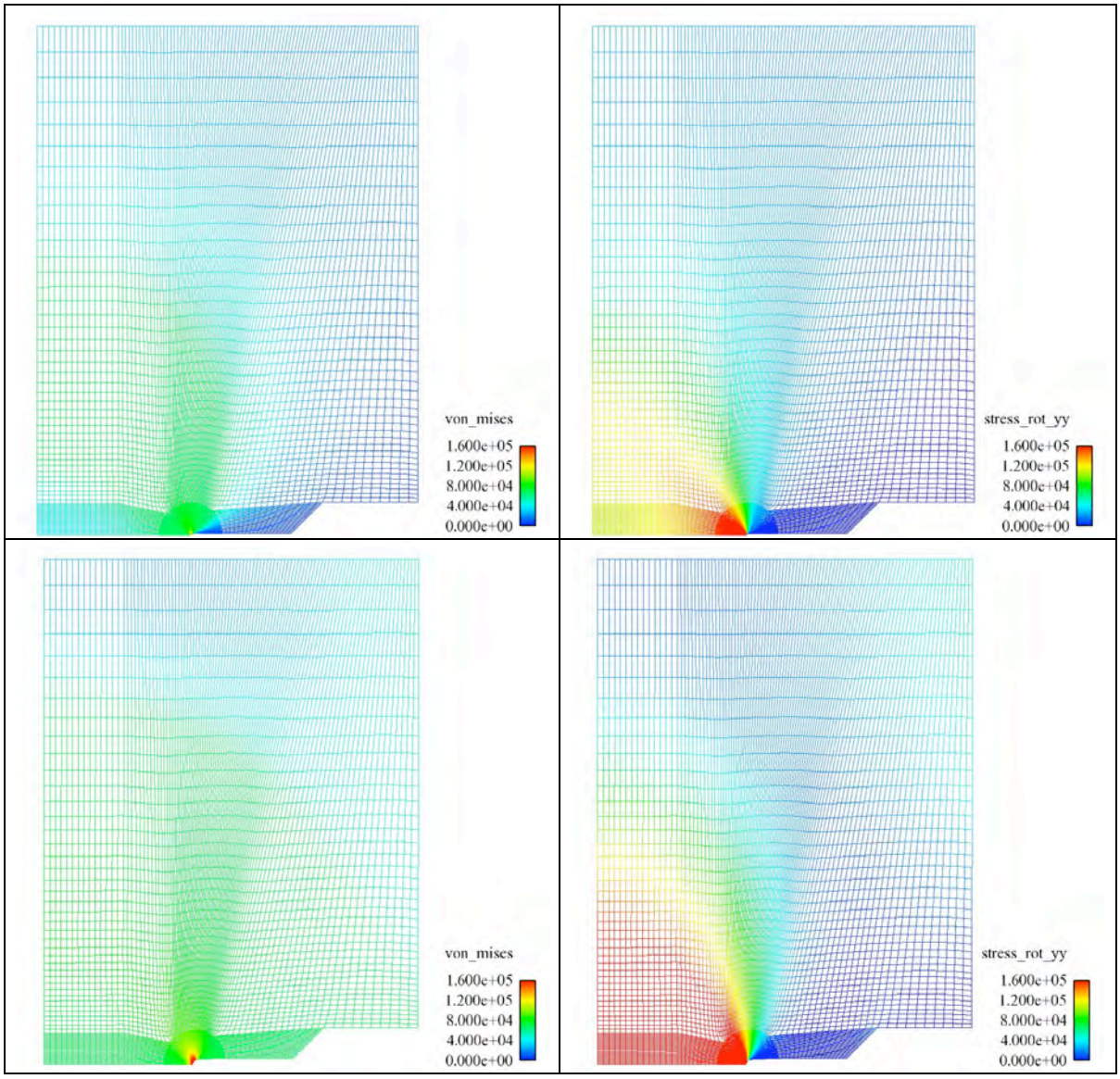


Figure 64 Von Mises effective stress and axial stress states of a hydrogen charged CRB specimen with a long initial crack, A1, at (a) displacement = 0.003 inches and (b) displacement = 0.012 inches. Regions in red meet or exceed the yield stress value of 1.6e5 psi.

Load versus displacement behavior calculated from the analysis is shown in the Figure 65 and Figure 66. Finite element analysis predicts the load versus displacement relationship within 2% of the experimental results.

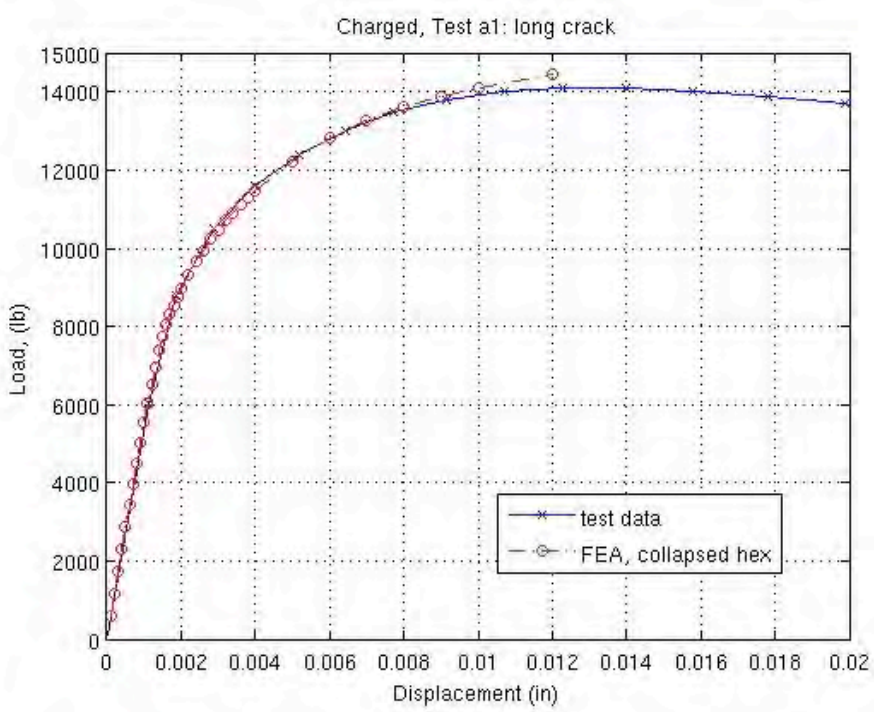


Figure 65 Load versus displacement of hydrogen charged CRB specimen with long crack, A1.

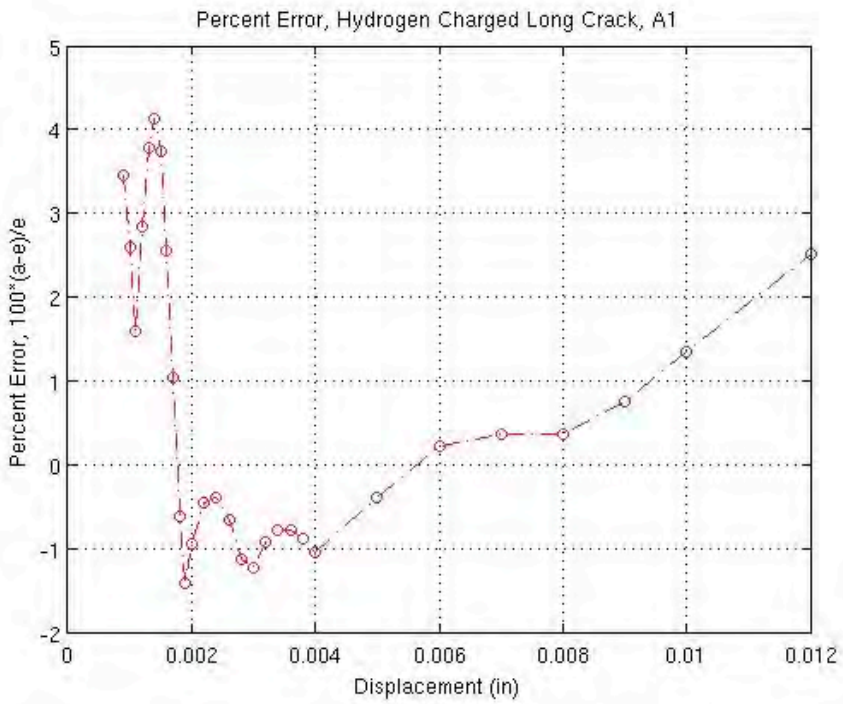


Figure 66 Error analysis of load versus displacement behavior of FEA results as compared with experimental data for charged CRB specimen with long crack, A1.

The errors calculated for the J-load curves may seem large compared with the errors from the load-displacement curve; however, if we ignore the errors at low J values, where a small deviation from the curve will yield a very large error due to the near horizontal nature of the curve, the expected trend of corresponding errors between load-displacement and J-load continues. The analysis under-predicts the both curves until a load of 12,500 lb. is reached, and then proceeds to slightly over-predict both curves above this load value. The error at the ultimate load for both curves is 3%. Plots are shown in Figure 67 and Figure 68.

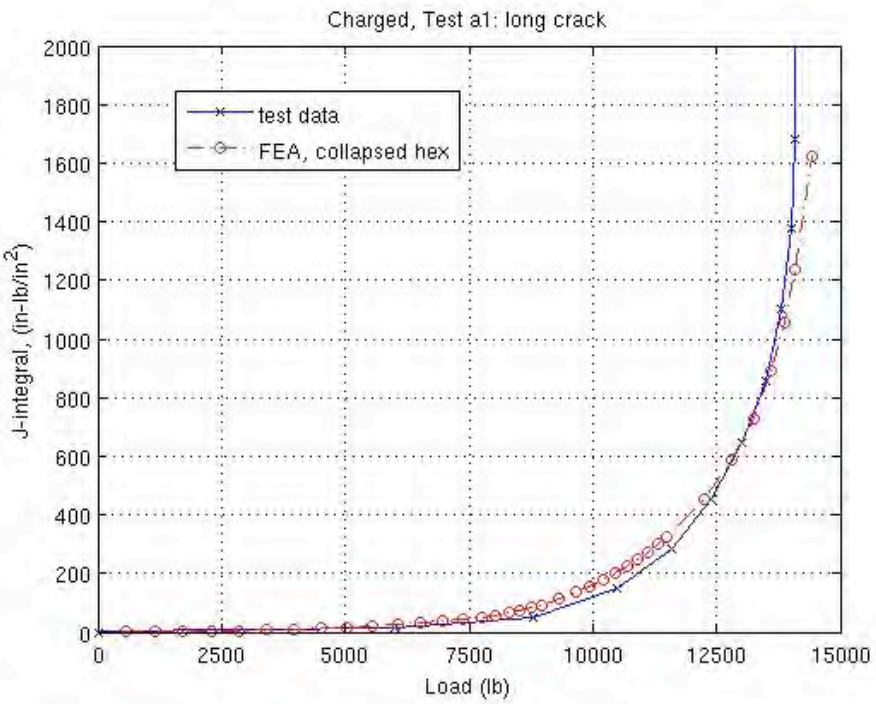


Figure 67 J versus load of hydrogen charged CRB specimen with long crack, A1.

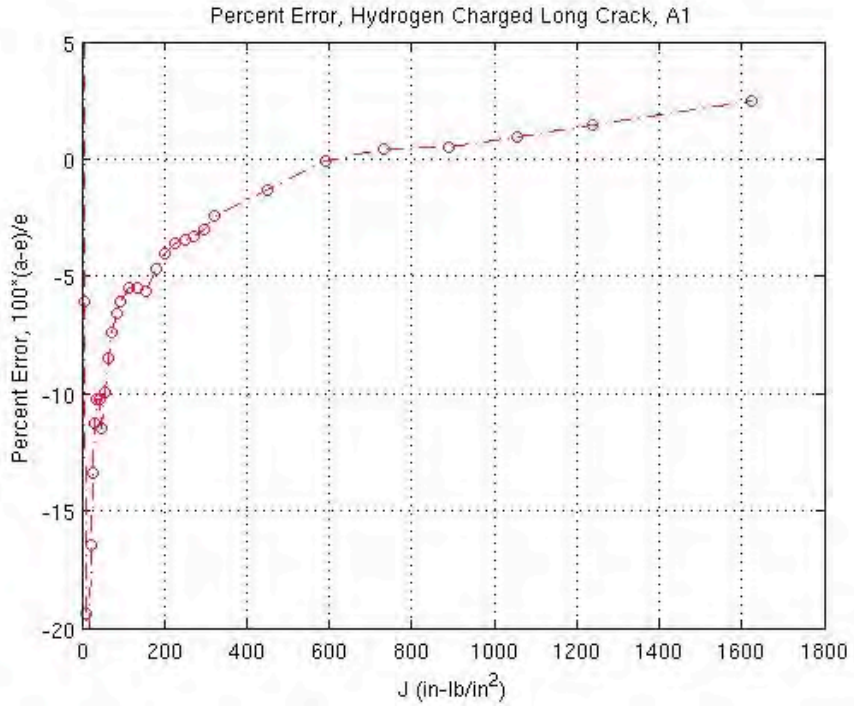
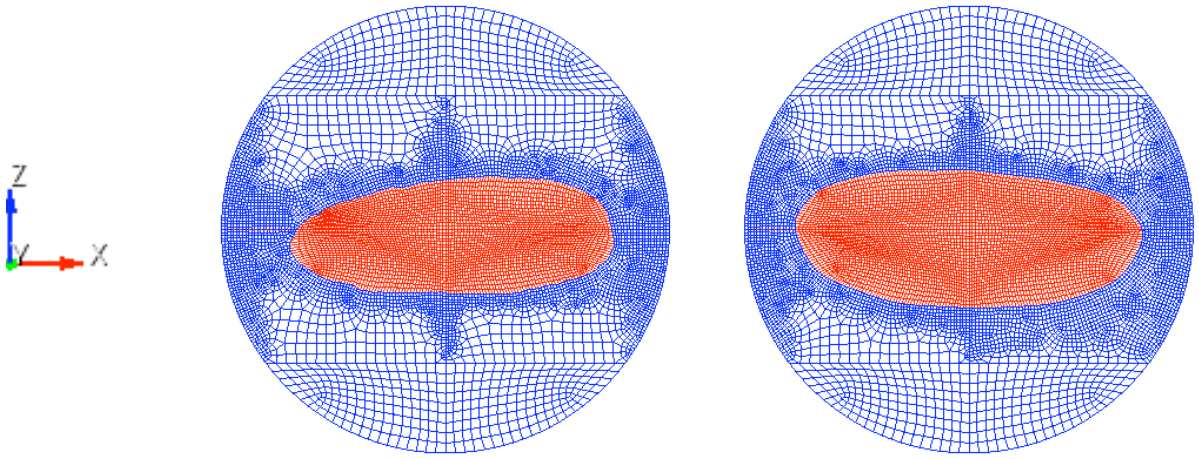


Figure 68 Error analysis of J versus load behavior of FEA results as compared with experimentally derived J values for hydrogen charged CRB specimen with long crack, A1.

5.2 Asymmetric, Three-Dimensional Specimens

The asymmetric specimens are cylindrical and have a notch cut from either one or both sides of the specimen. The specimens are then pre-cracked, yielding an asymmetric initial crack orientation. Of the several tests run, one annealed and one hydrogen charged specimen were chosen to model and compare. Both of these specimens had two side notches. Half symmetry across the crack plane was utilized, but because the pre-crack orientations were not perfectly symmetric across any other planes, a full 360-degree section was modeled.

Post-test measurements of the pre-crack orientations were supplied by the experimental group[21]. These points were fit to a curve using Matlab, and the pre-cracked area was scaled to represent the initial pre-crack dimensions based on the ratio between the original specimen radius and final specimen radius. These simplified points were used to create the finite element mesh in Cubit. Figure 69 and Figure 70 show the mesh of asymmetric specimens B1 and B2. Regions in blue represent the initial pre-cracked geometry.



(a) (b)

Figure 69 Cross-section of finite element mesh of asymmetric cylindrical specimens. (a) Specimen B1 is uncharged, and (b) specimen B2 is hydrogen charged. Regions in blue represent the pre-cracked area.

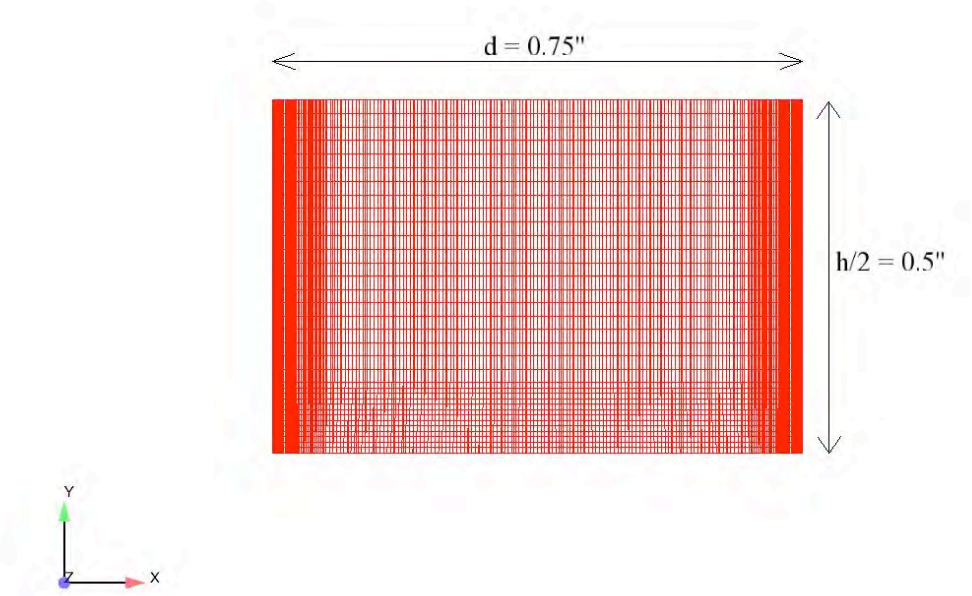


Figure 70 Side view of finite element mesh of asymmetric cylindrical specimens, B1 and B2. Symmetry across the crack plane is used for the finite element model.

For comparison purposes, the approximate crack ratios were calculated for the asymmetric specimens in each major axis. The annealed asymmetric specimen, B1, had a/r ratios of 0.30 and 0.75 in the two major directions of the oval shaped pre-crack, or the x- and z-directions, respectively, using the analysis coordinate axis in Figure 69. The hydrogen charged asymmetric specimen, B2, had a/r ratios of 0.23 and 0.69 in the corresponding axes.

Load versus displacement plots for B1 are shown below in Figure 71 and Figure 72. For the annealed case, the analysis prediction matches test results fairly well below yield. The analysis begins to over-predict the curve post-yield and is within 10% of the test results at the peak load. Results from the asymmetric charged specimen, B2, are shown below in Figure 73 and Figure 74. The analysis predicts both yield and post-yielding behavior within a few percent error.

Direct comparison of the J-Integral itself was not possible for these specimens as a method and analytical expression for estimating J from experimental data are not established for such asymmetric, three dimensional geometries. Potentially, the J3d code could be used to quantify J for numerous points along the crack fronts shown in Figure 69. Then, validation could be accomplished by comparing the values of J corresponding to values of load at which fracture occurs with any known values for fracture toughness J_c of the two materials (uncharged and Hydrogen charged, respectively). Since accurate fracture toughness measurements were not an objective of this project, we defer this comparison for future work.

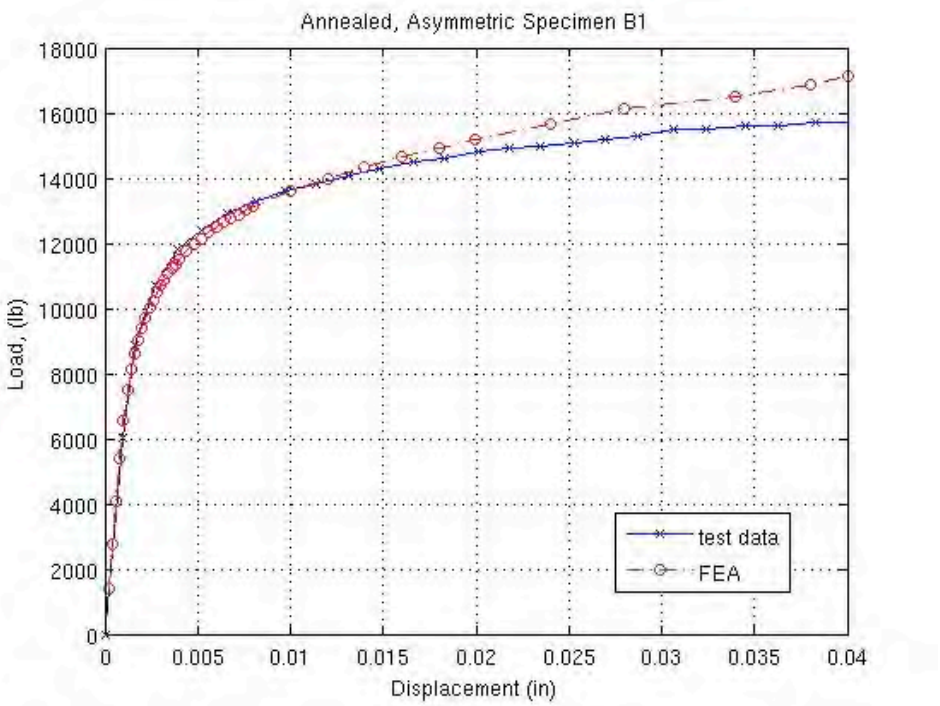


Figure 71 Load versus displacement of uncharged asymmetric CRB specimen.

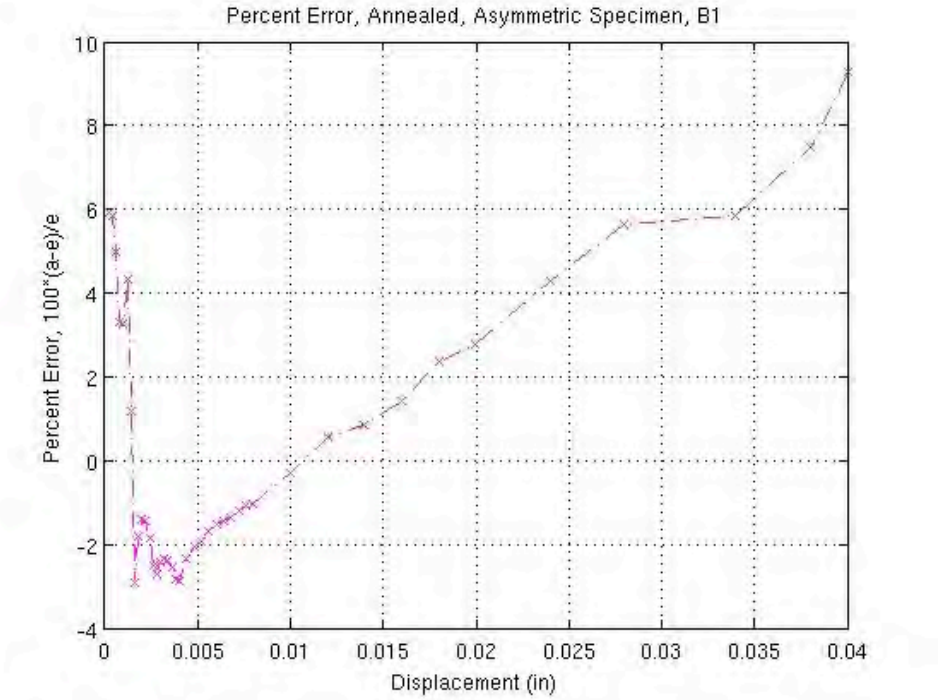


Figure 72 Error analysis of load versus displacement behavior of FEA results as compared with experimental data for uncharged asymmetric CRB specimen.

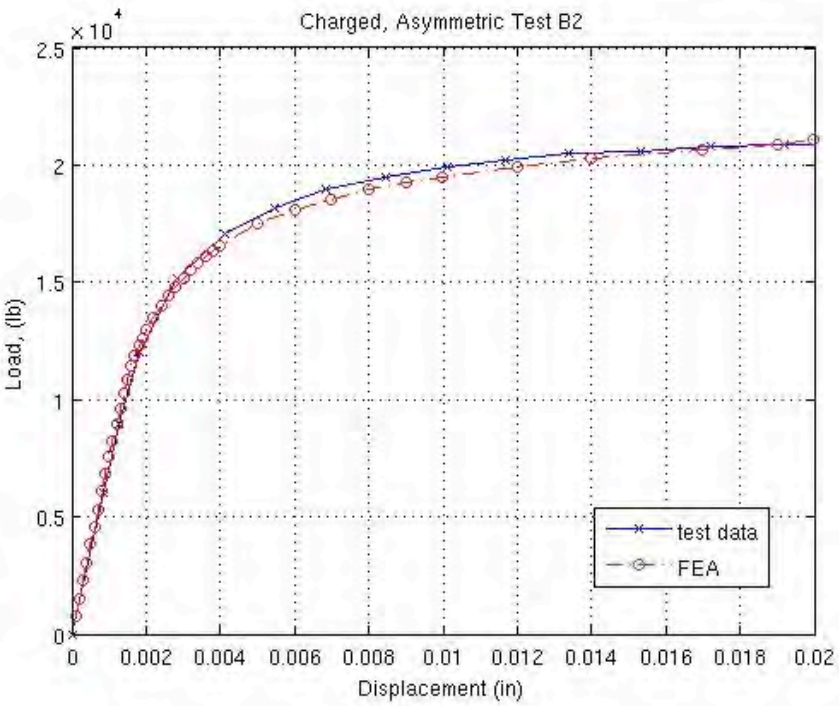


Figure 73 Load versus displacement of hydrogen charged asymmetric CRB specimen.

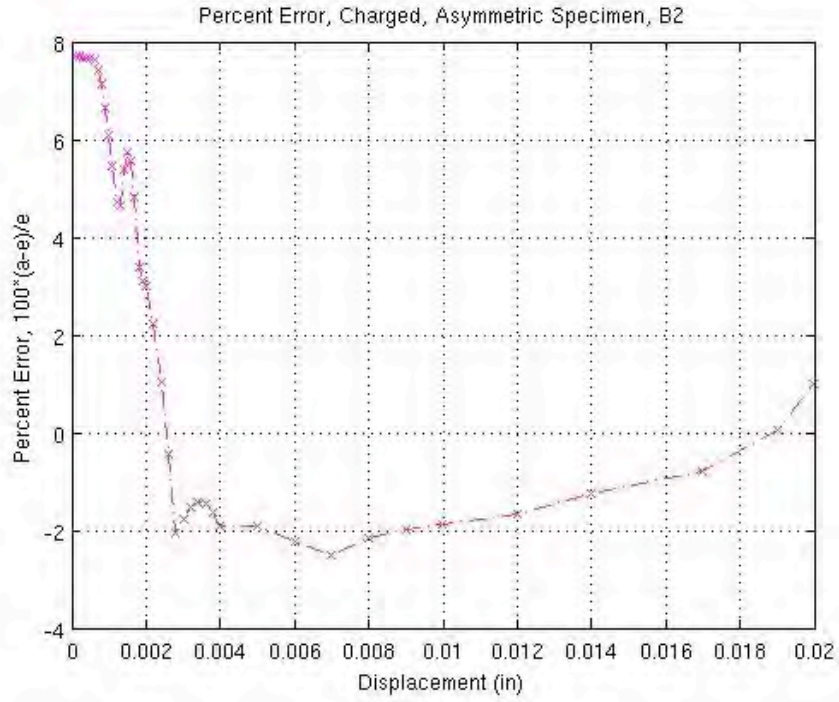


Figure 74 Error analysis of load versus displacement behavior of FEA results as compared with experimental data for hydrogen charged asymmetric CRB specimen.

6. Summary and Conclusions

The primary goal of this project was to determine the feasibility of using the J3d code in conjunction with Sandia's Sierra mechanics codes to perform J-integral evaluations in geometries with short cracks. A series of experiments and analyses was performed to verify and validate the process by which we determine J-integrals. Crack tip regions in the meshes were meshed with standard hexes and collapsed hexes and these results were compared with one another. Both annealed and hydrogen charged specimens were tested and analyzed.

- Single edge notched bend Ramberg-Osgood material model to verify J3d code against analytical solution published by Electric Power Research Institute, EPRI
- Notched tension specimens of varying notch depths to fit material data to EMMI model
- Compact tension specimen with EMMI material model
- Symmetric circumferentially cracked round bar with EMMI model and varying crack depths
- Asymmetric cracked round bar with EMMI model

Experimental results and methods will be discussed in a separate report. Using the J-integral values calculated from experimental results as the "actual" value, errors were calculated for the load versus displacement behavior as well as the J-integral versus load behavior.

We were able to see the effects of varying crack ratios and charging in the three-dimensional CRB series of tests and analyses. Comparisons between results from the standard hex meshes and the meshes with collapsed elements at the crack tip support the recommendation that collapsed elements be used at the crack tip. In general, we are better able to predict bulk behavior of the hydrogen charged specimens than of the annealed specimens.

Regardless of the charging status or geometry of a specimen, it would be expected that errors in the load versus displacement behavior would be reflected in the stress profile at the crack tip, and thus propagate into the J-integral calculations with similar error values and trends. For the symmetric CRB specimens with medium and long cracks, for both the annealed and hydrogen charged samples, this expectation is met.

Finite element analysis of both the annealed and hydrogen charged short cracked CRB specimens predicted experimental results within a few percent error. The J-integral estimates from J3d, however, had errors that were 5-10 times greater than in the load versus displacement behavior. Not only were the magnitudes of the errors calculated for the short cracked specimens undesirable, the trends of the behavior were also troubling. It would be expected that positive errors in the bulk displacement prediction would correspond to positive errors in the J-integral

calculation, with errors changing sign at the same load value. This is true for the medium and long cracked specimens, but it does not hold true for the short cracked specimens. Overall, we were unable to confirm that, for the case of short cracks, estimates of the J-integral from J3d are consistent with estimates of J obtained from experimental data using the standardized, semi-analytical relations. It is unclear whether the fault lies in the implementation of J3d, or in assumptions made during the development of those semi-analytical relations that are violated when applied to short cracks.

As mentioned in the previous section, evaluation of the J3d code was not possible for the asymmetric, three dimensional specimens as a method and analytical expression for estimating J from experimental data is not established for such geometries. Potentially, the J3d code could be used to quantify J for numerous points along the crack front in such a specimen. However, it is apparent that the construction of suitable paths necessary for the calculation of J is an unacceptably convoluted and mistake-prone process. Further work to refine and automate the path selection process is warranted for using J3d in the analysis of complex geometries, such as those in gas transfer systems.

Our efforts have shown that while J3d reasonably predicts J-integral values for medium and long cracked specimens, another method is recommended for short cracked specimens. As the specimens with shorter cracks are undetectable by current surveillance techniques, proving a resistance to fracture by analytical methods becomes of utmost importance to the ability to qualify gas transfer systems to our own specifications.

Appendix A: How to Compile and Run J3D

The following are instructions on how to compile and run the J3D code:

1. untar the tarball: tar -xvf j3d.tar

2. make sure you have ACCESS in your PATH

3. generate a makefile using the following: accmkmf

4. make the executable by tying: make

5. you now should have the j3d executable, j3dexe

6. set the following environment variables. In bash perform:
   export FOR007=name_of_j3d_input_file
   export FOR011=name_of_input_exodus_file
   export FOR012=name_of_output_exodus_file

7. run j3d: ./j3dexe

8. there are two output files from j3d:
   J.LIS - a text file containing input information and a list of the J values for each plane and each path at each time step of the simulation. It also contains the zero area J value for each plane, along with regression coefficients.
   name_of_output_exodus_file - an exodus file that contains all of the data from the input exodus file plus zero area J values for each plane at each time step.

Appendix B: Path Generation Script for J3D

This Appendix contains the Perl script J3d_general_paths.pl for defining concentric element paths enclosing a crack tip for a given mesh and provide them to the J3D code. It also contains a separate module, j3d_beam_subs.pm, for performing some of the lower level searches, and two additional modules, tims_general_subs.pm and tims_netcdf_subs_4_9_06.pm, for accessing information from Exodus II files.

j3d_general_paths.pl

```
#! /usr/bin/perl

use strict;
use lib "/home/alindbl/Projects/JIntegral/Perl/modules";
use tims_netcdf_subs_4_9_06;
use j3d_general_subs;
use FileHandle;

###############################################################################
#
# Some input checking
#
#### Test Input
die "Usage: j3d_paths.pl <exodus_file> <inner_crack_node> <crack_plane_dir> <next_plane_dir>
<num_paths> <num_planes> <path_offset=1>\n" if (@ARGV > 7);
die "Usage: j3d_paths.pl <exodus_file> <inner_crack_node> <crack_plane_dir> <next_plane_dir>
<num_paths> <num_planes> <path_offset=1>\n" if (@ARGV < 6);

## set standard out to flush after every print call
STDOUT->autoflush(1);

## grab the input arguments
my $exo_file = $ARGV[0];
die "Could not find input file $exo_file\n" if (! -e $exo_file);
my $inner_crack_node = $ARGV[1];
my $crack_plane_dir = $ARGV[2];
my $next_plane_dir = $ARGV[3];
my $num_paths = $ARGV[4];
my $num_planes = $ARGV[5];
my $offset = 1;
if (@ARGV > 6) { $offset = $ARGV[6]; }
## the output file which will be used as input for j3d
my $out_file = "output.dat";

## do some input error checking
die "ERROR: The plane_offset must not be < 0, you specified $offset\n" if ( $offset < 0 );
die "ERROR: The number of paths must be > 0, you specified $num_paths\n" if ( $num_paths < 1 );
die "ERROR: The number of planes must be > 0, you specified $num_planes\n" if ( $num_planes < 1
);
#
#
###############################################################################
```

```
##############################################################################
#
#### Gather Data from Exodus File
#
#
my $file_id = open_exodus($exo_file);
# get the number of nodes
my $num_nodes = get_num_nodes($file_id);
print STDOUT "# Nodes = $num_nodes\n";

# get the number of elements
my $num_elems = get_num_elems($file_id);
print STDOUT "# Elems = $num_elems\n";

# get the number of time steps
my $num_timesteps = get_num_timesteps($file_id);
print STDOUT "# Time Steps = $num_timesteps\n";

# get the node map, which is an array that given the internal
# node number returns the global nodal id
my @node_map = get_node_map($file_id);
print STDOUT "Node Number Map Obtained\n";

# get the nodal coordinates
my @nodal_coords = get_nodal_coords($file_id);
print STDOUT "Nodal Coordinates Obtained\n";

# similar to the node map, this maps an internal element
# id to the global element id
my @elem_map = get_elem_map($file_id);
print STDOUT "Element Number Map Obtained\n";

# this returns an array of arrays such that given an element
# id, it gives you an array of the nodes connected to that element
my @elem_conn = get_elem_connectivity($file_id);
print STDOUT "Element Connectivity Map Obtained\n";

# this is a map that given a node id, it returns an array
# of element ids it is attached to
my @node_to_elem = make_node_to_elem_map(\@elem_conn, $num_elems, $num_nodes);
print STDOUT "Made Node to Element Map\n";
close_exodus($file_id);
#
#
##############################################################################

#### Output Nodes and Element Conectivity to Setup File
open(SFILE, ">$out_file") or die "Can't open $out_file: $!\n";


##############################################################################
#
## get the local index of the inner crack node
#
my $icn_index = -1;
#print STDOUT "inner_crack_node = $inner_crack_node\n";
```

```
for (my $i=1; $i <= $num_nodes; ++$i ) {
   if ($inner_crack_node eq @node_map[$i]) {
      $icn_index = $i;
      $i = $num_nodes + 1;
   }
}
die "ERROR:  Could not locate the inner crack node\n" if ($icn_index eq -1);
#
###############################################################################


###############################################################################
#
# currently this script spits out a default youngs modulus
# and poisson ratio... this can be changed in the output file
print SFILE "YOUNGS 30.e6\n";
print SFILE "POISSON 0.3\n\n\n";
###############################################################################


###############################################################################
#
# Some pre looping set up:
#
# Determine the beginning element in the first plane and the first path
# Determine the crack tip nodes
# Calculate the center point between the two crack tip nodes
#
## get the z value of the inner crack node
my $icn_loc = $nodal_coords[$icn_index];
my $curr_z = $icn_loc->[2];

## get the number of elements the seed node is connected to
my $num_conn_elem = $#{$node_to_elem[$icn_index]} + 1;

## if the inner crack tip node is not connected to 1 or 2 elements, die
## if the mesh has collapsed elements, the inner crack node will only be
## connected to 1 element, if the elements are not collapsed it will be
## connected to 2 elements
die "ERROR:  The seed node passed in must only be connected to 2 element.\n
The node passed in is connected to $num_conn_elem elements\n
Please try again.\n" if( $num_conn_elem != 1 && $num_conn_elem != 2 );

## get the index number of the one/two elements who share the starting node
my $elem_1 = $node_to_elem[$icn_index][0];
my $elem_2 = $node_to_elem[$icn_index][1];
print STDOUT "elem_1 = $elem_1\n";
print STDOUT "elem_2 = $elem_2\n";

## determine which of those elements is in the direction of the crack plane
my $next_elem = get_first_elem( \@elem_conn, \@nodal_coords, $elem_1, $elem_2,
$crack_plane_dir  );

## determine the first element of the first path of the first plane
my $next_plane_seed_elem = $next_elem;
for ( my $i = 0; $i < $offset; ++$i ) {
```

```
   $next_plane_seed_elem= get_next_elem( \@node_to_elem, \@elem_conn, \@nodal_coords,
$next_plane_seed_elem, $num_elems, $crack_plane_dir  );
}
print STDOUT "next_plane_seed_elem = $next_plane_seed_elem\n";

## get the crack tip nodes for the first plane
## assume crack tip lies along the z-axis
my $crack_tip_node_1 = $icn_index;
my $crack_tip_node_2 = get_next_crack_tip_node( $crack_tip_node_1, \@nodal_coords,
\@{$elem_conn[$next_elem]}, \@node_map );
print STDOUT "crack_tip_node_1 = $crack_tip_node_1\n";
print STDOUT "crack_tip_node_2 = $crack_tip_node_2\n";

## get the middle point between these two nodes
## this is used to determine direction vectors for the path around the crack tip nodes
my @center = (0.0, 0.0, 0.0);
$center[0] = $nodal_coords[$node_map[$crack_tip_node_1]][0] +
$nodal_coords[$node_map[$crack_tip_node_2]][0];
$center[1] = $nodal_coords[$node_map[$crack_tip_node_1]][1] +
$nodal_coords[$node_map[$crack_tip_node_2]][1];
$center[2] = $nodal_coords[$node_map[$crack_tip_node_1]][2] +
$nodal_coords[$node_map[$crack_tip_node_2]][2];
$center[0] /= 2.0;
$center[1] /= 2.0;
$center[2] /= 2.0;
print STDOUT "center = $center[0], $center[1], $center[2]\n";
#
#
################################################################################

################################################################################
#
# The main loop
#
# This will determine the elements for all paths in all planes
#
## a counter, used to introduce line breaks in the input deck.  Since the supes library
## truncates input to 132 characters we need to introduce a line continuation before this
## to ensure proper inputs to j3d
my $elem_counter;
## the number of elements that will cause a path description to become longer than 128 char
## this is really dependent on how many elements are in the mesh, but 15 seems to work well ,
## even with meshes that have over a million elements
my $max_num_elem_per_path = 15;

## keeps track of the number of elements in each path
my $path_elem_count = 0;

## loop over the number of planes
for (my $i = 1; $i <= $num_planes; ++$i) {
   print STDOUT "Working on plane $i\n";
   my $first_path_elem = $next_plane_seed_elem;
   print SFILE "PLANE\n";

   ## write the crack tip information
```

```
   print SFILE "CRACK TIP NODES $node_map[$crack_tip_node_1]
$node_map[$crack_tip_node_2]\n";
   print STDOUT "CRACK TIP NODES $node_map[$crack_tip_node_1]
$node_map[$crack_tip_node_2]\n";

   ## loop over the number of paths per plane
   for (my $j = 1; $j <= $num_paths; ++$j ) {
      ## reset the element counter
      $elem_counter = 0;
      print STDOUT "Finding path $j for plane $i.";
      my $steps_in_y = $j + $offset;
      my $steps_in_x = 2*($offset + $j) - 1;
      my $path_elem = $first_path_elem;
      print SFILE "PATH ";


      ## for each path we keep looking until we hit the "stop" number
      while ( $path_elem != -9999999999 ) {
         print SFILE "$elem_map[$path_elem] ";
         $path_elem = get_next_arc_elem( \@node_to_elem, \@elem_conn, \@nodal_coords,
$path_elem, $num_elems, \@center, \@node_map );
         print STDOUT ".";

         ## increment the counters
         $elem_counter++;
         $path_elem_count++;
         if ( $elem_counter > 15 ) {
            print SFILE " *\n";
            $elem_counter = 0;
         }
      }

      print SFILE "\n";
      print STDOUT "\n";
      print STDOUT "Number of elements in path = $path_elem_count\n";
      $path_elem_count = 0;
      ## get next starting path elem
      $first_path_elem = get_next_elem( \@node_to_elem, \@elem_conn, \@nodal_coords,
$first_path_elem, $num_elems, $crack_plane_dir );
   }

   ## determine the new information for the next crack plane:
   #     first element
   #     crack tip nodes
   #     center of the two crack tip nodes
   print SFILE "END PLANE\n\n";
   if ( $i < $num_planes ) {
      ## get the next plane seed element
      $next_plane_seed_elem = get_next_elem( \@node_to_elem, \@elem_conn, \@nodal_coords,
$next_plane_seed_elem, $num_elems, $next_plane_dir );
      ## get the next crack tip node
      $next_elem = get_next_elem( \@node_to_elem, \@elem_conn, \@nodal_coords, $next_elem,
$num_elems, $next_plane_dir );
      print "next_elem = $next_elem\n";
      $crack_tip_node_1 = $crack_tip_node_2;
```

```
        $crack_tip_node_2 = get_next_crack_tip_node( $crack_tip_node_1, \@nodal_coords,
\@{$elem_conn[$next_elem]}, \@node_map  );
        $center[0] = $nodal_coords[$crack_tip_node_1][0] + $nodal_coords[$crack_tip_node_2][0];
        $center[1] = $nodal_coords[$crack_tip_node_1][1] + $nodal_coords[$crack_tip_node_2][1];
        $center[2] = $nodal_coords[$crack_tip_node_1][2] + $nodal_coords[$crack_tip_node_2][2];
        $center[0] /= 2.0;
        $center[1] /= 2.0;
        $center[2] /= 2.0;
    }
}

# finish the file
print SFILE "EXIT\n";
close SFILE;
print STDOUT "Done \n";

exit;
```

j3d_beam_subs.pm

```
#! /usr/bin/perl

use strict;
use NetCDF;
use tims_general_subs;

sub get_next_elem {
    my $elem_conn_ref = $_[0];
    my $nodal_coords_ref = $_[1];
    my $curr_elem = $_[2];
    my $num_elems = $_[3];
    my $dir_str = $_[4];
    my $max_min = $_[5];
    my @elem_conn = @$elem_conn_ref;
    my @nodal_coords = @$nodal_coords_ref;
    my $num_match = 0;
    my $curr_node;
    my $curr_elem_conn = $elem_conn[$curr_elem];
    my $result;
    my $curr_elem_val;
    my $dir;
    my @match_list = ();

    if ( $dir_str eq "x" ) {
        $dir = 0;
    } elsif ( $dir_str eq "y" ) {
        $dir = 1;
    } elsif ( $dir_str eq "z" ) {
        $dir = 2;
    } else {
        die "ERROR: Did not specify a supported direction when calling 'get_next_elem'.  Must be 'x', 'y'
or 'z'.\n";
    }
```

```
   my $elem_val = get_max_or_min_elem_coord( $dir, $max_min, \@{$elem_conn[$curr_elem]},
\@nodal_coords );


   for ( my $i = 1; $i <= $num_elems; ++ $i ) {
      if ( $i == $curr_elem ) {
         next;
      }
      for my $j ( 0 .. $#{$elem_conn[$i]} ) {
         $curr_node = $elem_conn[$i][$j];
         for my $k ( 0 .. $#{$elem_conn[$curr_elem]} ) {
            if ( $curr_node == $elem_conn[$curr_elem][$k] ) {
               $num_match++;
               push @match_list, $curr_node;
               last;
            }
         }
      }

      if ( $num_match == 4 ) {
         ## get the normal vector of the plane of matching nodes
         my @s = ( );
         $s[0] = $nodal_coords[$match_list[1]][0] - $nodal_coords[$match_list[0]][0];
         $s[1] = $nodal_coords[$match_list[1]][1] - $nodal_coords[$match_list[0]][1];
         $s[2] = $nodal_coords[$match_list[1]][2] - $nodal_coords[$match_list[0]][2];
         my @t = ( );
         $t[0] = $nodal_coords[$match_list[3]][0] - $nodal_coords[$match_list[0]][0];
         $t[1] = $nodal_coords[$match_list[3]][1] - $nodal_coords[$match_list[0]][1];
         $t[2] = $nodal_coords[$match_list[3]][2] - $nodal_coords[$match_list[0]][2];

         my @n = cross_prod( \@s, \@t );

         my $max_dir;
         $n[0] = abs($n[0]);
         $n[1] = abs($n[1]);
         $n[2] = abs($n[2]);
         if ( $n[0] >= $n[1] && $n[0] >= $n[2] ) {
            $max_dir = "x";
         } elsif ( $n[1] >= $n[2] && $n[1] >= $n[0] ) {
            $max_dir = "y";
         } elsif ( $n[2] >= $n[0] && $n[2] >= $n[1] ) {
            $max_dir = "z";
         }


         $curr_elem_val = get_max_or_min_elem_coord( $dir, $max_min, \@{$elem_conn[$i]},
\@nodal_coords );
         if ( ($max_min eq "max" && $curr_elem_val > $elem_val && $max_dir eq $dir_str) ||
              ($max_min eq "min" && $curr_elem_val < $elem_val && $max_dir eq $dir_str) ) {
            $result = $i;
            last;
         }
      }
      $num_match = 0;
      @match_list = ();
   }
```

```
    return $result;
}


sub get_max_or_min_elem_coord {
    my $dir = $_[0];
    my $max_min = $_[1];
    my $elem_conn_ref = $_[2];
    my $nodal_coords_ref = $_[3];
    my @elem_conn = @$elem_conn_ref;
    my @nodal_coords = @$nodal_coords_ref;
    my $result;
    my $val = $nodal_coords[$elem_conn[0]][$dir];
    $result = $val;

#    print "$nodal_coords[$elem_conn[0]][0] $nodal_coords[$elem_conn[0]][1]
$nodal_coords[$elem_conn[0]][2]\n";
    for my $i ( 1 .. $#elem_conn ) {
#        print "$nodal_coords[$elem_conn[$i]][0] $nodal_coords[$elem_conn[$i]][1]
$nodal_coords[$elem_conn[$i]][2]\n";
        $val = $nodal_coords[$elem_conn[$i]][$dir];
        if ( ($max_min eq "max" && $val > $result) ||
             ($max_min eq "min" && $val < $result) ) {
            $result = $val;
        }
    }

    return $result;
}

sub get_next_crack_tip_node {
    my $node_1 = $_[0];
    my $nodal_coords_ref = $_[1];
    my $next_elem_conn_ref = $_[2];
    my @nodal_coords = @$nodal_coords_ref;
    my @next_elem_conn  = @$next_elem_conn_ref;
    my @dir = ();
    my $result;
    my $node_1_index;

    for my $i ( 0 .. $#next_elem_conn ) {
        my $node_index = $next_elem_conn[$i];
        if ( $node_index == $node_1 ) {
            $node_1_index = $i;
            last;
        }
    }
    if ( $node_1_index < 4 ) {
        $result = $next_elem_conn[$node_1_index+4];
    } else {
        $result = $next_elem_conn[$node_1_index-4];
    }

    return $result;
}
```

```
sub get_next_arc_elem {
    my $node_to_elem_ref = $_[0];
    my $elem_conn_ref = $_[1];
    my $nodal_coords_ref = $_[2];
    my $curr_elem = $_[3];
    my $num_elems = $_[4];
    my $center_ref = $_[5];
    my @node_to_elem = @$node_to_elem_ref;
    my @elem_conn = @$elem_conn_ref;
    my @nodal_coords = @$nodal_coords_ref;
    my @center = @$center_ref;
    my $num_match = 0;
    my $curr_node;
    my $curr_elem_conn = $elem_conn[$curr_elem];
    my $result;
    my $curr_elem_val;
    my $dir;
    my @curr_elem_conn = @{$elem_conn[$curr_elem]};

    ## get the centroid of the element
    my @centroid = (0.0, 0.0, 0.0);

    for my $i ( 0 .. $#curr_elem_conn ) {
        $centroid[0] += $nodal_coords[$curr_elem_conn[$i]][0];
        $centroid[1] += $nodal_coords[$curr_elem_conn[$i]][1];
        $centroid[2] += $nodal_coords[$curr_elem_conn[$i]][2];
    }
    $centroid[0] /= ($#curr_elem_conn+1);
    $centroid[1] /= ($#curr_elem_conn+1);
    $centroid[2] /= ($#curr_elem_conn+1);

    ## determine the ray vector from the center to the centroid
    my @ray_vec = (0.0, 0.0, 0.0);

    $ray_vec[0] = $centroid[0] - $center[0];
    $ray_vec[1] = $centroid[1] - $center[1];
    $ray_vec[2] = $centroid[2] - $center[2];

    @ray_vec = unit_Vector( \@ray_vec );

    ## determine the dir vector, which is perpendicular to the
    ## ray vector in the direction of interest
    my @dir_vec = ( -$ray_vec[1], $ray_vec[0], 0.0 );

    ## find the nodes on the face in the direction of the dir vector
    my ($node_1, $node_2, $node_3);

    ($node_1, $node_2, $node_3) = get_arc_face_nodes( \@centroid, \@dir_vec,
\@{$elem_conn[$curr_elem]}, \@nodal_coords );

    ## find the element that shares these nodes
    my $node_1_elems = $node_to_elem[$node_1];
    my $node_2_elems = $node_to_elem[$node_2];
    my $node_3_elems = $node_to_elem[$node_3];
    my $my_next_elem = -9999999999;
```

```
    for my $i ( 0 .. $#{$node_1_elems} ) {
        my $n1e = $node_to_elem[$node_1][$i];

        for my $j ( 0 .. $#{$node_2_elems} ) {
            my $n2e = $node_to_elem[$node_2][$j];

            if ( $n1e == $n2e && $n1e != $curr_elem ) {
                for my $k ( 0 .. $#{$node_3_elems} ) {
                    my $n3e = $node_to_elem[$node_3][$k];

                    if ( $n2e == $n3e ) {
                        $my_next_elem = $n3e;
                        last;
                    }
                }
                if ( $my_next_elem == $n2e ) {
                    last;
                }
            }
        }
        if ( $my_next_elem == $n1e ) {
            last;
        }
    }
    $result = $my_next_elem;
    return $result;
}

sub get_arc_face_nodes {
    my $centroid_ref = $_[0];
    my $dir_ref = $_[1];
    my $elem_conn_ref = $_[2];
    my $nodal_coords_ref = $_[3];
    my @centroid = @$centroid_ref;
    my @dir_vec = @$dir_ref;
    my @elem_conn = @$elem_conn_ref;
    my @nodal_coords = @$nodal_coords_ref;
    my $result;
    my $node_1;
    my $node_2;
    my $node_3;

    my $i;
    my @node_dir = (0.0, 0.0, 0.0);
    my $dot_val;
    my @result = (0, 0, 0);
    my $index = 0;
    my @max_dot = (0.0, 0.0, 0.0 );

    for $i ( 0 .. $#elem_conn ) {
        $node_dir[0] = $nodal_coords[$elem_conn[$i]][0] - $centroid[0];
        $node_dir[1] = $nodal_coords[$elem_conn[$i]][1] - $centroid[1];
        $node_dir[2] = $nodal_coords[$elem_conn[$i]][2] - $centroid[2];
        @node_dir = unit_Vector( \@node_dir );

        $dot_val = dot_prod( \@node_dir, \@dir_vec );
```

```
        if ( $dot_val > $max_dot[0] ) {
            $result[2] = $result[1];
            $result[1] = $result[0];
            $result[0] = $elem_conn[$i];
            $max_dot[2] = $max_dot[1];
            $max_dot[1] = $max_dot[0];
            $max_dot[0] = $dot_val;
        } elsif ( $dot_val > $max_dot[1] ) {
            $result[2] = $result[1];
            $result[1] = $elem_conn[$i];
            $max_dot[2] = $max_dot[1];
            $max_dot[1] = $dot_val;
        } elsif ( $dot_val > $max_dot[2] ) {
            $result[2] = $elem_conn[$i];
            $max_dot[2] = $dot_val;
        }
    }
    return @result;
}

1;
```

tims_general_subs.pm

```
#! /usr/bin/perl

use strict;

sub get_lines {
        ### Returns an Array of Lines from the File given
        ### @lines = get_lines("file");
        ###
        my $file = $_[0];
        die "Could not find file $file\n" if (! -e $file);
        my @lines = ();
        open(INPUTFILE,"$file") or die "Unable to open $file";
        @lines = <INPUTFILE>;
        chomp(@lines);
        close(INPUTFILE);
        return @lines;
}

sub write_lines {
        ### Writes an Array of Lines to the File
        ### write_lines("file",\@lines);
        ###
        my $file = $_[0];
            my $lines_ref = $_[1];
        my @lines = @$lines_ref;
        open(INPUTFILE,">$file") or die "Unable to open $file";
            foreach my $line (@lines){
                    print INPUTFILE "$line\n";
            }
        close(INPUTFILE);
            return 1;
```

```
}

sub get_corresponding_string {
	### Search @a for $a_val and return value of @b at that location
	### $b_val = get_corresponding_string($a_val,\@a,\@b);
	my $a_val = $_[0];
	my $aref = $_[1];
	my $bref = $_[2];
	my @a = @$aref;
	my @b = @$bref;
	my $bval;
	for (my $i=0; $i<=$#b; ++$i) {
		if ($a[$i] =~ m/^$a_val$/) {
			if ($b[$i]) {
				$bval = $b[$i];
				return $bval;
			}
		}
	}
	die "Failed to find $a_val in array or array b has no value"
	  ." at that location.\n" unless ($bval);
}

sub search_and_substitue {
	### Search through @lines for things in @find
	### and replace with @replace
	### search_and_substitue(\@lines,\@find,\@replace);
	my $lines_ref = $_[0];

	my $find_ref = $_[1];
	my $replace_ref = $_[2];
	my @find = @$find_ref;
	my @replace = @$replace_ref;
	my $j = 0;
	foreach my $line_compare (@$lines_ref) {
		my $i=0;
		foreach my $find_in (@find) {
			if ($line_compare =~ m/$find_in/) {
				$line_compare =~ s/$find_in/$replace[$i]/g;
			}
			++$i;
		}
		$$lines_ref[$j]=$line_compare;
		++$j;
	}
	return 1;
}

sub contain_atleast {
	### Search through @a and return @b with the values that
	### contain atleast $string
	### @b = contain_atleast(\@a,$string);
	my $a_ref = $_[0];
	my $string = $_[1];
	my @b = ();
	foreach my $content (@$a_ref) {
```

```
                if ($content =~ m/$string/) {
                        push(@b,$content);
                }
        }
        return @b;
}

sub maxABS {
        ### Returns Maximum Absoulte Value of Array
        ### $max = maxABS(\@a);
        my $a_ref = $_[0];
        my $max = 0;
        foreach my $val (@$a_ref) {
                if (abs($val) > $max) {
                        $max = abs($val);
                }
        }
        return $max;
}

sub maxVAL {
      ### Returns Max Value of Array
      ### $max = maxVAL(\@a);
      my $a_ref = $_[0];
      my $max = -1.0e20;
      foreach my $val (@$a_ref) {
          if ($val > $max) {
              $max = $val;
          }
      }
      return $max;
}

sub minVAL {
      ### Returns Min Value of Array
      ### $min = minVAL(\@a);
      my $a_ref = $_[0];
          my $min = 1.0e20;
      foreach my $val (@$a_ref) {
          if ($val < $min) {
              $min = $val;
          }
      }
      return $min;
}

sub scaleVEC {
      ### Returns Scaled Vector
      ### @b = scaleVEC(\@a,$scale);
      my $a_ref = $_[0];
      my $scale = $_[1];
          my @b = ();
      foreach my $val (@$a_ref) {
                  push(@b,$val*$scale);
          }
          return @b;
```

```
}

sub sumVEC {
	### Returns Sum of Vector Values
	### $sum = sumVEC(\@a);
	my $a_ref = $_[0];
	my $sum = 0;
	foreach my $val (@$a_ref) {
			$sum = $sum + $val;
	}
	return $sum;
}


sub addVEC {
	### Returns Vectors Added Together
	### @c = addVEC(\@a,\@b);
	my $a_ref = $_[0];
	my $b_ref = $_[1];
	    my @a = @$a_ref;
	    my @b = @$b_ref;
	my @c = ();
	foreach my $val (@a) {
	      my $val2 = shift(@b);
			push(@c,($val+$val2));
	}
	return @c;
}

sub point_to_point_distance {
		### Return Distance Between Two Points
		### $dist = point_to_point_distance(\@p1,\@p2);
		###
		my $p1_ref = $_[0];
		my $p2_ref = $_[1];
		my @p1 = @$p1_ref;
		my @p2 = @$p2_ref;
		my $dist = 0;
		my $num = @p1;
		for (my $i=0;$i<$num;++$i) {
			$dist = $dist + ($p1[$i]-$p2[$i])*($p1[$i]-$p2[$i]);
		}
		$dist = sqrt($dist);
		return $dist;
}

sub magnitude_Vector {
		### Return Magnitude of Vector
		### $mag = magnitude_Vector(\@a);
		my $a_ref = $_[0];
		my @a = @$a_ref;
		my $mag = 0;
		foreach $a (@a) {
			$mag = $mag + $a*$a;
		}
		$mag = sqrt($mag);
```

```
        return $mag;
}

sub unit_Vector {
        ### Return Unit Vector
        ### @unitV = unit_Vector(\@a);
        my $a_ref = $_[0];
        my @a = @$a_ref;
        my $mag = magnitude_Vector(\@a);
    my @unitV = ( $a[0], $a[1]. $a[3] );
    if ( $mag > 0.0 ) {
       @unitV = scaleVEC(\@a,(1/$mag));
    }
        return @unitV;
}

sub point_to_point_SQdistance {
    ### Return Square Distance Between Two Points
    ### $dist = point_to_point_SQdistance(\@p1,\@p2);
    ###
    my $p1_ref = $_[0];
    my $p2_ref = $_[1];
    my @p1 = @$p1_ref;
    my @p2 = @$p2_ref;
    my $dist = 0;
    my $num = @p1;
    for (my $i=0;$i<$num;++$i) {
          $dist = $dist + ($p1[$i]-$p2[$i])*($p1[$i]-$p2[$i]);
    }
    return $dist;
}

sub point_to_plane_distance {
        ### Return Distance between point and plane
        ### $dist = point_to_plane_distance(\@point,\@plane);
        ###
        my ($point_ref,$plane_ref) = @_;
        my ($x,$y,$z) = @$point_ref;
        my ($a,$b,$c,$d) = @$plane_ref;
        my $dist = ($a*$x+$b*$y+$c*$z+$d)/(sqrt($a*$a+$b*$b+$c*$c+$d*$d));
        return $dist;
}

sub closest_point_on_plane {
        ### Return Closest Point on Plane to given Point
        ### @p = closest_point_on_plane(\@point,\@plane);
        my ($point_ref,$plane_ref) = @_;
    my ($x,$y,$z) = @$point_ref;
    my ($a,$b,$c,$d) = @$plane_ref;
        my @p0 = ($x,$y,$z);
        my $dist = ($a*$x+$b*$y+$c*$z+$d)/(sqrt($a*$a+$b*$b+$c*$c+$d*$d));
        my @n = ($a,$b,$c);
        my $magn = magnitude_Vector(\@n);
        my @n = scaleVEC(\@n,-1*($dist/$magn));
        my @p = addVEC(\@n,\@p0);
        return @p;
```

```
}

sub construct_Vector {
	### Return Vector Made from Two Points
	### @V = construct_Vector(\@tip,\@tail);
	my ($tip_ref,$tail_ref) = @_;
	my @tip = @$tip_ref;
	my @tail = @$tail_ref;
	my @V = ();
	my $ntip = @tip;
	my $ntail = @tail;
	die "ERROR: Vectors not same length.\n" if ($ntip != $ntail);
	for (my $i = 0;$i<$ntip;++$i) {
		push(@V,$tip[$i]-$tail[$i]);
	}
	return @V;
}

sub construct_Plane {
	### Create and Return a Plane made of 3 points
	### @plane = construct_Plane(\@p1,\@p2,\@p3);
	###
	my ($p1_ref,$p2_ref,$p3_ref) = @_;
	my ($x,$y,$z) = @$p1_ref;
	my @V1 = construct_Vector($p1_ref,$p2_ref);
	my @V2 = construct_Vector($p2_ref,$p3_ref);
	my @plane = cross_prod(\@V1,\@V2);
	my ($a,$b,$c) = @plane;
	my $d = (-1*$a*$x-$b*$y-$c*$z);
	push(@plane,$d);
	return @plane;
}

sub cross_prod {
	### Return Cross Product of @a and @b
	### @c = cross_prod(\@a,\@b);
	my ($a_ref,$b_ref) = @_;
	my @a = @$a_ref;
	my @b = @$b_ref;
	my @c = ();
	$c[0] = $a[1]*$b[2]-$a[2]*$b[1];
	$c[1] = $a[2]*$b[0]-$a[0]*$b[2];
	$c[2] = $a[0]*$b[1]-$a[1]*$b[0];
	return @c;
}

sub dot_prod {
	### Return Dot Product of @a and @b
	### $c = dot_prod(\@a,\@b);
    my ($a_ref,$b_ref) = @_;
    my @a = @$a_ref;
    my @b = @$b_ref;
    my $c = 0;
	for (my $i=0; $i<@a; ++$i) {
		$c = $c + $a[$i]*$b[$i];
	}
```

```
        return $c;
}


1;
```

tims_netcdf_subs_4_9_06.pm

```
#! /usr/bin/perl

use strict;
use NetCDF;
use tims_general_subs;

sub open_exodus {
        ### Open Exodus File and return $file_id
        ### open_exodus("filename.e");
        my $exo_file = $_[0];
        die "Could not find file $exo_file\n" if (! -e $exo_file);
    my $file_id = NetCDF::open($exo_file, NetCDF::NOWRITE); ### Open as Read Only
    die "ERROR: Could not open exodus file $exo_file\n" if ($file_id == -1);
        print "Opened $exo_file.\n";
        return $file_id;
}

sub open_exodus_for_write {
    ### Open Exodus File and return $file_id
    ### open_exodus("filename.e");
    my $exo_file = $_[0];
    die "Could not find file $exo_file\n" if (! -e $exo_file);
    my $file_id = NetCDF::open($exo_file, NetCDF::WRITE); ### Open with Write Priviliges
    die "ERROR: Could not open exodus file $exo_file\n" if ($file_id == -1);
    print "Opened $exo_file with write priviliges.\n";
    return $file_id;
}

sub close_exodus {
    ### Close Exodus File
    ### close_exodus("filename.e");
        my $file_id = $_[0];
    my $error = NetCDF::close($file_id);
    die "ERROR: Problem closing exodus file\n" if ($error == -1);
        print "Closed exodus file.\n";
        return $error;
}

sub exodus_info {
    ### Get Number of Dimensions, Variables, Attributes, and ID of Unlimited Dimension
        ### returns ($ndims,$nvars,$natts,$unldim)
        ### exodus_info($file_id);
        my $file_id = $_[0];
    my ($ndims,$nvars,$natts,$unldim);
        my $error;
    $error = NetCDF::inquire($file_id,$ndims,$nvars,$natts,$unldim);
    die "ERROR: Problem inquiring exodus file\n" if ($error == -1);
```

```
        return ($ndims,$nvars,$natts,$unldim);
}

sub get_dim_names {
        ### Gets the Dimension Names
        ### return @names
        ### @names = get_dim_names($file_id);
        my $file_id = $_[0];
        my ($ndims,$nvars,$natts,$unldim);
        ($ndims,$nvars,$natts,$unldim)=exodus_info($file_id);
        my ($num_dims);
        my $curname;
        my @names = ();
        for (my $i=0;$i<$ndims;++$i) {
                my $error = NetCDF::diminq($file_id,$i,$curname,$num_dims);
        die "ERROR: Problem inquiring for dimension names\n" if ($error == -1);
                $names[$i]=$curname;
        }
    return @names;
}

sub get_var_names {
    ### Gets the Variable Names
    ### return @names
    ### @names = get_var_names($file_id);
    my $file_id = $_[0];
    my ($ndims,$nvars,$natts,$unldim);
    ($ndims,$nvars,$natts,$unldim)=exodus_info($file_id);
        my ($data_type,@dim_ids);
    my $curname;
    my @names = ();
    for (my $i=0;$i<$nvars;++$i) {
          my $error = NetCDF::varinq($file_id,$i,$curname,$data_type,$ndims,\@dim_ids,$natts);
          die "ERROR: Problem inquiring for variable names\n" if ($error == -1);
          $names[$i]=$curname;
    }
    return @names;
}

sub get_num_nodes {
        ### Get the Number of Nodes in mesh
        ### returns $num_nodes
        ### get_num_nodes($file_id);
        my $file_id = $_[0];
    my $dimid;
    my $num_nodes;
    my $dim_name="num_nodes";
    $dimid = NetCDF::dimid($file_id,$dim_name);
    die "ERROR: Problem getting $dim_name 's dimension id\n" if ($dimid == -1);
        my $dim_name2;
    my $error = NetCDF::diminq($file_id,$dimid,$dim_name2,$num_nodes);
    die "ERROR: Problem inquiring. $dim_name = $dim_name2 ?\n" if ($error == -1);
        return $num_nodes;
}

sub get_num_elems {
```

```
	### Get the Number of Elements in mesh
	### returns $num_elems
	### get_num_elems($file_id);
	my $file_id = $_[0];
	my $dimid;
	my $num_elems;
	my $dim_name="num_elem";
	$dimid = NetCDF::dimid($file_id,$dim_name);
	die "ERROR: Problem getting $dim_name 's dimension id\n" if ($dimid == -1);
	my $dim_name2;
	my $error = NetCDF::diminq($file_id,$dimid,$dim_name2,$num_elems);
	die "ERROR: Problem inquiring. $dim_name = $dim_name2 ?\n" if ($error == -1);
	return $num_elems;
}

sub get_dim_size {
	### Get the Number of Dimensions
	### returns $num_dims;
	### get_dim_size($file_id,"dimension");
	my $file_id = $_[0];
	my $dim_name=$_[1];
	my $dimid;
	my $num_dims;
	$dimid = NetCDF::dimid($file_id,$dim_name);
	die "ERROR: Problem getting $dim_name 's dimension id\n" if ($dimid == -1);
	my $dim_name2;
	my $error = NetCDF::diminq($file_id,$dimid,$dim_name2,$num_dims);
	die "ERROR: Problem inquiring. $dim_name = $dim_name2 ?\n" if ($error == -1);
	return $num_dims;
}

sub get_num_timesteps {
	### Get Number of Time Steps
	    ### returns $num_timesteps
	    ### get_num_timesteps($file_id);
	    my $file_id = $_[0];
	my $num_timesteps;
	my $unl_dim_name;
	    my ($ndims,$nvars,$natts,$unldim)=exodus_info($file_id);
	my $error = NetCDF::diminq($file_id,$unldim,$unl_dim_name,$num_timesteps);
	die "ERROR: Problem inquiring exodus file about Number of Time Steps\n" if ($error == -1);
	if ($unl_dim_name =~ m/time_step/) {
	      ##print "Last Time Step = $last_time_step\n";
	} else {
	       print "Unlimited dimension name not time_step as expected.\n";
	       exit;
	}
	    return $num_timesteps;
}

sub get_node_map {
	    ### Get Node Number Map that is used for Variables
	    ### and Coordinates
	    ### returns $node_map[$num_nodes]=$node_id;
	    ### get_node_map($file_id);
	    my $file_id = $_[0];
```

```
        my $error;
        my $num_nodes=get_num_nodes($file_id);
        my @var_names = get_var_names($file_id);
        my @map_names = contain_atleast(\@var_names,"map");
        my @node_map_names = contain_atleast(\@map_names,"node");
        my @node_map=();
        if ($#node_map_names+1 != 1) {
           for ( my $i=0; $i <= $num_nodes; ++$i ) {
              $node_map[$i] = $i;
           }
        } else {
           print "Node map name = $node_map_names[0]\n";
           my $node_num_map_id = NetCDF::varid($file_id,$node_map_names[0]);
           die "ERROR: Problem getting node_num_map variable id\n" if ($node_num_map_id== -1);
           my @start=(0);
           my @count=($num_nodes);
           $error = NetCDF::varget($file_id,$node_num_map_id,\@start,\@count,\@node_map);
           die "ERROR: Problem getting variable values\n" if ($error == -1);
           die "ERROR: Node Map not equal in length to Number of Nodes.\n" if ($num_nodes !=
$#node_map+1);
           unshift(@node_map,0);
        }
        return @node_map;
}

sub get_nodal_varibles {
            ### Get Array with Nodal Values of a Variable at Time Step
            ### returns values[$num_nodes]=$node_value
            ### get_nodal_varibles($file_id,$time_step,$var_name);
            my $file_id = $_[0];
            my $time_step = $_[1];
            my $var_name = $_[2];
            my @values = ();
            my $error;
            my $num_nodes=get_num_nodes($file_id);
            my $varstorid = NetCDF::varid($file_id,"vals_nod_var");
            die "ERROR: Problem getting vals_nod_var variable id\n" if ($varstorid == -1);
        my $varid = NetCDF::varid($file_id,"name_nod_var");
            my $node_var_num = check_names($file_id,$varid,$var_name);
            die "ERROR: Could not find $var_name in nodal varibales\n" if ($node_var_num == -1);
            my @start=($time_step,$node_var_num,0);
            my @count=(1,1,$num_nodes);
            $error = NetCDF::varget($file_id,$varstorid,\@start,\@count,\@values);
            die "ERROR: Problem getting variable values\n" if ($error == -1);
            die "ERROR: Number of Variables not equal in length to Number of Nodes.\n" if ($num_nodes
!= $#values+1);
            unshift(@values,0);
            return @values;
}

sub replace_nodal_varibles {
        ### Replace the Nodal Values of a Variable at Time Step
        ### replace_nodal_varibles($file_id,$time_step,$var_name,\@values);
        my $file_id = $_[0];
        my $time_step = $_[1];
        my $var_name = $_[2];
```

```
	my $value_ref = $_[3];
	    my @values = @$value_ref;
	    my $topval = shift(@values);
	    die "ERROR: Shifted nodal variables and top value was not what expected.\n" if ($topval != 0);
	my $error;
	my $num_nodes=get_num_nodes($file_id);
	    die "ERROR: Number of Variables not equal in length to Number of Nodes.\n" if ($num_nodes
!= $#values+1);
	my $varstorid = NetCDF::varid($file_id,"vals_nod_var");
	die "ERROR: Problem getting vals_nod_var variable id\n" if ($varstorid == -1);
	    my $varid = NetCDF::varid($file_id,"name_nod_var");
	my $node_var_num = check_names($file_id,$varid,$var_name);
	die "ERROR: Could not find $var_name in nodal varibales\n" if ($node_var_num == -1);
	my @start=($time_step,$node_var_num,0);
	my @count=(1,1,$num_nodes);
	$error = NetCDF::varput($file_id,$varstorid,\@start,\@count,\@values);
	die "ERROR: Problem putting in variable values\n" if ($error == -1);
	return 1;
}

sub get_nodal_coords {
	### Get Array with Nodal Coordinates
	### returns $coords[$num_nodes]=\@loc
	    ### where @loc=($x,$y,$z);
	### get_nodal_coords($file_id);
	my $file_id = $_[0];
	my @coords = ();
	my $error;
	my $num_nodes=get_num_nodes($file_id);
	my $coordstorid = NetCDF::varid($file_id,"coord");
	die "ERROR: Problem getting coord variable id\n" if ($coordstorid == -1);
	    my $num_dim = get_dim_size($file_id,"num_dim");
	    die "ERROR: Only support 3d models and num_dim != 3\n" if ($num_dim != 3);
	    my (@x,@y,@z);
	my @start=(0,0);
	my @count=(1,$num_nodes);
	$error = NetCDF::varget($file_id,$coordstorid,\@start,\@count,\@x);
	die "ERROR: Problem getting coord values\n" if ($error == -1);
	my @start=(1,0);
	$error = NetCDF::varget($file_id,$coordstorid,\@start,\@count,\@y);
	die "ERROR: Problem getting coord values\n" if ($error == -1);
	my @start=(2,0);
	$error = NetCDF::varget($file_id,$coordstorid,\@start,\@count,\@z);
	die "ERROR: Problem getting coord values\n" if ($error == -1);
	die "ERROR: Number of Xcoords not equal in length to Number of Nodes.\n" if ($num_nodes !=
$#x+1);
	    die "ERROR: Number of Ycoords not equal in length to Number of Nodes.\n" if ($num_nodes
!= $#y+1);
	    die "ERROR: Number of Zcoords not equal in length to Number of Nodes.\n" if ($num_nodes
!= $#z+1);
	    for (my $i=0; $i < $num_nodes; ++$i) {
	        my @loc = ($x[$i],$y[$i],$z[$i]);
	        $coords[$i]=\@loc;
	    }
	    unshift(@coords,0);
	return @coords;
```

```
}

sub get_elem_map {
	### Get Elem Number Map
	### returns $elem_map[$num_elems]=$elem_id;
	### get_elem_map($file_id);
	my $file_id = $_[0];
	my $error;
	my $num_elems=get_num_elems($file_id);
	my @var_names = get_var_names($file_id);
	my @map_names = contain_atleast(\@var_names,"map");
	    my @elem_map_names = contain_atleast(\@map_names,"elem");
	my @elem_map=();
	    if ($#elem_map_names+1 != 1) {
	   for ( my $i=0; $i <= $num_elems; ++$i ) {
	      $elem_map[$i] = $i;
	   }
	} else {
	   print "Element map name = $elem_map_names[0]\n";
	   my $elem_num_map_id = NetCDF::varid($file_id,$elem_map_names[0]);
	   die "ERROR: Problem getting elem_num_map variable id\n" if ($elem_num_map_id== -1);
	   my @start=(0);
	   my @count=($num_elems);
	   $error = NetCDF::varget($file_id,$elem_num_map_id,\@start,\@count,\@elem_map);
	   die "ERROR: Problem getting variable values\n" if ($error == -1);
	   die "ERROR: Elem Map not equal in length to Number of elems.\n" if ($num_elems !=
$#elem_map+1);
	   unshift(@elem_map,0);
	}
	return @elem_map;
}

sub get_elem_connectivity {
	    ### Return Element Connectivity Map
	    ### returns $elem_con[$num_elems]=\@nodeids;
	    ### where @nodeids contains the internal id locs
	    ### that make up the connectivity of the element
	    ### get_elem_connectivity($file_id);
	    my $file_id = $_[0];
	    my @elem_con = ();
	my $error;
	my $num_elems=get_num_elems($file_id);
	    my @dim_names = get_dim_names($file_id);
	    my @var_names = get_var_names($file_id);
	    my @connect_names = contain_atleast(\@var_names,"connect");
	    my ($curname,$data_type,$ndims,@dim_ids,$natts);
	    foreach my $connect_name (@connect_names) {
	        my $varid = NetCDF::varid($file_id,$connect_name);
	        die "ERROR: Problem getting variable id\n" if ($varid== -1);
	        $error =
NetCDF::varinq($file_id,$varid,$curname,$data_type,$ndims,\@dim_ids,$natts);
	      die "ERROR: Problem inquiring for variable names\n" if ($error == -1);
	        die "ERROR: Number of dimensions used for defining connectivity unexpected.\n" if
($ndims != 2);
	        print "$connect_name ( $dim_names[$dim_ids[0]] , $dim_names[$dim_ids[1]] ) \n";
	        my $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
```

```
                my $dim2_size = get_dim_size($file_id,$dim_names[$dim_ids[1]]);
                for (my $i = 0; $i < $dim1_size; ++$i) {
                        my @start = ($i,0);
                        my @count = (1,$dim2_size);
                        my @nodeids=();
                        $error = NetCDF::varget($file_id,$varid,\@start,\@count,\@nodeids);
                    die "ERROR: Problem getting connectivity nodes\n" if ($error == -1);
                        push(@elem_con,\@nodeids);
                }
        }
        die "ERROR: Connectivity Map length not equal to number of elems.\n" if ($num_elems !=
$#elem_con+1);
        unshift(@elem_con,0);
        return @elem_con;
}

sub check_names {
        ### Return Id of Matching name or -1 if not found
        ### $id = check_names($file_id,$varid,"name");
        ###
        my $file_id = $_[0];
        my $varid = $_[1];
        my $name = $_[2];
        my ($varname,$data_type,$ndims,@dim_ids,$natts);
        my @dim_names = get_dim_names($file_id);

        my $error = NetCDF::varinq($file_id,$varid,$varname,$data_type,$ndims,\@dim_ids,$natts);
        die "ERROR: Problem inquiring for variable names\n" if ($error == -1);
        die "ERROR: Number of dimensions used when checking names unexpected.\n" if ($ndims !=
2);
    print "$varname ( $dim_names[$dim_ids[0]] , $dim_names[$dim_ids[1]] ) \n";
        my $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
        my $dim2_size = get_dim_size($file_id,$dim_names[$dim_ids[1]]);
    for (my $i = 0; $i < $dim1_size; ++$i) {
                my @start = ($i,0);
                my @count = (1,$dim2_size);
                my @NameArray = ("\0" x $dim2_size);
                $error = NetCDF::varget($file_id,$varid,\@start,\@count,\@NameArray);
          die "ERROR: Problem getting variable name\n" if ($error == -1);
          my $cur_name = "";
          for (my $j = 0; $j < $dim2_size; $j++) {
                my $chr = chr($NameArray[$j]);
                last if( $chr eq "\0" || $chr eq "\\" ) ;
                $cur_name .= $chr ;
          }
          if ($cur_name =~ m/^$name$/) {
                return $i;
          }
        }
        return -1;
}

sub get_side_set {
    ### Return Sidset $sides[$num_sides]=\@nodeids
    ### where @nodeids contains the internal id locs
    ### that make up the sides
```

```
    ### get_side_set($file_id,$surface_num,\@elem_conn);
    my $file_id = $_[0];
        my $surface_num = $_[1];
        my $elem_con_ref = $_[2];
        my $topval = $$elem_con_ref[0];
    die "ERROR: Shifted element connectivty passed into sideset routine "
         ."was not what expected.\n" if ($topval != 0);
        my @sides=();
        my @side_element=();
        my @side_face=();
        my @var_names = get_var_names($file_id);
    my @ss_names = contain_atleast(\@var_names,"ss");
        my @prop_names = contain_atleast(\@ss_names,"prop");
        die "ERROR: Can not find sideset id properties.\n" if ($#prop_names+1 != 1);
        print "Sideset ID Properties found in $prop_names[0]\n";
        my $ss_prop_id = NetCDF::varid($file_id,$prop_names[0]);
        my @ss_ids=();
    my ($varname,$data_type,$ndims,@dim_ids,$natts);
    my @dim_names = get_dim_names($file_id);
    my $error =
NetCDF::varinq($file_id,$ss_prop_id,$varname,$data_type,$ndims,\@dim_ids,$natts);
        die "ERROR: Problem inquiring variable\n" if ($error == -1);
        die "ERROR: Unexpected Number of Dimensions for sideset id properties\n" if ($ndims != 1);
        my $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
        my @start=(0);
    my @count=($dim1_size);
    $error = NetCDF::varget($file_id,$ss_prop_id,\@start,\@count,\@ss_ids);
    die "ERROR: Problem getting variable values\n" if ($error == -1);
        my $surface_index = 0;
        my $found = -1;
        foreach my $surface_id (@ss_ids) {
                if ($surface_id == $surface_num) {
                        $found = 0;
                        last;
                }
                ++$surface_index;
        }
        die "ERROR: Side Set $surface_num not found in side sets.\n" if ($found == -1);
        my @elem_names = contain_atleast(\@ss_names,"elem");
        my $elem_ss_var = $elem_names[$surface_index];
        my $elem_ss_id = NetCDF::varid($file_id,$elem_ss_var);
        print "Sideset Elements held in variable: $elem_ss_var\n";
        $error =
NetCDF::varinq($file_id,$elem_ss_id,$varname,$data_type,$ndims,\@dim_ids,$natts);
    die "ERROR: Problem inquiring variable\n" if ($error == -1);
        die "ERROR: Unexpected Number of Dimensions\n" if ($ndims != 1);
        $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
        @start=(0);
        @count=($dim1_size);
        $error = NetCDF::varget($file_id,$elem_ss_id,\@start,\@count,\@side_element);

        my @side_names = contain_atleast(\@ss_names,"side");
        my $side_ss_var = $side_names[$surface_index];
        my $side_ss_id = NetCDF::varid($file_id,$side_ss_var);
    print "Sideset Sides held in variable: $side_ss_var\n";
```

```
        $error =
NetCDF::varinq($file_id,$side_ss_id,$varname,$data_type,$ndims,\@dim_ids,$natts);
    die "ERROR: Problem inquiring variable\n" if ($error == -1);
    die "ERROR: Unexpected Number of Dimensions\n" if ($ndims != 1);
    $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
    @start=(0);
    @count=($dim1_size);
    $error = NetCDF::varget($file_id,$side_ss_id,\@start,\@count,\@side_face);

        die "ERROR: Number of Face does not match number of Elements.\n" if ($#side_face !=
$#side_element);


        for(my $i = 0; $i <= $#side_face; ++$i) {
                my $eid = $side_element[$i];
                my $fid = $side_face[$i];
                ##print "Converting face def: Element $eid, Face $fid\n";
                my $connect_ref = $$elem_con_ref[$eid];
                ##print join( ',', @$connect_ref );
                ##print "\n";
                my @facenodes = face_id_to_nodes($fid,$connect_ref);
                ##print "Face Nodes: ";
                ##print join( ',',@facenodes);
                ##print "\n";
                push(@sides,\@facenodes);
        }

    return @sides;
}

sub face_id_to_nodes {
        ### Convert Face ID to Face Nodes
        ### @facenodes = face_id_to_nodes($fid,\@node_ids);
        ###
        my $fid = $_[0];
        my $nodes_ref = $_[1];
        my @node_ids = @$nodes_ref;
        my @facenodes = ();
        die "ERROR: Only Convert Face ID to Nodes for 8 node Hex Elements.\n" if ( $#node_ids+1
!= 8);
        if ($fid == 1) {
                @facenodes=($node_ids[0],$node_ids[1],$node_ids[5],$node_ids[4]);
                return @facenodes;
        }
    if ($fid == 2) {
        @facenodes=($node_ids[1],$node_ids[2],$node_ids[6],$node_ids[5]);
        return @facenodes;
    }
    if ($fid == 3) {
        @facenodes=($node_ids[2],$node_ids[3],$node_ids[7],$node_ids[6]);
        return @facenodes;
    }
    if ($fid == 4) {
        @facenodes=($node_ids[0],$node_ids[4],$node_ids[7],$node_ids[3]);
        return @facenodes;
    }
```

```
        if ($fid == 5) {
                @facenodes=($node_ids[0],$node_ids[3],$node_ids[2],$node_ids[1]);
                return @facenodes;
        }
        if ($fid == 6) {
                @facenodes=($node_ids[4],$node_ids[5],$node_ids[6],$node_ids[7]);
                return @facenodes;
        }
            die "ERROR: = (); Face ID $fid not converted.\n" if ( 1 == 1);
}


sub get_side_sets {
        ### Return Sidsets @sidesets=(\@side_set_ids,\@side_faces)
            ### where @side_set_ids contains the side sets ID
            ### and $side_faces[$num_side_sets]=\@nodeids
        ### where @nodeids contains the internal id locs
        ### that make up the faces
        ### ($side_set_ids_ref, $side_faces_ref) = get_side_sets($file_id,\@elem_conn);
        my $file_id = $_[0];
            my $elem_con_ref = $_[1];
            my $topval = $$elem_con_ref[0];
        die "ERROR: Shifted element connectivty passed into sideset routine "
              ."was not what expected.\n" if ($topval != 0);
            my @side_set_ids =();
            my @side_faces = ();
            my @var_names = get_var_names($file_id);
        my @ss_names = contain_atleast(\@var_names,"ss");
            my @prop_names = contain_atleast(\@ss_names,"prop");
            die "ERROR: Can not find sideset id properties.\n" if ($#prop_names+1 != 1);
            print "Sideset ID Properties found in $prop_names[0]\n";
            my $ss_prop_id = NetCDF::varid($file_id,$prop_names[0]);
        my ($varname,$data_type,$ndims,@dim_ids,$natts);
        my @dim_names = get_dim_names($file_id);
        my $error =
NetCDF::varinq($file_id,$ss_prop_id,$varname,$data_type,$ndims,\@dim_ids,$natts);
            die "ERROR: Problem inquiring variable\n" if ($error == -1);
            die "ERROR: Unexpected Number of Dimensions for sideset id properties\n" if ($ndims != 1);
            my $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
            my @start=(0);
        my @count=($dim1_size);
        $error = NetCDF::varget($file_id,$ss_prop_id,\@start,\@count,\@side_set_ids);
        die "ERROR: Problem getting variable values\n" if ($error == -1);
            my $surface_index = 0;
            foreach my $surface_id (@side_set_ids) {
                        my @elem_names = contain_atleast(\@ss_names,"elem");
                        my $elem_ss_var = $elem_names[$surface_index];
                        my $elem_ss_id = NetCDF::varid($file_id,$elem_ss_var);
                        print "Sideset Elements held in variable: $elem_ss_var\n";
                        $error =
NetCDF::varinq($file_id,$elem_ss_id,$varname,$data_type,$ndims,\@dim_ids,$natts);
            die "ERROR: Problem inquiring variable\n" if ($error == -1);
                        die "ERROR: Unexpected Number of Dimensions\n" if ($ndims != 1);
                        $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
                        @start=(0);
                        @count=($dim1_size);
```

```
                my @side_element=();
                $error = NetCDF::varget($file_id,$elem_ss_id,\@start,\@count,\@side_element);
                my @side_names = contain_atleast(\@ss_names,"side");
                my $side_ss_var = $side_names[$surface_index];
                my $side_ss_id = NetCDF::varid($file_id,$side_ss_var);
            print "Sideset Sides held in variable: $side_ss_var\n";
                $error =
NetCDF::varinq($file_id,$side_ss_id,$varname,$data_type,$ndims,\@dim_ids,$natts);
                die "ERROR: Problem inquiring variable\n" if ($error == -1);
                die "ERROR: Unexpected Number of Dimensions\n" if ($ndims != 1);
            $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
            @start=(0);
            @count=($dim1_size);
                my @side_face = ();
            $error = NetCDF::varget($file_id,$side_ss_id,\@start,\@count,\@side_face);
                die "ERROR: Number of Face does not match number of Elements.\n" if ($#side_face
!= $#side_element);
                my @sides = ();
                for(my $i = 0; $i <= $#side_face; ++$i) {
                        my $eid = $side_element[$i];
                        my $fid = $side_face[$i];
                        ##print "Converting face def: Element $eid, Face $fid\n";
                        my $connect_ref = $$elem_con_ref[$eid];
                        ##print join( ',', @$connect_ref );
                        ##print "\n";
                        my @facenodes = face_id_to_nodes($fid,$connect_ref);
                        ##print "Face Nodes: ";
                        ##print join( ',',@facenodes);
                        ##print "\n";
                        push(@sides,\@facenodes);
                }
                push(@side_faces,\@sides);
                ++$surface_index;
        }
        my @return_vals = (\@side_set_ids,\@side_faces);
        return @return_vals;
}

sub get_elem_block_id_map {
        ### Return Element Block ID Map
        ### returns $elem_ids[$num_elems]=$block_id;
        ### @elem_block_id_map = get_elem_block_id_map($file_id);
        my $file_id = $_[0];
        my @elem_block_id_map = ();
    my $error;
    my $num_elems=get_num_elems($file_id);
        my @dim_names = get_dim_names($file_id);
        my @var_names = get_var_names($file_id);
        my @connect_names = contain_atleast(\@var_names,"connect");
        my ($curname,$data_type,$ndims,@dim_ids,$natts);
        my @block_ids = ();
    my @var_names = get_var_names($file_id);
    my @eb_names = contain_atleast(\@var_names,"eb");
    my @prop_names = contain_atleast(\@eb_names,"prop");
    die "ERROR: Can not find block id properties.\n" if ($#prop_names+1 != 1);
    print "Block ID Properties found in $prop_names[0]\n";
```

```
	my $eb_prop_id = NetCDF::varid($file_id,$prop_names[0]);
	my ($varname,$data_type,$ndims,@dim_ids,$natts);
	my @dim_names = get_dim_names($file_id);
	my $error =
NetCDF::varinq($file_id,$eb_prop_id,$varname,$data_type,$ndims,\@dim_ids,$natts);
	die "ERROR: Problem inquiring variable\n" if ($error == -1);
	die "ERROR: Unexpected Number of Dimensions for Block id properties\n" if ($ndims != 1);
	my $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
	my @start=(0);
	my @count=($dim1_size);
	$error = NetCDF::varget($file_id,$eb_prop_id,\@start,\@count,\@block_ids);
	    die "ERROR: Problem getting block IDS\n" if ($error == -1);
	    my $block_id_index = 0;
	    foreach my $connect_name (@connect_names) {
	        my $varid = NetCDF::varid($file_id,$connect_name);
	        die "ERROR: Problem getting variable id\n" if ($varid== -1);
	        $error =
NetCDF::varinq($file_id,$varid,$curname,$data_type,$ndims,\@dim_ids,$natts);
	      die "ERROR: Problem inquiring for variable names\n" if ($error == -1);
	        die "ERROR: Number of dimensions used for defining connectivity unexpected.\n" if
($ndims != 2);
	        my $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
	        my $dim2_size = get_dim_size($file_id,$dim_names[$dim_ids[1]]);
	        for (my $i = 0; $i < $dim1_size; ++$i) {
	            push(@elem_block_id_map,$block_ids[$block_id_index]);
	        }
	        ++$block_id_index
	    }
	    die "ERROR: Element Block ID Map length not equal to number of elems.\n" if ($num_elems
!= $#elem_block_id_map+1);
	    unshift(@elem_block_id_map,0);
	    return @elem_block_id_map;
}


sub get_elem_varibles {
	### Get Array with Element Values of a Variable at Time Step
	### returns values[$num_elems]=$elem_value
	### @elem_values = get_elem_varibles($file_id,$time_step,$var_name);
	my $file_id = $_[0];
	my $time_step = $_[1];
	my $var_name = $_[2];
	my @values = ();
	my $error;
	my $num_elems=get_num_elems($file_id);

	my $varid = NetCDF::varid($file_id,"name_elem_var");
	my $elem_var_num = check_names($file_id,$varid,$var_name);
	die "ERROR: Could not find $var_name in element varibales\n" if ($elem_var_num == -1);
	    ++$elem_var_num;
	my @dim_names = get_dim_names($file_id);
	my @var_names = get_var_names($file_id);
	    my $varstor_name = "vals_elem_var".$elem_var_num;
	my @ebvar_names = contain_atleast(\@var_names,$varstor_name);
	my ($curname,$data_type,$ndims,@dim_ids,$natts);
	foreach my $ebvar_name (@ebvar_names) {
```

```
		my $varid = NetCDF::varid($file_id,$ebvar_name);
		die "ERROR: Problem getting variable id\n" if ($varid== -1);
		$error = NetCDF::varinq($file_id,$varid,$curname,$data_type,$ndims,\@dim_ids,$natts);
		die "ERROR: Problem inquiring for variable names\n" if ($error == -1);
		die "ERROR: Number of dimensions used for defining element variable unexpected.\n" if
($ndims != 2);
		print "$ebvar_name ( $dim_names[$dim_ids[0]] , $dim_names[$dim_ids[1]] ) \n";
			if ($dim_names[$dim_ids[0]] =~ m/^time_step$/) {
				## Do Nothing
			} else {
				die "ERROR: Dimension 1 not = time_step as expected.\n" if ( 1 == 1);
			}
		my $dim1_size = get_dim_size($file_id,$dim_names[$dim_ids[0]]);
		my $dim2_size = get_dim_size($file_id,$dim_names[$dim_ids[1]]);
		my @start = ($time_step,0);
		my @count = (1,$dim2_size);
			my @cur_vals=();
		$error = NetCDF::varget($file_id,$varid,\@start,\@count,\@cur_vals);
			die "ERROR: Problem getting element values\n" if ($error == -1);
			push(@values,@cur_vals);
	}
	die "ERROR: Element Variable lenngth not equal to number of elems.\n" if ($num_elems !=
$#values+1);
	unshift(@values,0);
	return @values;
}

sub make_node_to_elem_map {
	### Makes Node to Element Map
	### returns $node_elem_map[$num_nodes] = \@elem_ids
	    ### where @elem_ids contains the internal element ids of the elements
	    ### containing the internal node id.
	### @node_elem_map = make_node_to_elem_map(\@connectivity,$num_elems,$num_nodes);
	    my $con_ref = $_[0];
	    my $num_elems = $_[1];
	    my $num_nodes = $_[2];
	    my @connectivity = @$con_ref;
	    my @node_elem_map = ();
	    for (my $i=1;$i<=$num_elems;++$i) {
			my $elem_nd_ids_ref = $connectivity[$i];
			my @elem_nd_ids = @$elem_nd_ids_ref;
			foreach my $nd_id (@elem_nd_ids) {
				if (defined($node_elem_map[$nd_id])) {
					my $elem_ids_ref = $node_elem_map[$nd_id];
					my @elem_ids = @$elem_ids_ref;
					push(@elem_ids,$i);
					$node_elem_map[$nd_id]=\@elem_ids;
				} else {
					my @elem_ids = ($i);
					$node_elem_map[$nd_id]=\@elem_ids;
				}
			}
	    }
	    ##my $node1elemmap_ref = $node_elem_map[1];
	    ##print "Internal Node Id 1 is part of internal element Ids:\n";
	    ##foreach my $eid (@$node1elemmap_ref) {
```

```
        ##      print "$eid, "
        ##}
        ##print "\n";
        return @node_elem_map;
}

sub elem_var_to_node_var {
        ### Convert Element Variable to Nodal Varaible by Weighted Average
        ### returns @nodal_values
        ### @nodal_values =
elem_var_to_node_var(\@node_elem_map,$num_nodes,\@elem_values,\@elem_weights);
        ###
        my $node_elem_map_ref = $_[0];
        my @node_elem_map = @$node_elem_map_ref;
        my $num_nodes = $_[1];
        my $elem_values_ref = $_[2];
        my @elem_values = @$elem_values_ref;
        my $elem_weights_ref = $_[3];
        my @elem_weights = @$elem_weights_ref;
        my @nodal_values = ();
        for( my $i=1;$i<=$num_nodes;++$i) {
                ##print "Getting Nodal Value for Node $i\n";
                my $eles_ref = $node_elem_map[$i];
                my @eles = @$eles_ref;
                my $weight_sum = 0;
                my $cur_value = 0;
                foreach my $elem (@eles) {
                        ##print "Element $elem 's weight = $elem_weights[$elem]\n";
                        ##print "Element $elem 's value = $elem_values[$elem]\n";
                        $weight_sum = $weight_sum + $elem_weights[$elem];
                        $cur_value = $cur_value + $elem_values[$elem]*$elem_weights[$elem];
                }
                ##print "Summed Weights = $weight_sum\n";
                $cur_value = $cur_value/$weight_sum;
                ##print "Value = $cur_value\n";
                $nodal_values[$i]=$cur_value;
        }
        return @nodal_values;
}

sub elem_var_to_node_var_distance_weighted {
      ### Convert Element Variable to Nodal Varaible by Inverse of Center Distance Weights
      ### returns @nodal_values
      ### @nodal_values =
elem_var_to_node_var_distance_weighted(\@node_elem_map,\@elem_values,\@nodal_coords,\@el
em_centers);
      ###
      my $node_elem_map_ref = $_[0];
      my @node_elem_map = @$node_elem_map_ref;
      my $elem_values_ref = $_[1];
      my @elem_values = @$elem_values_ref;
      my $nodal_coords_ref = $_[2];
      my @nodal_coords = @$nodal_coords_ref;
          my $elem_centers_ref = $_[3];
          my @elem_centers = @$elem_centers_ref;
      my @nodal_values = ();
```

```
	for( my $i=1;$i<@nodal_coords;++$i) {
		my $eles_ref = $node_elem_map[$i];
		my @eles = @$eles_ref;
			my $node_point_ref = $nodal_coords[$i];
		my $weight_sum = 0;
		my $cur_value = 0;
		foreach my $elem (@eles) {
				my $elem_cen_ref = $elem_centers[$elem];
				my $dist = point_to_point_distance($node_point_ref,$elem_cen_ref);
			$weight_sum = $weight_sum + (1/$dist);
			$cur_value = $cur_value + $elem_values[$elem]*(1/$dist);
		}
		$cur_value = $cur_value/$weight_sum;
		$nodal_values[$i]=$cur_value;
	}
	return @nodal_values;

}

sub calculate_elem_centers {
		### Calculate and Retrun the Element Centers
		### @elem_centers = calculate_elem_centers(\@elem_conn,\@nodal_coords);
		###
		my $elem_conn_ref = $_[0];
		my $nodal_coords_ref = $_[1];
		my @elem_conn=@$elem_conn_ref;
	my @nodal_coords=@$nodal_coords_ref;
		my $num = @elem_conn;
		$num = $num -1;
		print "Calculating $num centers\n";
		my @elem_centers = ();
		for (my $i=1;$i<=$num;++$i) {
			my @center = (0,0,0);
			my $conn_ref=$elem_conn[$i];
			my @nod_ids=@$conn_ref;
			my $num_nods = @nod_ids;
			foreach my $nod (@nod_ids) {
				my $point_ref = $nodal_coords[$nod];
			my @point = @$point_ref;
				@center = addVEC(\@point,\@center);
			}
			@center = scaleVEC(\@center,(1/$num_nods));
			$elem_centers[$i]=\@center;
		}
		return @elem_centers;
}

sub calculate_elem_max_diagonal {
	### Calculate and Retrun the Elements Max Diagonal
	### @elem_diags = calculate_elem_max_diagonal(\@elem_conn,\@nodal_coords);
	###
	my $elem_conn_ref = $_[0];
	my $nodal_coords_ref = $_[1];
	my @elem_conn=@$elem_conn_ref;
	my @nodal_coords=@$nodal_coords_ref;
	my $num = @elem_conn;
```

```
	$num = $num -1;
	print "Calculating $num Max Diagonals\n";
	my @elem_diags = ();
	for (my $i=1;$i<=$num;++$i) {
		my $conn_ref=$elem_conn[$i];
		my @nod_ids=@$conn_ref;
			my @diags = (0,0,0,0);
			$diags[0] =
point_to_point_SQdistance($nodal_coords[$nod_ids[0]],$nodal_coords[$nod_ids[6]]);
			$diags[1] =
point_to_point_SQdistance($nodal_coords[$nod_ids[1]],$nodal_coords[$nod_ids[7]]);
			$diags[2] =
point_to_point_SQdistance($nodal_coords[$nod_ids[2]],$nodal_coords[$nod_ids[4]]);
			$diags[3] =
point_to_point_SQdistance($nodal_coords[$nod_ids[3]],$nodal_coords[$nod_ids[5]]);
			my $max_sq_diag = maxVAL(\@diags);
			$elem_diags[$i] = sqrt($max_sq_diag);

	}
	return @elem_diags;
}

sub find_natural_coords {
		### Find Natural Coordinates of x,y,z location inside element
		### returns @nc and an $error_code
		### $error_code = -2 -> Not in Natural Domain and Iteration Limit Reached
		### $error_code = -1 -> Not in Natural Domain
		### $error_code = 0 -> Successful
		### $error_code = 1 -> Iteration Limit Reached
		### (\@nc,$error_code) =
find_natural_coords(\@xyz,$elemid,\@elem_conn,\@nodal_coords);
		###
		my($xyz_ref,$elemid,$elem_conn_ref,$nodal_coords_ref)=@_;
		my @xyz=@$xyz_ref;
		my @elem_conn=@$elem_conn_ref;
		my @nodal_coords=@$nodal_coords_ref;
		my $conn_ref=$elem_conn[$elemid];
		my @nod_ids=@$conn_ref;
		die "Can only use hex elements when getting Natural Coordinates.\n" if ( 8 != @nod_ids);
		my @x=();
		my @y=();
		my @z=();
		for (my $i = 0; $i < 8; ++$i) {
			my $point_ref = $nodal_coords[$nod_ids[$i]];
			my @point = @$point_ref;
			$x[$i] = $point[0];
			$y[$i] = $point[1];
			$z[$i] = $point[2];
		}
	##print "Element Coordinates:\n";
	##print "x = ".join(', ',@x);
	##print "\n";
	##print "y = ".join(', ',@y);
	##print "\n";
	##print "z = ".join(', ',@z);
	##print "\n";
```

```
	my @alpha = (-1,1,1,-1,-1,1,1,-1);
	my @beta =  (-1,-1,1,1,-1,-1,1,1);
	my @gamma = (-1,-1,-1,-1,1,1,1,1);
	####  Solve Loop
	my $error_tol = 1.0e-16;
	my $error = 1;
	my @nc0 = (0,0,0);
	my $iter = 0;
	while ($error >= $error_tol) {
		my @f0 = @xyz;
		my $addtoX = 0;
		my $addtoY = 0;
		my $addtoZ = 0;
		for (my $i = 0; $i < 8; ++$i) {
			$addtoX = $addtoX
+(1+$alpha[$i]*$nc0[0])*(1+$beta[$i]*$nc0[1])*(1+$gamma[$i]*$nc0[2])*$x[$i];
			$addtoY = $addtoY
+(1+$alpha[$i]*$nc0[0])*(1+$beta[$i]*$nc0[1])*(1+$gamma[$i]*$nc0[2])*$y[$i];
			$addtoZ = $addtoZ
+(1+$alpha[$i]*$nc0[0])*(1+$beta[$i]*$nc0[1])*(1+$gamma[$i]*$nc0[2])*$z[$i];
		}
		$f0[0]=$f0[0]-0.125*$addtoX;
		$f0[1]=$f0[1]-0.125*$addtoY;
		$f0[2]=$f0[2]-0.125*$addtoZ;
		##print "f0 = ";
		##print join(', ',@f0);
		##print "\n";
		$error = maxABS(\@f0);
		##print "Max Error = $error \n";
		### Calculate Jacobian Elements
		my ($j1,$j2,$j3,$j4,$j5,$j6,$j7,$j8,$j9) = (0,0,0,0,0,0,0,0,0);
		for (my $i = 0; $i < 8; ++$i) {
			$j1 = $j1 -
(1/8)*$x[$i]*$alpha[$i]*(1+$gamma[$i]*$nc0[2])*(1+$beta[$i]*$nc0[1]);
			$j2 = $j2 -
(1/8)*$x[$i]*$beta[$i]*(1+$gamma[$i]*$nc0[2])*(1+$alpha[$i]*$nc0[0]);
			$j3 = $j3 -
(1/8)*$x[$i]*$gamma[$i]*(1+$beta[$i]*$nc0[1])*(1+$alpha[$i]*$nc0[0]);
		$j4 = $j4 - (1/8)*$y[$i]*$alpha[$i]*(1+$gamma[$i]*$nc0[2])*(1+$beta[$i]*$nc0[1]);
		$j5 = $j5 - (1/8)*$y[$i]*$beta[$i]*(1+$gamma[$i]*$nc0[2])*(1+$alpha[$i]*$nc0[0]);
		$j6 = $j6 - (1/8)*$y[$i]*$gamma[$i]*(1+$beta[$i]*$nc0[1])*(1+$alpha[$i]*$nc0[0]);
		$j7 = $j7 - (1/8)*$z[$i]*$alpha[$i]*(1+$gamma[$i]*$nc0[2])*(1+$beta[$i]*$nc0[1]);
		$j8 = $j8 - (1/8)*$z[$i]*$beta[$i]*(1+$gamma[$i]*$nc0[2])*(1+$alpha[$i]*$nc0[0]);
		$j9 = $j9 - (1/8)*$z[$i]*$gamma[$i]*(1+$beta[$i]*$nc0[1])*(1+$alpha[$i]*$nc0[0]);
		 }
		 ##print "Jacobian Elements = ";
		 ##print "$j1,$j2,$j3,$j4,$j5,$j6,$j7,$j8,$j9\n";
		 ### Jacobian Determinant
		 my $jdet = -$j3*$j5*$j7+$j2*$j6*$j7+$j3*$j4*$j8-$j1*$j6*$j8-$j2*$j4*$j9+$j1*$j5*$j9;
		 ##print "Jacobian Determinant = $jdet\n";
		 ### Inverse Jacobian
		 my ($jI1,$jI2,$jI3,$jI4,$jI5,$jI6,$jI7,$jI8,$jI9) = (0,0,0,0,0,0,0,0,0);
		 $jI1 = (1/$jdet)*(-$j6*$j8+$j5*$j9);
		 $jI2 = (1/$jdet)*($j3*$j8-$j2*$j9);
		 $jI3 = (1/$jdet)*(-$j3*$j5+$j2*$j6);
		 $jI4 = (1/$jdet)*($j6*$j7-$j4*$j9);
```

```
                $jI5 = (1/$jdet)*(-$j3*$j7+$j1*$j9);
                $jI6 = (1/$jdet)*($j3*$j4-$j1*$j6);
                $jI7 = (1/$jdet)*(-$j5*$j7+$j4*$j8);
                $jI8 = (1/$jdet)*($j2*$j7-$j1*$j8);
                $jI9 = (1/$jdet)*(-$j2*$j4+$j1*$j5);
            ##print "Inverse Jacobian Elements = ";
                ##print "$jI1,$jI2,$jI3,$jI4,$jI5,$jI6,$jI7,$jI8,$jI9\n";
                ### Adjust @nc0
                my @nc_del = ();
                $nc_del[0] = -1*($jI1*$f0[0]+$jI2*$f0[1]+$jI3*$f0[2]);
                $nc_del[1] = -1*($jI4*$f0[0]+$jI5*$f0[1]+$jI6*$f0[2]);
                $nc_del[2] = -1*($jI7*$f0[0]+$jI8*$f0[1]+$jI9*$f0[2]);
                ##print "Natural Coordinates Adjustment = ";
            ##print join(', ',@nc_del);
            ##print "\n";
                @nc0 = addVEC(\@nc_del,\@nc0);
                ##print "Natural Coordinates = ";
            ##print join(', ',@nc0);
            ##print "\n";
                ++$iter;
                if ($iter > 50) {
                        $error = maxABS(\@nc0);
                        if ($error > 1) {
                                return (\@nc0,-2);
                        } else {
                                return (\@nc0,1);
                        }
                }
        }
        $error = maxABS(\@nc0);
        if ($error > 1) {
                return(\@nc0,-1);
        }
        return (\@nc0,0);
        ####
}

sub value_at_natural_coords {
    ### Return Value at Natural Coordinates inside element
    ### returns $val
    ### $val = value_at_natural_coords(\@nc,$elemid,\@elem_conn,\@nodal_values);
    ###
    my($nc_ref,$elemid,$elem_conn_ref,$nodal_values_ref)=@_;
    my @nc=@$nc_ref;
    my @elem_conn=@$elem_conn_ref;
    my @nodal_values=@$nodal_values_ref;
    my $conn_ref=$elem_conn[$elemid];
    my @nod_ids=@$conn_ref;
    die "Can only use hex elements when getting Natural Coordinates.\n" if ( 8 != @nod_ids);
    my @alpha = (-1,1,1,-1,-1,1,1,-1);
    my @beta =  (-1,-1,1,1,-1,-1,1,1);
    my @gamma = (-1,-1,-1,-1,1,1,1,1);
        my $val = 0;
    for (my $i = 0; $i < 8; ++$i) {
                $val = $val +(1+$alpha[$i]*$nc[0])*(1+$beta[$i]*$nc[1])*
                        (1+$gamma[$i]*$nc[2])*$nodal_values[$nod_ids[$i]];
```

```
        }
        $val = ($val/8);
        return $val;
}

sub place_in_bins_by_ID {
        ### Bin IDs in x,y,z bins
        ### returns Bins with Ids in them and Bin Information
        ###
($xbins,$ybins,$zbins,\@bin_centers,\@bin_xyz_maximums,\@bin_xyx_minimums,\@bins) =
        ### place_in_bins_by_ID(\@centers,\@diameters,$mult);
        ###
        my $centers_ref = $_[0];
        my $diameters_ref = $_[1];
        my $mult = $_[2];
        my @centers = @$centers_ref;
        my @diameters = @$diameters_ref;
        my ($xbins,$ybins,$zbins,@bin_centers,@bin_xyz_maximums,@bin_xyx_minimums,@bins);
        my $num_c = @centers;
        my $num_d = @diameters;
        die "ERROR: Number of Centers not equal to number of Diameters\n" if ($num_c != $num_d);

        ## Seperate Centers
        my @x = ();
        my @y = ();
        my @z = ();
        for (my $i=1;$i<$num_c;++$i) {
                my $point_ref = $centers[$i];
                my @point = @$point_ref;
                $x[$i]=$point[0];
                $y[$i]=$point[1];
                $z[$i]=$point[2];
        }
        ## Determine Bin Domain
        my $pad = 1.2;
        my $max_diameter = maxVAL(\@diameters);
        my $max_x =  maxVAL(\@x);
        $max_x = $max_x + $pad*$max_diameter;
        my $max_y =  maxVAL(\@y);
    $max_y = $max_y + $pad*$max_diameter;
        my $max_z =  maxVAL(\@z);
    $max_z = $max_z + $pad*$max_diameter;
        my $min_x =  minVAL(\@x);
    $min_x = $min_x - $pad*$max_diameter;
    my $min_y =  minVAL(\@y);
    $min_y = $min_y - $pad*$max_diameter;
    my $min_z =  minVAL(\@z);
    $min_z = $min_z - $pad*$max_diameter;
        print "Bin Domain:\nMin X,Y,Z = $min_x, $min_y, $min_z\n";
        print "Max X,Y,Z = $max_x, $max_y, $max_z\n";
        ## Set Spacing
        my $spacing = $mult*$max_diameter;
        ## Get Bin Dimension Info
        ## X
        my @xmin_bin_info= ();
        my @xmax_bin_info= ();
```

```
    my @xcenter_bin_info= ();
    my $xmin_bin = $min_x;
    my $xmax_bin = $min_x + $spacing;
    while ($xmin_bin < $max_x) {
            push(@xmin_bin_info,$xmin_bin);
            push(@xmax_bin_info,$xmax_bin);
            push(@xcenter_bin_info,($xmin_bin+($spacing/2)));
            $xmin_bin = $xmin_bin + $spacing;
            $xmax_bin = $xmin_bin + $spacing;
    }
    $xbins = @xmax_bin_info;
    ## Y
my @ymin_bin_info= ();
my @ymax_bin_info= ();
my @ycenter_bin_info= ();
my $ymin_bin = $min_y;
my $ymax_bin = $min_y + $spacing;
while ($ymin_bin < $max_y) {
      push(@ymin_bin_info,$ymin_bin);
      push(@ymax_bin_info,$ymax_bin);
      push(@ycenter_bin_info,($ymin_bin+($spacing/2)));
      $ymin_bin = $ymin_bin + $spacing;
      $ymax_bin = $ymin_bin + $spacing;
}
$ybins = @ymax_bin_info;
    ## Z
my @zmin_bin_info= ();
my @zmax_bin_info= ();
my @zcenter_bin_info= ();
my $zmin_bin = $min_z;
my $zmax_bin = $min_z + $spacing;
while ($zmin_bin < $max_z) {
      push(@zmin_bin_info,$zmin_bin);
      push(@zmax_bin_info,$zmax_bin);
      push(@zcenter_bin_info,($zmin_bin+($spacing/2)));
      $zmin_bin = $zmin_bin + $spacing;
      $zmax_bin = $zmin_bin + $spacing;
}
$zbins = @zmax_bin_info;
    print "$xbins X bins, $ybins Y bins, $zbins z bins\n";
    ## Slice with X
    print "Sorting in X\n";
    my @xslice = ();
    for (my $i=0; $i<$xbins; ++$i) {
            my @xIDS = ();
          for (my $p=1;$p<$num_c;++$p) {
                  my $rad = $diameters[$p]/2;
                  my $pmax = $x[$p] + $rad;
                  my $pmin = $x[$p] - $rad;
                  if ($pmax <= $xmax_bin_info[$i] && $pmin >= $xmin_bin_info[$i]) {
                          push(@xIDS,$p);
                  }
            }
            push(@xslice,\@xIDS);
    }
    ## Slice with Y
```

```
	print "Sorting in Y\n";
	my @yslice = ();
	for (my $i=0; $i<$xbins; ++$i) {
		my $IDS_ref = $xslice[$i];
		my @IDS = @$IDS_ref;
		my @yIDS = ();
		for (my $j=0; $j<$ybins; ++$j) {
			foreach my $p (@IDS) {
				my $rad = $diameters[$p]/2;
		my $pmax = $y[$p] + $rad;
		my $pmin = $y[$p] - $rad;
		if ($pmax <= $ymax_bin_info[$i] && $pmin >= $ymin_bin_info[$i]) {
					push(@yIDS,$p);
				}
			}
			$yslice[$i][$j] = \@yIDS;
		}
	}
	## Slice with Z
	print "Sorting in Z\n";
	for (my $i=0; $i<$xbins; ++$i) {
		for (my $j=0; $j<$ybins; ++$j) {
			my $IDS_ref = $yslice[$i][$j];
			my @IDS = @$IDS_ref;
			my @zIDS = ();
			for (my $k=0; $k<$zbins; ++$k) {
				foreach my $p (@IDS) {
					my $rad = $diameters[$p]/2;
			my $pmax = $z[$p] + $rad;
			my $pmin = $z[$p] - $rad;
			if ($pmax <= $zmax_bin_info[$i] && $pmin >= $zmin_bin_info[$i]) {
				push(@zIDS,$p);
			}
		}
				my @bincenter = ($xmin_bin_info[$i]+$spacing,
					$ymin_bin_info[$j]+$spacing,$zmin_bin_info[$k]+$spacing);
				my @bin_maxs =
($xmax_bin_info[$i],$ymax_bin_info[$j],$zmax_bin_info[$k]);
				my @bin_mins =
($xmin_bin_info[$i],$ymin_bin_info[$j],$zmin_bin_info[$k]);
				$bins[$i][$j][$k] = \@zIDS;
				$bin_centers[$i][$j][$k] = \@bincenter;
				$bin_xyz_maximums[$i][$j][$k] = \@bin_maxs;
				$bin_xyx_minimums[$i][$j][$k] = \@bin_mins;
			}
		}
	}
	## Return Results
	return
($xbins,$ybins,$zbins,\@bin_centers,\@bin_xyz_maximums,\@bin_xyx_minimums,\@bins);
}


1;
```

References

1. **J.R. Rice, *A Path Independent Integral and Approximate Analysis of Strain Concentration by Notches and Cracks.* Journal of Applied Mechanics, 1968. 35(2): p. 379.**

2. **J.D. Eshelby, *The Force on an Elastic Singularity.* Philosophical Transactions of the Royal Society of London, Series A: Mathematical and Physical Sciences, 1951. 244(877): p. 81-112.**

3. **B.N. Rao and S. Rahman, *An enriched meshless method for non-linear fracture mechanics.* International Journal for Numerical Methods in Engineering, 2004. 59: p. 197-223.**

4. **W.S. Blackburn, *Path independent integrals to predict the onset of crack instability in an elastic plastic material.* International Journal of Fracture Mechanics, 1972. 8: p. 343-346.**

5. **K. Kishimoto, S. Aoki, and M. Sakata, *On the Path Independent Integral - \hat{J}.* Engineering Fracture Mechanics, 1980. 13: p. 841-850.**

6. **M. Amestoy, H.D. Bui, and R. Labbens, *On the definition of local path independent integrals in three-dimensional crack problems.* Mechanics Research Communications, 1981. 8(4): p. 231-236.**

7. **A.D. Batte, W.S. Blackburn, A. Elsender, T.K. Hellen, and A.D. Jackson, *A comparison of the J* integral with other methods of post yield fracture mechanics.* International Journal of Fracture, 1983. 21: p. 49-66.**

8. **F.Z. Li, C.F. Shih, and A. Needleman, Engineering Fracture Mechanics, 1985. 21: p. 405-421.**

9. **C.F. Shih, B. Moran, and T. Nakamura, *Energy release rate along a three-dimensional crack front in a thermally stressed body.* International Journal of Fracture, 1986. 30: p. 79-102.**

10. **M. Chiarelli and A. Frediani, *A Computation of the Three-Dimensional J-Integral for Elastic Materials with a View to Applications in Fracture Mechanics.* Engineering Fracture Mechanics, 1993. 44(5): p. 763-788.**

11. **W.A. Meith and M.R. Hill, *Domain-independent values of the J-integral for cracks in three-dimensional residual stress bearing bodies.* Engineering Fracture Mechanics, 2002. 69: p. 1301-1314.**

12. **K. Eriksson, *A domain independent integral expression for the crack extension force of a curved crack in three dimensions.* Journal of the Mechanics and Physics of Solids, 2002. 50: p. 381-403.**

13. **T.D. Nguyen, S. Govindjee, P.A. Klein, and H. Gao, *A material force method for inelastic fracture mechanics.* Journal of the Mechanics and Physics of Solids, 2005. 53: p. 91-121.**

14. **G.W. Wellman, *J2D and J3D – Post Processing Codes to Calculate the J-Integral in Two and Three Dimensions*, in *SANDIA Technical Report*. 1991, Sandia National Laboratories.**

15. **T.L. Anderson, *Fracture Mechanics: Fundamentals and Applications*. 2nd edition ed. 1995, Boca Raton: CRC Press, Inc.**

16. **J.G. Mercer and T. Nicholas, *Growth of short cracks in a notch plastic zone.* International Journal of Fatigue, 1991. 13(3): p. 263-270.**

17. **N.P. O'Dowd and C.F. Shih, *Family of Crack-Tip Fields Characterized by a Triaxiality Parameter--I. Structure of Fields.* Journal of the Mechanics and Physics of Solids, 1991. 39(8): p. 989-1015.**

18. **N.P. O'Dowd and C.F. Shih, *Family of Crack-Tip Fields Characterized by a Triaxiality Parameter--II. Fracture Applications* Journal of the Mechanics and Physics of Solids, 1992. 40(5): p. 939-963.**

19. **R.A. Ainsworth and N.P. O'Dowd, *Constraint in the failure assessment diagram approach for fracture assessment.* Journal of Pressure Vessel Technology, 1995. 117(3): p. 260-267.**

20. **Y.G. Matvienko and E.M. Morozov, *Calculation of the energy J-integral for bodies with notches and cracks* International Journal of Fracture, 2004. 125(3-4): p. 249-261.**

21. **B.R. Antoun, K. Connelly, S. Hong, E.M. Huestis, J.A. Zimmerman, K.A. Nibur, B.P. Somerday, A.A. Brown, A.J. Lindblad, and Y.O. Ohashi, *Experimental Characterization and Validation of the J-Integral Method for GTS Reservoir Materials*. 2008, Sandia National Laboratories.**

22. **ASTM, *Standard Test Method for Measurement of Fracture Toughness*. 2007, ASTM International.**

23. **D.J. Bammann, M.L. Chiesa, and G.C. Johnson, *A Strain Rate Dependent Flow Surface Mode of Plasticity*. 1990, Sandia National Laboratories.**

24. **D.J. Bammann, M.L. Chiesa, and G.C. Johnson, *Modeling large deformation and failure in manufacturing processes.* Theoretical and Applied Mechanics, 1997. 19: p. 359-378.**

25. **E.B. Marin, D.J. Bammann, R.A. Regueiro, and G.C. Johnson, *On the Formulation, Parameter Identification and Numerical Integration of the***

***EMMI Model: Plasticity and Isotropic Damage*, in *SANDIA Technical Report*. 2006, Sandia National Laboratories.**

26. J.F. Lathrop, *BFIT - A Program to Analyze and Fit the BCJ Model Parameters to Experimental Data*, in *SANDIA Technical Report*. 1996, Sandia National Laboratories.

27. G.D. Wyss and K.H. Jorgensen, *A User s Guide to LHS: Sandia's Latin Hypercube Sampling Software*. 1998, Sandia National Laboratories: Albuquerque.

28. DAKOTA. 2008 [cited; Available from: http://www.cs.sandia.gov/DAKOTA/software.html.

Distribution

Internal:

1	MS 9035	Paul Spence, Org 8224
1	MS 9035	Dorian Balch, Org 8224
1	MS 9035	Steven Rice, Org 8224
1	MS 9042	Davina Kwon, Org 8220
1	MS 9042	Er-Ping Chen, 8220
1	MS 9042	Michael Chiesa, 8246
1	MS 9042	Bonnie Antoun, 8246
1	MS 9042	James Foulk, 8246
1	MS 9042	Carol Le Gall, 8249
1	MS 9042	Arthur Brown, 8249
1	MS 9042	Alex Lindblad, 8249
1	MS 9042	Yuki Ohashi, 8249
1	MS 9152	Michael Hardwick, 8964
1	MS 9153	Russ Miller, 8200
1	MS 9159	Stephen Margolis, 8964
1	MS 9404	Tom Felter, 8222
1	MS 9404	Kevin Nibur, 8222
1	MS 9404	Brian Somerday, 8222
1	MS 9404	Jonathan Zimmerman, 8246
1	MS 9409	Kevin Connelly, 8246
1	MS 0899	Technical Library, 9616
3	MS 9018	Central Technical Files, 8945-1
1	MS 9021	Classification Office, 8511 for Technical Library, MS 0899, 9616
	MS 9021	Classification Office, 8511 for DOE/OSTI via URL