MICROMECHANICS BASED MODELING OF THE DEFORMATION AND FAILURE OF AL 5083 WELDED PLATE WITH MINIMAL FITTING PARAMETERS

A Thesis
Presented to the Faculty of the Graduate School
Of Cornell University
In Partial Fulfillment of the Requirements for the Degree of
Master of Science

by
Ranny Ran Zhao
August 2016

ABSTRACT

A mechanistic model that does not depend on free fitting parameters was examined for prediction of ductile failure of Al 5083 welded plates against experimental tests provided by the U.S. Navy. The welded plate has three distinct material sections: BASE, HAZ, and WELD. The model studied here takes only three inputs: (1) the large deformation constitutive response of the matrix, (2) the void volume fraction represented by voids nucleated at second phase particle sties and voids from weld porosity,, and (3) the average nearest neighbor spacing of second phase particles and voids. The micromechanical model representative volume element (RVE) consists of a rectangular cell matrix with a spherical void at the center. The cell model of each material was assumed an isotropic elastic plastic matrix material response and was simulated under a common range of stress states. The responses of the micromechanical models were then homogenized and fed into the macromechanical models to predict the total ductile failure of the welded plate using Abaqus /Explicit finite element engineering application. Owing to its detailed comprehension, the model has robust capabilities predicting ductile failure of a multi-material laboratory specimen to within 15.27%. This validation is remarkable considering the transferability of a simple mechanistic model to different structural metals.

BIOGRAPHICAL SKETCH

Ranny received her undergraduate degree in Civil and Environmental Engineering from Carnegie Mellon University in May of 2013. Her study included a breath of the field with an interest in structural engineering. So she then pursued a graduate degree at Cornell University in Civil and Environmental Engineering department, concentrating in Structural Engineering with a minor in Structural Mechanics. Her work focused on Computational Mechanics at the micro- and continuum scale in the Fracture Research Group under the advisory guidance of Prof. Derek Warner. At Cornell University, Ranny have taken courses in material mechanics, linear and non-linear finite element method, material science, and numerical computation. Ranny also has an interest in entrepreneurship in advanced engineering technologies.

ACKNOWLEDGEMENT

I acknowledge my parents, Mark Zhao and Macy Ma, my boyfriend, Michael Meller, and my supportive friends, Ben Hasseldine and Pankaj Singh, for their encouragement that helped me get through the challenging times at Cornell and for helping me in rediscovering myself.

I also want to acknowledge my adviser, Prof. Warner, my committee member, Prof. McLaskey, and my group mates, Geoff Bomarito, Sepehr Saroukhani, Peipei Li, and Kelvin Leung, for their academic and professional advices during my work at Cornell.

TABLE OF CONTENTS

BIOGRAPHICAL SKETCH iii
ACKNOWLEDGEMENT iv
1. Introduction 1
2. Model assumptions 3
3. Micromechanical Simulation Descriptions 9
4. Micromechanical Simulation Results 17
5. Macromechanical Simulation Descriptions 25
6. Macromechanical Simulation Results 31
7. Conclusion 59
References 61
Appendix A 65
Appendix B 91
Appendix C 97

1. Introduction

Ductile failure, a common failure initiation mechanism in metallic structures, is controlled by the evolution of microscale voids within the metal. Multiscale models for ductile failure study have long been used by physical metallurgists since the 1960s when safe laboratory tests cannot be insured and for the development of new materials [Pardoen et al 2010, Puttick, 1959; Rice and Tacey, 1969]. The prediction of ductile failure with mechanistic based models, as opposed to parametrically fit phenomological models, is still continuously pursued today under the motivations for more robust prediction capabilities and easy transferability.

Most of the purely mechanistic based ductile failure models are founded on a porous plasticity model originally developed by Gurson [1977] and later refined by Tvergaard and Needleman [1984]; referred to as the standard Gurson-Tvergaard-Needleman (stdGTN) model in this work. Various modifications of this stdGTN model have been developed and studied by the material mechanism research community ever since [Becker et al, 1988; Zhang et al, 2000, 2001; Barsoum et al, 2011; Nahshon and Jutchinson, 2008; Kiran and Khandelwal, 2014; Bomarito and Warner, 2015]. In a more recent publication, Bomarito and Warner [2016] developed a modified GTN model (modGTN) that can predict the ductile failure of a fully dense structural material (Al 5083-H116) with surprising accuracy (to within 25%). Their mechanistic based model was fitted to microstructural void cell simulations over a common range of stress states. The microstructural simulations relied on only three input parameters that are obtainable from simple laboratory procedures.

In this study, the mechanistic based modGTN model for stress state dependent ductile failure study in Bomarito and Warner [2016] was validated for its predictive capability and transferability on tensile deformation of welded plate specimen in Abaqus /Explicit. The welded plate has three different material sections: BASE, HAZ, and WELD. Based on simple assumptions, the modGTN model required only three inputs to fully characterize a material failure: (1) the large deformation constitutive response of the matrix, (2) the void volume fraction represented by voids nucleated at second phase particle sties and voids from weld porosity, , and (3) the average nearest neighbor spacing of second phase particles and voids. Section 3 details the modGTN model at two length scales along with the homogenization method used to bridge the scales. Section 4 describes the resulting micromechanical responses for each material used. Section 5 then investigates the setup in Abaqus for testing the modGTN in the specimen scale model by comparing to experimental data provided by the U.S. Navy [2009]. The code for this modGTN model was also made available for public use in Abaqus (see Appendix A). The model is found to predict the ductile failure of the welded plate to within 15.27%, confirming that it has robust prediction capability and is transferrable to different materials.

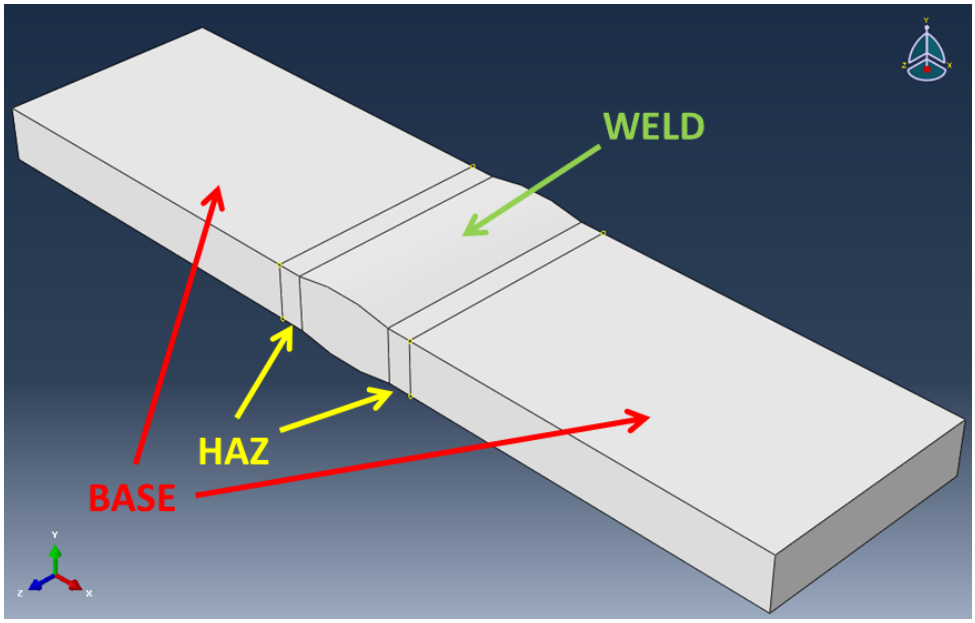
2. Model assumptions

In this multiscale model, a hierarchical approach is assumed. Hierarchical approach uses numerical techniques that run independently at separate length scales, bridging methodologies that connect the different length scales. The bridging methodologies use the characteristics found in the cause-effect relations at a lower scale to determine the pertinent effects for the next higher scale [Horstemeyer, 2009]. In the modGTN model, the result of the microstructure ductile failure responses are used as inputs for the plate models on the macroscale. Commonly, a model's internal state variables (ISV) are physically based on the microstructure properties, and must also exist and be utilized at the macroscale. The chosen ISV should capture the effects of a representative volume element (RVE) but not all of its complexities at the lower scale. In other words, the details of the microscopic arrangement would be averaged, or homogenized, at the macroscale by user defined ISV [Horstemeyer, 2009]. The RVE constructed in the microscale here is a rectangular prism cell with exactly one spherical void in its center, then a periodic distribution of this RVE is assume on the macroscale [Bomarito and Warner, 2015 and 2016] (details in Section 3). A spherical center void was chosen due to regularities in the mechanisms of void nucleation, growth, and coalescence in the microstructure of metallic alloys [Gurson, 1977; Tvergaard and Needleman, 1984; Pardoen et al, 2010].

At the macroscale, a specimen model of a welded plate was created assuming uniform material sections. Simulating welding is very complicated and literatures have shown many methods and approaches for it [Lindgren, 2001; Francis, 2002]. The focus of this paper is not to simulate thermal effects from welding process; but rather to examine the robustness in predicting ductile failure and transferability of a system that has different

materials using the modGTN model. The simplified welded plate model here has three material regions: the parent material (BASE) being joined together, the filler weld metal (WELD), and a region of reduced strength adjacent to the weld zone [Ship Structure Committee, 2010]. Cumulative thermal effects during the welding process have softened the parent material near the weld joint, and this softened region is called the heat-affected zone (HAZ) [Hakem et al 2012]. A photographic relation of the raw and simplified BASE, HAZ, and WELD material regions are shown in Figure 1. Commonly for Al 5xxx series, HAZ has a reduction in strength about 30%~50%, and normally extends out to about 10mm~30mm counting from the centerline of the weld [Zhang et al 2001; Collette, 2009].

(a)



(b)

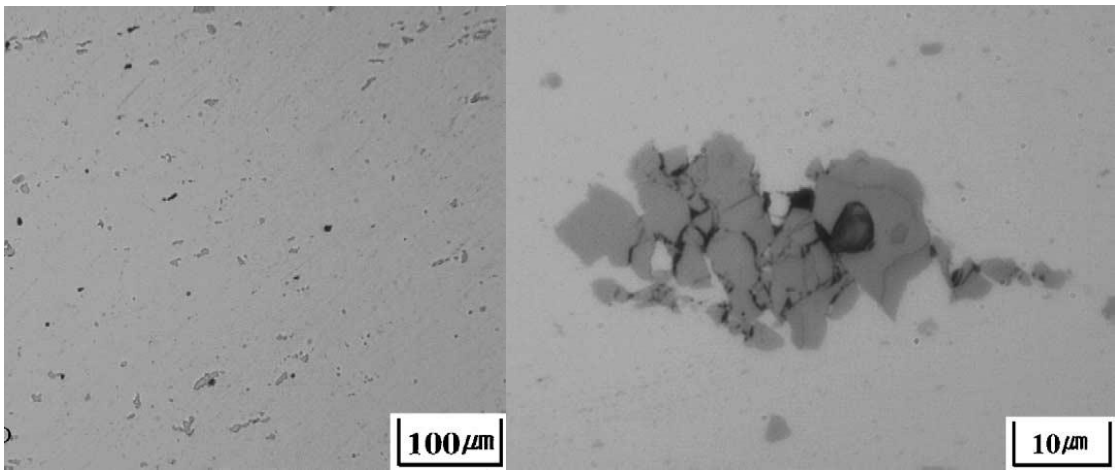
Figure 1. Material sections in a 90^0 welded plate. (a) X-ray image obtained from U.S. Navy [2009] showing approximate boundary of each material section. (b) Simplified boundary for each material section showing the middle section as WELD, two thin sections immediately adjacent to WELD are HAZ sections, and outside of the HAZ sections are the BASE sections.

Welding process leaves a specimen with inhomogeneity in the adjacent material and this material inhomogeneity can be observed in the microscale [Collette, 2009], so different matrix material responses must be considered for each of the BASE, HAZ, and WELD sections. The matrix material response of BASE has been established by Bomarito and Warner [2016] as Al 5083-H116. For the purpose of this study, the only difference between HAZ and BASE material is that the matrix material has gone from work hardened to annealed, which microstructurally means decreased dislocation density. Thus, the HAZ is reasonably assigned as annealed BASE material, Al 5083-H111 [Collette, 2009; Ship Structure Committee, 2010]. The matrix material of both the BASE and the HAZ cell models were assumed to act as an isotropic elastic plastic material using a Von-Mises yield function with normal flow and isotropic hardening (no void volume fraction assumption in a Von-

Mises criterion). The filler weld metal is often a weaker metal than the parent metal in practice to prevent weld metal cracking [Scott and Gittos, 1983; Collette, 2009; Ship Structure Committee, 2010]. WELD elements are solidified from liquefied metal, which also means fully annealed and no work hardened. So the WELD matrix material also assumed material parameters of the Al 5083-H111 [Francis 2002], but its matrix material is a bit more complex due to its total void volume fraction represented from second phase particles and from porosity introduced during welding (see next paragraph). Separate void volume fractions were assumed for the WELD matrix material and its center spherical void. The WELD matrix material was assumed to act as an isotropic elastic plastic material using the porous model, modGTN, with an associative flow rule. Consequently, this means that the WELD matrix material is using the homogenized HAZ cell model response as input for the WELD cell model simulation (see below for explanation).

The void volume fraction in each material section was another main assumption investigated. The ductile failure process begins in the microstructure with nucleation of voids on second phase particle sties, either by crushing the particle or delaminating at the interface between the particle and its surrounding material matrix during the work hardening process associated with manufacturing this material [Park et al, 2003; Pardoen et al, 2010; Scheck and Zupan, 2011]. These nucleated voids will then grow and calescence under non compressive loads. Park et al [2003] observed that, particularly in aluminum, the majority of these nucleated voids occurred at the second phase particle sites, and that the already existing void sites contributed much less to new void nucleation and growth. Accordingly, the modGTN model assumes all second phase particles are equivalently sized voids [Bomarito and Warner, 2016]. Park et al [2003] observed the second phase particle

volume fraction in aluminum as approximately 0.022~0.025 with existing void volume fraction being approximately 0.0008~0.0016. Representative image of second phase particles at the microscale is shown in Figure 2. The second phase particle volume fraction specifically for Al 5083-H116 was obtained from works of Scheck and Zupan [2011] at the value of 0.03. The RVE of the BASE cell model therefore has a center spherical void of 0.03 volume fraction. Even though HAZ is a fully annealed BASE material, its second phase particles will not significantly change during annealing due to their high melting temperature. Thus, the HAZ micromechanical cell model was assumed to have the same initial void volume fraction as that of the BASE, which is 0.03.



(a) (b)

Figure 2. Optical microscopy and scanning electron microscopy images showing second phase particles in aluminum 5083 [Park et al, 2003]. (a) Sample of distribution of second phase particles. (b) An example of severe break-up of a second phase particles into several pieces under loading.

In addition to voids from second phase particles for Al 5083-H111 (which is 0.03), the WELD material section has porosity formed during weld solidification due to the abrupt drop in hydrogen solubility with hydrogen contamination being unavoidable on the surface of weld filler metals [Collette 2009; Hakem et al 2012]. The HAZ could be assumed

with the same initial void volume fraction as BASE because no hydrogen contamination was introduced to HAZ. Porosity formed during welding was conservatively determined from an X-ray image provided from the U.S. Navy experimentalists [2009] (described in Section 3). Combining the two accounts of void volume fractions, voids nucleated from the second phase particles sites and voids from weld porosity, the matrix material of the WELD was assumed to have void volume fraction of 0.03 and the center spherical void reflecting the larger porosity formed during welding process instead. With all considerations mentioned above, the WELD cell model used the HAZ microscale material response as the input for the WELD matrix material (see graphical example in Figure 3), aligning with the hierarchy approach that this multiscale model is trying to convey.

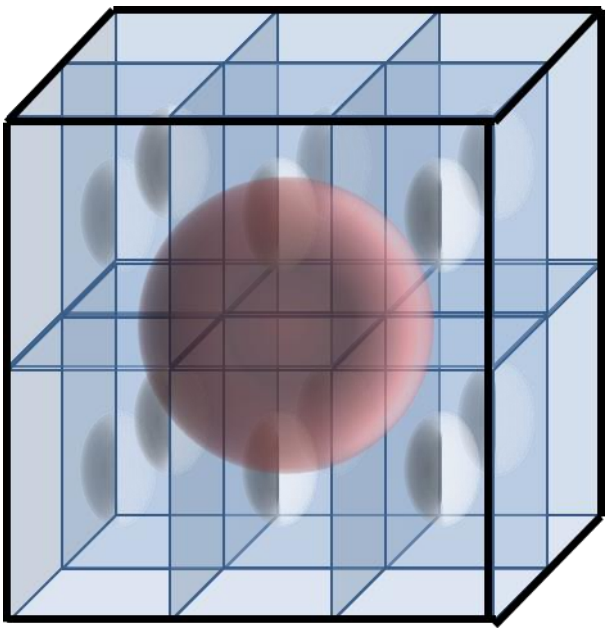


Figure 3. A graphical interpretation of the WELD microscale cell model. The smaller voids can represent the second phase particles observed in the Al 5083-H111 material, which is a void volume fraction of 0.03 and uses the HAZ micromechanical model response as input. The one large center void represents the porosity due to hydrogen contamination introduced during welding process.

3. Micromechanical Simulation Descriptions

The comprehensive details of the current model, modGTN, is defined in Bomarito and Warner [2015, 2016], only a brief description of the methodology will be given below.

The RVE exemplified in the microscale takes the form of a 3D computational cell model, which is common in the ductile failure literature. This micromechanical cell model consists of a void within a metallic matrix which is modeled using a 3D Finite Element (FE) model. Together with periodic boundary conditions, the cell represents a periodic array of voids within a matrix.

The stress state dependence of ductile failure is investigated by subjecting the micromechanical model to a number of proportional loadings across a range of stress states. Here, stress state is defined as the combination of stress triaxiality, T, and Lode parameter, ξ. They are defined as:

$$T = \frac{\sigma_h}{\sigma_e} \qquad \text{and} \qquad \xi = \frac{27 J_3}{2\sigma_e^3} \tag{1}$$

where $\sigma_h = \frac{I_1}{3}$ is the hydrostatic stress and $\sigma_e = \sqrt{3J_2}$ is the equivalent stress. I_1, J_2, and J_3 are the first invariant of stress, the second invariant of the stress deviator, and third invariant of the stress deviator, respectively. Deformation is applied to the cell such that the macroscopic stresses, Σ, stay proportional (constant stress state) during loading.

A group of loading orientations was considered for each given stress state. The loading orientations take into account the rotation of the far field stress relative to the array of voids. In this work, the loading orientation is described by the angle ϕ by which the far field stress is rotated about the X_2 axis. Where $\phi = 0$ corresponds to the case of the principle stresses, $\Sigma_1 \geq \Sigma_2 \geq \Sigma_3$, aligning with the coordinate axes X_1, X_2, and X_3 (see Figure 4(a)). For each of these simulations, a critical equivalent strain (ultimate failure

strain), $E_e^c(\phi)$, is found. Material failure is assumed to be governed by the weakest orientation; and thus, the failure strain for each stress state is defined as $E^f = min\left(E_e^c(\phi)\right)$.

The geometry of the computational cell model is a rectangular prism with dimensions D_1, D_1, and D_2. The center of cell consists of a spherical void with diameter d. The rectangular prism geometry allows for the prescription of the initial spacing ratio, χ_0, of the voids (the initial spacing between the void and its nearest periodic neighbor normalized by the void size) in addition to the initial void volume fraction f_0 of the cell. χ_0 and f_0 are related to the dimensions of the cell by:

$$\frac{d}{D_1} = \sqrt{\frac{6}{\pi} f_0 \chi_0} \qquad \text{and} \qquad \frac{d}{D_2} = \frac{1}{\chi_0} \tag{2}$$

Two sets of loading orientations are simulated for each stress state: (Set I) X_2 aligned with one of the D_1 dimensions (Figure 4(b)) and (Set II) X_2 aligned with the D_2 dimension (Figure 4(c)). The weakest orientation is thus found from the union of the two sets: $E^f = min(E_e^c(\phi)^I \cup E_e^c(\phi)^{II})$.

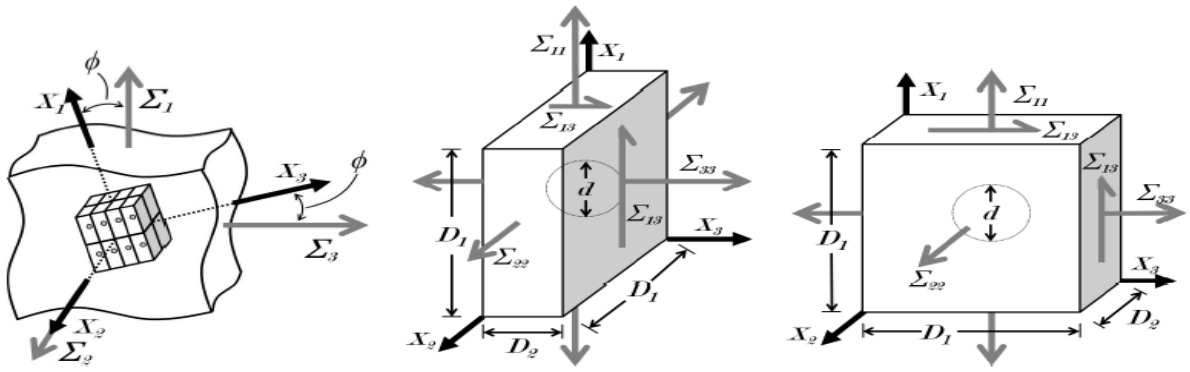


Figure 4. Orientations of loading on the cell model. (a) The rotation of the far field principle stresses by an angle ϕ about the χ_2 axis. The general form of the resulting stresses which are applied to the micromechanical cell model for the two sets of loading orientations: (b) Set I and (c) Set II. Set I corresponds to the loadings in which the χ_2 axis is aligned with the D_1 dimension of the model. Set II corresponds to the loadings in which the χ_2 axis is aligned with the smaller D_2 dimension.

To make these micromechanical results useful at a larger scale, a homogenization approach was taken. A homogenized material model with associative plastic flow rule was developed based on the GTN porous plasticity model. It has the following yield function:

$$\boldsymbol{\Phi = \left(\frac{\sigma_e}{\sigma_y^{(M)}}\right)^2 + 2q_1(T,\xi)f^* cosh\left(\frac{3}{2}q_2(T,\xi)\frac{\sigma_h}{\sigma_y^M}\right) - 1 - (q_1(T,\xi)f)^2} \quad \textbf{(3)}$$

with

$$\dot{\boldsymbol{f}}^* = \begin{cases} \dot{\boldsymbol{f}} & : \boldsymbol{f \le f_c(T,\xi)} \\ \boldsymbol{\kappa \dot{f}} & : \boldsymbol{f_c(T,\xi) < f \le f_u} \\ \boldsymbol{0} & : \boldsymbol{f_u < f} \end{cases} \quad , \quad \boldsymbol{\kappa = \frac{f_u - f_c(T,\xi)}{f_F - f_c(T,\xi)}} \, , \boldsymbol{f_u = \frac{1}{q_1(T,\xi)}} \quad \textbf{(4)}$$

and

$$\dot{\boldsymbol{f}} = \boldsymbol{(1-f)tr(\dot{\varepsilon}_p)} \quad \textbf{(5)}$$

where f and $\sigma_y^{(M)}$ are state variables for the void volume fraction and current yield stress of the matrix material, respectively. $\dot{\varepsilon}_p$ is the rate of plastic strain. The absolute failure porosity, f_F, is assigned a commonly assumed value of 0.25 [Tvergaard and Needleman, 1984; Needleman and Tvergaard, 1984; Koplik and Needleman, 1988; Becker et al., 1988]. The remaining three functions $q_1(T,\xi)$, $q_2(T,\xi)$, and $f_c(T,\xi)$ are the model's internal fitting parameters. In this work, the continuous parameter functions $q_1(T,\xi)$, $q_2(T,\xi)$, and $f_c(T,\xi)$ are defined by the interpolation of discrete arrays of q_1, q_2, and f_c values at each of the stress states simulated with the michromechanical model (denoted henceforth as A_{q_1}, A_{q_2}, and A_{f_c}). The optimization method of the functions $q_1(T,\xi)$ and $q_2(T,\xi)$ used all of the values in the arrays A_{q_1} and A_{q_2} in one large optimization procedure. Furthermore, bicubic interpolation is used to find internal fitting values that are in between the discrete values of A_{q_1}, A_{q_2}, and A_{f_c}.

The model relies on only three input parameters: (1) the large deformation constitutive response of the matrix, (2) the void volume fraction represented by voids nucleated at second phase particle sties and voids from weld porosity, and (3) the average nearest neighbor spacing of second phase particles and voids. Firstly, assuming that void growth and shearing do not contribute to the pre-failure macroscopic response of a material in compression, the large deformation constitutive response of the matrix is obtained from quasi-static large deformation compression data. The laboratory data of Skube [2009] for Al 5083-H116 was used to fit for the BASE material parameters (**Error! Reference source not found.**(black)). Similarly, the laboratory data of Winzer and Glinicka [2011] for Al 5083-H111 was used to fit for both the HAZ and WELD material parameters (**Error! Reference source not found.** (blue)) because both of these materials were assumed as fully annealed Al 5083-H116. The fitted hardening behavior was assumed to follow the power-law rule [Bomarito and Warner, 2015]:

$$\sigma_y = \sigma_0 \left(1 + \left(\frac{\dot{\varepsilon}^P}{Q_0} \right)^{\frac{1}{N}} \right) \tag{6}$$

where σ_0, Q_0, and N are the material parameters for initial yield stress, hardening coefficient and hardening exponent, respectively. This constitutive relationship is used both as the constitutive relationship of the matrix material in the micromechanical model and as an input to the homogenized material model. The material parameters for each of the material models are listed in Table 1.

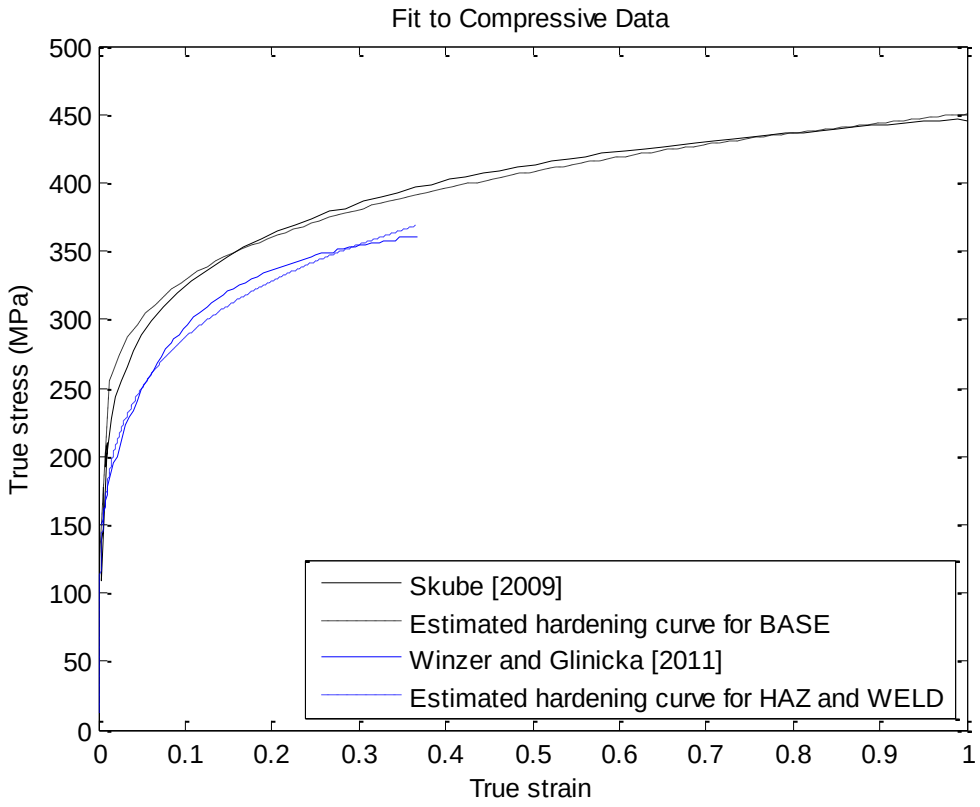


Figure 5. Large strain compression data of Al 5083-H116 [Skube, 2009] used to define the constitutive response of the BASE matrix material. Large strain compressive data of Al 5083-H111 [Winzer and Glinicka, 2011] used to define the constitutive response of the HAZ and WELD matrix material. The solid lines indicate the raw data taken from Skube [2009] and Winzer and Glinicka [2011], respectively, and the dotted lines indicate the fit of each curve using a power law model as described in Equation (6).

Table 1. Fitted hardening material parameters for each of the material models

Parameters	BASE	HAZ	WELD
ρ (kg/m³)	2660	2660	2660
σ_0 (MPa)	123.0	12.0	12.0
Q_0	0.007	1.230e-08	1.230e-08
N	5.06	5.07	5.07

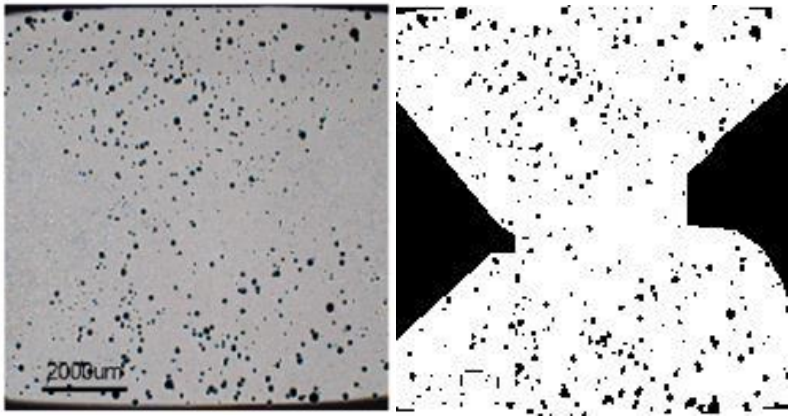
Secondly, assuming all second phase particles debond immediately upon loading, as discussed earlier, the initial second phase particle fraction can be used as f_0. For the BASE material model, Al 5083-H116, this value has been found to be 0.03 [Gao et al., 2009,

Scheck and Zupan, 2011]. By before mentioned reasoning, the f_0 for the HAZ material model is also 0.03. The weld zone void volume fraction was computed from a 2D X-ray image of weld zone porosity provided by the U.S. Navy [2009] experimentalists (Figure 1 cropped down to Figure 6(a)). If a medium is isotropic, the void fraction determined in 2D yields the same numerical value as determining void fraction in 3D [Ardell, 1971; Vander Voort, 1991; Bear and Bachmat, 2012; Dullien, 2012; Puli and Rajvanshi, 2012] (please refer to these references for detailed derivations). The image was taken at the mid-sagittal plane of a 90^0 welded plate (plate geometry description in Section 5). The dispersion of the voids captured in this image was assumed periodic into the width of the plate. This image has a reliable resolution of $52\mu m$, which means the detectable void in the image are $52\mu m$ in diameter or larger. Following image analysis technique for determining void fraction by Puli and Rajvanshi [2012], Figure 6(a) was first simplified to a binary image, Figure 6(b), where the two black trapezoid areas on left and right were defined as HAZ zone, with the white hour-glass area in the middle being the only true weld zone; and the black dots in the white area were the detectable voids in the weld zone. The void volume fraction determined from this image was:

$$\frac{\textit{black particle areas}}{\textit{total image area} - 2\ \textit{black trapezoid areas}} \times 100 = 5.7\% \qquad (7)$$

The mode of the second phase particles observed by Park et al [2003] have diameter of $1.5\sim3.5\mu m$, much smaller than the $>50\mu m$ diameter voids observed in the image, which mean the that the second phase particles are impossible to be detected from this image but must not be forgotten. Thus, for the WELD material model, the total f_0 corresponds to a voids nucleated at the second phase particle sties in the matrix material and the larger voids introduced during the welding process. For the WELD micromechanical cell, the

matrix material was assumed with $f_0 = 0.03$ and the centered spherical void was assumed to be 5.7% of the total cell volume, concluding the total f_0 of WELD material model as 0.087.



(a) (b)

Figure 6. 2D X-ray image of weld zone porosity provided by experimentalists from the U.S. Navy [2009]. (a) Raw image of the weld zone with scale marker of 2000$\boldsymbol{\mu m}$. (b) Post processed image for counting weld zone voids more accurately.

Thirdly, the average nearest neighbor spacing (*anns*) of the assumed voids was used to determine χ_0 in each of the micromechanical cell models. The volume fraction of each assumed center spherical void was used to determine the *anns*. Using X-ray tomography, Scheck and Zupan [2011] found this χ_0 value to be approximately 1.19 for Al 5083-H116. This value is used for both the BASE and HAZ cell models. For a void volume fraction of 0.057, the edge-to-edge average nearest neighbor spacing relationship is defined as *0.7133r = anns* by Bansal and Ardell [1972], where r is void radius. Furthermore, *anns* of the micromechanical cell geometry can be substituted with $D_2 - d$, which then combined with Equitation (2) to derive χ_0 to be 1.3567 for the WELD cell model. Table 2 lists the micromechanical cell dimensions for each material model. With these three set of inputs,

the micromechanical model for each of the BASE, HAZ, and WELD material models is then fully defined.

Table 2. Micromechanical cell model dimensions for the BASE, HAZ, and WELD material models.

Dimension (μm)	BASE	HAZ	WELD
D_1	15.3188	15.3188	10.4083
D_2	4.76	4.76	5.4268
r	2	2	2
χ_0	1.19	1.19	1.3567

4. Micromechanical Simulation Results

An approximate range in stress states of $0.4 \leq T \leq 1.8$ and $-1.0 \leq \xi \leq 1.0$ was simulated with the micromechanical models. This range resembles the stress state in terms of T and ξ at the center of the specimen up to the failure point. Bomarito and Warner [2015] have shown when $T < 0.4$, failure occurs by void collapse and self-contact. Because contact was not modeled in the current study, triaxialities below 0.4 were not simulated. The Lode parameter range of $-1.0 \leq \xi \leq 1.0$ was studied, with $\xi = 1$ being uniaxial tension, $\xi = 0$ pure shear, and $\xi = -1$ biaxial tension [Benzerga and Leblond, 2010; Barsoum et al, 2011; Zhou et al, 2014]. Using a resolution of 0.2 in both T and ξ, approximately 90 discrete stress states were investigated. For each stress state, 35 load orientations were simulated, resulting in a total of approximately 3,000 3D FE simulations for each of the three micromechanical material models, e.g. BASE, HAZ, and WELD. The 35 load orientations were a result of 19 simulations for Set I (X_2 aligned with one of the D_1 dimensions) and 16 for Set II (X_2 aligned with the D_2 dimension), with a discretization of ϕ into increments of 3^0 in regions of most interest. It should be noted that, due to symmetry, Set I has a unique range of ϕ values of 90^0 while Set II has a unique range of 45^0.

After E_e^c has been found for all loading orientations of a given stress state (or enough to provide confidence that a minimum has been found), E_f can be assigned to that stress state. Figure 7 shows the resulting relationship between E_f and stress state for each of the BASE, HAZ, and WELD micromechanical models. As predicted, different matrix material response and initial void volume fraction significantly affect ductility. The failure strain results have behaved as expected and in agreement with the experimental data [U.S. Navy, 2009], having WELD with the shortest failure strains and HAZ with the longest failure strains for

the corresponding stress states. Qualitatively speaking, all three of the material models maintained similar failure surface shapes ("U" shape) and trends (higher triaxialities result in lower failure strain). Large void growth failure primarily occurs at higher triaxiality with shorter failure strain because the void grows large enough to overcome the hardening of the matrix material and coalesce with neighboring voids. Failures due to void collapse or shearing occurs at low triaxialities with Lode parameter reaching -1 and tend to have longer failure strain. As discussed by Tvergaard [1982], plastic localization in a region of material occurs when a sub-region has been deformed to its maximum load. At this point, neighboring sub-regions can elastically unload by further deforming the failed sub-region. Void collapse and self-contact was observed in the cases of $T = 0.4, \xi < 0.0$ for BASE material model, and $T = 0.4,\ \xi < 0.2$ for HAZ and WELD material model; these points are removed from Figure 7. As a reminder of the hierarchy approach, the WELD cell simulations used the modGTN yield function, Equations (3) to (5), and the fitted parameters A_{q_1}, A_{q_2}, and A_{f_c} from the homogenized HAZ cell model as inputs. Comparing WELD failure surface, Figure 7(c), to HAZ failure surface, Figure 7(b), an increase in initial total void volume fraction has dramatically decreased failure strain for the same fitted matrix material response.

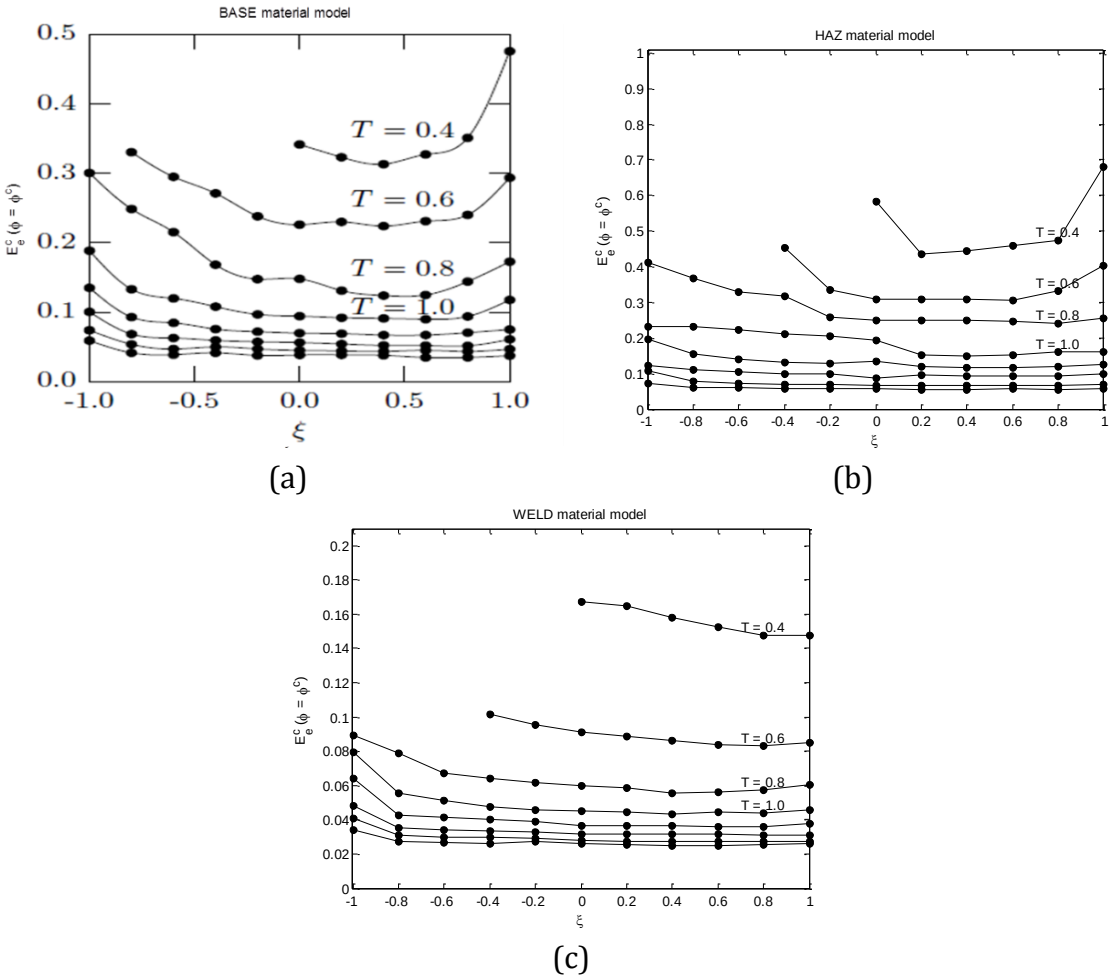


Figure 7. The dependence of failure strain on stress state of each micromechanical material model. Failure strain is given as a function of T and ξ; it was found based on the (a) BASE cell geometry, (b) HAZ cell geometry, and (c) WELD cell geometry. Each filled circle indicates the value of E_f calculated at a discrete stress state using the micromechanical model; the lines connect points of equal T value. Points not shown in the WELD model are those that are absent in the HAZ model. Since the fitted resulting parameters from the HAZ model were fed into the WELD model, if the HAZ model was uncertain in a region then when the WELD model is subjected to that same region, a large portion of the cell (modeled with HAZ) will be near that region of stress state and thus unreliable in the WELD model.

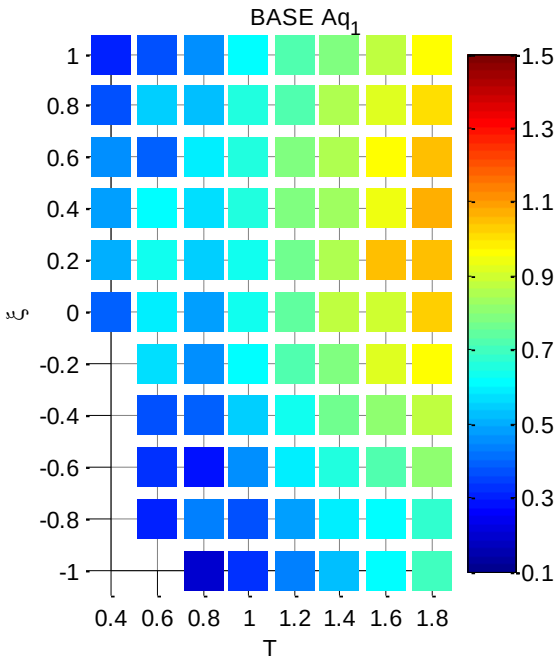
The aforementioned homogenization approach was then implemented, whereby the functions $q_1(T,\xi)$, $q_2(T,\xi)$, and $f_c(T,\xi)$ were fit to the micromechanical results. The $q_1(T,\xi)$, $q_2(T,\xi)$, and $f_c(T,\xi)$ functions were built upon the values of A_{q_1}, A_{q_2}, and A_{f_c} obtained by fitting all of the weakest orientations with the modGTN model, such that stress magnitude and void volume fraction of the modGTN model matched the respective values

of the micromechanical simulation. The A_{q_1}, A_{q_2}, and A_{f_c} values are the best fit values within the search range; however, many other values of these parameters in the same neighborhood resulted in nearly the same response. For this reason, the sensitivity to the exact fitted values is expected to be relatively small. Bicubic interpolation was used to extend the discrete values of A_{q_1}, A_{q_2}, and A_{f_c} to the continuous $q_1(T, \xi)$, $q_2(T, \xi)$, and $f_c(\mathrm{T}, \xi)$ functions.

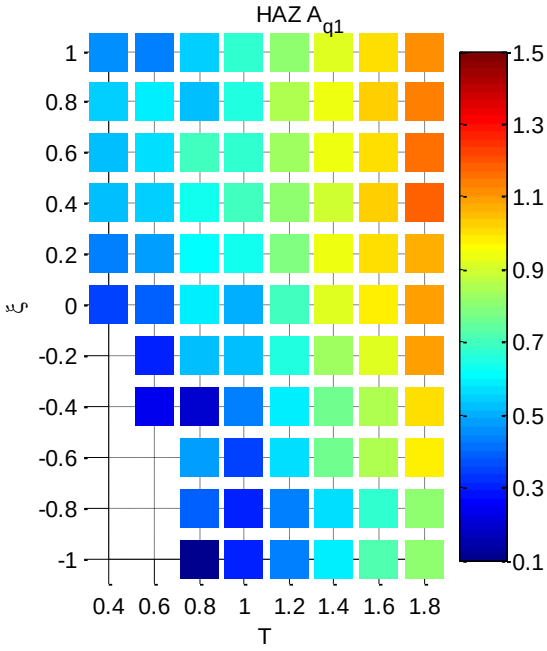
The A_{q_1}, A_{q_2}, and A_{f_c} values obtained for the BASE, HAZ, and WELD material models are shown in Figure 8. In agreement with Kim et al [2004] and Faleskog et al [1998], the fitted values of A_{q_1} and A_{q_2} are inversely related for all three of the material models: as A_{q_1} increases A_{q_2} decreases (compare Figure 8(a) and Figure 8(b) for corresponding materials). Larger A_{q_1} corresponds to lower GTN-type material strength [Slimane et al, 2015] as it is a direct multiplier of the void volume fraction. This is observed in the WELD material model that WELD has the highest A_{q_1} but the lowest E_f of all three materials. Considering that BASE and HAZ have the same void volume fraction but different matrix material responses, there is not a significant difference in A_{q_1} magnitudes ($\sim$0.1 difference). However, WELD used homogenized HAZ material response as input but used a much bigger f_0, magnitude of WELD A_{q_1} is much larger than HAZ A_{q_1} ($\sim$0.3 difference). This further confirms the direct relationship between A_{q_1} and void volume fraction across different materials concluded in Zhang et al [2000].

The amount of void growth before failure $(f_c - f_0)$ is not large and WELD has the least of this value. Generally, the small amount of void growth is attributed to the fact that the critical cell simulation for each stress state is a simulation with $\phi \neq 0$ allowing some shear

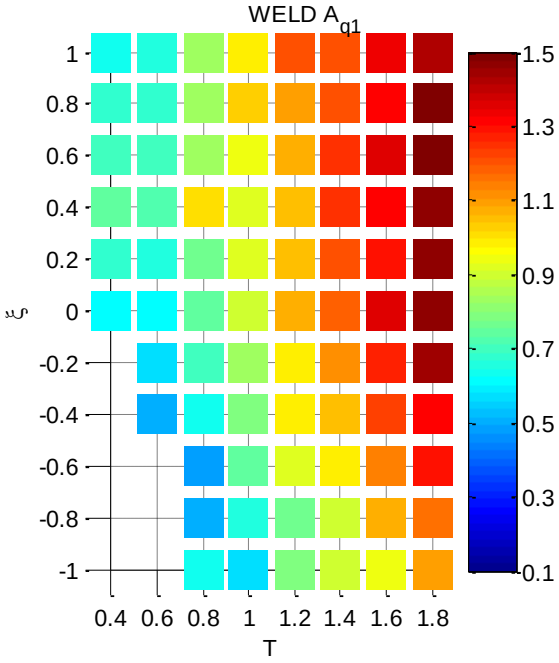
deformation that results in sooner failure of the cell [Bomarito and Warner, 2016]. The maximum void growth before failure ($f_c - f_0$) is approximately 0.06 for BASE, 0.095 for HAZ, and 0.023 for WELD. WELD had the largest f_0 among the three materials and incorporated the weakest matrix material responses; it has the smallest void growth before failure. While HAZ had a small f_0, its matrix material was a softened response of the BASE material; and subsequently, HAZ gained longer strain and void grow larger than that of BASE. Lastly, at higher stress triaxialities less void growth is needed to induce failure; and thus, lower values of result.


BASE Aq1
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ξ
0.4 0.6 0.8 1 1.2 1.4 1.6 1.8
T
1.5
1.3
1.1
0.9
0.7
0.5
0.3
0.1

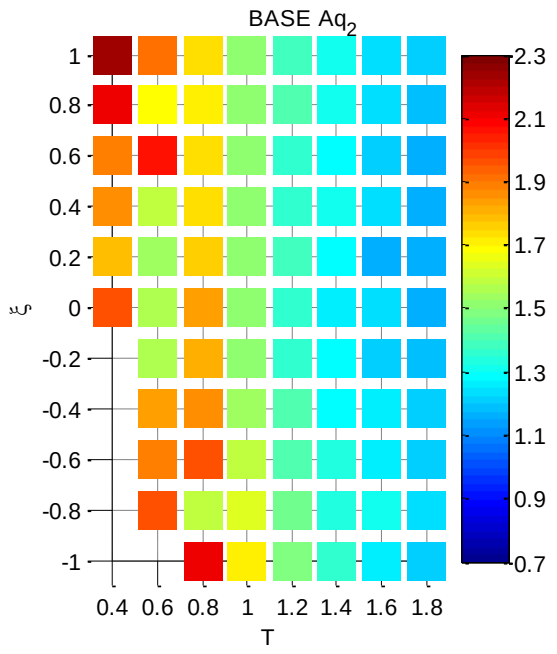
(a1)


HAZ Aq1
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ξ
0.4 0.6 0.8 1 1.2 1.4 1.6 1.8
T
1.5
1.3
1.1
0.9
0.7
0.5
0.3
0.1

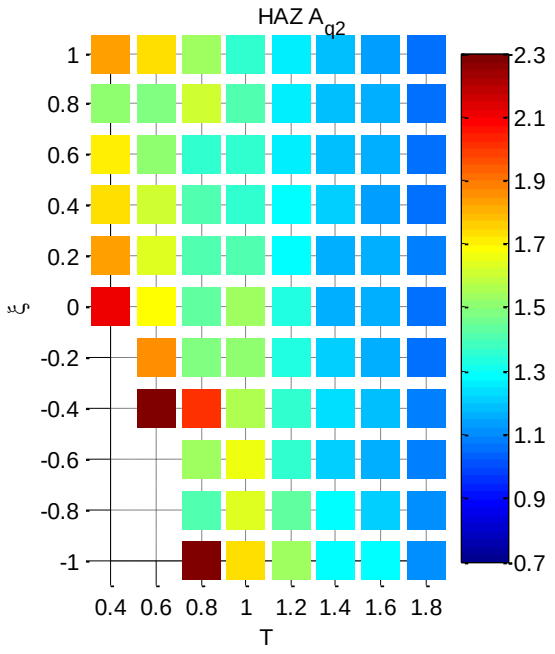
(a2)


WELD Aq1
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ξ
0.4 0.6 0.8 1 1.2 1.4 1.6 1.8
T
1.5
1.3
1.1
0.9
0.7
0.5
0.3
0.1

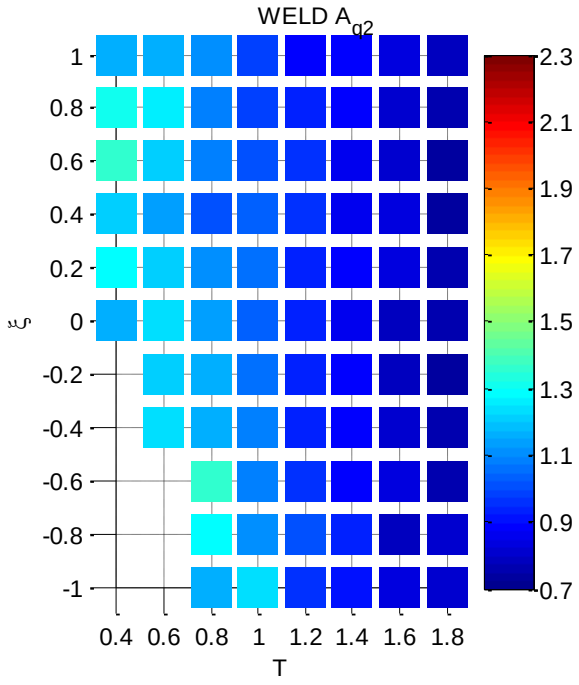
(a3)


BASE Aq2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ξ
0.4 0.6 0.8 1 1.2 1.4 1.6 1.8
T
2.3
2.1
1.9
1.7
1.5
1.3
1.1
0.9
0.7

(b1)


HAZ Aq2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ξ
0.4 0.6 0.8 1 1.2 1.4 1.6 1.8
T
2.3
2.1
1.9
1.7
1.5
1.3
1.1
0.9
0.7

(b2)


WELD Aq2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ξ
0.4 0.6 0.8 1 1.2 1.4 1.6 1.8
T
2.3
2.1
1.9
1.7
1.5
1.3
1.1
0.9
0.7

(b3)

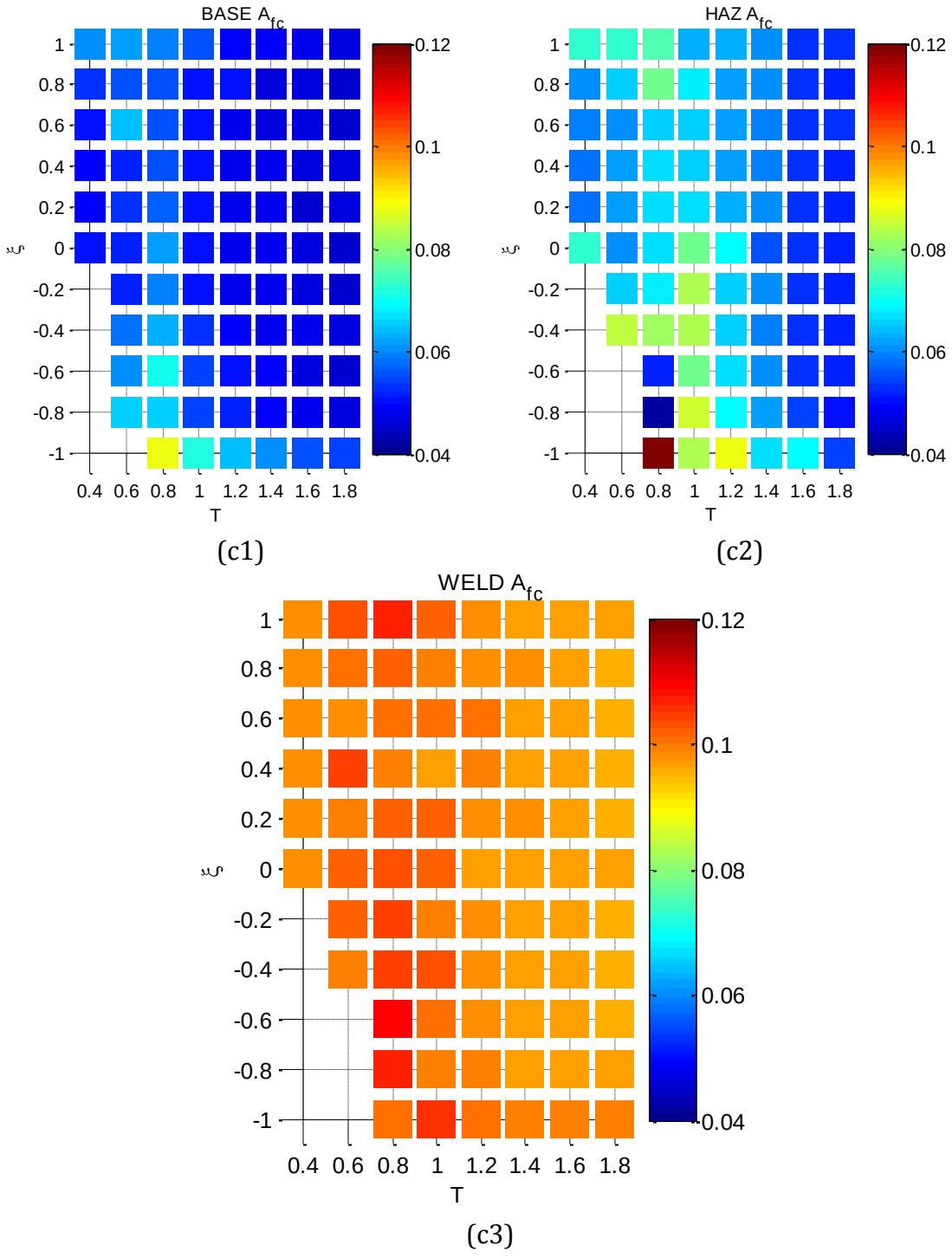
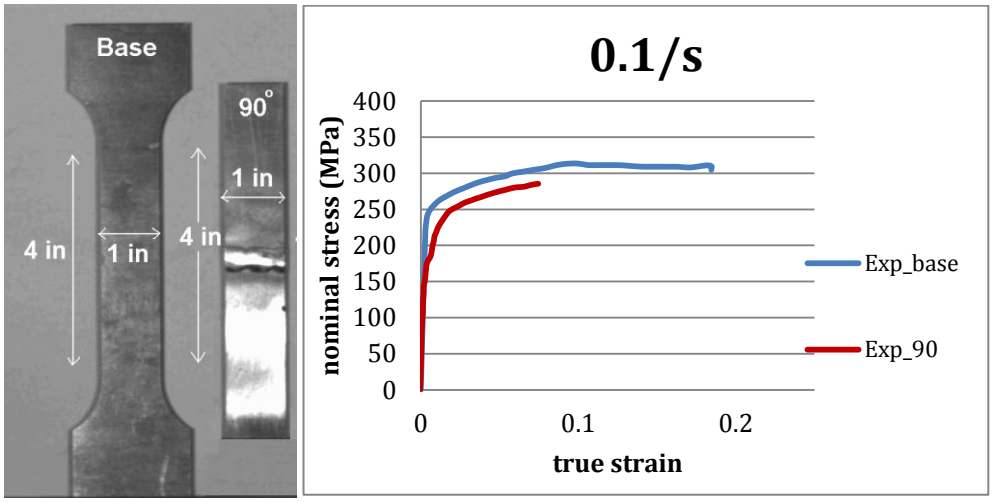


Figure 8. Results of optimization of homogenized material model parameters. (a), (b), and (c) are color contour plots of the fitted arrays of values A_{q_1}, A_{q_2}, and A_{f_c}, respectively, as a function of stress state for each of the three material models. The absence of data in some regions is due to collapse and self-contact of voids, except for the case of T = 0.6 and ξ = -1.0 which was caused by numerical difficulties. The cases of T = 0.6 and ξ = -0.8 ~ -0.6 were removed due to an artifact in collecting E_f for those stress states, so the fitted values for these stress states were not accurate.

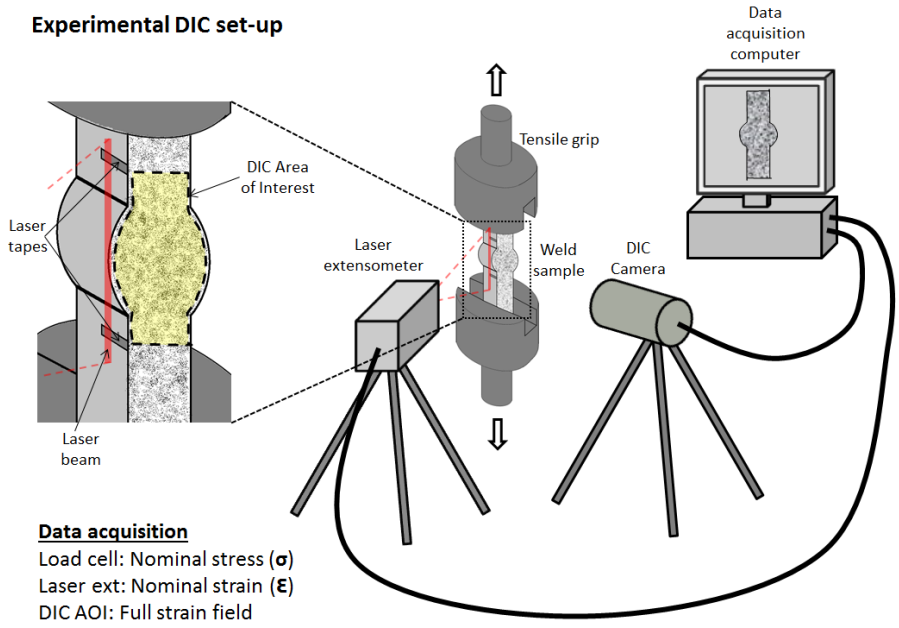
5. Macromechanical Simulation Descriptions

To quantify the accuracy of the modGTN model at the component scale, 3D FE models were constructed in Abaqus/ Explicit 6.14-2 using the homogenized material models as inputs, e.g. BAE, HAZ, and WELD models. Specifically, the true stress – true strain curves were examined for the Abaqus plate simulations against laboratory tests performed by U.S. Navy [2009]. The U.S. Navy [2009] experiments included a standard couple test apparatus of 2 specimens, and a digital image correlation (DIC) recording with a laser extensometer for local strain measurements on just the welded specimen. All tests conducted were uniaxial tensile tests and assumed uniform elongation on each specimen. In the standard tests, the gage length of each of the 2 specimens was 4" long, 1" wide, and 0.25" thick (Figure 9(a)). Figure 9(b) shows the engineering stress – true strain curves for these 2 specimens. In the DIC observation setup, only the 90° welded plate was recorded (Figure 10(a)), and its local engineering stress - engineering strain of each material sections (e.g. BASE, HAZ, WELD) were obtained and plotted in Figure 10(b). Since experimental images (not shown) have shown material heterogeneity around the weld region, the DIC experimental data account for the average response of the areas close to the weld center.

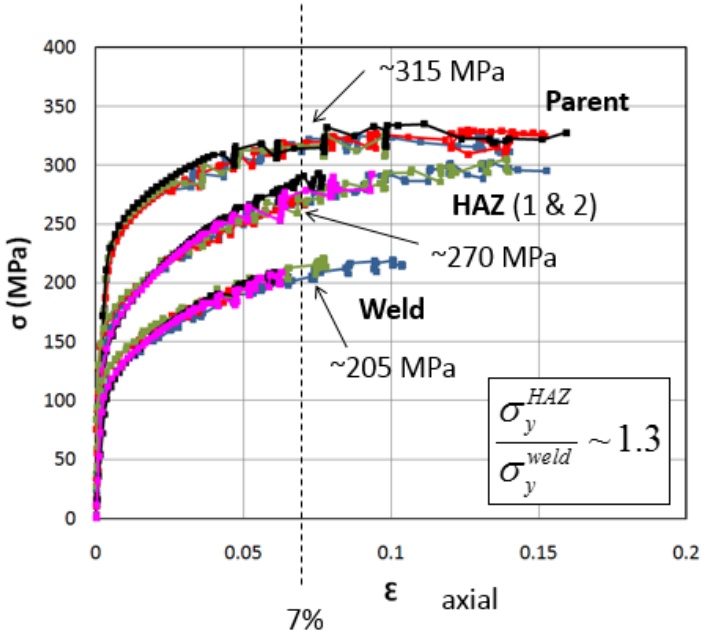


(a) (b)

Figure 9. All U.S. Navy experiment specimens used Al 5083-H116 as the parental material. (a) Left: single material, no weld plate; Right: 90^0 welded plate with the weld perpendicular to the loading direction. The weld zone can be identified as the shiny band in the mid region of the welded plate. (b) Engineering stress – true strain of the BASE material plate (blue) and the 90^0 welded plate (red) obtained at strain rate 0.1/s.



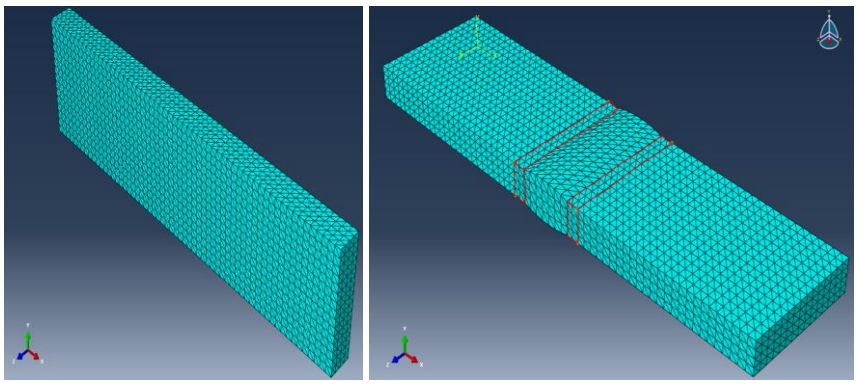
(a)



(b)

Figure 10. U.S. Navy [2009] DIC experiment. (a) DIC setup of the 90^0 welded plate. (b) Engineering stress – engineering strain of each material section obtained via DIC method. The DIC curves represent the stress – strain responses averaged over all of the spackle points (the different colors) in a designated area from each material section near the weld zone.

The two micromechanical models made in Abaqus used ten-node, quadratic tetrahedral elements. Mesh convergence studies of each Abaqus model was performed to ensure convergence of the predicted failure stress and failure strain. The converged meshes all have an element size of 1700um. First model, 1_material_plate, was studied for tensile deformation and failure of unwelded single material plate (Figure 11(a)), and was used to help determine the plasticity input of each material in Abaqus for using the Power Law and stdGTN models (see Section 6). The second model, 90°_welded_plate (Figure 11(b)) was used to test the capability of the current stress dependent modGTN constitutive material model using different materials to predict the ductile failure of the whole system. The welded plate consists of multiple material sections as mentioned before. Plate model specifics are listed in Table 3.



(a) (b)

Figure 11. (a) Plate with only 1 material. (b) 90^0_welded_plate with 3 materials. A welded plate is partitioned into a total of 5 sections. In (b), the middle rounded section is the WELD section, two marked sections immediately adjacent to the weld zone are the HAZ sections, and outside of the HAZ sections are the BASE sections.

Table 3. Specimen scale model dimensions used in Abaqus.

	1 material plate (Figure 11(a))	**90^0 welded plate (Figure 11(b))**
X (μm)	101600	101600
Y (μm)	25400	6350
Z (μm)	6350	25400
HAZ width (μm)	N/A	2800
Weld width (μm)	N/A	12000
Crown height (μm)	N/A	800
Elements	22643	24745
Nodes	35076	37978

Both models in Abaqus have the same roller boundary conditions to resemble uniaxial tensile tests. The displacement on one end face is fixed in the loading direction, and roller boundary conditions were used in the other two directions. On the other end face, a velocity was attached to the face in the loading direction, and again, roller boundary condition were used on the other two directions Boundary conditions are depicted in Figure 12. The velocity used was 10160 $\mu m/s$ in order to imitate the strain rate at which the experimental specimen was loaded, at strain rate 0.1/s. Since the simulations studied

are assumed to be quasi-static, a mass scaling factor of 40,000 is incorporated in Abaqus to speed up the explicit FE analysis. In Abaqus, the true stress – true strain can be extracted directly from an integration point in the FE method. However, in order to compare to the experimental data, displacement over a gage length was used to obtain the true strain (as was done in the experiment). Gage length used for the global strain of a whole plate would be the total length of that plate.

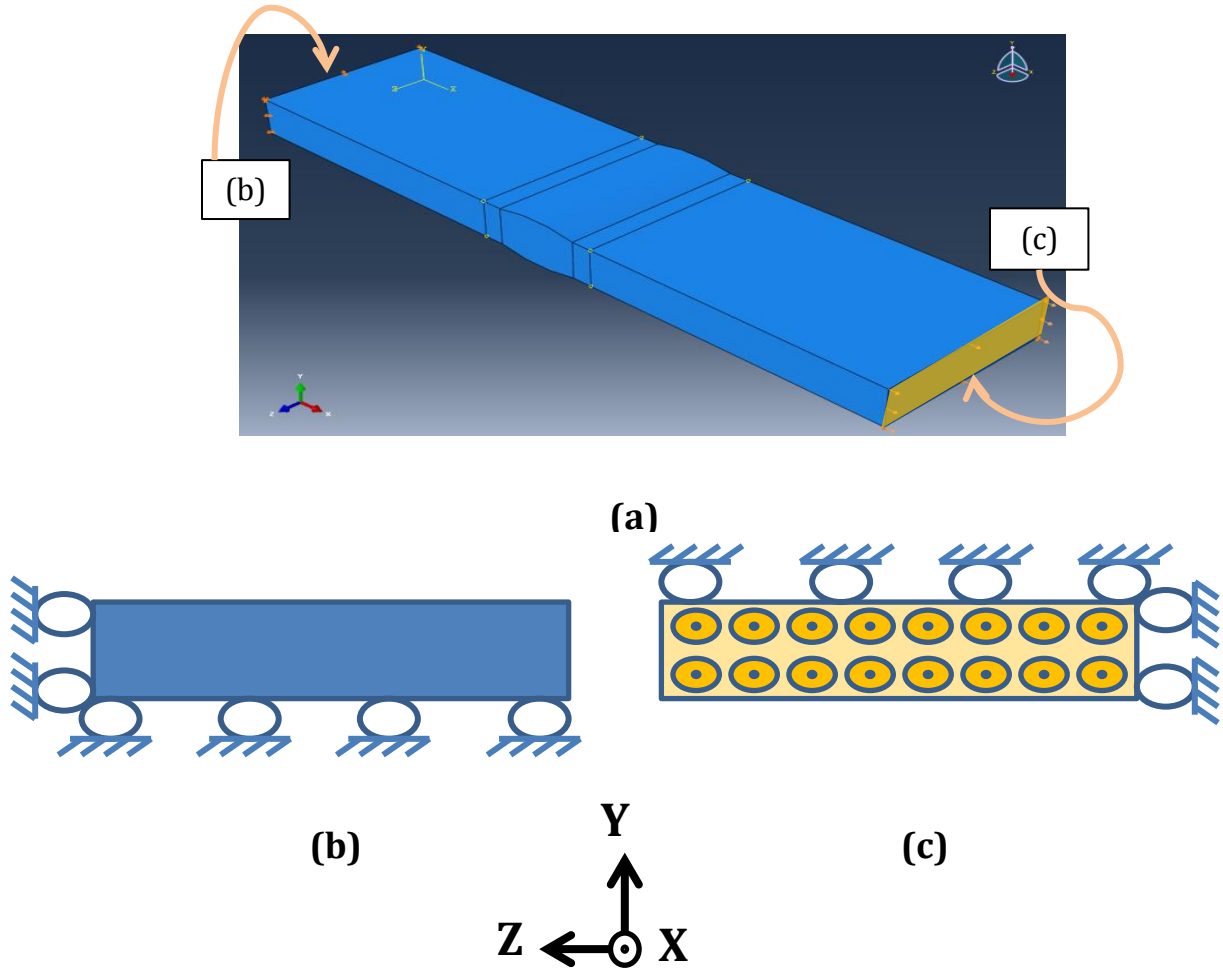


Figure 12. Uniaxial tensile boundary condition. If the loading direction is in the X-axis coming out of the page, (a) shows the end faces depicted in (b) and (c) . (b) One end face has fixed displacement in the X direction and rollers on two edges in the Y and Z directions. (c) The other end face would have velocity coming out of the page in the loading direction and rollers on opposite two edges in the Y and Z directions.

The Abaqus simulations required three piece of inputs: (1) geometry and boundary condition specifics, (2) user defined subroutine as material constitutive model (the modGTN model code), and (3) homogenized material parameters from each material model. To run modGTN constitutive model in Abaqus, a user subroutine VUMAT was generated in Fortran 77 language (see Appendix A). For each material section in the plate geometry, the material user subroutine reads in the corresponding material parameters, e.g. the material constitutive parameters and its internal fitting parameters A_{q_1}, A_{q_2}, and A_{fc}. Double precision was enforced during analysis, and time step was determined by Abaqus automatically based on the size of the critical element at that time increment.

6. Macromechanical Simulation Results

True stress and true strain obtained from modGTN component scale models in Abaqus were first examined against the U.S. Navy [2009] standard experimental tests (experimental data refer to Figure 9(b)). The engineering stress in the experimental data was initially converted to true stress by the following relationships:

$$\varepsilon_{tr} = ln(1 + \varepsilon_{eng}) \tag{8}$$

$$\sigma_{tr} = \sigma_{eng}(1 + \varepsilon_{eng}) \tag{9}$$

To obtain the global response of the whole plate model in Abaqus, the true stress was evaluated at an end element (using integration points) and true strain was obtained from the nodal displacement of a node that connects to that end element. The nodal displacement was then translated to true strain by equation

$$\varepsilon_{tr} = ln\left(1 + \left(\frac{nodal\ displacement}{gage\ length}\right)\right) \tag{10}$$

using gage length of $101600\mu m$, or 4", which is the plate model's longitudinal length. The results of the global response of the specimen models simulated in Abaqus are presented in **Error! Reference source not found.**. To obtain the local responses within the 90^0_welded_plate in Abaqus, the true stress and true strain were evaluated at several representative elements in each material section (using integration points).

In order to place the accuracy of the modGTN material model in context, two other common constitutive models were tested on the same specimen models in Abaqus. The first constitutive model is denoted as the Power Law model, which assumes Von-Mises yield function with normal flow and isotropic hardening. This model does not depend on void volume fraction and stress states (T, ξ) but only depends on the large tensile deformation of matrix material response. This model was fitted to tensile deformation of material response instead of the compressive response because tensile deformation

already includes the effect of void evolution during loading. For each of the BASE, HAZ, and WELD materials, a power regression curve was fitted to the tensile experimental data provided by the U.S. Navy DIC experiment tests (Figure 9(b)). A detailed Power Law model operational procedure is as follows:

- Convert the tensile DIC experimental data from engineering stress and true strain to true stress and true strain.
- Fit a power regression curve on the true stress – true strain data in Excel. This is only a phenomenological fit to the material tensile deformation response. The equation of the curve takes the format of $\sigma_{true} = A{\varepsilon_{true}}^{B}$.
- Interpolate true plastic strain and true stress from the fitted power law equation.
- Feed the pairs of true plastic strain and true stress data into Abaqus plasticity definition for the corresponding material.

The second constitutive model is the classic Standard GTN model (stdGTN) with phenomenologically fitted parameters. The yield function described earlier by Equation (3) through (5) describes this model with exception that in the stdGTN model, the $q_1(T,\xi)$, $q_2(T,\xi)$ and $f_c(T,\xi)$ are constants because stdGTN does not depends on Lode parameter. The constant internal fitting parameters used for stdGTN were $q_1 = 1.5, q_2 = 1.0$, and $f_c = 0.25$ from the original GTN model paper [Tvergaard and Needleman, 1984]. Using the stdGTN model in Abaqus has the same operational procedure (shown above) as the Power Law model except that stdGTN was fitted to the large compressive deformation material response (Figure 5) given its porous characteristics. Unlike the Power Law model, stdGTN is famously used in ductile fracture community for porous material with the void evolution

being its main dependence. In this study, the stdGTN model assumed the same initial void volume fraction as that of the modGTN for the corresponding materials. A summary of the fitted power regression relationship $\sigma_{true} = A{\varepsilon_{true}}^{B}$ for the Power Law model and the stdGTN model are listed in Table 4.

Table 4. Coefficients and exponents of the fitted power regression equation in the format of $\boldsymbol{\sigma_{true} = A{\varepsilon_{true}}^{B}}$ for the Power Law model and the stdGTN model.

Models		**Coefficient A**	**Exponent B**
Power Law	**BASE**	505.31	0.1501
	HAZ	549.09	0.2398
	WELD	426.76	0.2464
stdGTN	**BASE**	477.96	0.2065
	HAZ	487.8	0.2095
	WELD	487.8	0.2095

Table 5 below summarizes parameters used for each of the three constitutive models tested in this paper (Power Law model, stdGTN model, and modGTN model) and the experiments to which each material parameter was fitted to. Each of the three material models was tested in the two component-scale geometries in Abaqus and compared to the experiments of U.S. Navy [2009]; the resulting true stress – true strain curves are shown in Figure 14.

Table 5. Summary of parameters used in the three tested constitutive material models.

Models		f_0	Matrix Hardening Curve	f_F	$q_1(T,\xi)$	$q_2(T,\xi)$	$f_c(T,\xi)$
Power Law	**BASE**	n/a	Power regression fitted to uniaxial tensile test from U.S. Navy [2009] DIC experimental data	n/a			
	HAZ						
	WELD						
stdGTN	**BASE**	0.03	Power regression fitted to uniaxial compression test from Skube [2009]	assumed value of 0.25	constant 1.5	constant 1.0	constant 0.25
	HAZ	0.03	Power regression fitted to uniaxial compression test from Winzer and Glinicka [2011]				
	WELD	0.087					
modGTN	**BASE**	0.03	Hardening Equation (6) fitted to uniaxial compression test from Skube [2009]		from micro-scale simulations χ_0 = 1.19		
	HAZ	0.03	Hardening Equation (6) fitted to uniaxial compression test from Winzer and Glinicka [2011]				
	WELD	0.087			from micro-scale simulations χ_0 = 1.3567		

In the single material plate test of the BASE material (Figure 13), the stdGTN constitutive model has the greatest degree of accuracy in predicting failure strain (but not failure stress). The BASE plate laboratory specimen failed at a true stress of approximately

372MPa and a true strain 0f 0.18, while the currently investigated modGTN model failed at a true stress 335MPa and a true strain 0.15; a 18.10% under-prediction of ductility. Bomarito and Warner [2016] noted that the current version of the modGTN model tends to under predict ductility at $\xi = 1.0$ (uniaxial tension). The under prediction could possibly because that a weakest loading orientation is chosen to represent the response at every point in the material. But in reality, the response of a material point will depend on the local geometry of particles at that point, which may not include particles aligned in the weakest orientation. The modGTN has higher degree of constitutive accuracy than the stdGTN during early loading even though they were both fitted to the same compressive matrix material responses. This phenomenon could be attributed to the high level of detail that was present in the micromechanical cell model and the definition of $q_1(T, \xi)$, $q_2(T, \xi)$, and $f_c(T, \xi)$ obtained from a hierarchy approach for the modGTN. While the BASE failure strain predicted by the stdGTN model had only a 1.67% under prediction, the large under prediction of stress could be because the q_1 and q_2 values was were not determined from current stress states at a material point but rather from constant values. The stdGTN did not include stress states that were outside of the compressive experimental data from Skube [2009].

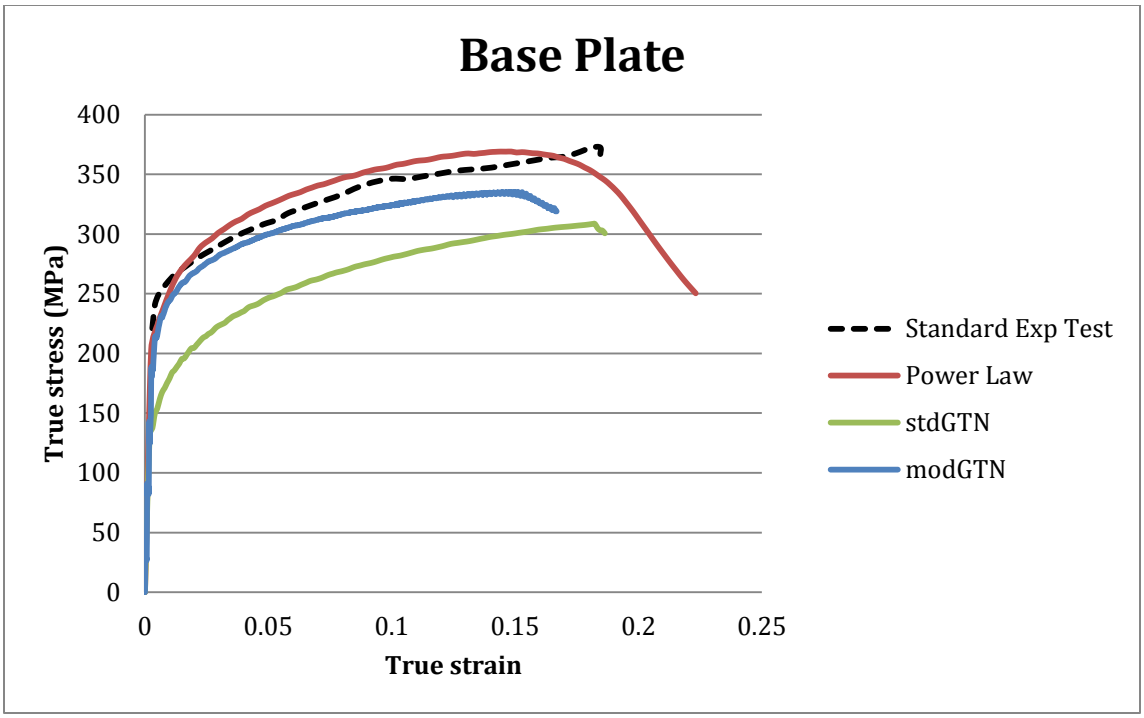


Figure 13. Global response of the single material model using each of the three constitutive models. Simulation results compared to the experiment data for the BASE 1_material_plate.

In the 90^0_welded_plate model, the result was consistent with experiment and literature [Kinsey and Wu, 2011; Ship Structure Committee, 2010], in that a weld causes less ductility in the system than without a weld. In Figure 14, the 90^0_welded_plate laboratory specimen failed at approximately a true stress of 307MPa and a true strain of 0.075, while the currently studied modGTN model failed at true stress of 300MPa and true strain of 0.086: a 15.27% over-prediction of ductility. The over-prediction of ductility could be because the alignments of particles were closer to the preferential orientation for the stress state fitted. Experiments have shown that while uniform and non-uniform distribution of voids show little difference in stress – strain response during early loading, however the localization of the deformation does depend significantly on the details of the distribution with the uniform distribution predicting longer ductility [Bansal and Ardell,

1972; Becker, 1988]. The modGTN model predicted similar stress – strain responses during early loading, closely following the experimental data. Once again, this was very likely due to the high level of detail that was present in the micromechanical cell model and the definition of $q_1(T,\xi)$, $q_2(T,\xi)$, and $f_c(T,\xi)$ obtained from a hierarchy approach for the modGTN model.

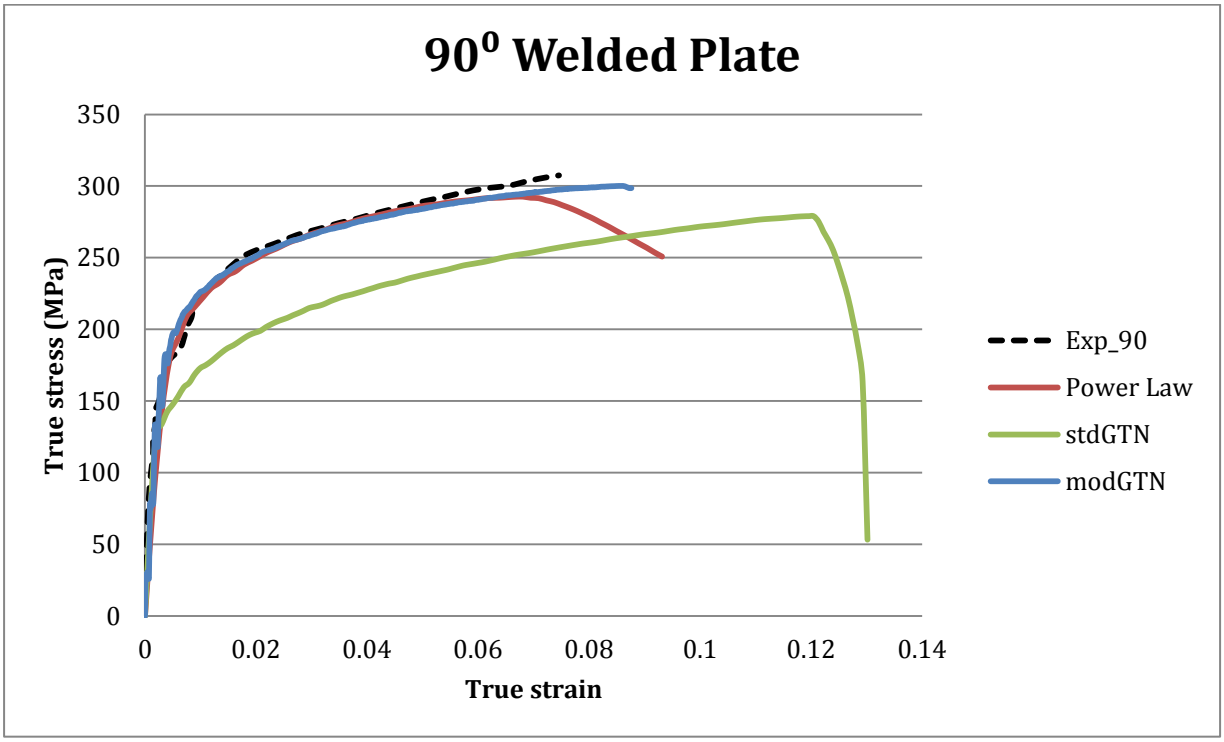
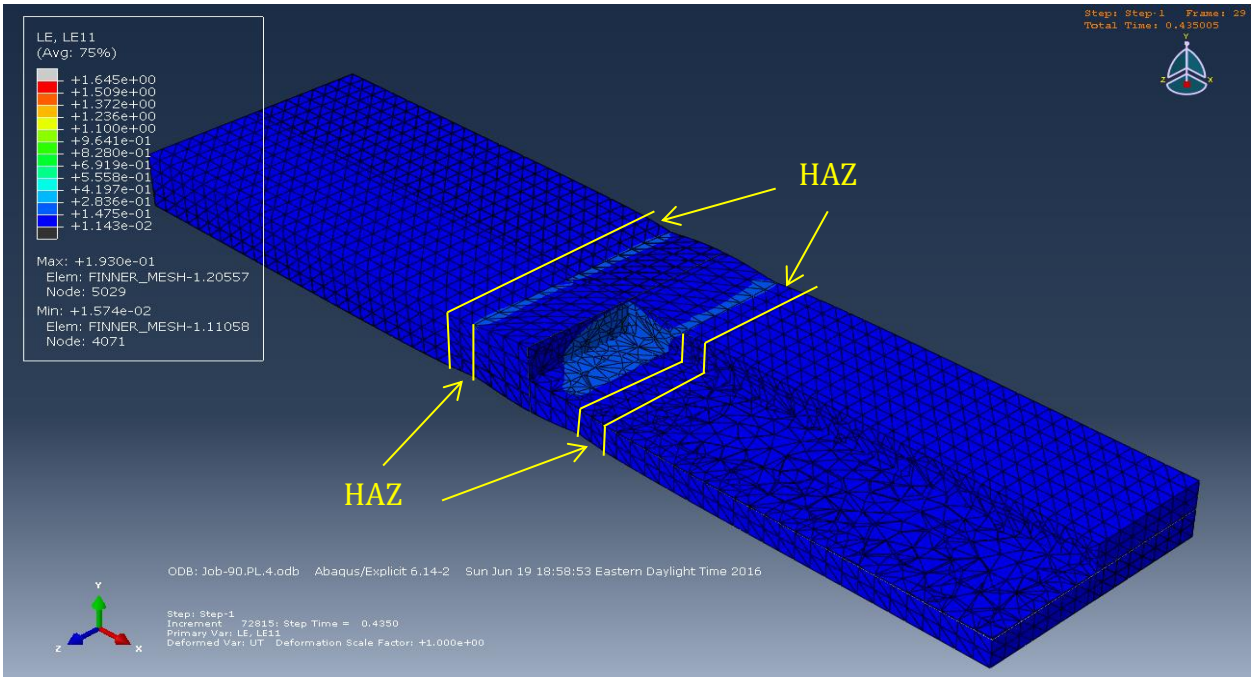


Figure 14. Global response of the 90^0_welded_plate model using each of the three constitutive models. Simulation results compared to the experiment data.

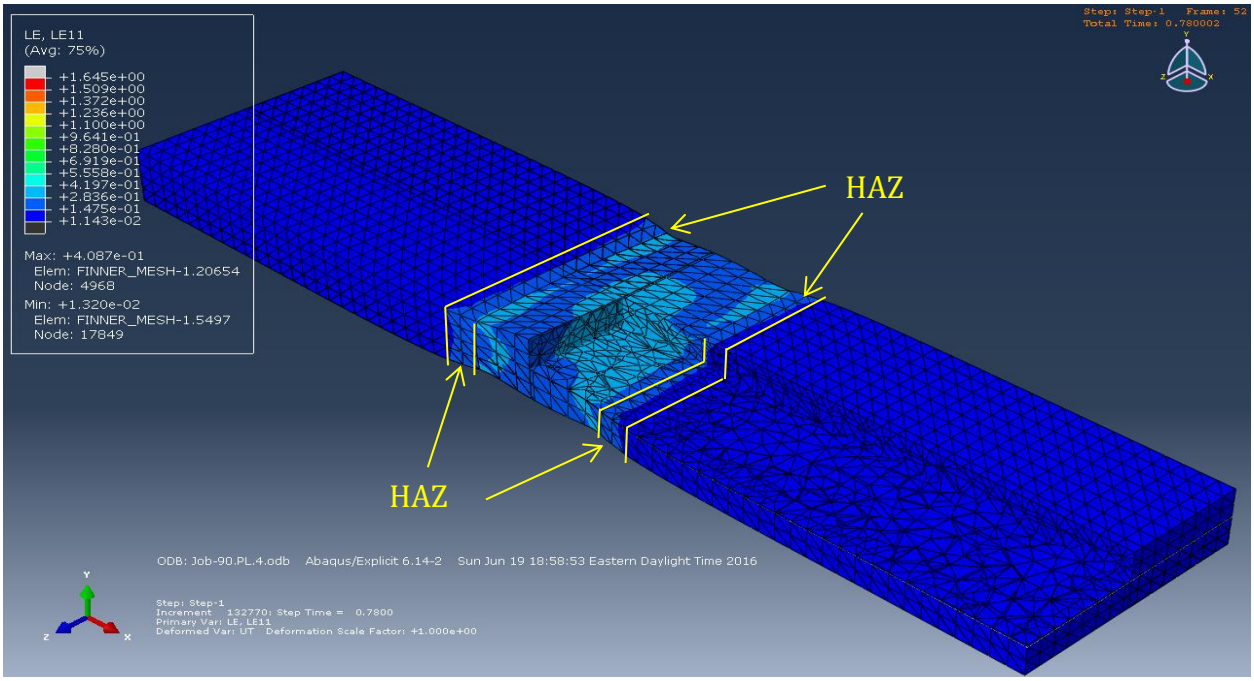
The plastic strain progression in the 90^0_welded_plated was scrutinized first. The modGTN welded plate simulation behaved in accordance with literature in that plastic strain first initiated in the HAZ section, then propagated into material sections in the following order: BASE, HAZ, and WELD; finally the system failed due to strain localization about the HAZ/WELD section borders in the HAZ material. The WELD has larger cross sectional area than the rest of the plate so stress and strain would naturally concentrate at

the sharp change in geometry between WELD and HAZ sections. In the stdGTN simulation, plastic strain was first observed in the two inner edges of the WELD section, then propagated into the HAZ zone, and then into the BASE zone before fully propagating through the WELD zone. At the failure point, the stdGTN simulation predicted failure due to strain localization on the inner edge of the WELD section bordering the HAZ section. While the true stress – true strain global responseportraied the simple Power Law model as adequate, a closer inspection of failure progression portrays otherwise. In the Power Law simulation, the plastic strain initiates at the two inner edges of the WELD section and propagates throughout the entire weld region first. During strain propagation, higher strain was observed at the four corners and the middle region of WELD, and failure due to necking occurred in middle of the WELD section. The behavior predicted by the Power Law was not fully consistent with the literatures or experimental observations. One interesting observation right before failure initiation was that, while the Power Law model necks at the very center of the WELD zone (see Figure 15(c)), the GTN-type simulations indicated a shear band of roughly 45^0 in the WELD, but the failure locations did not necessarily fall on this band. Moreover, both experimental studies and GTN-type simulations showed that the maximum strain point occurs on the side of the plate about the HAZ/WELD section border, and then failure rips across the width of the plate. Similar observation was also seen in experimental study from the U.S. Navy [2009], indicating shearing stress domination upon plate failure. The subsequent groups of figures show the progression of strain for the 90^0 welded plate using each of the three constitutive material models: the Power Law (Figure 15), the stdGTN (Figure 16), and the modGTN (Figure 17).

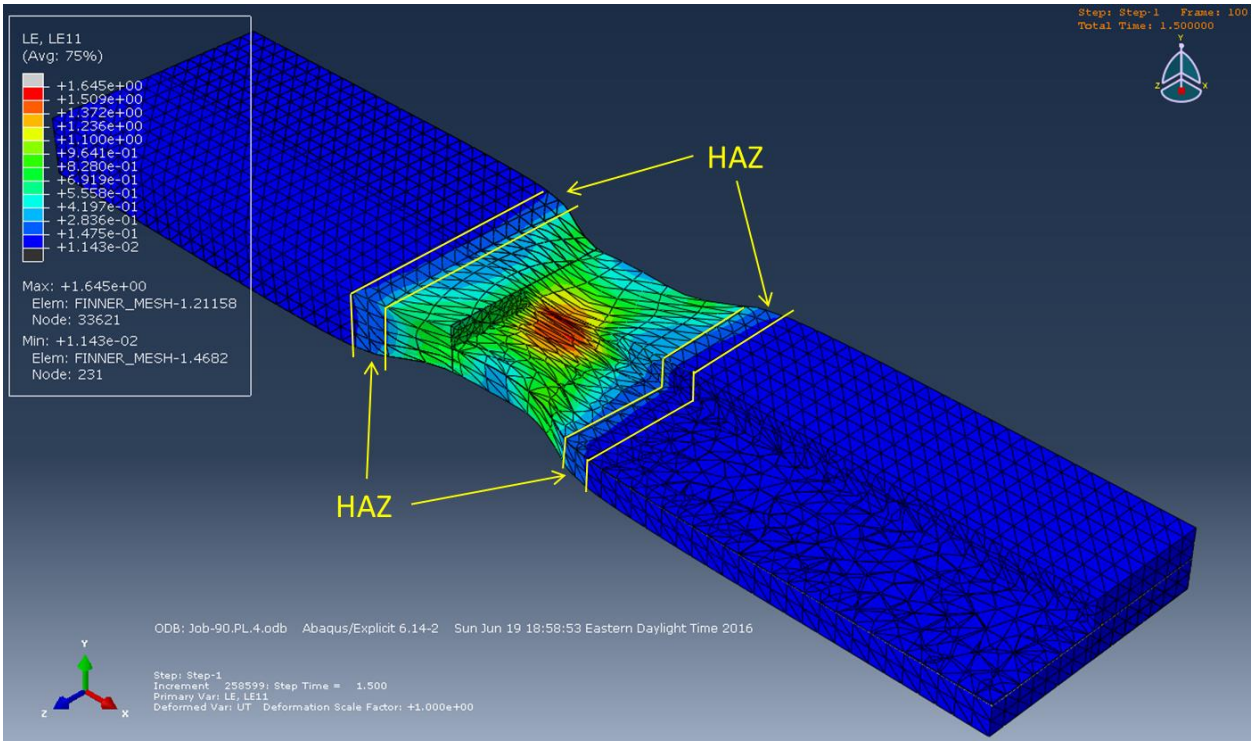
General border of the HAZ section is drawn out except for where it would obstruct the contour gradient for the viewer, such as on the border between HAZ and WELD section.



(a)

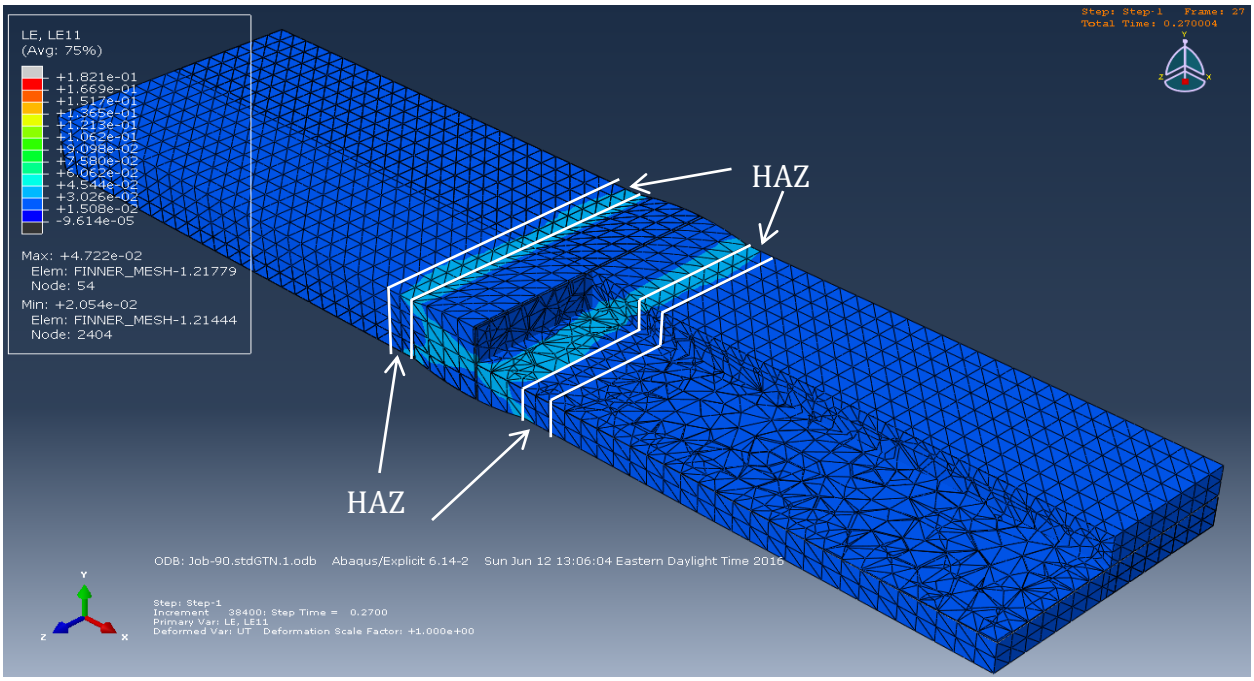


(b)

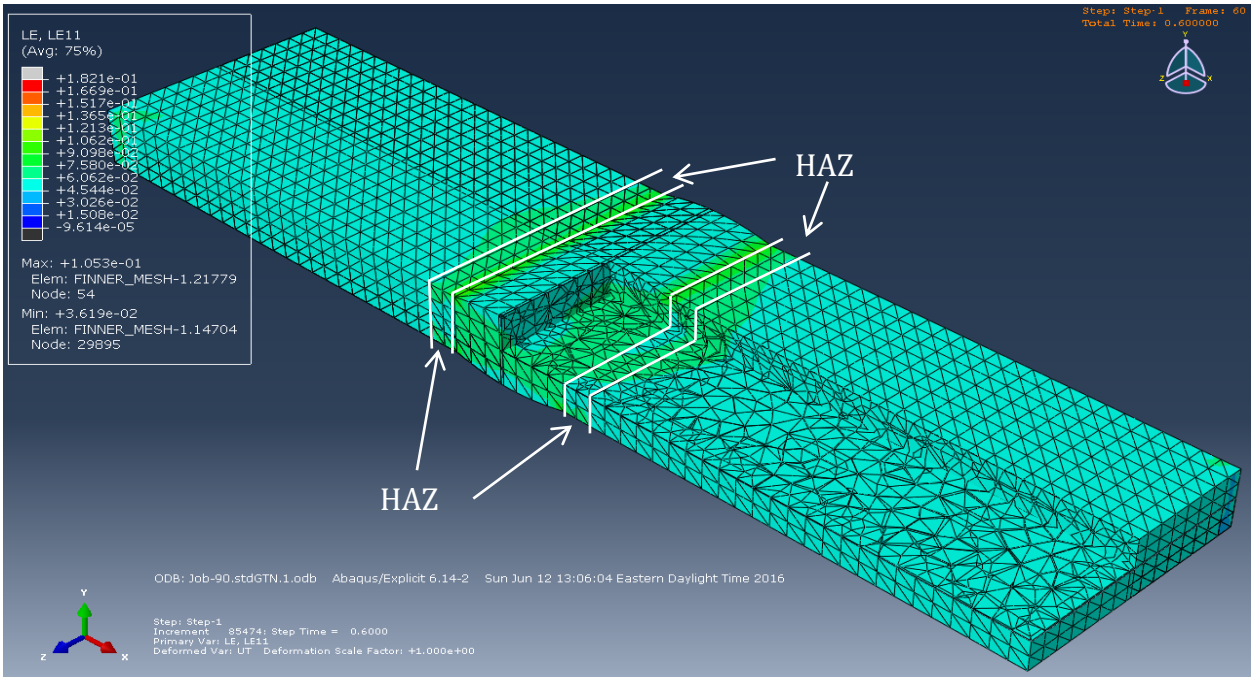


(c)

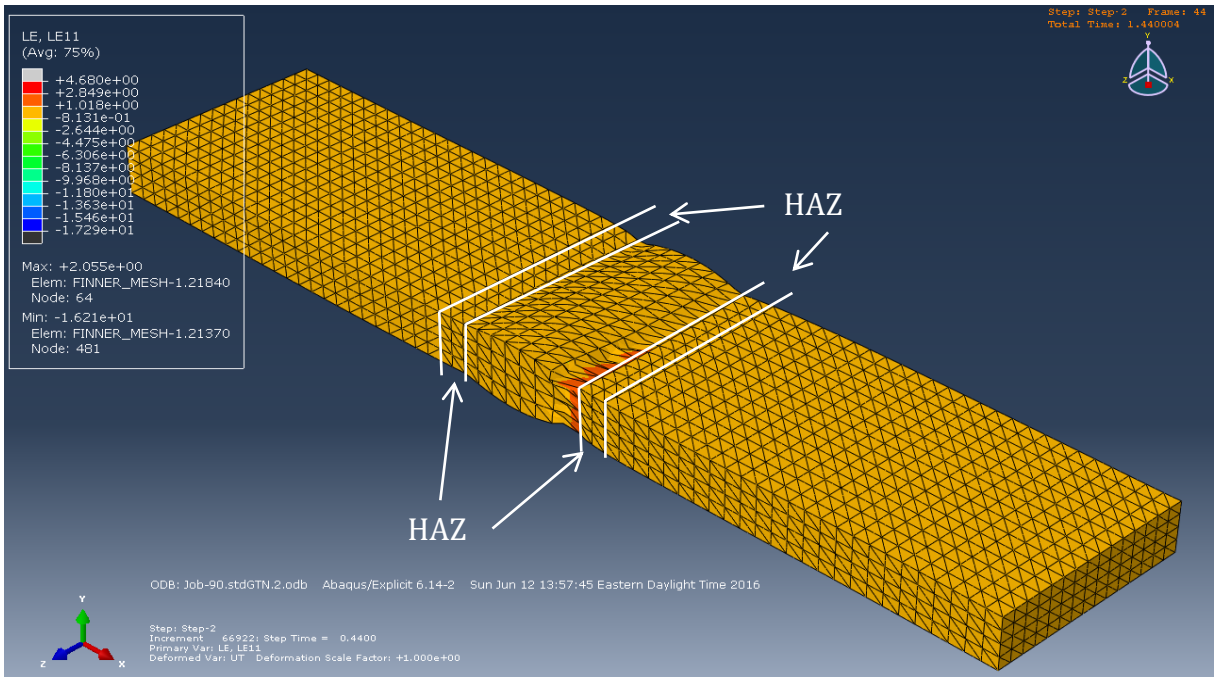
Figure 15. Progression of strain in the 90^0_welded_plate using the Power Law model. $1/8^{th}$ of the plate was cut away for visualization purpose only. (a) Strain first increase at the boarder of WELD and HAZ but in the WELD section. (b) Higher strain concentration starts to develop at the center and the four corners of the WELD zone, and propagates throughout the entire weld area. (c) At failure point, highest strain occurs at the center of WELD section forming a necking failure mode.


LE, LE11
(Avg: 75%)
+1.821e-01
+1.669e-01
+1.517e-01
+1.365e-01
+1.213e-01
+1.062e-01
+9.098e-02
+7.580e-02
+6.062e-02
+4.544e-02
+3.026e-02
+1.508e-02
-9.614e-05
Max: +4.722e-02
Elem: FINNER_MESH-1.21779
Node: 54
Min: +2.054e-02
Elem: FINNER_MESH-1.21444
Node: 2404
HAZ
HAZ
ODB: Job-90.stdGTN.1.odb Abaqus/Explicit 6.14-2 Sun Jun 12 13:06:04 Eastern Daylight Time 2016
Step: Step-1
Increment 38400: Step Time = 0.2700
Primary Var: LE, LE11
Deformed Var: UT Deformation Scale Factor: +1.000e+00

(a)

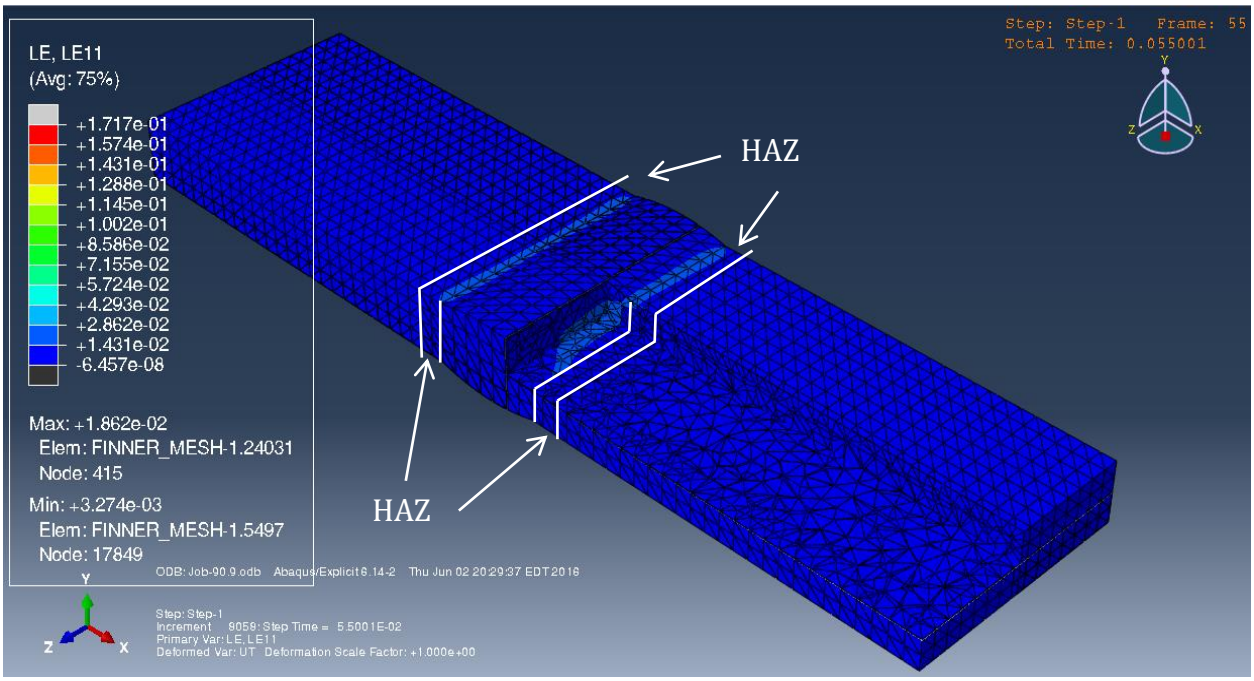

LE, LE11
(Avg: 75%)
+1.821e-01
+1.669e-01
+1.517e-01
+1.365e-01
+1.213e-01
+1.062e-01
+9.098e-02
+7.580e-02
+6.062e-02
+4.544e-02
+3.026e-02
+1.508e-02
-9.614e-05
Max: +1.053e-01
Elem: FINNER_MESH-1.21779
Node: 54
Min: +3.619e-02
Elem: FINNER_MESH-1.14704
Node: 29895
HAZ
HAZ
ODB: Job-90.stdGTN.1.odb Abaqus/Explicit 6.14-2 Sun Jun 12 13:06:04 Eastern Daylight Time 2016
Step: Step-1
Increment 85474: Step Time = 0.6000
Primary Var: LE, LE11
Deformed Var: UT Deformation Scale Factor: +1.000e+00

(b)

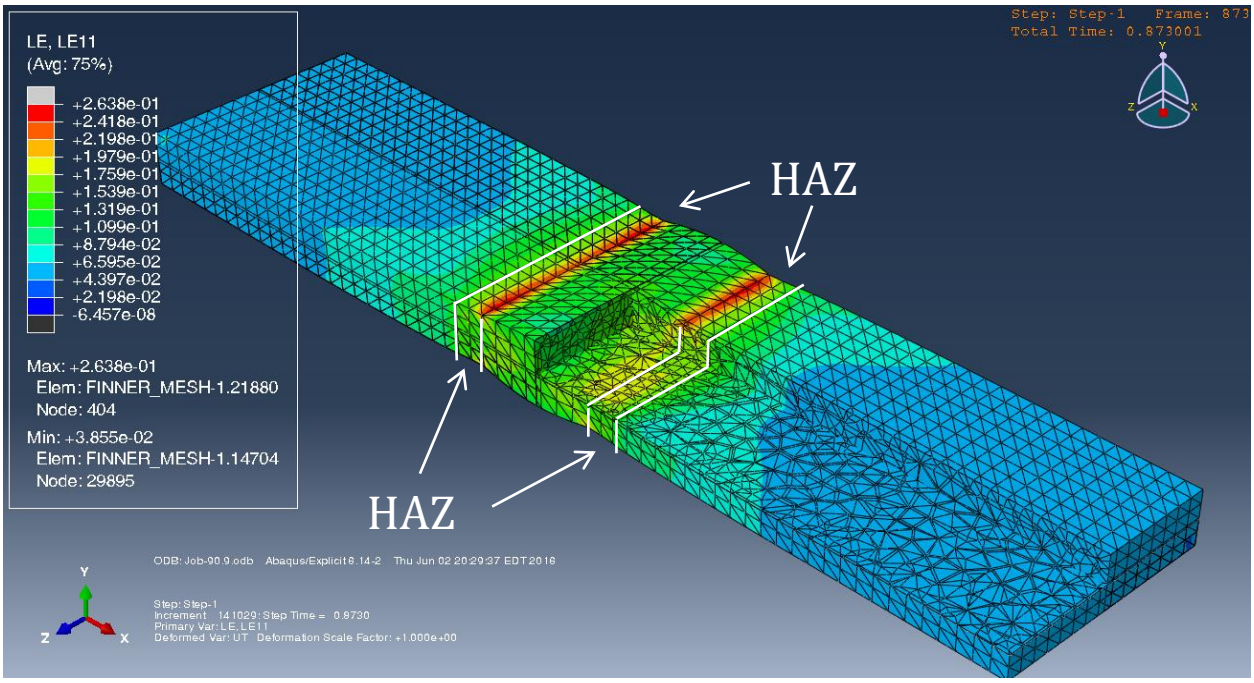


(c)

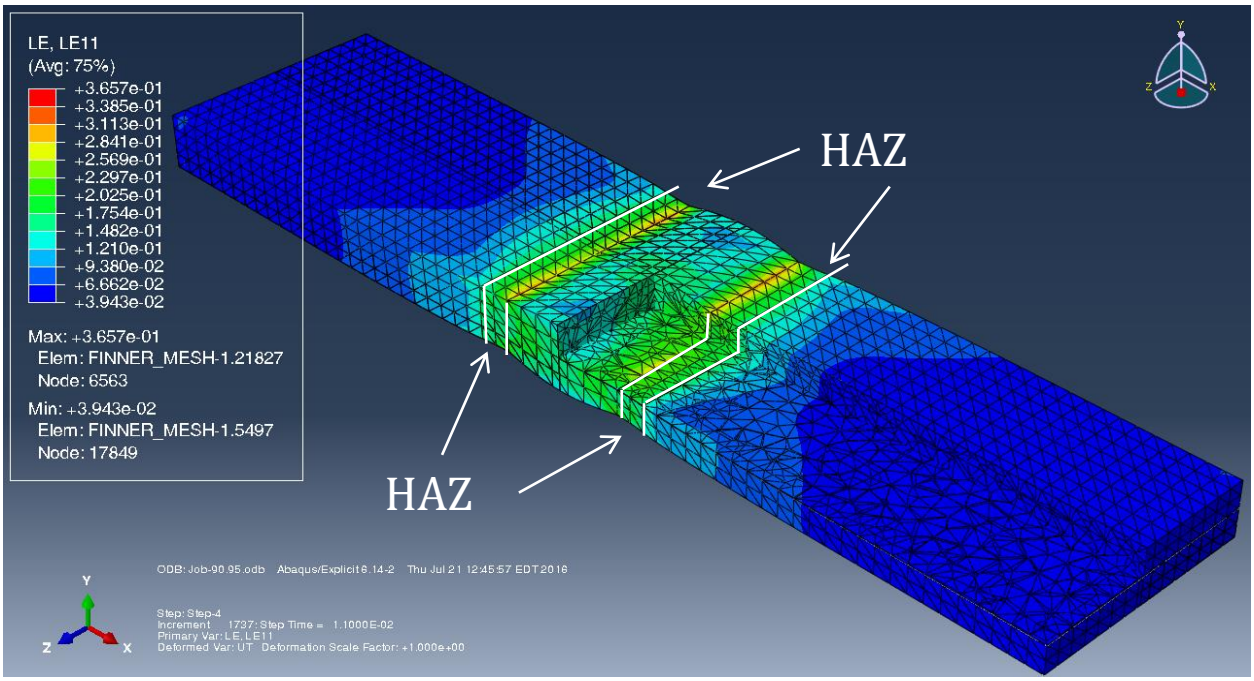
Figure 16. Progression of strain in the 90^0_welded_plate using the stdGTN model. $1/8^{th}$ of the plate was cut away for visualization purpose only. (a) High strain first occur on the inner edges of the WELD/HAZ section border but in the WELD section, (b) then propagated into the HAZ, the BASE, and the WELD section , and finally (c) at failure, highest strain occurs at the in the WELD section.


LE, LE11
(Avg: 75%)
+1.717e-01
+1.574e-01
+1.431e-01
+1.288e-01
+1.145e-01
+1.002e-01
+8.586e-02
+7.155e-02
+5.724e-02
+4.293e-02
+2.862e-02
+1.431e-02
-6.457e-08
Max: +1.862e-02
Elem: FINNER_MESH-1.24031
Node: 415
Min: +3.274e-03
Elem: FINNER_MESH-1.5497
Node: 17849
Step: Step-1 Frame: 55
Total Time: 0.055001
HAZ
HAZ
ODB: Job-90.9.odb Abaqus/Explicit 6.14-2 Thu Jun 02 20:29:37 EDT 2016
Step: Step-1
Increment 6059: Step Time = 5.5001E-02
Primary Var: LE, LE11
Deformed Var: UT Deformation Scale Factor: +1.000e+00

(a)


LE, LE11
(Avg: 75%)
+2.638e-01
+2.418e-01
+2.198e-01
+1.979e-01
+1.759e-01
+1.539e-01
+1.319e-01
+1.099e-01
+8.794e-02
+6.595e-02
+4.397e-02
+2.198e-02
-6.457e-08
Max: +2.638e-01
Elem: FINNER_MESH-1.21880
Node: 404
Min: +3.855e-02
Elem: FINNER_MESH-1.14704
Node: 29895
Step: Step-1 Frame: 873
Total Time: 0.873001
HAZ
HAZ
ODB: Job-90.9.odb Abaqus/Explicit 6.14-2 Thu Jun 02 20:29:37 EDT 2016
Step: Step-1
Increment 141029: Step Time = 0.8730
Primary Var: LE, LE11
Deformed Var: UT Deformation Scale Factor: +1.000e+00

(b)



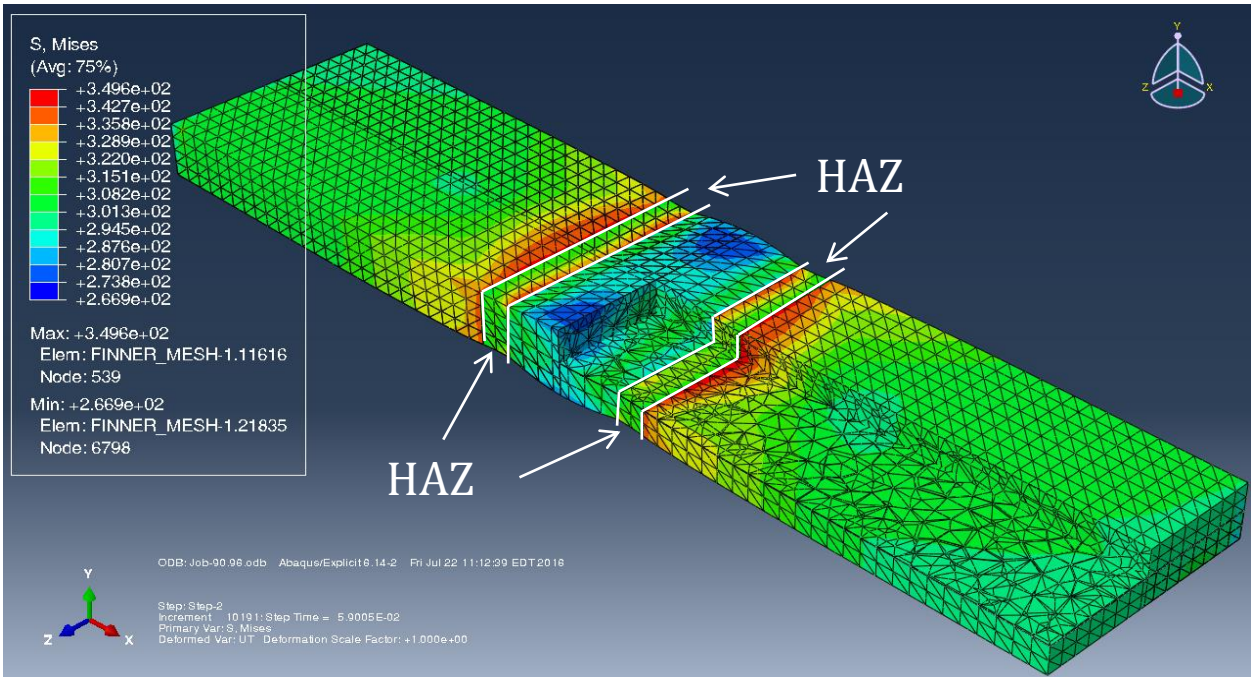
(c)

Figure 17. Progression of strain in the 90^0_welded_plate using the modGTN model. 1/8th of the plate was cut away for visualization purpose only. (a) Plastic strain initially occurring on the boarder of WELD/HAZ section border but in the HAZ section, then (b) the propagated into the HAZ section first before fully propagating through the WELD zone, and finally the (c) system failed due to plastic strain localization in the HAZ section at the HAZ/WELD section border.

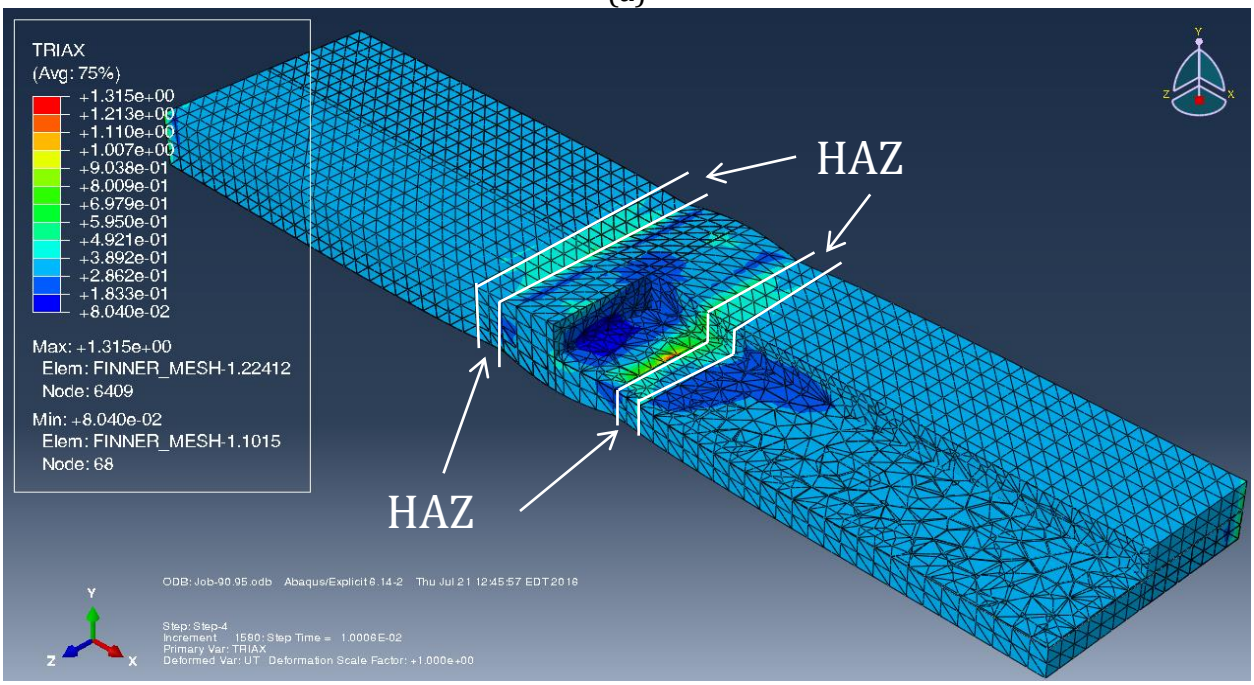
The stress distribution of the 90^0_welded_plate simulated with modGTN model was inspected next. Figure 18(a) shows the equivalent stress distribution at the failure initiation point, where maximum σ_e was 349MPa in the BASE section and minimum σ_e was 269MPa in the HAZ section. The minimum σ_e point was adjacent to the maximum σ_e point. As discussed by Bomarito and Warner [2016], plastic localization in a region of material occurs when a sub-region has been deformed to its maximum load. At this point, neighboring sub-regions can elastically unload by further deforming the failed sub-region, fueling localization and ultimately mechanical failure. Figure 18(b) shows triaxiality distribution at the same failure initiation point, where maximum T was 1.3 in the WELD section about the WELD/HAZ section border. Overall, the simulation shows deviatoric

stress domination. The stress distribution color contour plots showed similar trend with the strain distribution color contour plots at the failure initiation point. Both modGTN and stdGTN models showed similar stress distributions. During early portion of the loading, overall stress states were around and below the lower bound of the triaxiality values that were fitted in the study ($T \leq 0.4$) indicating that elements mostly experience shearing stress. In these cases, the values of the $q_1(T,\xi)$, $q_2(T,\xi)$, and $f_c(T,\xi)$ were taken from the nearest stress state that micromechanical simulation results are available. In contrast to the early loading stress-strain response, the failure prediction is not impacted by this deficiency because failure always occurred at points having values within the calibrated range(see Figure(19)). Throughout Abaqus simulation, high traxility values always gathered in the WELD section on the WELD/HAZ section border, which means that this border region is weak and is constrained by the surrounding stronger material [Ship Structure Committee, 2010]. Abaqus simulation shows that the middle center of the HAZ section has higher triaxiality than the other parts of the HAZ section during early loading. In Figure(19), representative elements and nodes were picked from each material section and their triaxiality values were plotted against their strain values. An element(at its first integration point) was picked from the very center inside of a material section, denoted as "Middle center" in green lines in Figure(19). Another element(at its first integration point) was picked from the top surface of the plate at the section borders, denoted as "- / - top surface border in red lines in Figure(19). Lastly, a node was picked from the side face of the plate (the surface where the DIC data were originally collected from), denoted as " 'DIC face' side -". These points were used for plotting in the later discussions of this section also. In the BASE section, the representative elements display an increase of T during

deformation, and a drop in T about the BASE/HAZ section border occurred upon failure. This element is in the highest stress region of the BASE section, denoted by the red line (Figure(19(a))). The growth in T in the BASE section consequently means a void growth, which is not observed in the other two material sections. The representative elements in the HAZ section remain fairly constant with distinct T values as strain increases, suggesting a variety in stress mode. However, T about the HAZ/WELD section border dropped upon failure, indicating the region experiences more shear stress. This element in the HAZ section border, denoted by the red line (Figure(19(b))), corresponds to the region where failure seemed to initiate from. In the WELD section, the representative elements have shown much variety in T development, but overall T decreases as strain increases in the critical regions. The element in the WELD section border, denoted by the red line (Figure(19(b))), also corresponds to the general region where failure seemed to initiate from, but it is in the WELD material section. Material inhomogeneity can be identified around the HAZ section as sheen in Figure 18 and Figure 19, which is in agreement with literature mentioned earlier.

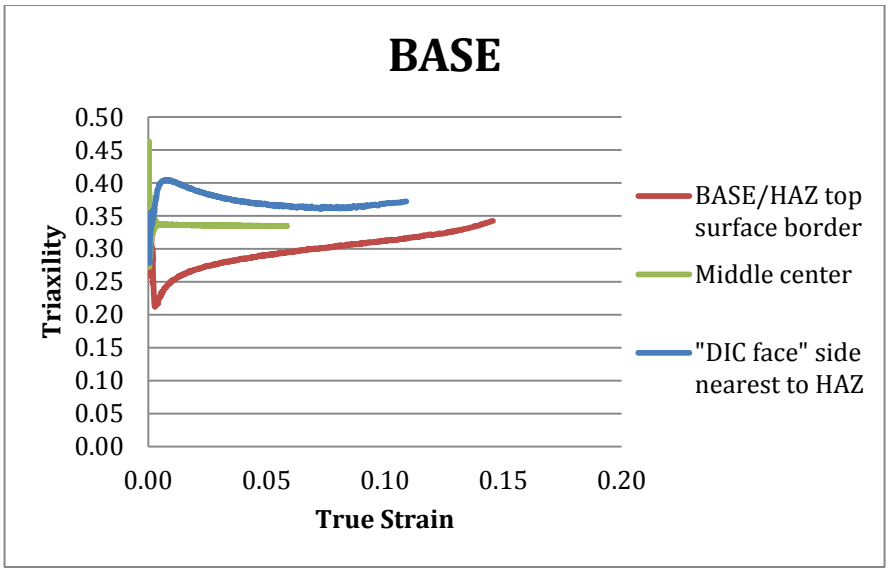


(a)

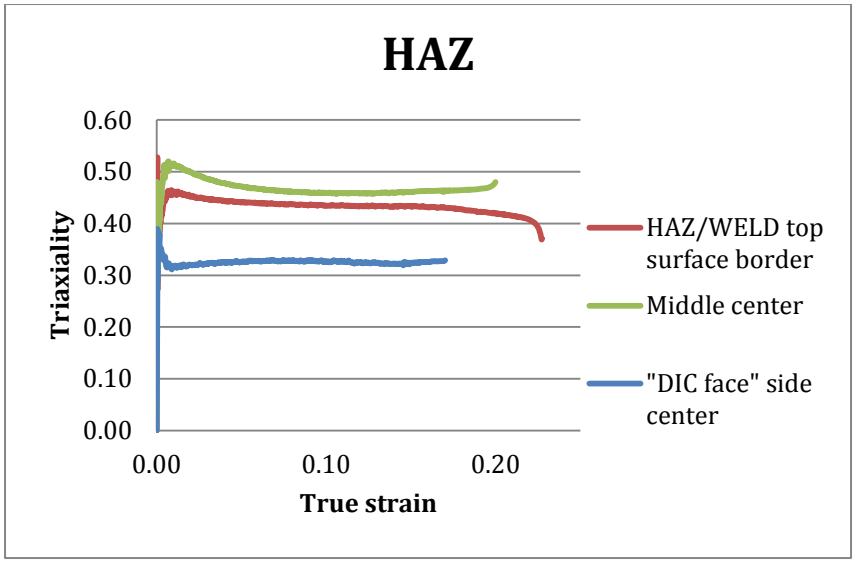


(b)

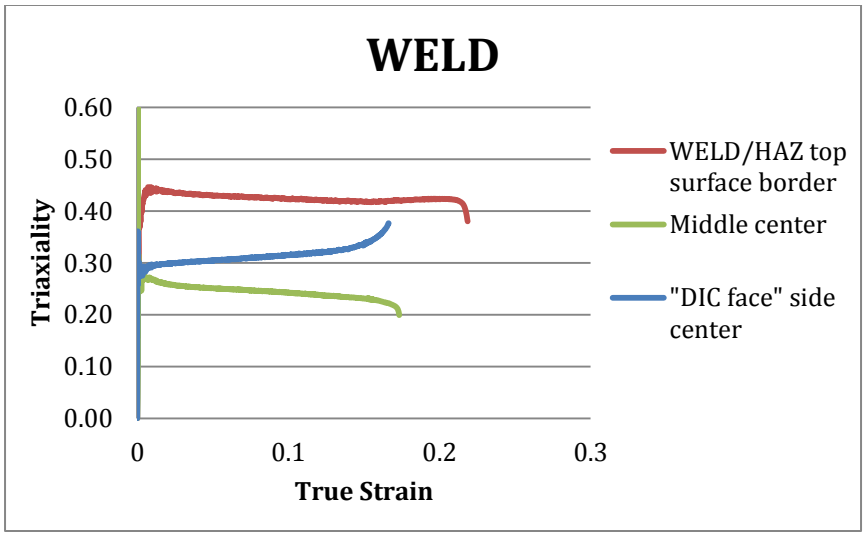
Figure 18. Stress distribution of the 90^0_welded_plate with $1/8^{th}$ of the plate removed for display. Abaqus welded plate simulations using the modGTN constitutive material model: (a) equivalent stress distribution and (b) triaxiality distribution.


BASE
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Triaxility
0.00
0.05
0.10
0.15
0.20
True Strain
BASE/HAZ top surface border
Middle center
"DIC face" side nearest to HAZ

(a)


HAZ
0.60
0.50
0.40
0.30
0.20
0.10
0.00
Triaxiality
0.00
0.10
0.20
True strain
HAZ/WELD top surface border
Middle center
"DIC face" side center

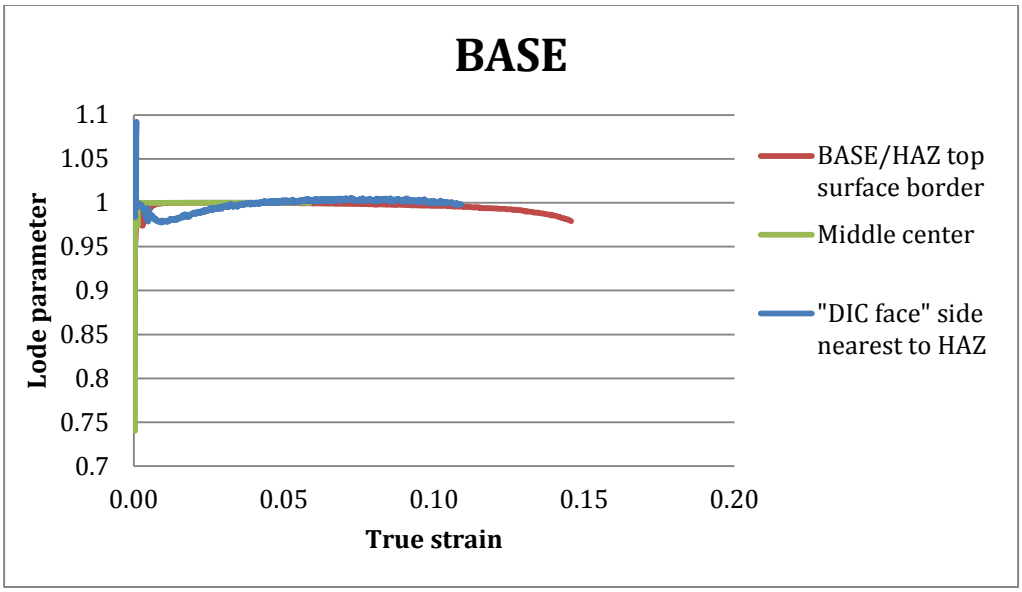
(b)



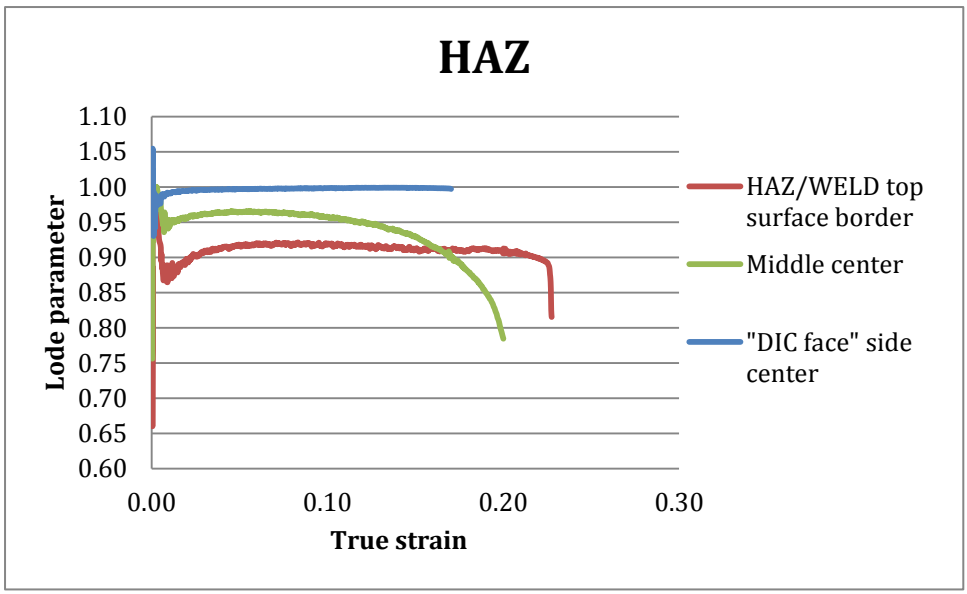
(c)

Figure 19. Triaxiality progression in each material section. In (a) BASE section, triaxiality increases overall. In (b) HAZ section, triaxiality stayed relatively constant until failure point. In (c) WELD section, triaxiality decreases overall.

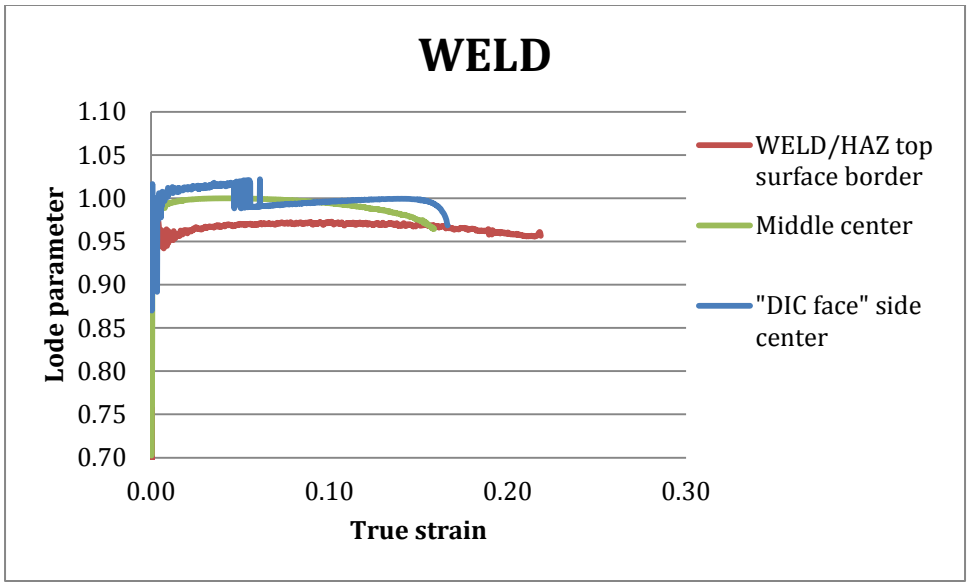
The Lode parameter of the 90^0_welded_plate using the modGTN model was explored subsequently. Figure 20 shows the progression of ξ against strain using the exact same representative elements and nodes that were plotted in the triaxiality discussion. Equivalent stress and stress tensor were extracted from these elements (at the first integration point) and nodes in each material section to compute ξ using Equation (1). Each material section clearly shows uniaxial tension ($\xi = +1$) in early loading and shear stress increase ($\xi \to 0$) as strain increases. The oscillation of ξ at the beginning of the simulation could be because loading was still propagating through all the elements in the plate over the first several time increments. Some material points in the HAZ and WELD sections showed less of a pure uniaxial tension loading, interestingly the triaxialiaty values were relatively higher around these points too. Around failure initiation, ξ about the HAZ/WELD section border decreased dramatically, showing shear domination.


BASE
Lode parameter
1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
0.00
0.05
0.10
0.15
0.20
True strain
BASE/HAZ top surface border
Middle center
"DIC face" side nearest to HAZ

(a)


HAZ
Lode parameter
1.10
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.00
0.10
0.20
0.30
True strain
HAZ/WELD top surface border
Middle center
"DIC face" side center

(b)

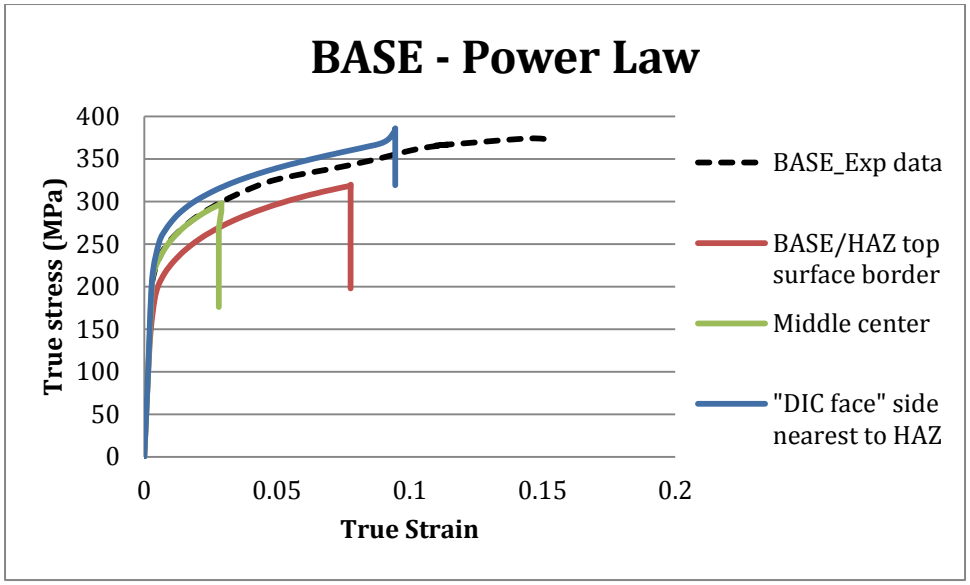


(c)

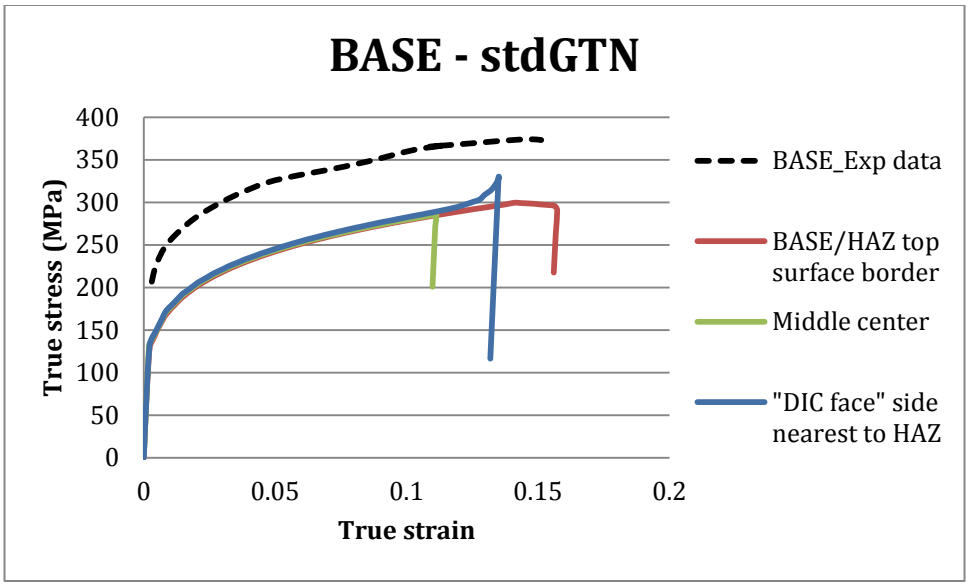
Figure 20. Lode parameter progression in the 90^0_welded_plate using the modGTN model. Overall, majority of the material points showed uniaxial tension throughout the simulation, such as in the (a) BASE section. Middle center material points in (b) HAZ section experienced less pure uniaxial tension as triaxiality was relatively higher in that section. Top surface (c) WELD material points about the WELD/HAZ border experienced less pure uniaxial tension.

After inspecting the global true stress – true strain response of the specimen scale models, the local response of the different material regions was examined in the 90^0_weld_plate among the three constitutive material models to showcase the variety of response in a material section. The same group of elements and nodes that have been plotted thus far were also used here to compare with the corresponding material DIC data (experimental data displayed in Figure 10). As a reminder, the true stress - true strain evaluated at an element used the first integration point. Different degrees of response variability are observed in each material section predicted by different constitutive material model. All of Power Law, stdGTN, and modGTN models have under predicted local BASE failure strain; this could be because the weaker materials, such as HAZ and WELD, would undertake much more plastic deformation within the system than BASE. The only

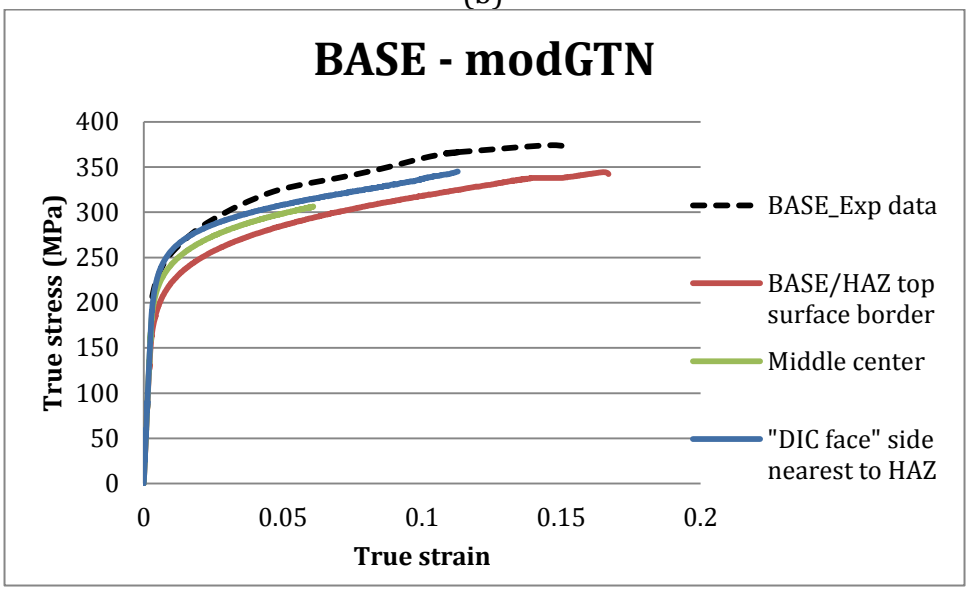
exception was modGTN BASE element about the BASE/HAZ section border, where it has a local over-prediction in ductility. Failure strains of the HAZ and WELD material sections were indeed over predicted by all of the constitutive models to different degrees. In the BASE section, the modGTN model predicted highest equivalent stresses occurring on the interface of the BASE/HAZ section border, but the Power Law model had predicted the corners of the BASE/HAZ section border as the highest equivalent stress. As simulation progress past failure point, stress relaxation in BASE elements modeled with Power Law was observed, but not in the other two constitutive models.



(a)



(b)

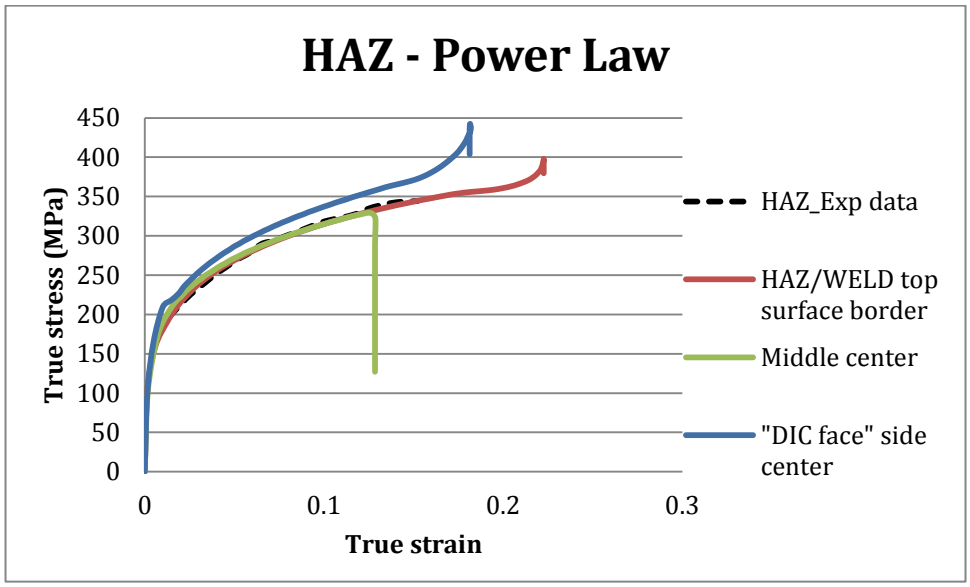


(c)

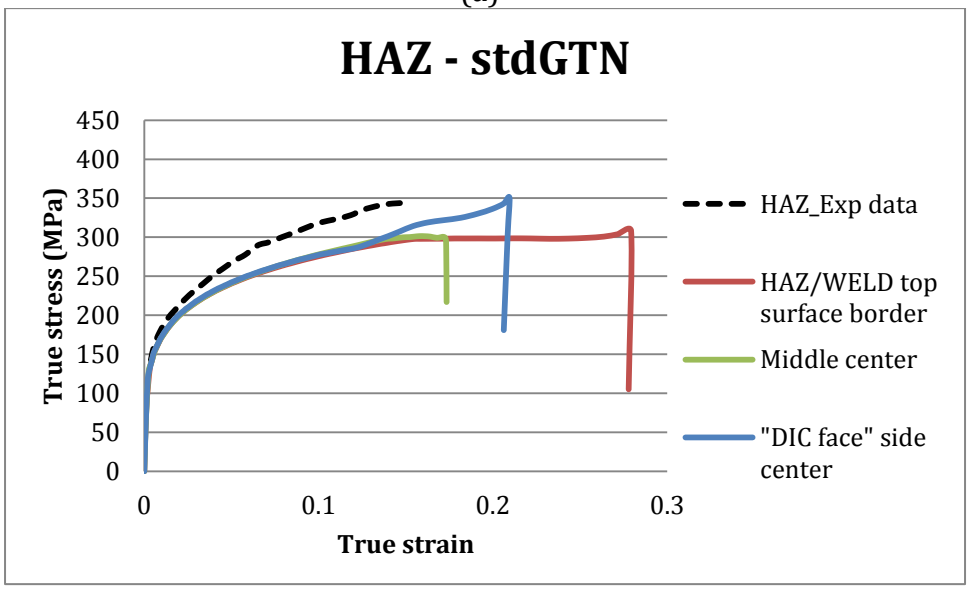
Figure 21. Local true stress – true strain response of the BASE material section in the 90^0_welded_plate simulated with (a) Power Law model, (b) stdGTN model, and (c) modGTN model.

In the HAZ material section, elements about the HAZ/WELD section border shown to have more ductility than other regions of HAZ. This region is the weakest and corresponds to where failure was first initiated. During failure propagation, the HAZ section simulated with the Power Law model continued to convex towards the BASE section, leaving the corners touching the WELD section at the highest stress and the center

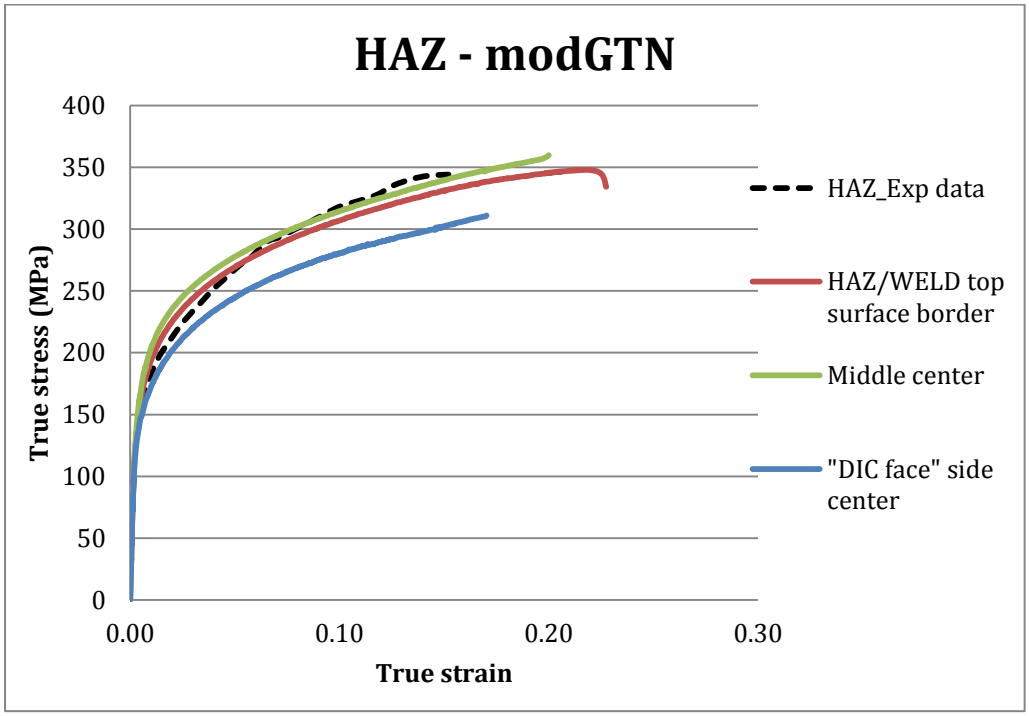
of the HAZ section at the lowest stress. While the modGTN model did not show this convex behavior at all, the stdGTN model showed some but not as obvious as the Power Law model did.



(a)



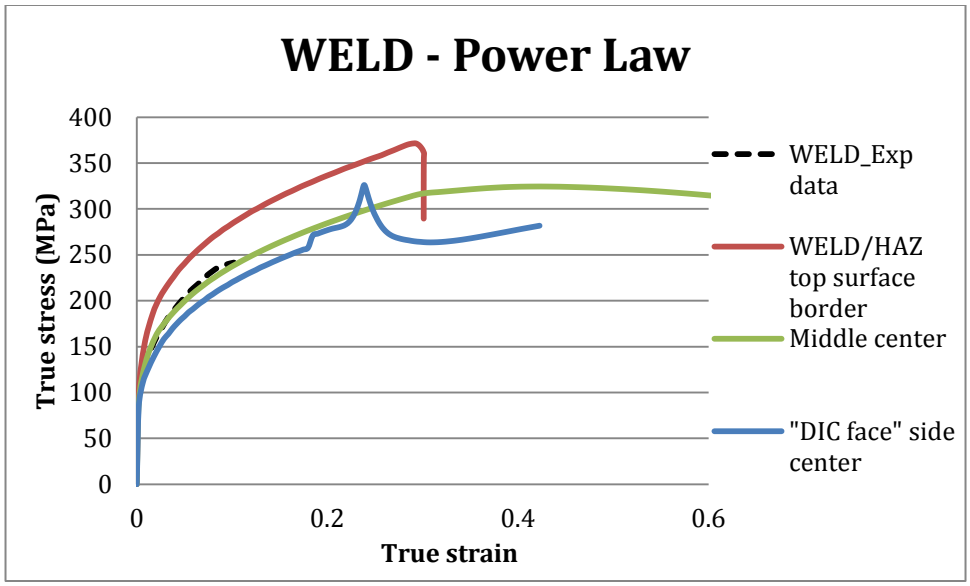
(b)



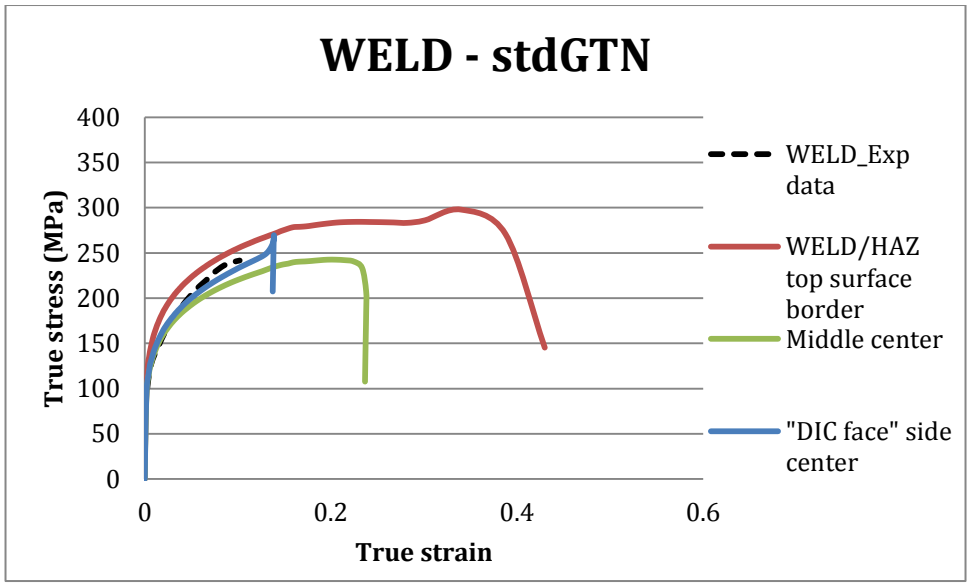
(c)

Figure 22. Local true stress – true strain response of the HAZ material section in the 90^0_welded_plate simulated with (a) Power Law model, (b) stdGTN model, and (c) modGTN model.

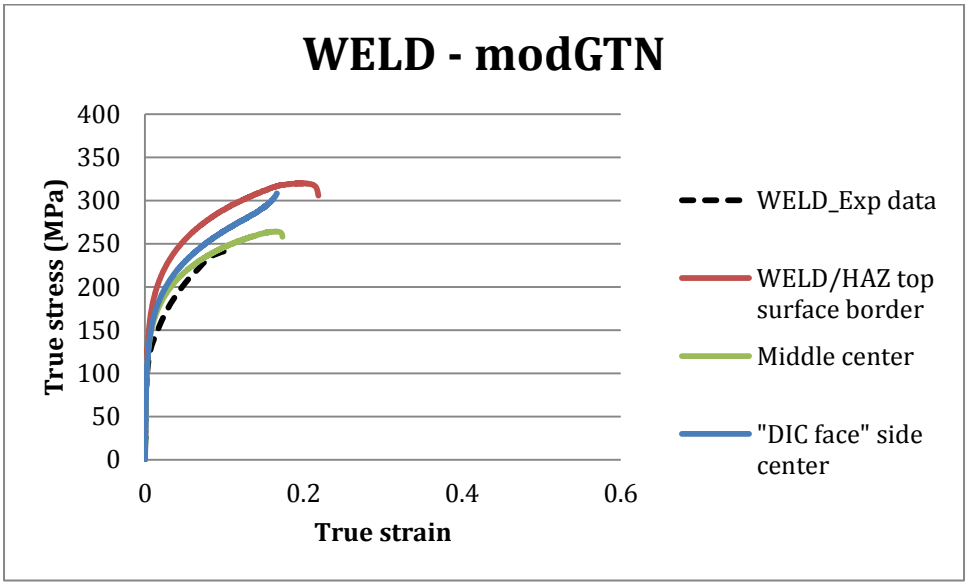
In the WELD section, all three constitutive models predicted the WELD/HAZ section border region as highest stress. All three models over predicted the WELD failure strain, but the modGTN model had closest prediction to the DIC experimental data. In the modGTN simulation, the mid-center element is very likely to lie on the 45^0 shearing band of the plate, thus have the earliest failure strain. Because the Power Law model predicted necking failure, the WELD section was deformed into a disk shape with center being the thinnest and highest stress.


WELD - Power Law
True stress (MPa)
400
350
300
250
200
150
100
50
0
0
0.2
0.4
0.6
True strain
WELD_Exp data
WELD/HAZ top surface border
Middle center
"DIC face" side center

(a)


WELD - stdGTN
True stress (MPa)
400
350
300
250
200
150
100
50
0
0
0.2
0.4
0.6
True strain
WELD_Exp data
WELD/HAZ top surface border
Middle center
"DIC face" side center

(b)



(c)

Figure 23. Local true stress – true strain response of the WELD material section in the 90^0_welded_plate simulated with (a) Power Law model, (b) stdGTN model, and (c) modGTN model.

In summary, the modGTN model predicted the most accurate material response compared to the experiments due to its high level of details that were present in the micromechanical cell model and the definition of $q_1(T,\xi)$, $q_2(T,\xi)$, and $f_c(T,\xi)$ obtained from a hierarchy approach. Overall, triaxiality observed in the 90^0 welded plate was not large, and Lode parameter portrayed uniaxial tension in the specimen. The Lode parameter did not show significant relationship with strain with the exception of the region near the HAZ/WELD section borders. When considered the effects of loading orientation, the modGTN model outperformed the stdGTN model in predicted response. However, the modGTN model still has room for improvement for capturing the material inhomogeneity around the weld region under shear dominated deformation at low triaxiality. As discussed in literature, the GTN –type model has its deficiencies under low triaxiality in prediction of void evolution [Barsoum, 2008]. The currently studied modGTN model has not

incorporated damage growth under low triaxiality for shear dominated stress states. While Power Law can predict the failure point adequately on the global true stress – true strain behavior, the predicted system deformation development was not in agreement with experimental observations at all because it is not mechanics based. The Power Law does not capture changes in triaxiality and Lode parameter throughout test in the specimen, nor does it consider loading angle on a material point. It also does not know about damage due to micro-voids in the material where void evolution is a key component in understanding ductile failure of metallic materials. So the Power Law only predicted a general failure mode of necking in the material section that had the lowest failure strain and stress. In the stdGTN model, the reasons for stress under prediction could be that the model inputs were too simple and that the $q_1, q_2,$ and f_c parameters in the porous model were constants without consideration of load orientation and current stress state at a material point. Also, the stress characteristics predicted by the stdGTN model were not determined with mechanics based response from the microscale, but rather used a more phenomological fitting approach. On the contrary, the modGTN model included all of the important factors that were missing in both the Power Law and stdGTN models. The modGTN model requires only three physics-based inputs: (1) the large deformation constitutive response of the matrix, (2) the void volume fraction represented by voids nucleated at second phase particle sties and voids from weld porosity, and void from weld porosity, and (3) the average nearest neighbor spacing of second phase particles and voids. The modGTN model has shown robust prediction capability and transferability among different materials.

7. Conclusion

While a simple model like the Power Law is adequate for a quick and easy check of specimen global response, it does not provided the correct deformation behavior information. To obtain accurate damage development in a specimen, a much more detailed model is preferred. In this study, a mechanistic model that does not depend on free fitting parameters was investigated for prediction of ductile failure of Al 5083 welded plate against experimental tests provided by the U.S. Navy. The welded plate has three distinct material sections: BASE, HAZ, and WELD. The welded plate has weld zone at 90^0 to the uniaxial loading direction. The model studied here takes only three inputs that are easily obtainable in laboratory procedures: (1) the large deformation constitutive response of the matrix material, (2) the void volume fraction represented by voids nucleated at second phase particle sties and voids from weld porosity, and (3) the average nearest neighbor spacing of second phase particle and voids. Based on these inputs, a micromechanical cell model consisting of a spherical void inside a rectangular prismatic matrix was defined. The mechanical response of each cell was simulated, examining approximately 3,000 distinct loading paths over a common range of stress states using a large deformation finite element approach. The responses of the micromechanical models were then homogenized and fed into the macromechanical models, a single material plate and a multi-material plate, to predict the total ductile failure of different systems using large deformation FE method in Abaqus /Explicit. The performance of the hierarchal mechanistic model was compared to other common phenomenological models used in the industry: Power Law model and the stdGTN model. The primary finding of this thesis is that the model studied

here has robust capability to predict the failure strain of various tempered Al 5083 alloy laboratory specimens to within 15.27%, owing to the details of the model. This finding is remarkable considering the transferability of a mechanistic model to multiple structural alloys. However it should be noted that, the computation cost across the three tested constitutive models were dramatically different. The simple Power Law model used time on the order tens of seconds, the stdGTN model used on the order of tens of minutes, and modGTN model used on the order of tens of days.

Looking forward, more welded plate geometries could be tested using this modGTN model. Another welding orientation that has been thoroughly characterized by experiment has the weld at 30^0 to the loading direction, and it would interest to see if similar strain localization would be predicted in this geometry by the modGTN model. The modGTN model could also be tested on other metallic materials beyond aluminum. The incorporation of shear deformation could better the performance of this model since the geometry modeled here failed on shear dominated loading. Lastly, if desire for simulating welded specimen with the modGTN model continues, the addition of a thermal effect model can be incorporated. The change in microstructures due to annealing could be determined by a thermal effect model instead of assuming uniformity in microscale models throughout a material section, as was done in the modGTN model assumptions.

References

Ardell, A.J., 1972. The effect of volume fraction on particle coarsening: theoretical considerations. *Acta metallurgica*, *20*(1), pp.61-71.

Bansal, P.P. and Ardell, A.J., 1972. Average nearest-neighbor distances between uniformly distributed finite particles. *Metallography*, *5*(2), pp.97-111.

Barsoum, I., 2008. The effect of stress state in ductile failure.

Barsoum, I., Faleskog, J. and Pingle, S., 2011. The influence of the lode parameter on ductile failure strain in steel. *Procedia Engineering*, 10, pp.69-75.

Becker, R., Needleman, A., Richmond, O. and Tvergaard, V., 1988. Void growth and failure in notched bars. *Journal of the Mechanics and Physics of Solids*, *36*(3), pp.317-351.

Bear, J. and Bachmat, Y., 2012. *Introduction to modeling of transport phenomena in porous media* (Vol. 4). Springer Science & Business Media.

Benzerga, A.A. and Leblond, J.B., 2010. Ductile fracture by void growth to coalescence. *Advances in Applied Mechanics*, *44*, pp.169-305.

Bomarito, G.F. and Warner, D.H., 2015. Micromechanical investigation of ductile failure in Al 5083-H116 via 3D unit cell modeling. *Journal of the Mechanics and Physics of Solids*, *74*, pp.97-110.

Bomarito, G.F. and Warner, D.H., 2016. Predicting the mechanical failure of Al5083-H116 specimens with a physics-based model and no free fitting parameters. *International Journal of Scientific Study*. In review.

Collette, M., 2009. SR-1460 Literature Review.

Dullien, F.A., 2012. *Porous media: fluid transport and pore structure*. Academic press.

Faleskog, J., Gao, X. and Shih, C.F., 1998. Cell model for nonlinear fracture analysis–I. Micromechanics calibration. *International Journal of Fracture*,*89*(4), pp.355-373.

Francis, J.D., 2002. *Welding simulations of aluminum alloy joints by finite element analysis* (Doctoral dissertation, Virginia Polytechnic Institute and State University).

Gao, X., Zhang, T., Hayden, M. and Roe, C., 2009. Effects of the stress state on plasticity and ductile failure of an aluminum 5083 alloy. *International Journal of Plasticity*, *25*(12), pp.2366-2382.

Gurson, A.L., 1977. Continuum theory of ductile rupture by void nucleation and growth: Part I—Yield criteria and flow rules for porous ductile media. *Journal of engineering materials and technology*, *99*(1), pp.2-15.

Hakem, M., Lebaili, S., Miroud, J., Bentaleb, A. and Toukali, S., 2012. Welding and characterization of 5083 aluminum alloy. *METAL23, Brno, Czech Republic, EU*, *25*(5).

Horstemeyer, M.F., 2009. Multiscale modeling: a review. *In Practical aspects of computational chemistry* (pp. 87-135). Springer Netherlands.

Kim, J., Gao, X. and Srivatsan, T.S., 2004. Modeling of void growth in ductile solids: effects of stress triaxiality and initial porosity. *Engineering Fracture Mechanics*, *71*(3), pp.379-400.

Kinsey, B. and Wu, X. eds., 2011. *Tailor welded blanks for advanced manufacturing*. Elsevier.

Kiran, R. and Khandelwal, K., 2014. Fast-to-compute weakly coupled ductile fracture model for structural steels. *Journal of Structural Engineering*, *140*(6), p.04014018.

Koplik, J. and Needleman, A., 1988. Void growth and coalescence in porous plastic solids. *International Journal of Solids and Structures*, *24*(8), pp.835-853.

Lindgren, L.E., 2001. Finite element modeling and simulation of welding part 1: Increased complexity. *Journal of thermal stresses*, *24*(2), pp.141-192.

Nahshon, K. and Hutchinson, J.W., 2008. Modification of the Gurson model for shear failure. *European Journal of Mechanics-A/Solids*, *27*(1), pp.1-17.

Needleman, A. and Tvergaard, V., 1984. An analysis of ductile rupture in notched bars. *Journal of the Mechanics and Physics of Solids*, *32*(6), pp.461-490.

Pardoen, T., Scheyvaerts, F., Simar, A., Tekoğlu, C. and Onck, P.R., 2010. Multiscale modeling of ductile failure in metallic alloys. *Comptes Rendus Physique*, *11*(3), pp.326-345.

Park, K.T., Myung, S.H., Shin, D.H. and Lee, C.S., 2004. Size and distribution of particles and voids pre-existing in equal channel angular pressed 5083 Al alloy: their effect on cavitation during low-temperature superplastic deformation. *Materials Science and Engineering: A*, *371*(1), pp.178-186.

Puli, U. and Rajvanshi, A.K., 2012. An image analysis technique for determination of void fraction in subcooled flow boiling of water in horizontal annulus at high pressures. *International Journal of Heat and Fluid Flow*, *38*, pp.180-189.

Puttick, K.E., 1959. Ductile fracture in metals. *Philosophical magazine*,*4*(44), pp.964-969.

Rice, J.R. and Tracey, D.M., 1969. On the ductile enlargement of voids in triaxial stress fields∗. *Journal of the Mechanics and Physics of Solids*, *17*(3), pp.201-217.

Scheck, C. and Zupan, M., 2011. Ductile fracture evaluated through micro-computed X-ray tomography. *Scripta Materialia*, *65*(12), pp.1041-1044.

Scott, M.H. and Gittos, M.F., 1983. Tensile and toughness properties of Arc-welded 5083 and 6082 aluminum alloys. *Welding Journal*, *62*(9), pp.243s-252s.

Ship Structure Committee, 2010. Effect of Welded Properties on Aluminum Structures. SSC – 4 report.

Skube, S., 2009. *Large strain compression testing of ductile materials at quasi-static and dynamic strain rates* (Doctoral dissertation, PURDUE UNIVERSITY).

Slimane, A., Bouchouicha, B., Benguediab, M. and Slimane, S.A., 2015. Parametric study of the ductile damage by the Gurson–Tvergaard–Needleman model of structures in carbon steel A48-AP. *Journal of Materials Research and Technology, 4*(2), pp.217-223.

Tvergaard, V., 1982. On localization in ductile materials containing spherical voids. *International Journal of Fracture*, *18*(4), pp.237-252.

Tvergaard, V. and Needleman, A., 1984. Analysis of the cup-cone fracture in a round tensile bar. *Acta metallurgica*, *32*(1), pp.157-169.

U.S. Navy. "UCSB Weld Characterization". U.S. Navy Surface Warfare Center, Bethesda, MD. 2009.

Vander Voort, G.F., 1991. MiCon 90: advances in video technology for microstructural control. American Society for Testing and Materials.

Winzer, R. and Glinicka, A., 2011. The static and dynamic compressive behaviour of selected aluminium alloys. *Engineering transactions*, *59*(2), pp.85-100.

Zhang, Z.L., Thaulow, C. and Ødegård, J., 2000. A complete Gurson model approach for ductile fracture. *Engineering Fracture Mechanics, 67*(2), pp.155-168.

Zhang, Z.L., Ødegård, J., Myhr, O.R. and Fjaer, H., 2001. From microstructure to deformation and fracture behaviour of aluminium welded joints–a holistic modelling approach. *Computational materials science*, *21*(3), pp.429-435.

Zhou, J., Gao, X., Sobotka, J.C., Webler, B.A. and Cockeram, B.V., 2014. On the extension of the Gurson-type porous plasticity models for prediction of ductile fracture under shear-dominated conditions. *International Journal of Solids and Structures*, *51*(18), pp.3273-3291.

Appendix A

User subroutine VUMAT of the modGTN model used in Abaqus /Explicit

```
      SUBROUTINE VUMAT(
C     Read only -
     1 NBLOCK, NDIR, NSHR, NSTATEV, NFIELDV, NPROPS, LANNEAL,
     2 STEPTIME, TOTALTIME, DT, CMNAME, COORDMP, CHARLENGTH,
     3 PROPS, DENSITY, STRAININC, RELSPININC,
     4 TEMPOLD, STRETCHOLD, DEFGRADOLD, FIELDOLD,
     5 STRESSOLD, STATEOLD, ENERINTERNOLD, ENERINELASOLD,
     6 TEMPNEW, STRETCHNEW, DEFGRADNEW, FIELDNEW,
     7 STRESSNEW, STATENEW, ENERINTERNNEW, ENERINELASNEW)
C   VUMAT function inputs above
      INCLUDE 'VABA_PARAM.INC'
C  VUMAT inputs dimension below

      DIMENSION PROPS(NPROPS), DENSITY(NBLOCK), COORDMP(NBLOCK),
     1 CHARLENGTH(NBLOCK), STRAININC(NBLOCK, NDIR+NSHR),
     2 RELSPININC(NBLOCK, NSHR), TEMPOLD(NBLOCK),
     3 STRETCHOLD(NBLOCK, NDIR+NSHR),DEFGRADOLD(NBLOCK,NDIR+NSHR+NSHR),
     4 FIELDOLD(NBLOCK, NFIELDV), STRESSOLD(NBLOCK, NDIR+NSHR),
     5 STATEOLD(NBLOCK, NSTATEV), ENERINTERNOLD(NBLOCK),
     6 ENERINELASOLD(NBLOCK), TEMPNEW(NBLOCK),
     7 STRETCHNEW(NBLOCK,
NDIR+NSHR),DEFGRADNEW(NBLOCK,NDIR+NSHR+NSHR),
     8 FIELDNEW(NBLOCK, NFIELDV), STRESSNEW(NBLOCK,NDIR+NSHR),
     9 STATENEW(NBLOCK, NSTATEV), ENERINTERNNEW(NBLOCK),
     1 ENERINELASNEW(NBLOCK)
C
CKL----------------------------------------------------------------
C       program VUMAT

       double precision rho,G,E,nu,K,cspeed,n
       double precision properties(8),pmat(8),q00(22),q01(22)
       double precision  Tlb, Tub, Llb, Lub, Tdisc, Ldisc
       integer dummy, Tnum, Lnum,II
       double precision q_ones(88),q_twos(88),f_cs(88),f_fs(88)
       double precision Fn(9),F0(9),F1(9),Fp0(9),Fp1(9),P(9),SS(9)
       double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)
C       dimension a_q1((Tnum-1)*(Lnum-1)*16),a_q2((Tnum-1)*(Lnum-1)*16),
C     1  a_fc((Tnum-1)*(Lnum-1)*16),a_ff((Tnum-1)*(Lnum-1)*16)
C-----------------------------------------------------------------
      CHARACTER*8 CMNAME
      PARAMETER( ZERO = 0.D0, ONE = 1.D0, TWO = 2.D0, THREE = 3.D0,
     1 THIRD = 1.D0/3.D0, HALF = .5D0, TWO_THIRDS = 2.D0/3.D0,
     2 THREE_HALFS = 1.5D0 )
CKL----------------------------------------------------------------

       IF(CMNAME(1:4).eq."BASE")then
       open(unit=13,file='~/BASE.dat',
     1 status='unknown')
       read(13,*)dummy
       read(13,*) properties(1),E,nu,properties(5),
     1 properties(6),properties(7),properties(8)

      rho= properties(1)
      G = E/2./(1+nu)
      K = E/3./(1-2*nu)
      properties(2)= G
      properties(3)= K
      cspeed = sqrt((K+4./3*G)/rho)
      properties(4)= cspeed
      read(13,*) Tnum, Tlb, Tub
      read(13,*) Lnum, Llb, Lub
      Tdisc = (Tub-Tlb)/(DBLE(Tnum)-1.0)
      Ldisc = (Lub-Llb)/(DBLE(Lnum)-1.0)


      read(13,*) dummy
      DO row = 1, Tnum
       READ (13,*)( q_ones((row-1)*Lnum+col), col=1,Lnum)
      END DO
      read(13,*) dummy
      DO row = 1, Tnum
       READ (13,*)( q_twos((row-1)*Lnum+col), col=1,Lnum)
      END DO

      read(13,*) dummy
      DO row = 1, Tnum
       READ (13,*)( f_cs((row-1)*Lnum+col), col=1,Lnum)
      END DO

      read(13,*) dummy
      DO row = 1, Tnum
       READ (13,*)( f_fs((row-1)*Lnum+col), col=1,Lnum)
      END DO

      close(13)
C  //calculate A's for bicubic splines
      call calculateBicubicAs(a_q1,q_ones,Lnum,Tnum)
      call calculateBicubicAs(a_q2,q_twos,Lnum,Tnum)
      call calculateBicubicAs(a_fc,f_cs,Lnum,Tnum)
      call calculateBicubicAs(a_ff,f_fs,Lnum,Tnum)

C-----------------------------------------------------------------
```

```
C If stepTime equals to zero, assume the material pure elastic and use
C initial elastic modulus
C
C
      DO II = 1,NBLOCK

CKL----------------------------------------------------------------------
    P(1)=STATEOLD(II,14)
    P(2)=STATEOLD(II,15)
    P(3)=STATEOLD(II,16)
    P(4)=STATEOLD(II,17)
    P(5)=STATEOLD(II,18)
    P(6)=STATEOLD(II,19)
    P(7)=STATEOLD(II,20)
    P(8)=STATEOLD(II,21)
    P(9)=STATEOLD(II,22)
    F0(1)= DEFGRADOLD(II,1)
    F0(2)= DEFGRADOLD(II,4)
    F0(3)= DEFGRADOLD(II,9)
    F0(4)= DEFGRADOLD(II,7)
    F0(5)= DEFGRADOLD(II,2)
    F0(6)= DEFGRADOLD(II,5)
    F0(7)= DEFGRADOLD(II,6)
    F0(8)= DEFGRADOLD(II,8)
    F0(9)= DEFGRADOLD(II,3)
    F1(1)= 1.
    F1(2)= 0.
    F1(3)= 0.
    F1(4)= 0.
    F1(5)= 1.
    F1(6)= 0.
    F1(7)= 0.
    F1(8)= 0.
    F1(9)= 1.
    Fn(1)= DEFGRADNEW(II,1)
    Fn(2)= DEFGRADNEW(II,4)
    Fn(3)= DEFGRADNEW(II,9)
    Fn(4)= DEFGRADNEW(II,7)
    Fn(5)= DEFGRADNEW(II,2)
    Fn(6)= DEFGRADNEW(II,5)
    Fn(7)= DEFGRADNEW(II,6)
    Fn(8)= DEFGRADNEW(II,8)
    Fn(9)= DEFGRADNEW(II,3)
    Fp1(1)= 1.
    Fp1(2)= 0.
    Fp1(3)= 0.
    Fp1(4)= 0.
    Fp1(5)= 1.
    Fp1(6)= 0.
    Fp1(7)= 0.
    Fp1(8)= 0.
    Fp1(9)= 1.
    q01(1)=STATEOLD(II,1)
    q01(2)=STATEOLD(II,2)
    q01(3)=STATEOLD(II,3)
    q01(4)=STATEOLD(II,4)
    q01(5)=STATEOLD(II,5)
    q01(6)=STATEOLD(II,6)
    q01(7)=STATEOLD(II,7)
    q01(8)=STATEOLD(II,8)
    q01(9)=STATEOLD(II,9)
    q01(10)=STATEOLD(II,10)
    q01(11)=STATEOLD(II,11)
    q01(12)=STATEOLD(II,12)
    q01(13)=STATEOLD(II,13)

C  /* compute gradient of deformation */
    do i=1,9
     F1(i)=Fn(i)
    end do

C  /* extract the plastic part of the gradient of deformation */
    do i=1,9
     Fp1(i)=Fp1(i)+q01(i+1)  !plastic def grad = q0[2-10]
    end do

C  /* initialize internal variables */
    do i=1,9
     Fp0(i)=Fp1(i)
    end do
    do i=1,13
     q00(i)=q01(i)
    end do

    do i=1,9
     SS(i)=0. !what is SS() doing here??
    end do

    call update(properties,F0,F1,Fp0,Fp1,q00,q01,P,SS,
   1  Tlb,Tub,Llb,Lub,Tdisc,Ldisc,Tnum,Lnum,a_q1,a_q2,a_fc,a_ff)


C  /* put-back plastic Fp in internal array */
C  P() is updated so reset STATENEW()
    STATENEW(II,14)=P(1)
    STATENEW(II,15)=P(2)
    STATENEW(II,16)=P(3)
```

```
STATENEW(II,17)=P(4)
STATENEW(II,18)=P(5)
STATENEW(II,19)=P(6)
STATENEW(II,20)=P(7)
STATENEW(II,21)=P(8)
STATENEW(II,22)=P(9)

do i=1,9
  q01(i+1)=Fp1(i)
end do
q01(2)=q01(2)-1.  !Fp_11
q01(6)=q01(6)-1.  !Fp_22
q01(10)=q01(10)-1.!Fp_33
do i=1,13
  STATENEW(II,i)=q01(i) !reset STATENEW() to the updated q01()
end do

STRESSNEW(II,1)=SS(1) !11
STRESSNEW(II,2)=SS(5) !22
STRESSNEW(II,3)=SS(9) !33
STRESSNEW(II,4)=SS(2) !12
STRESSNEW(II,5)=SS(6) !23
STRESSNEW(II,6)=SS(3) !31

END DO

ELSE IF(CMNAME(1:3).eq."HAZ")then
  open(unit=13,file='~HAZ.dat',
1 status='unknown')
  read(13,*)dummy
  read(13,*) properties(1),E,nu,properties(5),
1 properties(6),properties(7),properties(8)

rho= properties(1)
G = E/2./(1+nu)
K = E/3./(1-2*nu)
properties(2)= G
properties(3)= K
cspeed = sqrt((K+4./3*G)/rho)
properties(4)= cspeed
read(13,*) Tnum, Tlb, Tub
read(13,*) Lnum, Llb, Lub
Tdisc = (Tub-Tlb)/(DBLE(Tnum)-1.0)
Ldisc = (Lub-Llb)/(DBLE(Lnum)-1.0)

read(13,*) dummy
DO row = 1, Tnum
  READ (13,*)( q_ones((row-1)*Lnum+col), col=1,Lnum)
END DO

    read(13,*) dummy
    DO row = 1, Tnum
      READ (13,*)( q_twos((row-1)*Lnum+col), col=1,Lnum)
    END DO

    read(13,*) dummy
    DO row = 1, Tnum
      READ (13,*)( f_cs((row-1)*Lnum+col), col=1,Lnum)
    END DO

    read(13,*) dummy
    DO row = 1, Tnum
      READ (13,*)( f_fs((row-1)*Lnum+col), col=1,Lnum)
    END DO

    close(13)

C  //calculate A's for bicubic splines
    call calculateBicubicAs(a_q1,q_ones,Lnum,Tnum)
    call calculateBicubicAs(a_q2,q_twos,Lnum,Tnum)
    call calculateBicubicAs(a_fc,f_cs,Lnum,Tnum)
    call calculateBicubicAs(a_ff,f_fs,Lnum,Tnum)

C-----------------------------------------------------------------------
C If stepTime equals to zero, assume the material pure elastic and use
C initial elastic modulus
    DO II = 1,NBLOCK
CKL-----------------------------------------------------------------------
    P(1)=STATEOLD(II,14)
    P(2)=STATEOLD(II,15)
    P(3)=STATEOLD(II,16)
    P(4)=STATEOLD(II,17)
    P(5)=STATEOLD(II,18)
    P(6)=STATEOLD(II,19)
    P(7)=STATEOLD(II,20)
    P(8)=STATEOLD(II,21)
    P(9)=STATEOLD(II,22)
    F0(1)= DEFGRADOLD(II,1)
    F0(2)= DEFGRADOLD(II,4)
    F0(3)= DEFGRADOLD(II,9)
    F0(4)= DEFGRADOLD(II,7)
    F0(5)= DEFGRADOLD(II,2)
    F0(6)= DEFGRADOLD(II,5)
    F0(7)= DEFGRADOLD(II,6)
    F0(8)= DEFGRADOLD(II,8)
    F0(9)= DEFGRADOLD(II,3)
    F1(1)= 1.
    F1(2)= 0.
    F1(3)= 0.
```

```
      F1(4)= 0.
      F1(5)= 1.
      F1(6)= 0.
      F1(7)= 0.
      F1(8)= 0.
      F1(9)= 1.
      Fn(1)= DEFGRADNEW(II,1)
      Fn(2)= DEFGRADNEW(II,4)
      Fn(3)= DEFGRADNEW(II,9)
      Fn(4)= DEFGRADNEW(II,7)
      Fn(5)= DEFGRADNEW(II,2)
      Fn(6)= DEFGRADNEW(II,5)
      Fn(7)= DEFGRADNEW(II,6)
      Fn(8)= DEFGRADNEW(II,8)
      Fn(9)= DEFGRADNEW(II,3)
      Fp1(1)= 1.
      Fp1(2)= 0.
      Fp1(3)= 0.
      Fp1(4)= 0.
      Fp1(5)= 1.
      Fp1(6)= 0.
      Fp1(7)= 0.
      Fp1(8)= 0.
      Fp1(9)= 1.
      q01(1)=STATEOLD(II,1)
      q01(2)=STATEOLD(II,2)
      q01(3)=STATEOLD(II,3)
      q01(4)=STATEOLD(II,4)
      q01(5)=STATEOLD(II,5)
      q01(6)=STATEOLD(II,6)
      q01(7)=STATEOLD(II,7)
      q01(8)=STATEOLD(II,8)
      q01(9)=STATEOLD(II,9)
      q01(10)=STATEOLD(II,10)
      q01(11)=STATEOLD(II,11)
      q01(12)=STATEOLD(II,12)
      q01(13)=STATEOLD(II,13)

C     /* compute gradient of deformation */
      do i=1,9
        F1(i)=Fn(i)
      end do

C     /* extract the plastic part of the gradient of deformation */
      do i=1,9
        Fp1(i)=Fp1(i)+q01(i+1)
      end do

C     /* initialize internal variables */
      do i=1,9
        Fp0(i)=Fp1(i)
      end do
      do i=1,13
        q00(i)=q01(i)
      end do

      do i=1,9
        SS(i)=0.
      end do

      call update(properties,F0,F1,Fp0,Fp1,q00,q01,P,SS,
     1  Tlb,Tub,Llb,Lub,Tdisc,Ldisc,Tnum,Lnum,a_q1,a_q2,a_fc,a_ff)


C     /* put-back plastic Fp in internal array */
C     P() is updated so reset STATENEW()
      STATENEW(II,14)=P(1)
      STATENEW(II,15)=P(2)
      STATENEW(II,16)=P(3)
      STATENEW(II,17)=P(4)
      STATENEW(II,18)=P(5)
      STATENEW(II,19)=P(6)
      STATENEW(II,20)=P(7)
      STATENEW(II,21)=P(8)
      STATENEW(II,22)=P(9)

      do i=1,9
        q01(i+1)=Fp1(i)
      end do
      q01(2)=q01(2)-1.  !Fp_11
      q01(6)=q01(6)-1.  !Fp_22
      q01(10)=q01(10)-1.!Fp_33
      do i=1,13
        STATENEW(II,i)=q01(i) !reset STATENEW() to the updated q01()
      end do

      STRESSNEW(II,1)=SS(1) !11
      STRESSNEW(II,2)=SS(5) !22
      STRESSNEW(II,3)=SS(9) !33
      STRESSNEW(II,4)=SS(2) !12
      STRESSNEW(II,5)=SS(6) !23
      STRESSNEW(II,6)=SS(3) !31

      END DO

       ELSE IF(CMNAME(1:4).eq."WELD")then
        open(unit=13,file='~WELD.dat',
     1 status='unknown')
```

```
      read(13,*)dummy
      read(13,*) properties(1),E,nu,properties(5),
     1 properties(6),properties(7),properties(8)

      rho= properties(1)
      G = E/2./(1+nu)
      K = E/3./(1-2*nu)
      properties(2)= G
      properties(3)= K
      cspeed = sqrt((K+4./3*G)/rho)
      properties(4)= cspeed
      read(13,*) Tnum, Tlb, Tub
      read(13,*) Lnum, Llb, Lub
      Tdisc = (Tub-Tlb)/(DBLE(Tnum)-1.0)
      Ldisc = (Lub-Llb)/(DBLE(Lnum)-1.0)


      read(13,*) dummy
      DO row = 1, Tnum
       READ (13,*)( q_ones((row-1)*Lnum+col), col=1,Lnum)
      END DO
      read(13,*) dummy
      DO row = 1, Tnum
       READ (13,*)( q_twos((row-1)*Lnum+col), col=1,Lnum)
      END DO

      read(13,*) dummy
      DO row = 1, Tnum
       READ (13,*)( f_cs((row-1)*Lnum+col), col=1,Lnum)
      END DO

      read(13,*) dummy
      DO row = 1, Tnum
       READ (13,*)( f_fs((row-1)*Lnum+col), col=1,Lnum)
      END DO

      close(13)

C  //calculate A's for bicubic splines
      call calculateBicubicAs(a_q1,q_ones,Lnum,Tnum)
      call calculateBicubicAs(a_q2,q_twos,Lnum,Tnum)
      call calculateBicubicAs(a_fc,f_cs,Lnum,Tnum)
      call calculateBicubicAs(a_ff,f_fs,Lnum,Tnum)

C-----------------------------------------------------------------------
C If stepTime equals to zero, assume the material pure elastic and use
C initial elastic modulus
      DO II = 1,NBLOCK

CKL----------------------------------------------------------------------
      P(1)=STATEOLD(II,14)
      P(2)=STATEOLD(II,15)
      P(3)=STATEOLD(II,16)
      P(4)=STATEOLD(II,17)
      P(5)=STATEOLD(II,18)
      P(6)=STATEOLD(II,19)
      P(7)=STATEOLD(II,20)
      P(8)=STATEOLD(II,21)
      P(9)=STATEOLD(II,22)
      F0(1)= DEFGRADOLD(II,1)
      F0(2)= DEFGRADOLD(II,4)
      F0(3)= DEFGRADOLD(II,9)
      F0(4)= DEFGRADOLD(II,7)
      F0(5)= DEFGRADOLD(II,2)
      F0(6)= DEFGRADOLD(II,5)
      F0(7)= DEFGRADOLD(II,6)
      F0(8)= DEFGRADOLD(II,8)
      F0(9)= DEFGRADOLD(II,3)
      F1(1)= 1.
      F1(2)= 0.
      F1(3)= 0.
      F1(4)= 0.
      F1(5)= 1.
      F1(6)= 0.
      F1(7)= 0.
      F1(8)= 0.
      F1(9)= 1.
      Fn(1)= DEFGRADNEW(II,1)
      Fn(2)= DEFGRADNEW(II,4)
      Fn(3)= DEFGRADNEW(II,9)
      Fn(4)= DEFGRADNEW(II,7)
      Fn(5)= DEFGRADNEW(II,2)
      Fn(6)= DEFGRADNEW(II,5)
      Fn(7)= DEFGRADNEW(II,6)
      Fn(8)= DEFGRADNEW(II,8)
      Fn(9)= DEFGRADNEW(II,3)
      Fp1(1)= 1.
      Fp1(2)= 0.
      Fp1(3)= 0.
      Fp1(4)= 0.
      Fp1(5)= 1.
      Fp1(6)= 0.
      Fp1(7)= 0.
      Fp1(8)= 0.
      Fp1(9)= 1.
      q01(1)=STATEOLD(II,1)
      q01(2)=STATEOLD(II,2)
      q01(3)=STATEOLD(II,3)
```

```
      q01(4)=STATEOLD(II,4)
      q01(5)=STATEOLD(II,5)
      q01(6)=STATEOLD(II,6)
      q01(7)=STATEOLD(II,7)
      q01(8)=STATEOLD(II,8)
      q01(9)=STATEOLD(II,9)
      q01(10)=STATEOLD(II,10)
      q01(11)=STATEOLD(II,11)
      q01(12)=STATEOLD(II,12)
      q01(13)=STATEOLD(II,13)

C     /* compute gradient of deformation */
      do i=1,9
       F1(i)=Fn(i)
      end do

C     /* extract the plastic part of the gradient of deformation */
      do i=1,9
       Fp1(i)=Fp1(i)+q01(i+1)
      end do

C     /* initialize internal variables */
      do i=1,9
       Fp0(i)=Fp1(i)
      end do
      do i=1,13
       q00(i)=q01(i)
      end do

      do i=1,9
       SS(i)=0.
      end do

      call update(properties,F0,F1,Fp0,Fp1,q00,q01,P,SS,
     1  Tlb,Tub,Llb,Lub,Tdisc,Ldisc,Tnum,Lnum,a_q1,a_q2,a_fc,a_ff)


C     /* put-back plastic Fp in internal array */
C     P() is updated so reset STATENEW()
      STATENEW(II,14)=P(1)
      STATENEW(II,15)=P(2)
      STATENEW(II,16)=P(3)
      STATENEW(II,17)=P(4)
      STATENEW(II,18)=P(5)
      STATENEW(II,19)=P(6)
      STATENEW(II,20)=P(7)
      STATENEW(II,21)=P(8)
      STATENEW(II,22)=P(9)
      do i=1,9
       q01(i+1)=Fp1(i)
      end do
      q01(2)=q01(2)-1.  !Fp_11
      q01(6)=q01(6)-1.  !Fp_22
      q01(10)=q01(10)-1.!Fp_33
      do i=1,13
       STATENEW(II,i)=q01(i) !reset STATENEW() to the updated q01()
      end do

      STRESSNEW(II,1)=SS(1) !11
      STRESSNEW(II,2)=SS(5) !22
      STRESSNEW(II,3)=SS(9) !33
      STRESSNEW(II,4)=SS(2) !12
      STRESSNEW(II,5)=SS(6) !23
      STRESSNEW(II,6)=SS(3) !31

      END DO

      END IF

      END SUBROUTINE VUMAT


C=====================================================================
=======

      subroutine update(pmat,F0,F,Fp0,Fp1,q00,q01,P1,S,
     1  Tlb,Tub,Llb,Lub,Tdisc,Ldisc,Tnum,Lnum,a_q1,a_q2,a_fc,a_ff)
      double precision Tlb,Tub,Llb,Lub,Tdisc,Ldisc,Jac
      double precision pmat(8),F0(9),F(9),Fp0(9),Fp1(9),q00(13)
      double precision P1(9),eps,sy,vvf,f_star_0
      integer i,j,ij,iters,exitcode,Tnum,Lnum
      double precision Fpinv(9),Fe0(9),Ce(9),Ee(9),ST(9),q01(13)
      double precision det,TOL,Phi,dEp(9),dvvf,deps,Fc,Ff,q1,P(9),b(8)
      logical noStress,DEBUG
      double precision S(9),Fe1(9),f_star,PhiMin,FDstep,eps_0,vvf_0
      double precision X0(8),Xmin(8),dS(9),dFp(9),dFp_inv(9)
      double precision df_star,q22,H,TOL2
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

      eps = q00(1)              ! equivalent plastic strain
      sy = pmat(5)+q00(11)      ! yield stress = initial_sy + yield stress increment
      vvf = pmat(8) + q00(12)   ! vvf = f0 + vvf increment
      f_star_0 = pmat(8) + q00(13)   ! trial f* = f0 + f* increment
      TOL = 1.0e-06

C     //Compute elastic predictor
      do i =1,9
```

```
      Fp1(i) = Fp0(i)    ! reset trial Fp from previous step Fp !*0 is trial value
    end do
    q01(1) = q00(1)
    q01(11) = q00(11)
    q01(12) = q00(12)

    call matlib_inverse(Fp1,Fpinv,det)
    call matlib_mulss(F,Fpinv,Fe0)
    call matlib_mults(Fe0,Fe0,Ce)
    call matlib_logmat(Ce,Ee)
    do i =1,9
      Ee(i)= 0.5*Ee(i)
    end do

C  //Compute trial stress ST
    call stress(ST,Ee,pmat)
C  /* Test plasticity */
    noStress = .false.
    TOL2=TOL*sy

    call calculatePhi(ST,Fe0,sy,f_star_0,noStress,Phi,Tub,Tlb,
   1  Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff) !noStress is reset here


    if (noStress.neqv..false.)then
      ST = (/0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/)
      do i = 1,9
        Fp1(i) = F(i)
      end do

    else if (Phi > TOL2 )then  !! TOL2 = 2e-8*sy
C   /* Plastic correction */
C     print *, "***Plasticity***"

C   //initialize NR vars
     dEp=(/0.,0.,0.,0.,0.,0.,0.,0.,0./)
     dvvf=0.
     deps=0.

C   //initialize some helper vars
    do i = 1,9
      S(i) = ST(i)
    end do
    do i = 1,9
      Fe1(i) = Fe0(i)
    end do
    f_star=f_star_0

C   //----MODIFIED NEWTON, McDougall----
    X0 = (/0.,0.,0.,0.,0.,0.,0.,0.,0./)
    PhiMin = 999.
    FDstep=1.e-10
    eps_0 = eps
    vvf_0 = vvf
    DEBUG = .false.

    call mod_newton_mcdougall(X0,Xmin,PhiMin,TOL2,FDstep,100,
   1    iters,exitcode,pmat,ST,Fe0,f_star_0,eps_0,vvf_0,DEBUG,Tub,
   2  Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)


C   // 8 VARIABLE UPDATE
C   //update everything after solution is found
    deps = Xmin(1)
    eps  = eps_0 + deps
    dvvf = Xmin(2)
    vvf  = vvf_0 + dvvf
    dEp(1) = Xmin(3)
    dEp(5) = Xmin(4)
    dEp(9) = Xmin(5)
    dEp(2) = Xmin(6)
    dEp(4) = Xmin(6)
    dEp(3) = Xmin(7)
    dEp(7) = Xmin(7)
    dEp(6) = Xmin(8)
    dEp(8) = Xmin(8)
C   //ST
    call stress(dS,dEp,pmat)
    do i =1,9
    S(i) = ST(i) - dS(i)
    end do
C   //get new Fe1
    call matlib_expmat(dEp,dFp)
    call matlib_inverse(dFp, dFp_inv,det)
    call matlib_mulss(Fe0,dFp_inv, Fe1)


C   //f_star
    call matlib_mulss(Fe1,S,P)
    q2 = 10101.0
    call calculateNewQs(P,Fe1, q1, q2, Fc, Ff,Tub,Tlb,Tnum,
   1  Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
    df_star = 10101.0
    call updateF_star(vvf,dvvf,f_star_0,df_star,Fc,Ff,q1,f_star)
C   //sy
    call harden(pmat, eps, sy)
```

```
      DEBUG = .false.
      call calculateResiduals(b,dEp,S,Fe1,sy,f_star,deps,dvvf,vvf,
     1  DEBUG,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
         if(isnan(b(1)).eqv..true.)then
            print *, "calculateResiduals(b(1)...) = ",b(1)
            call exit(0)
         end if

      Phi=0
      do i = 1,8
         Phi=Phi+b(i)*b(i)
      end do
      Phi=Phi**0.5

      do i =1,9
         ST(i) = S(i)
      end do

      call matlib_expmat(dEp,dFp)
      call matlib_mulss(dFp,Fp0,Fp1)

      q01(1) = eps
      q01(11) = sy-pmat(5)
      q01(12) = vvf - pmat(8)
      q01(13) = f_star - pmat(8)
      end if
c       print *, "After plasticity loop"

      call matlib_inverse(Fp1,Fpinv,det)
      call matlib_mulss(F,Fpinv,Fe1)

      if (isnan(F(1)).neqv..false.)then
         print *, "***ERROR*** |F| is nan"
         call exit(0)
      end if

C  // PK2 -> PK1
      call matlib_mulss(Fe1,ST,S)
      call matlib_mulst(S,Fpinv,P1)

      if (isnan(P1(1)).neqv..false.) then
         print *, "***Error*** |P1| is nan"
          call exit(0)
      end if

c       print *, ".....exiting update()....."
c       print *, "  "
      end subroutine update
```

```
      subroutine mod_newton_mcdougall(X0,Xmin,minErr,tolerance,FDstep,
     1  maxIter,niters,exitcode,pmat,ST,Fe0,f_star_0,eps_0,vvf_0,DEBUG,
     2  Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)

c  Purpose:

c    mod_newton_mcdougall minimizes a the norm of a multivariate function using
c    an extension of the modifired Newton method suggested by Mcdougal to
c    multivariable systems.

c  Reference:

c    TJ McDougal, SJ Wotherspoon
c    Applied Mathematics Letters
c    Volume 29, 2014, pages 20-25.


c  Parameters:

c    Input, double[8] X0, the initial guess.

c    Output, double[8] Xmin, the minimizer off errFunction found.

c    Output, double *minErr, the error associated with the minimizer.

c    Input, double tolerance, the tolerance of the error function (assumes the
c    absolute minimum value is 0)

c    Input, int maxIter, the maximum number of iterations.

c    Output, int *exitcode, error indicator.
c    0, no errors detected.
c    1, iteration terminated because maxIter was exceeded without convergence.

c    Input, the other variables are used as secondary inputs to the errFunction.
      double precision X0(8),Xmin(8),minErr,tolerance,FDstep
      double precision pmat(8),dPhi_dS(9),ST(9),Fe0(9)
      double precision f_star_0,eps_0,vvf_0,alpha
      logical DEBUG,calculatePredictorJacobian
      logical calculatePreconditioner
      integer i,iter,iter_since_min,restarts,maxRestarts,Lnum
      integer backtracking_iter,maxIter,niters,exitcode,Tnum
      double precision J(64),X(8),Xt(8),dX(8),X_star(8),R(8),Rnorm
      double precision Rnormt,Rnorm_last,dPhi_dS2(9),Precond(8),PREC
      integer iter_without_improvement
      double precision Tub,Tlb,Tdisc,Lub,Llb,Ldisc,Res(8)
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)
```

```
      PREC = 1.e-16
      iter=0
      calculatePredictorJacobian = .true.
      calculatePreconditioner = .true.
      iter_since_min=0
      iter_without_improvement=0
      restarts=0
      maxRestarts=1

c     //initialize X
      do i=1,8
        X(i) = X0(i)
      end do
      exitcode= -1
      do while (exitcode<0)

      if (calculatePredictorJacobian.eqv..true.)then
        call calculateJacobian(X, J, R, FDstep, pmat,
     1           ST, Fe0, f_star_0, eps_0, vvf_0,Tub,Tlb,
     2  Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      end if
        do i = 1,8
          if(isnan(R(i)).eqv..true.)then
            print *, "R(",i,") =",R(i)
            call exit(0)
          end if
        end do

      if (calculatePreconditioner.eqv..true.)then
C       //initialize Preconditioner
        do i=1,8
          if (abs(J(i*9-8))>PREC)then
            Precond(i) = 1.0/J(i*9-8)   !
          else
            Precond(i) = 1.0
          end if
        end do
        if(DEBUG.eqv..true.)then
          print *, "Preconditioner = ", Precond
           print *, "in mod_newton_mcdougall lin603"
        end if
        calculatePreconditioner = .false.
      end if

C     //update Rnorm_last and initialize bests
      if (iter==0)then
        Rnorm_last=0
        do i=1,8
          Rnorm_last = Rnorm_last + R(i)*R(i)
        end do
        Rnorm_last = Rnorm_last**0.5
C         //initialize mins
        minErr=Rnorm_last
        do i=1,8
          Xmin(i) = X(i)
        end do
      else
        Rnorm_last = Rnorm
      end if

      if(isnan(Rnorm_last).eqv..true.)then
       print *, "Rnorm_last =",Rnorm_last
        call exit(0)
      end if

c     //solve with preconditioner
      call solve_8x8_w_precond(J,dX,R,Precond)

        do i=1,8
          X_star(i) = X(i) + 0.5*dX(i)
        end do
C         //---corrector---
        Res =(/10101.0,10101.0,10101.0,10101.0,10101.0,10101.0,
     1       10101.0,10101.0/)
        call calculateJacobian(X_star,J,Res,FDstep,pmat,ST,
     1            Fe0, f_star_0, eps_0, vvf_0,Tub,Tlb,
     2  Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
                        !
C         //solve with preconditioner
        call solve_8x8_w_precond(J,dX,R,Precond)                ! computed dX is super big
        do i=1,8
          Xt(i) = X(i) + dX(i)
        end do

      call errFunctionScalar(Xt,pmat,ST,Fe0,f_star_0,eps_0,vvf_0,Rnormt,
     1  Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
       if(isnan(Rnormt).eqv..true.)then
          print *, "Rnormt =",Rnormt
          call exit(0)
       end if

C     //line search (alternate to backtracking)
      if (Rnormt > Rnorm_last)then
         call linesearch(X,dX,Rnorm_last,Xt,Rnormt,pmat,ST,Fe0,
     1           f_star_0, eps_0, vvf_0, DEBUG,Tub,Tlb,Tnum,Tdisc,
     2           Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      end if
```

```
      Rnorm=Rnormt
      do i=1,8
         X(i) = Xt(i)
      end do

C     //check for improvement
      if (Rnorm<minErr) then
         minErr=Rnorm
         do i=1,8
            Xmin(i) = X(i)
         end do
         iter_since_min=0
      end if

      if (Rnorm>Rnorm_last)then
         iter_without_improvement = iter_without_improvement+1
      end if

C     //check for restart
      if (iter_without_improvement>5 .and. iter_since_min>5) then
         Rnorm = minErr
         do i=1,8
            X(i) = Xmin(i)
         end do
         calculatePredictorJacobian = .true.
         calculatePreconditioner = .true.
         iter_since_min = 0
         iter_without_improvement = 0
         restarts = restarts+1
      end if

C     //completion checks
      if (Rnorm<tolerance) then
         exitcode=0
      else if (iter >maxIter) then
         exitcode=1
      else if (restarts >maxRestarts) then
         exitcode=2
      end if

      iter = iter+1
      iter_since_min = iter_since_min+1
      if (DEBUG.eqv..true.)then
         print *,"Phi(",iter,")=",Rnorm,"<",tolerance,"?"
          print *, "In mod_newton_mcdougall line 701"
      end if
     end do   !out of whileloop

C    //fill output vars
      niters=iter

      end subroutine mod_newton_mcdougall

      subroutine linesearch(X,dX,Rnorm_last,Xt,Rnormt,pmat,ST,Fe0,
     1      f_star_0,eps_0,vvf_0,DEBUG,Tub,Tlb, Tnum,Tdisc,Lub,
     2      Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      double precision X(8),dX(8),Rnorm_last,Xt(8),Rnormt,pmat(8)
      double precision ST(9),Fe0(9),F_star_0,eps_0,vvf_0,Tub,Tlb
      logical DEBUG
      integer i,expansion_iter,ls_iter,Tnum,Lnum
      double precision a,b,c,d,Tdisc,Lub,Llb,Ldisc
      double precision Xa(8),Xb(8),Xc(8),Xd(8),Ra,Rb,Rc,Rd,gm
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

C     //initialize vars
      gm = 0.618033988
      a=-1.0
      b= 2.0
      c=b+gm*(a-b)
      d=a+gm*(b-a)
      do i =1,8
         Xc(i) = X(i)+c*dX(i)
         Xd(i) = X(i)+d*dX(i)
      end do

      call errFunctionScalar(Xd,pmat,ST,Fe0,f_star_0,eps_0,vvf_0,Rd,
     1  Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      if (DEBUG.eqv..true.)then
       print *, "Rd(",d,") = ",Rd
         print *, "In linesearch line 862"
      end if
      if (Rd < Rnorm_last) then
         Rnormt=Rd
         do i = 1,8
            Xt(i) = Xd(i)
         end do
         return
      end if

      call errFunctionScalar(Xc,pmat,ST,Fe0,f_star_0,eps_0,vvf_0,Rc,
     1  Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      if (DEBUG.eqv..true.)then
         print *, "Rc(",c,") = ",Rc
         print *, "In linesearch line 876"
      end if

      if (Rc< Rnorm_last)then
         Rnormt=Rc
```

```
      do i = 1,8
         Xt(i) = Xc(i)
      end do
      return
    end if

    do i = 1,8
      Xa(i) = X(i)+a*dX(i)
      Xb(i) = X(i)+b*dX(i)
    end do

    call errFunctionScalar(Xa,pmat,ST,Fe0,f_star_0,eps_0,vvf_0,Ra,
   1  Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
    if (DEBUG.eqv..true.)then
      print *, "Ra(",a,") = ",Ra
      print *, "In linesearch line 895"
    end if

    if (Ra< Rnorm_last)then
      Rnormt=Ra
      do i = 1,8
         Xt(i) = Xa(i)
      end do
      return
    end if

    call errFunctionScalar(Xb,pmat,ST,Fe0,f_star_0,eps_0,vvf_0,Rb,
   1    Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)

    if (DEBUG.eqv..true.)then
     print *, "Rb(",b,") = ",Rb
      print *, "In linesearch line 909"
    end if
    if (Rb< Rnorm_last)then
      Rnormt=Rb
      do i = 1,8
         Xt(i) = Xb(i)
      end do
      return
    end if

C   //check to see that a and b arent the min
    expansion_iter=0

    do while (MIN(Rc,Rd) > MIN(Ra,Rb))
C     //expand range around min
      if (Rb<Ra)then
         c=d
         do i = 1,8
            Xc(i) = Xd(i)
         end do
         Rc=Rd
         d=b
         do i = 1,8
            Xd(i) = Xb(i)
         end do
         Rd=Rb
         b=a+(d-a)/gm
         do i = 1,8
            Xb(i) = X(i)+b*dX(i)
         end do
         call errFunctionScalar(Xb,pmat,ST,Fe0,f_star_0,
   1          eps_0,vvf_0,Rb,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,
   2          Ldisc,a_q1,a_q2,a_fc,a_ff)
         if (DEBUG.eqv..true.) then
            print *, "Ra(",a,") = ",Ra
            print *, "Rc(",c,") = ",Rc
            print *, "Rd(",d,") = ",R
            print *, "Rb(",b,") = ",Rb
            print *, "In linesearch line 947"
         end if
      else
         d=c
         do i = 1,8
            Xd(i) = Xc(i)
         end do
         Rd=Rc
         c=a
         do i = 1,8
            Xc(i) = Xa(i)
         end do
         Rc=Ra
         a=b+(c-b)/gm;
         do i=1,8
            Xa(i) = X(i)+a*dX(i)
         end do
         call errFunctionScalar(Xa,pmat,ST,Fe0,f_star_0,
   1       eps_0,vvf_0,Ra,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,
   2       Ldisc,a_q1,a_q2,a_fc,a_ff)
         if (DEBUG.eqv..true.) then
            print *, "Ra(",a,") = ",Ra
            print *, "Rc(",c,") = ",Rc
            print *, "Rd(",d,") = ",R
            print *, "Rb(",b,") = ",Rb
            print *, "In linesearch line 972"
            call exit(0)
         end if
      end if
```

```
      if (expansion_iter>5)then
        exit
      end if
      expansion_iter = expansion_iter+1
    end do ! end while (MIN(Rc,Rd) > MIN(Ra,Rb))

C   //initialize
    Rnormt=MIN(Rc,Rd)

C   //main loop (line search)
    ls_iter=0
    do while (Rnormt>Rnorm_last)

      if (Rc<Rd) then
        b=d
        do i = 1,8
          Xb(i) = Xd(i)
        end do
        Rb=Rd
        d=c
        do i = 1,8
          Xd(i) = Xc(i)
        end do
        Rd=Rc
        c=b+gm*(a-b);
        do i =1,8
          Xc(i) = X(i)+c*dX(i)
        end do
        call errFunctionScalar(Xc,pmat,ST,Fe0,
     1       f_star_0,eps_0,vvf_0,Rc,Tub,Tlb,Tnum,Tdisc,Lub,
     2       Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
        Rnormt = Rc
      else
        a=c
        do i = 1,8
          Xa(i) = Xc(i)
        end do
        Ra=Rc
        c=d
        do i = 1,8
          Xc(i) = Xd(i)
        end do
        Rc=Rd;
        d=a+gm*(b-a)
        do i = 1,8
          Xd(i) = X(i)+d*dX(i)
        end do
        call errFunctionScalar( Xd,pmat,ST,Fe0,f_star_0,eps_0,
     1    vvf_0,Rd, Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,
     2                a_q1,a_q2,a_fc,a_ff)
        Rnormt = Rd
      end if
      if (DEBUG.eqv..true.) then
        print *, "Ra(",a,") = ",Ra
        print *, "Rc(",c,") = ",Rc
        print *, "Rd(",d,") = ",R
        print *, "Rb(",b,") = ",Rb
        print *, "In linesearch line 1029"
        call exit(0)
      end if
      if (ls_iter >15) then
        if(DEBUG.eqv..true.)then
          print *, "---"
          print *, "In linesearch line 1035"
        end if
        exit
      end if
      ls_iter = ls_iter+1
    end do

C   //set finish vals
    if (Rc<Rd) then
      Rnormt=Rc
      do i = 1,8
        Xt(i) = Xc(i)
      end do
    else
      Rnormt=Rd
      do i = 1,8
        Xt(i) = Xd(i)
      end do
    end if

    end subroutine linesearch


    subroutine calculateJacobian(x,Jac,Res,FDstep,pmat,ST,Fe0,
   1  f_star_0,eps_0,vvf_0,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,
   2  a_q2,a_fc,a_ff)
    double precision x(8),pmat(8),ST(9),Fe0(9),f_star_0,eps_0,vvf_0
    integer i,j,Tnum,Lnum
    double precision x1(8),FDstep,R0(8),R(8),Res(8),lambda,Jac(64)
    double precision Tub,Tlb,Tdisc,Lub,Llb,Ldisc
    double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

    call errFunction(x, R0, pmat, ST, Fe0, f_star_0, eps_0, vvf_0,
   1  Tub,Tlb,Tdisc,Tnum,Lub,Llb,Ldisc,Lnum,a_q1,a_q2,a_fc,a_ff)
    do i = 1,8
```

```
        if(isnan(R0(i)).eqv..true.)then
           print *, "R0(",i,") = ",R0(i)
           call exit(0)
        end if
      end do
C   //Finite Difference loop
      do j = 1,8

C       //add FD increment
        do i = 1,8
           x1(i) = x(i)
        end do
        x1(j) =x1(j)+FDstep
C       //calculate R
        call errFunction(x1,R,pmat,ST,Fe0,f_star_0,eps_0,vvf_0,
     1  Tub,Tlb,Tdisc,Tnum,Lub,Llb,Ldisc,Lnum,a_q1,a_q2,a_fc,a_ff)

C       //save FD deriv into jacobian matrix
        do i = 1,8
           Jac((i-1)*8+j) = (R(i)-R0(i))/FDstep

        end do
      end do
C   //return -1*R0 if requested
      do i = 1,8
      if (Res(i)/=10101.0) then
           Res(i) = -R0(i)
      end if
      end do

      end subroutine calculateJacobian


      subroutine errFunction(x,b,pmat,ST,Fe0,f_star_0,eps_0,vvf_0,
     1    Tub,Tlb,Tdisc,Tnum,Lub,Llb,Ldisc,Lnum,a_q1,a_q2,a_fc,a_ff)
      double precision x(8),pmat(8),ST(9),Fe0(9),f_star_0,eps_0,vvf_0
      double precision deps,eps,dvvf,vvf,dEp(9),dS(9),S(9),det
      integer i,Tnum,Lnum
      double precision dFp(9),dFp_inv(9),Fe1(9),Tub,Tlb,Tdisc
      double precision f_star,fc,ff,q1,q2,P(9),sy,b(8),Phi,df_star,H
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

      deps = x(1)
       eps  = x(1) + eps_0
       dvvf = x(2)
       vvf  = x(2) + vvf_0
      dEp(1) = x(3)
      dEp(5) = x(4)
      dEp(9) = x(5)
      dEp(2) = x(6)
      dEp(4) = x(6)
      dEp(3) = x(7)
      dEp(7) = x(7)
      dEp(6) = x(8)
      dEp(8) = x(8)

C     // ----check new phi----
C     //get new stress
      call stress(dS,dEp,pmat)
      do i = 1,9
         S(i) = ST(i) - dS(i)
      end do

C     //get new Fe1
      call matlib_expmat(dEp,dFp)
      call matlib_inverse(dFp, dFp_inv,det)
      call matlib_mulss(Fe0,dFp_inv, Fe1)

C     //update fstar
      call matlib_mulss(Fe1,S,P)
      q2 = 10101.0
      call calculateNewQs(P,Fe1, q1, q2, fc, ff,Tub,Tlb,Tnum,
     1  Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      df_star = 10101.0
      call updateF_star(vvf, dvvf, f_star_0, df_star,fc, ff,q1,f_star)

C     //:::error handling:::
      f_star= MIN(f_star, 1.0/q1)
c     //::::::::::::::::::::
C     //update sy
      call harden(pmat, eps,sy)

C     //calculate Phi
      call calculateResiduals(b,dEp,S,Fe1,sy,f_star,
     1    deps,dvvf,vvf,.false.,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,
     2  Ldisc,a_q1,a_q2,a_fc,a_ff)
      do i = 1,8
         if(isnan(b(i)).eqv..true.)then
            print *, "b(",i,") = ",b(i)
            call exit(0)
         end if
      end do

      end subroutine errFunction


      subroutine errFunctionScalar(x,pmat,ST,Fe0,f_star_0,
     1    eps_0,vvf_0,Phi,Tub,Tlb,Tnum,Tdisc,Lub,Llb, Lnum,Ldisc,
```

```
     2     a_q1,a_q2,a_fc,a_ff)
      double precision x(8),pmat(8),ST(9),Fe0(9),f_star_0,eps_0,vvf_0
      double precision deps,eps,dvvf,vvf,dEp(9),dS(9),S(9),det
      integer i,Tnum,Lnum
      double precision dFp(9),dFp_inv(9),Fe1(9),Tub,Tlb,Tdisc,Lub,Llb
      double precision f_star,fc,ff,q1,q2,P(9),sy,Ldisc,df_star,H
      double precision b(8),Phi
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

       deps = x(1)
       eps  = x(1) + eps_0
       dvvf = x(2)
       vvf  = x(2) + vvf_0
      dEp(1) = x(3)
      dEp(5) = x(4)
      dEp(9) = x(5)
      dEp(2) = x(6)
      dEp(4) = x(6)
      dEp(3) = x(7)
      dEp(7) = x(7)
      dEp(6) = x(8)
      dEp(8) = x(8)

C     // ----check new phi----
C     //get new stress
      call stress(dS,dEp,pmat)
      do i = 1,9
         S(i) = ST(i) - dS(i)
      end do

C     //get new Fe1
      call matlib_expmat(dEp,dFp)
      call matlib_inverse(dFp, dFp_inv,det)
      call matlib_mulss(Fe0,dFp_inv, Fe1)
C     //update fstar
      call matlib_mulss(Fe1,S,P)
      q2 = 10101.0
      call calculateNewQs(P,Fe1, q1, q2, fc, ff,Tub,Tlb,Tnum,
     1  Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)

      df_star = 10101.0
      call updateF_star(vvf,dvvf,f_star_0,df_star,fc,ff,q1,f_star)

C     //:::error handling:::
      f_star= MIN(f_star, 1.0/q1)
C     //::::::::::::::::::::

C     //update sy
      call harden(pmat, eps, sy)

C     //calculate Phi
      call calculateResiduals(b,dEp,S,Fe1,sy,f_star,deps,dvvf,vvf,
     1     .false.,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,
     2     a_q1,a_q2,a_fc,a_ff)
      Phi=0
      do i=1,8
         Phi=Phi+b(i)*b(i)
      end do
      Phi=Phi**0.5

C     //:::error handling:::
      if (eps <0) then
         Phi=Phi+1.0e9
      end if
      if (vvf <0)then
         Phi=Phi+1.0e9
      end if
      if((isnan(Phi)).neqv..false.)then
         Phi=1.e09
      end if

      end subroutine errFunctionScalar



      subroutine calculateResiduals(Res,dEp,S,Fe,sy,f_star,
     1    deps,dvvf,vvf,DEBUG,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,
     2  Ldisc,a_q1,a_q2,a_fc,a_ff)
      double precision Res(8),dEp(9),S(9),Fe(9),sy,Tub,Tlb,Tdisc
      double precision f_star,deps,dvvf,vvf,Lub,Llb,Ldisc
      logical DEBUG,noStress
      double precision H1helper,dPhi_dS(9), Mhelper(9)
      integer i,Tnum,Lnum
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

C     //----Phi----
      call calculatePhi(S,Fe,sy,f_star,noStress,Res(1),Tub,Tlb,
     1  Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
         if(isnan(Res(1)).eqv..true.)then
            call exit(0)
         end if

C     //----H----
      H1helper=0
      do i =1,9
         H1helper = H1helper+ S(i)*dEp(i)
      end do
      Res(2) = dvvf - (1-vvf)*(dEp(1)+dEp(5)+dEp(9))
```

```
      Res(3) = deps - H1helper/((1-vvf)*sy)

C   //----M----
      call calculateDPhiDS(dPhi_dS, S, Fe, sy, f_star,DEBUG,Tub,Tlb,
     1  Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)

      if(DEBUG.neqv..false.)then
      print *,"============DEBUG============="
      print *,"dPhidS= ",dPhi_dS
      print *,"==========End Debug==========="
      end if

      Res(4) = dEp(1)*dPhi_dS(5) - dEp(5)*dPhi_dS(1)
      Res(5) = dEp(5)*dPhi_dS(9) - dEp(9)*dPhi_dS(5)
      Res(6) = dEp(9)*dPhi_dS(2) - dEp(2)*dPhi_dS(9)
      Res(7) = dEp(2)*dPhi_dS(3) - dEp(3)*dPhi_dS(2)
      Res(8) = dEp(3)*dPhi_dS(6) - dEp(6)*dPhi_dS(3)

      do i = 1,8
        if(isnan(Res(i)).eqv..true.)then
          print *, "Res(",i,") = ",Res(i)
          print *, "dEp =",dEp
          print *, "dPhi_dS =",dPhi_dS
          call exit(0)
        end if
      end do

      end subroutine calculateResiduals


      subroutine calculateDPhiDS(dPhi_dS,S,Fe,sy,f_star,DEBUG,Tub,Tlb,
     1  Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      double precision dPhi_dS(9),S(9),Fe(9),sy,f_star,Phi
      double precision P(9),q1,q2,fc,ff,Tub,Tlb,Tdisc,Lub,Llb,Ldisc
      logical DEBUG
      double precision I1,I2,J2,expIn,COSHH,SINHH,C1,C2,C3,C4,eye(9)
      double precision S_dev(9),sh,dq1_dS(9),dq2_dS(9)
      integer i,Tnum,Lnum
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

C   //calculate pk-1 stress
      call matlib_mulss(Fe,S,P)
C   //calculate deviatoric stress
       I1 = S(1)+S(5)+S(9)
       do i = 1,9
         S_dev(i) = S(i)
      end do
      sh = I1/3.0
      S_dev(1) = S_dev(1)-sh
      S_dev(5) = S_dev(5)-sh
      S_dev(9) = S_dev(9)-sh

C   //calculate q's and dq's
      call calculateDqDS(S,Fe,dq1_dS,dq2_dS,DEBUG,Tub,Tlb,Tnum,
     1  Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      fc = 10101.0
      ff = 10101.0
      call calculateNewQs(P,Fe, q1, q2,fc,ff,Tub,Tlb,Tnum,
     1  Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)

      if (DEBUG.neqv..false.)then
        print *,"I1=",I1
        print *,"q1=",q1
        print *,"q2=",q2
      end if

C   //calculate dPhi_dS
       expIn = 0.5*q2*I1/sy
       COSHH = cosh(expIn)
       SINHH = sinh(expIn)
       C1=3./(sy*sy)
       C2=q1*q2*f_star/sy*SINHH
       C3=2.*f_star*(COSHH - q1*f_star)
       C4=f_star*q1*I1/sy*SINHH
       eye = (/1.,0.,0.,0.,1.,0.,0.,0.,1./)
       do i = 1,9
        dPhi_dS(i)=C1*S_dev(i)+ C2*eye(i)+ C3*dq1_dS(i)+ C4*dq2_dS(i)
       end do
        do i = 1,8
          if(isnan(dPhi_dS(i)).eqv..true.)then
            print *,"S(1,5,9) =",S
            print *,"Fe(",i,") =",Fe(i)
            print *,"P(",i,") =",P(i)
            print *, "S_dev(",i,") = ",S_dev(i)
            print *, "C1 =",C1
            print *, "sy =", sy
            print *, "C2 =",C2
            print *, "q1 =",q1," q2 =",q2," f_star =",f_star
            print *, "SINHH =",SINHH," I1 =",I1
            print *, "dq1_dS =",dq1_dS
            print *, "C3 =",C3
            print *, "dq2_dS =",dq2_dS
            print *, "C4 =",C4
            call exit(0)
          end if
        end do

      if (DEBUG.neqv..false.)then
```

```
      print *,"============================DEBUG======================="
      print *,"c1=",C1
      print *,"c2=",C2
      print *,"c3=",C3
      print *,"c4=",C4
      print *,"S_dev=",S_dev
      print *,"dq1_ds=",dq1_ds
      print *,"dq2_ds=",dq2_ds
      print *,"====================== END DEBUG====================="
      end if

      end subroutine calculateDPhiDS


      subroutine calculatePhi(S,Fe,sy,f_star,noStress,Phi,Tub,Tlb,
     1  Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      double precision S(9),Fe(9),sy,f_star,Phi,P(9),q1,q2,fc,ff
      double precision Tub,Tlb,Tdisc,Lub,Llb,Ldisc,expIn,COSHH
      logical noStress
      double precision I1,I2,J2
      integer Tnum,Lnum
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

C  // Get q's
      call matlib_mulss(Fe,S,P)

      fc = 10101.0
      ff = 10101.0
       call calculateNewQs(P,Fe, q1, q2,fc,ff,Tub,Tlb,
     1  Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)

      if (noStress .eqv..true.)then
         if (f_star>(1.0/q1)) then
            noStress=.true.
         end if
      else if (noStress .eqv..false.)then
         if (f_star>(1.0/q1)) then
         noStress=.true.
         end if
      end if

C  // calculate yield function
       I1 = S(1)+S(5)+S(9)
       I2 = S(1)*(S(5)+S(9))+S(5)*S(9)-S(2)*S(4)-S(3)*S(7)-S(6)*S(8)
       J2 = I1*I1/3.0 - I2
       expIn = 0.5*q2*I1/sy
       COSHH = cosh(expIn)
      Phi=3.0*J2/(sy*sy)+2.0*q1*f_star*COSHH-(1.0+q1*q1*f_star*f_star)

      end subroutine calculatePhi

      subroutine calculateDqDS(S,Fe,dq1_dS,dq2_dS,DEBUG,Tub,Tlb,Tnum,
     1  Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      double precision S(9),Fe(9),dq1_dS(9),dq2_dS(9)
      logical DEBUG
      double precision PREC,P(9),Sig(9),J
      integer i
      double precision Sig_dev(9),I1,sh
      double precision I2,J2,J3,VM,Tri,Lode,dq1dT,dq1dL,dq2dT,dq2dL
      double precision dTdSig(9),dLdSig(9),eye(9),Sig_squared(9)
      double precision Ct1,Ct2,Cl1,Cl2,Cl3
      double precision dq1_dSig(9),dq2_dSig(9)
      double precision Tub,Tlb,Tdisc,Lub,Llb,Ldisc,T,L
      integer Tnum,Lnum
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)
      PREC = 1.0e-16

C   //calculate pk-1 stress
      call matlib_mulss(Fe,S,P)

C       //calculate cauchy stress
      call matlib_determinant(Fe,J)
      call matlib_mulst(P,Fe,Sig)

      do i = 1,9
         Sig(i) =  Sig(i)/J
      end do

C   //calculate deviatoric stress
      do i = 1,9
         Sig_dev(i) = Sig(i)
      end do
       I1 = Sig(1)+Sig(5)+Sig(9)
       sh = I1/3
      Sig_dev(1) = Sig_dev(1)-sh
      Sig_dev(5) = Sig_dev(5)-sh
      Sig_dev(9) = Sig_dev(9)-sh

C   //calculate invariants
       I2 = Sig(1)*(Sig(5)+Sig(9)) + Sig(5)*Sig(9) - Sig(2)*Sig(4)
     1      - Sig(3)*Sig(7) - Sig(6)*Sig(8)
      J2 = I1*I1/3.0 - I2
      call matlib_determinant(Sig_dev,J3)
      VM = (3.0*J2)**0.5
      Tri = sh/VM
      Lode = 13.5*J3/(VM*VM*VM)

C   //get dqdT and dqdL
```

```
      if (abs(J2)<PREC) then
        if (abs(I1)<PREC) then
          dq1dT=0.
          dq2dT=0.
          dq1dL=0.
          dq2dL=0.
          return
        else
          Tri=10.
          Lode=0.
         call interpolate2dBicubicDeriv(Tri,Lode,dq1dT,dq1dL,dq2dT,
     1  dq2dL,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
        end if
      else
        call interpolate2dBicubicDeriv(Tri,Lode,dq1dT,dq1dL,dq2dT,
     1  dq2dL,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      end if

C     //calculate deriv wrt cauchy stress
      eye = (/1.,0.,0.,0.,1.,0.,0.,0.,1./)
      call matlib_mulss(Sig,Sig,Sig_squared)
      Ct1=(27.*J2)**(-0.5)
      Ct2=I1/(2.*J2)
      Cl1=13.5*((3.0*J2)**(-1.5))
      Cl2=(1.5*J3/J2 + 2./3.*I1)
      Cl3=(2./3.*I2 - I1*I1/3.)

      if (DEBUG .neqv..false.) then
      print *,"============================DEBUG======================="
      print *,"ct1=",ct1
      print *,"ct2=",ct2
      print *,"cl1=",cl1
      print *,"cl2=",cl2
      print *, "dq1dT = ",dq1dT
      print *, "dq1dL = ",dq1dL
      print *, "dq2dT = ",dq2dT
      print *, "dq2dL = ",dq2dL
      print *,"Sig^2=",Sig_squared
      end if

        do i = 1,9
          dTdSig(i)=Ct1*(eye(i) - Ct2*Sig_dev(i))
          dLdSig(i)=Cl1*(Sig_squared(i)-Cl2*Sig_dev(i)+Cl3*eye(i))
        end do

        do i = 1,9
          dq1_dSig(i) = dq1dT*dTdSig(i) + dq1dL*dLdSig(i)
          dq2_dSig(i) = dq2dT*dTdSig(i) + dq2dL*dLdSig(i)
        end do

      if (DEBUG.neqv..false.) then
        print *,"dq1_dSig=",dq1_dSig
        print *,"dq2_dSig=",dq2_dSig
        print *,"========================END DEBUG================="
        print *, "   "
      end if

C     //convert to pk2 derivative
      call convertCauchyDerivToPK2deriv(dq1_dSig,Fe,J,dq1_dS)
      call convertCauchyDerivToPK2deriv(dq2_dSig,Fe,J,dq2_dS)

      end subroutine calculateDqDS

      subroutine calculateNewQs(P,F,q1,q2,fc,ff,Tub,Tlb,Tnum,
     1  Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
      integer i
      double precision trace,VM,Tri,Lode
      double precision q1,q2,fc,ff,PREC
      double precision P(9),F(9),T(9)
      double precision J,I2,J3
      double precision Tub,Tlb,Tdisc,Lub,Llb,Ldisc
      integer Tnum,Lnum
      double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)
      VM = 0
      PREC = 1.0e-16

C     //get Cauchy stress
      call matlib_determinant(F,J)
      call matlib_mulst(P,F,T)

      do i = 1,9
        T(i) =  T(i)/J
      end do

C     //get VM stress
       trace = (T(1)+T(5)+T(9))/3
       I2 = T(1)*(T(5)+T(9))+T(5)*T(9)-T(2)*T(4)-T(3)*T(7)-T(6)*T(8)
      do i = 1,9,+4
        T(i) = T(i)-trace
      end do

      VM = 9.0*trace*trace-3.0*I2
      VM = VM**0.5
      call matlib_determinant(T,J3)

C     //Triaxiality and Lode
      Tri = trace/VM
      Lode = 13.5*J3/(VM*VM*VM)
```

```
      if (abs(VM)<PREC) then
        if (abs(trace)<PREC)then
             q1 = 1.0
             q2 = 1.0
             fc = 1.0
             ff = 1.0
          return
        else
          Tri=10.
          Lode=0.
        end if
      end if

      call interpolate2dBicubic(Tri,Lode,q1,q2,fc,ff,Tub,Tlb,Tnum,
     1  Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
     end subroutine calculateNewQs

      subroutine solve_LUP_8x8(b,T,L,U,P,skip,x)
      double precision x(8),b(8),T(8),L(64),U(64),c(8)
      double precision PREC,summ
      integer i,j,P(8)
      logical skip(8)
      PREC = 1.0e-16

C     //get c //includes preconditioning
      do i = 1,8
        c(i) = b(P(i))*T(P(i))
      end do

      do i = 2,8
        if (skip(i).eqv..false.) then
          do j = 1,(i-1)
            c(i) = c(i) - L((i-1)*8.0+j)*c(j)
          end do
        end if
      end do

C     //get x (back substitution)
      do i = 8,1,-1
        if (skip(i).eqv..true.)then
          x(P(i))=0.0
        else
           summ=c(i)
          do j = (i+1),8
          summ = summ - x(P(j))*U((i-1)*8+j)
          end do
        x(P(i)) = summ/U((i-1)*8+i)
        end if
      end do
      end subroutine solve_LUP_8x8


      subroutine LUP_8x8(A,T,L,U,P,skip)
      double precision A(64),T(8),L(64),U(64)
      double precision PREC,tmp,multiplier
      integer i,j,k,P(8),swaprow
      logical skip(8)
C     //this works for dense, nonsingular matrices which do not require permutation for
pivots
      PREC = 1.0e-16
      P = (/1,2,3,4,5,6,7,8/)  !(/0,1,2,3,4,5,6,7/)
      skip = (/.false.,.false.,.false.,.false.,.false.,.false.,
     1    .false.,.false./)
      do i=1,64
        L(i) = 0.0
      end do
      do i = 1,64,+9
        L(i) = 1.0
      end do

C     //do pecondtioning
      do i = 1,8
        do j = 1,8
          U((i-1)*8+j) = A((i-1)*8+j)*T(i)
        end do
      end do

C     //reduce to upper triangular
       do i=1,7
        if (abs(U((i-1)*8+i))<PREC) then
          swaprow = -1
          do j = (i+1),8
            if (abs(U((j-1)*8+i))>PREC)then
              swaprow = j
              exit
            end if
          end do
C         //if swapable row, then swap
          if (swaprow/=-1)then
            do j = i,8
              tmp = U((i-1)*8+j)
              U((i-1)*8+j) = U((swaprow-1)*8+j)
              U((swaprow-1)*8+j) = tmp
            end do
            tmp=P(i)
            P(i)=P(swaprow)
            P(swaprow)=int(tmp)
```

```
          else
            skip(i)=.true.
          end if
        end if

        if (skip(i).eqv..false.) then
          do j = (i+1),8
            if (abs(U((j-1)*8+i))>PREC)then
              multiplier = U((j-1)*8+i)/U((i-1)*8+i)
              U((j-1)*8+i)=0.0
              do k = (i+1),8
            U((j-1)*8+k)=U((j-1)*8+k)-multiplier*U((i-1)*8+k)
              end do
              L((j-1)*8+i) = multiplier
            else
              U((j-1)*8+i) = 0.0
            end if
          end do
        end if
      end do

      if (abs(A(64))<PREC)then
        skip(8)=.true.
      end if
      end subroutine LUP_8x8

      subroutine solve_8x8_w_precond(A0,x,b0,T)
      double precision x(8),A(64),b(8),A0(64),b0(8),T(8)
      double precision PREC,tmp,multiplier,summ
      integer i,j,k,P(8),swaprow
      logical skip(8)

C     //this works for dense, nonsingular matrices which do not require permutation for
pivots
      PREC = 1.0e-16
      P = (/1,2,3,4,5,6,7,8/)
      skip = (/.false.,.false.,.false.,.false.,.false.,.false.,
     1    .false.,.false./)

      do i = 1,8
        do j = 1,8
          A((i-1)*8+j) = A0((i-1)*8+j)*T(i)
        end do
      end do

      do i = 1,8
        b(i) = b0(i)*T(i)
      end do

C     //reduce to upper triangular
      do i=1,7

        if (abs(A((i-1)*8+i))<PREC) then
C         //try to find row to swap
          swaprow = -1
          do j = (i+1),8
            if (abs(A((j-1)*8+i))>PREC)then
              swaprow = j
              exit
            end if
          end do
C         //if swapable row, then swap
          if (swaprow/=-1)then
            do j = i,8
              tmp = A((i-1)*8+j)
              A((i-1)*8+j) = A((swaprow-1)*8+j)
              A((swaprow-1)*8+j) = tmp
            end do
            tmp=b(i)
            b(i)=b(swaprow)
            b(swaprow)=tmp
            tmp=P(i)
            P(i)=P(swaprow)
            P(swaprow)=tmp
          else
            skip(i)=.true.
          end if
        end if

        if (skip(i).eqv..false.) then
          do j = (i+1),8
            if (abs(A((j-1)*8+i))>PREC)then
              multiplier = A((j-1)*8+i)/A((i-1)*8+i)
              A((j-1)*8+i)=0.0
              do k = (i+1),8
            A((j-1)*8+k)=A((j-1)*8+k)-multiplier*A((i-1)*8+k)
              end do
              b(j) = b(j) - multiplier*b(i)
            else
              A((j-1)*8+i) = 0.0
            end if
          end do
        end if
      end do

      if (abs(A(64))<PREC)then
```

```
        skip(8)=.true.
      end if

C   //do back substitution
      do i = 8,1,-1
        if (skip(i).eqv..true.)then
          x(P(i))=0.0
        else
           summ=b(i)
          do j = (i+1),8,+1
             summ = summ - x(P(j))*A((i-1)*8+j)
          end do
            x(P(i))= summ/A((i-1)*8+i)
        end if
       end do
      end subroutine solve_8x8_w_precond

      subroutine solve_8x8(A0,x,b0)
      double precision x(8),A(64),b(8),A0(64),b0(8)
      double precision PREC,tmp,multiplier,summ
      integer i,j,k,P(8),swaprow
      logical skip(8)

C   //this works for dense, nonsingular matrices which do not require permutation for
pivots
      PREC = 1.0e-16
      P = (/1,2,3,4,5,6,7,8/)
      skip = (/.false.,.false.,.false.,.false.,.false.,.false.,
     1   .false.,.false./)

C   //this duplication is done so the original A and b are not overewritten
        Do j= 1,64
          A(j) = A0(j)
        End Do
        Do j= 1,8
          b(j) = b0(j)
        End Do

C   //reduce to upper triangular
       do i=1,7
         if (abs(A((i-1)*8+i))<PREC) then
           swaprow = -1
           do j = (i+1),8
             if (abs(A((j-1)*8+i))>PREC)then
               swaprow = j
               exit
             end if
           end do
C          //if swapable row, then swap
           if (swaprow/=-1)then
             do j = i,8
               tmp = A((i-1)*8+j)
               A((i-1)*8+j) = A((swaprow-1)*8+j)
               A((swaprow-1)*8+j) = tmp
             end do
             tmp=b(i)
             b(i)=b(swaprow)
             b(swaprow)=tmp
             tmp=P(i)
             P(i)=P(swaprow)
             P(swaprow)=tmp
           else
             skip(i)=.true.
           end if
         end if

         if (skip(i).eqv..false.) then
           do j = (i+1),8
             if (abs(A((j-1)*8+i))>PREC)then
               multiplier = A((j-1)*8+i)/A((i-1)*8+i)
               A((j-1)*8+i)=0.0
               do k = (i+1),8
             A((j-1)*8+k)=A((j-1)*8+k)-multiplier*A((i-1)*8+k)
               end do
               b(j) = b(j) - multiplier*b(i)
             else
               A((j-1)*8+i) = 0.0
             end if
           end do
         end if
       end do

       if (abs(A(64))<PREC)then
         skip(8)=.true.
       end if

C   //do back substitution
       do i = 8,1,-1
         if (skip(i).eqv..true.)then
           x(P(i))=0.0
         else
            summ=b(i)
           do j = (i+1),8,+1
             summ = summ - x(P(j))*A((i-1)*8+j)
           end do
             x(P(i))= summ/A((i-1)*8+i)
         end if
```

```
 end do
  end subroutine solve_8x8

  subroutine interpolate2dBicubicDeriv(Tri,Lode,dq1dT,dq1dL,dq2dT,
1  dq2dL,Tub,Tlb,Tnum,Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
  double precision Tub,Tlb,Tdisc,Lub,Llb,Ldisc
  integer Tnum,Lnum,aloc,aq1,aq2,afc,aff
  double precision Tri,Lode,x,y,x2,x3,y2,y3
  double precision dq1dT,dq2dT,dq1dL,dq2dL,T,L
  double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

  T = (max(min(Tri,Tub-.0000001),Tlb) - Tlb)/Tdisc
  L = (max(min(Lode,Lub-.0000001),Llb) - Llb)/Ldisc

  x = L-(int(L))
  y = T-(int(T))
 x2= x*x
 x3= x2*x
 y2= y*y
 y3= y2*y

 aloc=(int(T)*(Lnum-1) + int(L))*16

  if (Tri>=Tlb.and.Tri<=Tub)then
   aq1=aloc
   dq1dT=1.0/Tdisc*((a_q1(aq1+5) + a_q1(aq1+6)*x + a_q1(aq1+7)*x2
1       + a_q1(aq1+8)*x3)
2 + 2.*y*(a_q1(aq1+9) + a_q1(aq1+10)*x + a_q1(aq1+11)*x2
3       + a_q1(aq1+12)*x3)
4 + 3.*y2*(a_q1(aq1+13) + a_q1(aq1+14)*x + a_q1(aq1+15)*x2
5       + a_q1(aq1+16)*x3))

   aq2=aloc
   dq2dT=1.0/Tdisc*((a_q2(aq2+5) + a_q2(aq2+6)*x + a_q2(aq2+7)*x2
1       + a_q2(aq2+8)*x3)
2 +  2.*y*(a_q2(aq2+9)  + a_q2(aq2+10)*x  + a_q2(aq2+11)*x2
3       + a_q2(aq2+12)*x3)
4 + 3.*y2*(a_q2(aq2+13) + a_q2(aq2+14)*x + a_q2(aq2+15)*x2
5       + a_q2(aq2+16)*x3))
   else
  dq1dT=0.
  dq2dT=0.
   end if

  if (Lode>=Llb.and.Lode<=Lub) then
   aq1=aloc
   dq1dL=1.0/Ldisc*((a_q1(aq1+2)+a_q1(aq1+3)*2.*x+a_q1(aq1+4)*3*x2)
1     + y* (a_q1(aq1+6)  + a_q1(aq1+7)*2.*x  + a_q1(aq1+8)*3.*x2 )
2    + y2*(a_q1(aq1+10)  + a_q1(aq1+11)*2.*x + a_q1(aq1+12)*3.*x2)
3    + y3*(a_q1(aq1+14) + a_q1(aq1+15)*2.*x + a_q1(aq1+16)*3.*x2))

   aq2=aloc
  dq2dL=1.0/Ldisc*((a_q2(aq2+2)+a_q2(aq2+3)*2.*x+a_q2(aq2+4)*3.*x2)
1      + y* (a_q2(aq2+6)  + a_q2(aq2+7)*2.*x +a_q2(aq2+8)*3.*x2)
2     + y2*(a_q2(aq2+10)  + a_q2(aq2+11)*2.*x+a_q2(aq2+12)*3.*x2)
3    + y3*(a_q2(aq2+14) + a_q2(aq2+15)*2.*x +a_q2(aq2+16)*3.*x2))
  else
  dq1dL=0.
  dq2dL=0.
  end if
  end subroutine interpolate2dBicubicDeriv


  subroutine interpolate2dBicubic(T,L,q1,q2,fc,ff,Tub,Tlb,Tnum,
1  Tdisc,Lub,Llb,Lnum,Ldisc,a_q1,a_q2,a_fc,a_ff)
 double precision q1,q2,fc,ff,Tub,Tlb,Tdisc,Lub,Llb,Ldisc
 integer Tnum,Lnum,aloc,aq1,aq2,afc,aff
 double precision T,L,x,y,x2,x3,y2,y3,TT,LL, weird
  double precision a_q1(1120),a_q2(1120),a_fc(1120),a_ff(1120)

  TT = (max(min(T,Tub-.0000001),Tlb) - Tlb)/Tdisc
  LL = (max(min(L,Lub-.0000001),Llb) - Llb)/Ldisc

  x = LL-(int(LL))
  y = TT-(int(TT))
 x2= x*x
 x3= x2*x
 y2= y*y
 y3= y2*y
 aloc=((int(TT))*(Lnum-1) + int(LL))*16

 if (q1/=10101.0) then
  aq1=aloc
  q1=   (a_q1(aq1+1)+a_q1(aq1+2)*x+a_q1(aq1+3)*x2+a_q1(aq1+4)*x3)
1     +y*(a_q1(aq1+5)+a_q1(aq1+6)*x+a_q1(aq1+7)*x2+a_q1(aq1+8)*x3)
2  +y2*(a_q1(aq1+9)+a_q1(aq1+10)*x+a_q1(aq1+11)*x2+a_q1(aq1+12)*x3)
3 +y3*(a_q1(aq1+13)+a_q1(aq1+14)*x+a_q1(aq1+15)*x2+a_q1(aq1+16)*x3)
  end if

 if (q2/=10101.0) then
  aq2=aloc
  q2=   (a_q2(aq2+1)+a_q2(aq2+2)*x+a_q2(aq2+3)*x2+a_q2(aq2+4)*x3)
1    + y*(a_q2(aq2+5)+a_q2(aq2+6)*x+a_q2(aq2+7)*x2+a_q2(aq2+8)*x3)
2  +y2*(a_q2(aq2+9)+a_q2(aq2+10)*x+a_q2(aq2+11)*x2+a_q2(aq2+12)*x3)
3 +y3*(a_q2(aq2+13)+a_q2(aq2+14)*x+a_q2(aq2+15)*x2+a_q2(aq2+16)*x3)
  end if

  if (fc/=10101.0)then
```

```
        afc=aloc
       fc=(a_fc(afc+1)+a_fc(afc+2)*x+a_fc(afc+3)*x2+a_fc(afc+4)*x3)
     1    + y*(a_fc(afc+5)+a_fc(afc+6)*x+ a_fc(afc+7)*x2+a_fc(afc+8)*x3)
     2  +y2*(a_fc(afc+9)+a_fc(afc+10)*x+a_fc(afc+11)*x2+a_fc(afc+12)*x3)
     3 +y3*(a_fc(afc+13)+a_fc(afc+14)*x+a_fc(afc+15)*x2+a_fc(afc+16)*x3)
       end if

       if (ff/=10101.0)then
        aff=aloc
       ff=(a_ff(aff+1)+a_ff(aff+2)*x+a_ff(aff+3)*x2+a_ff(aff+4)*x3)
     1     +y*(a_ff(aff+5)+a_ff(aff+6)*x+a_ff(aff+7)*x2+a_ff(aff+8)*x3)
     2  +y2*(a_ff(aff+9)+a_ff(aff+10)*x+a_ff(aff+11)*x2+a_ff(aff+12)*x3)
     3 +y3*(a_ff(aff+13)+a_ff(aff+14)*x+a_ff(aff+15)*x2+a_ff(aff+16)*x3)
       end if
       end subroutine interpolate2dBicubic

      subroutine calculateBicubicAs(A,F,X,Y)
       integer i,j,X,Y,xinc,yinc,aa
       integer i00,i10,i01,i11
       integer ii,jj,counter
       double precision A,F,k(16),Minv(256)
       double precision f00,f01,f10,f11
       double precision f00x, f01x, f10x, f11x
       double precision f00y, f01y, f10y, f11y
       double precision f00xy, f01xy, f10xy, f11xy
       dimension A((X-1)*(Y-1)*16),F(X*Y)

       xinc=1
       yinc=X
       counter = 0
       do i = 1,X-1
          do j = 1,Y-1
             counter = counter+1
             aa=((j-1)*(X-1)+(i-1))*16
             i00= i+(j-1)*X
             i10= (i+1)+(j-1)*X
             i01= i+j*X
             i11= (i+1)+j*X


           f00=F(i00)
           f10=F(i10)
           f01=F(i01)
           f11=F(i11)

C          fx
          if (i==1) then
             f00x=(F(i00+xinc)-F(i00))
             f01x=(F(i01+xinc)-F(i01))
          else
             f00x=(F(i00+xinc)-F(i00-xinc))/2.
             f01x=(F(i01+xinc)-F(i01-xinc))/2.
          end if

          if (i==X-1) then
             f10x=(F(i10)-F(i10-xinc))
             f11x=(F(i11)-F(i11-xinc))
           else
             f10x=(F(i10+xinc)-F(i10-xinc))/2.
             f11x=(F(i11+xinc)-F(i11-xinc))/2.
          end if

C          fy
          if (j==1) then
             f00y=(F(i00+yinc)-F(i00))
             f10y=(F(i10+yinc)-F(i10))
           else
             f00y=(F(i00+yinc)-F(i00-yinc))/2.
             f10y=(F(i10+yinc)-F(i10-yinc))/2.
          end if

          if (j==Y-1) then
             f01y=(F(i01)-F(i01-yinc))
             f11y=(F(i11)-F(i11-yinc))
          else
             f01y=(F(i01+yinc)-F(i01-yinc))/2.
             f11y=(F(i11+yinc)-F(i11-yinc))/2.
          end if

C          f00xy
          if (i>1 .and. j>1) then
             f00xy=(F(i00+xinc+yinc)-F(i00-xinc+yinc)
     1            -F(i00+xinc-yinc)+F(i00-xinc-yinc))/4.
          else if (i==1 .and. j==1) then
             f00xy=(F(i00+xinc+yinc)-F(i00+yinc)-F(i00+xinc)+F(i00))
          else if (i==1) then
             f00xy=(F(i00+xinc+yinc)-F(i00+yinc)-F(i00+xinc-yinc)
     1            +F(i00-yinc))/2.
          else if (j==1) then
             f00xy=(F(i00+xinc+yinc)-F(i00-xinc+yinc)-F(i00+xinc)
     1            +F(i00-xinc))/2.
          else
             print *,"   "
             print *,"   "
             print *,"bad code error!"
             print *,"   "
             print *,"   "
          end if
```

```
C       f01xy
        if (i>1 .and. j<Y-1) then
          f01xy=(F(i01+xinc+yinc)-F(i01-xinc+yinc)
     1          -F(i01+xinc-yinc)+F(i01-xinc-yinc))/4.
        else if (i==1 .and. j==Y-1) then
          f01xy=(F(i01+xinc)-F(i01)-F(i01+xinc-yinc)+F(i01-yinc))
        else if (i==1) then
          f01xy=(F(i01+xinc+yinc)-F(i01+yinc)-F(i01+xinc-yinc)
     1           +F(i01-yinc))/2.
        else if (j==Y-1)then
          f01xy=(F(i01+xinc)-F(i01-xinc)-F(i01+xinc-yinc)
     1           +F(i01-xinc-yinc))/2.
        else
          print *,"   "
          print *,"   "
          print *,"bad code error!"
          print *,"   "
          print *,"   "
        end if

C       f10xy
        if (i<X-1 .and. j>1) then
          f10xy=(F(i10+xinc+yinc)-F(i10-xinc+yinc)
     1           -F(i10+xinc-yinc)+F(i10-xinc-yinc))/4.
        else if (i==X-1 .and. j==1) then
          f10xy=(F(i10+yinc)-F(i10-xinc+yinc)-F(i10)+F(i10-xinc))
        else if (i==X-1) then
          f10xy=(F(i10+yinc)-F(i10-xinc+yinc)-F(i10-yinc)
     1           +F(i10-xinc-yinc))/2.
        else if (j==1) then
          f10xy=(F(i10+xinc+yinc)-F(i10-xinc+yinc)
     1           -F(i10+xinc)+F(i10-xinc))/2.
        else
          print *,"   "
          print *,"   "
          print *,"bad code error!"
          print *,"   "
          print *,"   "
        end if

C       f11xy
        if (i<X-1 .and. j<Y-1) then
          f11xy=(F(i11+xinc+yinc)-F(i11-xinc+yinc)
     1           -F(i11+xinc-yinc)+F(i11-xinc-yinc))/4.
        else if (i==X-1 .and. j==Y-1) then
          f11xy=(F(i11)-F(i11-xinc)-F(i11-yinc)+F(i11-xinc-yinc))
        else if (i==X-1) then
          f11xy=(F(i11+yinc)-F(i11-xinc+yinc)-F(i11-yinc)
     1           +F(i11-xinc-yinc))/2.
        else if (j==Y-1) then
          f11xy=(F(i11+xinc)-F(i11-xinc)-F(i11+xinc-yinc)
     1           +F(i11-xinc-yinc))/2.
        else
          print *,"   "
          print *,"   "
          print *,"bad code error!"
          print *,"   "
          print *,"   "
        end if

        Minv =  (/1.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,
     1 0.,0.,0.,0.,1.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,-3.,3.,0.,0.,-2.,
     2 -1.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,2.,-2.,0.,0.,1.,1.,0.,0.,0.,0.,
     3 0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,1.,0.,0.,0.,0.,0.,0.,
     4 0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,1.,0.,0.,0.,0.,0.,0.,
     5 0.,0.,0.,0.,-3.,3.,0.,0.,-2.,-1.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,
     6 2.,-2.,0.,0.,1.,1.,0.,0.,-3.,0.,3.,0.,0.,0.,0.,0.,-2.,0.,-1.,0.,
     7 0.,0.,0.,0.,0.,0.,0.,0.,-3.,0.,3.,0.,0.,0.,0.,0.,-2.,0.,-1.,0.,
     8 9.,-9.,-9.,9.,6.,3.,-6.,-3.,6.,-6.,3.,-3.,4.,2.,2.,1.,-6.,6.,6.,
     9 -6.,-3.,-3.,3.,3.,-4.,4.,-2.,2.,-2.,-2.,-1.,-1.,2.,0.,-2., 0.,0.,
     & 0.,0.,0.,1., 0.,1.,0.,0.,0.,0.,0.,0.,0.,0.,0.,2.,0.,-2.,0.,0.,0.,
     & 0.,0.,1.,0.,1.,0.,-6.,6.,6.,-6.,-4.,-2.,4.,2.,-3.,3.,-3.,3.,-2.,
     & -1.,-2.,-1.,4.,-4.,-4.,4.,2.,2.,-2.,-2.,2.,-2.,2.,-2.,2.,-2.,1.,1.,
     & 1.,1./)

        k(1) =f00
        k(2) =f10
        k(3) =f01
        k(4) =f11
        k(5) =f00x
        k(6) =f10x
        k(7) =f01x
        k(8) =f11x
        k(9) =f00y
        k(10) =f10y
        k(11)=f01y
        k(12)=f11y
        k(13)=f00xy
        k(14)=f10xy
        k(15)=f01xy
        k(16)=f11xy

        do ii = 1,16
          A(aa+ii)=0.
          do jj = 1,16
            A(aa+ii) = A(aa+ii)+ Minv((ii-1)*16+jj)*k(jj)
          end do
```

```
      end do
      end do
    end do
    end subroutine calculateBicubicAs


    subroutine updateF_star(vvf,dvvf,f_star_0,df_star,f_c,f_f,q_1,
   1         f_star)
    double precision vvf,dvvf,f_star_0,df_star,f_c,f_f,q_1
    double precision f_u,Kappa,f_star,start,finish,F,B1,B2,B3
    double precision inc,xl,xr,yl,yr
     f_u = 1.0/q_1
     Kappa = (f_u-f_c)/(f_f-f_c)
     f_star=f_star_0
     start=vvf-dvvf
     finish=vvf
     F=0.0005
     B1=f_c-F
     B2=f_c+F
     B3=f_f

C    normal
     if (start<=B1 ) then
       inc = MIN(finish, B1) - start
       f_star = f_star+inc
       if (df_star /= 10101.0) then
         df_star = 1.0
       end if
     end if

C    transition
     if (start<=B2 .and. finish>B1) then
       xl=MAX(start,B1)
       xr=MIN(finish,B2)
       yl=(Kappa-1.0)/2.0 *((xl-f_c)/F+1) + 1.
       yr=(Kappa-1.0)/2.0 *((xr-f_c)/F+1) + 1.
       f_star = f_star + (yl+yr)/2.0 * (xr-xl)
       if (df_star/=10101.0) then
         df_star = yr
       end if
     end if

C    coalescence
     if (start<=B3 .and. finish>B2)then
       inc = MIN(B3,finish) - MAX(start,B2)
       f_star = f_star+ Kappa*inc
       if (df_star/=10101.0)then
         df_star = Kappa
       end if
     end if

C    failure
     if (finish>B3) then
       if (df_star/=10101.0) then
         df_star = 0.0
       end if
     end if
    end subroutine updateF_star

    subroutine convertCauchyDerivToPK2deriv(CauchyDeriv,
   1         F,Jac,PK2deriv)
    double precision CauchyDeriv(9),F(9),PK2deriv(9),Jac
    integer i,j,k,l

       do i = 1,9
        PK2deriv(i) = 0.0
       end do
     do k=1,3
       do l = k,3
         do i=1,3
           do j = 1,3
           PK2deriv(3*(k-1)+l) = PK2deriv(3*(k-1)+l)
   1          +CauchyDeriv((i-1)*3+j)*F((k-1)*3+i)*F((l-1)*3+j)
           end do
         end do
       end do
     end do
     do k=1,3
       do l=1,k
         PK2deriv((k-1)*3+l) = PK2deriv((l-1)*3+k)
       end do
     end do
       do i=1,9
         PK2deriv(i)=PK2deriv(i)/Jac
       end do
    end subroutine convertCauchyDerivToPK2deriv

    subroutine solve_3x3(a,b,x)
    double precision a(9), C(3,3),x(3),b(3)
    C(1,1) = a(1)
    C(1,2) = a(2)
    C(1,3) = a(3)
    C(2,1) = a(4)
    C(2,2) = a(5)
    C(2,3) = a(6)
    C(3,1) = a(7)
    C(3,2) = a(8)
    C(3,3) = a(9)
```

```
 if (0 /= (C(1,1)*C(2,2)*C(3,3) - C(1,1)*C(3,2)*C(2,3)
1      - C(1,2)*C(2,1)*C(3,3) + C(2,1)*C(3,2)*C(1,3)
2       + C(3,1)*C(1,2)*C(2,3) - C(3,1)*C(2,2)*C(1,3)) ) then

 x(1) = (b(3)*C(2,1)*C(3,2) - b(3)*C(3,1)*C(2,2)
1         - b(2)*C(2,1)*C(3,3) + b(2)*C(3,1)*C(2,3)
2         + b(1)*C(2,2)*C(3,3) - b(1)*C(3,2)*C(2,3))
3         / (C(1,1)*C(2,2)*C(3,3) - C(1,1)*C(2,3)*C(3,2)
4         - C(1,2)*C(2,1)*C(3,3) + C(2,1)*C(3,2)*C(1,3)
5         + C(3,1)*C(1,2)*C(2,3) - C(3,1)*C(2,2)*C(1,3))

 x(2) = -(b(3)*C(1,1)*C(3,2) - b(3)*C(3,1)*C(1,2)
1         - b(2)*C(1,1)*C(3,3) + b(2)*C(1,3)*C(3,1)
2         + b(1)*C(1,2)*C(3,3) - b(1)*C(3,2)*C(1,3))
3         / (C(1,1)*C(2,2)*C(3,3) - C(1,1)*C(2,3)*C(3,2)
4         - C(1,2)*C(2,1)*C(3,3) + C(2,1)*C(3,2)*C(1,3)
5         + C(3,1)*C(1,2)*C(2,3) - C(1,3)*C(2,2)*C(3,1))

 x(3) = (b(3)*C(1,1)*C(2,2) - b(3)*C(1,2)*C(2,1)
1         - b(2)*C(1,1)*C(2,3) + b(2)*C(2,1)*C(1,3)
2         + b(1)*C(1,2)*C(2,3) - b(1)*C(2,2)*C(1,3))
3         / (C(1,1)*C(2,2)*C(3,3) - C(1,1)*C(3,2)*C(2,3)
4         - C(2,1)*C(1,2)*C(3,3) + C(2,1)*C(3,2)*C(1,3)
5         + C(3,1)*C(1,2)*C(2,3) - C(1,3)*C(2,2)*C(3,1))
 else
  return
 end if
 end subroutine solve_3x3

 subroutine matlib_determinant(A,det)
  double precision A(9),det
  det=A(1)*(A(5)*A(9)-A(6)*A(8))
1    -A(2)*(A(4)*A(9)-A(6)*A(7))
2    +A(3)*(A(4)*A(8)-A(5)*A(7))
 end subroutine matlib_determinant

 subroutine matlib_inverse(A,Ainv,d)
  double precision A(9),Ainv(9),d,dinv
  call matlib_determinant(A,d)
  if (abs(d)<1.e-10) then
    d=0.
    return
  else
    dinv=1./d
    Ainv(1) = dinv*( A(5)*A(9)-A(6)*A(8))
    Ainv(2) = dinv*(-A(2)*A(9)+A(3)*A(8))
    Ainv(3) = dinv*( A(2)*A(6)-A(3)*A(5))
    Ainv(4) = dinv*(-A(4)*A(9)+A(6)*A(7))
    Ainv(5) = dinv*( A(1)*A(9)-A(3)*A(7))
    Ainv(6) = dinv*(-A(1)*A(6)+A(3)*A(4))
    Ainv(7) = dinv*( A(4)*A(8)-A(5)*A(7))
    Ainv(8) = dinv*(-A(1)*A(8)+A(2)*A(7))
    Ainv(9) = dinv*( A(1)*A(5)-A(2)*A(4))
  end if
end subroutine matlib_inverse

subroutine matlib_mulss(A,B,C)
  double precision A(9),B(9),C(9)
  C(1) = A(1) * B(1) + A(4) * B(2) + A(7) * B(3)
  C(2) = A(2) * B(1) + A(5) * B(2) + A(8) * B(3)
  C(3) = A(3) * B(1) + A(6) * B(2) + A(9) * B(3)
  C(4) = A(1) * B(4) + A(4) * B(5) + A(7) * B(6)
  C(5) = A(2) * B(4) + A(5) * B(5) + A(8) * B(6)
  C(6) = A(3) * B(4) + A(6) * B(5) + A(9) * B(6)
  C(7) = A(1) * B(7) + A(4) * B(8) + A(7) * B(9)
  C(8) = A(2) * B(7) + A(5) * B(8) + A(8) * B(9)
  C(9) = A(3) * B(7) + A(6) * B(8) + A(9) * B(9)
end subroutine matlib_mulss

subroutine matlib_mulst(A,B,C)
  double precision A(9),B(9),C(9)
  C(1) = A(1) * B(1) + A(4) * B(4) + A(7) * B(7)
  C(2) = A(2) * B(1) + A(5) * B(4) + A(8) * B(7)
  C(3) = A(3) * B(1) + A(6) * B(4) + A(9) * B(7)
  C(4) = A(1) * B(2) + A(4) * B(5) + A(7) * B(8)
  C(5) = A(2) * B(2) + A(5) * B(5) + A(8) * B(8)
  C(6) = A(3) * B(2) + A(6) * B(5) + A(9) * B(8)
  C(7) = A(1) * B(3) + A(4) * B(6) + A(7) * B(9)
  C(8) = A(2) * B(3) + A(5) * B(6) + A(8) * B(9)
  C(9) = A(3) * B(3) + A(6) * B(6) + A(9) * B(9)
end subroutine matlib_mulst

subroutine matlib_mults(A,B,C)
  double precision A(9),B(9),C(9)
  C(1) = A(1) * B(1) + A(2) * B(2) + A(3) * B(3)
  C(2) = A(4) * B(1) + A(5) * B(2) + A(6) * B(3)
  C(3) = A(7) * B(1) + A(8) * B(2) + A(9) * B(3)
  C(4) = A(1) * B(4) + A(2) * B(5) + A(3) * B(6)
  C(5) = A(4) * B(4) + A(5) * B(5) + A(6) * B(6)
  C(6) = A(7) * B(4) + A(8) * B(5) + A(9) * B(6)
  C(7) = A(1) * B(7) + A(2) * B(8) + A(3) * B(9)
  C(8) = A(4) * B(7) + A(5) * B(8) + A(6) * B(9)
  C(9) = A(7) * B(7) + A(8) * B(8) + A(9) * B(9)
end subroutine matlib_mults


subroutine matlib_logmat(A,logA)
```

```
 double precision A(9),logA(9)
 integer i,j
 do i=1,3
  do j=1,3
   if (i==j) then
    logA(i+(j-1)*3)= A(i+(j-1)*3)-1.
   else
    logA(i+(j-1)*3)= A(i+(j-1)*3)
   end if
  end do
 end do
end subroutine matlib_logmat

subroutine matlib_expmat(A,logA)
 double precision A(9),logA(9)
 integer i,j
 do i=1,3
  do j=1,3
   if (i==j) then
    logA(i+(j-1)*3)= A(i+(j-1)*3)+1.
   else
    logA(i+(j-1)*3)= A(i+(j-1)*3)
   end if
  end do
 end do
end subroutine matlib_expmat

subroutine harden(pmat,eps,Sy)
 double precision pmat(8),eps,Sy
 double precision Y0,coef_Q0,n
 double precision coef,expo,val1,val2,x

 x=.0001
 Y0 = pmat(5)
 coef_Q0 = pmat(7)
 n  = pmat(6)
 if (eps<x) then
  val1 = x/coef_Q0
  expo = 1./n
  val2 =val1**expo
  Sy = Y0*(1.+eps/x*val2)
 else
  val1 = eps/coef_Q0
  expo = 1./n
  val2 = val1**expo
  Sy = Y0*(1.+val2)
 end if
end subroutine harden
```

```
subroutine stress(S,E,pmat)
 double precision S(9), E(9), pmat(8)
 double precision K3, G2, press, trace, vall
 integer i

 K3 = 3*pmat(3)
 G2 = pmat(2)+pmat(2)
 trace = (E(1)+E(5)+E(9))/3.
 press = K3*trace
 do i=1,9
  S(i) = G2*E(i)
 end do
 vall = press-G2*trace
 S(1) = S(1)+vall
 S(5) = S(5)+vall
 S(9) = S(9)+vall

end subroutine stres
```

Appendix B

The geometry input file used in Abaqus / Explicit

90^0 welded plate:

```
*Heading
** Job name: Job-8 Model name: 3mat_finner
** Generated by: Abaqus/CAE 6.14-2
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=finner_mesh
*Node
      ⋮
*Element, type=C3D10M
      ⋮
*Nset, nset=Set-Base
      ⋮
*Elset, elset=Set-Base, generate
  1018,  10923,      1
*Nset, nset=Set-BASE2
      ⋮
*Elset, elset=Set-BASE2, generate
 10924,  20507,      1
*Nset, nset=Set-HAZ1
      ⋮
*Elset, elset=Set-HAZ1, generate
 23759,  24745,      1
*Nset, nset=Set-HAZ2
      ⋮
*Elset, elset=Set-HAZ2, generate
    1,  1017,     1
*Nset, nset=Set-WELD
      ⋮
*Elset, elset=Set-WELD, generate
 20508,  23758,      1
** Section: HAZ
*Solid Section, elset=Set-HAZ2, material=HAZ
,
** Section: BASE
*Solid Section, elset=Set-Base, material=BASE
,
** Section: BASE
*Solid Section, elset=Set-BASE2, material=BASE
,
** Section: WELD
*Solid Section, elset=Set-WELD, material=WELD
,
** Section: HAZ
*Solid Section, elset=Set-HAZ1, material=HAZ
,
*End Part
**
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
```

```
*Instance, name=finner_mesh-1, part=finner_mesh
*End Instance
**
*Nset, nset=Set-1, instance=finner_mesh-1
      ⋮
*Elset, elset=Set-1, instance=finner_mesh-1
      ⋮
*Elset, elset=Set-2, instance=finner_mesh-1
      ⋮
*Nset, nset=Set-3, instance=finner_mesh-1
      ⋮
*Elset, elset=Set-3, instance=finner_mesh-1
      ⋮
*Nset, nset=Set-4, instance=finner_mesh-1
      ⋮
*Elset, elset=Set-4, instance=finner_mesh-1
      ⋮
*Nset, nset=Set-5, instance=finner_mesh-1
      ⋮
*Elset, elset=Set-5, instance=finner_mesh-1
      ⋮
*Nset, nset=Set-6, instance=finner_mesh-1
      ⋮
*Elset, elset=Set-6, instance=finner_mesh-1
      ⋮
*End Assembly
**
** MATERIALS
**
*Material, name=BASE
*Density
 2.66e-15,
*Depvar
     22,
*User Material, constants=1
0.,
*Material, name=HAZ
*Density
 2.66e-15,
*Depvar
     22,
*User Material, constants=1
0.,
*Material, name=WELD
*Density
 2.66e-15,
*Depvar
     22,
*User Material, constants=1
0.,
** ----------------------------------------------------------------
**
** STEP: Step-1
**
*Step, name=Step-1, nlgeom=YES
*Dynamic, Explicit
,1
*Bulk Viscosity
0.06, 1.2
** Mass Scaling: Semi-Automatic
**               Whole Model
*Fixed Mass Scaling, factor=4e+4
```

```
**
** BOUNDARY CONDITIONS
**
** Name: BC-1 Type: Velocity/Angular velocity
*Boundary, type=VELOCITY
Set-1, 1, 1, 1.016e4
** Name: BC-2 Type: Displacement/Rotation
*Boundary
Set-2, 2, 2
** Name: BC-3 Type: Displacement/Rotation
*Boundary
Set-3, 3, 3
** Name: BC-4 Type: Displacement/Rotation
*Boundary
Set-4, 2, 2
** Name: BC-5 Type: Displacement/Rotation
*Boundary
Set-5, 3, 3
** Name: BC-6 Type: Displacement/Rotation
*Boundary
Set-6, 1, 1
**
** OUTPUT REQUESTS
**
*Restart, write, number interval=50, time marks=yes
**
** FIELD OUTPUT: F-Output-2
**
*Output, field, number interval=1000
*Element Output, directions=YES
LE, S, TRIAX
**
** FIELD OUTPUT: F-Output-1
**
*Node Output
RT, UT
*Output, history, frequency=0
*End Step
```

30^0 welded plate:

```
*Heading
** Job name: Job-4 Model name: Model-1
** Generated by: Abaqus/CAE 6.14-2
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=P30degreeWeld
*End Part
**
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
*Instance, name=P30degreeWeld-1, part=P30degreeWeld
*Node
⋮
*Element, type=C3D10M
⋮
*Nset, nset=Set-1
⋮
*Elset, elset=Set-1, generate
  6626,  29217,      1
```

```
*Nset, nset=Set-2
⋮
*Elset, elset=Set-2
⋮
*Nset, nset=Set-3
⋮
*Elset, elset=Set-3, generate
  752,  5881,     1
** Section: Section-2
*Solid Section, elset=Set-2, material=HAZ
,
** Section: Section-3
*Solid Section, elset=Set-3, material=WELD
,
** Section: Section-1
*Solid Section, elset=Set-1, material=BASE
,
*End Instance
**
*Nset, nset=Set-1
⋮
*Elset, elset=Set-1, generate
  6626,  29217,      1
*Nset, nset=Set-2
⋮
*Elset, elset=Set-2
⋮
*Nset, nset=Set-3
⋮
*Elset, elset=Set-3, generate
  752,  5881,     1
*End Instance
**
*Nset, nset=Set-11, instance=P30degreeWeld-1
⋮
*Elset, elset=Set-11, instance=P30degreeWeld-1
⋮
*Nset, nset=Set-12, instance=P30degreeWeld-1
⋮
*Elset, elset=Set-12, instance=P30degreeWeld-1
⋮
*Nset, nset=Set-13, instance=P30degreeWeld-1
⋮
*Elset, elset=Set-13, instance=P30degreeWeld-1
⋮
*Nset, nset=Set-14, instance=P30degreeWeld-1
⋮
*Elset, elset=Set-14, instance=P30degreeWeld-1
⋮
*Nset, nset=Set-15, instance=P30degreeWeld-1
⋮
*Elset, elset=Set-15, instance=P30degreeWeld-1
⋮
*Nset, nset=Set-16, instance=P30degreeWeld-1
⋮
*Nset, nset=Set-23, instance=P30degreeWeld-1
⋮
*Elset, elset=Set-23, instance=P30degreeWeld-1
⋮
*Nset, nset=Set-24, instance=P30degreeWeld-1
⋮
```

```
*Elset, elset=Set-24, instance=P30degreeWeld-1
⋮

*End Assembly
** MATERIALS
**
*Material, name=BASE
*Density
 2.66e-15,
*Depvar
     22,
*User Material, constants=1
0.,
*Material, name=HAZ
*Density
 2.66e-15,
*Depvar
     22,
*User Material, constants=1
0.,
*Material, name=WELD
*Density
 2.66e-15,
*Depvar
     22,
*User Material, constants=1
0.,
** ----------------------------------------------------------------
**
** STEP: Step-1
**
*Step, name=Step-1, nlgeom=YES
*Dynamic, Explicit
, 2
*Bulk Viscosity
0.06, 1.2
** Mass Scaling: Semi-Automatic
**               Whole Model
*Fixed Mass Scaling, factor=4e+4
**
** BOUNDARY CONDITIONS
**
** Name: BC-1 Type: Displacement/Rotation
*Boundary
Set-11, 1, 1
** Name: BC-2 Type: Displacement/Rotation
*Boundary
Set-12, 3, 3
** Name: BC-3 Type: Displacement/Rotation
*Boundary
Set-13, 2, 2
** Name: BC-4 Type: Displacement/Rotation
*Boundary
Set-14, 1, 1
** Name: BC-5 Type: Displacement/Rotation
*Boundary
Set-15, 3, 3
** Name: BC-6 Type: Velocity/Angular velocity
*Boundary, type=VELOCITY
Set-16, 2, 2, 1.016e4
**
** OUTPUT REQUESTS
**
*Restart, write, number interval=50, time marks=yes
```

```
**
** FIELD OUTPUT: F-Output-2
*
*Output, field, number interval=1000
*Element Output, directions=YES
LE, S, TRIAX
**
** FIELD OUTPUT: F-Output-1
**
*Node Output
RT, UT
*Output, history, frequency=0
*End Step
```

Appendix C

The material input parameters used for the modGTN model are listed in this section as follow:

1. The VUMAT selects the modGTN model for computation
2. Density, Young's modulus, Poisson's ratio, initial yield stress, hardening exponent, hardening coefficient, and initial void volume fraction
3. Total number of triaxialities fitted, lower bound of triaxiality range, upper bound of triaxiality range
4. Total number of Lode parameter fitted, lower bound of Lode parameter range, upper bound of Lode parameter range
5. Place holder in Fortran 77 language
6. The first matrix is the fitted $q_1(T, \xi)$ values. Its row is number of triaxility fitted in order, and its column is number of Lode parameter fitted in order
7. The second matrix is the fitted $q_2(T, \xi)$ values
8. The third matrix is the fitted $q_2(T, \xi)$ values

BASE material parameters:

```
1
2660e-18 6.84e4 0.3 123.0 5.06 .007 0.03
8 0.4 1.8
11 -1.0 1.0
279
 0.198000 0.386000 0.282000 0.364000 0.490000 0.398000 0.508000 0.478000 0.468000 0.364000 0.300000
 0.198000 0.300000 0.340000 0.374000 0.572000 0.588000 0.644000 0.612000 0.398000 0.538000 0.370000
 0.198000 0.444000 0.290000 0.400000 0.470000 0.476000 0.548000 0.568000 0.582000 0.520000 0.458000
 0.330000 0.372000 0.460000 0.548000 0.614000 0.634000 0.644000 0.652000 0.668000 0.658000 0.610000
 0.436000 0.482000 0.588000 0.630000 0.730000 0.752000 0.760000 0.794000 0.786000 0.728000 0.716000
 0.536000 0.584000 0.654000 0.778000 0.798000 0.866000 0.862000 0.832000 0.864000 0.844000 0.786000
 0.624000 0.616000 0.718000 0.810000 0.914000 0.908000 1.046000 0.936000 0.958000 0.922000 0.876000
 0.702000 0.688000 0.808000 0.876000 0.962000 1.020000 1.046000 1.066000 1.052000 1.002000 0.960000
279
 2.114000 2.070000 2.192000 1.972000 1.748000 1.960000 1.788000 1.862000 1.896000 2.102000 2.230000
 2.114000 1.960000 1.892000 1.846000 1.558000 1.564000 1.534000 1.576000 2.070000 1.680000 1.918000
 2.114000 1.584000 1.972000 1.864000 1.810000 1.830000 1.764000 1.742000 1.726000 1.716000 1.746000
 1.700000 1.640000 1.588000 1.538000 1.502000 1.510000 1.518000 1.520000 1.504000 1.504000 1.520000
 1.488000 1.466000 1.416000 1.416000 1.368000 1.374000 1.382000 1.364000 1.374000 1.422000 1.390000
 1.362000 1.344000 1.338000 1.284000 1.298000 1.272000 1.290000 1.318000 1.296000 1.304000 1.316000
 1.272000 1.302000 1.274000 1.252000 1.212000 1.234000 1.172000 1.234000 1.224000 1.240000 1.248000
 1.212000 1.246000 1.204000 1.200000 1.178000 1.158000 1.156000 1.164000 1.174000 1.192000 1.202000
279
 0.143319 0.143319 0.057688 0.050905 0.048923 0.051005 0.049921 0.049763 0.051031 0.052595 0.060004
 0.087831 0.065694 0.060625 0.057641 0.052242 0.051537 0.052813 0.052320 0.064530 0.055009 0.061355
 0.087831 0.065720 0.070203 0.063211 0.059964 0.061661 0.057317 0.055373 0.055307 0.055781 0.059594
 0.071511 0.054984 0.054127 0.052583 0.050413 0.050779 0.050619 0.050913 0.050063 0.050725 0.056127
 0.064618 0.051818 0.050641 0.049194 0.048641 0.048546 0.048735 0.048098 0.048552 0.050298 0.049885
 0.061142 0.049142 0.048757 0.047857 0.048343 0.047684 0.047674 0.047698 0.046870 0.046845 0.049705
 0.056141 0.048654 0.046640 0.048527 0.047026 0.047066 0.045857 0.046818 0.047298 0.046977 0.047660
 0.054101 0.046396 0.045163 0.047491 0.046238 0.046105 0.046329 0.046927 0.045517 0.045807 0.046406
279
 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25
```

HAZ material parameters:

```
1
2.6600e-15 6.8400e+04 0.3000 12.0000 5.0700 1.2300e-08 0.0300
8 0.4000 1.8000
11 -1.0000 1.0000
279
 0.002486 0.072228 0.082796 0.238728 0.261986 0.356023 0.448465 0.516444 0.525104 0.550230 0.462705
 0.002849 0.066067 0.151598 0.250886 0.310980 0.405248 0.484036 0.539275 0.565893 0.590239 0.444734
 0.116250 0.388696 0.478795 0.188454 0.532067 0.602893 0.618636 0.640277 0.704823 0.533990 0.541968
 0.307404 0.311157 0.357052 0.443462 0.521064 0.510016 0.640800 0.697695 0.684681 0.655211 0.688118
 0.447492 0.432760 0.566520 0.597320 0.653797 0.691854 0.792772 0.809274 0.837824 0.855480 0.811634
 0.592334 0.573019 0.767661 0.765221 0.831935 0.922369 0.951138 0.895012 0.943564 0.944489 0.916827
 0.715035 0.669328 0.853700 0.865512 0.919323 0.985071 1.005963 1.029408 1.000383 1.034277 1.014860
 0.817519 0.812604 0.994076 0.999615 1.099415 1.091589 1.066653 1.171963 1.158874 1.143869 1.117246
279
 3.807433 1.868489 3.632572 2.148386 2.445709 2.108785 1.843925 1.741438 1.710496 1.509616 1.847548
 3.878999 1.906597 2.052225 2.332454 1.873462 1.698773 1.636787 1.604138 1.518661 1.499417 1.727197
 2.352621 1.419549 1.548620 2.012333 1.491998 1.445080 1.421391 1.422255 1.352465 1.611854 1.532728
 1.749639 1.636157 1.656343 1.562114 1.515400 1.531113 1.408251 1.365850 1.370892 1.410521 1.352574
 1.534835 1.428107 1.364495 1.372171 1.345141 1.331614 1.283700 1.290979 1.258511 1.268057 1.252813
 1.297631 1.294417 1.218980 1.231004 1.220646 1.164570 1.163161 1.207419 1.183122 1.181602 1.175577
 1.280532 1.209762 1.164982 1.175331 1.157776 1.154024 1.165040 1.148814 1.158783 1.160469 1.125608
 1.100636 1.113713 1.076101 1.079975 1.061347 1.063883 1.082847 1.052552 1.057812 1.062053 1.067527
279
 0.000000 0.000000 0.096418 0.065353 0.079994 0.072767 0.057572 0.058413 0.059774 0.060510 0.073381
 1.000000 0.121944 0.070551 0.084115 0.065042 0.060910 0.061704 0.061906 0.061019 0.065609 0.073022
 0.125160 0.041422 0.052202 0.082477 0.067817 0.066646 0.066813 0.067450 0.065982 0.078103 0.076118
 0.083060 0.085219 0.078489 0.083419 0.083534 0.077580 0.067293 0.065422 0.065272 0.068180 0.063596
 0.087672 0.069164 0.066370 0.065289 0.065145 0.069261 0.063150 0.062401 0.062381 0.062489 0.063556
 0.067345 0.061797 0.060335 0.059997 0.060797 0.056216 0.060582 0.059554 0.059077 0.060475 0.061069
 0.069089 0.054058 0.053280 0.052572 0.053321 0.052572 0.052638 0.053418 0.052946 0.053699 0.053301
 0.054643 0.050992 0.051686 0.051749 0.051986 0.051878 0.051675 0.051652 0.053235 0.051773 0.053098
279
 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
```

WELD material parameters:

```
1
2.6600e-15 6.8400e+04 0.3000 12.0000 5.0700 1.2300e-08 0.0870
8 0.4000 1.8000
11 -1.0000 1.0000
279
1.440728 0.476431 0.426420 0.496075 0.560121 0.619458 0.671644 0.736479 0.697979 0.690157 0.637792
0.961191 0.414905 0.410713 0.499745 0.561439 0.609357 0.658149 0.728602 0.691368 0.671972 0.657975
0.646287 0.514474 0.483706 0.633325 0.694796 0.753569 0.773394 1.014240 0.827416 0.839389 0.841815
0.572232 0.657947 0.735548 0.786913 0.839198 0.903916 0.925223 0.923176 0.946825 1.026488 0.991000
0.782739 0.764677 0.916542 0.976924 0.984194 1.070106 1.057352 1.056272 1.071057 1.103714 1.202968
0.903371 0.897178 0.996615 1.056703 1.112906 1.180888 1.205769 1.237804 1.256492 1.211319 1.201096
0.948092 1.083271 1.140168 1.226724 1.274438 1.364857 1.282727 1.303262 1.361367 1.315793 1.328136
1.103823 1.170006 1.282205 1.321351 1.445447 1.470643 1.460891 1.468365 1.511782 1.522500 1.428178
279
0.618566 1.474358 1.337441 1.404070 1.182333 1.173973 1.282544 1.222168 1.355589 1.314429 1.167211
0.608949 1.386141 1.392575 1.233274 1.201820 1.226528 1.205137 1.146939 1.201837 1.262651 1.152165
1.159253 1.298976 1.371379 1.172518 1.153853 1.132379 1.107001 1.003001 1.087352 1.092728 1.119086
1.229099 1.110145 1.076498 1.076597 1.054254 1.040166 1.055270 1.033487 1.016037 0.995207 0.999261
0.951179 1.015418 0.958457 0.949991 0.932982 0.936835 0.933785 0.956519 0.961606 0.926184 0.896338
0.921345 0.928446 0.899051 0.885282 0.883776 0.860894 0.876031 0.852746 0.857651 0.892257 0.885970
0.829336 0.797517 0.825396 0.807526 0.775993 0.778835 0.827362 0.828435 0.807453 0.814529 0.830069
0.817512 0.813905 0.769153 0.765439 0.745647 0.751291 0.761243 0.743113 0.744100 0.750609 0.784931
279
0.071684 0.092260 0.097377 0.097701 0.098478 0.098306 0.098404 0.098352 0.098536 0.098363 0.098595
0.076786 0.097096 0.101375 0.099503 0.101678 0.102269 0.098782 0.103971 0.098065 0.100502 0.102892
0.100673 0.106468 0.108858 0.104933 0.104244 0.103518 0.101862 0.099336 0.100305 0.102403 0.107079
0.105563 0.099496 0.100493 0.102816 0.099675 0.101307 0.101410 0.096681 0.100758 0.099644 0.101571
0.100950 0.098867 0.098351 0.097848 0.098066 0.097413 0.097638 0.099304 0.100461 0.097655 0.098298
0.099832 0.097254 0.097152 0.097067 0.097496 0.096707 0.098093 0.096877 0.096860 0.097612 0.097035
0.099729 0.097223 0.096919 0.096959 0.096771 0.096968 0.096815 0.096430 0.096431 0.096505 0.097182
0.098948 0.096620 0.095788 0.095451 0.095793 0.096450 0.096138 0.095569 0.095684 0.096202 0.096924
279
0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000 0.250000
```