REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
15-10-2007	FINAL	15-7-2004 to 15-7-2007

4. TITLE AND SUBTITLE

PRECISION COMPOSITE SPACE STRUCTURES

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-04-1-0445

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Garnich, Mark, R
Long, David
Venkata . Akula. MK
Fitch. John. F
Liu, Pu

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Wyoming
Office of Research Dept. 3355
1000 E. University Ave.
Laramie, WY 82071

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AF Office of Scientific Research
4015 Wilson Blvd Room 713
Arlington, VA 22203-1954

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This project was aimed at developing analytical approaches to improving and better understanding the dimensional stability of composite space structures under thermal environments and changing material behavior due to microscopic damage. A comprehensive review of the literature on damage modeling of polymer matrix composite laminates was conducted. Damage due to matrix cracking was characterized using micromechanics. Aspects of predicting and utilizing degraded properties at the fiber and matrix level were studied. A computational approach to optimization of structure dimensional stability by introduction of "anti-distortion appliqués" was developed for minimizing thermally induced instability. The concept involves adding material to offset and eliminate measured instabilities. The concept was demonstrated but the approach is limited by the current lack of precision measurement systems for large structures.

15. SUBJECT TERMS
Composite materials, dimensional stability, microcracking, thermal expansion, space structures, degradation modeling

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Victor Giurgiutiu
Unclassified	Unclassified	Unclassified	UU		**19b. TELEPHONE NUMBER** *(include area code)* 703-696-7259

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Advanced civilian and defense space missions require substantially larger and more stable structures to support higher resolution interferometric optical benches, optical mirrors, microwave frequency reflectors, as well as other high performance space based applications. Some of these applications demand structure dimensional stability that is well beyond the capabilities of existing technology. Thermal expansion and degradation of properties due to damage are two of the most problematic sources for loss of dimensional precision in space structures. Graphite fiber reinforced polymers have emerged as one of the best materials for dimensionally stable space structures. This is due to their combination of high stiffness, high strength, low density and adjustable properties; including CTE. In fact, due to the slight negative axial CTE of the fibers it is possible to design laminates with theoretical zero CTE in the plane of the laminate. The out of plane CTE is still large and problematic. Also, the "theoretical" zero CTE is only approximated as well as the manufacturing precision allows. That is, deviations in fiber volume fraction, fiber angles, constituent properties, as well as temperature dependence of properties all contribute to real parts having some finite CTE. Using current technology the general approach to precision structure fabrication is; use precision fabrication technology to fabricate parts for an assembly, test each part for CTE, reject parts with unacceptable CTE, assemble the structure and use numerical models to predict the assembly performance in the expected environment. Joining technology is a huge issue with this approach as adhesives, joint materials, and joint geometric effects can be the dominant contributors to dimensional instability.

This project addressed the issue of dimensional instability from two perspectives. The primary objective was to develop and test a composite space structure design modification strategy based on intentional defects to achieve a "sum of the defects equals zero" result. That is, small material additions referred to as "anti-distortion appliqués" were considered as modifications to the structures thermomechanical behavior with the goal of eliminating unwanted thermal deformations and thus improving the structures dimensional stability. A composite T-joint structure was selected as a demonstration problem. A finite element model was developed to characterize the structure's thermomechanical properties. Initially some assumed thermal distortions were shown to be reduced by trial and error positioning of anti-distortion appliqués. Subsequently, optimization software was employed to automatically adjust appliqué design parameters so as to minimize an objective function based on distortional displacements in the structure. It was found that an effective and practical approach was to introduce a small number of appliqués and simply set there location coordinates as design parameters. Reductions of about 90% in the objective function were realized. Finally, the T-joint structure was fabricated for testing by Alliant Techsystems Corporation. The T-joint was modified to include an adhesively bonded sheet of titanium foil that would induce measurable distortions of the vertical web on the T-joint. Photogrammetry was used to measure displacements at target points on the vertical web during a 230 °C temperature change. The finite element model combined with the a program written to drive the optimization software was used to optimize the location of two appliqués to minimize the out of plane distortions of the vertical web at the target points. The computer code to generate these models is listed in Appendix C. The optimization predicted a 61% reduction in the objective function. The relatively poor improvement was blamed in part on the limited precision of the photogrammetry equipment. Two aluminum appliqués were bonded to the vertical web in optimized positions. However the test to measure reduction in distortion was not

finished before the end of the project due to low priority of scheduling in the environmental chamber.

The anti-distortion appliqués methodology was demonstrated and shown to have theoretical potential for order of magnitude improvement in dimensional stability in thermal environments. However, the modeling is computationally expensive and large structures would require significant computational resources. Also, there seems to be no presently available experimental techniques for precision measurement of structure distortional displacements. Since the present approach requires such measurements it will have limited applicability until better measurement technology is developed.

Another issue with polymer composites is the susceptibility of the polymer to damage. Even though the damage may be insignificant to structural integrity, the small changes in effective composite properties can lead to dimensional instability. In the cryogenic environment of space polymers become susceptible to microcracking due to the large internal stresses that develop due to temperature change. A study was conducted on the effects of matrix cracking on effective lamina properties. A literature review [Akula and Garnich (2007a,b)] assessed to the various models available for simulating degraded properties. Most all of these were found to be directed at progressive failure analysis and lacked the fidelity needed to be useful for dimensional stability studies. Hence, a modeling effort was directed at predicting effective residual thermomechanical properties for two types of matrix cracking. The details of this effort were documented by Akula (2007). Both types of cracks were assumed transverse to the lamina; one perpendicular to the fibers and the other parallel to the fibers. Periodicity of cracking was assumed and unit cells were modeled using the finite element method. Computer code to generate these models is listed in Appendix A and Appendix B. Comparison with data in the literature indicates the models do an excellent job of predicting residual properties. Cases of poor agreement were consistently where the experiments were attempting to measure extremely small changes in stiffness. The models also exposed several subtle dependencies of the residual properties on adjacent undamaged ply parameters. This was explained by the observation that reduced effective properties are a function of crack opening behavior and the constraining effect of adjacent plies to limit that deformation is dependent on the adjacent ply properties.

Finally, with the perspective of efficient future modeling of composite structures from a more fundamental basis, an approach was developed to determine effective constituent properties based on the degraded lamina properties. Details of this study were documented by Akula and Garnich (2007c). A significant finding was that for matrix ply cracking both the fiber and matrix properties must be degraded to achieve effective lamina properties that were consistent with the average behavior predicted by the finite element unit cell model.

1.0 INTRODUCTION

The technical challenge of precision space structures is to establish and maintain the geometric precision of equipment that relies on precise geometry for operational performance. Dimensional instabilities can be static or dynamic. The primary focus of this project was the static instability associated with thermal strains and the most common material damage mode; matrix cracking. However, the project was also intended to examine alternate materials in the form of nano-reinforced resins and adhesives that may offer properties that can enhance the dimensional stability of composite structures.

Thermal instabilities occur as a result of either gradients in material properties in concert with a uniform temperature change, gradients in temperature in concert with nonzero thermal expansion properties, or combinations of both. Therefore, a material with a uniform zero coefficient of thermal expansion (CTE) would eliminate both sources of thermal instability. The ultimate minimization of thermal instabilities would come from a combination of materials and design methodology that would enable the extremely fine control of the thermal deformation behavior in mass efficient composite structures.

Closely related to CTE effects are effects of the coefficient of moisture expansion (CME). Moisture absorbed or desorbed from polymers causes expansion or contraction behavior that has adverse effects on the dimensional stability of composites [Prunty (1978)]. Off-gassing of residual resin volatiles can have a similar effect. In the space vacuum, desorption is generally the source of dimensional instability. Issues associated with CME were not directly addressed by this project. However, due to the similarities of CME and CTE effects, technologies for reducing CTE effects have applicability for reducing CME effects.

Perhaps the single greatest advance in the technology of dimensionally stable space structures was the design of graphite fiber composite laminates that achieve a nominally zero CTE. With the use of sufficiently high modulus graphite fibers and proper laminate design, the "zero" CTE can be unidirectional or isotropic in the plane of the laminate. The CTE of the fiber, matrix, and individual lamina effectively interact so that a theoretical CTE of zero is possible. In practice, CTE less than 1 ppm/°C are routine and can be controlled to less than 0.1 ppm/°C. The transverse, or through thickness CTE remains large (~20 ppm/°C) and is problematic in complex structures where its effects are generally unavoidable. This basic technology was developed and applied to a number of space structures in the 1970s [e.g. Prunty (1978)]. Since then, only incremental improvements have been made in space structure thermal stability.

In some cases where difficult stability criteria are to be met, a brute force approach is employed. That is, multiples of each of the structure components or subassemblies is fabricated so that through individual testing the best of the batch (e.g. lowest CTE) can be selected for use in the structure. For example, Maji et al. (2001), who designed a spider truss support structure for a system of mirrors in an optical telescope, found that their pultruded graphite-epoxy truss members had a CTE that ranged from about 0.1 to 0.3 ppm/ºC. They attributed the scatter to statistical variations in fiber misalignment and fiber volume fraction. Though effective, this method of testing and selecting only the best parts is clearly wasteful, and does nothing to address the problems associated with subsequent joining.

Dodson and Rule (1989) discussed a few important factors affecting thermal stability of space flight optical benches. Their analysis of graphite/epoxy benches has indicated the selection of core types in sandwich structures, typical joint configurations, adhesive effects and moisture effects are all important considerations in design. Shin et al. (2001) have analyzed the thermal distortion of an orbiting solar array including composite material degradation effects. It was observed in their study that the strength, stiffness and CTEs of graphite/epoxy composite materials after exposure to simulated low earth orbit environments decreased in proportion to increasing thermal cycles. It was also found that the solar arrays with composite face sheets were advantageous in view of weight savings, temperature distribution and thermal distortion when compared to those with aluminum face sheets. In a discussion on the design of composite surrogate mirror support structure, Maji et al. (2001) analyzed the mirror surface figure errors caused by temperature fluctuation and suggested a design to be a good one if it achieves the best balance between mass, dynamic properties and thermal deformation.

To minimize the thermal deformation, it is generally accepted that materials with minimum CTE should be used. As early as late 1970s, Prunty (1978) discussed several types of dimensionally stable graphite composites for spacecraft structures. He suggested that the adoption of composites for planned spacecraft structures can achieve significant benefits in stiffness-critical and strength-critical applications and can be a virtual necessity where extreme dimensional stability is required.

Compared with several decades ago, there are now a lot more applications for advanced composite materials. Composites with a near zero CTE can meet stringent dimensional requirements. Among various composite materials, graphite fiber laminated composites have been the primary material for precision space structures due to their high structural mass efficiency and capability for theoretical zero in plane coefficient of thermal expansion. Graphite fiber reinforced polymers (GFRP) with high specific strengths, moduli, and design flexibility are widely developed and used as structural materials of modern aircraft and spacecraft. Through proper design, it is not only possible to have near zero CTE, but also to design the CTE of the composite to match that of other system components to minimize thermal mismatch and the resulting thermal distortions.

Grimaldi et al. (1989) have discussed several types of composite materials in designing satellite antenna structures. All the composite materials used for antenna structures share the need for high strength and low CTE. Ishikawa et al. (1989) have analyzed the thermal behavior of graphite-epoxy laminates with almost null coefficients of thermal expansion under a wide range of temperature. They proposed a lamination tailoring technique to control the CTEs of graphite-epoxy composites. This technique consists of two concepts of the thermoelastic invariants and the lamination parameters. More recently, Bansemir and Haider (1998) have reviewed the development of fiber composite structures for space applications. They argue that improvement of materials evaluation methods and analysis techniques can be applied for optimal tradeoffs between properties such as CTE, elastic modulus and thermal conductivity in fiber-reinforced composites.

In some applications uniform temperature changes may not cause problems since the associated uniform thermal expansion may cause little or no distortion. However, spatial temperature gradients may cause serious distortions. In these cases materials with high thermal conductivity that minimized temperature gradients are advantageous. The combination of low CTE (α) and high thermal conductivity (K) is best, and the ratio (α/K) is a useful measure of material performance in this regard.

Unique to this project was the parallel modeling and experimental efforts largely facilitated by the experimental contributions of industry partner ATK. The modeling had two focuses. The first, explained below was a global structure modeling effort aimed at reducing structure instability behavior after fabrication. The second studied material behavior from a microstructure perspective considering microdamage (microcracks) and nanocomposite structure-property relationships [e.g. Fertig and Garnich (2004)].

Form a structure modeling perspective, previous research in *design of structures* to achieve higher levels of dimensional stability have taken a completely different approach than was taken in this project. Most previous work has been performed from a local perspective where initial design and fabrication flaws were identified and attempts are made to minimize them. That is, the location of problematic deformation, or issues of laminate design, are isolated and the designer attempts to devise a strategy to reduce or eliminate that local source [e.g. Dodson and Rule (1989), Yoon and Kim (2001), Bansemir and Haider (1998), Farmer et al. (1992)]. For example, the anisotropic CTE causes an angle change in curved laminates during a temperature change [Yoon and Kim (2001)].

In comparison, the approach developed here was to design from a global perspective while employing cost effective local strategies associated with current practice. The global approach reduces the thermal instabilities after initial fabrication and assembly. This was demonstrated through a combination of experiments and modeling. The experiments characterized the nature of overall thermal deformations and the models devise a set of "anti-distortion appliqués" that create offsetting thermal deformations. In effect, this creates offsetting forces within the structure in much the same way that the constituent materials and lamina do at the local material scales when designed for a net CTE of zero. A clear advantage of this approach is that adjustments can be made after initial fabrication so that success in meeting CTE requirements is not reliant on the initial perfection in materials and construction. The significant disadvantage is that it requires experimental deformation measurement technology that matches the performance objectives. For high precision structures such measurement technology is not yet available.

The mechanical and thermal properties of composite materials can be designed through micromechanical analysis. The required properties of materials can be achieved by controlling their texture at microscopic scales, by controlling the ply orientation, modifying the volume fraction of fibers, etc. For example, Fertig and Garnich (2004) have studied the influence of constituent properties and microstructural parameters on the tensile modulus of a polymer/clay nanocomposite. Nanocomposites have potential as matrix materials in more conventional micron scale composites. They proposed a multi-scale model to estimate the relative influence of constituent properties on the anisotropic tensile modulus. The model can be extended to predict the influence of components on the thermal properties of materials. Garnich and Karami (2004)

and Karami and Garnich (2005b) have studied the mechanical properties and thermoelastic behavior of wavy fiber composites. Their results show that the waviness has effects on both material stiffness properties and the coefficients of thermal expansion.

A significant portion of this project was directed at better understanding the relationship between microstructure and macroscopic thermo-mechanical properties. In particular, effort was focused on better understanding how the most common matrix damage modes affect properties. Two types of matrix cracks were studied using micromechanics/unit cell models to better understand how the effective properties of a laminate embedded lamina are modified by such damage. Kim et al. (2000) have shown that crack density in a cross-ply laminate can have a significant effect on the CTE and the variation in CTE due to ply cracking can be quantitatively predicted whether due to mechanical loading or thermal cycling. Also, a review of the literature was conducted to survey methods used to model the residual properties of damaged laminae.

In summary, this project relates to the current state of knowledge as follows. The project was centered on GFRP, the most effective and widely used material for dimensionally stable space structures. Alternative materials were considered only from the standpoint of engineered polymers and their multi-functionality either as enhanced adhesives or enhanced matrix materials. Material damage characterization through micromechanics modeling for modes of matrix cracking was also conducted. Unlike prior efforts, the project was not be restricted to limited local (e.g. joints) perspective of performance improvement but also developed a global design modification strategy with the potential to dramatically improve the thermal stability of complex composite space structures.

2.0 THE ANTI-DISTORTION APPLIQUÉ APPROACH

The sources of dimensional instability in space structures are numerous and can be both microscopic and macroscopic in nature. Instabilities can be functions of environmental changes, component geometry, composite lamina orientation, microcracking, moisture content, mechanical loading and other factors. Among various types of dimensional instability, thermal instability is one of the most commonly observed. Generally, thermally activated strains can create distortion in structures when the coefficients of thermal expansion (CTEs) vary among the components. Even a small CTE mismatch can lead to substantial distortion. Thus thermal effects are one of the most important sources of space structures' dimensional instability.

Macroscopically, the dimensional stability of structures can be improved through refined material processing, modifying the shapes, sizes and relative positions of components. Farmer et al. (1992) performed a thermal distortion analysis of an antenna-support truss in a simulated geosynchronous orbit environment. Their study has shown that some surface processing of truss elements, e.g., using coating or multilayer insulation, can achieve certain success in alleviating thermal distortion problems. In theory, if only the thermal effect from unwanted fabrication defects is measurable, then the structure can be rectified by the offsetting effect generated through components' geometrical changes. This possibility was the primary subject of this research.

2.1 Anti-Distortion Appliqués

It is important to carefully select the materials in order to avoid or minimize the dimensional instability of space structures. For example, there are many applications where a near-zero CTE materials can lead to significant benefits, but even with the use of materials with near-zero coefficient of thermal expansion (CTE), instability can still occur through the large temperature changes and the complication of temperature dependent CTE. Also, some new applications simply have extreme stability requirements that are beyond current technology. Another strategy for performance improvement is to improve the dimensional stability by modifying a structure's geometry. Intentional introduction of material may be used to work in a beneficiary way, i.e., structures can improve their static or dynamic performance through purposely designed and strategically located material additions which can cancel effects from unwanted manufacturing imperfections. Since these material additions are to cancel unwanted thermal deformations; they were referred to as *anti-distortion appliqués*.

Presented here is a study of how the dimensional stability can be improved by the use of anti-distortion appliqués. A joint structure, which is typical in composite space structures, was taken as a demonstration problem. The basic approach consists of a three phase analysis. First, a finite element model to predict the joint deformation due to thermal expansion was developed using the finite element analysis (FEA) software ABAQUS. Second, the actual part is tested in the laboratory under simulated operating environment to measure the adverse thermal distortions. In this case that meant lowering the temperature to simulate the cryogenic environment of space. Finally, the finite element model is exercised repeatedly as needed with material additions (anti-distortion appliqués) with the objective of eliminating the unwanted distortions by effectively introducing offsetting distortions. In simple cases this last step might be conducted by trial and error by an analyst. However, for more general application an optimization study can be automated with software that is adapted to modify and rerun the FEA over and over again until an optimized configuration of anti-distortion appliqués has been achieved. This last step was accomplished using the VisualDoc (2004) software which interrogates a text output file from the FEA and then modifies a text input file for the next FEA based on an optimization algorithm that seeks to minimize an objective function based on the output. In this case, the objective function involves displacements that quantify the distortion.

2.2 The Joint Structure

This research used a composite T-joint with reinforcing clips as a model problem to illustrate the concept of 'anti-distortion appliqués' do reduce the unwanted thermal distortion and thus improve a structure's dimensional stability. The significance of studying the basic joint structure has two basic aspects; first, some typical sources of adverse thermal distortions will be illustrated, and second, to conveniently show that such thermal distortions can be reduced through the addition of material as determined through FEA. Overall, a strategy is sought for systematic analysis and modification of composite structures to dramatically improve their dimensional stability. Fig. 1 shows the basic configuration of the composite T-joint structure.

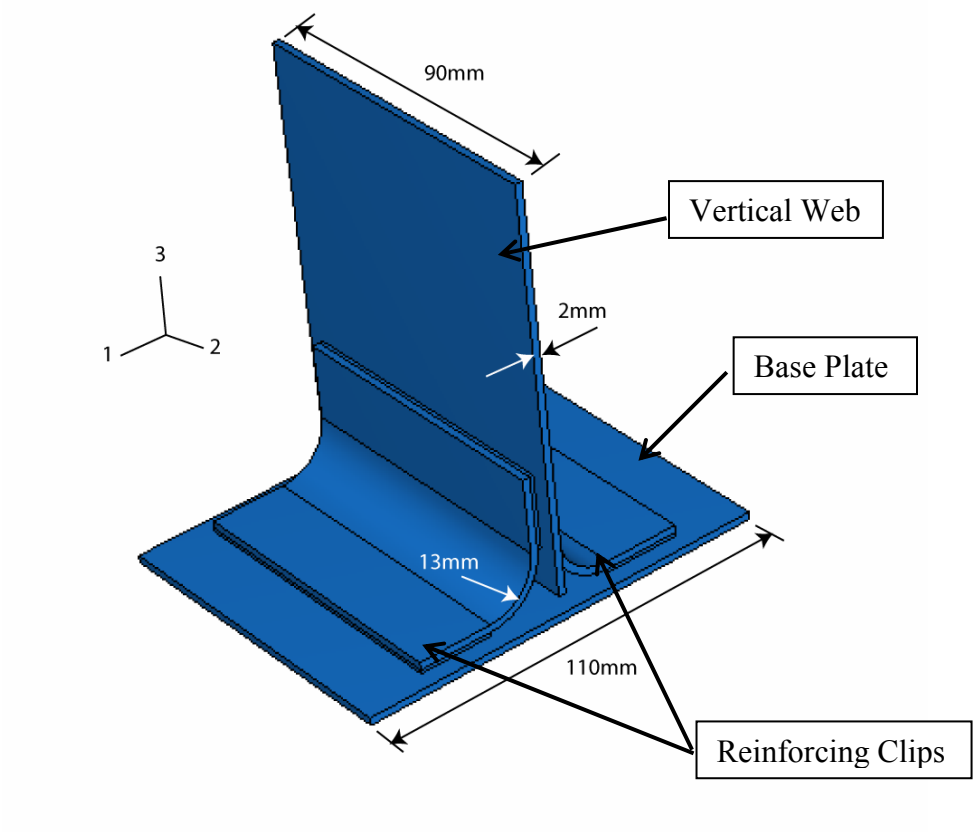


Figure 1. The T-Joint Structure

The horizontal and vertical composite plates consist of GFRP laminates. The two curved clips are also GFRP laminates used to support the joint structure. Thin film adhesives that connect the plates with the clips were explicitly included in the models. The structure is assumed to sustain a uniform temperature change. To analyze the thermal deformation use has been made of the FEA software ABAQUS. ABAQUS provides different mathematical options for modeling such 3-D thin structures. Either shell element or continuum element formulations can be selected to perform the analysis. For displacement solutions they give similar results for most example problems in the ABAQUS documentation. Depending on the thickness of modeled components, the two types of elements can be used alternatively but generally for laminated composite structures the shell elements are numerically much more efficient.

With the contemporary highly developed fabrication technology of composite materials, the negative CTE of fiber and positive CTE of matrix can be combined to manipulate and control the CTE of the formed composites. This can be done to obtain desired unidirectional properties or isotropic properties in the plane of the laminate. In this research, a transversely isotropic (quasi-isotropic) GRFP laminate was assumed. A negative CTE along the fiber direction in a ply (or fiber alone) and a positive CTE in the perpendicular direction were assumed. Through a certain combination of their volume fraction and ply angles, they can provide a theoretical null CTE for the in-plane material behavior [Ishikawa et al. (1989)]. A material with 62% fiber volume fraction and ply thickness of 0.127 mm was studied. The mechanical and thermal parameters of the composite [Matthews et al. (2000)] are listed on Table 1.

Table 1. The mechanical and thermal properties of the GFRP laminae

E_1(GPa)	E_2(GPa)	ν_{12}	ν_{23}	G_{12}(GPa)	G_{23}(GPa)	CTE α_1(ppm)	CTE α_2(ppm)
132	10.8	0.30	0.59	5.61	3.17	-0.77	25

The adhesives are assumed to be isotropic with a Young's modulus of 6.0 GPa and a Poisson's ratio of 0.3. The CTE of the adhesive was taken as 54 ppm/°C in the computations. To illustrate some potential sources of thermal distortion of the joint, the contribution of adhesive thickness and the clip misalignment was studied and discussed in the following sections.

2.3 Simulated thermal distortions

One of the attractive features of the anti-distortion appliqué methodology is that you do not need to know the source of the distortions; only that they are manifest by the application environment. However, the method does require actual distortion data. In the absence of experimental data simulated data can be used to demonstrate the method. Hence, the FEA was used to predict distortions due to assumed defects in manufacture. For the T-joint structure, the laminates were assumed perfectly fabricated as balanced and symmetric so no out of plane deformations would occur due to their properties. Yoon and Kim (2001) studied the process-induced distortions of carbon/epoxy curved laminates. Their work included the effect of anisotropic CTE in curved laminates with a temperature variation. The clips on the T-joint are curved but if the structure is symmetric then their effects will cancel and the deformations will be perfectly symmetric.

To simulate distortions two different manufacturing defects were assumed. First, the two clips were assumed misaligned and second the thickness of the adhesive was assumed unsymmetric for the two clips. Fig. 2 and Fig. 3 show the modeled joint distortion due to non-uniform adhesive layers and clip dislocation respectively.

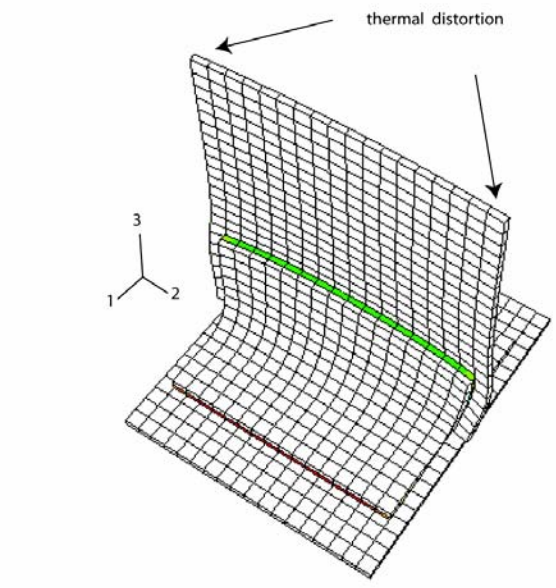


Figure 2. The distortion due to the inhomogeneous adhesives

What deformations are adverse to a structures performance will be application dependent. In this case, the difference between the 1-axis displacements of the two top corner nodes was to be minimized. To highlight the effect of clip misalignment on the thermal distortion, a laminae structure with 8-ply orientation $[0/30/-30/90]_s$ has been studied. If no manufacture defects exist, this arrangement of the orientation generates only symmetric thermal deformation and no distortion. With the effect from the misalignment, the modeling results indicate that the joint's thermal distortion is closely related to how much the two clips are misaligned. Table 2 shows the predicted distortion for a range of values of clip misalignment.

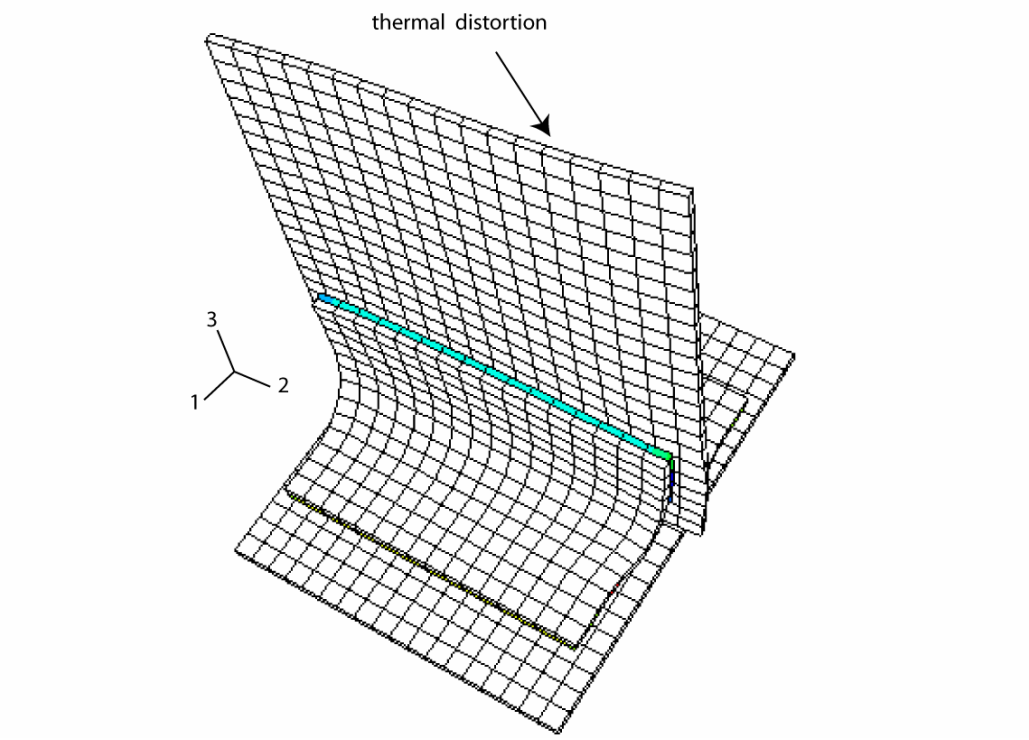


Figure 3. The distortion due to the clip dislocation

Table 2. A distortion parameter for a range of clip misalignment magnitudes

Misalignment [mm]	2.0	4.0	6.0	8.0	10.0
Distortion [mm]	0.025719	0.036654	0.053764	0.071521	0.092974

2.4 Anti-distortion Appliqués by Trial and Error

To reduce the effect from manufacturing imperfections of adhesives or clips, some purposely-designed material additions can be introduced onto the structure. With the present joint structure being relatively simple in geometry and the sources of the distortions known, using a trial-and-error approach is possible to test some appliqués. For the case where there exists a thickness difference between the adhesives on the two sides (Fig. 2), it was found that if the plate was modified with a protruding strip added on the side with the thicker adhesive, the thermal distortion can be reduced. Fig. 4 shows the analysis results for the modified structure.

A similar procedure was implemented for the case where there exists dislocation between the two clips. It was found that if two strips are attached to the plate with also a dislocation existing between them, then they will generate the required offsetting effect, which just cancels that caused by the clip dislocation. This result is shown in Fig. 5.

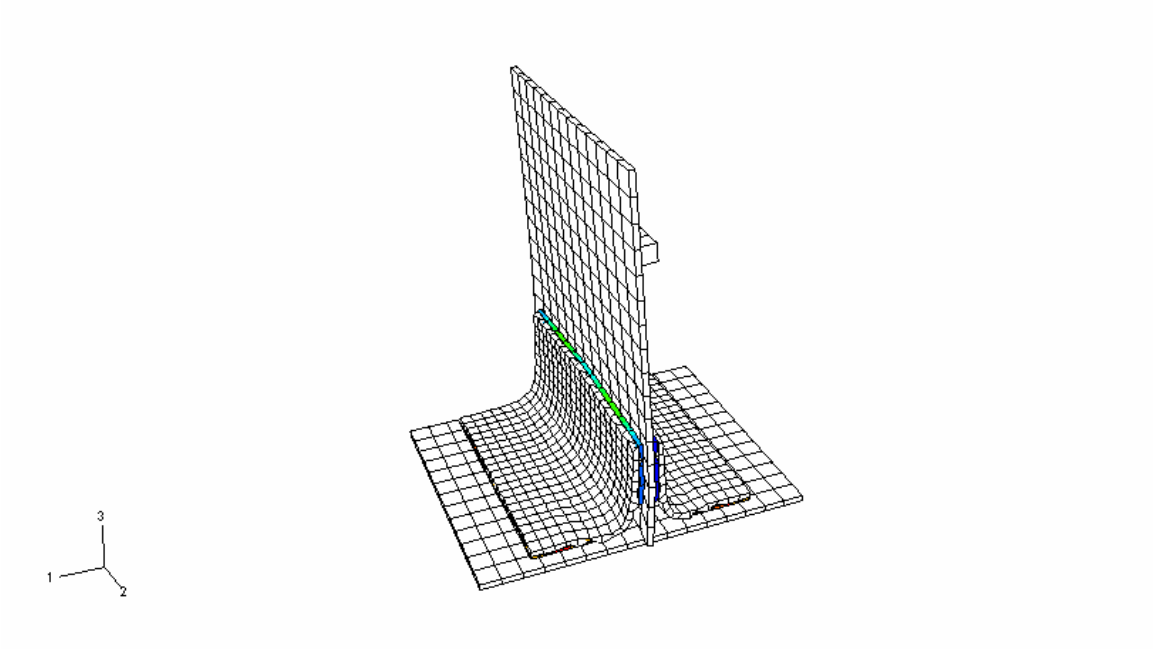


Figure 4. Eliminating distortion due to imbalanced adhesive by adding an appliqué

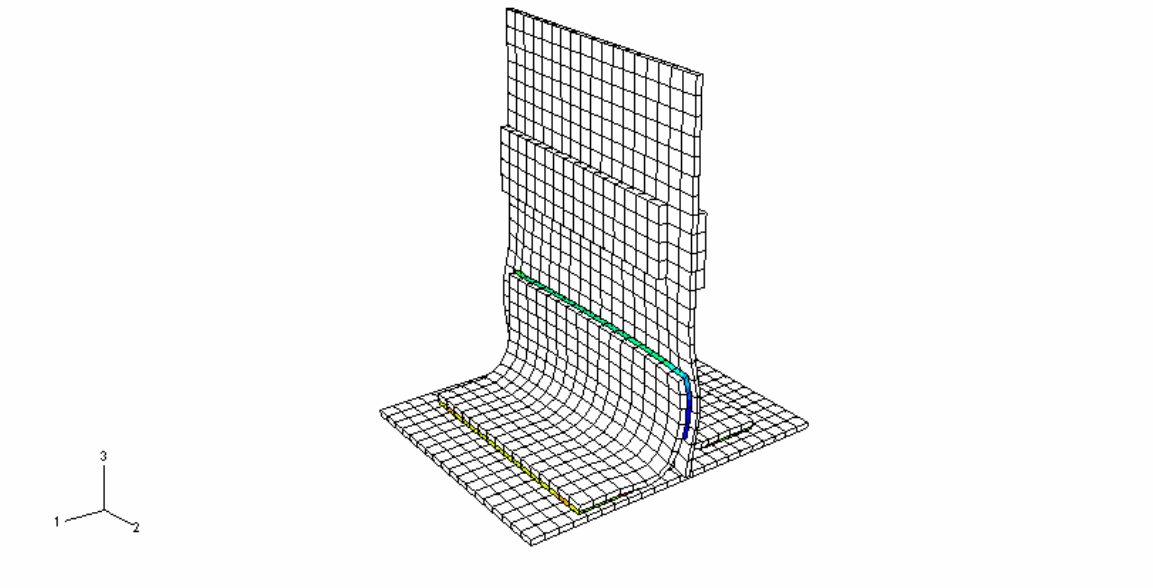


Figure 5. Eliminating distortion due to clip misalignment by adding two appliqués

The trial-and-error approach works easily only for simple structures where the deformation is not too complicated and especially when the source of the deformation is known. But even for such simple structures, trial-and-error can be an inefficient approach. Considering the numerous possibilities of various appliqué shapes and locations, a trial-and-error approach may not yield the best design of appliqués quantitatively, or it may be very time consuming or computational expensive, even though the method can in some ideal cases generate a close to perfect result. So for more complicated structures parametric optimization analysis is preferred. In the following section an optimization study will be discussed, still using the joint structure as an example.

2.5 Designing Appliqués Using an Optimization Procedure

Here, the design optimization was performed with ABAQUS working jointly with the optimization software VisualDOC (2004). ABAQUS is used for the thermo-mechanical analysis and VisualDOC provides an approach to search for the best array of appliqué parameters. Among the different algorithms used to find the optimum solution of an objective function, the methods in VisualDOC can be classified as gradient-based and non-gradient based approaches. The gradient-based approaches take the standard strategy of finding an extreme value of a function. They first calculate the derivatives of an objective function with respect to the design variables and then use the calculated derivatives, or the "gradients" to determine the search path to find the next good approximation of the optimum value to be determined. With enough iteration, the procedure will generally converge to the optimum values. The gradient-based approaches are usually computationally efficient in finding optimum solutions for continuous problems but do not work well for discrete/integer design problems where some parameters take values from a discrete set instead of continuous functions like typical field variables. The optimization problem studied in this research is generally of the mixed type where some design variables such as appliqué positions are continuous but others may be discrete such as the number of appliqués. A non-gradient based algorithm, namely the particle swarm optimization (PSO) [VisualDOC (2004)] algorithm was applied when the number of appliqués was a design variable. This algorithm can handle either continuous or discrete design variables or a mixed problem.

In the context of the present problem, design variables could include shape, position, size, physical properties, the number of appliqués, etc. In the following sections shape optimization and position optimization of the appliqués for the joint structure will be discussed using a few modeling examples.

2.5.1 Optimization with respect to appliqué shape

In this study two rectangular appliqués were attached to the T-joint vertical plate. In an attempt to optimize the appliqués' shape, perturbations were continually made to the corner points. The coordinates of all relevant nodes were changed so that the component's geometrical shape is correspondingly modified. For every new perturbed shape, ABAQUS performs an analysis of the deformation and feeds back the results to the optimization routine. This process continues until a minimization of the objective function for the structure has been achieved within the limitations of a fixed range of design variables. Fig. 6 shows two initial rectangular appliqués added onto one surface of the vertical web.

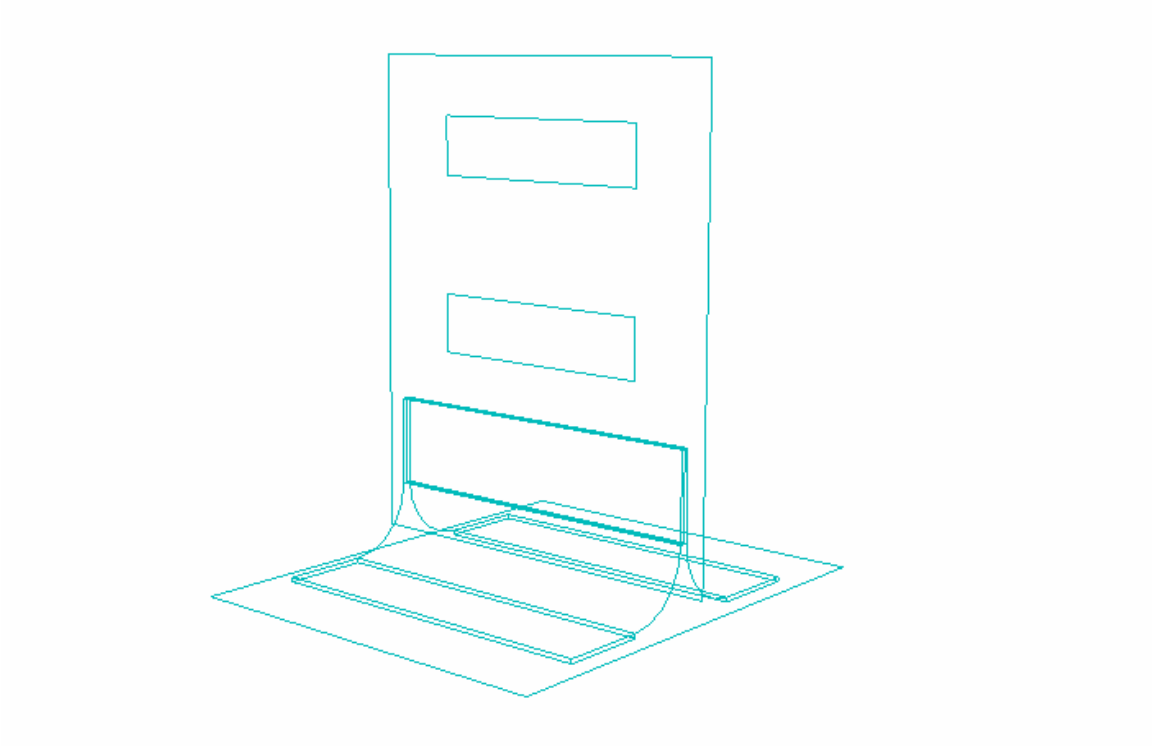


Figure 6. Two rectangular appliqués attached on the two surfaces of the plate

Isoparametric mapping was used to compute the coordinate changes of all inner nodes due to the effect from perturbations on the corner node locations of the appliqués. The parametric coordinates of the nodes are taken as the same as those used to calculate the shape functions in common finite element analysis. The perturbations to all node coordinates were calculated as follows [Leiva and Watson (1998), Candan et al. (2000)],

$$P_{xi} = \sum_{J} P_{xj} N_j (\xi_i, \eta_i, \zeta_i) \qquad (1a)$$

$$P_{yi} = \sum_{J} P_{yj} N_j (\xi_i, \eta_i, \zeta_i) \qquad (1b)$$

$$P_{zi} = \sum_{J} P_{zj} N_j (\xi_i, \eta_i, \zeta_i) \qquad (1c)$$

The summations in equations (1) are over all the perturbed points where P_{xj}, P_{yj} and P_{zj} are the perturbations on the coordinates of these points. ξ_i, η_i and ζ_i are the parametric coordinates of all nodes and $N_j (\xi_i, \eta_i, \zeta_i)$ are the corresponding shape functions. For a linear mapping, the shape functions of a 4-node 2-dimensional element have the form:

$$N_1^{(j)} = 0.25*(1- \xi_i)(1- \eta_i) \qquad (2a)$$

$$N_2^{(j)} = 0.25*(1+ \xi_i)(1- \eta_i) \qquad (2b)$$

$$N_3^{(j)} = 0.25*(1+ \xi_i)(1+ \eta_i) \quad (2c)$$

$$N_4^{(j)} = 0.25*(1- \xi_i)(1+ \eta_i) \quad (2d)$$

The shape functions for 8-node quadratic elements are given by:

$$N_1^{(j)} = -0.25*(1- \xi_i)(1- \eta_i)(\xi_i + \eta_i +1) \quad (3a)$$

$$N_2^{(j)} = 0.25*(1+ \xi_i)(1- \eta_i)(\xi_i - \eta_i -1) \quad (3b)$$

$$N_3^{(j)} = 0.25*(1+ \xi_i)(1+ \eta_i)(\xi_i + \eta_i -1) \quad (3c)$$

$$N_4^{(j)} = -0.25*(1- \xi_i)(1+ \eta_i)(\xi_i - \eta_i +1) \quad (3d)$$

$$N_5^{(j)} = 0.5*(1- \xi_i^2)(1- \eta_i) \quad (3e)$$

$$N_6^{(j)} = 0.5*(1+ \xi_i)(1- \eta_i^2) \quad (3f)$$

$$N_7^{(j)} = 0.5*(1- \xi_i^2)(1+ \eta_i) \quad (3g)$$

$$N_8^{(j)} = 0.5*(1- \xi_i)(1- \eta_i^2) \quad (3h)$$

With all perturbations calculated, the basic vectors that produce the desired shapes for the appliqué can be generated as follows,

$$XB_i = X_i^0 + P_{xi} \quad (4a)$$

$$YB_i = Y_i^0 + P_{yi} \quad (4b)$$

$$ZB_i = Z_i^0 + P_{zi} \quad (4c)$$

Where X_i^0, Y_i^0 and Z_i^0 are the initial coordinates of node *i*. Linear modifications can be performed with the perturbations occurring only on the corner points. This technique transfers the initial rectangular shape into a new quadrilateral if the initial rectangular shape is treated as one domain or into a polygon if two domains are used. By resorting to higher order mapping with mid-side node points also perturbed along with the corner points, the initial rectangular shape can be modified into a curved geometrical shape. Fig. 7 illustrates two examples with such perturbations implemented.

Alternative shapes are usually first specified in shape optimization and these shapes are represented by a set of displacement vectors known as basic vectors [Candan et al. (2000)], which depict the deviation from the component's initially assumed configuration to the alternative shapes. These basic vectors form the design space in optimization process. With several potentially optimum shapes generated in terms of the basic vectors for the components to

be designed, the real coordinates of all nodes can be assumed to be the combination of these possible shapes [Candan et al. (2000)],

$$X_i = X_i^0 + \sum_k DV_k(XB_{ik} - X_i^0) \quad (5a)$$

$$Y_i = Y_i^0 + \sum_k DV_k(YB_{ik} - Y_i^0) \quad (5b)$$

$$Z_i = Z_i^0 + \sum DV_k(ZB_{ik} - Z_i^0) \quad (5c)$$

where DV_k are the design variables. The goal of the design optimization is reduced to seeking the values of these design variables, which extremize the objective functions.

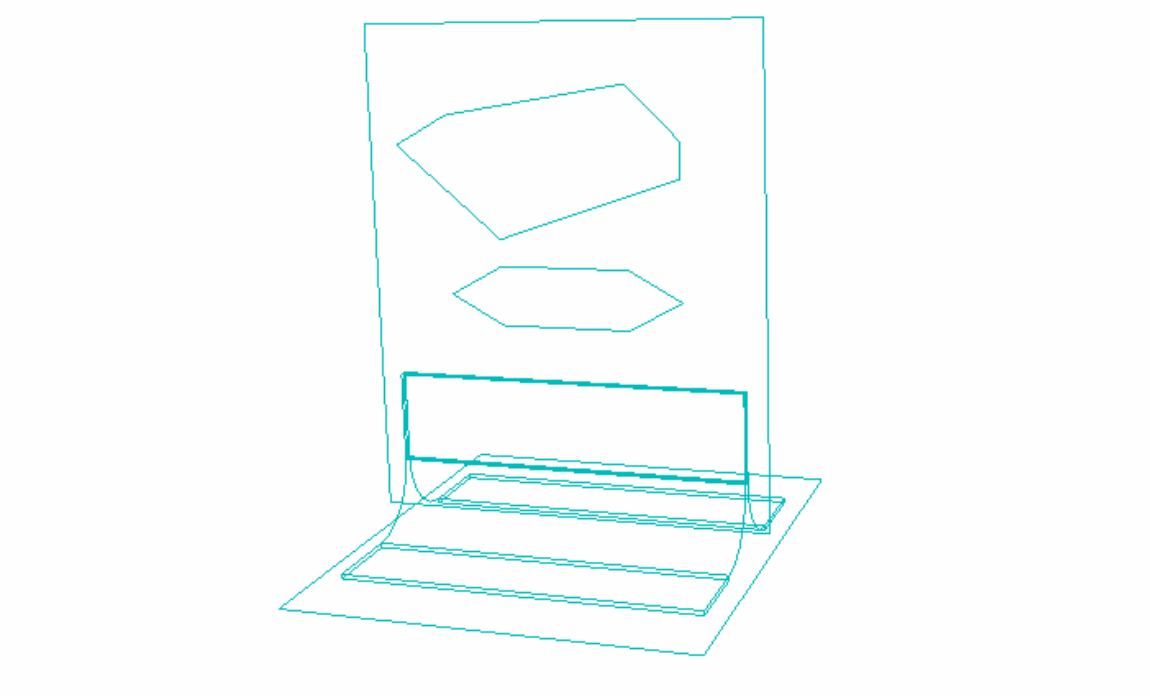
Next, an example is presented for the minimization of the thermal distortion of the T-joint structure. The joint is assumed to suffer a temperature change of 500 ^oF, which causes inhomogeneous thermal expansion and distortion if some manufacturing defects exist in the structure. One possible measure of thermal distortion is the difference between the displacements of the two top corner nodes of the vertical plate. Note that a perfectly fabricated joint structure would be perfectly symmetric and this measure of distortion would be zero. For this example, all the plates and supporting clips are still assumed to consist of 8-ply laminates with orientations $[0/30/-30/90]_s$. For a defect it is assumed that the two clips have a gross misalignment of 6 mm. The elastic and thermal properties are also assumed the same as in the previous sections. Without the addition of anti-distortion appliqués, the two out of plane displacements were computed as;

	U
Displacement at top left node	**-3.0510E-02**
Displacement at top right node	**2.3254E-02**

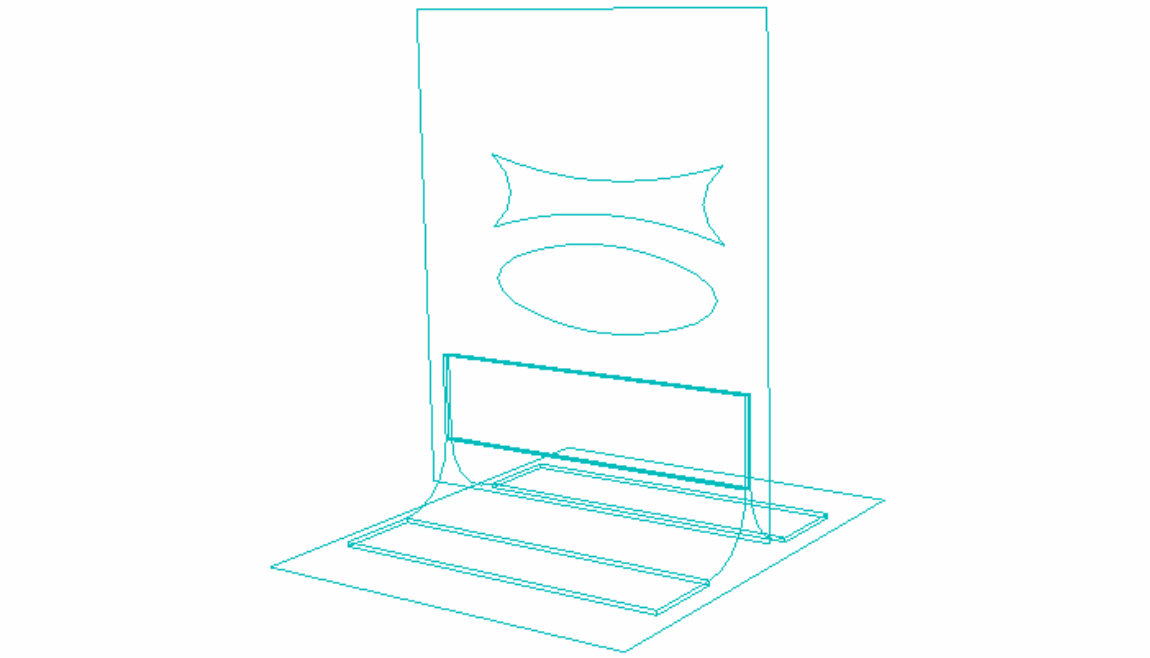
With two appliqués added onto the two opposite surfaces of the vertical plate as shown on Fig. 7, optimization can be performed with respect to both the position and shape of these appliqués. For instance, if the above displacements were used to define an objective function defined as the square of the difference of the two displacements, the optimization computation produces the following results.

	U
Displacement at top left node	**-8.8133E-03**
Displacement at top right node	**1.4520E-02**

The objective function history was as indicated by Fig. 8. The difference in displacement was reduced from approximately 0.054 to 0.023. The optimized appliqués are illustrated in Fig. 9. In this case, the improvement is modest but the problem serves to illustrate the fundamentals of the approach.



(a)



(b)

Figure 7. Examples of perturbed appliqué shapes

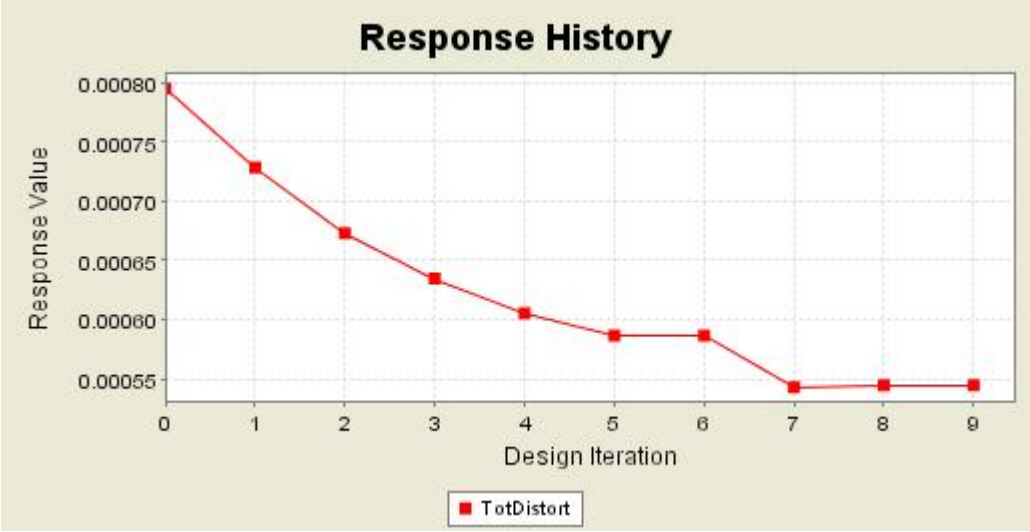


Figure 8. The minimization of the objective $(U_{(1)}-U_{(2)})^2$

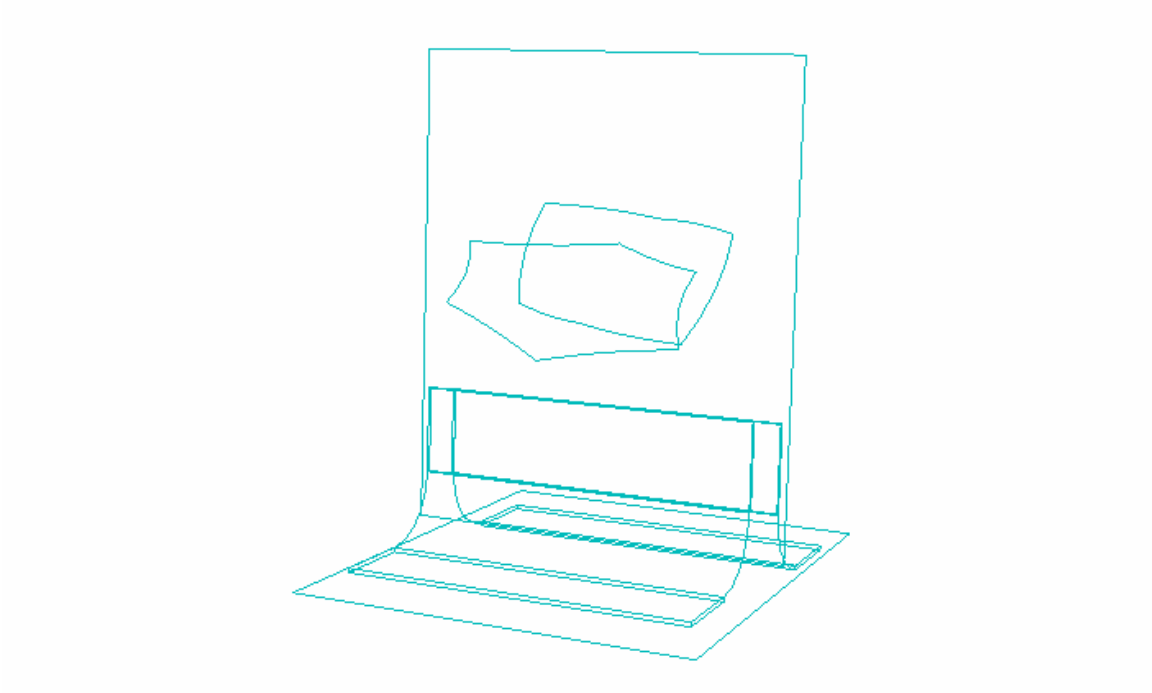


Figure 9. Optimized appliqués

2.5.2 Optimization with respect to appliqué position

For industrial application, there would be some limitations on the appliqués. For example, the total area of the appliqués, the mass of the appliqués, the relative positions of appliqués on the major components will all generally have limits. In the currently studied joint structure it makes sense that the size of the attached appliqués should be much smaller than that of the major components. Small appliqués have more flexibility in their possible positions. It is expected that

the same correction effect can be achieved through a few smaller appliqués as does by the two larger ones. In this case it would be expected that the position of the appliqués would have a dominant effect versus the shape. The example computation presented next shows that through finding their optimal position the attached smaller appliqués can similarly reduce the distortion deformation due to thermal effect.

The computation was carried out for a composite material with almost null CTEs studied by Ishikawa et al. (1989). This material is a 20-ply laminate with the fiber angle of $\pm$ 46.4^o and their mechanical and thermal ply properties are shown in Table 3.

Table 3. The mechanical and thermal ply properties for a near zero CTE laminate

E_1(GPa)	E_2(GPa)	ν_{12}	ν_{23}	G_{12}(GPa)	G_{23}(GPa)	CTE A_1(ppm)	CTE A_2(ppm)
135	11.0	0.30	0.59	5.5	3.17	-0.26	3.2

Fig. 10 shows the magnified distortion state of the joint due to a little misalignment of the two clips with the structure assumed to sustain a temperature change of 500 ºK. To reduce the distortion of the vertical web a few small appliqués were added on the two sides of the web. As a simple illustration, two rectangular appliqués were attached to each side of the plate respectively with their positions to be determined through the optimization. The appliqués were initially randomly placed on the two sides of the vertical web. Fig. 11 illustrates the initial deformed state of the structure after the small appliqués were added but before the optimization procedure was started.

The optimization goal was to minimize the summation of the absolute displacement values of all the 19 topmost nodes (see Fig. 10) of the vertical web. With the optimization procedure implemented with respect to the positions of the four small appliqués, the four pairs of their coordinates on the vertical plate were defined as the design variables. The computation was carried out to find the appliqués' best positions through changing these variables continually so that the objective function was minimized. The results obtained from the optimization procedure are listed in Table 4.

It can be observed that the displacements of most nodes have been significantly decreased and their summation has been reduced by nearly an order of magnitude. Fig. 12 illustrates the deformations of the joint after the four small rectangular appliqués were attached to the two sides of the vertical plate at their optimal positions.

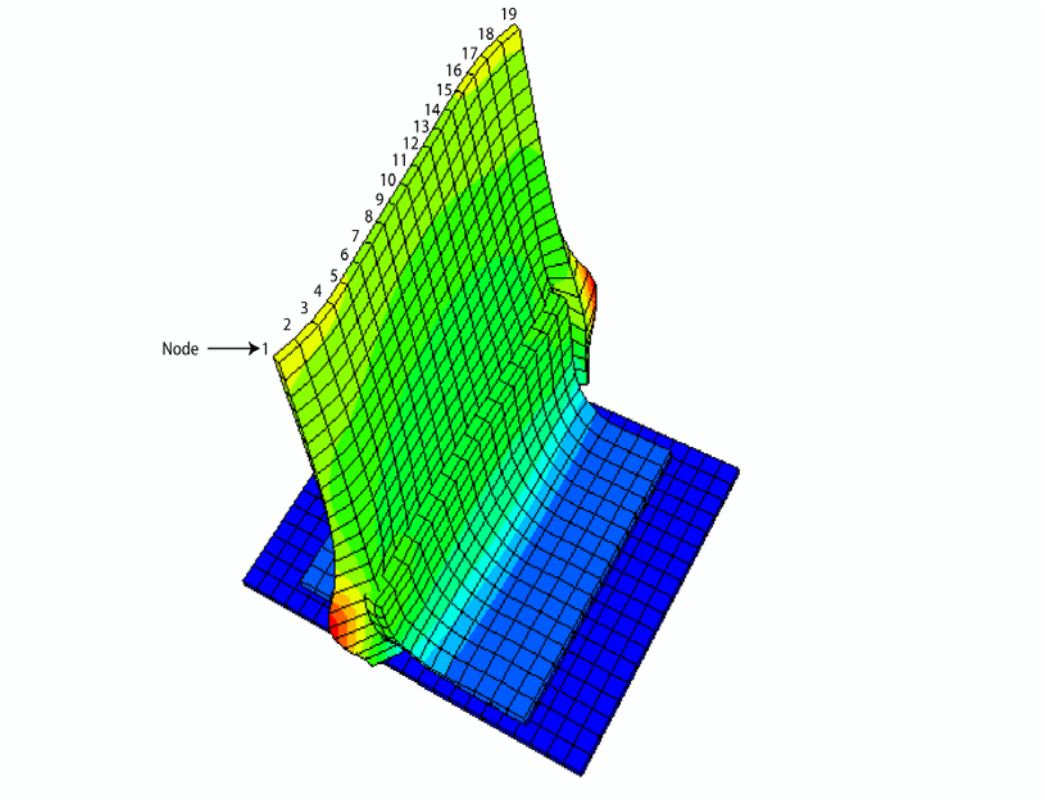


Figure 10. The distorted joint due to clip misalignment

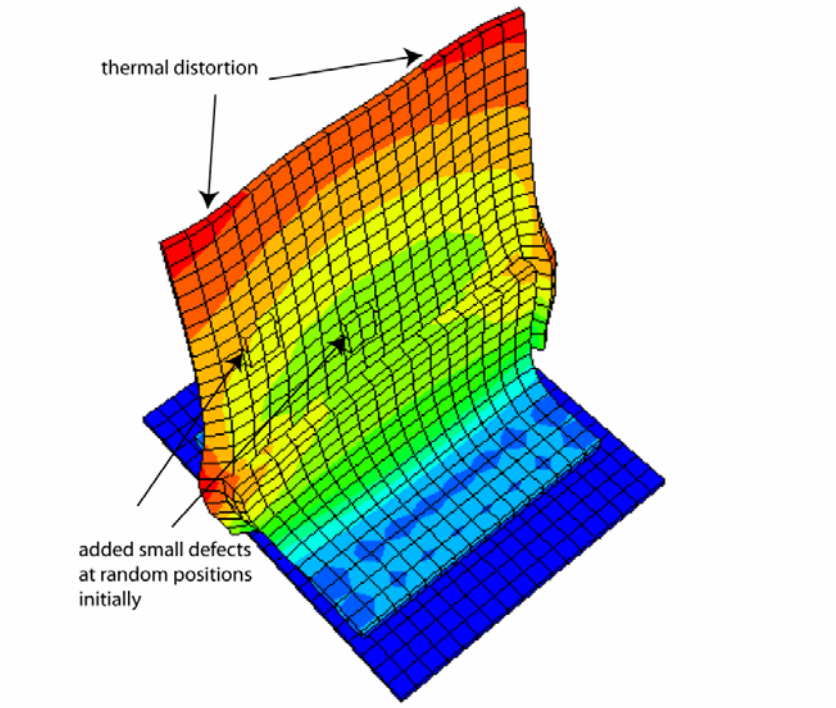


Figure 11. The joint's initial state with appliqués added but before optimization

Table 4. Optimized deformation in terms of top nodes displacement (unit: 10^{-2} mm)

Node	1	2	3	4	5	6	7	8	9	10
No appliqués	-2.9	1.8	4.9	5.8	5.3	4.3	3.1	2.0	0.96	-0.01
With appliqués	-0.39	0.8	1.1	0.67	0.27	0.10	0.06	0.08	0.13	0.21
Node	11	12	13	14	15	16	17	18	19	Sum
No appliqués	-0.98	-2.0	-3.1	-4.3	-5.3	-5.8	-4.9	-1.9	2.9	**62.3**
With appliqués	0.30	0.39	0.38	0.08	0.18	0.18	-0.94	-1.6	-1.3	**9.12**

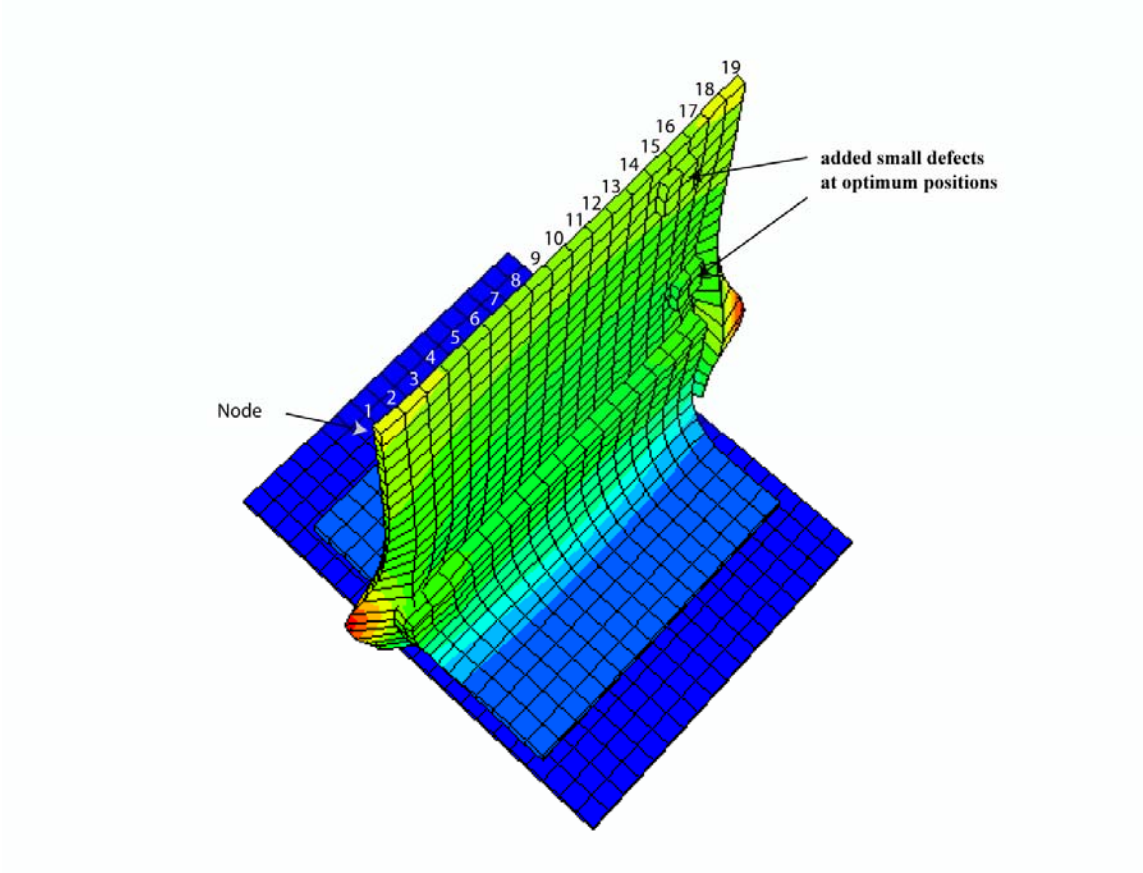


Figure 12. The optimized structure after adding four small appliqués

2.6 Hardware Design and Optimization

In cooperation with ATK Corp. a specific T-joint geometry was developed for experimental demonstration of the anti-distortion appliqué methodology. To obtain measurable distortions the basic T-joint was modified to include a metal foil covering a portion of one side of the vertical web. Photogrammetry was used to measure deflections due to temperature change. The system

used has an accuracy limit of about 0.025 mm. A finite element model of the joint indicated the metal foil would cause thermal distortion deflections about one order of magnitude larger so they would be measurable within reasonable error. Several views of the finite element model are shown in Fig. 13-16. Shown in these figures are nominal dimensions of the actual assembly including the metal foil. All the composite parts; base plate, vertical web, and curved clips were made up of a 48 ply laminate with a $([-45/0/45/90]_S)6$ laminate sequence. Initially the metal foil was assumed to be aluminum. The thermo-elastic properties used in the analyses for the adhesive, the laminae, and the aluminum foil are listed in Table 5.

As a result of the previously described analyses it was concluded that perhaps the most efficient approach for implementing the anti-distortion appliqué methodology was to use simple rectangular appliqués and limit the design variables to coordinate locations of the appliqués. Hence, this basic strategy was used for the remainder of the study. Of course, depending on the application different materials might be used for the appliqués. For example, coupons of unidirectional composite would introduce anisotropy of the appliqué thermo-mechanical response and possibly be more effective in certain situations. Then, the appliqué orientation could also be an effective design variable in the optimization.

Prior to any testing a purely analytical optimization was performed that mimicked the process eventually used on the actual part. The photogrammetry was capable of tracking a finite number of predefined target points. Nine target points were established on the foil side of the vertical web as illustrated in Fig. 17. Deflections at these target points then serve as a basis for defining an objective function for the analysis. Physically, it was decided that the objective would be to keep the vertical web flat with no regard for rigid body motion. Mathematically this meant using three of the target points to define a plane and then minimizing the relative out of plane displacements of the remaining target points.

2.6.1 Objective function

In what follows, the objective function was defined in terms of the displacements at the 9 target points utilized by the photogrammetry. Three of those points define a plane in the deformed coordinates. The points chosen to define a plane were B, D, and C as labeled in Fig. 17.

Two position vectors were defined on the plane using points B, D, and C as follows:

$$\vec{V}_1 = \underbrace{(x_B - x_D)}_{a_1}\vec{i} + \underbrace{(y_B - y_D)}_{b_1}\vec{j} + \underbrace{(z_B - z_D)}_{c_1}\vec{k}$$

$$\vec{V}_2 = \underbrace{(x_C - x_D)}_{a_2}\vec{i} + \underbrace{(y_C - y_D)}_{b_2}\vec{j} + \underbrace{(z_C - z_D)}_{c_2}\vec{k}$$

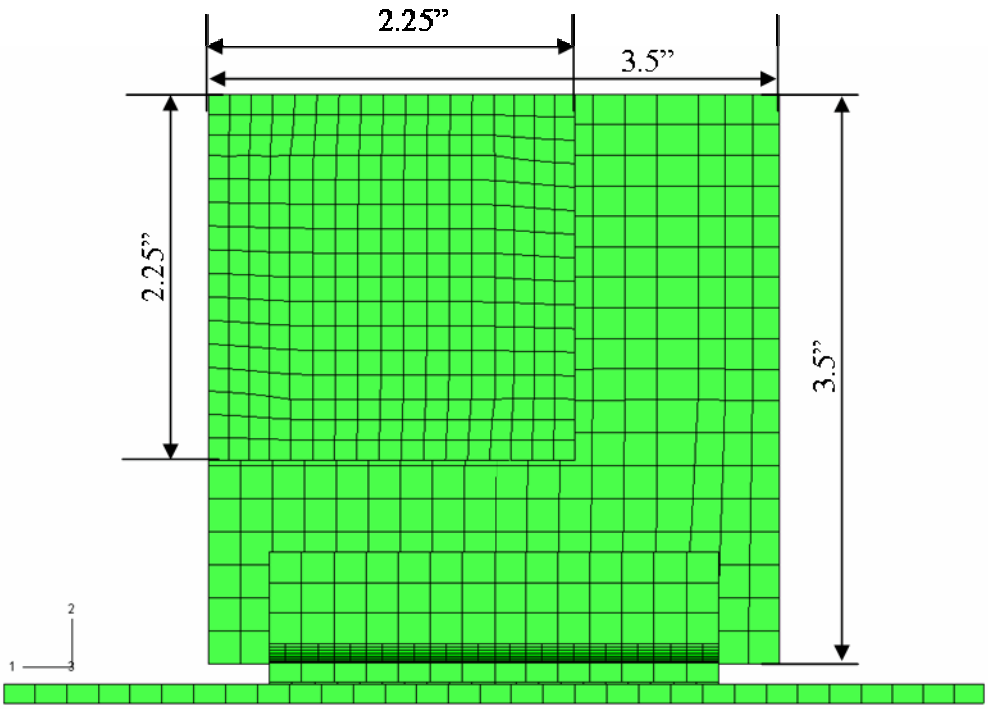


Figure 13. Front view of the T-joint depicting the attached layer.

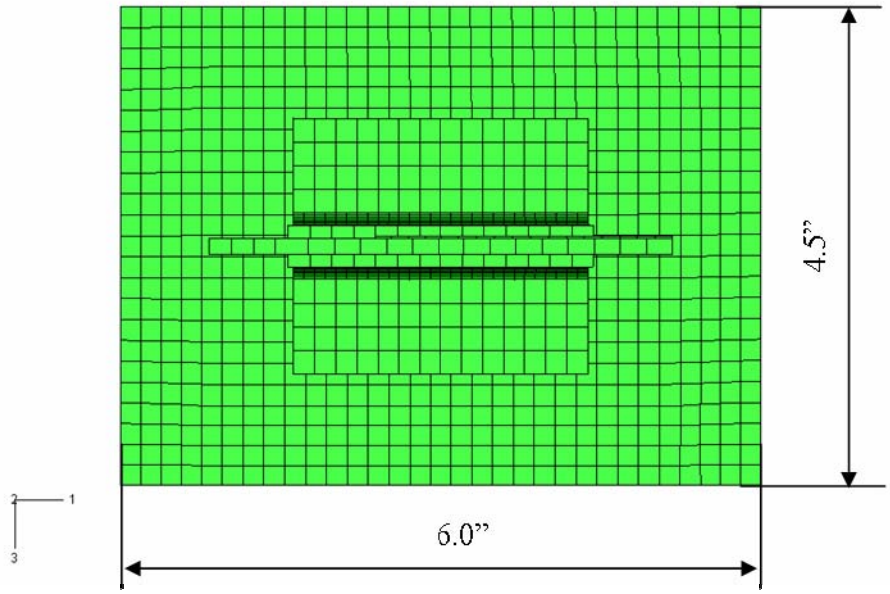


Figure 14. Top view of the T-joint model

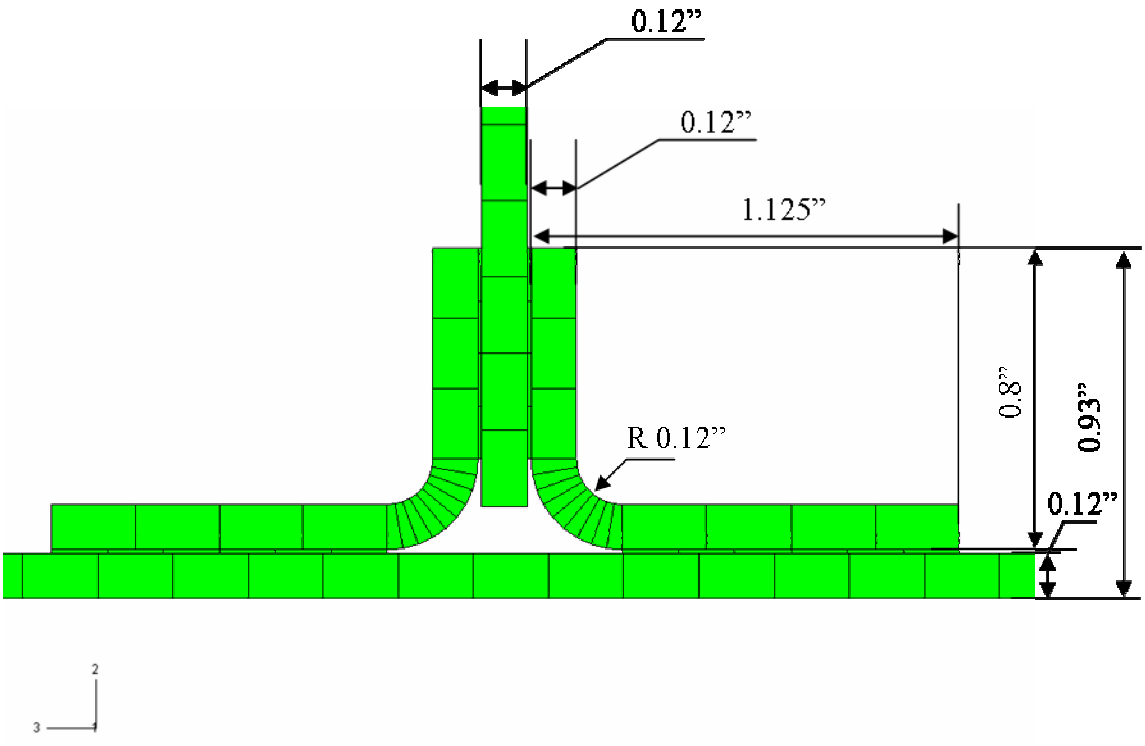


Figure 15. Front view of the T-joint model depicting the clips

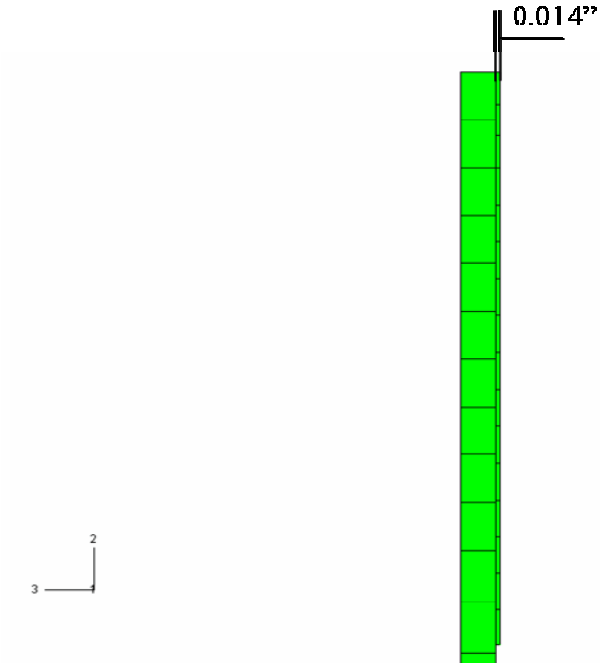


Figure 16. The T-joint model with a thin layer attached to the vertical plate

Table 5. Thermo-elastic properties of the constituents

Property	Lamina	Al	Adhesive	Ti
E_{11} (Msi)	30	10.5	0.3	16.8
E_{22} , E_{33} (Msi)	1.5	10.5	0.3	16.8
υ_{12}, υ_{13}	0.3	0.33	0.41	0.36
G_{12}, G_{13} (Msi)	0.4	3.946	0.1064	6.185
υ_{23}	0.6	0.33	0.41	0.36
α_{11} (μ $^{o}F^{-1}$)	-0.2	13	42	4.72
α_{22}, α_{33} (μ $^{o}F^{-1}$)	15	13	42	4.72

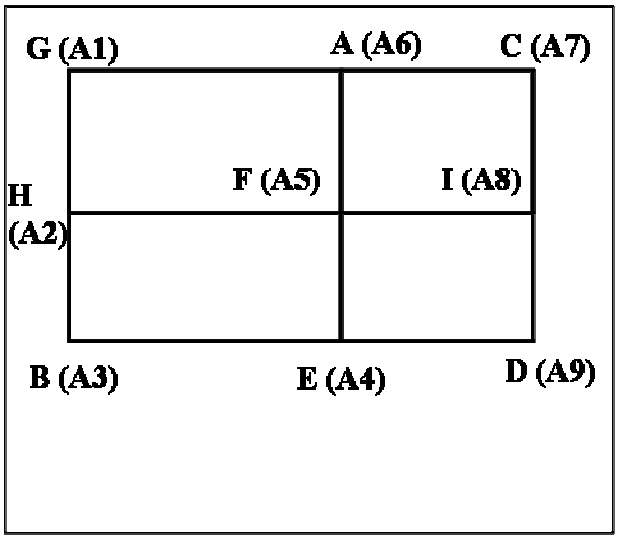


Figure 17. Side view of the T-joint with target points

The unit normal that defines this plane was obtained from the normalized cross-product of the vectors $\vec{V}_1$ and $\vec{V}_2$ and is given by,

$$\vec{u}_n = \frac{\vec{V}_1 \times \vec{V}_2}{|V_1 \times V_2|} = \frac{(d_1\vec{i} + d_2\vec{j} + d_3\vec{k})}{\sqrt{d_1^2 + d_2^2 + d_3^2}}$$

where

$$d_1 = b_1c_2 - b_2c_1$$
$$d_2 = -(a_1c_2 - a_2c_1)\,.$$
$$d_3 = (a_1b_2 - a_2b_1)$$

The position vectors from point D to each of the remaining points A, G, H, F, and E in the model are given in the following equations. Note that the target point "I" was not considered in the objective function.

$$\vec{V}_3 = \underbrace{(x_A - x_D)}_{a_3}\vec{i} + \underbrace{(y_A - y_D)}_{b_3}\vec{j} + \underbrace{(z_A - z_D)}_{c_3}\vec{k}$$

$$\vec{V}_4 = \underbrace{(x_G - x_D)}_{a_4}\vec{i} + \underbrace{(y_G - y_D)}_{b_4}\vec{j} + \underbrace{(z_G - z_D)}_{c_4}\vec{k}$$

$$\vec{V}_5 = \underbrace{(x_H - x_D)}_{a_5}\vec{i} + \underbrace{(y_H - y_D)}_{b_5}\vec{j} + \underbrace{(z_H - z_D)}_{c_5}\vec{k}$$

$$\vec{V}_6 = \underbrace{(x_F - x_D)}_{a_6}\vec{i} + \underbrace{(y_F - y_D)}_{b_6}\vec{j} + \underbrace{(z_F - z_D)}_{c_6}\vec{k}$$

$$\vec{V}_7 = \underbrace{(x_E - x_D)}_{a_7}\vec{i} + \underbrace{(y_E - y_D)}_{b_7}\vec{j} + \underbrace{(z_E - z_D)}_{c_7}\vec{k}$$

The out of plane deflections of points A, G, H, F, and E were then obtained from the scalar product of the above position vectors and the unit normal vector. The objective function to be minimized was defined as *the sum of the squares of the scalar products between the plane normal* $\vec{u}_n$ *and the vectors* $\vec{V}_3$ *through* $\vec{V}_7$. This results in:

$$Obj = ((d_1a_3 + d_2b_3 + d_3c_3)^2 + (d_1a_4 + d_2b_4 + d_3c_4)^2 + (d_1a_5 + d_2b_5 + d_3c_5)^2 + (d_1a_6 + d_2b_6 + d_3c_6)^2 + (d_1a_7 + d_2b_7 + d_3c_7)^2)/(d_1^2 + d_3^2 + d_3^2)$$

2.6.2 Numerical Implementation

The original coordinates of the target points A through H in the global system are given in the second column of Table 6. The notation for the displacement components and the updated coordinates are given in the third and fourth columns. The deformed coordinates are then used to calculate the position vectors defined above which were used to compute the updated scalar value of the objective function. In an actual application of the method the displacements would come from experiments that simulate the structure's application environment. These actual displacements would be added to the FEA model predicted displacements so that the minimization process would yield predicted displacements that cancel with the measured ones yielding a net minimized distortion. Here, in the absence of experimental data, the appliqués are to introduce distortions that cancel with the simulated displacements due to the aluminum foil.

2.6.2.1 Optimization with 3 appliqués

To introduce displacements that minimize the above-described objective function, three aluminum appliqués each of size 0.5"x 0.5"x0.04" were modeled as attached to the vertical web on the side opposite the aluminum foil as depicted in Fig. 18. Each of the appliqués was

simulated as attached to the vertical web with a 0.01" thick adhesive layer included. The objective function was minimized by altering the position of these three appliqués. The appliqués positions were altered and the analysis was repeated as driven by the optimization software VisualDOC. Fig. 19 shows the numerical value of the objective function as the position of the appliqués was modified by the gradient based optimization routine. Excessive iterations were forced to help assure convergence to an optimum. Ultimately the objective function was reduced by approximately 90% by finding the optimal positions of the three appliqués. The optimized positions are given in Table 7 where the x and y coordinate indicate the position of the center of appliqués on the vertical plate. Note that the origin for this coordinate system is the same as that of the model and located at the mid-point of the bottom of the base plate with the x-y-z. The x-y-z directions correspond to the 1-2-3 coordinate system shown in Fig.18.

Table 6. Original and deformed coordinates of target points used in the objective function

Label	Coordinates	Displacements	Deformed coordinates
Xa	0	Uxa	Uxa
Ya	3.37	Uya	3.37 + Uya
Za	-0.074	Uza	-0.074 + Uza
Xb	1.25	Uxb	1.25 + Uxb
Yb	1.87	Uyb	1.87 + Uyb
Zb	-0.074	Uzb	-0.074 + Uzb
Xc	-1.25	Uxc	-1.25 + Uxc
Yc	3.37	Uyc	3.37 + Uyc
Zc	-0.074	Uzc	-0.074 + Uzc
Xd	-1.25	Uxd	-1.25 + Uxd
Yd	1.87	Uyd	1.87 + Uyd
Zd	-0.074	Uzd	-0.074 + Uzd
Xe	0	Uxe	Uxe
Ye	1.87	Uye	1.87 + Uye
Ze	-0.074	Uze	-0.074 + Uze
Xf	0	Uxf	Uxf
Yf	2.62	Uyf	2.62 + Uyf
Zf	-0.074	Uzf	-0.074 + Uzf
Xg	1.25	Uxg	1.25 + Uxg
Yg	3.37	Uyg	3.37 + Uyg
Zg	-0.074	Uzg	-0.074 + Uzg
Xh	1.25	Uxh	1.25 + Uxh
Yh	2.62	Uyh	2.62 + Uyh
Zh	-0.074	Uzh	-0.074 + Uzh
Xi	-1.25	Uxi	-1.25 + Uxi
Yi	2.62	Uyi	2.62 + Uyi
Zi	-0.074	Uzi	-0.074 + Uzi

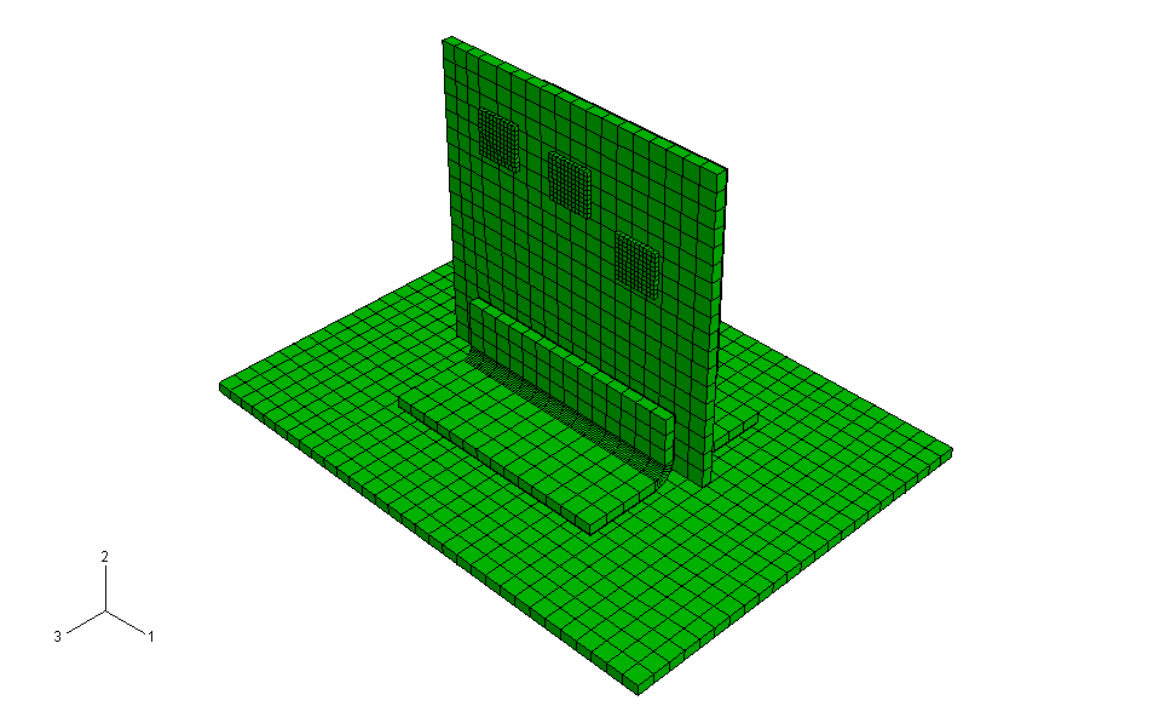


Figure 18. FE model showing aluminum appliqués located to minimize the objective function

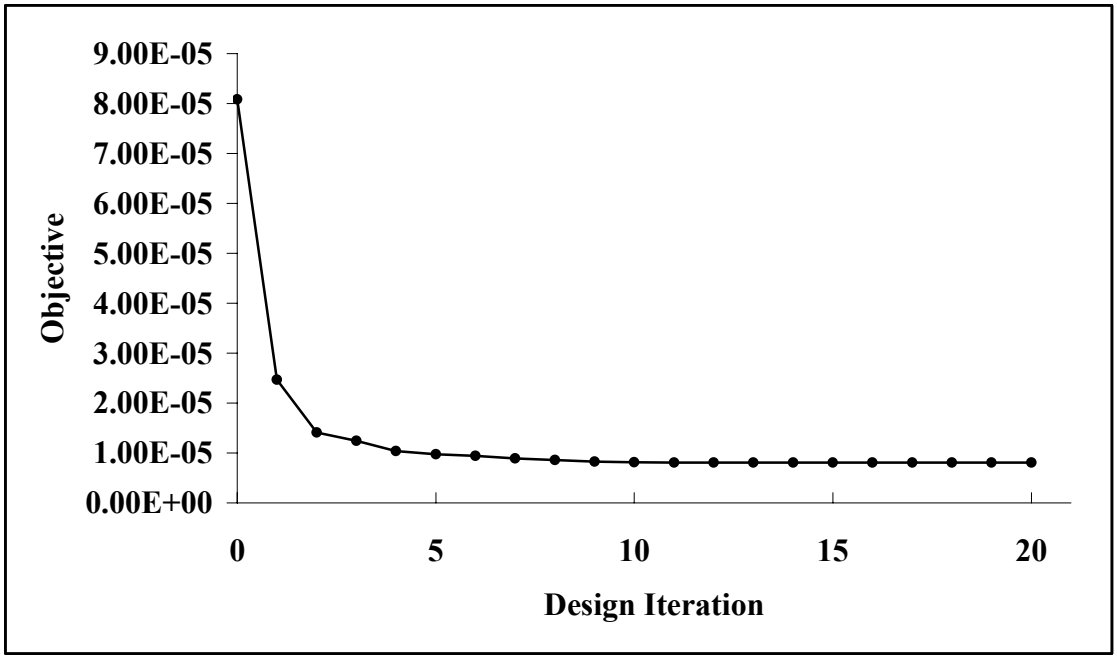


Figure 19. Objective function as a function of design iterations for three appliqués

Table 7. Optimized positions of the three appliqués

Defect	x	y
1	-0.003081051	2.891623
2	0.858794	2.404868
3	-0.9660705	2.979695

2.6.2.2 Optimization with 4 appliqués

For actual fabrication, titanium foil was selected instead of the aluminum modeled as described above. The properties of the titanium are shown in Table 5. Another "analysis only" optimization was conducted with titanium in the model and with four appliqués used instead of three. The thickness of the appliqués was reduced to 0.025 inch. The positions of the four appliqués were optimized to reduce the objective function defined earlier. The optimized positions of the four appliqués are listed in Table 8. Fig. 20 shows the history of the magnitude of the objective function. The optimization reduced the objective function by 87.5%.

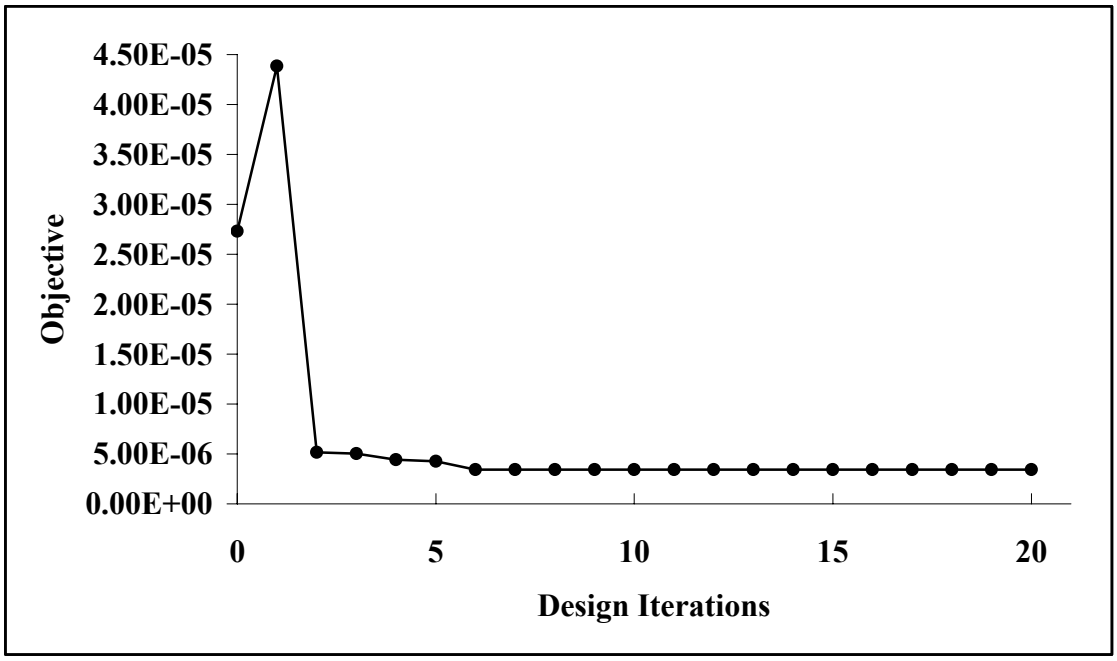


Figure 20. Objective function as a function of design iterations for a model with four appliqués

Table 8. Optimized position of four appliqués

Defect	x	y
1	-0.1145515	2.364928
2	0.729481	2.133869
3	-0.5826635	1.98095
4	-0.4651298	1.18

2.7 Validation with Experiments

For experimental correlation the objective function was slightly modified and the target points at which the displacements were measured were altered. The laminate properties were also altered and were held proprietary by an industry partner. The actual composite laminate was similar to a quasi-isotropic graphite epoxy system modeled in the previous section. The target points utilized in this new objective are shown in Fig. 17. Point E was not considered in the analysis. Also, as previously described, the experimental displacements in the three coordinate directions at the target points were added to each of the displacements in the objective function so that a minimized objective function results in appliqué driven displacements that cancel with those measured in the laboratory; thus minimizing the overall distortion of the structure. The new objective function is defined as follows where the terms a_i, b_i, and c_i are defined in Table 9.

$$Obj = ((d_1a_3 + d_2b_3 + d_3c_3)^2 + (d_1a_4 + d_2b_4 + d_3c_4)^2 + (d_1a_5 + d_2b_5 + d_3c_5)^2 + (d_1a_6 + d_2b_6 + d_3c_6)^2 + (d_1a_8 + d_2b_8 + d_3c_8)^2)/(d_1^2 + d_3^2 + d_3^2)$$

Table 9. Variables and initial coordinates input to the objective function

Label	Deformed Coordinates	Displacements
a1	Xb - Xd	2.494993 + Uxb - Uxd
b1	Yb - Yd	0.005813 + Uyb - Uyd
c1	Zb - Zd	0.000209 + Uzb - Uzd
a2	Xc - Xd	0.001652 + Uxc - Uxd
b2	Yc - Yd	1.506772 + Uyc -Uyd
c2	Zc - Zd	-0.01261 + Uzc - Uzd
a3	Xa - Xd	1.24383 + Uxa -Uxd
b3	Ya - Yd	1.50113 + Uya - Uyd
c3	Za - Zd	-0.007942 + Uza - Uzd
a4	Xg - Xd	2.494082 + Uxg - Uxd
b4	Yg - Yd	1.50188 + Uyg - Uyd
c4	Zg - Zd	-0.004613 + Uzg - Uzd
a5	Xh - Xd	2.498276 + Uxh - Uxd
b5	Yh - Yd	0.759291 + Uyh - Uyd
c5	Zh - Zd	-0.008923 + Uzh - Uzd
a6	Xf - Xd	1.244347 + Uxf - Uxd
b6	Yf - Yd	0.756278 + Uyf - Uyd
c6	Zf - Zd	-0.008263 + Uzf - Uzd
a7	Xe - Xd	1.245789 + Uxe - Uxd
b7	Ye - Yd	0.009036 + Uye - Uyd
c7	Ze - Zd	-0.006377 + Uze - Uzd
a8	Xi - Xd	0.002422 + Uxi - Uxd
b8	Yi - Yd	0.753504 + Uyi - Uyd
c8	Zi - Zd	-0.003331 + Uzi - Uzd

2.7.1 Experimental Results

Alliant Techsystems (ATK) Corporation conducted the photogrammetry experiments. The T-joint structure prepared for photogrammetry is shown in Fig. 21. The titanium foil can be seen on the vertical web along with the 9 target points where the displacements are measured. The various dot patterns on the base plate are utilized by the photogrammetry software to solve for the camera position. The T-joint was subjected to a temperature change from 80 C to -150 C.

The measured displacement data is summarized in Fig. 22-24. Only the displacements perpendicular to the web were of practical interest since they will dominate the characterization of the nonplanar distortion of the vertical web. In each figure, the raw displacements for each row of 3 points are shown that include displacement due to rigid body motion of the T-joint structure. Subsequently the rigid body part of the total displacements was removed and the resulting net displacements are seen to be in reasonable agreement with finite element calculations performed both at ATK and at the University of Wyoming.

The top row, where the deflections are the largest, are in particularly good agreement with the results from the FE analyses. The bottom row has what could be described as an anomaly. The experimental data suggests that the web has a curvature opposite that of the other two rows and opposite that of the FEA predictions. The source of the anomaly is unknown. However, it is known that the precision of the experimental apparatus is marginal (on the order of 0.001 inch) compared to the small deflections occurring along the bottom row of target points (on the order of 0.002 inch). With that observation and the relatively good agreement with the other two target rows, the experiment was believed to be a success.

2.7.2 Appliqué Optimization

Two aluminum appliqués, half inch square and of 0.025 inch thickness were modeled as attached to the vertical web using an adhesive of 0.010" thickness. Larger numbers of appliqués were found to be ineffective in further reducing the objective function. To be effective, it was known that the appliqués would have to be on the side opposite the titanium foil. The optimization procedure was performed using VisualDOC combined with the ABAQUS FE model of the joint to minimize the objective function that relates to non-planar distortion of the web. The optimized positions of the appliqués result in an approximately 61% decrease in the objective function according to the FE model predictions. The resulting position of the appliqués on the FE model is as shown in Fig. 25. Fig. 26 shows the T-joint with the two appliqués bonded in place in preparation for testing.

The 61% decrease was much less than the decrease predicted previously with "analysis only" optimization. This may have been in part due to the limitations of the photogrammetry system. The numerical optimization procedure attempts to eliminate the measured deflections. If components of those deflections are not real (experimental error) then they will generally be counter to the physics of the problem. Since the models presumably do a good job of simulating the physics they will have extreme difficulty offsetting nonphysical deformations with the appliqués.

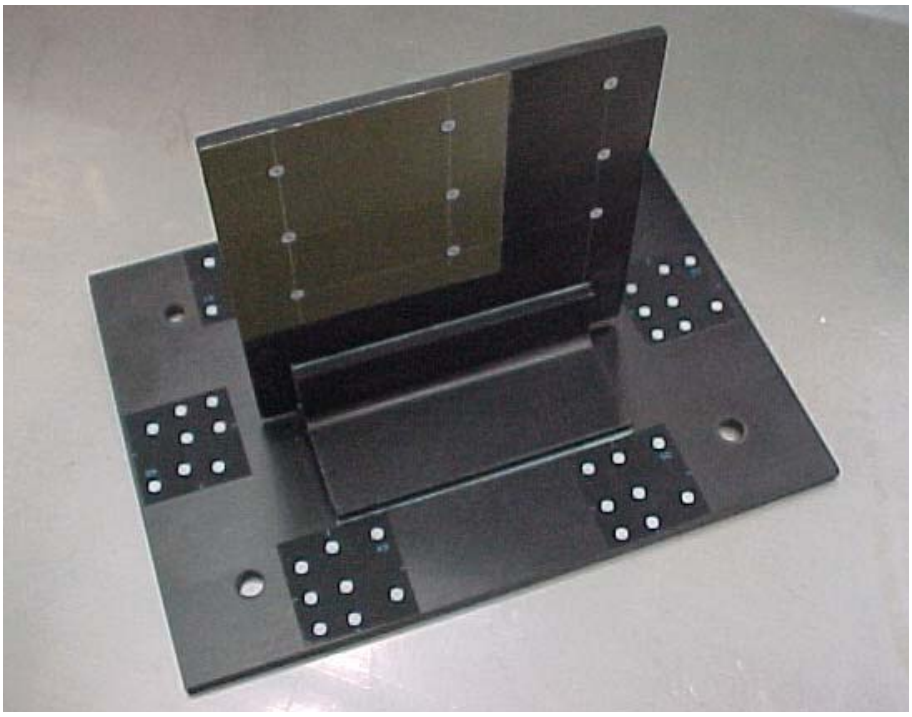


Figure 21. Composite T-joint prepared for thermal distortion testing

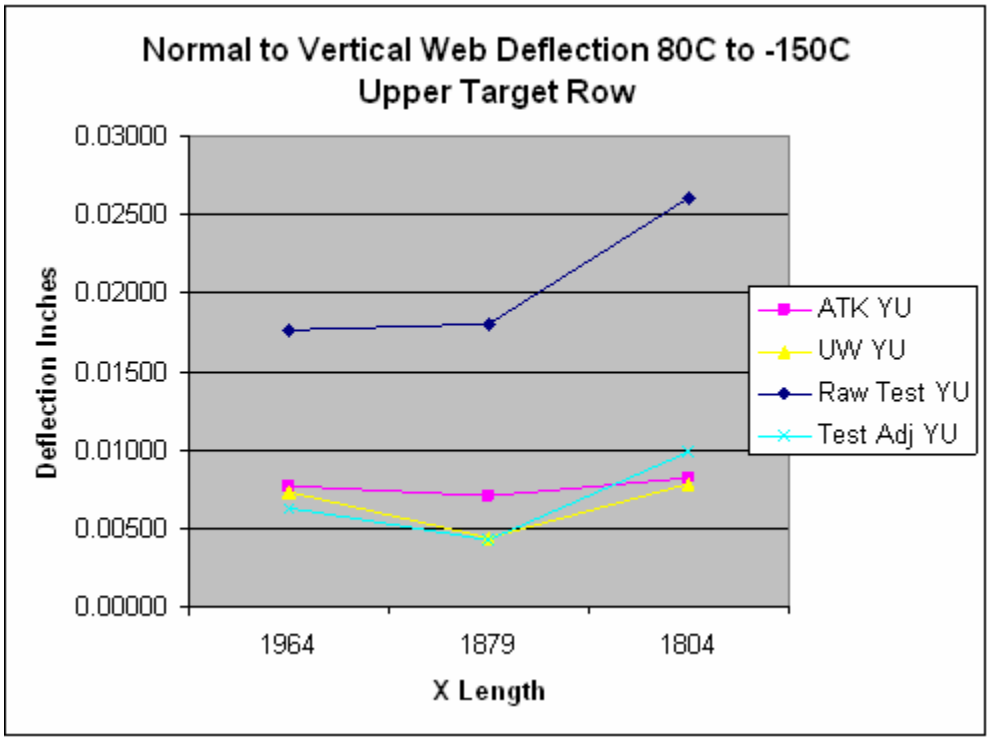


Figure 22. Top row target point normal deflections compared to FE predictions

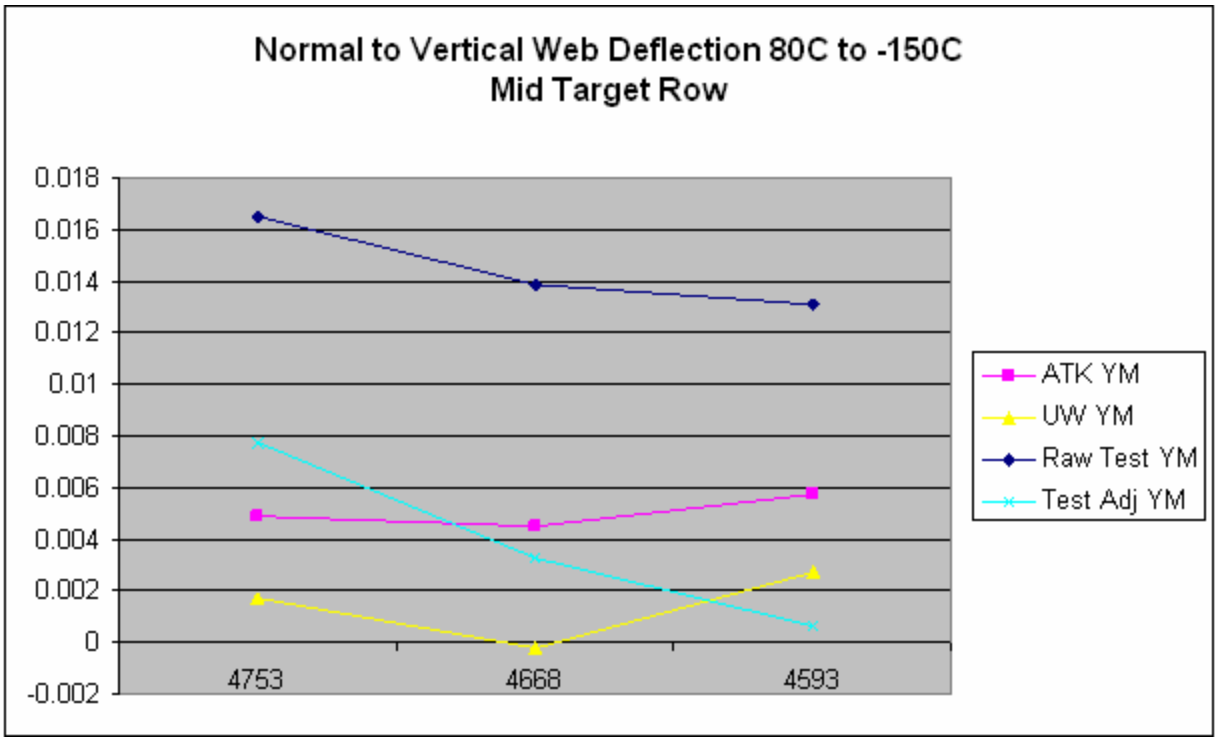


Figure 23. Middle row target point normal deflections compared to FE predictions

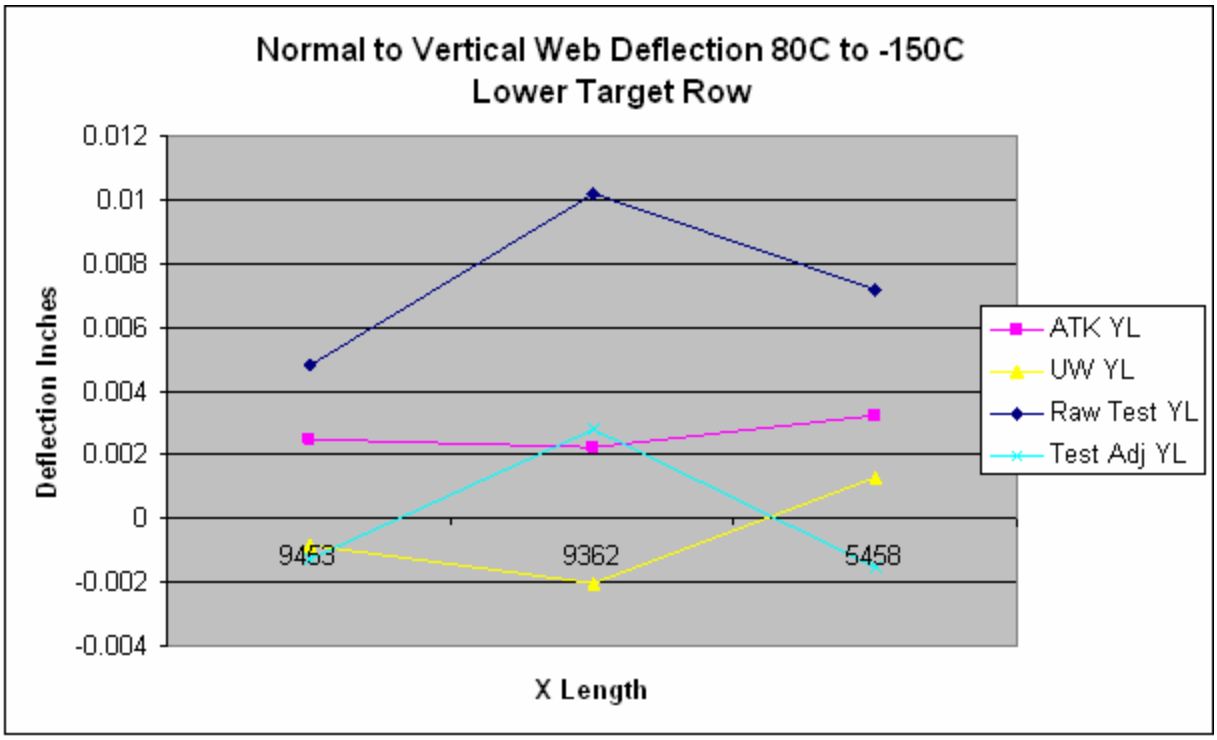


Figure 24. Bottom row target point normal deflections compared to FE predictions

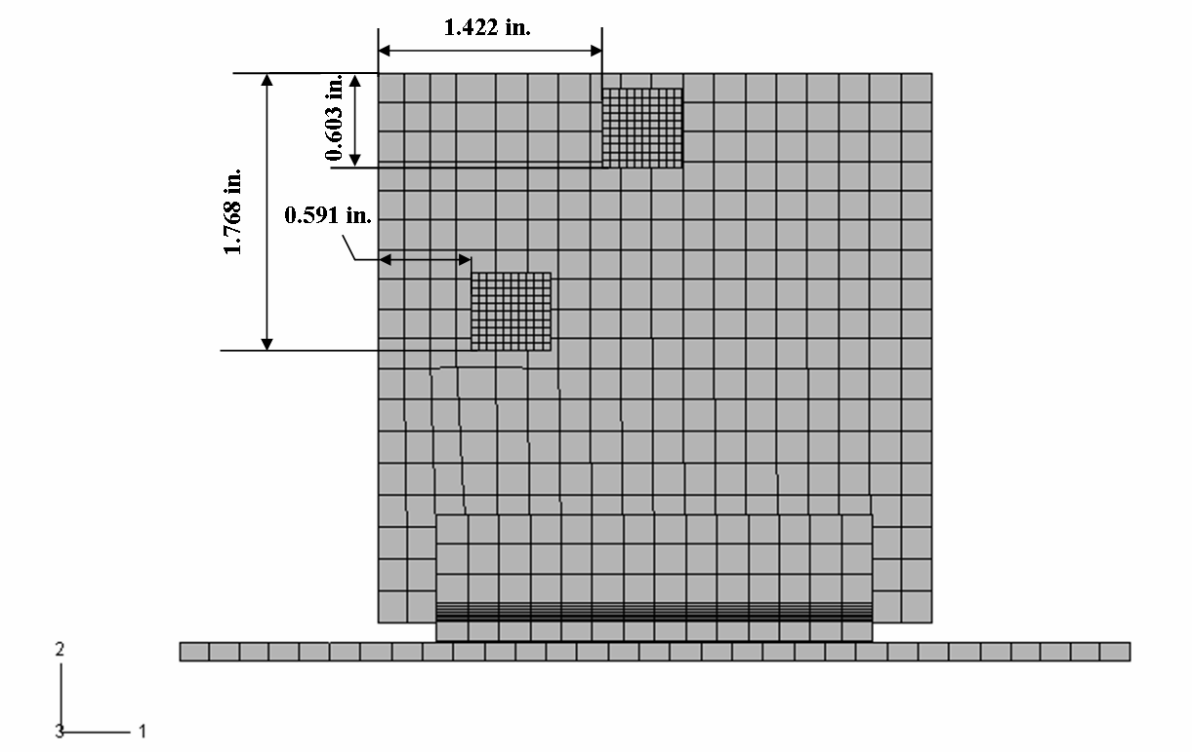


Figure 25. Optimized position of two appliqués

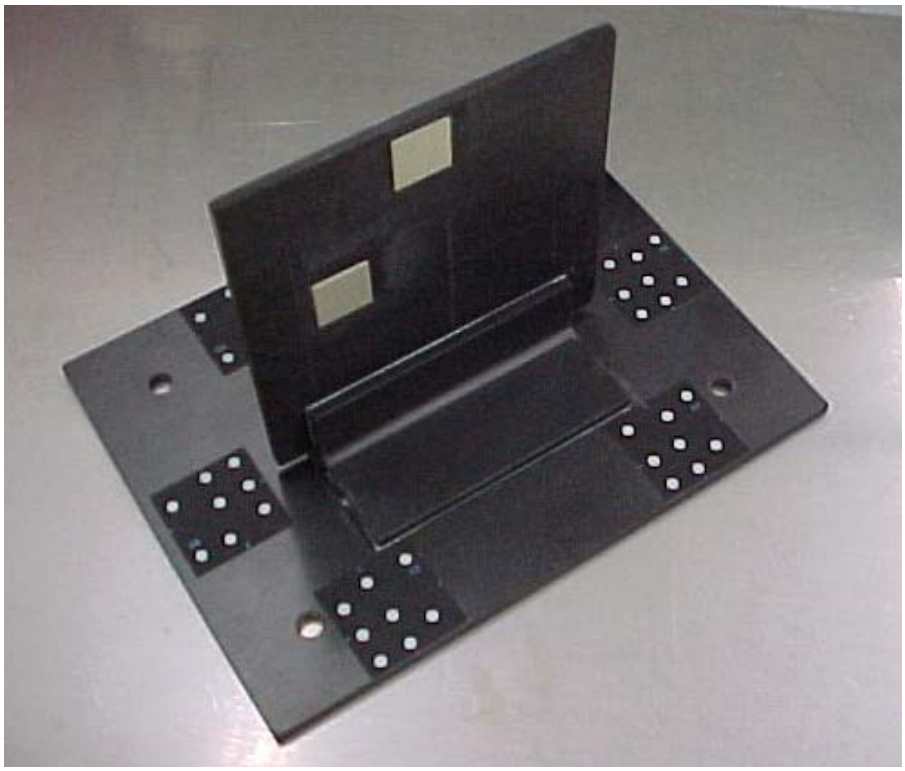


Figure 26. Adhesively bonded appliqués

2.8 Summary

In this chapter an approach for improving dimensional stability of composite structures was discussed. The basic procedure is quite different than the contemporary strategy involving ever increasing refinement of manufacturing technology. Here it was proposed that the as built structure can be modified to remove dimensional instabilities stemming from inevitable manufacturing defects. The approach relies on a numerical (FE) representation of the thermo-mechanical properties of the structure. The model can be idealized because it is unnecessary to simulate the sources of thermal distortion. In fact, the sources can be unknown. The model is then used to design modifications through material additions referred to as "anti-distortion appliqués" that cause distortions opposite to those measured in the laboratory. To automate the process, optimization software was used in conjunction with the FE software so that optimal configurations could be determined without trial and error efforts by an analyst.

It has been shown through several modeling examples that the adverse thermal distortions due to unwanted fabrication defects can be reduced through the addition of designed geometrical appliqués. A trial-and-error approach can also be used for simple structures. Optimization through numerical modeling was shown to provide a systematic approach to reducing the unwanted thermal distortion and thus to improve the dimensional stability of space structures. This concept of 'anti-distortion appliqués' has the potential of improving space structures' dimensional stability in a cost effective way.

This approach requires only the ability to model the structure's idealized thermo-mechanical properties, knowledge of the environmental stimulus (i.e. temperature change) that drives the instability, and a laboratory measurement of the instability to be eliminated. However, only measurable instabilities can be eliminated through this approach. This is in fact a major drawback of the approach since displacement measurement technology is not sufficient to meet the needs of the most demanding applications.

3.0 EFFECTS OF MATRIX CRACKING ON LAMINA BEHAVIOR

The complexity in analyzing laminated composite structures arises from characteristic features such as anisotropy of material properties, failure modes, heterogeneity, etc. Generally, the fiber and the matrix constituents have different properties and exhibit complex thermo-mechanical interactions in a lamina. Furthermore, a composite laminate is made up of unidirectional fiber reinforced composite laminae with possibly different properties, thicknesses, and fiber orientations. The heterogeneity in a laminate and a lamina coupled with complex damage modes which may induce further anisotropy as well as nonlinearity, causes the modeling of post-damage material behavior to be a significant challenge.

3.1 Degradation Model Review

Here, constitutive models or degradation models that describe the damage state of composite laminae were studied with detailed emphasis on understanding damage due to matrix cracking in unidirectional laminates. To begin, a synopsis of a comprehensive literature review of various degradation models implemented in progressive failure analysis (see Appendix D, [D.1-D.310]) utilizing the Finite Element Method (FEM) and Classical Laminate Theory (CLT) is presented. The review is at this time submitted for publication [Akula and Garnich (2007a,b)]. The models were classified as "sudden degradation models" and "gradual degradation models". In a sudden degradation model, the reduction in properties due to damage is not a function of other evolving field variables. In a gradual degradation model the properties are degraded as a function of an evolving field variable(s) such as strain.

Relatively few studies have employed fully 3D structural FE formulations despite the importance of out-of-plane behavior for general progressive failure analysis (PFA). The form of the degradation model is partly restricted by the dimensionality of the constitutive behavior captured by the structural model. With that in mind, the governing equations involved in three-dimensional, two-dimensional, and classical laminate theory were briefly described along with the procedures for implementing the degradation models in PFA. Due to its pervasive use, all models were considered in the context of the finite element method (FEM). Models described in terms of composite laminate theory (CLT) were included because they are readily implemented in the FEM. The nature and success of a degradation model is intimately tied to the failure criterion used. Hence, some aspects of various failure criteria were also described and mention is made of the specific failure criteria used with each of the degradation models. Whether or not the failure criterion explicitly identifies each failure mode and other means of identifying failure modes are discussed. All models reviewed identify failure and implement degraded properties at the lamina level.

3.1.1 Sudden degradation models

Due to the explicit identification of failure modes, the Hashin failure criterion [D.24], and several modified forms, have been most popular for PFA studies [D.55, D.58-60, D.63-65, D.68-71, D.73, D.155-178, D.181-183, D.186-189]. Tensor polynomial criteria such as Tsai-Wu [D.21] do not identify the failure modes. For these criteria most authors define failure indices (usually coefficients of the terms in the failure criterion) and assume the largest index identifies the direction of failure which in turn identifies the failure mode. A common assumption is that

the longitudinal direction failure is a fiber failure, transverse direction and shear failures are matrix failure. Out-of-plane shear and normal stresses are used when delamination is included. Some degradation models avoid the issue of failure mode identification by simply degrading all the properties independent of the failure mode [D.51, D.78-81, D.119-123]. Delamination was considered by only a few authors [D.49, D.51, D.52, D.54, D.55, D.63, D.70, D.71, D.73, D.89, D.176-178] and often it is simply assumed catastrophic to the structure upon initiation [D.100-102, D.161, D.162]. Many authors conducted several PFA using different failure criteria [D.49, D.79, D.80, D.98, D.99, D.103, D.116, D.117, D.118, D.121, D.174, D.176, D.181-189] and some general observations can be made. The maximum strain criterion is often reported as giving the poorest results while the maximum stress criterion is frequently associated with unconservative ultimate strength predictions. The Tsai-Wu criterion is often associated with the most conservative results.

There exists a remarkable number and variety of sudden degradation models. Most sudden degradation models involve repeated solution of linear elastic FE models (or CLT equations) where the linear material constants are reduced either at an integration point or in an entire element in some relation to predicted local failure modes. An exception is when nonlinear shear behavior is included in the initial material model. Usually the failure criterion and the degradation model must be modified to be consistent with the nonlinear shear material model. Several authors describe such modeling approaches [D.95, D.129, D.163-165, D.167-170, D.173]. Fiber failure is often considered complete material failure and the total ply discount method is used (all material moduli reduced to zero). Some authors have applied this method for all failure modes with some success in certain situations. However, the general consensus is that some level of mode-dependent degradation is necessary to achieve reliable PFA predictions over a wide range of conditions. Most often, the directionality/mode of the failure is identified and associated material constants are reduced to zero so that the dominant stress(es) associated with the failure are completely relieved. Many authors take a mechanistic view and go beyond this simple directional degradation of properties [D.49, D.52, D.55, D.178]. That is, they associate failure with crack surfaces that can no longer transmit out-of-plane normal or shear stresses and will degrade all the moduli associated with those stress components. In all of these situations the lamina level degraded properties are typically used to calculate modified laminate properties as part of recalculating the new damaged structure stiffness. An exception is where superposed finite elements have been used [D.155, D.173] each superposed element represents a group of lamina with the same fiber orientation. Then, the laminate stiffness reduction is automatically accounted for by the collective stiffness of the superposed elements.

Relatively few authors have implemented partial degradation of material moduli where some significant fraction of the original modulus is retained [D.58-60, D.94, D.123, D.181-183]. The residual stiffness after the prediction of failure typically depends on empirical observations or micromechanics model predictions. For example, transverse ply cracks have a quite localized effect due to the constraining effect of the adjacent plies [D.66, D.67]. Hence, even at a fairly small scale (a few ply thicknesses) the ply will contribute to the local laminate stiffness in the direction normal to the crack face via shear transfer. Physically viewed, this kind of damage generally induces additional anisotropy into the material behavior. For example, a transversely isotropic lamina may become orthotropic. The implementation of such effects depends on whether the analysis utilizes a fully 3D constitutive law or some reduced form.

Because of the intuitive relation between the engineering material constants and the physical material behavior, most authors have described their degradation models in terms of Young's moduli, shear moduli and Poisson's ratios. However, some degradation models were defined in terms of material stiffness matrix components C_{ij} and Q_{ij}. It was shown using inverted relations that these models may have unintended consequences and may not obey energy based restrictions on the engineering constants.

The most attractive feature of the sudden degradation models is their simplicity in implementation due to the binary nature of the models. Additional simplicity can be achieved from the assumptions associated with the failure mode identification (e.g. neglecting delamination or treating all matrix failures the same) or simplified loading such as plane stress conditions. The desire for simplicity and computational expediency has resulted in nearly all of the models in this review employing significant simplifications and therefore achieving limited fidelity of the PFA.

3.1.2 Gradual Degradation Models

The gradual degradation models are models where at least one material property was degraded gradually as a function of some evolving field variable(s). These gradual degradation models generally have the disadvantage of higher computational cost due to the persistent need for iteration of the equilibrium equations.

The gradual degradation models can be separated into two groups. The first group utilize classical forms of failure criteria such as due to Hashin or Tsai-Wu, and upon an prediction of failure, prescribe a fixed function (usually an exponential decay) for the property change that is driven by another evolving field variable (usually strain). These models tend to be pragmatic and assign gradual degradation selectively to certain properties while applying a more efficient sudden degradation to others. The second group of models constructs the degradation model in the framework of classical damage mechanics that was originally developed for modeling plasticity, void growth and rupture in metals. A damage tensor is defined and typically the authors identify the need for several damage parameters due to the anisotropy and multiplicity of failure mechanisms that occur in laminated composites. Because of this complexity, these models generally are restricted to plane stress conditions at the lamina level and/or a single failure mode such as matrix transverse cracking. The failure criterion forms the initial damage surface and may be based on a classical failure criterion but must be recast in terms of the damage variables. Often, the principles of thermodynamics of irreversible processes are adhered to. The details of such models can be significant and were beyond the scope of this review. The goal was to outline the essentials so the reader can identify basic differences or commonalities of the various models.

As a general observation, fiber failure is usually treated as complete failure in sudden degradation models and the properties are degraded to zero. However, in gradual degradation models, to simulate fiber failure, several authors [D.196, D.197, D.199-201, D.211, D.220, D.222-224] have utilized exponential decay functions based on fiber bundle theory [D.198]. Depending upon the parameters involved in the function, these may be effectively step functions

or gradually degrading functions. However, the models were based on mathematical functions of other evolving field variables and hence were classified as gradual degradation models.

The degradation models that utilize damage tensors are not necessarily based on the thermodynamics of irreversible processes (TIP) for the evolution of the damage parameters. As the number of damage variables increases, the complexity in formulating evolution laws for the variables using TIP also increases. However, as the number of damage variables increase, the ability to capture different damage mode progressions increases as well. All the formulations based on TIP follow the same process but do not necessarily involve the same damage parameters, potential functions, and dissipation potentials. Therefore, describing these models in detail is a formidable task and was beyond the scope of the review.

3.2 Matrix Cracking and Prediction of Residual Properties

For an accurate representation of damage, gradual reduction of properties with increasing damage is important. In what follows two different types of cracking are characterized. First, a matrix failure mode characterized by matrix cracks oriented perpendicular to the fibers and termed as *matrix normal cracking* is described. A unit cell model is devised based on the characteristics of the failure mode. Residual properties of the cracked lamina with increasing density of matrix normal cracks are estimated utilizing the volume-averaged stresses and strains obtained from a finite element model of the unit cell. Several characteristics of the failure/damage mode are examined through the finite element model. Second, *matrix ply cracking,* characterized by cracks that run parallel to the fibers and are oriented in the transverse direction of the lamina are modeled. This damage mode is described and a unit cell model for cross-ply laminates with cracked 90^o laminae is developed based on simplified assumptions of the failure mode. The residual properties of a cracked lamina were estimated utilizing volume-averaged stress/strain information obtained from a finite element model of the laminate unit cell. The predictions of the finite element model are then compared with experimental data found in the literature. Utilizing the finite element model, several characteristics of the failure mode were examined. The results serve to illustrate the constraining effect of the adjacent plies on the residual properties of a cracked ply. A unit cell model is also described for a laminate with cracked angle plies. Again, a finite element model of the unit cell was utilized for estimating the residual properties of the cracked angle plies.

3.2.1 Matrix normal cracking (MNC)

In this section, a finite element micromechanics model is described for estimating the lamina properties assuming the constituent properties and the volume fractions are known. The finite element micromechanics model is intended to represent a continuum point in a lamina by simulating a periodic unit cell that is representative of the lamina behavior. For a thorough discussion of the concept of representative volume and unit cell concepts the reader is referred to Nemat-Nasser and Hori (1993).

3.2.1.1 FEM Micromechanics of a unit cell

Uniform hexagonal fiber packing was assumed and the rectangular unit cell shown in Fig. 27 was modeled using the FEM. The boundary conditions for the rectangular model are based on the periodicity of the unit cell. That is, the unit cell can be translated in all the three Cartesian

coordinate directions to generate the composite lamina. The principle of periodicity was applied as outlined by Whitcomb et al (2000).

For estimating the lamina extensional moduli E_{11} and Poisson's ratios υ_{12} and υ_{13}, the unit cell is subjected to displacement gradient in the 1-direction. For estimating the extensional modulus E_{22} and Poisson's ratio υ_{23}, the unit cell is subjected to a displacement gradient in the 2-direction. For estimating the lamina shear moduli G_{12}, G_{13}, and G_{23}, the composite is subjected to displacement gradients that invoke volume average pure shear in the unit cell. The coefficients of thermal expansion α_{11}, α_{22}, and α_{33} are estimated by subjecting the composite to a uniform temperature change.

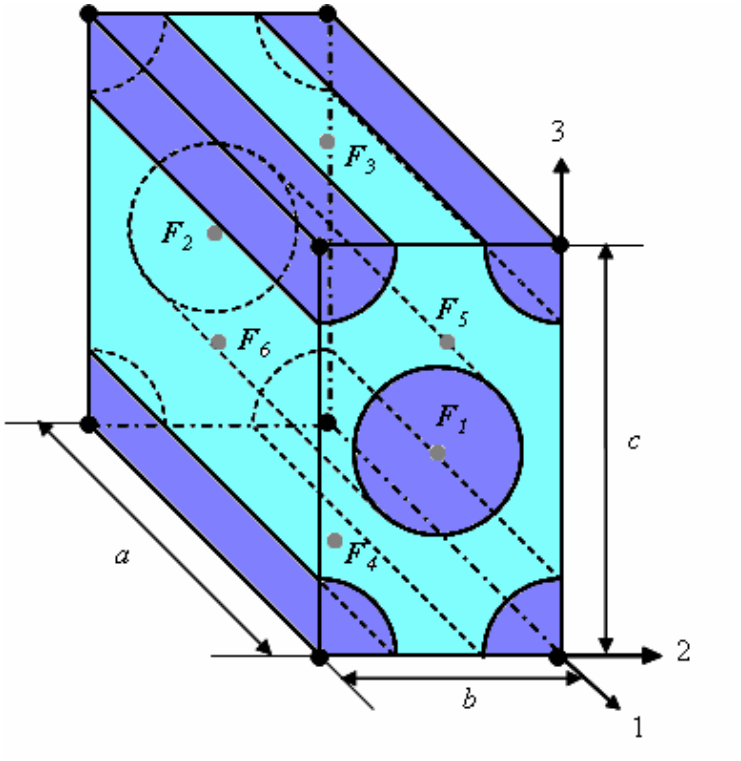


Figure 27. Rectangular unit cell model with assumed hexagonal fiber packing

The thermo-elastic properties of the constituents are listed in Table 10 and the corresponding predicted thermo-elastic properties of various composite laminae are listed in Table 11. The properties of AS4 and 3501-6 are obtained from Karami and Garnich (2005a). The laminae properties are the baseline (undamaged) properties for comparison with predictions of damaged material properties that follow.

Table 10. Thermo-elastic properties of the constituents

Property	AS4	3501-6
E_{11} (GPa)	201.0	4.3
E_{22} , E_{33} (GPa)	13.5	4.3
υ_{12}, υ_{13}	0.22	0.35
G_{12}, G_{13} (GPa)	95	1.593
G_{23}	0.38	0.35
α_{11} (μ $^{o}C^{-1}$)	-0.7	55
α_{22}, α_{33} (μ $^{o}C^{-1}$)	8.0	55

Table 11. Thermo-elastic properties of composite laminae

Property		
Fiber/Matrix	AS4/ 3501-6	AS4/ 3501-6
Fiber volume fraction	0.45	0.66
E_{11} (GPa)	92.83	134
E_{22} , E_{33} (GPa)	7.3	9.1
υ_{12}, υ_{13}	0.287	0.261
G_{12}, G_{13} (GPa)	4.06	7.3
G_{23} (GPa)	2.482	3.16
α_{11} (μ $^{o}C^{-1}$)	0.8402	-0.0178
α_{22}, α_{33} (μ $^{o}C^{-1}$)	42.02	28.55

3.2.1.2 Predicted Properties for a Lamina with MNC

Matrix normal cracks are characterized by cracks that span the entire thickness of the ply and are oriented perpendicular to the fibers. Also, the fibers are assumed undamaged and still perfectly bonded to the matrix in this damage mode. The rectangular unit cell model described earlier was modified to incorporate matrix normal cracks. The matrix cracks are incorporated into the rectangular model by changing/removing some of the periodic boundary conditions on the unit cell model of an undamaged material. Specifically, the crack planes perpendicular to the fibers are free surfaces on the matrix portion and all constraints are removed from those surfaces (faces 1 and 2 indicated in Fig. 28 with shaded regions).

Damage in the lamina due to matrix normal cracks was defined as the increase in the number of cracks or the decrease in average crack spacing. The crack density parameter "ς" defined as the inverse of the crack spacing ($1/a$) relates the effective properties of a cracked lamina with damage. However, to relate the crack density to physical parameters of the composite, we utilize

a normalized crack density parameter "ξ" defined as the crack density times the distance between the centers of two adjacent fibers.

In extracting the residual effective lamina properties, the volume-averaged stresses and strains obtained from the finite element unit cell model were used. In describing the degraded properties, the normalized properties were used for comparison purposes. The normalized properties denoted by a subscript "(n)" are obtained by dividing the degraded properties with their respective undamaged values. Note that the effective properties of cracked laminae estimated here define the elastic behavior of the laminae for crack opening loads. For crack closing loads, it is assumed that the cracked composite lamina damaged by MNC retains its stiffness and the elastic behavior is the same as that of an undamaged lamina.

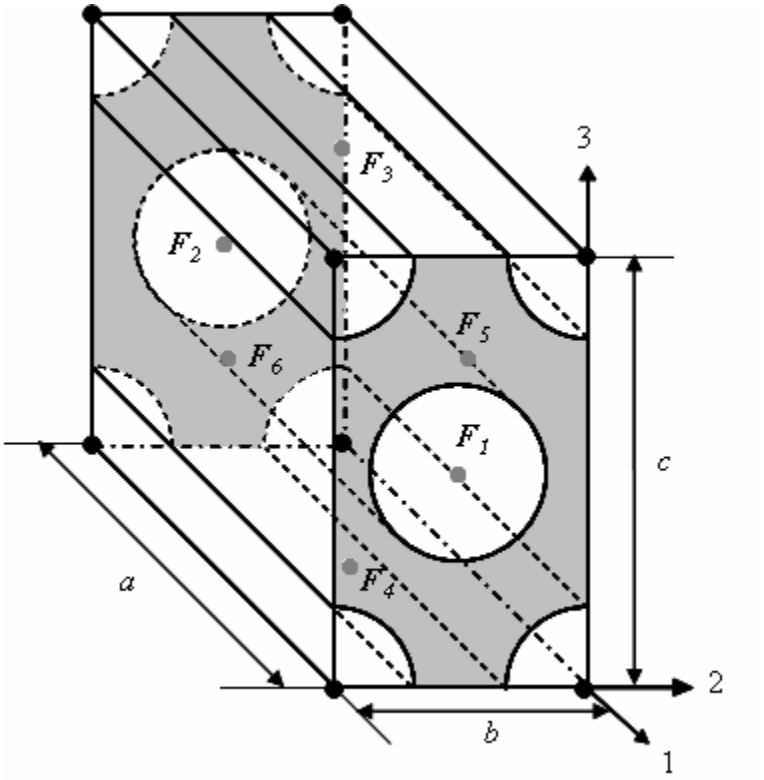


Figure 28. Unit cell model for a lamina with MNC showing the crack surfaces in grey

To study the influence of varying fiber volume fraction on the effective properties of a cracked lamina, two AS4/3501-6 composites with fiber volume fractions of 0.45 and 0.66 were studied. Fig. 29 shows the variation of the normalized extensional moduli $E_{11(n)}$, $E_{22(n)}$, and $E_{33(n)}$ with normalized crack density (ξ). Fig. 30 shows the variation of the normalized shear moduli $G_{12(n)}$, $G_{13(n)}$, and $G_{23(n)}$ with normalized crack density. Fig. 31 shows the variation of normalized Poisson's ratios $\upsilon_{12(n)}$, $\upsilon_{13(n)}$, and $\upsilon_{23(n)}$ with normalized crack density. Fig. 32 shows the variation of CTE $\alpha_{11(n)}$, $\alpha_{22(n)}$, and $\alpha_{33(n)}$ with normalized crack density of a AS4/3501-6 lamina

with 0.45 fiber volume fraction whereas Fig. 33 shows the variation of CTE $\alpha_{11(n)}$, $\alpha_{22(n)}$, and $\alpha_{33(n)}$ of a AS4/3501-6 lamina with 0.66 fiber volume fraction.

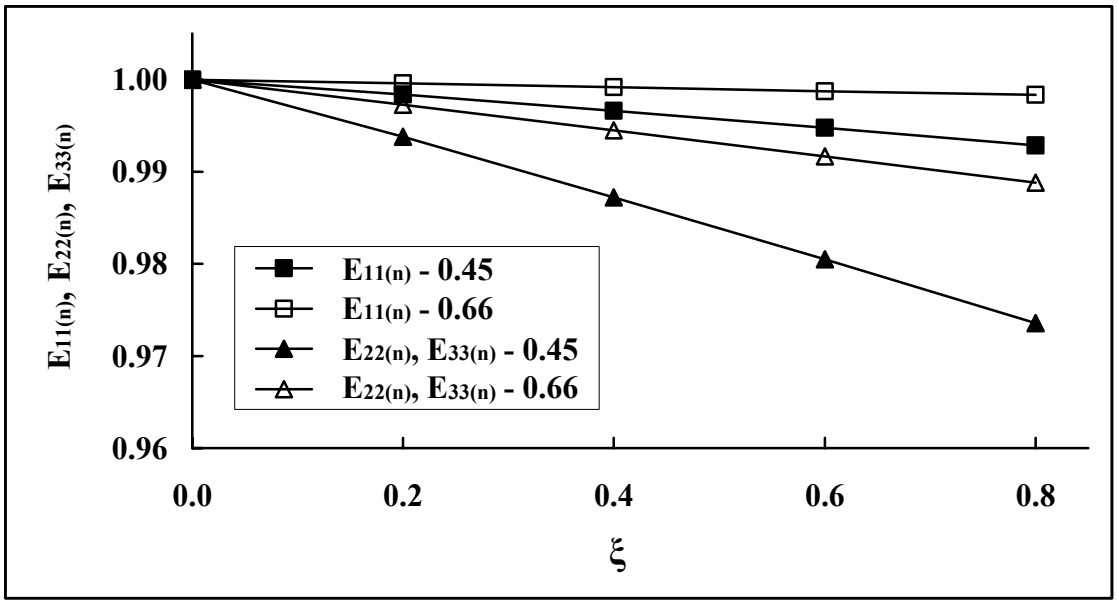


Figure 29. Variation of normalized extensional moduli $E_{11(n)}$, $E_{22(n)}$, and $E_{33(n)}$ with normalized crack density of AS4/3501-6 composite laminae with volume fractions of 0.45 and 0.66 and damaged with MNC

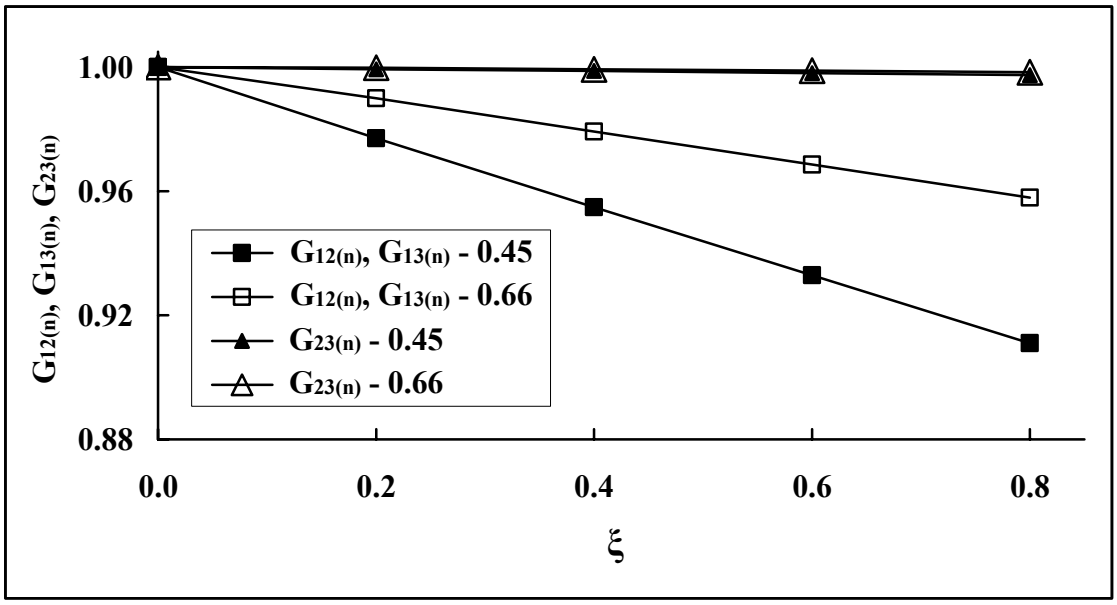


Figure 30. Variation of normalized shear moduli $G_{12(n)}$, $G_{13(n)}$, and $G_{23(n)}$ with normalized crack density of a AS4/3501-6 composite laminae with volume fractions of 0.45 and 0.66 and damaged with MNC

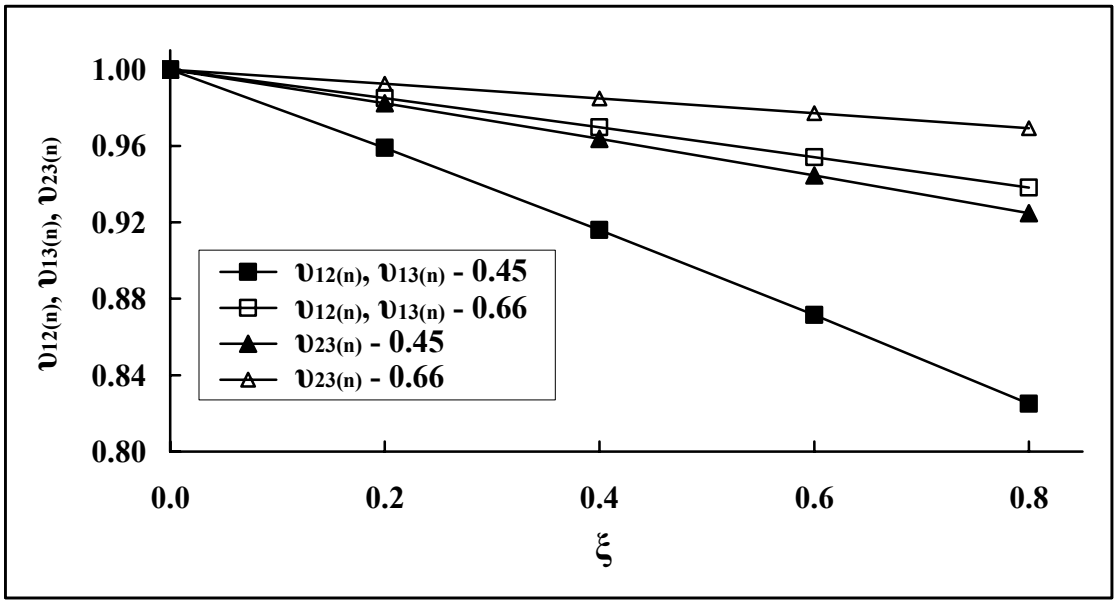


Figure 31. Variation of normalized Poisson's ratios $\upsilon_{12(n)}$, $\upsilon_{13(n)}$, and $\upsilon_{23(n)}$ with normalized crack density of an AS4/3501-6 composite laminae with volume fractions of 0.45 and 0.66 and damaged with MNC

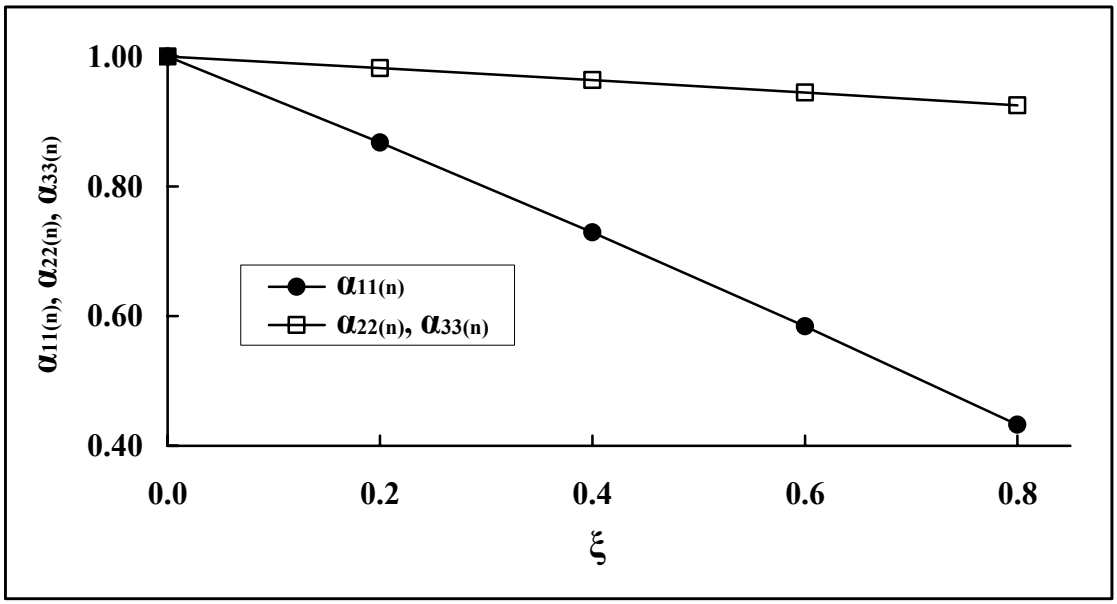


Figure 32. Variation of normalized coefficients of thermal expansion $\alpha_{11(n)}$, $\alpha_{22(n)}$, and $\alpha_{33(n)}$ with normalized crack density of an AS4/3501-6 composite lamina with a volume fraction of 0.45 and damaged with MNC

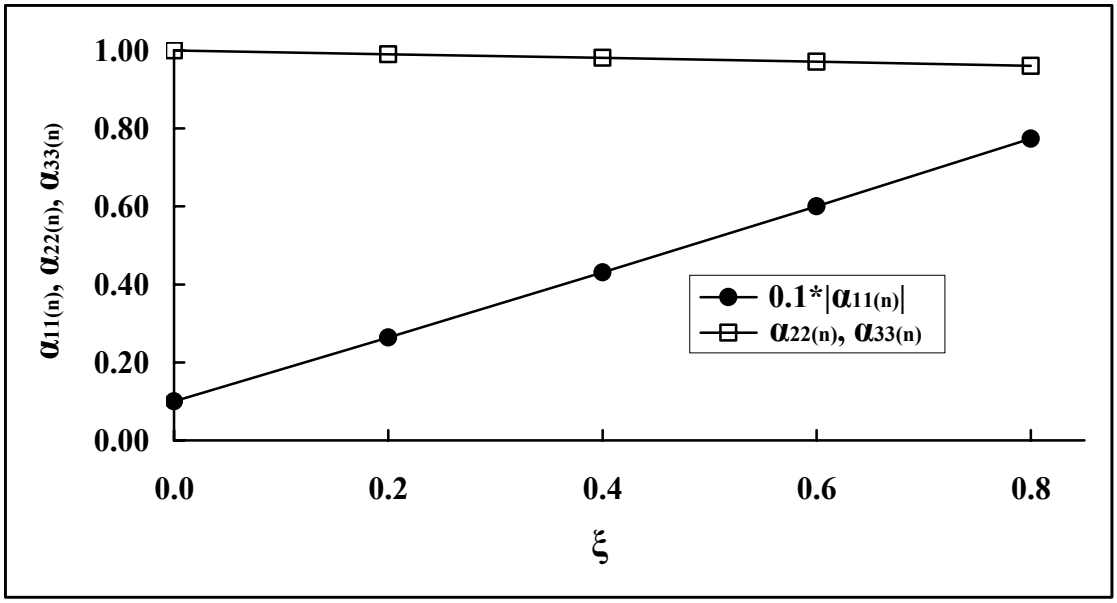


Figure 33. Variation of normalized coefficients of thermal expansion $\alpha_{11(n)}$, $\alpha_{22(n)}$, and $\alpha_{33(n)}$ with normalized crack density of an AS4/3501-6 composite lamina with a fiber volume fraction of 0.66 and damaged with MNC

3.2.1.3 Discussion of MNC

Matrix normal cracking in unidirectional composite laminae was studied for AS4/3501-6 composite laminae in two different fiber volume fractions. From the analysis, it was observed that composite lamina retains transverse isotropy after damage due to MNC. This is because the plane of the matrix cracks coincides with the plane of transverse isotropy of the lamina. It was also observed that all the properties of the lamina were influenced by matrix normal cracks. This is because of the Poisson's ratio interaction between the intact fibers and the damaged matrix.

MNC primarily influence the behavior of the composite in the fiber direction because of the orientation of the cracks. However, since the fibers are intact in this failure/damage mode for a composite system with a high ratio of fiber and matrix modulus in the fiber direction, MNC moderately altered the composite modulus. Hence, it was observed that matrix normal cracks did not significantly alter the elastic behavior of the composite. The maximum reduction in elastic moduli of a composite due to MNC was less than 10% even at very high crack densities. However, the coefficient of thermal expansion in the fiber direction for graphite fiber composites was strongly influenced by MNC. Note, however, the thermal expansion for the fiber direction is very small so that small absolute changes correspond to large percentage changes.

Based on the analysis presented in this section it may be concluded that although MNC does affect lamina properties, the effect is not large compared to other failure/damage modes. Also, this damage mode is difficult to detect since the intact fibers generally close the cracks. Also, many matrix materials have strain capability exceeding that of the fibers so this failure mode does not occur in many materials. An exception would be for a large temperature decrease in graphite systems, as in space applications, where the negative axial CTE of the fibers causes

large tensile stress/strain in the matrix even in the absence of mechanical loads. As a result of these observations this failure/damage mode has seen almost no attention in the literature.

3.2.2 Matrix Ply Cracking (MPC)

Consider a $[0/90/90/0]_T$ laminate subjected to a tensile load in as shown in Fig. 34. Here, the *x*-direction corresponds to the fiber direction in the 0^o plies. After sufficient loading, the 90^o plies develop cracks (as depicted in Fig. 34) that form parallel to the fibers and span the length and thickness of the plies. This classic failure/damage mode is termed matrix ply cracking. Even though this phenomenon was first addressed in the context of cross-ply laminates, this failure mode is most often the first failure mode observed in most laminates under multi-axial loading and especially in extreme low temperature environments as in space applications.

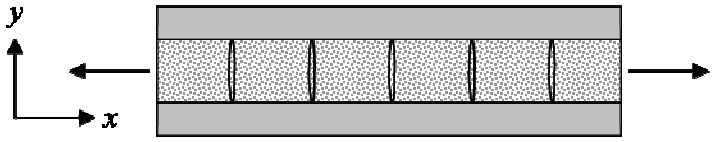


Figure 34. Matrix microcracking in a [0/90/90/0] laminate

The characteristics of MPC were investigated by Garrett and Bailey (1977a,b), Parvizi et al. (1978), Parvizi and Bailey (1978), Bailey et al. (1979), Bader et al. (1979) and Bailey and Parvizi (1981) in a series of experiments on cross-ply laminates made up of glass reinforced and carbon reinforced unidirectional laminae. They reported whitening of the transverse plies was prior to the transverse crack initiation. Studies on glass fiber reinforced epoxy matrix and polyester matrix composites revealed that this whitening effect was associated with fiber/matrix debonding. Upon further loading, the laminates exhibited multiple cracks in the transverse plies eventually forming a saturated crack distribution with almost even spacing. The rate of crack formation was rapid after the first crack and slowed with increasing load. The cracks often penetrated into the adjacent plies but were hindered by the fibers.

Based on the above-described experimental observations, the mechanism of matrix ply cracking may be summarized as follows. The fiber debonds and other matrix sub-microcracks associated with the whitening eventually coalesce to form microcracks that span a few fiber diameters. Upon continued straining, the microcracks grow to form transverse matrix cracks that span the entire thickness and width of the plies. On further loading, multiple cracks are formed in the transverse plies. The multiple crack formation eventually reaches a maximum, characterized by crack spacing that is nearly uniform. Highsmith and Reifsnider (1982) investigated this crack formation and termed this maximum crack density as a *Saturated Damage State*.

Pradhan et al. (1999b) postulated that the minimum crack spacing in the 90^o laminae of a laminate (prior to the failure of the laminate) is typically of the order of the thickness of the 90^o plies. Even though the thickness of the cracked plies is a major factor influencing the minimum crack spacing, experimental data obtained from three sources, Groves et al. (1987), Lee and Hong (1993), and Joffe and Varna (1999) suggests that the properties and relative orientations of the cracked lamina and the adjacent laminae also influence the minimum crack spacing.

The past three decades have seen numerous studies of the thermo-mechanical behavior of laminates damaged by matrix ply cracks. Several analytical methods were utilized in studying matrix ply cracking of which the finite element method has been utilized not only to study the stress distribution in the plies, crack opening displacements, etc., but also to estimate the residual thermo-mechanical properties of a cracked laminate/lamina. In the following, results obtained from several finite element models are compared with those obtained by various researchers both analytically and experimentally for cracked laminates/laminae.

In analyzing matrix ply cracks, typically the response of a laminate with increasing matrix cracks is studied. In particular, the normalized laminate modulus forms the basis for characterizing the effect of matrix ply cracks. However, as the fibers are generally stiffer than the matrix, the laminate modulus is dominated by the undamaged fiber direction lamina modulus. Consequently, the small changes in modulus caused by matrix ply cracks are difficult to measure accurately. For numerical illustration purposes, consider a $[0/90]_S$ cross-ply laminate made up of composite laminae with a E_{11}/E_{22} ratio of 9. A rule of mixtures calculation assuming 80% reduction in modulus E_{22} for the cracked 90^o plies, the resulting laminate modulus reduces by only 8%. Experimentally capturing this small change in the laminate modulus requires highly accurate techniques.

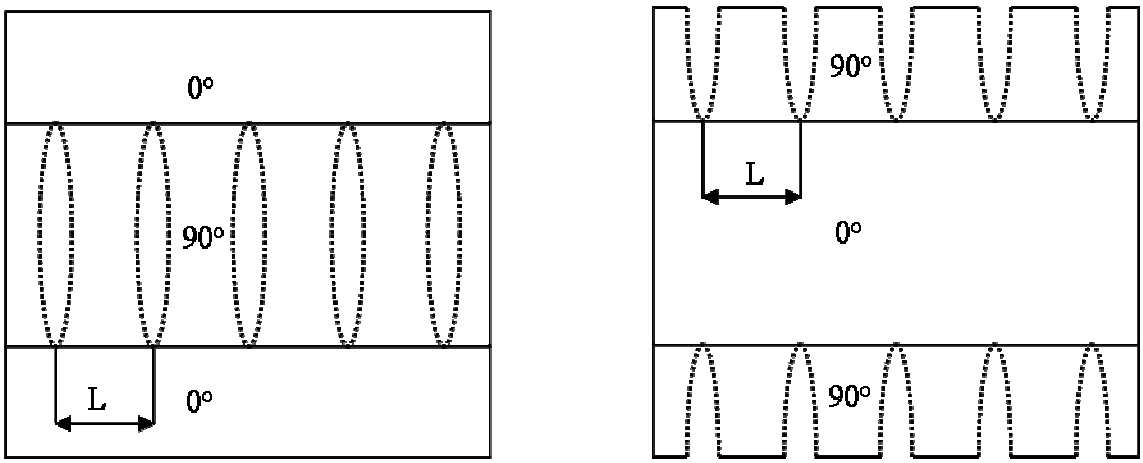
Based on review of the literature, a unit cell model utilizing the finite element method was pursued to study the influence of MPC on the effective properties of a cracked lamina. The primary objective was to understand the thermo-mechanical behavior of cracked 90^o plies with increasing MPC. Since the material properties define the behavior of a lamina, the effects of various parameters such as the thickness of the cracked plies, the position of the cracked plies, the orientation of the neighboring plies, etc. on the effective thermo-mechanical properties of a cracked lamina were investigated. In this pursuit, some previously established results were examined.

3.2.2.1 Unit Cell Model for Cracked 90^o Laminae

Both two-dimensional and three-dimensional finite element models have been utilized in the past to study MPC. The unit cell geometries and the boundary conditions imposed on the models were based on periodicity derived from characteristic assumptions of MPC. In designing the present unit cell model for the cracked lamina, the following simplifying assumptions were employed.

1. The cracks span the entire length of the cracked lamina.
2. The cracks span the entire thickness of the cracked ply, are orthogonal to the ply, and do not penetrate into the adjacent plies.
3. The cracks in the lamina are uniformly spaced.
4. The crack faces are flat and run parallel to the fibers.

Based on the above assumptions for matrix ply cracking, the arrangement of the matrix cracks in cracked laminae can be depicted as shown in Fig. 35(a) and 35(b) for interior and outer cracked plies respectively. These arrangements are for cross-ply laminates with cracks in the 90^o laminae. Note that Fig. 35 depicts the cross-sectional front view of a cracked laminate where the curved dotted lines represent crack faces.

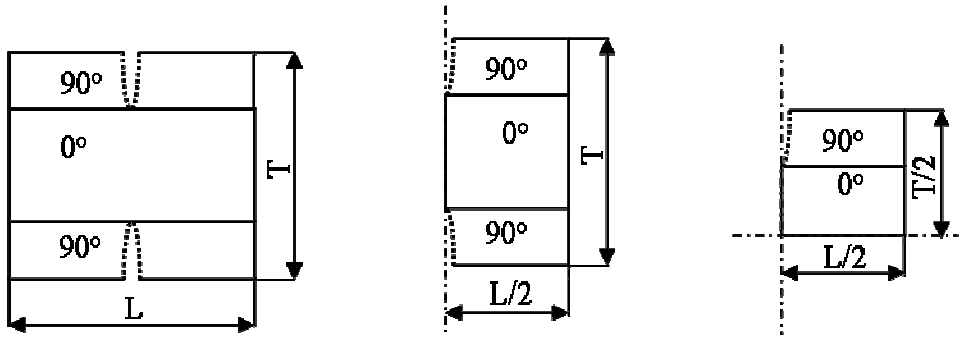


(a) (b)

Figure 35. (a) Cracked laminate with periodic matrix ply cracks in the 90° interior plies (b) Cracked laminate with periodic matrix ply cracks in the outer 90° plies

In view of the idealizations shown in Fig. 35, Fig. 36 depicts possible unit cell geometries based on periodicity for a cracked cross-ply laminate with an outer 90° ply. Similar unit cells may be developed for a centered cracked lamina. Note that "L" is the crack spacing (distance between two cracks) and "T" is the thickness of the laminate.

The unit cell depicted by Figures 36(b) and 36(c) can be reflected and translated in the plane of the laminate to obtain the entire laminate with uniformly spaced matrix ply cracks. For increased computational efficiency, these reduced unit cell models may be utilized for studying MPC. However, the required boundary conditions to be imposed on the finite element model of these unit cells are more complicated for reduced unit cells. Here, for simplicity a model of the full unit cell (Fig. 36(a)) was utilized.



(a) (b) (c)

Figure 36. (a) Full unit cell for a cracked cross-ply laminate (b) Half unit cell for a cracked cross-ply laminate (c) Quarter unit cell for a cracked cross-ply laminate

The full three-dimensional volume modeled to represent a cross-ply laminate with cracks in the outer plies is depicted in Fig. 37. Note that the thickness of the laminate "T" is denoted by "c" and the length of the unit cell "L" (also the crack spacing) is denoted by "a", whereas the width

of the unit cell is denoted by "b". Note that the width "b" is arbitrary as there are no variations in the *y*-direction.

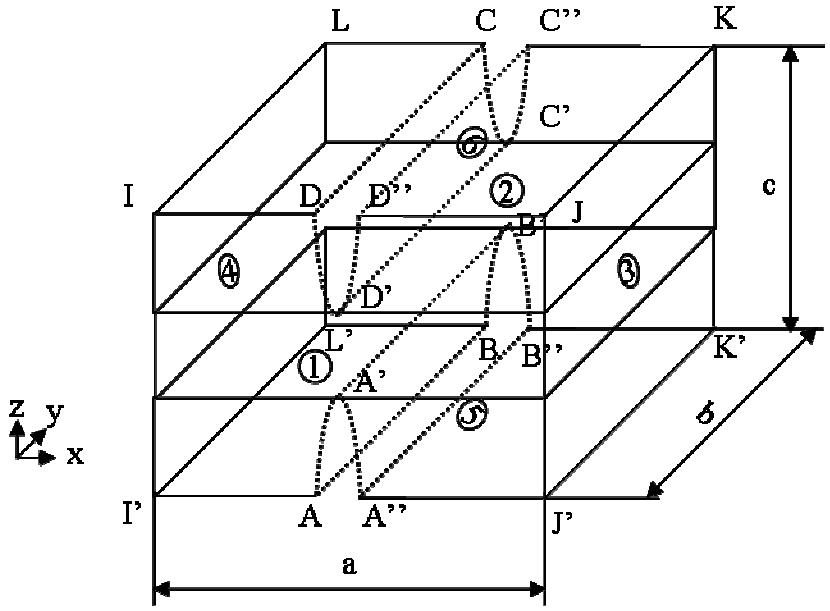


Figure 37. A three-dimensional unit cell model for a cracked cross-ply composite laminate with matrix ply cracks in the outer plies

The three-dimensional finite element model was used to estimate the effective residual properties of a cracked lamina. The boundary conditions for the unit cell models depicted in Figures 37 and 38 were based on periodicity. Two sets of boundary conditions were required; one for periodicity in the plane of the laminate and a second for periodicity in all the three Cartesian directions.

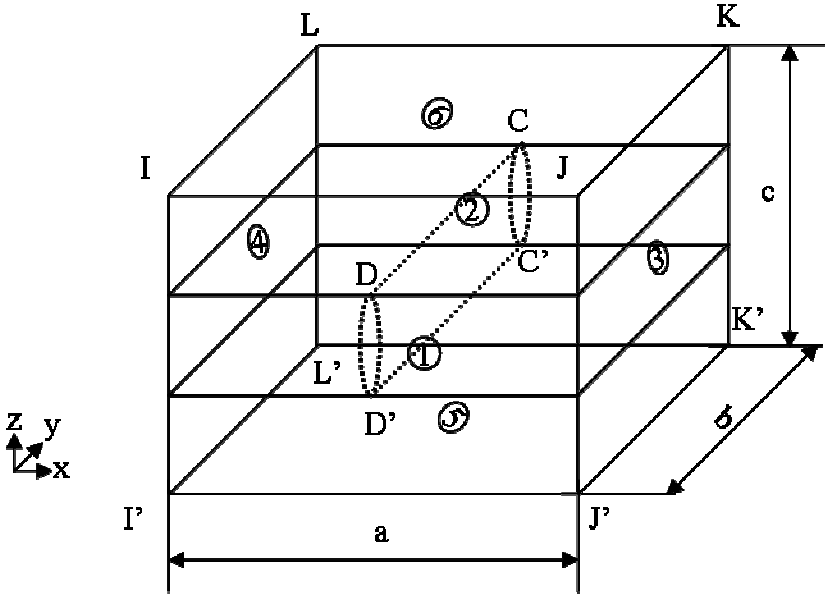


Figure 38. A unit cell model for a cracked laminate with matrix ply cracks in the 90^o plies

Finally, the thermal deformation of the cracked ply was considered. Based on the assumption that the matrix cracks are oriented such that the crack-face-normal is oriented in the 2-direction the coefficients of thermal expansion (CTE) parallel to the crack face (in the 1 and 3 directions) are assumed unaffected by the cracks. Therefore, it is assumed that only α_{22}^{d} changes due to matrix ply cracking. Note that the influence of MPC on lamina CTE does not seem to have been previously investigated.

Four different load cases are utilized to estimate the properties. The first load case simulates a three-dimensional normal stress/strain state in the lamina to estimate E_{22}^{d}, υ_{21}^{d}, and υ_{23}^{d}. The second load case creates inplane shear stress in the composite from which the effective inplane shear modulus G_{12}^{d} was estimated. The third load case consists of a thermal load and was used to estimate the residual CTE α_{22}^{d}. The fourth load case simulated the out-of-plane shear stress to estimate the effective out-of-plane shear modulus G_{23}^{d}.

3.2.2.2 Volume-averaged Properties of a Cracked Lamina

Due to periodicity, the stress/strain state in the unit cell is representative of the average stress/strain response of a larger volume. Utilizing the volume-averaged stress/strain information obtained from the finite element model, the residual properties of a cracked lamina were estimated. The general procedure to obtain the volume-averaged stresses and strains for the deformed configuration of the cracked lamina is described below. Note that the analysis was based on the assumptions of small-displacement formulation and linear elastic behavior of the damaged composite laminae.

To validate the present models predictions were compared with experimental data obtained from three sources, Groves et al. (1987), Lee and Hong (1993), and Joffe and Varna (1999). Table 12 shows the measured properties of the undamaged laminae for the three materials. Note that the thickness (h) reported in Table 12 is for one ply thickness and the υ_{23} are assumed values for the AS4/3502 and the glass/epoxy. Also, prior to MPC the laminae materials were assumed transversely isotropic.

Table 12. Properties of different uncracked lamina

Data	Material	E_{11} (G Pa)	E_{22} (G Pa)	υ_{12}	υ_{23}	G_{12} (G Pa)	h (mm)
Groves	AS4/3502	144.78	9.58	0.31	0.4	4.785	0.127
Lee	P3051-F05	127.8	9.4	0.28	0.52	4.2	0.05
Varna	Glass/Epoxy	44.73	12.76	0.297	0.42	5.8	0.144

For a range of crack densities, Fig. 39 compares the calculated laminate modulus with the experimental data obtained from Groves et al. (1987). In this and several figures that follow, (E) indicates experimental values whereas (A) indicates analytical finite element predictions. Also,

the subscript '(*n*)' denotes the normalized values of the respective properties obtained by dividing the degraded property by its undamaged value. Fig. 40 and Fig. 41 show the variation of the normalized lamina modulus $E_{22(n)}$ with increasing crack density for the laminates studied experimentally by Groves et al. (1987). Note that one of the cross-ply laminate sequences considered in the study by Groves et al. (1987) is a $[0/90/0]s$ laminate with cracks in the *inner* 90^o cracked laminae. The cracked laminae are termed *inner* because the cracked lamina was located in between two intact 0^o plies.

Fig. 42 shows a comparison of the normalized *laminate* modulus $E_{X(n)}$ obtained from the predictions of the finite element unit cell model with the experimental data obtained from Lee and Hong (1993). Fig. 43 shows the variation of the normalized cracked *lamina* modulus $E_{22(n)}$ with increasing crack density corresponding to the laminate test cases defined in Fig. 42. Fig. 44 and Fig. 45 show the comparison of the normalized laminate modulus and the normalized laminate Poisson's ratio obtained from the analytical finite element model and the experimental data from Joffe and Varna (1999). Fig. 46 shows the comparison of the normalized lamina modulus $E_{22(n)}$ with experimental results from Joffe and Varna (1999).

The analytical predictions of normalized laminate modulus $E_{X(n)}$ shown in Fig. 39 (4.8) and 42 were obtained using the cracked lamina modulus and a rule-of-mixtures formula. The analytical predictions of normalized laminate modulus and normalized laminate Poisson's ratio shown in Fig. 44 and 45 were obtained from Classical Laminate Theory (CLT) [Hyer (1998)]. Note that the analytical normalized laminate moduli obtained from the rule-of-mixtures and CLT are approximately the same for cross-ply laminates. The experimental normalized lamina moduli shown in Fig. 40, 41, 42, and 46 were obtained by reverse engineering using the laminate modulus and the rule-of-mixtures formula.

From Fig. 39 through 46, as a general observation, the results obtained from the finite element model compared well with the experimental results. However, in cases where the 0^o plies severely dominate the stiffness (e.g. Fig. 41) the comparisons were poor for the normalized laminate moduli and the normalized lamina moduli. This difference may be partly attributed to the fact that the cracked plies cause a small laminate stiffness change that is difficult to measure with accuracy. The most credible experimental data provides the best agreement and it was concluded that the present models do an excellent job of predicting degraded lamina properties.

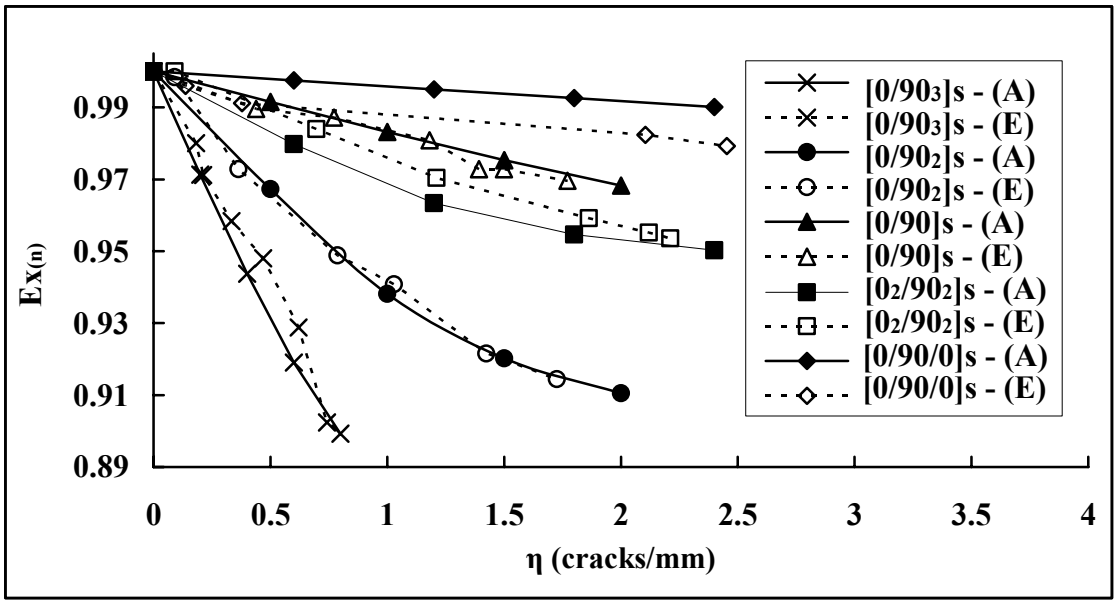


Figure 39. A comparison of normalized <u>laminate</u> modulus Ex(n) of cross-ply laminates obtained from the finite element model with experimental results obtained from Groves et al. (1987)

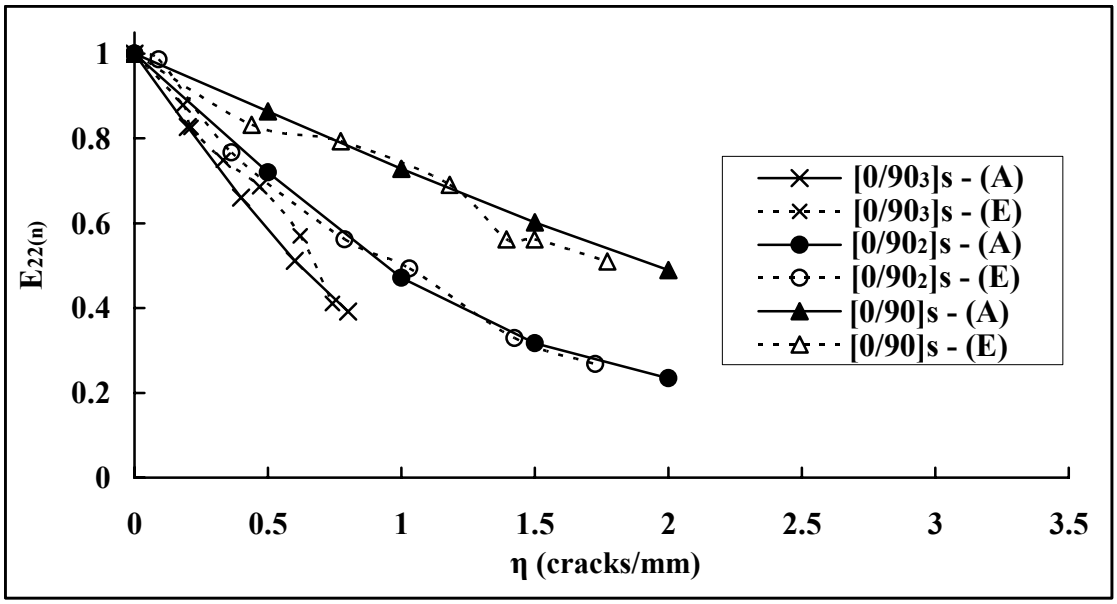


Figure 40. Normalized <u>laminae</u> modulus E22(n) obtained from finite element model for cross-ply laminates compared to experimental data from Groves et al. (1987)

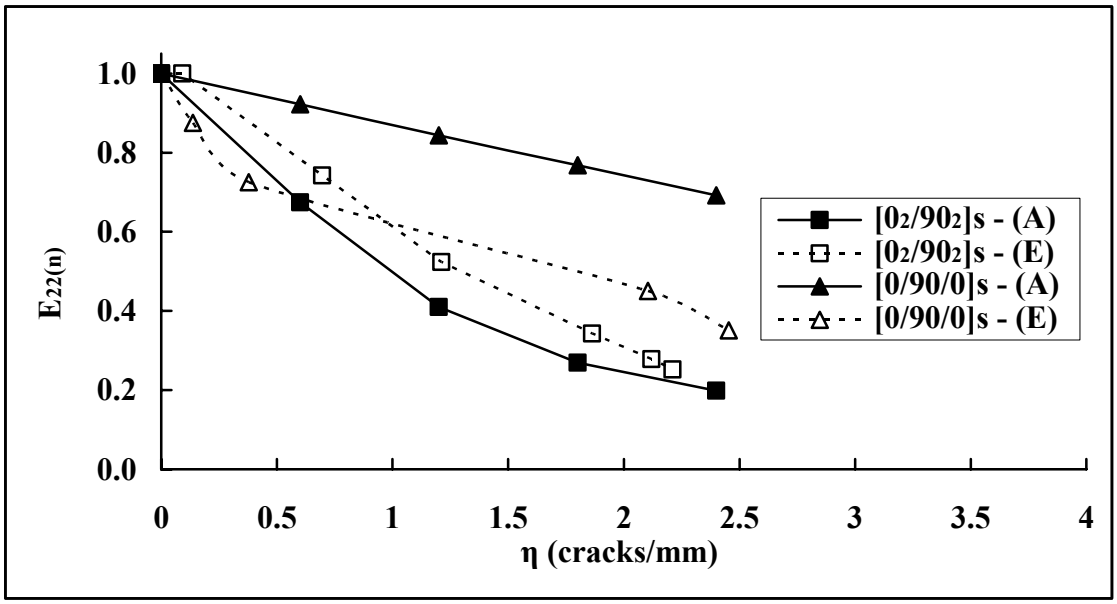


Figure 41. Normalized laminae modulus E22(n) obtained from finite element model for cross-ply laminates compared to experimental data from Groves et al. (1987)

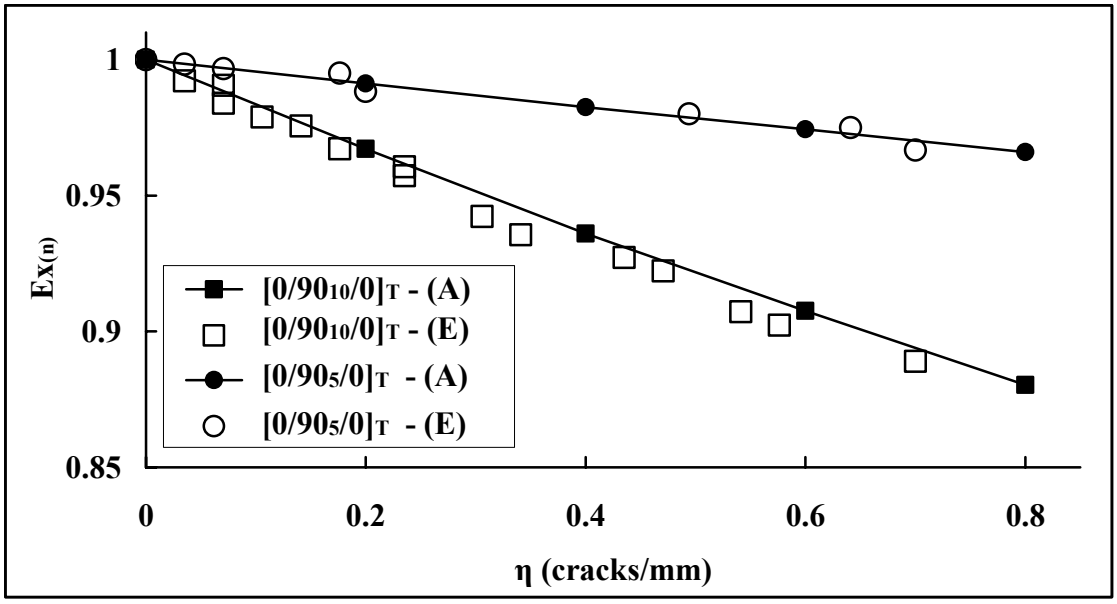


Figure 42. A comparison of normalized laminate modulus Ex(n) of cross-ply laminates obtained from the finite element model with experimental results obtained by Lee and Hong (1993)

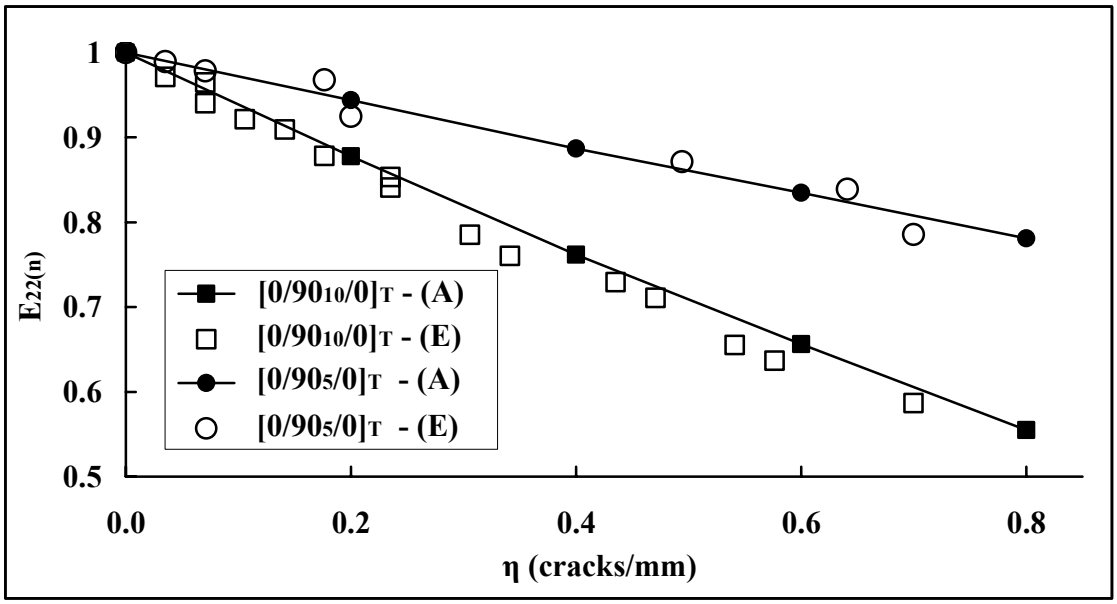


Figure 43. Normalized <u>laminae</u> modulus E22(n) obtained from finite element model for cross-ply laminates compared to experimental data from Lee and Hong (1993)

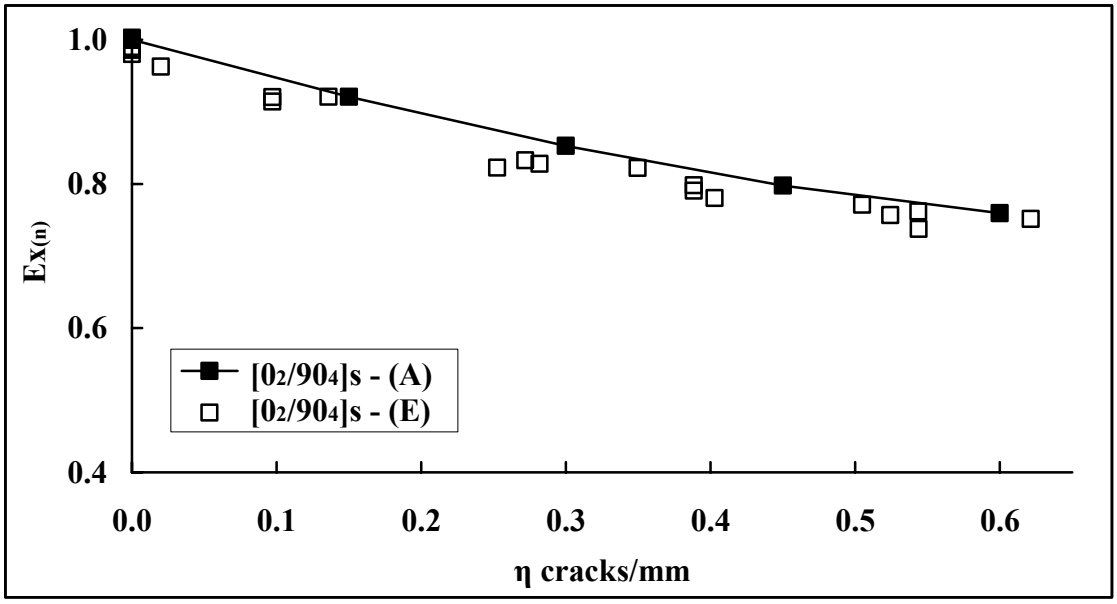


Figure 44. A comparison of normalized <u>laminate</u> modulus Ex(n) of cross-ply laminates obtained from the finite element model with experimental results obtained from Joffe and Varna (1999)

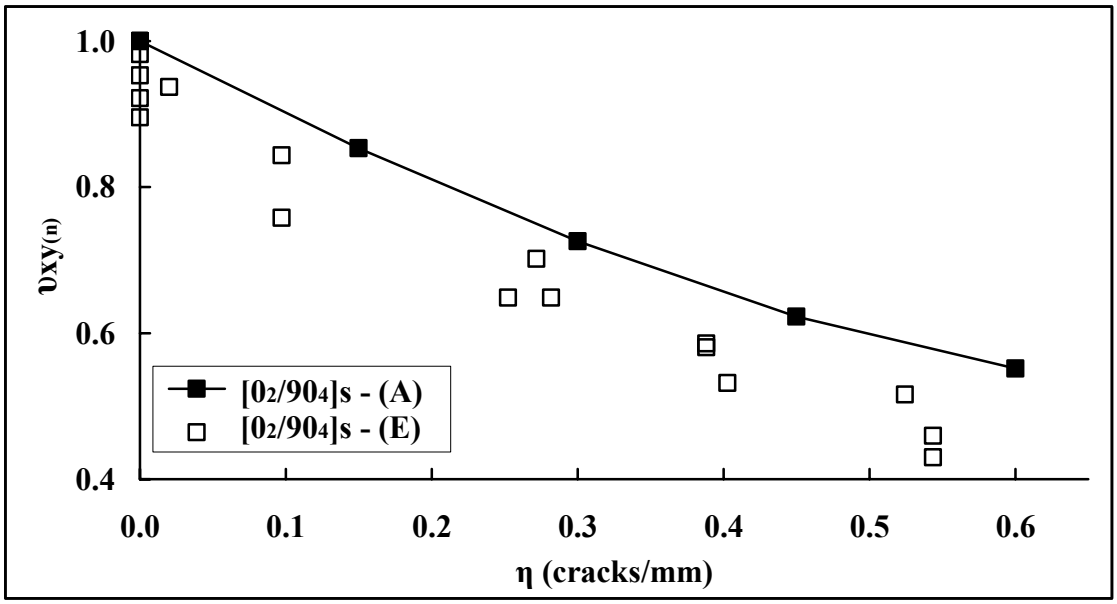


Figure 45. A comparison of normalized laminate Poisson's ratio υxy(n) obtained from the finite element model with experimental results obtained from Joffe and Varna (1999)

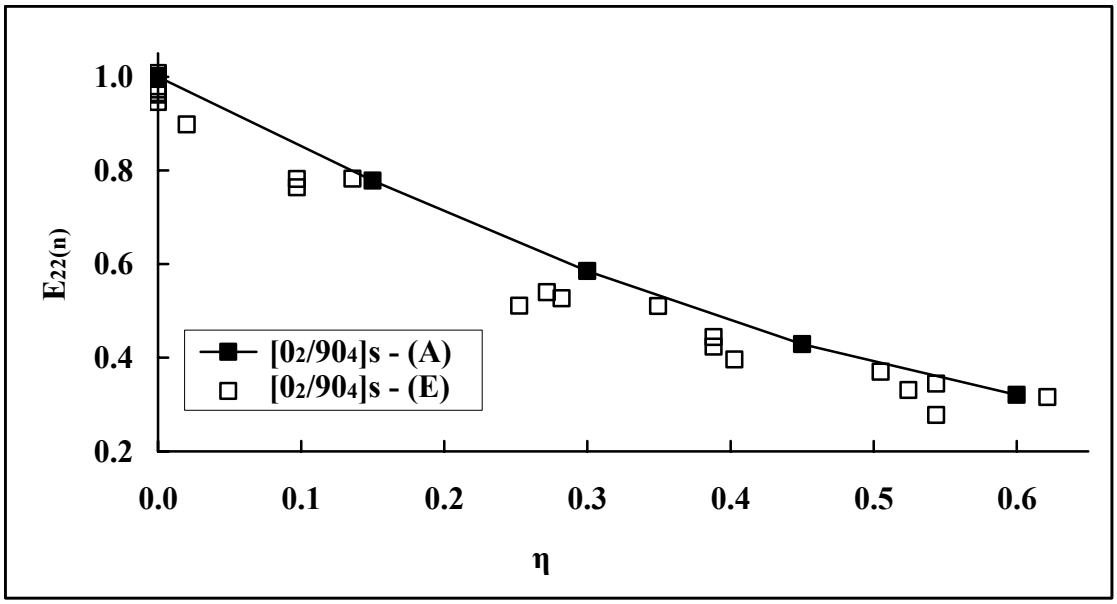


Figure 46. Normalized laminae modulus E22(n) obtained from the finite element model for cross-ply laminate compared to experimental data from Joffe and Varna (1999)

In extracting the residual properties of a cracked lamina for a given crack density, it was observed that the degraded lamina modulus E_{22}^{d} and the Poisson's ratios υ_{21}^{d} and υ_{23}^{d} degrade by the same percentage as a function of crack density. However, the inplane shear modulus and out-of-plane shear modulus degrade by different percentages.

Utilizing the finite element unit cell model for a cracked lamina it was also observed that the CTE of the cracked lamina is not affected by matrix ply cracking. This is explained by the fact that thermal deformation is a stress-free state. Any change in thermal expansion behavior must come from a mechanical response of the crack. Since the degraded mechanical properties already account for the mechanical response of the crack during thermal deformation, the effective CTE of the cracked lamina remains unchanged. As a consequence, the remaining discussions are restricted to the effective mechanical properties of a cracked lamina.

3.2.2.3 Effect of Thickness of the Cracked Lamina

Fig. 40 shows the variation of normalized lamina modulus $E_{22(n)}$ with increasing crack density for three laminates; $[0/90]_S$, $[0/90_2]_S$, and $[0/90_3]_S$ where the thickness of the centered 90^o ply group is increased systematically. Note that these laminates were the experimental test cases investigated by Groves et al. (1987). From Fig. 40 it is clear that with increasing thickness of the inner 90^o plies, the modulus of the cracked lamina decreases for a given crack density. This is because the thicker ply groups result in longer cracks and larger zones of stress relief around the crack. Furthermore, the thickness of the adjacent 0^o plies is kept constant so the constraining effect on the cracks becomes relatively less with increased thickness of the cracked ply group. To further investigate the thickness effect a $[0/90_n]s$ laminate with cracks in the centered 90^o plies was modeled. The plies were made up of Silenka E-glass/epoxy laminae. The properties of different undamaged laminae with 0.15 mm thickness were obtained from Mayes and Hansen (2004) and are shown in Table 13.

Table 13. Thermo-elastic properties of undamaged composite laminae

Property	Silenka E-glass/Epoxy	T300/Epoxy	Highsmith Glass/Epoxy
E_{11} (GPa)	45.763	138	41.7
E_{22} , E_{33} (GPa)	16.167	11	13
υ_{12}, υ_{13}	0.2786	0.28	0.3
G_{12}, G_{13} (GPa)	5.8626	5.5	3.4
G_{23} (GPa)	5.7415	3.9286	4.6429
α_{11} ($\mu\,^{o}C^{-1}$)	8.6	-1.0	*n/a*
α_{22}, α_{33} ($\mu\,^{o}C^{-1}$)	26.15	26.0	*n/a*

A normalized crack density "ζ" (defined as the crack density multiplied by the crack length) was used to compare the effects of cracked layer thickness on property degradation. Two cross-ply laminates were analyzed here for the same normalized crack densities. Note that the cross-ply laminates with the highest and the lowest thicknesses previously studied were considered here.

Fig. 47 shows the variation of normalized lamina properties $E_{22(n)}$, $\upsilon_{21(n)}$, and $\upsilon_{23(n)}$ for two cross-ply laminates with cracked centered 90^o laminae. Fig. 48 and Fig. 49 show the variation of normalized lamina shear moduli $G_{12(n)}$ and $G_{23(n)}$ respectively.

As similarly reported by Tao and Sun (1996), from Fig. 47 through 49, it is evident that in terms of normalized crack density the behavior of a cracked lamina is not significantly influenced by the thickness. This is particularly true for $E_{22(n)}$, $\upsilon_{21(n)}$, $\upsilon_{23(n)}$, and $G_{12(n)}$ where the maximum difference was less than 5% for the two thicknesses.

3.2.2.4 Effect of Position of the Cracked Lamina

Tao and Sun (1996) analytically estimated the effective properties of a cracked lamina by varying the position of the cracked 90^o plies in a $[0_m / 90_n / 0_q]_T$ laminate. They reported that the position of the cracked laminae did not alter the estimated properties. However, the cracked lamina was never an outer ply in their study. Symmetric cracking was assumed so both the 90^o laminae in the laminate are cracked as depicted by Fig. 34.

It was found that the maximum difference in the predicted effective properties $E_{22(n)}$, $\upsilon_{21(n)}$, and $\upsilon_{23(n)}$ for the outer and centered plies was less than 5% . The difference was zero for the inplane shear modulus. For the out of plane shear modulus an exterior ply crack model was not developed.

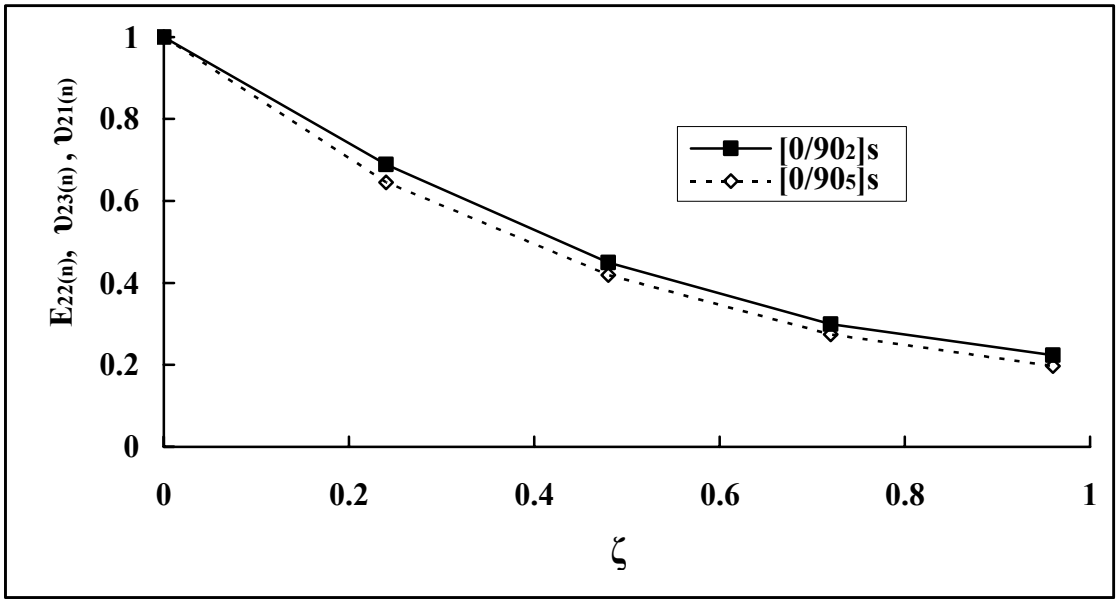


Figure 47. Variation of normalized lamina properties $E_{22(n)}$, $\upsilon_{23(n)}$, and $\upsilon_{21(n)}$ with normalized crack density for two cross-ply laminate configurations with cracks in centered 90^o laminae

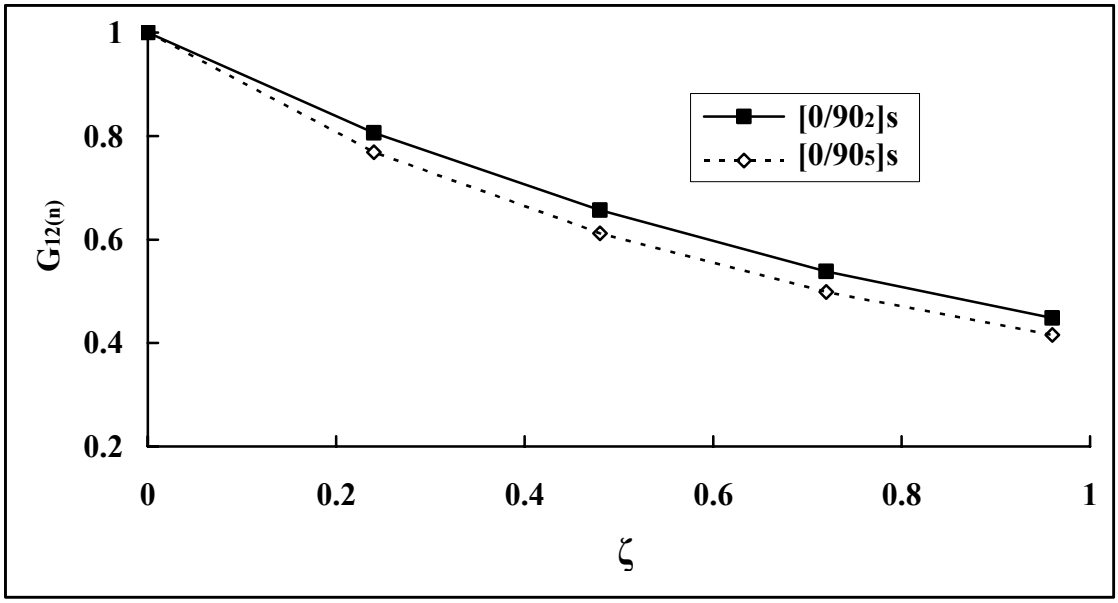


Figure 48. Variation of normalized lamina shear modulus $G_{12(n)}$ with normalized crack density for two cross-ply laminate configurations with cracks in centered 90^o laminae

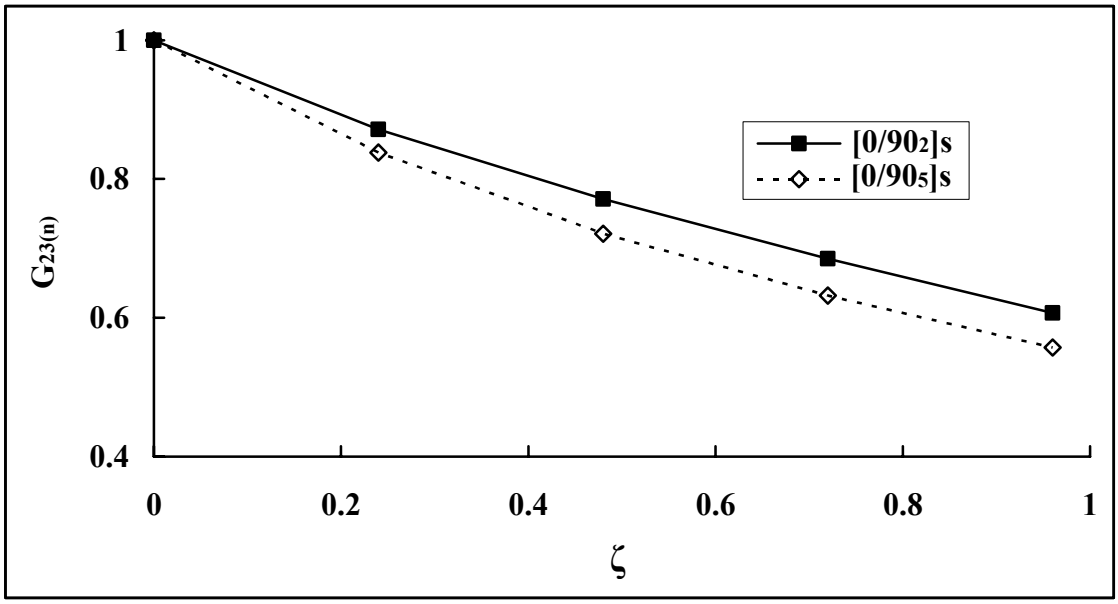


Figure 49. Variation of normalized lamina shear modulus $G_{23(n)}$ with normalized crack density for two cross-ply laminate configurations with cracks in centered 90^o laminae

3.2.2.5 Effect of Adjacent Ply Properties

The effects of thickness, fiber orientation, and mechanical properties of the adjacent plies on the effective mechanical properties of a cracked lamina were all similar. In each of these cases the changing stiffness of the adjacent plies was found to have a small influence on the effective properties of the cracked plies.

For example, Fig. 50 shows the variation of the normalized laminae properties $E_{22(n)}$, $\upsilon_{21(n)}$, and $\upsilon_{23(n)}$ with increasing fiber orientation of the adjacent plies. Fig. 51 and 52 show the variation of the normalized lamina shear modulus $G_{12(n)}$ and normalized out-of-plane shear modulus $G_{23(n)}$ with increasing fiber orientation of the adjacent plies respectively. These results indicate what turned out to be a universal trend where adjacent plies that contribute more constraint (resistance to crack opening deformation) result in slightly less effective degradation of stiffness of the cracked plies. Note that for the inplane shear properties it is adjacent ± 45 plies that provide the most constraint against shear deformation.

The general conclusion was that for a particular deformation mode (e.g. extension or shear), if the uncracked ply properties (e.g. orientation or modulus) are increased to be more resistant to that deformation mode then the corresponding effective property of the cracked ply group will increase. This is understood by observing that the softening of the cracked ply group is the result of the added deformation around the crack. Stiffer adjacent plies tend to resist and minimize this added deformation through a constraining effect and thus increase the effective properties of the cracked ply group.

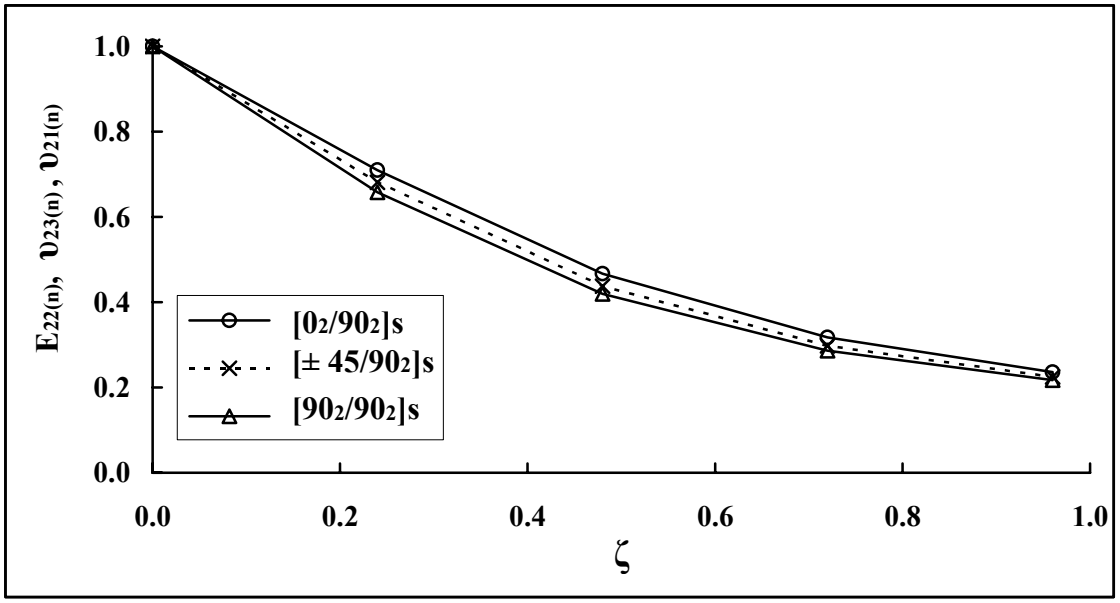


Figure 50. Variation of normalized lamina properties $E_{22(n)}$, $\upsilon_{23(n)}$, and $\upsilon_{21(n)}$ of cracked centered laminae for various fiber orientations of the adjacent plies

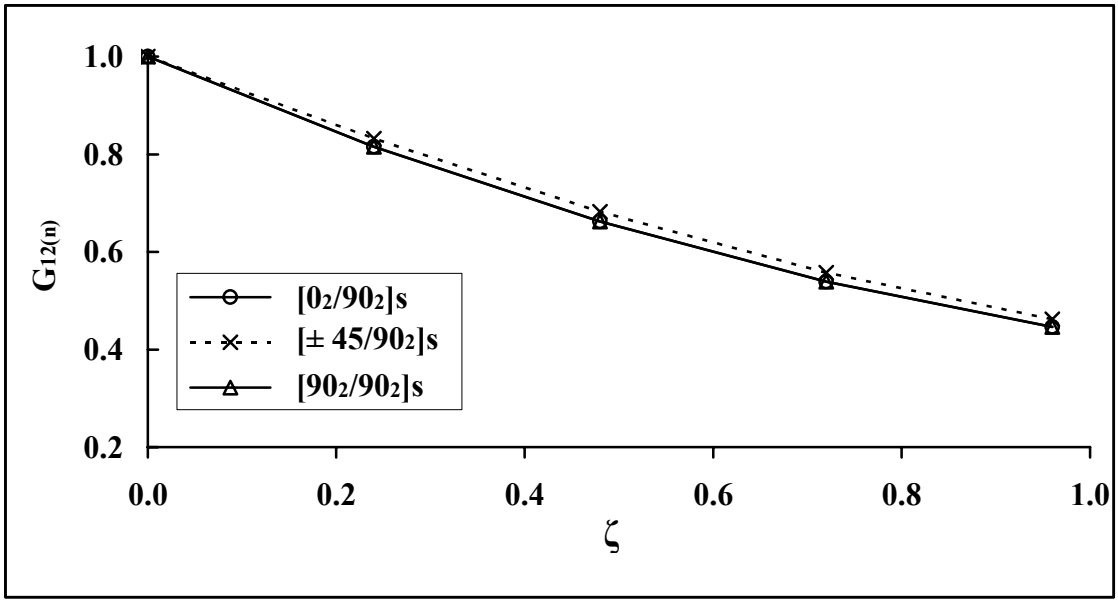


Figure 51. Variation of normalized lamina shear modulus $G_{12(n)}$ of cracked centered laminae for various fiber orientations of the adjacent plies

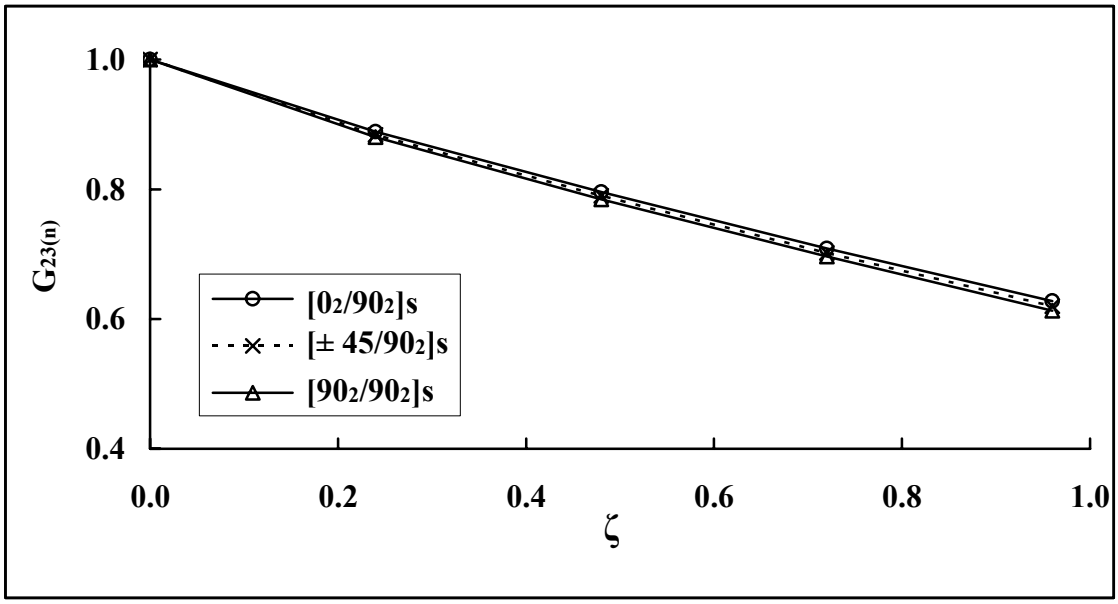


Figure 52. Variation of normalized lamina shear modulus $G_{23(n)}$ of cracked centered laminae for various fiber orientations of the adjacent plies

4.0 MULTI-SCALE DAMAGE MODELING OF COMPOSITES

In the analysis of composite structures using numerical methods such as the finite element method (FEM), a composite structure is viewed as a homogenized material with anisotropic properties. The analysis at this level is termed *macroscale analysis*. Even though a composite laminate is assumed to be a homogeneous material at the macroscale, a laminate is made up of laminae stacked at different orientations. Thus, the laminate properties are influenced by the properties of the laminae and their orientations. In view of this, analysis of individual laminae is also considered in the analysis of composite structures and the analysis at the lamina level is referred to here as *mesoscale analysis*. However, uni-directional composite laminae consist of reinforcing fibers in a continuous matrix and are characterized by a fiber volume fraction, the microstructure, and the properties of the fiber and matrix constituents. Thus, the properties of a composite structure depend on the constituent (fiber and matrix) properties. Based on this, analysis at the constituent level, referred to here as *microscale analysis*, is sometimes utilized to correlate constituent level material behavior to structural level analysis.

Due to the fact that a composite structure is made up of fiber and matrix, damage in a composite structure is the result of damage of the constituents. However, in most damage analyses, only the macroscale and the mesoscale are considered. That is, damage in laminated composite structures made up of unidirectional laminae is typically defined, measured, and analyzed as a reduction of thermo-mechanical properties of the composite laminae. This is further evident from the review of degradation models discussed earlier. The underlying assumption for defining damage at the mesoscale level is that a composite lamina is viewed as a homogenized material. However, the homogenized properties of a composite lamina depend on the constituent properties. Therefore, it is logical to represent damage in a composite structure as a damage of the constituent(s) for a more fundamental representation of damage in a composite structure.

Here, utilizing an extraction process, the effective damaged constituent properties that represent the damaged lamina properties were reverse engineered utilizing a finite element micromechanics model and Multicontinuum Technology (MCT). This extraction process requires damaged composite lamina properties and some of the constituent properties. With the effective properties of the constituents available, MCT can then be used to analyze damage in a composite structure in terms of damage of the constituents. The fundamentals of MCT are well documented in the literature [Garnich and Hansen (1997a,b)] and are not repeated here. The essential hypothesis of the technology is that each constituent in a composite can be represented with a separate superposed continuum field that is rigorously tied to that of the composite.

MCT requires composite properties obtained from experiments that are related to be consistent with those of a micromechanics model. The rectangular finite element micromechanics model described for hexagonal fiber packing was utilized here to supplement MCT with the lamina properties.

4.1Effective Constituent Properties using MCT and Micromechanics

A procedure to obtain the constituent properties utilizing the composite properties, MCT, and a finite element micromechanics model is described. The only other requirement for this process

is that the properties of one of the constituents are known. The basic iterative procedure is depicted in Fig. 53. The "**A** matrix" referred to in Fig. 53 is the reduced form of a fourth order tensor that relates the constituent strains [Garnich and Hansen (1997a)].

Using this method, a degraded lamina with cracks can be described as a continuum with a combination of reduced and undamaged constituent properties. The method was implemented for two possible matrix damage modes; matrix normal cracking (MNC) and matrix ply cracking (MPC). With this extraction process, the gradual degradation of effective properties for the two failure modes expressed as a reduction of lamina properties are described as a reduction of constituent properties.

The degraded properties of a lamina with matrix normal cracks were previously estimated using a micromechanical model. The laminae analyzed were AS4/3501-6 with 0.45 and 0.66 fiber volume fractions The transverse properties E_{22}, υ_{23}, and G_{23} remained unchanged with increasing crack density. Using the extraction procedure the effective matrix properties that contribute to the reduction of the remaining thermo-elastic properties of the lamina due to the matrix normal cracks were estimated. A lamina damaged by MNC retains its plane of isotropy with intact fibers. The fiber properties are assumed to remain unchanged and satisfactory results were achieved.

For damage due to matrix ply cracks it was found that even though the cracks damage only the matrix it was necessary to degrade both the fiber and matrix properties in order to achieve lamina properties matching those predicted by the micromechanics. A $[0/90_8/0]_T$ cross-ply laminate made up of E-glass/MY750 composite laminae with single lamina thickness of 0.15 mm was analyzed with matrix cracks in the inner 90^o plies. The properties of the composite lamina and the properties of the Silenka E-glass fibers and the MY750/HY917/DY063 epoxy matrix that constitute the composite lamina (with 0.6 fiber volume fraction) are given in Table 14. Fig. 54 shows the variation of degraded lamina properties (normalized with their respective un-damaged values) with increasing crack density "λ" (cracks per mm). Note that the *normalized* properties E_{22}, υ_{23}, and υ_{21} degrade by the same percentage and are shown with a single curve.

Table 14. Thermo-elastic Properties of the Constituents and the Lamina

Property	E-Glass	Epoxy	E-Glass/Epoxy
E_{11} (GPa)	73	4.85	45.76
E_{22} , E_{33} (GPa)	73	4.85	16.167
υ_{12}, υ_{13}	0.235	0.36	0.2786
G_{12}, G_{13} (GPa)	29.55	1.78	5.86
υ_{23}	0.235	0.36	5.74
α_{11} ($\mu\,^oC^{-1}$)	6.6	49	8.6
α_{22}, α_{33} ($\mu\,^oC^{-1}$)	6.6	49	26.15

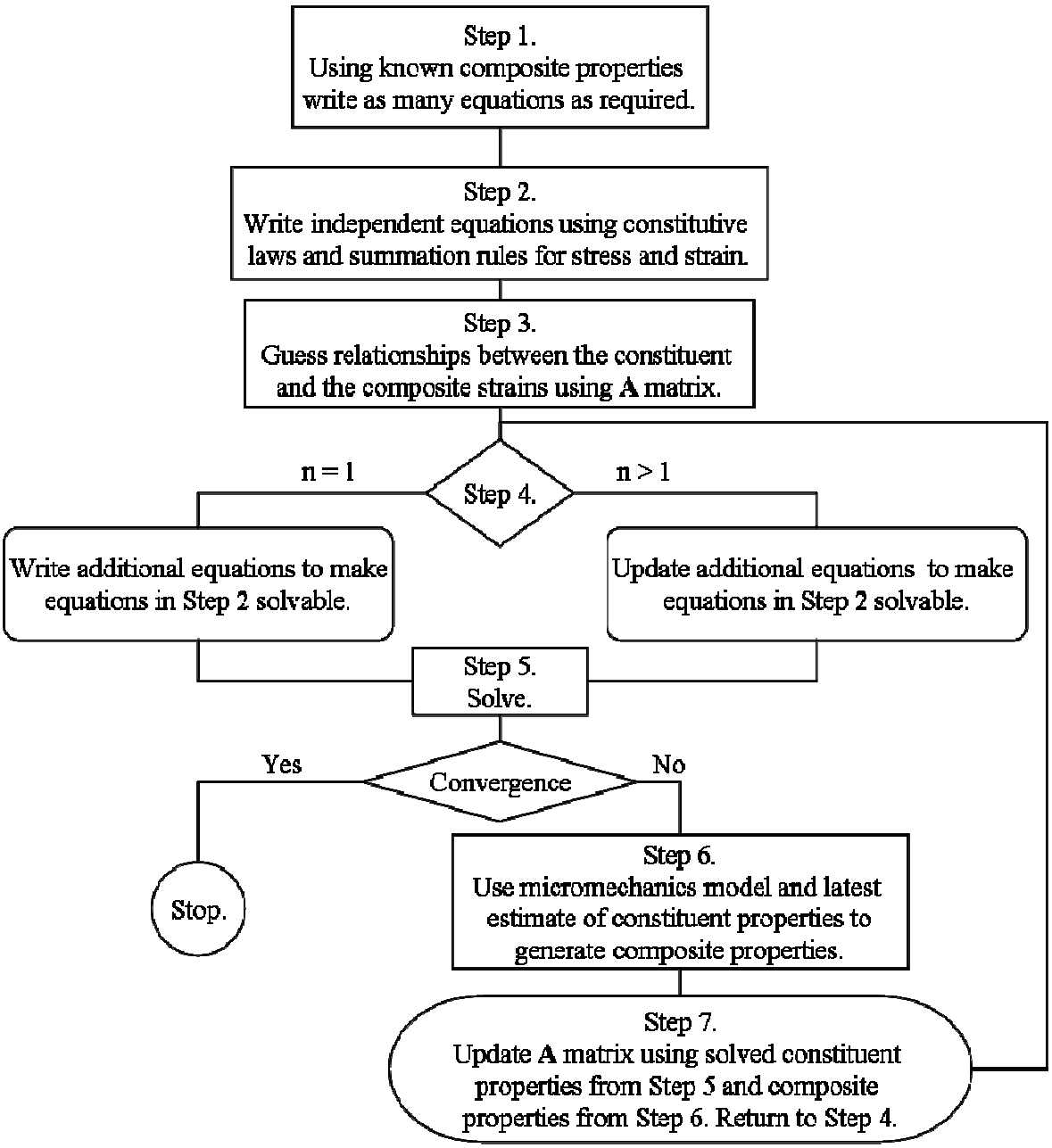


Figure 53. Flowchart of the extracting process

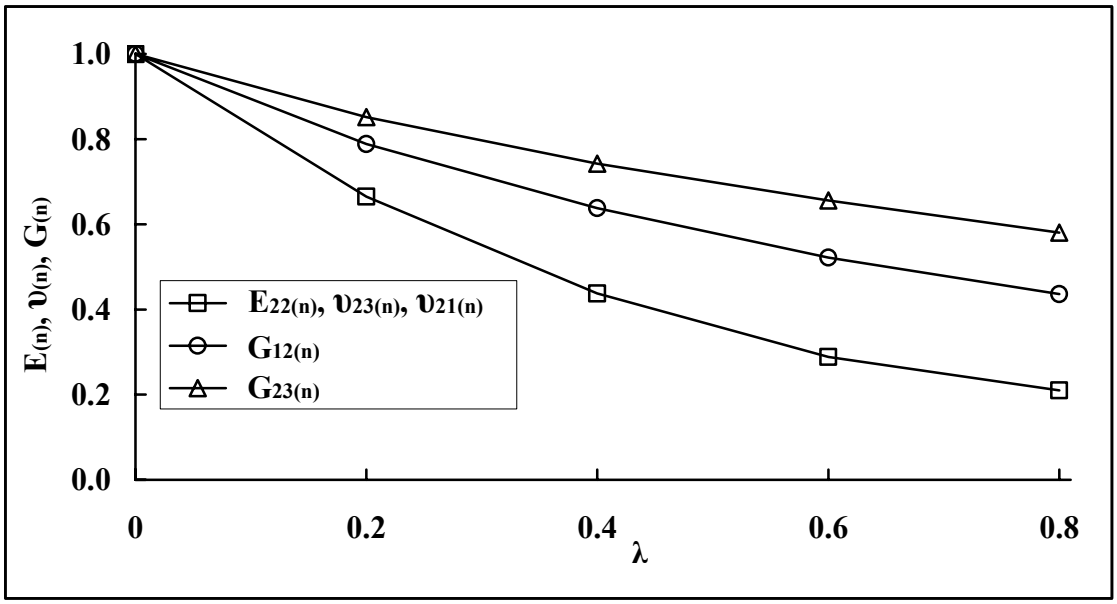


Figure 54. Variation of normalized lamina properties with crack density for a lamina with matrix ply cracks

The effective properties of the constituents that coincide with the reduced lamina properties with MNC were estimated are shown in Fig. 55. The validity of this observation is realized by considering that the average stress in the fiber must decrease while the average strain is increasing due to damage. The only way this can happen is if the fiber stiffness is reduced along with the matrix stiffness.

To estimate the effective constituent properties, effective constituent strains that correspond to the effective strains of the composite (including the crack) were required along with the volume-averaged stresses. The effective strains of the constituents are unknown along with some of the properties of the constituents. Therefore, additional conditions were required to solve for the constituent strains which can then be used to estimate the unknown constituent properties. To realize a solvable system it was assumed that the ratio of the constituent strains remains the same before and after the formation of cracks.

It was also assumed that properties parallel to a crack are not affected by the crack. Therefore, E_{22}, υ_{23}, υ_{21}, G_{12}, and G_{23} were assumed to change for both the fibers and the matrix and the resulting properties were calculated. Fig. 55 shows the variation of fiber (indicated by "F") and matrix (indicated by "M") normalized properties with increasing crack density. The normalized values of G_{12} and G_{23} were found to be the same for the fiber, matrix, and the cracked lamina. This is explained by the fact that the relationship between the lamina shear moduli and constituent shear moduli is linear and independent of other properties.

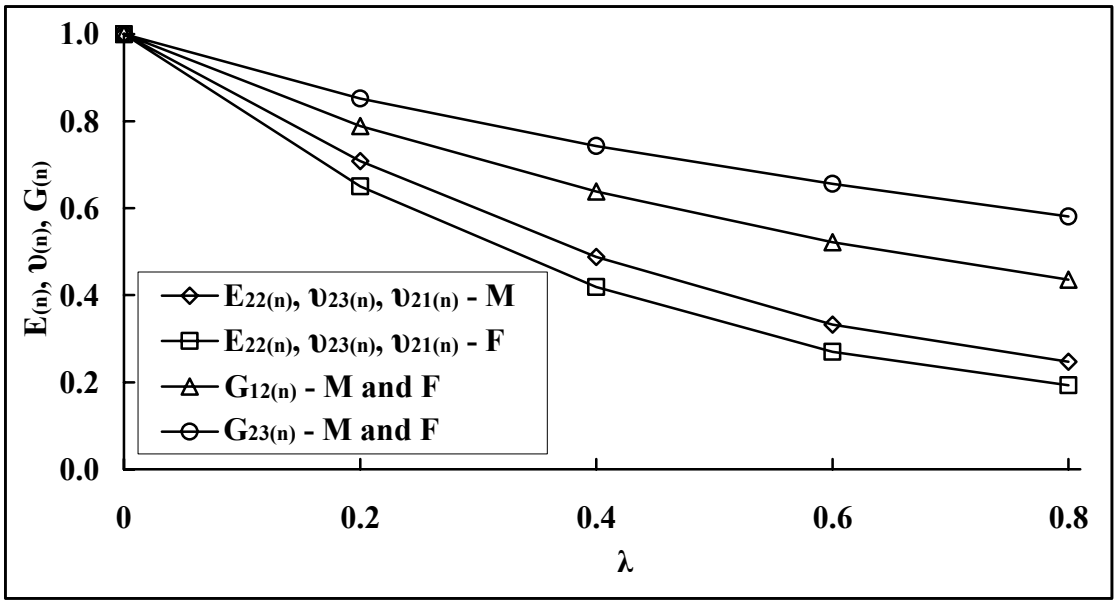


Figure 55. Variation of effective matrix and fiber properties with crack density for a lamina with matrix ply cracks

4.2 Discussion of Effective Constituent Property Modeling

The process utilized to extract effective constituent properties that relate damage in the lamina as damage in the constituents was described. Thereafter, the process was implemented for two different possible matrix failure modes, namely, matrix normal cracks and matrix ply cracks. For the matrix normal cracking damage mode, damage in the lamina was described as damage of the matrix with the fiber properties unchanged. Furthermore, based on the characteristics of the damage mode, it was assumed that the damaged matrix is transversely isotropic. The reduced properties of the matrix material coupled with the intact fiber properties successfully describe the damage state of the cracked laminae with MNC.

The second failure/damage mode analyzed was matrix ply cracking. In contrast to MNC, for this damage mode properties of both the fibers and the matrix were degraded to represent the damage in the lamina. To solve for the effective constituent properties it was necessary to assume that the ratio of fiber and the matrix strains remained the same before and after damage. This is a reasonable assumption since the fibers remain embedded in the matrix. With this assumption it was possible to relate the damage in the lamina due to matrix ply cracks as damage of the constituents.

Using the constituent property extraction process, the damaged constituents can be replaced by continuum constituents but with degraded properties. Furthermore, with the effective constituent properties that represent the damage in a lamina available, Multi-continuum technology can be used to estimate the degraded continuum response of the composite and the constituents in a structural analysis. This provides a means to incorporate constituent level damage into a structural level damage analysis thus bridging the mesoscale and microscale damage levels.

REFERENCES

Akula VMK, (2007), Constitutive Modeling of Damaged Unidirectional Composite Laminae, PhD Dissertation, University of Wyoming.

Akula VMK and Garnich MR, (2007a), Review of Degradation Models for Progressive Failure Analysis of FRP Composites: Part I – Sudden Degradation Models, submitted to Compos Sci Technol.

Akula VMK and Garnich MR, (2007b), Review of Degradation Models for Progressive Failure Analysis of FRP Composites: Part II – Gradual Degradation Models, submitted to Compos Sci Technol.

Akula VMK and Garnich MR, (2007c), Extracting the Effective Constituent Properties of Damaged Uni-directional Composite Lamina, presented at SAMPE 2007/Baltimore, Baltimore, MD, published in Society for Advancement of Material & Process's 2007 Proceedings CD-ROM.

Bader MG, Bailey JE, Curtis PT and Parvizi A, (1979), The Mechanisms of Initiation and Development of Damage in Multi-axial Fibre-Reinforced Plastics Laminates, Proceedings of the Third International Conference on Mechanics of Materials, Volume 3, Cambridge, England, August, pp. 227-239.

Bailey JE, Curtis PT and Parvizi A, (1979), On the Transverse Cracking and Longitudinal Splitting of Glass and Carbon Fibre Reinforced Epoxy Cross Ply Laminates and the Effect of Poisson and Thermally Generated Strain, P Roy Soc Lond A Mat;366(1727):599-623.

Bailey JE and Parvizi A, (1981), On Fibre Debonding Effects and the Mechanism of Transverse-Ply Failure in Cross-ply Laminates of Glass Fibre/Thermoset Composites, J Mater Sci;16(3):649-659.

Bansemir H and Haider O, (1998), Fibre Composite Structures for Space Applications – recent and future developments, Cryogenics;38(1):51-59

Candan S, Garcelon J, Balabanov V and Venter G, (2000), Shape Optimization using ABAQUS and VisualDOC, AIAA publications, AIAA-2000-4769.

Dodson KJ and Rule JE, (1989), Thermal Stability Considerations for Space Flight Optical Benches, 34th Int. SAMPE Symposium, pp1578-1589

Farmer JT, Wahls DM, and Wright RL, (1992), Thermal Distortion Analysis of an Antenna-Support Truss in Geosynchronous Orbit, J Spacecraft and Rockets, 29(3), 386-393.

Fertig RS and Garnich MR, (2004), Influence of Constituent Properties and Microstructural Parameters on the Tensile Modulus of a Polymer/Clay Nanocomposite, Compos Sci Technol;64(16):2577-2588.

Garnich MR and Hansen AC, (1997a), A Multicontinuum Approach to Structural Analysis of Linear Viscoelastic Composite Materials, J Applied Mechanics;64(4):795-803.

Garnich MR and Hansen AC, (1997b), A Multicontinuum Theory for Thermal-Elastic Structural Analysis of Composites, J Compos Mater;31(1):71-86.

Garnich MR and Karami G, (2004), Finite Element Micromechanics for Stiffness and Strength of Wavy Fiber Composites, J Compos Mater;38(4):273-292.

Garrett KW and Bailey JE, (1977a), Multiple Transverse Fracture in 90^{o} Cross-ply Laminates of a Glass Fibre-Reinforced Polyester, J Mater Sci;12(1):157-168.

Garrett KW and Bailey JE, (1977b), The Effect of Resin Failure Strain on the Tensile Properties of Glass Fibre-Reinforced Polyester Cross-ply Laminates, J Mater Sci;12(11):2189-2194.

Grimaldi F, Tempesta G, Pastorelli F, and Pesciarelli S, (1989), Development of Dimensionally Stable Lightweight Composite Satellite Antenna Structures, 34th Int. SAMPE Symposium, pp 1590-1602.

Groves SE, Harris CE, Highsmith AL, Allen DH and Norvell RG, (1987), An Experimental and Analytical Treatment of Matrix Cracking in Cross-ply Laminates, Exper Mech;27(1):73-79.

Highsmith AL and Reifsnider KL, (1982), Stiffness-Reduction Mechanisms in Composite Laminates," Damage in Composite Materials. ASTM STP 775, K. L. Reifsnider, Ed., American Society for Testing and Materials, pp. 103-117.

Hyer MW, (1998), Stress Analysis of Fiber-Reinforced Composite Materials. New York: McGraw-Hill.

Ishikawa T, Fukunaga H and Ono K-I, (1989), Graphite-Epoxy Laminates with Almost Null Coefficient of Thermal Expansion Under a Wide Range of Temperature, J Materials Science 24, 2011-2017.

Joffe R and Varna J, (1999), Analytical Modeling of Stiffness Reduction in Symmetric and Balanced Laminates Due to Cracks in 90^{o} Layers. Compos Sci Technol;59(11):1641-1652.

Karami G and Garnich MR, (2005a), Effective Moduli and Failure Considerations for Composites with Periodic Fiber Waviness, Compos Struct;67(4):461-475.

Karami G and Garnich MR, (2005b), Micromechanical Study of Thermoelastic Behavior of Composites with Periodic Fiber Waviness, Composites Part B: Engineering;36(3):241-248.

Kim RY, Crasto AS, and Schoeppner GA, (2000), Dimensional Stability of Composite in a Space Thermal Environment, Compos Sci Technol;60(12):2601-2608.

Lee JH and Hong CS, (1993), Refined Two-dimensional Analysis of Cross-ply Laminates with Transverse Cracks Based on the Assumed Crack Opening Deformation, Compos Sci Technol;46(2):157-166.

Leiva JP and Watson BC, (1998), Automatic Generation on Basic Vectors for Shape Optimization in the Genesis Program, AIAA publications, AIAA-98-4852

Maji A, Kozola B and Griffin S, (2001), Design of Composite Surrogate Mirror Support Structure, J. Aerospace Engineering, pp 112-118

Matthews FL, Davies GAO, Hitchings D and Soutis C, (2000), Finite Element Modeling of Composite Materials and Structures, Woodhead Publishing.

Mayes JS, Hansen AC, (2004), Composite Laminate Failure Analysis Using Multicontinuum Theory. Compos Sci Technol;64(3-4):379-394.

Nemat-Nasser S, Hori M, (1993), Micromechanics: Overall Properties of Heterogeneous Materials, Elsevier, Asterdam.

Parvizi A and Bailey JE, (1978), On Multiple Transverse Cracking in Glass Fibre Epoxy Cross-ply Laminates, J Mater Sci;13(10):2131-2136.

Parvizi A, Garrett KW and Bailey JE, (1978), Constrained Cracking in Glass Fibre-Reinforced Epoxy Cross-ply Laminates, J Mater Sci;13(1):195-201.

Pradhan B, Venu Kumar N and Rao NS, (1999b), Stiffness Degradation Resulting From 90^{o} Ply Cracking in Angle-ply Composite Laminates, Compos Sci Technol;59(10):1543-1552.

Prunty J, (1978), Dimensionally Stable Graphite Composites For Spacecraft Structures, SAMPE Quarterly, pp41-51.

Shin K-B, Kim C-G, Hong C-S, and Lee H-H, (2001), Thermal Distortion Analysis of Orbiting Solar Array Including Degradation Effects of Composite Materials, Composites Part B;32(4):271-285.

Tao JX and Sun CT, (1996), Effect of Matrix Cracking on Stiffness of Composite Laminates, Mech Compos Mater Struct;3(3):225-239.

Vanderplaats Research & Development, Inc, (2004), VisualDOC Manual, Version 4.

Whitcomb JD, Chapman CD and Tang X, (2000), Derivation of Boundary Conditions for Micromechanics Analyses of Plain and Satin Weave Composites, J Compos Mater;34(9):724-747.

Yoon KJ and Kim J-S, (2001), Effect of Thermal Deformation and Chemical Shrinkage on the Process Induced Distortion of Carbon/Epoxy Curved Laminates, J Compos Mater;35(3):253-263.

APPENDIX A

PYTHON CODE FOR MNC MICROMECHANICS MODEL

A computer code written in *Python* for use in ABAQUS Version 6.6 is listed below. The code consists of three python script files, namely, "*run_mnc.py*", "*model_mnc.py*" and "*bc_mnc.py*". The entire script is run by executing the statement "**execfile('run_mnc.py')**" in ABAQUS command window. This runs the script file "*run_mnc.py*" which executes the other two files "*model_mnc.py*" and "*bc_mnc.py*" in sequence.

The file "*run_mnc.py*" is the central file that controls the creation and execution of the model. The file has the parameters such as, the material definition, volume fraction, crack density or crack spacing, etc. that are required to define the unit cell model that simulates a point in a lamina with matrix normal cracks. With the required parameters defined, the file "*model_mnc.py*" is executed which creates the micromechanics model in ABAQUS and produces an input file. This input file has all node and element information defining the model. The file "*bc_mnc.py*" then modifies the input file by adding the periodic boundary conditions or kinematic constraints for the nodes in the models. This requires identifying the boundary nodes on the model. The model is then run in ABAQUS using the input file by "*run_mnc.py*".

Central program: ***run_mnc.py***

```
global zeta, ped, vf1, sq3, sq2, pi, a, b, c, d, seed_1
global theta, vf, r_fiber, RR, f_mat, m_mat, case

## crack density of the lamina
## zeta = 1/crack spacing
zeta = 0.2
## depth of the model.
ped = 1.0/zeta
# vf = fiber volume fraction
vf1 = 0.66
## constants
sq3 = sqrt(3.0)
sq2 = sqrt(2.0)
pi = acos(-1.0)
##
# a = width of the rectangular model
a = 1.0
# c = half the width of the model
c = a/2.0
# b = length of the model
b = sq3 * a
# d = half the width
d = b/2.0
# the volume fraction is approximated using a polygon so
# the vf is changed
seed_1 = 8 ## number of elements on the radius.
theta = pi/(3.0*seed_1)
vf = pi*vf1/(3.0*sin(theta)*seed_1)
```

```
##
## r_fiber = radius of the fiber for the given volume fraction
r_fiber = round(sqrt(a*a*vf*sq3 /(2.0*pi)), 12)
RR = r_fiber /sq3

## fiber and matrix material indications
## these numbers are material numbers minus one.
## for example material 1 is indicated by zero and so on.
f_mat = 2
m_mat = 3
## indicate whether the material has suffered matrix failure
matrix_failure = 1
## Note 1 - yes, 0 - no
## the temperature range for thermal loading
deltat = -200

##-----------------------------------------------------
## material database
mat_name = [0]

## material - 1
NAME = 'Boron'
temp = [1, NAME]
E11, E22, E33 =  399.896E9, 399.896E9, 399.896E9
G12, G13, G23 =  166.6233E9, 166.6233E9, 166.6233E9
NU12, NU13, NU23 = 0.2, 0.2, 0.2
A11, A22, A33 = 16.2E-6, 16.2E-6, 16.2E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## material - 2
NAME = 'Aluminum'
temp = [2, NAME]
E11, E22, E33 =  72.36737E9, 72.36737E9, 72.36737E9
G12, G13, G23 =  27.20577E9, 27.20577E9, 27.20577E9
NU12, NU13, NU23 = 0.33, 0.33, 0.33
A11, A22, A33 = 23.4E-6, 23.4E-6, 23.4E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## material - 3
NAME = 'AS4'
temp = [3, NAME]
E11, E22, E33 =  201E9, 13.5E9, 13.5E9
G12, G13, G23 =  95E9, 95E9, 4.9E9
NU12, NU13, NU23 = 0.22, 0.22, 0.3775
A11, A22, A33 = -0.7E-6, 8.0E-6, 8.0E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]
## if you need to add more materials then add before this line.

## material - 4
NAME = 'M3501-6'
temp = [4, NAME]
E11, E22, E33 =  4.3E9, 4.3E9, 4.3E9
G12, G13, G23 =  1.5926E9, 1.5926E9, 1.5926E9
NU12, NU13, NU23 = 0.35, 0.35, 0.35
```

```
A11, A22, A33 = 55.0E-6, 55.0E-6, 55.0E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## if you need to add more materials then add before this line.

## end of database
del mat_name[mat_name.index(0)]

from abaqus import *
from abaqusConstants import *
from section import*
import part
import regionToolset
import displayGroupMdbToolset as dgm
import material
import section
import sketch
import assembly
import step
import interaction
import load
import mesh
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import job
import math

## creating the model based on the crack density.
GO = 0
#### building the model
## note that the path of the file needs to be specified.
exec open('c:/model_mnc.py')
#### writing the boundary conditions
exec open('c:/bc_mnc.py')

global case
GO = 1
num = 0
if(GO == 1):
    for num in range(7):
        case = num + 1
        if (case == 1):
            print '                '
            print 'Runnning the File - Case 1 E11'
            hexmodel = mdb.JobFromInputFile(name = 'Run_case_1',
inputFileName = 'Case_1.inp')
            hexmodel.submit()
            hexmodel.waitForCompletion()
            AA = session.openOdb(name='c:/Run_case_1.odb')
            ## the path needs to be changed according to the directory in
which the program is run
            import visualization
            session.viewports['Viewport: 1'].setValues(displayedObject=AA)
            session.viewports['Viewport: 1'].odbDisplay.setDeformedVariable(
variableLabel='U')
```

```
            session.viewports['Viewport: 1'].odbDisplay.setPrimaryVariable(
                variableLabel='S', outputPosition=INTEGRATION_POINT,
refinement=(COMPONENT, 'S11'))

        elif (case == 2):
            print '               '
            print 'Runnning the File - Case 2 E22'
            hexmodel = mdb.JobFromInputFile(name = 'Run_case_2',
inputFileName = 'Case_2.inp')
            hexmodel.submit()
            hexmodel.waitForCompletion()
            print 'Case - 2 : Transverse direction loading'
            AA = session.openOdb(name='c:/Run_case_2.odb')

        elif (case == 3):
            print '               '
            print 'Runnning the File - Case 3 G12'
            hexmodel = mdb.JobFromInputFile(name = 'Run_case_3',
inputFileName = 'Case_3.inp')
            hexmodel.submit()
            hexmodel.waitForCompletion()
            print 'Case - 3 : Inplane Shear loading'
            AA = session.openOdb(name='c:/Run_case_3.odb')

        elif (case == 4):
            print '               '
            print 'Runnning the File - Case 4 G23'
            hexmodel = mdb.JobFromInputFile(name = 'Run_case_4',
inputFileName = 'Case_4.inp')
            hexmodel.submit()
            hexmodel.waitForCompletion()
            print 'Case - 4 : out-of-plane 13 Shear loading'
            AA = session.openOdb(name='c:/Run_case_4.odb')

        elif (case == 5):
            print '               '
            print 'Runnning the File - Case 5 Thermal'
            hexmodel = mdb.JobFromInputFile(name = 'Run_case_5',
inputFileName = 'Case_5.inp')
            hexmodel.submit()
            hexmodel.waitForCompletion()
            print 'Case - 5 : Temperature loading'
            AA = session.openOdb(name='c:/Run_case_5.odb')

        elif (case == 6):
            print '               '
            print 'Runnning the File - Case 6 G13'
            hexmodel = mdb.JobFromInputFile(name = 'Run_case_6',
inputFileName = 'Case_6.inp')
            hexmodel.submit()
            hexmodel.waitForCompletion()
            print 'Case - 6 : G13 loading'
            AA = session.openOdb(name='c:/Run_case_6.odb')

        elif (case == 7):
            print '               '
            print 'Runnning the File - Case 7 E33'
```

```
            hexmodel = mdb.JobFromInputFile(name = 'Run_case_7',
inputFileName = 'Case_7.inp')
            hexmodel.submit()
            hexmodel.waitForCompletion()
            import visualization
            print 'Case - 7 : E33 loading'
            AA = session.openOdb(name='c:/Run_case_7.odb')

else:
    print '         '
    print ' Only Modeling '
```

Code for creating the model in ABAQUS: ***model_mnc.py***

```
##-----------------------------------------------------------------------------
hexmodel =mdb.Model(name='Hex Pack Model')
def hexpack_part():
# coordinates of the vertices
    rectcd = ((a/2.0,b/2.0), (-a/2.0,b/2.0), (-a/2.0,-b/2.0), (a/2.0,-b/2.0),
(0.0,0.0))
    s = hexmodel.Sketch(name='rect', sheetSize = 200.0)
# creating the rectangle.
    s.rectangle(point1=rectcd[1], point2=rectcd[3])
    hexpart = hexmodel.Part(name='recta', dimensionality=THREE_D,
      type=DEFORMABLE_BODY)
    hexpart.BaseSolidExtrude(sketch=s, depth=ped)

##-----------------------------------------------------------------
def hexpack_pit():

    p2 = hexmodel.parts['recta']
    fa = p2.faces
    c = p2.cells
    e = p2.edges
    face_2 = fa.findAt( (0.0,0.0,0.0) )
    sketch_2 = hexmodel.Sketch(name='sketch 2', sheetSize=200.0)
    sketch_2.CircleByCenterPerimeter((0.0,0.0), (r_fiber,0))
    faceset_1 = (face_2)
    p2.PartitionFaceBySketch(faces=faceset_1, sketch=sketch_2)

    face_3 = fa.findAt( (-a/2.0+r_fiber/2.0, b/2.0-r_fiber*sq3/2.0,0) )
    sketch_3 = hexmodel.Sketch(name='sketch 3', sheetSize=200.0)
    sketch_3.CircleByCenterPerimeter((-a/2.0,b/2.0), (-a/2.0+r_fiber,b/2.0))
    faceset_2 = (face_3)
    p2.PartitionFaceBySketch(faces=faceset_2, sketch=sketch_3)

    face_4 = fa.findAt((a/2.0-r_fiber/2.0,b/2.0-r_fiber*sq3/2.0,0))
    sketch_4 = hexmodel.Sketch(name='sketch 4', sheetSize=200.0)
    sketch_4.CircleByCenterPerimeter((a/2.0,b/2.0), (a/2.0,b/2.0-r_fiber))
    faceset_3 = (face_4)
    p2.PartitionFaceBySketch(faces=faceset_3, sketch=sketch_4)

    face_5 = fa.findAt((a/2.0-r_fiber/2.0,-b/2.0+r_fiber*sq3/2.0,0))
    sketch_5 = hexmodel.Sketch(name='sketch 5', sheetSize=200.0)
    sketch_5.CircleByCenterPerimeter((a/2.0,-b/2.0), (a/2.0,-b/2.0+r_fiber))
    faceset_4 = (face_5)
```

```
    p2.PartitionFaceBySketch(faces=faceset_4, sketch=sketch_5)

    face_6 = fa.findAt((-a/2.0+r_fiber/2.0,-b/2.0+r_fiber*sq3/2.0,0))
    sketch_6 = hexmodel.Sketch(name='sketch 6', sheetSize=200.0)
    sketch_6.CircleByCenterPerimeter((-a/2.0,-b/2.0), (-a/2+r_fiber,-b/2.0))
    faceset_5 = (face_6)
    p2.PartitionFaceBySketch(faces=faceset_5, sketch=sketch_6)

    cell_2 = c.findAt((a/4.0+r_fiber/2.0, 0, 0))
    edge_1 = e.findAt((-a/2.0, 0, 0))
    edge_2 = e.findAt((-a/2.0+r_fiber/2.0, b/2.0 -r_fiber*sq3/2.0,0))
    edge_3 = e.findAt((0,b/2.0,0))
    edge_4 = e.findAt((a/2.0-r_fiber/2.0, b/2.0-r_fiber*sq3/2.0,0))
    edge_5 = e.findAt((a/2.0,0,0))
    edge_6 = e.findAt((a/2.0-r_fiber/2.0, -b/2.0+r_fiber*sq3/2.0,0))
    edge_7 = e.findAt((0,-b/2.0,0))
    edge_8 = e.findAt((-a/2.0+r_fiber/2.0, -b/2.0+r_fiber*sq3/2.0,0))
    edge_sweep_1 = e.findAt((-a/2.0,-b/2.0,ped/2.0))
    edgeset_1 = (edge_1, edge_2, edge_3, edge_4, edge_5,
                 edge_6, edge_7, edge_8)
    cellset_2 = (cell_2)
    p2.PartitionCellBySweepEdge(cells=cellset_2, edges=edgeset_1,
sweepPath=edge_sweep_1)

    cell_1 = c.findAt((0.0,0.0,0.0))
    cellset_1 = (cell_1)
    edge_9 = e.findAt((r_fiber,0.0,0.0))
    edgeset_2 = (edge_9)
    edge_sweep_2 = e.findAt((-a/2.0,-b/2.0,ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cellset_1, edges=edgeset_2,
      line=edge_sweep_2, sense=FORWARD)

    p = mdb.models['Hex Pack Model'].parts['recta']
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
#-----------------------------------------------------------------------
def hexpack_section():
    p2 = hexmodel.parts['recta']
    cnew = p2.cells
    hexmodel.HomogeneousSolidSection(name='fiber section',
material=mat_name[f_mat][1])
    hexmodel.HomogeneousSolidSection(name='matrix section',
material=mat_name[m_mat][1])
    c_1 = cnew.findAt((a/2.0,b/2.0,ped/3.0))
    c_2 = cnew.findAt((-a/2.0,b/2.0,ped/3.0))
    c_3 = cnew.findAt((-a/2.0,-b/2.0,ped/3.0))
    c_4 = cnew.findAt((a/2.0,-b/2.0,ped/3.0))
    c_5 = cnew.findAt((0,0,0))
    c_6 = cnew.findAt((a/4.0 + r_fiber/2.0, 0, ped/3.0))

    c_cells = cnew.findAt(((a/2.0,b/2.0,ped/3.0),),
                          ((-a/2.0,b/2.0,ped/3.0),),
                          ((-a/2.0,-b/2.0,ped/3.0),),
                          ((a/2.0,-b/2.0,ped/3.0),),
                          ((0,0,0),))

    regi1 = p2.Set(cells=c_cells, name='Set_1')
    p2.SectionAssignment(region=regi1, sectionName='fiber section')
```

```
    c_cells = cnew.findAt(((a/4.0 + r_fiber/2.0, 0, ped/3.0),),)
    regi2 = p2.Set(cells=c_cells, name='Set_2')
##    regi2 = regionToolset.Region(cells=c_cells)
    p2.SectionAssignment(region=regi2, sectionName='matrix section')
# creating the coordinate system for the material orientation of the fibers
    CS_1 = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN,
origin=(0.,0.,0.),
            point1=(0.,0.,ped), point2=(a/2.0,0.,ped), name='CS1')
    cs1 = CS_1.id
# assigning the MCS
    p2.assignMaterialOrientation(region = regi1, localCsys=p2.datums[cs1])
    p2.assignMaterialOrientation(region = regi2, localCsys=p2.datums[cs1])

#-----------------------------------------------------------------------
def hexpack_mater():
    num_mat = 0
## creating lamina material
    for inte in range(len(mat_name)):
        algraphiteepoxy=hexmodel.Material(name=mat_name[inte][1])
        propes = (mat_name[inte][2], mat_name[inte][3], mat_name[inte][4],
                  mat_name[inte][5], mat_name[inte][6], mat_name[inte][7],
                  mat_name[inte][8], mat_name[inte][9], mat_name[inte][10])
        algraphiteepoxy.Elastic(table=(propes, ), type=ENGINEERING_CONSTANTS)
        CTEprop = (mat_name[inte][11], mat_name[inte][12],
mat_name[inte][13])
        algraphiteepoxy.Expansion(type=ORTHOTROPIC, table=(CTEprop,))
        num_mat = num_mat + 1
    print num_mat, '- Materials created'
##-----------------------------------------------------------------------
def hexpack_partition_1():
    p2 = hexmodel.parts['recta']
    f = p2.faces
# the inner hexagon

    X = sq3*RR/2.0
    Y = RR/2.0

    innhexcoord = ((X,Y),(X,-Y),(0,-RR),(-X,-Y),(-X,Y),(0,RR),(X,Y))
    face_1 = f.findAt((0.,0.,0.),)
    sketch_1 = hexmodel.Sketch(name='sketch 7', sheetSize=200.0)
    for int in range(len(innhexcoord)-1):
        sketch_1.Line(point1 = innhexcoord[int], point2= innhexcoord[int+1])
    p2.PartitionFaceBySketch(faces=face_1, sketch=sketch_1)
#-----------------------------------------------------------
    X = a/2.0
    Y = (sq3/2.0 - 1.0/sq3)*a
    XX = a/sq3
    outhexcoord = ((X,Y), (X,-Y), (0, -XX), (-X,-Y), (-X,Y), (0, XX), (X, Y))
    face_2 = f.findAt((0,XX,0),)
    sketch_2 = hexmodel.Sketch(name='sketch 8', sheetSize=200.0)
    for int in range(len(outhexcoord)-1):
        sketch_2.Line(point1 = outhexcoord[int], point2= outhexcoord[int+1])
    p2.PartitionFaceBySketch(faces=face_2, sketch=sketch_2)
#-----------------------------------------------------------
# four fibers this is fiber 1 go anti clockwise to check starting at (a/2,
b/2)
```

```
    X = -sq3*RR/2.0 + a/2.0
    Y = sq3 *a /2.0
    X1= X
    Y1 =  -RR/2.0 + sq3 * a /2.0
    X2 = a/2.0
    Y2 = sq3*a/2.0 - RR
    X3 = a/2.0
    Y3 = b/2.0
    face_3 = f.findAt((X1,Y1,0.), )
    fibe_p = ((X,Y), (X1,Y1), (X2,Y2), (X3, Y3), (X,Y))
    sketch_3 = hexmodel.Sketch(name='sketch 9', sheetSize=200.0)
    for int in range(len(fibe_p)-1):
        sketch_3.Line(point1 = fibe_p[int], point2= fibe_p[int+1])
    p2.PartitionFaceBySketch(faces=face_3, sketch=sketch_3)
#-------------------------------------------------------
# fiber 2
    X = sq3*RR/2.0 - a/2.0
    Y = sq3 *a /2.0
    X1= X
    Y1 =  -RR/2.0 + sq3 * a /2.0
    X2 = -a/2.0
    Y2 = sq3*a/2.0 - RR
    X3 = -a/2.0
    Y3 = b/2.0
    face_4 = f.findAt((X1,Y1,0.), )
    fibe_p = ((X,Y), (X1,Y1), (X2,Y2), (X3, Y3), (X,Y))
    sketch_4 = hexmodel.Sketch(name='sketch 10', sheetSize=200.0)
    for int in range(len(fibe_p)-1):
        sketch_4.Line(point1 = fibe_p[int], point2= fibe_p[int+1])
    p2.PartitionFaceBySketch(faces=face_4, sketch=sketch_4)
#-------------------------------------------------------
# fiber 3
    X = -sq3*RR/2.0 + a/2.0
    Y = -sq3 *a /2.0
    X1= -sq3*RR/2.0 + a/2.0
    Y1 =  RR/2.0 - sq3 * a /2.0
    X2 = a/2.0
    Y2 = -sq3*a/2.0 + RR
    X3 = a/2.0
    Y3 = -b/2.0
    face_5 = f.findAt((X1,Y1,0.), )
    fibe_p = ((X,Y), (X1,Y1), (X2,Y2), (X3, Y3), (X,Y))
    sketch_5 = hexmodel.Sketch(name='sketch 11', sheetSize=200.0)
    for int in range(len(fibe_p)-1):
        sketch_5.Line(point1 = fibe_p[int], point2= fibe_p[int+1])
    p2.PartitionFaceBySketch(faces=face_5, sketch=sketch_5)
#-------------------------------------------------------
# fiber 4
    X = sq3*RR/2.0 - a/2.0
    Y = -sq3 *a /2.0
    X1= sq3*RR/2.0 - a/2.0
    Y1 =  RR/2.0 - sq3 * a /2.0
    X2 = -a/2.0
    Y2 = -sq3*a/2.0 + RR
    X3 = -a/2.0
    Y3 = -b/2.0
    face_6 = f.findAt((X1,Y1,0.), )
```

```
    fibe_p = ((X,Y), (X1,Y1), (X2,Y2), (X3, Y3), (X,Y))
    sketch_6 = hexmodel.Sketch(name='sketch 12', sheetSize=200.0)
    for int in range(len(fibe_p)-1):
        sketch_6.Line(point1 = fibe_p[int], point2= fibe_p[int+1])
    p2.PartitionFaceBySketch(faces=face_6, sketch=sketch_6)
#------------------------------------------------------------
    X = 0
    Y = b/2.0
    X1 = 0
    Y1 = a/sq3
    X2 = 0
    Y2 = b/4.0 + a/(2*sq3)
    face_7 = f.findAt((X2,Y2,0.),)
    fibe_p = ((X,Y),(X1,Y1))
    sketch_7 = hexmodel.Sketch(name='sketch 13', sheetSize=200.0)
    for int in range(len(fibe_p)-1):
        sketch_7.Line(point1 = fibe_p[int], point2= fibe_p[int+1])
    p2.PartitionFaceBySketch(faces=face_7, sketch=sketch_7)
###------------------------------------------------------------
    X = 0
    Y = -b/2.0
    X1 = 0
    Y1 = -a/sq3
    X2 = 0
    Y2 = -b/4.0 - a/(2*sq3)
    face_8 = f.findAt((X2,Y2,0.),)
    fibe_p = ((X,Y),(X1,Y1))
    sketch_8 = hexmodel.Sketch(name='sketch 14', sheetSize=200.0)
    for int in range(len(fibe_p)-1):
        sketch_8.Line(point1 = fibe_p[int], point2= fibe_p[int+1])
    p2.PartitionFaceBySketch(faces=face_8, sketch=sketch_8)
###------------------------------------------------------------
    X = sq3*(r_fiber+RR)/4.0
    Y = (r_fiber+RR)/4.0
    X2 = sq3*r_fiber/4.0 + a/4.0
    Y2 = r_fiber/4.0 + (sq3/2.0 - 1/sq3) * a/2.0
    X1 = sq3*RR/2.0
    Y1 = RR/2.0
    X3 = a/2.0
    Y3 = (sq3/2.0 - 1/sq3)*a
    face_9 = f.findAt(((X,Y,0.),), ((X2,Y2,0.),))
    fibe_p = ((X3,Y3),(X1,Y1))
    sketch_9 = hexmodel.Sketch(name='sketch 15', sheetSize=200.0)
    sketch_9.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_9, sketch=sketch_9)
#------------------------------------------------------------
#
    face_10 = f.findAt(((X,-Y,0.),), ((X2,-Y2,0.),))
    fibe_p = ((X3,-Y3),(X1,-Y1))
    sketch_10 = hexmodel.Sketch(name='sketch 16', sheetSize=200.0)
    sketch_10.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_10, sketch=sketch_10)
#------------------------------------------------------------
#
    face_11 = f.findAt(((-X,-Y,0.),), ((-X2,-Y2,0.),))
    fibe_p = ((-X3,-Y3),(-X1,-Y1))
    sketch_11 = hexmodel.Sketch(name='sketch 17', sheetSize=200.0)
```

```
    sketch_11.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_11, sketch=sketch_11)
#------------------------------------------------------------
#
    face_12 = f.findAt(((-X,Y,0.),), ((-X2,Y2,0.),))
    fibe_p = ((-X3,Y3),(-X1,Y1))
    sketch_12 = hexmodel.Sketch(name='sketch 18', sheetSize=200.0)
    sketch_12.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_12, sketch=sketch_12)
#------------------------------------------------------------
    X = (-sq3*r_fiber + a)/2.0 # RX
    Y = (-r_fiber + sq3*a)/2.0 # RY
    X1 = (-sq3*RR + a)/2.0 # PQX
    Y1 = (-RR + sq3*a)/2.0 # PQY
    X2 = 0 # ABX
    Y2 = a/sq3 # ABY
    X3 = (X + X2)/2.0  # (AB)M(R)X
    Y3 = (Y + Y2)/2.0 # (AB)M(R)Y
    X4 = (X1 + X)/2.0 # (PQ)M(R)X
    Y4 = (Y1 + Y)/2.0 # (PQ)M(R)Y
    face_13 = f.findAt(((X3,Y3,0.),), ((X4,Y4,0.),))
    fibe_p = ((X2,Y2),(X1,Y1))
    sketch_13 = hexmodel.Sketch(name='sketch 19', sheetSize=200.0)
    sketch_13.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_13, sketch=sketch_13)
#------------------------------------------------------------
    face_13 = f.findAt(((-X3,Y3,0.),), ((-X4,Y4,0.),))
    fibe_p = ((-X2,Y2),(-X1,Y1))
    sketch_13 = hexmodel.Sketch(name='sketch 20', sheetSize=200.0)
    sketch_13.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_13, sketch=sketch_13)
#------------------------------------------------------------
    face_13 = f.findAt(((-X3,-Y3,0.),), ((-X4,-Y4,0.),))
    fibe_p = ((-X2,-Y2),(-X1,-Y1))
    sketch_13 = hexmodel.Sketch(name='sketch 21', sheetSize=200.0)
    sketch_13.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_13, sketch=sketch_13)
#------------------------------------------------------------
    face_13 = f.findAt(((X3,-Y3,0.),), ((X4,-Y4,0.),))
    fibe_p = ((X2,-Y2),(X1,-Y1))
    sketch_13 = hexmodel.Sketch(name='sketch 22', sheetSize=200.0)
    sketch_13.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_13, sketch=sketch_13)
#------------------------------------------------------------
    X2 = 0 # ABX
    Y2 = a/sq3 # ABY
    X3 = 0 # GX
    Y3 = r_fiber # GY
    X4 = 0 # FX
    Y4 = RR # FY
    X5 = 0 # (AB)M(G)X
    Y5 = (r_fiber + a/sq3)/2.0 # (AB)M(G)Y
    X6 = 0 # (G)M(F)X
    Y6 = (r_fiber + RR)/2.0 # (G)m(F)Y
    fibe_p = ((X2,Y2),(X4,Y4))
    face_14 = f.findAt(((X5,Y5,0.),), ((X6,Y6,0.),))
    sketch_14 = hexmodel.Sketch(name='sketch 23', sheetSize=200.0)
```

```
    sketch_14.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_14, sketch=sketch_14)
#-------------------------------------------------------------
    X2 = 0 # -ABX
    Y2 = -a/sq3 # -ABY
    X3 = 0 # GX
    Y3 = -r_fiber # -GY
    X4 = 0 # -FX
    Y4 = -RR # -FY
    X5 = 0 # (AB)M(G)X
    Y5 = -(r_fiber + a/sq3)/2.0 # -(AB)M(G)Y
    X6 = 0 # (G)M(F)X
    Y6 = -(r_fiber + RR)/2.0 # -(G)M(F)Y
    fibe_p = ((X2,Y2),(X4,Y4))
    face_14 = f.findAt(((X5,Y5,0.),), ((X6,Y6,0.),))
    sketch_14 = hexmodel.Sketch(name='sketch 24', sheetSize=200.0)
    sketch_14.Line(point1 = fibe_p[0], point2= fibe_p[1])
    p2.PartitionFaceBySketch(faces=face_14, sketch=sketch_14)
#-------------------------------------------------------------
def hexpack_partition_2():
    p2 = hexmodel.parts['recta']

# partitioning the cells
# the fibers 1 2 3 4 with Sketches 3 4 5 6
    c = p2.cells
    e = p2.edges
    X1 = -sq3*RR/2.0 + a/2.0 # PQ X
    Y1 = -RR/2.0 + sq3*a/2.0 # PQ Y
    X2 = -sq3*RR/2.0 + a/2.0 # PX X
    Y2 = sq3*a/2.0 # PX Y
    X3 = a/2.0 # PW X
    Y3 = sq3*a/2.0 - RR # PW Y
    X4 = (X1 + X2) / 2.0
    Y4 = (Y1 + Y2) / 2.0
    X5 = a/2.0
    Y5 = b/2.0
    X6 = (X3 + X5) / 2.0
    Y6 = (Y3 + Y5) / 2.0
    X7 = (X2 + X5) / 2.0
    Y7 = (Y2 + Y5) / 2.0
    X8 = (X1 + X3) / 2.0
    Y8 = (Y1 + Y3) / 2.0
# cells in first quadrant ALL
    edges = e.findAt(((X4,Y4,0.),), ((X6,Y6,0.),), ((X7,Y7,0.),),
((X8,Y8,0.),))
    cells = c.findAt(((a/2.0, b/2.0, 0.),))
    lines = e.findAt((a/2.0, b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in second quadrant SILVER
    edges = e.findAt(((-X4,Y4,0.),), ((-X6,Y6,0.),), ((-X7,Y7,0.),), ((-
X8,Y8,0.),))
    cells = c.findAt(((-a/2.0, b/2.0, 0.),))
    lines = e.findAt((-a/2.0, b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in third Quadrant TEA
```

```
    edges = e.findAt(((-X4,-Y4,0.),), ((-X6,-Y6,0.),), ((-X7,-Y7,0.),), ((-
X8,-Y8,0.),))
    cells = c.findAt(((-a/2.0, -b/2.0, 0.),))
    lines = e.findAt((-a/2.0, -b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in fourth Quadrant CUPS
    edges = e.findAt(((X4,-Y4,0.),), ((X6,-Y6,0.),), ((X7,-Y7,0.),), ((X8,-
Y8,0.),))
    cells = c.findAt(((a/2.0, -b/2.0, 0.),))
    lines = e.findAt((a/2.0, -b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in matrix
    X1 = -sq3*RR/2.0 + a/2.0 # PQ X
    Y1 = -RR/2.0 + sq3*a/2.0 # PQ Y
    X2 = -sq3*RR/2.0 + a/2.0 # PX X
    Y2 = sq3*a/2.0 # PX Y
    X3 = (-sq3*r_fiber + a)/2.0 # RX
    Y3 = (-r_fiber + sq3*a)/2.0 # RY
    X5 = 0 # ABX
    Y5 = a/sq3 # ABY
    X6 = 0 #TAX
    Y6 = b/2.0 # TAY
    X7 = a/2.0 - r_fiber # SX
    Y7 = b/2.0 # SY
    X8 = (X5 + X3)/2.0  # (AB)M(R)X
    Y8 = (Y5 + Y3)/2.0 # (AB)M(R)Y
    X9 = (X1 + X3)/2.0 # (PQ)M(R)X
    Y9 = (Y1 + Y3)/2.0 # (PQ)M(R)Y
    X10 = (X1 + X2)/2.0 # (PQ)M(PX)X
    Y10 = (Y1 + Y2)/2.0 # (PQ)M(PX)Y
    X11 = (X2 + X7)/2.0 # (PX)M(S)X
    Y11 = (Y2 + Y7)/2.0 # (PX)M(S)Y
    X12 = (X6 + X7)/2.0 # (S)M(TA)X
    Y12 = (Y6 + Y7)/2.0 # (S)M(TA)Y
    X13 = (X6 + X5)/2.0 # (AB)M(TA)X
    Y13 = (Y6 + Y5)/2.0 # (AB)M(TA)Y
    X14 = (X5 + X7)/2.0 # (AB)M(S)X
    Y14 = (Y5 + Y7)/2.0 # (AB)M(S)Y
    X15 = (X2 + X3)/2.0 # (AB)M(R)X
    Y15 = (Y2 + Y3)/2.0 # (AB)M(R)Y

# cells in first quadrant ALL
    edges = e.findAt(((X8,Y8,0.),), ((X9,Y9,0.),), ((X10,Y10,0.),),
                     ((X11,Y11,0.),), ((X12,Y12,0.),), ((X13,Y13,0.),))
    cells = c.findAt(((X14,Y14, 0.),), ((X15,Y15,0.),))
    lines = e.findAt((a/2.0, b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in  quadrant SILVER
    edges = e.findAt(((-X8,Y8,0.),), ((-X9,Y9,0.),), ((-X10,Y10,0.),),
                     ((-X11,Y11,0.),), ((-X12,Y12,0.),), ((X13,Y13,0.),))
    cells = c.findAt(((-X14,Y14, 0.),), ((-X15,Y15,0.),))
    lines = e.findAt((-a/2.0, b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
```

```
# cells in  quadrant TEA
    edges = e.findAt(((-X8,-Y8,0.),), ((-X9,-Y9,0.),), ((-X10,-Y10,0.),),
                     ((-X11,-Y11,0.),), ((-X12,-Y12,0.),), ((-X13,-Y13,0.),))
    cells = c.findAt(((-X14,-Y14, 0.),), ((-X15,-Y15,0.),))
    lines = e.findAt((-a/2.0, -b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in  quadrant CUPS
    edges = e.findAt(((X8,-Y8,0.),), ((X9,-Y9,0.),), ((X10,-Y10,0.),),
                     ((X11,-Y11,0.),), ((X12,-Y12,0.),), ((X13,-Y13,0.),))
    cells = c.findAt(((X14,-Y14, 0.),), ((X15,-Y15,0.),))
    lines = e.findAt((a/2.0, -b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# PARTITIONING THE INNER CELLS

    X1 = 0 # ABX
    Y1 = a/sq3 # ABY
    X2 = a/2.0 #ADX
    Y2 = (sq3/2.0 - 1/sq3)*a # ADY
    X3 = sq3*r_fiber/2.0 #
    Y3 = r_fiber/2.0
    X4 = sq3*RR/2.0 # AX
    Y4 = RR/2.0 #AY
    X5 = 0 #FX
    Y5 = RR #FY
    X6 = 0 # GX
    Y6 = r_fiber #GY

    X7 = (X1 + X2)/2.0 #ABMAD
    Y7 = (Y1 + Y2)/2.0 #ABMAD
    X8 = (X2 + X3)/2.0 #ADM
    Y8 = (Y2 + Y3)/2.0 #ADM
    X9 = (X3 + X4)/2.0 #MAX
    Y9 = (Y3 + Y4)/2.0 #MAY
    X10 = (X4 + X5)/2.0 # AMFX
    Y10 = (Y4 + Y5)/2.0 # AMFY
    X11 = (X5 + X6)/2.0 #FMGX
    Y11 = (Y5 + Y6)/2.0 #FMGY
    X12 = (X1 + X6)/2.0 #GMABX
    Y12 = (Y1 + Y6)/2.0 #GMABY
# cells in first quadrant ALL
    edges = e.findAt(((X7,Y7,0.),), ((X8,Y8,0.),), ((X9,Y9,0.),),
                     ((X10,Y10,0.),), ((X11,Y11,0.),), ((X12,Y12,0.),))
    cells = c.findAt(((X7,Y7, 0.),), ((X10,Y10,0.),))
    lines = e.findAt((a/2.0, b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in second quadrant SILVER
    edges = e.findAt(((-X7,Y7,0.),), ((-X8,Y8,0.),), ((-X9,Y9,0.),),
                     ((-X10,Y10,0.),), ((-X11,Y11,0.),), ((-X12,Y12,0.),))
    cells = c.findAt(((-X7,Y7, 0.),), ((-X10,Y10,0.),))
    lines = e.findAt((-a/2.0, b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in third quadrant TEA
    edges = e.findAt(((-X7,-Y7,0.),), ((-X8,-Y8,0.),), ((-X9,-Y9,0.),),
```

```
                    ((-X10,-Y10,0.),), ((-X11,-Y11,0.),), ((-X12,-Y12,0.),))
    cells = c.findAt(((-X7,-Y7, 0.),), ((-X10,-Y10,0.),))
    lines = e.findAt((-a/2.0, -b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in fourth quadrant CUPS
    edges = e.findAt(((X7,-Y7,0.),), ((X8,-Y8,0.),), ((X9,-Y9,0.),),
                    ((X10,-Y10,0.),), ((X11,-Y11,0.),), ((X12,-Y12,0.),))
    cells = c.findAt(((X7,-Y7, 0.),), ((X10,-Y10,0.),))
    lines = e.findAt((a/2.0, -b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in x axis
    X1 = a/2.0 #ADX
    Y1 = (sq3/2.0 - 1/sq3)*a # ADY
    X2 = sq3*r_fiber/2.0 #
    Y2 = r_fiber/2.0
    X3 = sq3*RR/2.0 # AX
    Y3 = RR/2.0 #AY
    X4 = sq3*RR/2.0 #BX
    Y4 = -RR/2.0
    X5 = sq3*r_fiber/2.0 #
    Y5 = -r_fiber/2.0
    X6 = a/2.0 #AEX
    Y6 = -(sq3/2.0 - 1/sq3)*a # AEY

    X7 = (X1 + X2)/2.0 #ADM
    Y7 = (Y1 + Y2)/2.0 #ADM
    X8 = (X2 + X3)/2.0 #MAX
    Y8 = (Y2 + Y3)/2.0 #MAY
    X9 = (X3 + X4)/2.0 #AMBX
    Y9 = (Y3 + Y4)/2.0 #AMBY
    X10 = (X4 + X5)/2.0 # BMX
    Y10 = (Y4 + Y5)/2.0 # BMY
    X11 = (X5 + X6)/2.0 #MAEX
    Y11 = (Y5 + Y6)/2.0 #MAEY
    X12 = (X1 + X6)/2.0 #AEMADX
    Y12 = (Y1 + Y6)/2.0 #AEMADY

    edges = e.findAt(((X7,Y7,0.),), ((X8,Y8,0.),), ((X9,Y9,0.),),
                    ((X10,Y10,0.),), ((X11,Y11,0.),), ((X12,Y12,0.),))
    cells = c.findAt(((X7,Y7, 0.),), ((X10,Y10,0.),))
    lines = e.findAt((a/2.0, b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
# cells in -x axis
    edges = e.findAt(((-X7,Y7,0.),), ((-X8,Y8,0.),), ((-X9,Y9,0.),),
                    ((-X10,Y10,0.),), ((-X11,Y11,0.),), ((-X12,Y12,0.),))
    cells = c.findAt(((-X7,Y7, 0.),), ((-X10,Y10,0.),))
    lines = e.findAt((-a/2.0, b/2.0, ped/3.0))
    p2.PartitionCellByExtrudeEdge(cells=cells, edges=edges, line=lines,
sense=FORWARD)
#-----------------------------------------------------------------------------
def hexpack_partition_3():
    p2 = hexmodel.parts['recta']
    global divid, ratio
# diving the whole section into ten parts
```

```
# variable divid = 10
    if((zeta < 0.2)):
        divid = 20
        ratio = 0.01875
    elif((zeta >= 0.2) and (zeta < 0.4)):
        divid = 12
        ratio = 0.0375
    elif((zeta >= 0.4) and (zeta < 0.6)):
        divid = 6
        ratio = 0.0725
    elif((zeta >= 0.6) and (zeta < 0.8)):
        divid = 4
        ratio = 0.1125
    elif((zeta >= 0.8) and (zeta < 1.0)):
        divid = 3
        ratio = 0.15
    else:
        divid = 2
        ratio = 0.1875

    incr = ped*ratio
    incre = ped*(1-ratio*2)/divid
    for num in range(divid+1):
         dspace = incre*num+ incr
         temp_6 = p2.DatumPlaneByThreePoints(point1=(0.0,0.0,dspace),
point2=(a/2.0,0,dspace), point3=(0,b/2.0,dspace))
         datams = temp_6.id
         ccall = p2.cells
         p2.PartitionCellByDatumPlane(cells=ccall,
datumPlane=p2.datums[datams])

#-----------------------------------------------------
def hexpack_partition_4():
    p2 = hexmodel.parts['recta']
# creating datum plane for dividing the model
# first diagonal partition
    ccall = p2.cells
    temp_8 = p2.DatumPlaneByThreePoints(point1=(0,0,ped),
point2=(a/2.0,b/2.0,0), point3=(-a/2.0,-b/2.0,0))
    datams = temp_8.id

    p2.PartitionCellByDatumPlane(cells=ccall, datumPlane=p2.datums[datams])
# second diagonal partition
    ccall = p2.cells
    temp_9= p2.DatumPlaneByThreePoints(point1=(0,0,ped), point2=(-
a/2.0,b/2.0,0), point3=(a/2.0,-b/2.0,0))
    datams = temp_9.id
    p2.PartitionCellByDatumPlane(cells=ccall, datumPlane=p2.datums[datams])
# horizontal partition
    ccall = p2.cells
    temp_9= p2.DatumPlaneByThreePoints(point1=(a/2.0,0,ped/2.0),
point2=(0,0,ped), point3=(0,0,0))
    datams = temp_9.id
    p2.PartitionCellByDatumPlane(cells=ccall, datumPlane=p2.datums[datams])
# hiding the datum planes.. not supressing
    session.viewports['Viewport:
1'].partDisplay.geometryOptions.setValues(datumPlanes=OFF)
```

```
##-----------------------------------------------------------------------------
def hexpack_assembly():
    p2 = hexmodel.parts['recta']
    p4 = hexmodel.rootAssembly
    hexinsta_1 = p4.Instance(name = 'hexpck_1', part = p2, dependent = ON)

##-----------------------------------------------------------------------------
def hexpack_step():
    p4 = hexmodel.rootAssembly
    hexmodel.StaticStep(name='Step-1', previous='Initial')
##-----------------------------------------------------------------------
def hexpack_load():
    p4 = hexmodel.rootAssembly
    inst = p4.instances['hexpck_1']
    v = inst.vertices
    v_1 = v.findAt(((a/2.0,b/2.0,0.0),))
    region = regionToolset.Region(vertices=v_1)
    hexmodel.ConcentratedForce(name='Load', createStepName='Step-1',
region=region, cf1=1000)
##-----------------------------------------------------------------------
##-----------------------------------------------------------------------------
def hexpack_bc():
    p4 = hexmodel.rootAssembly
    vv = p4.instances['hexpck_1']
    v =vv.vertices
    v_1 = v.findAt(((a/2.0,b/2.0,ped),))
    region = regionToolset.Region(vertices=v_1)
    hexmodel.DisplacementBC(name='BC1', createStepName='Initial',
region=region,
            u1=SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET,
            amplitude=UNSET, localCsys=None)
##-----------------------------------------------------------------------------
def hexpack_mesh():
    p2 = hexmodel.parts['recta']
# e_seed_i = seeding numbers
    e_seed_3 = int(seed_1/2.0)
##    e_seed_3 = 2
## this seed number determines how many elements are in the model

    e_seed_4 = 4

## the points on the one half.
    xaa, yaa = 0.0, 0.0
    xab, yab = sq3*RR/2.0, 0.0
    xac, yac = r_fiber, 0.0
    xad, yad = a/2.0, 0.0
    xae, yae = a/2.0, a/(2*sq3)
    xaf, yaf = a/2.0, b/2.0-r_fiber
    xag, yag = a/2.0, b/2.0-RR
    xah, yah = a/2.0, b/2.0
    xai, yai = a/2.0-sq3*RR/2.0, b/2.0
    xaj, yaj = a/2.0-r_fiber, b/2.0
    xak, yak = 0.0, b/2.0
    xal, yal = 0.0, a/sq3
    xam, yam = 0.0, r_fiber
    xan, yan = 0.0, RR
    xao, yao = sq3*RR/2.0, RR/2.0
```

```
xap, yap = sq3*RR/4.0, 3.0*RR/4.0
xaq, yaq = sq3*r_fiber/2.0, r_fiber/2.0
xar, yar = r_fiber/2.0, sq3*r_fiber/2.0
xas, yas = a/4.0, b/4.0
xat, yat = a/2.0-r_fiber/2.0, b/2.0- sq3*r_fiber/2.0
xau, yau = a/2.0-sq3*r_fiber/2.0, b/2.0-r_fiber/2.0
xav, yav = a/2.0-sq3*RR/2.0, b/2.0-RR/2.0
xaw, yaw = a/2.0-sq3*RR/4.0, b/2.0-3.0*RR/4.0

xamb, yamb = 0.5*(xaa + xab), 0.5*(yaa + yab)
xbmc, ybmc = 0.5*(xab + xac), 0.5*(yab + yac)
xcmd, ycmd = 0.5*(xac + xad), 0.5*(yac + yad)
xdme, ydme = 0.5*(xad + xae), 0.5*(yad + yae)
xemf, yemf = 0.5*(xae + xaf), 0.5*(yae + yaf)
xfmg, yfmg = 0.5*(xaf + xag), 0.5*(yaf + yag)
xgmh, ygmh = 0.5*(xag + xah), 0.5*(yag + yah)
xhmi, yhmi = 0.5*(xah + xai), 0.5*(yah + yai)
ximj, yimj = 0.5*(xai + xaj), 0.5*(yai + yaj)
xjmk, yjmk = 0.5*(xaj + xak), 0.5*(yaj + yak)
xkml, ykml = 0.5*(xak + xal), 0.5*(yak + yal)
xlmm, ylmm = 0.5*(xal + xam), 0.5*(yal + yam)
xmmn, ymmn = 0.5*(xam + xan), 0.5*(yam + yan)
xnma, ynma = 0.5*(xan + xaa), 0.5*(yan + yaa)

xlms, ylms = 0.5*(xal + xas), 0.5*(yal + yas)
xsme, ysme = 0.5*(xas + xae), 0.5*(yas + yae)

xamp, yamp = 0.5*(xaa + xap), 0.5*(yaa + yap)

xumv, yumv = 0.5*(xau + xav), 0.5*(yav + yau)
xwmt, ywmt = 0.5*(xaw + xat), 0.5*(yaw + yat)

xpmr, ypmr = 0.5*(xap + xar), 0.5*(yap + yar)
xqmo, yqmo = 0.5*(xaq + xao), 0.5*(yaq + yao)


s15, c15 = sin(15*pi/180), cos(15*pi/180)
s45, c45 = 1/sq2, 1/sq2
s75, c75 = sin(75*pi/180), cos(75*pi/180)

xcmq, ycmq = c15*r_fiber, s15*r_fiber
xrmq, yrmq = c45*r_fiber, s45*r_fiber
xrmm, yrmm = c75*r_fiber, s75*r_fiber

xjmu, yjmu = a/2.0-xcmq, b/2.0-ycmq
xumt, yumt = a/2.0-xrmq, b/2.0-yrmq
xtmf, ytmf = a/2.0-xrmm, b/2.0-yrmm

ximv, yimv = 0.5*(xai + xav), 0.5*(yai + yav)
xvmw, yvmw = 0.5*(xav + xaw), 0.5*(yav + yaw)
xwmg, ywmg = 0.5*(xaw + xag), 0.5*(yag + yaw)
xhmw, yhmw = 0.5*(xah + xaw), 0.5*(yah + yaw)
xuml, yuml = 0.5*(xau + xal), 0.5*(yau + yal)
xrms, yrms = 0.5*(xar + xas), 0.5*(yar + yas)
xsmt, ysmt = 0.5*(xas + xat), 0.5*(yas + yat)
xqme, yqme = 0.5*(xaq + xae), 0.5*(yaq + yae)
xomb, yomb = 0.5*(xao + xab), 0.5*(yao + yab)
```

```
xpmo, ypmo = 0.5*(xap + xao), 0.5*(yap + yao)
xnmp, ynmp = 0.5*(xan + xap), 0.5*(yan + yap)


e, c, f = p2.edges, p2.cells, p2.faces
incr = ped*ratio
incre = ped*(1-ratio*2)/divid
depthset = [0]
for num in range(divid+1):
    temp = incre*num + incr
    depthset = depthset + [temp]
depthset = depthset + [ped]
pedd = ped/divid
for num in range(len(depthset)):
    dd = depthset[num]
    edges = e.findAt(((xamb, yamb, dd),), ((xamb, -yamb, dd),),
                     ((-xamb, yamb, dd),), ((-xamb, -yamb, dd),),
                     ((xgmh, ygmh, dd),), ((-xgmh, ygmh, dd),),
                     ((xgmh, -ygmh, dd),), ((-xgmh, -ygmh, dd),),
                     ((-xhmi, yhmi, dd),), ((-xhmi, -yhmi, dd),),
                     ((xhmi, yhmi, dd),), ((xhmi, -yhmi, dd),),)
    p2.seedEdgeByNumber(edges=edges, number=e_seed_3, constraint=FIXED)

    edges = e.findAt(((xdme, ydme, dd),), ((xdme, -ydme, dd),),
                     ((-xdme, ydme, dd),), ((-xdme, -ydme, dd),),
                     ((xemf, yemf, dd),), ((xemf, -yemf, dd),),
                     ((-xemf, yemf, dd),), ((-xemf, -yemf, dd),),
                     ((xlmm, ylmm, dd),), ((-xlmm, ylmm, dd),),
                     ((xlmm, -ylmm, dd),), ((-xlmm, -ylmm, dd),),
                     ((-xkml, ykml, dd),), ((-xkml, -ykml, dd),),
                     ((xkml, ykml, dd),), ((xkml, -ykml, dd),),)
    p2.seedEdgeByNumber(edges=edges, number=e_seed_3, constraint=FIXED)

    edges = e.findAt(((xamp, yamp, dd),), ((xamp, -yamp, dd),),
                     ((-xamp, yamp, dd),), ((-xamp, -yamp, dd),))

    p2.seedEdgeByNumber(edges=edges, number=e_seed_3, constraint=FIXED)

    edges = e.findAt(((xcmq, ycmq, dd),), ((xcmq, -ycmq, dd),),
                     ((-xcmq, ycmq, dd),), ((-xcmq, -ycmq, dd),),
                     ((xrmq, yrmq, dd),), ((xrmq, -yrmq, dd),),
                     ((-xrmq, yrmq, dd),), ((-xrmq, -yrmq, dd),),
                     ((xrmm, yrmm, dd),), ((-xrmm, yrmm, dd),),
                     ((xrmm, -yrmm, dd),), ((-xrmm, -yrmm, dd),),
                     ((xjmu, yjmu, dd),), ((xjmu, -yjmu, dd),),
                     ((-xjmu, yjmu, dd),), ((-xjmu, -yjmu, dd),),
                     ((xumt, yumt, dd),), ((xumt, -yumt, dd),),
                     ((-xumt, yumt, dd),), ((-xumt, -yumt, dd),),
                     ((xtmf, ytmf, dd),), ((-xtmf, ytmf, dd),),
                     ((xtmf, -ytmf, dd),), ((-xtmf, -ytmf, dd),))

    p2.seedEdgeByNumber(edges=edges, number=e_seed_3, constraint=FIXED)

    edges = e.findAt(((ximj, yimj, dd),), ((ximj, -yimj, dd),),
                     ((-ximj, yimj, dd),), ((-ximj, -yimj, dd),),
                     ((-xumv, yumv, dd),), ((-xumv, -yumv, dd),),
                     ((xumv, -yumv, dd),), ((xumv, yumv, dd),),
```

```
                    ((xwmt, -ywmt, dd),), ((-xwmt, -ywmt, dd),),
                    ((xwmt, ywmt, dd),), ((-xwmt, ywmt, dd),),
                    ((xfmg, -yfmg, dd),), ((-xfmg, -yfmg, dd),),
                    ((xfmg, yfmg, dd),), ((-xfmg, yfmg, dd),),
                    ((xpmr, ypmr, dd),), ((-xpmr, ypmr, dd),),
                    ((xpmr, -ypmr, dd),),((-xpmr, -ypmr, dd),),
                    ((xqmo, yqmo, dd),), ((-xqmo, -yqmo, dd),),
                    ((-xqmo, yqmo, dd),), ((xqmo, -yqmo, dd),),
                    ((xamb, yamb, dd),), ((-xamb, yamb, dd),),
                    ((xjmk, yjmk, dd),), ((xjmk, -yjmk, dd),),
                    ((-xjmk, yjmk, dd),), ((-xjmk, -yjmk, dd),),
                    ((xlmm, ylmm, dd),), ((xlmm, -ylmm, dd),),
                    ((xmmn, ymmn, dd),), ((xmmn, -ymmn, dd),),
                    ((-xcmd, ycmd, dd),), ((xcmd, ycmd, dd),),
                    ((-xbmc, ybmc, dd),), ((xbmc, ybmc, dd),),)

    p2.seedEdgeByNumber(edges=edges, number=e_seed_3, constraint=FIXED)

    edges = e.findAt(((xlms, ylms, dd),), ((xlms, -ylms, dd),),
                    ((-xlms, ylms, dd),), ((-xlms, -ylms, dd),),
                    ((-xsme, ysme, dd),), ((-xsme, -ysme, dd),),
                    ((xsme, -ysme, dd),), ((xsme, ysme, dd),) )
    p2.seedEdgeByNumber(edges=edges, number=e_seed_3, constraint=FIXED)

    edges = e.findAt(((ximv, yimv, dd),), ((ximv, -yimv, dd),),
                    ((-ximv, -yimv, dd),), ((-ximv, yimv, dd),),
                    ((xvmw, yvmw, dd),), ((xvmw, -yvmw, dd),),
                    ((-xvmw, -yvmw, dd),), ((-xvmw, yvmw, dd),),
                    ((xwmg, ywmg, dd),), ((-xwmg, ywmg, dd),),
                    ((-xwmg, -ywmg, dd),), ((xwmg, -ywmg, dd),),
                    ((xhmw, yhmw, dd),), ((-xhmw, yhmw, dd),),
                    ((-xhmw, -yhmw, dd),), ((xhmw, -yhmw, dd),),
                    ((xuml, yuml, dd),), ((-xuml, yuml, dd),),
                    ((-xuml, -yuml, dd),), ((xuml, -yuml, dd),),
                    ((xrms, yrms, dd),), ((-xrms, yrms, dd),),
                    ((-xrms, -yrms, dd),), ((xrms, -yrms, dd),),
                    ((xsmt, ysmt, dd),), ((-xsmt, ysmt, dd),),
                    ((-xsmt, -ysmt, dd),), ((xsmt, -ysmt, dd),),
                    ((xqme, yqme, dd),), ((-xqme, yqme, dd),),
                    ((-xqme, -yqme, dd),), ((xqme, -yqme, dd),),
                    ((xomb, yomb, dd),), ((-xomb, yomb, dd),),
                    ((-xomb, -yomb, dd),), ((xomb, -yomb, dd),),
                    ((xpmo, ypmo, dd),), ((-xpmo, ypmo, dd),),
                    ((-xpmo, -ypmo, dd),), ((xpmo, -ypmo, dd),),
                    ((xnmp, ynmp, dd),), ((-xnmp, ynmp, dd),),
                    ((-xnmp, -ynmp, dd),), ((xnmp, -ynmp, dd),))
    p2.seedEdgeByNumber(edges=edges, number=e_seed_3, constraint=FIXED)
depthsetmid = [0]
for ij in range((len(depthset)-1)):
    temp = (depthset[ij] + depthset[ij+1])*0.5
    depthsetmid = depthsetmid +[temp]
del depthsetmid[depthsetmid.index(0)]
for num in range(divid+2):
    if(e_seed_4 > 2):
        if( (num == 0) or (num == (len(depthset)-2)) ):
            ## seed change for mesh density tests.
            e_seed_42 = e_seed_4 + 2
```

```
            else:
                e_seed_42 = e_seed_4 - 2
        else:
            e_seed_42 = e_seed_4

        dd = depthsetmid[num]
        for i in range(2):
            if(i == 0):
                t = 1.0
            else:
                t = -1.0
            for j in range(2):
                if(j == 0):
                    tt = -1.0
                else:
                    tt = 1.0

                edges = e.findAt(((t*xab, tt*yab, dd),),
                                 ((t*xac, tt*yac, dd),), ((t*xad, tt*yad,
dd),),
                                 ((t*xae, tt*yae, dd),), ((t*xaf, tt*yaf,
dd),),
                                 ((t*xag, tt*yag, dd),), ((t*xah, tt*yah,
dd),),
                                 ((t*xai, tt*yai, dd),), ((t*xaj, tt*yaj,
dd),),
                                 ((t*xao, tt*yao, dd),), ((t*xap, tt*yap,
dd),),
                                 ((t*xaq, tt*yaq, dd),), ((t*xar, tt*yar,
dd),),
                                 ((t*xat, tt*yat, dd),), ((t*xas, tt*yas,
dd),),
                                 ((t*xau, tt*yau, dd),), ((t*xav, tt*yav,
dd),),
                                 ((t*xaw, tt*yaw, dd),))
                p2.seedEdgeByNumber(edges=edges, number=e_seed_42,
constraint=FIXED)

        edges = e.findAt(((t*xak, tt*yak, dd),), ((t*xal, tt*yal, dd),),
                         ((t*xam, tt*yam, dd),), ((t*xan, tt*yan, dd),),
                         ((t*xaa, tt*yaa, dd),))

## assigning the elements and meshing the model
    elemType1 = mesh.ElemType(elemCode=C3D8I, elemLibrary=STANDARD)
    cell = p2.cells
    cells = (cell, )
    elements = (elemType1)
    p2.setElementType(regions=cells , elemTypes=(elemType1, ))

    p2.generateMesh(regions=cells)
##-----------------------------------------------------------------------
def hexpack_job():

    hexjob = mdb.Job(name='Hex_Job',model=hexmodel)
    hexjob.writeInput()
```

```
##-----------------------------------------------------------------------------
from abaqus import *
from abaqusConstants import *
from section import*
import part
import regionToolset
import displayGroupMdbToolset as dgm
import material
import section
import sketch
import assembly
import step
import interaction
import load
import mesh
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import job
import math

hexpack_part()
hexpack_pit()
hexpack_mater()
hexpack_section()
print ' Sections assigned'
hexpack_partition_1()
hexpack_partition_2()
hexpack_partition_3()
hexpack_partition_4()
print ' Partition Done'
hexpack_mesh()
print ' Mesh Created'
hexpack_assembly()
print ' Assembly Part created'
hexpack_step()
print ' Step created'
hexpack_load()
print ' Loads created'
hexpack_bc()
print ' BCs created'
hexpack_job()
print ' Input file created'
```

Code for writing the boundary conditions: ***bc_mnc.py***

```
## this program writes the bc for the hexpack model
## and  matrix failure (crack perpendicular to the fibers)

##  Note: this program requires the information from run_fb.py

for num in range(7):
    case = num + 1
    if (case == 1):
        name = 'Case_'+ str(case) + '.inp'
        output_1 = open(name, 'w')
```

```
    elif(case == 2):
        name = 'Case_'+ str(case) + '.inp'
        output_2 = open(name, 'w')
    elif(case == 3):
        name = 'Case_'+ str(case) + '.inp'
        output_3 = open(name, 'w')
    elif(case == 4):
        name = 'Case_'+ str(case) + '.inp'
        output_4 = open(name, 'w')
    elif(case == 5):
        name = 'Case_'+ str(case) + '.inp'
        output_5 = open(name, 'w')
    elif(case == 6):
        name = 'Case_'+ str(case) + '.inp'
        output_11 = open(name, 'w')
    elif(case == 7):
        name = 'Case_'+ str(case) + '.inp'
        output_12 = open(name, 'w')


output_6 = open('runfile_2.inp', 'w')
output_7 = open('element_sets.inp', 'w')
output_8 = open('debug_sets.inp', 'w')
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def reading_inp_file():
    global num_node, inte, intee, num_elem, inte_el
    global x, y, z, num_n, xa, xb, ya, yb, za, zb, xab, yab, zab
    global node1, node2, node3, node4, node5, node6, node7, node8, num_el
    global tem, tm, nola, crack_layer, thickness, crack_angle, outer_layer
    global num_ori, bc3, bc4
    x, y, z, num_n = [0], [0], [0], [0]
    node1, node2, node3, node4= [0], [0], [0], [0]
    node5, node6, node7, node8 = [0], [0], [0], [0]
    num_el = [0]
    inte, intee = 0, 0
    endfile, node_flag, elem_flag = 0, 0, 0 # logical operators
    num_node, num_elem = 0, 0 # initializing nodes and elements
    num_ori = 0
    flag_elsets = 0

    input = open('Hex_Job.inp', 'r')
    strg = input.readlines()

    while inte in range(len(strg)):
        strg_2 = strg[inte]

        if(strg_2[1:8]=='Element'):
            temporary_name = strg_2
            print 'Reading Elements'
            int_el = inte
            while(elem_flag != 1):
                int_el = int_el + 1
                strg_3 = strg[int_el]
                if(strg_3[0]=='*'):
                    elem_flag = 1
                else:
```

```
                    num_elem = num_elem + 1
                    elem_reading(na=num_elem, stt= strg_3)
                    num_el = num_el + [num_elem]
                    node1 = node1 + [nod1]
                    node2 = node2 + [nod2]
                    node3 = node3 + [nod3]
                    node4 = node4 + [nod4]
                    node5 = node5 + [nod5]
                    node6 = node6 + [nod6]
                    node7 = node7 + [nod7]
                    node8 = node8 + [nod8]
                    inte = int_el
                    nn = num_elem
            del num_el[num_el.index(0)]
            del node1[node1.index(0)]
            del node2[node2.index(0)]
            del node3[node3.index(0)]
            del node4[node4.index(0)]
            del node5[node5.index(0)]
            del node6[node6.index(0)]
            del node7[node7.index(0)]
            del node8[node8.index(0)]
            face_pairs()
## writing nodes and elements

            jjj = 0
            while jjj in range(len(x)):
                strrg = '   '+str(num_n[jjj])+','+'   '+ str(x[jjj]) + ','+'
' + str(y[jjj]) + ','+'   ' + str(z[jjj]) + '\n'
                output_1.write(strrg)
                output_2.write(strrg)
                output_3.write(strrg)
                output_4.write(strrg)
                output_5.write(strrg)
                output_11.write(strrg)
                output_12.write(strrg)
                jjj = jjj + 1
            strg_temp = '** End of Nodes\n'
            output_1.write(strg_temp)
            output_2.write(strg_temp)
            output_3.write(strg_temp)
            output_4.write(strg_temp)
            output_5.write(strg_temp)
            output_11.write(strg_temp)
            output_12.write(strg_temp)


            output_1.write(temporary_name)
            output_2.write(temporary_name)
            output_3.write(temporary_name)
            output_4.write(temporary_name)
            output_5.write(temporary_name)
            output_11.write(temporary_name)
            output_12.write(temporary_name)

            jjj = 0
            while jjj in range(len(num_el)):
```

```
                strg_t_1 = ('  '+str(num_el[jjj])+','+'   '+ str(node1[jjj])
+
                ','+'   ' + str(node2[jjj]) + ','+'   ' + str(node3[jjj]) +
                ','+'   ' + str(node4[jjj]) + ','+'   ' + str(node5[jjj]) +
                ','+'   ' + str(node6[jjj]) + ','+'   ' + str(node7[jjj]) +
                ','+'   ' + str(node8[jjj]) +'\n')
                output_1.write(strg_t_1)
                output_2.write(strg_t_1)
                output_3.write(strg_t_1)
                output_4.write(strg_t_1)
                output_5.write(strg_t_1)
                output_11.write(strg_t_1)
                output_12.write(strg_t_1)
                jjj = jjj + 1

            strg_temp = '** End of Elements\n'
            output_1.write(strg_temp)
            output_2.write(strg_temp)
            output_3.write(strg_temp)
            output_4.write(strg_temp)
            output_5.write(strg_temp)
            output_11.write(strg_temp)
            output_12.write(strg_temp)
## this subroutine creates the different sets required to access
## the change in properties through the depth of the model.
            centroids()

        elif(strg_2[1:5] == 'Node'):
            output_1.write(strg_2)
            output_2.write(strg_2)
            output_3.write(strg_2)
            output_4.write(strg_2)
            output_5.write(strg_2)
            output_11.write(strg_2)
            output_12.write(strg_2)
            print 'Reading Nodes'
            intee = inte
            while (node_flag != 1):
                intee = intee + 1
                strg_2 = strg[intee]
                if(strg_2[0]=='*'):
                    node_flag = 1
                else:
                    num_node = num_node + 1
                    node_reading(n=num_node, st=strg_2)
                    num_n = num_n + [num_node]
                    x = x + [x1]
                    y = y + [y1]
                    z = z + [z1]
                    nn = num_node
##                    strrg = '   '+str(num_n[nn])+','+'   '+ str(x[nn]) +
','+'   ' + str(y[nn]) + ','+'   ' + str(z[nn]) + '\n'
##                    output_1.write(strrg)
##                    output_2.write(strrg)
##                    output_3.write(strrg)
##                    output_4.write(strrg)
##                    output_5.write(strrg)
```

```
                inte = intee
            del x[x.index(0)]
            del y[y.index(0)]
            del z[z.index(0)]
            del num_n[num_n.index(0)]
            xa, xb, ya, yb, za, zb = max(x), min(x), max(y), min(y), max(z),
min(z)
            xab = (xa + xb)/2.0
            yab = (ya + yb)/2.0
            zab = (za + zb)/2.0
            cal_shf()
            node_swap(nt = num_n, xt = x, yt = y, zt = z)

        elif(strg_2[0:2]=='**'):
            output_1.write(strg[inte])
            output_2.write(strg[inte])
            output_3.write(strg[inte])
            output_4.write(strg[inte])
            output_5.write(strg[inte])
            output_11.write(strg[inte])
            output_12.write(strg[inte])

        elif(strg_2[0:9]=='*Boundary'):
            rigid_body_modes()
            inte = inte + 1

        elif(strg_2[0:8]=='*Restart'):
            print ' writing Loads'
            output_1.write(strg_2)
            output_2.write(strg_2)
            output_3.write(strg_2)
            output_4.write(strg_2)
            output_5.write(strg_2)
            output_11.write(strg_2)
            output_12.write(strg_2)

        elif(strg_2[0:6]=='*Cload'):
            loads()
            print 'Writing Loads'
            inte = inte + 1

        elif(strg_2[0:13]=='*End Instance'):
            print 'Writing Constraints'
            center_constraints()
            corner_constraints()
            face_constraints_3_5_1f()
            edge_constraints_35_36_45_46()
            edge_constraints_13_14_23_24_f()
            edge_constraints_15_25_16_26_f()
            if(matrix_failure == 1):
                print 'For matrix failure'
                edge_constraints_13_14_23_24_15_25_26_16m()
            else:
                ## nf - indicates no failure
                print ' No failure'
                face_constraints_1m()
                edge_constraints_13_14_23_24_nf()
```

```
                edge_constraints_15_25_16_26_nf()

            output_1.write(strg_2)
            output_2.write(strg_2)
            output_3.write(strg_2)
            output_4.write(strg_2)
            output_5.write(strg_2)
            output_11.write(strg_2)
            output_12.write(strg_2)

        elif((strg_2[0:6] == '*Elset') and (strg_2[14:17]=='Set')):
            output_1.write(strg_2)
            output_2.write(strg_2)
            output_3.write(strg_2)
            output_4.write(strg_2)
            output_5.write(strg_2)
            output_11.write(strg_2)
            output_12.write(strg_2)
            intel = inte
            flag = 0
            while(flag == 0):
                intel = intel + 1
                strg_5 = strg[intel]
                if(strg_5[0] != '*'):
                    output_1.write(strg_5)
                    output_2.write(strg_5)
                    output_3.write(strg_5)
                    output_4.write(strg_5)
                    output_5.write(strg_5)
                    output_11.write(strg_5)
                    output_12.write(strg_5)
                else:
                    flag = 1
                    inte = intel - 1

            if(flag_elsets == 0):
                flag_elsets = 1
            for ij in range(len(s_z_elems)):
                strgg_10 = '*Elset, elset='+s_z_elems[ij][0]+',   '+
str(s_z_elems[ij][1])+' \n'
                output_7.write(strgg_10)
                lenself = len(s_z_elems[ij])
                lensel = lenself
                im = 0

                for im in range(2, lenself, 8):
                    ik = im
                    check10, check11, check12, check13 = ik,ik+1,ik+2,ik+3
                    check14, check15, check16, check17 = ik+4, ik+5, ik+6,
ik+7
                    if(check10 < lensel):
                        strgg_11 = str(s_z_elems[ij][ik])+',  '
                        if(check11<lensel):
                            strgg_11 = strgg_11+str(s_z_elems[ij][ik+1])+',
'
                            if(check12<lensel):
```

```
                                strgg_11 =
strgg_11+str(s_z_elems[ij][ik+2])+',  '
                                if(check13<lensel):
                                    strgg_11 =
strgg_11+str(s_z_elems[ij][ik+3])+',  '
                                    if(check14<lensel):
                                        strgg_11 =
strgg_11+str(s_z_elems[ij][ik+4])+',  '
                                        if(check15<lensel):
                                            strgg_11 =
strgg_11+str(s_z_elems[ij][ik+5])+',  '
                                            if(check16<lensel):
                                                strgg_11 =
strgg_11+str(s_z_elems[ij][ik+6])+',  '
                                                if(check17<lensel):
                                                    strgg_11 =
strgg_11+str(s_z_elems[ij][ik+7])+',  '

                    strgg_11 = strgg_11 + '\n'
                    output_7.write(strgg_11)

            for ij in range(len(s_z_elemsm)):
                strgg_10 = '*Elset, elset='+s_z_elemsm[ij][0]+',   '+
str(s_z_elemsm[ij][1])+' \n'
                output_7.write(strgg_10)
                lenself = len(s_z_elemsm[ij])
                lensel = lenself
                im = 0

                for im in range(2, lenself, 8):
                    ik = im
                    check10, check11, check12, check13 = ik,ik+1,ik+2,ik+3
                    check14, check15, check16, check17 = ik+4, ik+5, ik+6,
ik+7
                    if(check10 < lensel):
                        strgg_11 = str(s_z_elemsm[ij][ik])+',  '
                        if(check11<lensel):
                            strgg_11 = strgg_11+str(s_z_elemsm[ij][ik+1])+',
'
                            if(check12<lensel):
                                strgg_11 =
strgg_11+str(s_z_elemsm[ij][ik+2])+',  '
                                if(check13<lensel):
                                    strgg_11 =
strgg_11+str(s_z_elemsm[ij][ik+3])+',  '
                                    if(check14<lensel):
                                        strgg_11 =
strgg_11+str(s_z_elemsm[ij][ik+4])+',  '
                                        if(check15<lensel):
                                            strgg_11 =
strgg_11+str(s_z_elemsm[ij][ik+5])+',  '
                                            if(check16<lensel):
                                                strgg_11 =
strgg_11+str(s_z_elemsm[ij][ik+6])+',  '
                                                if(check17<lensel):
                                                    strgg_11 =
strgg_11+str(s_z_elemsm[ij][ik+7])+',  '
```

```
            strgg_11 = strgg_11 + '\n'
            output_7.write(strgg_11)

elif(strg_2[0:14]=='*Solid Section'):
    output_1.write(strg_2)
    output_2.write(strg_2)
    output_3.write(strg_2)
    output_4.write(strg_2)
    output_5.write(strg_2)
    output_11.write(strg_2)
    output_12.write(strg_2)

    output_7.write(strg_2)
    intel = inte
    flag = 0
    while(flag == 0):
        intel = intel + 1
        strg_5 = strg[intel]
        if(strg_5[0] != '*'):
            output_1.write(strg_5)
            output_2.write(strg_5)
            output_3.write(strg_5)
            output_4.write(strg_5)
            output_5.write(strg_5)
            output_11.write(strg_5)
            output_12.write(strg_5)

            output_7.write(strg_5)
        else:
            flag = 1
            inte = intel - 1

elif(strg_2[1:7]=='Orient'):
    output_1.write(strg_2)
    output_2.write(strg_2)
    output_3.write(strg_2)
    output_4.write(strg_2)
    output_5.write(strg_2)
    output_11.write(strg_2)
    output_12.write(strg_2)

    output_7.write(strg_2)
    intel = inte
    flag = 0
    num_ori = num_ori + 1
    while(flag == 0):
        intel = intel + 1
        strg_5 = strg[intel]
        if(strg_5[0] != '*'):
            output_1.write(strg_5)
            output_2.write(strg_5)
            output_3.write(strg_5)
            output_4.write(strg_5)
            output_5.write(strg_5)
            output_11.write(strg_5)
            output_12.write(strg_5)
```

```
            output_7.write(strg_5)
        else:
            flag = 1
            inte = intel - 1

elif(strg_2[1:13]=='End Assembly'):
    strgg0 = '*Nset, nset=n_set_1, instance=hexpck_1 \n'
    strgg1 = str(c_1)+'\n'
    strgg = strgg0 + strgg1
    output_1.write(strgg)
    output_2.write(strgg)
    output_3.write(strgg)
    output_4.write(strgg)
    output_5.write(strgg)
    output_11.write(strgg)
    output_12.write(strgg)

    strgg0 = '*Nset, nset=n_set_2, instance=hexpck_1 \n'
    strgg1 = str(c_2)+'\n'
    strgg = strgg0 + strgg1
    output_1.write(strgg)
    output_2.write(strgg)
    output_3.write(strgg)
    output_4.write(strgg)
    output_5.write(strgg)
    output_11.write(strgg)
    output_12.write(strgg)

  strgg0 = '*Nset, nset=n_set_3, instance=hexpck_1 \n'
    strgg1 = str(c_5)+'\n'
    strgg = strgg0 + strgg1
    output_1.write(strgg)
    output_2.write(strgg)
    output_3.write(strgg)
    output_4.write(strgg)
    output_5.write(strgg)
    output_11.write(strgg)
    output_12.write(strgg)

  strgg0 = '*Nset, nset=n_set_4, instance=hexpck_1 \n'
  strgg1 = str(c_3)+'\n'
    strgg = strgg0 + strgg1
    output_1.write(strgg)
    output_2.write(strgg)
    output_3.write(strgg)
    output_4.write(strgg)
    output_5.write(strgg)
    output_11.write(strgg)
    output_12.write(strgg)

    strgg0 = '*Nset, nset=n_set_5, instance=hexpck_1 \n'
  strgg1 = str(bodyc)+'\n'
    strgg = strgg0 + strgg1
    output_1.write(strgg)
    output_2.write(strgg)
    output_3.write(strgg)
```

```
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)

            strgg0 = '*Nset, nset=n_set_6, instance=hexpck_1 \n'
          strgg1 = str(c_6)+'\n'
            strgg = strgg0 + strgg1
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)

            strgg0 = '*Nset, nset=n_set_7, instance=hexpck_1 \n'
          strgg1 = str(c_6_x)+'\n'
            strgg = strgg0 + strgg1
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)

            strgg0 = '*Nset, nset=n_set_8, instance=hexpck_1 \n'
          strgg1 = str(c_6_y)+'\n'
            strgg = strgg0 + strgg1
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)

            strgg0 ='*Nset, nset=n_set_9, instance=hexpck_1, internal,
generate \n'
          strgg1 = str(1)+','+'  '+ str(num_node)+','+'  '+str(1)+'\n'
            strgg = strgg0 + strgg1
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)

          output_1.write(strg_2)
            output_2.write(strg_2)
            output_3.write(strg_2)
            output_4.write(strg_2)
            output_5.write(strg_2)
            output_11.write(strg_2)
            output_12.write(strg_2)
```

```
            print 'Nsets for rigid body modes and loads - created'

        elif(strg_2[1:14]=='Output, field'):
              output_1.write(strg_2)
              output_2.write(strg_2)
              output_3.write(strg_2)
              output_4.write(strg_2)
              output_5.write(strg_2)
              output_11.write(strg_2)
              output_12.write(strg_2)

              strgg0 = '*Element output,position=centroid \n'
              strgg1 = 'Evol, S \n'
              strgg = strgg0 + strgg1
              output_1.write(strgg)
              output_2.write(strgg)
              output_3.write(strgg)
              output_4.write(strgg)
              output_5.write(strgg)
              output_11.write(strgg)
              output_12.write(strgg)

          else:
              output_1.write(strg[inte])
              output_2.write(strg[inte])
              output_3.write(strg[inte])
              output_4.write(strg[inte])
              output_5.write(strg[inte])
              output_11.write(strg[inte])
              output_12.write(strg[inte])

          inte = inte + 1
      else:
          print 'End of File'

      output_1.close()
      output_2.close()
      output_3.close()
      output_4.close()
      output_5.close()
      output_11.close()
      output_12.close()
      output_6.close()
      output_7.close()
      output_8.close()
##_____________________________________________________________________________
##-----------------------------------------------------------------------------
def cal_shf():
    global shf
    temp_x, temp_y, temp_z  = [0], [0], [0]
    for j in range(len(x)):
        tt = abs(x[j])/10000.0
        tt = str(tt)
        temp_x = temp_x + [int(len(tt))]

        tt = abs(y[j])/10000.0
```

```
        tt = str(tt)
        temp_y = temp_y + [int(len(tt))]

        tt = abs(z[j])/10000.0
        tt = str(tt)
        temp_z = temp_z + [int(len(tt))]

    del temp_x[temp_x.index(0)]
    del temp_y[temp_y.index(0)]
    del temp_z[temp_z.index(0)]

    tx_i, ty_i = temp_x.index(max(temp_x)), temp_y.index(max(temp_y))
    tz_i = temp_z.index(max(temp_z))

    tx_max = x[tx_i]
    ty_max = y[ty_i]
    tz_max = z[tz_i]
    tt = [tx_max, ty_max, tz_max]
    ttt = [len(str(tx_max)), len(str(ty_max)), len(str(tz_max))]
    tttt = str(tt[ttt.index(max(ttt))])
    tttt = tttt.replace('.', ' ')
    tttt = tttt.split()
    t5 = tttt[1]
    t5 = len(t5)
    sh = '1E-'+str(t5-3)
    shf = float(sh)
    if(shf > 1e-03):
        print '            '
        print 'Warning : SHF too low - Possible Error'
        print '            '
##-----------------------------------------------------------------------------
def node_reading(n, st):
    global num_n1, x1, y1, z1
    num_n1 = n
    x1, y1, z1 = 0., 0., 0.
    n_ck = 0

    strg_n = st.replace(',', ' ')
    strg_n = strg_n.split()
    n_k = int(strg_n[0])
    x1 = float(strg_n[1])
    y1 = float(strg_n[2])
    z1 = float(strg_n[3])

#-------------------------------------------------------------------------------
def elem_reading(na, stt):
    global nod1, nod2, nod3, nod4, nod5, nod6, nod7, nod8
    nod1, nod2, nod3, nod4, nod5, nod6, nod7, nod8  = 0, 0, 0, 0, 0, 0, 0, 0

    strg_n = stt.replace(',', ' ')
    strg_n = strg_n.split()
    n_k = int(strg_n[0])
    nod1 = int(strg_n[1])
    nod2 = int(strg_n[2])
    nod3 = int(strg_n[3])
    nod4 = int(strg_n[4])
    nod5 = int(strg_n[5])
```

```
    nod6 = int(strg_n[6])
    nod7 = int(strg_n[7])
    nod8 = int(strg_n[8])

#-------------------------------------------------------------
#-------------------------------------------------------------------------------
def node_swap(nt, xt, yt, zt):
## this subroutine detects the center nodes, corner nodes, edge nodes and
face nodes.
    global tem, tm
    global c_1, c_2, c_3, c_4, c_5, c_6, c_1_1, c_1_2, bodyc
    global c135, c136, c146, c145, c235, c236, c246, c245
    global ct_35, ct_36, ct_45, ct_46
    global e_35, e_36, e_45, e_46
    global e_15f, e_16f, e_25f, e_26f
    global e_13f, e_14f, e_23f, e_24f
    global e_15m, e_16m, e_25m, e_26m
    global e_13m, e_14m, e_23m, e_24m
    global f_1, f_2, f_3, f_4, shf
    global c_6_x, c_6_y
    global f135, f136, f145, f146
    global f235, f236, f245, f246
    global radi

    global xtestc1, ytestc1, ztestc1, xtestc2, ytestc2, ztestc2
    global xtestc3, ytestc3, ztestc3, xtestc4, ytestc4, ztestc4

    j  = 0
    c_1, c_2, c_3, c_4, c_5, c_6 = 0,0,0,0,0,0
    c_11, c_12, c_21, c_22 = 0, 0, 0, 0
    c_1_1, c_1_2 = 0,0
    c135, c136, c146, c145 = 0,0,0,0
    c235, c236, c246, c245 = 0,0,0,0

    f135, f136, f145, f146 = 0,0,0,0
    f235, f236, f245, f246 = 0,0,0,0

    while j in range(len(nt)):
        tx, ty, tz = x[j], y[j], z[j]

        if((abs(tx-xab)<= shf)and(abs(ty-yab)<=shf)and(abs(tz-za)<=shf)):
            c_1 = nt[j]
            print 'Face center - 1', c_1
        elif((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-zb)<=shf)):
            c_2 = nt[j]
            print 'Face center - 2', c_2
        elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-zab)<=shf)):
          c_3 = nt[j]
            print 'Face center - 3', c_3
        elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf)and(abs(tz-zab)<=shf)):
            c_4 = nt[j]
          print 'Face center - 4', c_4
        elif((abs(tx-xa)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-zab)<=shf)):
            c_5 = nt[j]
          print 'Face center - 5', c_5
      elif((abs(tx-xb)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-zab)<=shf)):
            c_6 = nt[j]
```

```
      print 'Face center - 6', c_6

  elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(za-tz)<=shf)):
      c135 = nt[j]
      print 'Corner 135', c135
  elif((abs(xb-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(za-tz)<=shf)):
      c136 = nt[j]
        print 'Corner 136', c136
  elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(za-tz)<=shf)):
      c146 = nt[j]
        print 'Corner 146', c146
  elif((abs(xa-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(za-tz)<=shf)):
      c145 = nt[j]
        print 'Corner 145', c145
  elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(zb-tz)<=shf)):
      c235 = nt[j]
        print 'Corner 235', c235
  elif((abs(xb-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(zb-tz)<=shf)):
      c236 = nt[j]
        print 'Corner 236', c236
    elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(zb-tz)<=shf)):
      c246 = nt[j]
      print 'Corner 246', c246
  elif((abs(xa-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(zb-tz)<=shf)):
      c245 = nt[j]
        print 'Corner 245', c245
    elif((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-zab)<=shf)):
        bodyc = nt[j]

    elif((abs(tx-xb)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-za)<=shf)):
        c_6_x = nt[j]
    elif((abs(xab-tx)<=shf)and(abs(ty-ya)<=shf)and(abs(tz-za)<=shf)):
        c_6_y = nt[j]

    j = j + 1

    tm = [c135, c136, c145, c146, c235, c236, c245, c246]
    tem = [c_1, c_2, c_3, c_4, c_5, c_6]

if(0 in tm):
    print 'Zero in tm'
else:
    print 'TM is complete '

print 'Center nodes & Corner nodes are identified'

xtestc1=x[c135 -1]
ytestc1=y[c135-1]
ztestc1=z[c135-1]

xtestc2=x[c136 -1]
ytestc2=y[c136 -1]
ztestc2=z[c136 -1]

xtestc3=x[c146 -1]
ytestc3=y[c146 -1]
ztestc3=z[c146 -1]
```

```
    xtestc4=x[c145 -1]
    ytestc4=y[c145 -1]
    ztestc4=z[c145 -1]

    radi = r_fiber
    j = 0
    while j in range(len(nt)):
        tx, ty, tz = x[j], y[j], z[j]
        if( (abs(ty-ya)<=shf)and(abs(tz-za)<=shf)and(abs(tx+radi-
xtestc1)<=shf) ):
            f135 = nt[j]
##            print 'f135 '  , f135

        elif( (abs(ty-ya)<=shf)and(abs(tz-za)<=shf)and(abs(tx-radi-
xtestc2)<=shf) ):
            f136 = nt[j]
##            print 'f136 '  , f136

        elif( (abs(ty-yb)<=shf)and(abs(tz-za)<=shf)and(abs(tx+radi-
xtestc1)<=shf) ):
            f145 = nt[j]
##            print 'f145 '  , f145

        elif( (abs(ty-yb)<=shf)and(abs(tz-za)<=shf)and(abs(tx-radi-
xtestc2)<=shf) ):
            f146 = nt[j]
##            print 'f146 '  , f146

        elif( (abs(ty-ya)<=shf)and(abs(tz-zb)<=shf)and(abs(tx+radi-
xtestc1)<=shf) ):
            f235 = nt[j]
##            print 'f235 '  , f235

        elif( (abs(ty-ya)<=shf)and(abs(tz-zb)<=shf)and(abs(tx-radi-
xtestc2)<=shf) ):
            f236 = nt[j]
##            print 'f236 '  , f236

        elif( (abs(ty-yb)<=shf)and(abs(tz-zb)<=shf)and(abs(tx+radi-
xtestc1)<=shf) ):
            f245 = nt[j]
##            print 'f245 '  , f245

        elif( (abs(ty-yb)<=shf)and(abs(tz-zb)<=shf)and(abs(tx-radi-
xtestc2)<=shf) ):
            f246 = nt[j]
##            print 'f246 '  , f246

        j = j + 1
##_____________________________________________________________________________
##-----------------------------------------------------------------------------
## subroutine to detect the edge node 35 36 45 46

    ct_35, ct_36, ct_45, ct_46 = 0, 0, 0, 0
    ct_15f, ct_16f, ct_25f, ct_26f = 0, 0, 0, 0
    ct_13f, ct_14f, ct_23f, ct_24f = 0, 0, 0, 0
```

```
    ct_15m, ct_16m, ct_25m, ct_26m = 0, 0, 0, 0
    ct_13m, ct_14m, ct_23m, ct_24m = 0, 0, 0, 0


    j = 0
    front, fronta, frontb, frontc = 0, 0, 0, 0
    frontd, fronte, frontf, frontg = 0, 0, 0, 0
    fronth, fronti, frontj, frontk = 0, 0, 0, 0

    frontmm, frontma, frontmb, frontmc = 0, 0, 0, 0
    frontmd, frontme, frontmf, frontmg = 0, 0, 0, 0
    frontmh, frontmi, frontmj, frontmk = 0, 0, 0, 0

    e_35, e_36, e_45, e_46 = [0], [0], [0], [0]
    e_15f, e_16f, e_25f, e_26f = [0], [0], [0], [0]
    e_13f, e_14f, e_23f, e_24f = [0], [0], [0], [0]
    e_15m, e_16m, e_25m, e_26m = [0], [0], [0], [0]
    e_13m, e_14m, e_23m, e_24m = [0], [0], [0], [0]

    while j in range(len(nt)):
        tx, ty, tz = x[j], y[j], z[j]
        check = nt[j]
        front, fronta, frontb, frontc  = 0, 0, 0, 0
        frontd, fronte, frontf, frontg = 0, 0, 0, 0
        fronth, fronti, frontj, frontk = 0, 0, 0, 0

        frontmd, frontme, frontmf, frontmg = 0, 0, 0, 0
        frontmh, frontmi, frontmj, frontmk = 0, 0, 0, 0

        if check in tem:
            front, fronta, frontb, frontc  = 1, 1, 1, 1
            frontd, fronte, frontf, frontg = 1, 1, 1, 1
            fronth, fronti, frontj, frontk = 1, 1, 1, 1

            frontmd, frontme, frontmf, frontmg = 1,1,1,1
            frontmh, frontmi, frontmj, frontmk = 1,1,1,1

        if check in tm:
            front, fronta, frontb, frontc  = 1, 1, 1, 1
            frontd, fronte, frontf, frontg = 1, 1, 1, 1
            fronth, fronti, frontj, frontk = 1, 1, 1, 1

            frontmd, frontme, frontmf, frontmg = 1,1,1,1
            frontmh, frontmi, frontmj, frontmk = 1,1,1,1

##       EDGE 35
      if((abs(tx-xa)<=shf)and(abs(ty-ya)<=shf)):
             if(front!=1):
                ct_35 = ct_35 + 1
            e_35 = e_35 + [check]

##     EDGE 36
      elif((abs(tx-xb)<=shf)and(abs(ty-ya)<=shf)):
            if(fronta!=1):
                ct_36 = ct_36 + 1
                e_36 = e_36 + [check]
```

```
##      EDGE 45
        elif( (abs(tx-xa)<=shf)and(abs(ty-yb)<=shf) ):
            if(frontb!=1):
                ct_45 = ct_45 + 1
                e_45 = e_45 + [check]

##      EDGE 46
        elif((abs(tx-xb)<=shf)and(abs(ty-yb)<=shf)):
            if(frontc!=1):
                ct_46 = ct_46 + 1
                e_46 = e_46 + [check]

##      EDGE 15
        elif((abs(tx-xa)<=shf)and(abs(tz-za)<=shf)):

            if(frontd!=1):
            if(((abs(-ty+ytestc4)-radi)<=shf) or
                   ((abs(-ty+ytestc1)-radi)<=shf)):
                    ct_15f = ct_15f + 1
                    e_15f = e_15f+ [check]
                else:
                    ct_15m = ct_15m + 1
                    e_15m = e_15m+ [check]

##      EDGE 25
      elif((abs(tx-xa)<=shf)and(abs(tz-zb)<=shf)):
            if(fronte!=1):
                if(((abs(-ty+ytestc4)-radi)<=shf) or
                   ((abs(-ty+ytestc1)-radi)<=shf)):
                    ct_25f = ct_25f + 1
                    e_25f = e_25f + [check]
                else:
                    ct_25m = ct_25m + 1
                    e_25m = e_25m + [check]

##      EDGE 16
      elif((abs(tx-xb)<=shf)and(abs(tz-za)<=shf)):
            if(frontf!=1):
                if(((abs(-ty+ytestc3)-radi)<=shf) or
                   ((abs(-ty+ytestc2)-radi)<=shf)):
                ct_16f = ct_16f + 1
                    e_16f = e_16f + [check]
                else:
                ct_16m = ct_16m + 1
                    e_16m = e_16m + [check]

##      EDGE 26
      elif((abs(tx-xb)<=shf)and(abs(tz-zb)<=shf)):
          if(frontg!=1):
            if(((abs(-ty+ytestc3)-radi)<=shf) or
                   ((abs(-ty+ytestc2)-radi)<=shf)):
                    ct_26f = ct_26f + 1
                    e_26f = e_26f + [check]
                else:
                    ct_26m = ct_26m + 1
                    e_26m = e_26m + [check]
```

```
##      EDGE 13
      elif((abs(ty-ya)<=shf)and(abs(tz-za)<=shf)):
            if(fronth!=1):
                if( (tx<x[f135-1]) and  (tx>x[f136-1]) ):
                    ct_13m = ct_13m + 1
                    e_13m = e_13m + [check]
                else:
                    ct_13f = ct_13f + 1
                    e_13f = e_13f + [check]

##      EDGE 14
        elif((abs(ty-yb)<=shf)and(abs(tz-za)<=shf)):
            if(fronti!=1):
            if((tx<x[f145-1]) and  (tx>x[f146-1]) ):
                    ct_14m = ct_14m + 1
                    e_14m = e_14m + [check]
                else:
                    ct_14f = ct_14f + 1
                    e_14f = e_14f + [check]

##      EDGE 23
        elif((abs(ty-ya)<=shf)and(abs(tz-zb)<=shf)):
            if(frontj!=1):
                if((tx<x[f235-1]) and  (tx>x[f236-1]) ):
                    ct_23m = ct_23m + 1
                    e_23m = e_23m + [check]
                else:
                    ct_23f = ct_23f + 1
                    e_23f = e_23f + [check]

##      EDGE 24
        elif((abs(ty-yb)<=shf)and(abs(tz-zb)<=shf)):
            if(frontk!=1):
            if((tx<x[f245-1]) and  (tx>x[f246-1]) ):
                    ct_24m = ct_24m + 1
                    e_24m = e_24m + [check]
                else:
                    ct_24f = ct_24f + 1
                    e_24f = e_24f + [check]


        j = j + 1
    del e_35[e_35.index(0)]
    del e_36[e_36.index(0)]
    del e_45[e_45.index(0)]
    del e_46[e_46.index(0)]
    del e_15f[e_15f.index(0)]
    del e_16f[e_16f.index(0)]
    del e_25f[e_25f.index(0)]
    del e_26f[e_26f.index(0)]
    del e_13f[e_13f.index(0)]
    del e_23f[e_23f.index(0)]
    del e_14f[e_14f.index(0)]
    del e_24f[e_24f.index(0)]
    del e_15m[e_15m.index(0)]
    del e_16m[e_16m.index(0)]
    del e_25m[e_25m.index(0)]
```

```
    del e_26m[e_26m.index(0)]
    del e_13m[e_13m.index(0)]
    del e_23m[e_23m.index(0)]
    del e_14m[e_14m.index(0)]
    del e_24m[e_24m.index(0)]
    print 'Edge nodes are identified'
####-------------------------------------------------------------------------
#### nodes on the edges are identified. These nodes have to be paired for
writing
#### the constraint equations.  the node pairs are based on the boundary
conditions.
    global e_3635, e_4635, e_4535
    global ct_e_3635, ct_e_4535, ct_e_4635
    e_3635, e_4535, e_4635 = [0], [0], [0]
    ct_e_3635, ct_e_4635, ct_e_4535 = 0,0,0
    global e_1615f, e_2615f, e_2515f
    global ct_e_1615f, ct_e_2615f , ct_e_2515f
    e_2615f, e_1615f, e_2515f = [0], [0], [0]
    ct_e_1615f, ct_e_2615f,  ct_e_2515f = 0, 0, 0
    global e_1615m, e_2615m, e_2515m
    global ct_e_1615m, ct_e_2615m , ct_e_2515m
    e_2615m, e_1615m, e_2515m = [0], [0], [0]
    ct_e_1615m, ct_e_2615m, ct_e_2515m = 0, 0, 0
## for matrix failure
    global e_2423m, e_2625m
    global ct_e_2423m, ct_e_2625m
    e_2423m, e_2625m = [0], [0]
    ct_e_2423m, ct_e_2625m = 0, 0

    k,m = 0,0
    if(ct_16m != ct_15m):
            print 'The edges nodes are not compatible 16 15 -- check the
code'
    while k in range(len(e_16m)):
        check1 = e_16m[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_15m)):
            check2 = e_15m[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_1615m = ct_e_1615m + 2
                e_1615m = e_1615m + [check1]
                e_1615m = e_1615m + [check2]
            m = m + 1
        k = k + 1
    del e_1615m[e_1615m.index(0)]
    if((ct_e_1615m/ct_15m != 2) and (ct_e_1615m/ct_15m != 2)):
        print '  Error in identifying edge pairs -  Edges 16 and 15 M'
    else:
        print '  Edge pairs 16 and 15 M are identified'
####-----------------------------------------------
    k,m = 0,0
    if(ct_16f != ct_15f):
            print 'The edges nodes are not compatible 16 15 -- check the
code'
    while k in range(len(e_16f)):
```

```
        check1 = e_16f[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_15f)):
            check2 = e_15f[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_1615f = ct_e_1615f + 2
                e_1615f = e_1615f + [check1]
                e_1615f = e_1615f + [check2]
            m = m + 1
        k = k + 1
    del e_1615f[e_1615f.index(0)]
    if((ct_e_1615f/ct_15f != 2) and (ct_e_1615f/ct_15f != 2)):
        print '  Error in identifying edge pairs -  Edges 16 and 15 F'
    else:
        print '  Edge pairs 16 and 15 F are identified'
####-----------------------------------------------
    k, m = 0, 0
    if(ct_15f != ct_26f):
            print 'The edges nodes are not compatible 15 26 -- check the
code'
    while k in range(len(e_26f)):
        check1 = e_26f[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_15f)):
            check2 = e_15f[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_2615f = ct_e_2615f + 2
                e_2615f = e_2615f + [check1]
                e_2615f= e_2615f + [check2]
            m = m + 1
        k = k + 1
    del e_2615f[e_2615f.index(0)]
    if((ct_e_2615f/ct_15f != 2) and (ct_e_2615f/ct_26f != 2)):
        print '  Error in identifying edge pairs -  Edges 15 and 26 F '
    else:
        print '  Edge pairs 15 and 26 F are identified'
##-----------------------------------------------
    k, m = 0, 0
    if(ct_15m != ct_26m):
            print 'The edges nodes are not compatible 15 26 M -- check the
code'
    while k in range(len(e_26m)):
        check1 = e_26m[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_15m)):
            check2 = e_15m[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_2615m = ct_e_2615m + 2
                e_2615m = e_2615m + [check1]
                e_2615m = e_2615m + [check2]
            m = m + 1
```

```
        k = k + 1
    del e_2615m[e_2615m.index(0)]
    if((ct_e_2615m/ct_15m != 2) and (ct_e_2615m/ct_26m != 2)):
        print '  Error in identifying edge pairs -  Edges 15 and 26 M'
    else:
        print '  Edge pairs 15 and 26 M are identified'
##-----------------------------------------------------------------------
    k, m = 0, 0
    if(ct_25f != ct_15f):
            print 'The edges nodes are not compatible 15 25 -- F check the
code'
    while k in range(len(e_25f)):
        check1 = e_25f[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_15f)):
            check2 = e_15f[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_2515f = ct_e_2515f + 2
                e_2515f = e_2515f + [check1]
                e_2515f = e_2515f + [check2]
            m = m + 1
        k = k + 1
    del e_2515f[e_2515f.index(0)]
    if((ct_e_2515f/ct_25f != 2) and (ct_e_2515f/ct_15f != 2)):
        print '  Error in identifying edge pairs -  F Edges 25 and 15'
    else:
        print '  Edge pairs 25 and 15 F are identified'
####-------------------------------------------------
    k, m = 0, 0
    if(ct_25m != ct_15m):
            print 'The edges nodes are not compatible 15 25 -- M check the
code'
    while k in range(len(e_25m)):
        check1 = e_25m[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_15m)):
            check2 = e_15m[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_2515m = ct_e_2515m + 2
                e_2515m = e_2515m + [check1]
                e_2515m = e_2515m + [check2]
            m = m + 1
        k = k + 1
    del e_2515m[e_2515m.index(0)]
    if((ct_e_2515m/ct_25m != 2) and (ct_e_2515m/ct_15m != 2)):
        print '  Error in identifying edge pairs M -  Edges 25 and 15'
    else:
        print '  Edge pairs 25 and 15 M are identified'
####-------------------------------------------------
    k, m = 0, 0
    if(ct_35 != ct_46):
            print 'The edges nodes are not compatible  35 45 -- check the
code'
```

```
    while k in range(len(e_46)):
        check1 = e_46[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_35)):
            check2 = e_35[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tz1-tz2)<=shf):
                ct_e_4635 = ct_e_4635 + 2
                e_4635 = e_4635 + [check1]
                e_4635 = e_4635 + [check2]
            m = m + 1
        k = k + 1
    del e_4635[e_4635.index(0)]
    if((ct_e_4635/ct_35 != 2) and (ct_e_4635/ct_46 != 2)):
        print '  Error in identifying edge pairs -  Edges 35 and 46'
    else:
        print '  Edge pairs 35 and 46 are identified'
####-----------------------------------------------------
    k, m = 0, 0
    if(ct_36 != ct_35):
            print 'The edges nodes are not compatible  36 35 -- check the
code'
    while k in range(len(e_36)):
        check1 = e_36[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_35)):
            check2 = e_35[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tz1-tz2)<=shf):
                ct_e_3635 = ct_e_3635 + 2
                e_3635 = e_3635 + [check1]
                e_3635 = e_3635 + [check2]
            m = m + 1
        k = k + 1
    del e_3635[e_3635.index(0)]
    if((ct_e_3635/ct_36 != 2) and (ct_e_3635/ct_35 != 2)):
        print '  Error in identifying edge pairs -  Edges 36 and 35'
    else:
        print '  Edge pairs 36 and 35 are identified'
####-----------------------------------------------------
    k, m = 0, 0
    if(ct_45 != ct_35):
            print 'The edges nodes are not compatible  45 46 -- check the
code'
    while k in range(len(e_45)):
        check1 = e_45[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_35)):
            check2 = e_35[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tz1-tz2)<=shf):
                ct_e_4535 = ct_e_4535 + 2
                e_4535 = e_4535 + [check1]
                e_4535 = e_4535 + [check2]
```

```
            m = m + 1
        k = k + 1
    del e_4535[e_4535.index(0)]
    if((ct_e_4535/ct_45 != 2) and (ct_e_4535/ct_35 != 2)):
        print '  Error in identifying edge pairs -  Edges 45 and 35'
    else:
        print '  Edge pairs 45 and 35 are identified'
####------------------------------------------------
####------------------------------------------------
    global ct_e_1413f, ct_e_2313f, ct_e_2413f
    global e_1413f, e_2313f, e_2413f
    e_1413f, e_2313f, e_2413f = [0], [0], [0]

    ct_e_1413f, ct_e_2313f, ct_e_2413f = 0, 0, 0
    k, m = 0, 0
    if(ct_14f != ct_13f):
        print 'The edges nodes are not compatible  14 13 F -- check the code'
    while k in range(len(e_14f)):
        check1 = e_14f[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_13f)):
            check2 = e_13f[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if((abs(tx1-tx2)<=shf) and (abs(ty1+ty2)<=shf)):
                ct_e_1413f = ct_e_1413f + 2
                e_1413f = e_1413f + [check1]
                e_1413f = e_1413f + [check2]
            m = m + 1
        k = k + 1
    del e_1413f[e_1413f.index(0)]
    if((ct_e_1413f/ct_14f != 2) and (ct_e_1413f/ct_13f != 2)):
        print '  Error in identifying edge pairs -  Edges 14 and 13  F
Region'
    else:
        print '  Edge pairs 14 and 13 F are identified'
####------------------------------------------------
    k, m = 0, 0
    if(ct_23f != ct_13f):
            print 'The edges nodes are not compatible  23 24 F -- check the
code'
    while k in range(len(e_23f)):
        check1 = e_23f[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_13f)):
            check2 = e_13f[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tx1-tx2)<=shf):
                ct_e_2313f = ct_e_2313f + 2
                e_2313f = e_2313f + [check1]
                e_2313f = e_2313f + [check2]
            m = m + 1
        k = k + 1
    del e_2313f[e_2313f.index(0)]
    if((ct_e_2313f/ct_23f != 2) and (ct_e_2313f/ct_13f != 2)):
```

```
        print '  Error in identifying edge pairs -  Edges 23 and 13  F
Region'
    else:
        print '  Edge pairs 23 and 13 F are identified'
####-------------------------------------------------
    k, m = 0, 0
    if(ct_13f != ct_24f):
            print 'The edges nodes are not compatible  13 24 F  -- check the
code'
    while k in range(len(e_24f)):
        check1 = e_24f[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_13f)):
            check2 = e_13f[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if((abs(tx1-tx2)<=shf) and (abs(ty1+ty2)<=shf)):
                ct_e_2413f = ct_e_2413f + 2
                e_2413f = e_2413f + [check1]
                e_2413f = e_2413f + [check2]
            m = m + 1
        k = k + 1
    del e_2413f[e_2413f.index(0)]
    if((ct_e_2413f/ct_13f != 2) and (ct_e_2413f/ct_24f != 2)):
        print '  Error in identifying edge pairs -  Edges 13 and 24  F
Region'
    else:
        print '  Edge pairs 13 and 24 F are identified '
####-------------------------------------------------------------------------
    global ct_e_1413m, ct_e_2313m, ct_e_2413m
    global e_1413m, e_2313m, e_2413m
    e_1413m, e_2313m, e_2413m = [0], [0], [0]

    ct_e_1413m, ct_e_2313m, ct_e_2413m = 0, 0, 0
    k, m = 0, 0
    if(ct_14m != ct_13m):
        print 'The edges nodes are not compatible  14 24 M -- check the code'
    while k in range(len(e_14m)):
        check1 = e_14m[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_13m)):
            check2 = e_13m[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if((abs(tx1-tx2)<=shf) and (abs(ty1+ty2)<=shf)):
                ct_e_1413m = ct_e_1413m + 2
                e_1413m = e_1413m + [check1]
                e_1413m = e_1413m + [check2]
            m = m + 1
        k = k + 1
    del e_1413m[e_1413m.index(0)]
    if((ct_e_1413m/ct_14m != 2) and (ct_e_1413m/ct_13m != 2)):
        print '  Error in identifying edge pairs -  Edges 14 and 13 M
Region'
    else:
        print '  Edge pairs 14 and 13 M are identified'
######-----------------------------------------------
```

```
    k, m = 0, 0
    if(ct_23m != ct_13m):
            print 'The edges nodes are not compatible  23 24 M -- check the
code'
    while k in range(len(e_23m)):
        check1 = e_23m[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_13m)):
            check2 = e_13m[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tx1-tx2)<=shf):
                ct_e_2313m = ct_e_2313m + 2
                e_2313m = e_2313m + [check1]
                e_2313m = e_2313m + [check2]
            m = m + 1
        k = k + 1
    del e_2313m[e_2313m.index(0)]
    if((ct_e_2313m/ct_23m != 2) and (ct_e_2313m/ct_13m != 2)):
        print '  Error in identifying edge pairs -  Edges 23 and 24 M
Region'
    else:
        print '  Edge pairs 23 and 13 M are identified'
######-------------------------------------------------
    k, m = 0, 0
    if(ct_13m != ct_24m):
            print 'The edges nodes are not compatible  13 24 -- check the
code'
    while k in range(len(e_24m)):
        check1 = e_24m[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_13m)):
            check2 = e_13m[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if((abs(tx1-tx2)<=shf) and (abs(ty1+ty2)<=shf)):
                ct_e_2413m = ct_e_2413m + 2
                e_2413m = e_2413m + [check1]
                e_2413m = e_2413m + [check2]
            m = m + 1
        k = k + 1
    del e_2413m[e_2413m.index(0)]
    if((ct_e_2413m/ct_13m != 2) and (ct_e_2413m/ct_24m != 2)):
        print '  Error in identifying edge pairs -  Edges 13 and 24  M
Uncracked Region'
    else:
        print '  Edge pairs 13 and 24 M are identified '

## for matrix failure only..
######-------------------------------------------------
    k, m = 0, 0
    if(ct_23m != ct_24m):
            print 'The edges nodes are not compatible  13 24 -- check the
code'
    while k in range(len(e_24m)):
        check1 = e_24m[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
```

```
        m = 0
        while m in range(len(e_23m)):
            check2 = e_23m[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if((abs(tx1-tx2)<=shf) and (abs(ty1+ty2)<=shf)):
                ct_e_2423m = ct_e_2423m + 2
                e_2423m = e_2423m + [check1]
                e_2423m = e_2423m + [check2]
            m = m + 1
        k = k + 1
    del e_2423m[e_2423m.index(0)]
    if((ct_e_2423m/ct_23m != 2) and (ct_e_2423m/ct_24m != 2)):
        print '  Error in identifying edge pairs -  Edges 13 and 24  M
Uncracked Region'
    else:
        print '  Edge pairs 23 and 24 M - for matrix failure are identified '
##-------------------------------------------------
    k, m = 0, 0
    if(ct_25m != ct_26m):
            print 'The edges nodes are not compatible 15 26 M -- check the
code'
    while k in range(len(e_26m)):
        check1 = e_26m[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_25m)):
            check2 = e_25m[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_2625m = ct_e_2625m + 2
                e_2625m = e_2625m + [check1]
                e_2625m = e_2625m + [check2]
            m = m + 1
        k = k + 1
    del e_2625m[e_2625m.index(0)]
    if((ct_e_2625m/ct_25m != 2) and (ct_e_2625m/ct_26m != 2)):
        print '  Error in identifying edge pairs -  Edges 26 and 25 M'
    else:
        print '  Edge pairs 26 and 25 M  - for matrix failure are identified'

##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
def face_pairs():
## nodes on faces are recognized now. we do not need face 5 and face 6 nodes.
    global ct_f_1f, ct_f_2f, ct_f_3f, ct_f_4f, ct_f_5f, ct_f_6f
    global ct_f_1f, ct_f_2f, ct_f_3f, ct_f_4f, ct_f_5f, ct_f_6f
    global f_1f, f_2f, f_3f, f_4f, f_5f, f_6f
    global f_1m, f_2m, f_3m, f_4m, f_5m, f_6m
    global xtestfcen1, ytestfcen1, zztestfcen1

    global cnt
    ct_f_1f, ct_f_2f, ct_f_3f, ct_f_4f, ct_f_5f, ct_f_6f = 0, 0, 0, 0, 0, 0
    ct_f_1m, ct_f_2m, ct_f_3m, ct_f_4m, ct_f_5m, ct_f_6m = 0, 0, 0, 0, 0, 0
    f_1f, f_2f, f_3f, f_4f, f_5f, f_6f = [0], [0], [0], [0], [0], [0]
    f_1m, f_2m, f_3m, f_4m, f_5m, f_6m = [0], [0], [0], [0], [0], [0]

    j = 0
```

```
    while j in range(len(num_n)):
        tx, ty, tz = x[j], y[j], z[j]
        check = num_n[j]
        frontl, frontm, frontn, fronto = 0, 0, 0, 0
        front_5, front_6 = 0, 0

        if check in tm:     frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in tem:    frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_35:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_36:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_45:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_46:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_15f:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_25f:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_16f:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_26f:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_13f:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_23f:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_14f:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_24f:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_15m:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_25m:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_16m:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_26m:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_13m:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_23m:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_14m:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_24m:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1

        xtestfcen1=x[c_1-1]
        ytestfcen1=y[c_1-1]
        ztestfcen1=z[c_1-1]

        if(abs(tz-za)<=shf):
            if(frontl != 1):
```

```
                radi_c1=sqrt((ty-ytestc1)**2+(tx-xtestc1)**2)
                radi_c2=sqrt((ty-ytestc2)**2+(tx-xtestc2)**2)
                radi_c3=sqrt((ty-ytestc3)**2+(tx-xtestc3)**2)
                radi_c4=sqrt((ty-ytestc4)**2+(tx-xtestc4)**2)
                radi_center=sqrt((ty-ytestfcen1)**2+(tx-xtestfcen1)**2)
                if((abs(radi_c1-radi)<shf) or (abs(radi_c2-radi)<shf) or
                   (abs(radi_c3-radi)<shf) or (abs(radi_c4-radi)<shf)
or(abs(radi_center-radi)<shf)
                   or (radi_c1<radi) or (radi_c2<radi) or (radi_c3<radi) or
                   (radi_c4<radi) or (radi_center<radi)):
                    ct_f_1f = ct_f_1f + 1
                    f_1f = f_1f + [check]
                else:
                    ct_f_1m = ct_f_1m + 1
                    f_1m = f_1m + [check]

        elif(abs(tz-zb)<=shf):
            if(frontm != 1):
                radi_c1=sqrt((ty-ytestc1)**2+(tx-xtestc1)**2)
                radi_c2=sqrt((ty-ytestc2)**2+(tx-xtestc2)**2)
                radi_c3=sqrt((ty-ytestc3)**2+(tx-xtestc3)**2)
                radi_c4=sqrt((ty-ytestc4)**2+(tx-xtestc4)**2)
                radi_center=sqrt((ty-ytestfcen1)**2+(tx-xtestfcen1)**2)
                if((abs(radi_c1-radi)<shf)or(abs(radi_c2-radi)<shf) or
                   (abs(radi_c3-radi)<shf)or(abs(radi_c4-
radi)<shf)or(abs(radi_center-radi)<shf)
                   or(radi_c1<radi)
or(radi_c2<radi)or(radi_c3<radi)or(radi_c4<radi)
                   or(radi_center<radi)):
                    ct_f_2f = ct_f_2f + 1
                    f_2f = f_2f + [check]
                else:
                    ct_f_2m = ct_f_2m + 1
                    f_2m = f_2m + [check]


        elif(abs(ty-ya)<=shf):
            if(frontn != 1):
                if(((abs(tx-xtestc1)<radi) or (abs(tx-xtestc2)<radi)) or
                    ((abs(tx-xtestc2)-radi)<=shf) or ((abs(tx-xtestc1)-
radi)<=shf)):
                    ct_f_3f = ct_f_3f + 1
                    f_3f = f_3f + [check]
                else:
                    ct_f_3m = ct_f_3m + 1
                    f_3m = f_3m + [check]

        elif(abs(ty-yb)<=shf):
             if(fronto != 1):
                if(((abs(tx-xtestc1)<radi) or (abs(tx-xtestc2)<radi))or
                   ((abs(tx-xtestc2)-radi)<shf)or((abs(tx-xtestc1)-
radi)<shf)):
                    ct_f_4f = ct_f_4f + 1
                    f_4f = f_4f + [check]
                else:
                    ct_f_4m = ct_f_4m + 1
                    f_4m = f_4m + [check]
```

```
        elif(abs(tx-xa)<=shf):
            if(front_5 != 1):

                if(((abs(ty-ytestc1)<radi)or(abs(ty-ytestc4)<radi))or
                   ((abs(ty-ytestc1)-radi)<shf) or ((abs(ty-ytestc4)-
radi)<shf)):
                    ct_f_5f = ct_f_5f + 1
                    f_5f = f_5f + [check]
                else:
                    ct_f_5m = ct_f_5m + 1
                    f_5m = f_5m + [check]

        elif(abs(tx-xb) <= shf):
            if(front_6 != 1):
                if(((abs(ty-ytestc1)<radi)or(abs(ty-ytestc4)<radi))or
                   ((abs(ty-ytestc1)-radi)<shf)or((abs(ty-ytestc4)-
radi)<shf)):
                    ct_f_6f = ct_f_6f + 1
                    f_6f = f_6f + [check]
                else:
                    ct_f_6m = ct_f_6m + 1
                    f_6m = f_6m + [check]

        j = j + 1

    del f_1f[f_1f.index(0)]
    del f_2f[f_2f.index(0)]
    del f_3f[f_3f.index(0)]
    del f_4f[f_4f.index(0)]
    del f_5f[f_5f.index(0)]
    del f_6f[f_6f.index(0)]
    del f_1m[f_1m.index(0)]
    del f_2m[f_2m.index(0)]
    del f_3m[f_3m.index(0)]
    del f_4m[f_4m.index(0)]
    del f_5m[f_5m.index(0)]
    del f_6m[f_6m.index(0)]

    print 'Face nodes are identified'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
    global ct_f_12f, ct_f_34f, fp_12f, fp_34f, ct_f_56f, fp_56f
    global ct_f_12m, ct_f_34m, fp_12m, fp_34m, ct_f_56m, fp_56m

    fp_12f, fp_34f, fp_56f = [0], [0], [0]
    ct_f_12f, ct_f_34f, ct_f_56f  = 0, 0, 0
    fp_12m, fp_34m, fp_56m = [0], [0], [0]
    ct_f_12m, ct_f_34m, ct_f_56m  = 0, 0, 0

    j,k = 0,0
    while j in range(len(f_1f)):
        if(ct_f_1f != ct_f_2f):
            print 'Face pairs F1 and F2 F are not matched'
        check1 = f_1f[j]
        tx1, ty1, tz1 = x[check1 - 1],y[check1 - 1], z[check1 - 1]
        k = 0
```

```
        while k in range(len(f_2f)):
            check2 = f_2f[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tx1 - tx2) <= 2.0e-3) and (abs(ty1 - ty2) <= 2.0e-3)):
                ct_f_12f = ct_f_12f + 2
                fp_12f = fp_12f + [check1]
                fp_12f = fp_12f + [check2]
                x[check1 - 1] = x[check2 - 1]
                y[check1 - 1] = y[check2 - 1]
            k = k + 1
        j = j + 1
    del fp_12f[fp_12f.index(0)]
    if((ct_f_12f/ct_f_1f != 2) and (ct_f_12f/ct_f_2f != 2)):
        print '  Error in identifying face pairs -  Face 1 and Face 2 F '
    else:
        print '  Face pairs F1 and F2 F are identified'
##-----------------------------------------------------------------
    j,k = 0,0
    while j in range(len(f_1m)):
        if(ct_f_1m != ct_f_2m):
            print 'Face pairs F1 and F2 M are not matched'
        check1 = f_1m[j]
        tx1, ty1, tz1 = x[check1 - 1],y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_2m)):
            check2 = f_2m[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tx1 - tx2) <= 1.0e-3) and (abs(ty1 - ty2) <= 1.0e-3)):
                ct_f_12m = ct_f_12m + 2
                fp_12m = fp_12m + [check1]
                fp_12m = fp_12m + [check2]
                x[check1 - 1] = x[check2 - 1]
                y[check1 - 1] = y[check2 - 1]
            k = k + 1
        j = j + 1
    del fp_12m[fp_12m.index(0)]
    if((ct_f_12m/ct_f_1m != 2) and (ct_f_12m/ct_f_2m != 2)):
        print '  Error in identifying face pairs -  Face 1 and Face 2 M '
    else:
        print '  Face pairs F1 and F2 M are identified'
####-------------------------------------------------
##    ## debugging routine...
##        j = 0
##        while j in range(len(f_5)):
##            check1 = f_5[j]
##            if check1 in fp_56:
##                pass
##            else:
##                srat = str(check1)+ '  Face -5 \n'
##                output_6.write(srat)
##            j = j + 1
##
##        j = 0
##        cnt = 0
##        while j in range(len(f_6)):
##            check1 = f_6[j]
##            if check1 in fp_56:
```

```
##                cnt = cnt + 1
##                pass
##            else:
##                srat = str(check1)+ '  Face -6 \n'
##                output_6.write(srat)
##            j = j + 1
####------------------------------------------------
    j, k = 0,0
    while j in range(len(f_3f)):
        if(ct_f_3f != ct_f_4f):
            print 'Face pairs F1 and F2 F are not matched'
        check1 = f_3f[j]
        tx1, ty1, tz1 = x[check1 - 1],y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_4f)):
            check2 = f_4f[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tx1 - tx2) <= shf) and (abs(tz1 - tz2) <= shf) and
(abs(ty1 + ty2) <= shf)):
                ct_f_34f = ct_f_34f + 2
                fp_34f = fp_34f + [check1]
                fp_34f = fp_34f + [check2]
            k = k + 1
        j = j + 1
    del fp_34f[fp_34f.index(0)]
    if((ct_f_34f/ct_f_3f != 2) and (ct_f_34f/ct_f_4f != 2)):
        print '  Error in identifying face pairs -  Face 3 and Face 4 F'
    else:
        print '  Face pairs F3 and F4 F are identified'
####------------------------------------------------
    j, k = 0,0
    while j in range(len(f_3m)):
        if(ct_f_3m != ct_f_4m):
            print 'Face pairs F1 and F2 M are not matched'
        check1 = f_3m[j]
        tx1, ty1, tz1 = x[check1 - 1],y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_4m)):
            check2 = f_4m[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tx1 - tx2) <= shf) and (abs(tz1 - tz2) <= shf) and
(abs(ty1 + ty2) <= shf)):
                ct_f_34m = ct_f_34m + 2
                fp_34m = fp_34m + [check1]
                fp_34m = fp_34m + [check2]
            k = k + 1
        j = j + 1
    del fp_34m[fp_34m.index(0)]
    if((ct_f_34m/ct_f_3m != 2) and (ct_f_34m/ct_f_4m != 2)):
        print '  Error in identifying face pairs -  Face 3 and Face 4 M'
    else:
        print '  Face pairs F3 and F4 M are identified'
####------------------------------------------------
    j, k = 0,0
    while j in range(len(f_5f)):
        if(ct_f_5f != ct_f_6f):
            print 'Face pairs F5 and F6 F are not matched'
```

```
        check1 = f_5f[j]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_6f)):
            check2 = f_6f[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tz1 - tz2) <= shf)  and (abs(ty1 - ty2) <= shf) ):
                ct_f_56f = ct_f_56f + 2
                fp_56f = fp_56f + [check1]
                fp_56f = fp_56f + [check2]
            k = k + 1
        j = j + 1
    del fp_56f[fp_56f.index(0)]
    if((ct_f_56f/ct_f_5f != 2) and (ct_f_56f/ct_f_6f != 2)):
        print '  Error in identifying face pairs -  Face 5 and Face 6 F'

    else:
        print '  Face pairs F5 and F6 F are identified'
####-----------------------------------------------------
    j, k = 0,0
    while j in range(len(f_5m)):
        if(ct_f_5m != ct_f_6m):
            print 'Face pairs F5 and F6 M are not matched'
        check1 = f_5m[j]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_6m)):
            check2 = f_6m[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tz1 - tz2) <= shf)  and (abs(ty1 - ty2) <= shf) ):
                ct_f_56m = ct_f_56m + 2
                fp_56m = fp_56m + [check1]
                fp_56m = fp_56m + [check2]
            k = k + 1
        j = j + 1
    del fp_56m[fp_56m.index(0)]
    if((ct_f_56m/ct_f_5m != 2) and (ct_f_56m/ct_f_6m != 2)):
        print '  Error in identifying face pairs -  Face 5 and Face 6 M'

    else:
        print '  Face pairs F5 and F6 M are identified'

##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
def center_constraints():
    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(2) + '\n'
        strgg2= ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3
        output_1.write(strgg)
        output_2.write(strgg)
        output_3.write(strgg)
        output_4.write(strgg)
        output_5.write(strgg)
```

```
        output_11.write(strgg)
        output_12.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(2) + '\n'
        strgg2 = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3
        output_1.write(strgg)
        output_2.write(strgg)
        output_3.write(strgg)
        output_4.write(strgg)
        output_5.write(strgg)
        output_11.write(strgg)
        output_12.write(strgg)

        m = m + 1

    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(2) + '\n'
        strgg2 = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3
        output_1.write(strgg)
        output_2.write(strgg)
        output_3.write(strgg)
        output_4.write(strgg)
        output_5.write(strgg)
        output_11.write(strgg)
        output_12.write(strgg)
        m = m + 1

    print '  Face Center Constraints are written'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def corner_constraints():
    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(4) + '\n'
        strgg2 = ' ' + str(c146)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) +
'\n'
        strgg4 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg5 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4 + strgg5
        output_1.write(strgg)
        output_2.write(strgg)
        output_3.write(strgg)
        output_4.write(strgg)
        output_5.write(strgg)
        output_11.write(strgg)
```

```
        output_12.write(strgg)

        m = m + 1

    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(4) + '\n'
        strgg2 = ' ' + str(c236)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) +
'\n'
        strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg5 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4 + strgg5
        output_1.write(strgg)
        output_2.write(strgg)
        output_3.write(strgg)
        output_4.write(strgg)
        output_5.write(strgg)
        output_11.write(strgg)
        output_12.write(strgg)

        m = m + 1

    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(4) + '\n'
        strgg2 = ' ' + str(c245)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) +
'\n'
        strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg5 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4 + strgg5
        output_1.write(strgg)
        output_2.write(strgg)
        output_3.write(strgg)
        output_4.write(strgg)
        output_5.write(strgg)
        output_11.write(strgg)
        output_12.write(strgg)

        m = m + 1

    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(3) + '\n'
        strgg2 = ' ' + str(c145)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) +
'\n'
        strgg4 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
        output_1.write(strgg)
        output_2.write(strgg)
        output_3.write(strgg)
        output_4.write(strgg)
```

```
        output_5.write(strgg)
        output_11.write(strgg)
        output_12.write(strgg)

        m = m + 1

    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(3) + '\n'
        strgg2 = ' ' + str(c235)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) +
'\n'
        strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
        output_1.write(strgg)
        output_2.write(strgg)
        output_3.write(strgg)
        output_4.write(strgg)
        output_5.write(strgg)
        output_11.write(strgg)
        output_12.write(strgg)

        m = m + 1

    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(5) + '\n'
        strgg2 = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) +
'\n'
        strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg5 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg6 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4 + strgg5 + strgg6
        output_1.write(strgg)
        output_2.write(strgg)
        output_3.write(strgg)
        output_4.write(strgg)
        output_5.write(strgg)
        output_11.write(strgg)
        output_12.write(strgg)

        m = m + 1

    m = 0
    while m in range(3):
        strgg0 = '*Equation\n'
        strgg1 = str(3) + '\n'
        strgg2 = ' ' + str(c136)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        strgg3 = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) +
'\n'
        strgg4 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
        output_1.write(strgg)
        output_2.write(strgg)
```

```
        output_3.write(strgg)
        output_4.write(strgg)
        output_5.write(strgg)
        output_11.write(strgg)
        output_12.write(strgg)

        m = m + 1

    print '  Corner Constraints are written '
#-------------------------------------------------------------------------
## this routine is to write the equations that relate the faces 3 and 4
## which has nothing to do with the cracks we are going to incorporate.

def face_constraints_3_5_1f():
##-----------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_34f)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(fp_34f[k+1])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(fp_34f[k])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            strgg4 = ' ' + str(c_3)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 3 and 4 F are written'
##-------------------------------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_34m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(fp_34m[k+1])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(fp_34m[k])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            strgg4 = ' ' + str(c_3)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
```

```
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 3 and 4 M are written'
##-----------------------------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_56f)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(fp_56f[k+1])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(fp_56f[k])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            strgg4 = ' ' + str(c_5)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 5 and 6 F are written'
##-------------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_56m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(fp_56m[k+1])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(fp_56m[k])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            strgg4 = ' ' + str(c_5)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 5 and 6 M are written'
##-----------------------------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_12f)):
        m = 0
```

```
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(fp_12f[k+1])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(fp_12f[k])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 1 and 2 F are written'

##-----------------------------------------------------------------------------
def edge_constraints_35_36_45_46():
    k,m = 0,0
    while k in range(len(e_3635)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_3635[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_3635[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 36 and 35'

    k,m = 0,0
    while k in range(len(e_4535)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_4535[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_4535[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
```

```
            strgg4 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 45 and 35'

    k,m = 0,0
    while k in range(len(e_4635)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(4) + '\n'
            strgg2 = ' ' + str(e_4635[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_4635[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg5 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4 + strgg5
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 46 and 35'
##-----------------------------------------------------------------------------
def edge_constraints_13_14_23_24_f():
    k,m = 0,0
    while k in range(len(e_1413f)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_1413f[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_1413f[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
```

```
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 14 and 13 F'

    k,m = 0,0
    while k in range(len(e_2313f)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_2313f[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2313f[k+1])+',' +  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_1)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 23 and 13 F'

    k,m = 0,0
    while k in range(len(e_2413f)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(4) + '\n'
            strgg2 = ' ' + str(e_2413f[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2413f[k+1])+',' +  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_3)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            strgg5 = ' ' + str(c_1)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4 +strgg5
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 24 and 13 F'
```

```
##-------------------------------------------------------------------------
def edge_constraints_15_25_16_26_f():
    k,m = 0,0
    while k in range(len(e_1615f)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_1615f[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_1615f[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 16 and 15 F'

    k,m = 0,0
    while k in range(len(e_2615f)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(4) + '\n'
            strgg2 = ' ' + str(e_2615f[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2615f[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg5 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4  + strgg5
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 26 and 15 F'

    k,m = 0,0
    while k in range(len(e_2515f)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
```

```
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_2515f[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2515f[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 25 and 15 F'
##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
def edge_constraints_13_14_23_24_nf():
    k,m = 0,0
    while k in range(len(e_1413m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_1413m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_1413m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 14 and 13 M NF'

    k,m = 0,0
    while k in range(len(e_2313m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_2313m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2313m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
```

```
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 23 and 13 M NF'

    k,m = 0,0
    while k in range(len(e_2413m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(4) + '\n'
            strgg2 = ' ' + str(e_2413m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2413m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg5 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4 +strgg5
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 24 and 13 M NF'
##-------------------------------------------------------------------------
def edge_constraints_15_25_16_26_nf():
    k,m = 0,0
    while k in range(len(e_1615m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_1615m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_1615m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
```

```
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 16 and 15 M NF'

    k,m = 0,0
    while k in range(len(e_2615m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(4) + '\n'
            strgg2 = ' ' + str(e_2615m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2615m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg5 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4  + strgg5
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 26 and 15 M NF'

    k,m = 0,0
    while k in range(len(e_2515m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_2515m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2515m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 25 and 15 M NF'
##-----------------------------------------------------------------------
def edge_constraints_13_14_23_24_15_25_26_16m():
```

```
    k,m = 0,0
    while k in range(len(e_1413m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_1413m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_1413m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 14 and 13 M F'

    k,m = 0,0
    while k in range(len(e_2423m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_2423m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2423m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 23 and 24 M F'

    k,m = 0,0
    while k in range(len(e_1615m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_1615m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
```

```
            strgg3 = ' ' + str(e_1615m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 16 and 15 M F'

    k,m = 0,0
    while k in range(len(e_2625m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(e_2625m[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(e_2625m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            strgg4 = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 26 and 25 M F'

##-------------------------------------------------------------------------
def face_constraints_1m():
##-------------------------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_12m)):
        m = 0
        while m in range(3):
            strgg0 = '*Equation\n'
            strgg1 = str(3) + '\n'
            strgg2 = ' ' + str(fp_12m[k+1])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            strgg3 = ' ' + str(fp_12m[k])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            strgg4 = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            strgg = strgg0 + strgg1 + strgg2 + strgg3 + strgg4
            output_1.write(strgg)
```

```
            output_2.write(strgg)
            output_3.write(strgg)
            output_4.write(strgg)
            output_5.write(strgg)
            output_11.write(strgg)
            output_12.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 1 and 2 M are written'
##-----------------------------------------------------------------------------
def rigid_body_modes():
    strgg0 ='** Boundary conditions written\n'
    strgg1 ='*Boundary\n'
##    if(case == 3):
##        strgg2 = ' n_set_5'+','+' '+str(1) + ','+ ' '+ str(3)+'\n'
##        strgg3 = ' n_set_3'+','+' '+str(3) + ','+ ' '+ str(3)+'\n'
##        strgg4 = ' n_set_1'+','+' '+str(2) + ','+ ' '+ str(2)+'\n'
##        strgg5 = ' n_set_3'+','+' '+str(2) + ','+ ' '+ str(2)+'\n'
##    else:
    strgg2 = ' n_set_5'+','+' '+str(1) + ','+ ' '+ str(3)+'\n'
    strgg3 = ' n_set_1'+','+' '+str(1) + ','+ ' '+ str(1)+'\n'
    strgg4 = ' n_set_1'+','+' '+str(2) + ','+ ' '+ str(2)+'\n'
    strgg5 = ' n_set_3'+','+' '+str(2) + ','+ ' '+ str(2)+'\n'

    strgg = strgg0 + strgg1 +strgg2 + strgg3 + strgg4 + strgg5
    output_1.write(strgg)
    output_2.write(strgg)
    output_3.write(strgg)
    output_4.write(strgg)
    output_5.write(strgg)
    output_11.write(strgg)
    output_12.write(strgg)
##-----------------------------------------------------------------------------
def loads():
    strgg0 = '*Cload \n'
    strgg00 = '*Temperature \n'
    strgg010 = '*Boundary  \n'

##  E11 simulation
    strgg1 = ' n_set_1'+','+' '+ str(3) + ','+ ' '+ str(1000.0)+'\n'
    strgg = strgg0 + strgg1
    output_1.write(strgg)

##  E22 simulation
    strgg1 = ' n_set_3'+','+' '+ str(1) + ','+ ' '+ str(1000.0)+'\n'
    strgg = strgg0 + strgg1
    output_2.write(strgg)

##  G12 similation
    strgg1 = ' n_set_3'+','+' '+ str(3) + ','+ ' '+ str(1000.0)+'\n'
    strgg = strgg0 + strgg1
    output_3.write(strgg)

##  G23 simulation
    strgg1 = ' n_set_4'+','+' '+ str(1) + ','+ ' '+ str(1000.0)+'\n'
    strgg = strgg0 + strgg1
    output_4.write(strgg)
```

```
##  G13 - simulation
    strgg1 = ' n_set_4'+','+' '+ str(3) + ','+ ' '+ str(1000.0)+'\n'
    strgg = strgg0 + strgg1
    output_11.write(strgg)

##  E33 - simulation
    strgg1 = ' n_set_4'+','+' '+ str(2) + ','+ ' '+ str(1000.0)+'\n'
    strgg = strgg0 + strgg1
    output_12.write(strgg)

##  Thermal simulation
    strgg1 = ' n_set_9' + ','+' '+ str(-200)+'\n'
    strgg = strgg00 + strgg1
    output_5.write(strgg)
    print 'Case 5 - Thermal Loading Only'

##-------------------------------------------------------------------------
## the subroutine that arranges the depth sets
def centroids():
    global fiber_elem, matrix_elem
    global z_depth, elem_z, s_z_elems
    global z_depthm, elem_zm, s_z_elemsm
    fiber_elem, matrix_elem = [0], [0]

    num = 0
    while num in range(len(num_el)):
        check = num_el[num]
        tx1, tx2 = x[node1[num]-1], x[node2[num]-1]
        tx3, tx4 = x[node3[num]-1], x[node4[num]-1]
        tx5, tx6 = x[node5[num]-1], x[node6[num]-1]
        tx7, tx8 = x[node7[num]-1], x[node8[num]-1]

        ty1, ty2 = y[node1[num]-1], y[node2[num]-1]
        ty3, ty4 = y[node3[num]-1], y[node4[num]-1]
        ty5, ty6 = y[node5[num]-1], y[node6[num]-1]
        ty7, ty8 = y[node7[num]-1], y[node8[num]-1]

        tx_cen = (tx1 + tx2 + tx3+ tx4  + tx5 + tx6 + tx7 + tx8)/8.0
        ty_cen = (ty1 + ty2 + ty3+ ty4  + ty5 + ty6 + ty7 + ty8)/8.0

        radi_c1=sqrt((ty_cen-ytestc1)**2+(tx_cen-xtestc1)**2)
        radi_c2=sqrt((ty_cen-ytestc2)**2+(tx_cen-xtestc2)**2)
        radi_c3=sqrt((ty_cen-ytestc3)**2+(tx_cen-xtestc3)**2)
        radi_c4=sqrt((ty_cen-ytestc4)**2+(tx_cen-xtestc4)**2)
        radi_center=sqrt((ty_cen-ytestfcen1)**2+(tx_cen-xtestfcen1)**2)

        if((abs(radi_c1)<radi)or(abs(radi_c2)<radi) or

(abs(radi_c3)<radi)or(abs(radi_c4)<radi)or(abs(radi_center<radi))):
            fiber_elem = fiber_elem + [check]
        else:
            matrix_elem = matrix_elem + [check]


        num = num + 1
    del fiber_elem[fiber_elem.index(0)]
```

```
del matrix_elem[matrix_elem.index(0)]


z_depth = [0]
elem_z = [0]

num = 0
for num in range(len(fiber_elem)):
    check = fiber_elem[num]
    tz1, tz2 = z[node1[check-1]-1], z[node2[check-1]-1]
    tz3, tz4 = z[node3[check-1]-1], z[node4[check-1]-1]
    tz5, tz6 = z[node5[check-1]-1], z[node6[check-1]-1]
    tz7, tz8 = z[node7[check-1]-1], z[node8[check-1]-1]
    tz_cen = (tz1 + tz2 + tz3+ tz4  + tz5 + tz6 + tz7 + tz8)/8.0
    tz_cenn = round(tz_cen, 3)
    elem_z = elem_z + [[tz_cenn, check]]
    if tz_cenn in z_depth:
        pass
    else:
        z_depth = z_depth + [tz_cenn]

del elem_z[elem_z.index(0)]
del z_depth[z_depth.index(0)]

z_depth.sort()
s_z_elems = [0]
for ij in range(len(z_depth)):
    name = 'Set_fiber_' + str(ij+1)
    s_z_elems = s_z_elems + [[name, z_depth[ij]]]

del s_z_elems[s_z_elems.index(0)]

ij = 0
while ij in range(len(elem_z)):
    check = elem_z[ij][1]
    tzz = elem_z[ij][0]
    for ik in range(len(s_z_elems)):
        tzc = s_z_elems[ik][1]
        if(tzz == tzc):
            s_z_elems[ik]=s_z_elems[ik] + [check]
    ij = ij + 1

z_depthm = [0]
elem_zm = [0]

num = 0
while num in range(len(matrix_elem)):
    check = matrix_elem[num]
    tz1, tz2 = z[node1[check-1]-1], z[node2[check-1]-1]
    tz3, tz4 = z[node3[check-1]-1], z[node4[check-1]-1]
    tz5, tz6 = z[node5[check-1]-1], z[node6[check-1]-1]
    tz7, tz8 = z[node7[check-1]-1], z[node8[check-1]-1]
    tz_cen = (tz1 + tz2 + tz3+ tz4  + tz5 + tz6 + tz7 + tz8)/8.0
    tz_cenn = round(tz_cen, 3)
    elem_zm = elem_zm + [[tz_cenn, check]]

    if tz_cenn in z_depthm:
```

```
            pass
        else:
            z_depthm = z_depthm + [tz_cenn]
        num = num + 1
    del elem_zm[elem_zm.index(0)]
    del z_depthm[z_depthm.index(0)]

    z_depthm.sort()
    s_z_elemsm = [0]
    for ij in range(len(z_depthm)):
        name = 'Set_matrix_' + str(ij+1)
        s_z_elemsm = s_z_elemsm + [[name, z_depthm[ij]]]

    del s_z_elemsm[s_z_elemsm.index(0)]

    ij = 0
    while ij in range(len(elem_zm)):
        check = elem_zm[ij][1]
        tzz = elem_zm[ij][0]
        for ik in range(len(s_z_elemsm)):
            tzc = s_z_elemsm[ik][1]
            if(tzz == tzc):
                s_z_elemsm[ik]=s_z_elemsm[ik] + [check]
        ij = ij + 1

##-----------------------------------------------------------------------------
from abaqus import *
from abaqusConstants import *
from section import*
import math
reading_inp_file()
```

APPENDIX – B

PYTHON CODE FOR MPC UNIT CELL MODEL

In this appendix, a computer code written in the *Python* object oriented programming language that is intended to run with ABAQUS Version 6.6 is listed. The code consists of four python script files, namely, "*run_ninety.py*", "*model_ninety.py*", "*bc_ninety.py*", and "*bc_ninety_23.py*". The entire script is run by executing the statement "**execfile('run_ninety.py')**" in an ABAQUS command window. This runs the script file "*run_mnc.py*" which executes the other files based on the loading scenario. For imposing an out-of-plane shear stress on the laminate, the files "*model_ninety.py*" and "*bc_ninety_23.py*" are executed in sequence. For all other loading scenarios, the files "*model_ninety.py*" and "*bc_ninety.py*" are executed in sequence.

The file "*run_ninety.py*" is the central file that controls the creation and execution of the model. The file has the parameters such as, the material definition, laminate sequence, crack density or crack spacing, etc. that are required to define the model. With the required parameters defined, the file "*model_ninety.py*" is executed which creates the laminate model in ABAQUS and produces an input file. This input file has all node and element information defining the model. The file "*bc_ninety.py*" modifies the input file by adding the periodic boundary conditions or kinematic constraints based on periodicity in the plane of the laminate for the nodes in the models. The file "*bc_ninety_23.py*" modifies the input file by adding the periodic boundary conditions or kinematic constraints based on periodicity in all the three Cartesian coordinate directions of the laminate for the nodes in the models. The model is then run in ABAQUS using the input file by "*run_ninety.py*".

Central program: ***run_ninety.py***

```
## THIS PROGRAM IS THE CENTRAL PROGRAM THAT RUNS ALL OTHER
## PROGRAMS to EVALUATE THE CRACKED LAMINA PROPERTIES.
#-------------------------------------------------------------
from abaqus import *
from abaqusConstants import *
import math

global layup, theta, i, ped, nol, t, zeta
global qconst, a, b, loc_ply_1, thickness
global thick_location, thick_midpoints
global lumped_inner, outerlayer, thick
global mechanical, thermal, mat_name, materia, load_case

## Loading on the model.
mechanical = 0 ## one - yes, zero - no
thermal = 1 ## one - yes, zero - no
## thermo-mechanical loading -- load and thermal = 1
## NOTE: If mechanical loading is specified, then the case needs to be
specified.
## note further that these loading are on the laminate and not on the cracked
ply.
## case = 1 => biaxial loading,
## case = 2 => inplane shear loading
```

```
## case = 3 => out-ofplane shear S23
if (mechanical == 1):
    load_case = 1
else:
    load_case = 0

if(thermal == 1):
    theta_range = -196 ## temperature range
##-----------------------------------------------------
pi = acos(-1.0)
## layup of the laminate
layup = [0, 90, 90, 0]
## material for the layers
materia = [ 3, 3, 3, 3]
## individual thicknesses of the plies
t_uc = 0.15/1000.0 ## thickness of the uncracked plies in METERS.
ratio = 1.0   ## ratio of the cracked ply thickness and one ply thickness.
ratioo = 1.0  ## ratio of the uncracked ply thickness and one ply thickness.
t_c = t_uc * ratio  ## thickness of the cracked plies.
t_ucc = t_uc * ratioo  ## thickness of the uncracked plies.
thickness = [t_ucc, t_c, t_c, t_ucc]
## the cracked layer
## note: remember that the range is from 0 -> (max-1)
## if symmetric cracking, then you need to specify only one number (the
first)
i = 1
crack_layer_thickness = thickness[i]
theta = layup[i]
## total thickness of the laminate
ped = 0
for k in range(len(thickness)):
    ped = ped + thickness[k]
## numer of layers
nol = len(layup)
loc_ply_1 = i + 1   # location of the cracked ply from top
##-------------------------------------------------------
thick_location = [0]
thick_midpoints = [0]
increm = 0
increm3 = 0
for inte in range(len(thickness)):
    increm = increm + thickness[inte]
    thick_location = thick_location + [increm]
    if(inte == 0):
        increm2 = increm/2.0
    else:
        increm3 = increm3 + thickness[inte-1]
        increm2 = (increm + increm3)/2.0
    thick_midpoints = thick_midpoints + [increm2]
del thick_location[thick_location.index(0)]
del thick_midpoints[thick_midpoints.index(0)]
##-------------------------------------------------------

## Tripper if the layer is an outer layer.
outerlayer = 0
if(i == 0):
    outerlayer = 1
```

```
## Tripper if the inner layers are lumped.
lumped_inner = 0
locaaa = nol - i - 2
if(i == locaaa):
    lumped_inner = 1
thick = [0]
thick = thick + thick_location

##-------------------------------------------------------
## crack density
c_d = 1.2 * 1000 # expressed in cracks/METERS
## normalized crack density > 0
if(lumped_inner == 1):
    zeta = c_d * thickness[i]
else:
    zeta = c_d * thickness[i]

##-------------------------------------------------------
## material database
## NOTE THAT LAYUP IS FROM THE BOTTOM OF THE LAMINATE
mat_name = [0]

## material - 1
NAME = 'P3051-F05'
temp = [1, NAME]
E11, E22, E33 =  127.8e9, 9.4e9, 9.4e9
G12, G13, G23 =  4.2e9, 4.2e9, 3.1e9
NU12, NU13, NU23 = 0.28, 0.28, 0.51613
A11, A22, A33 = 17.628e-6, 20.61E-6, 20.61E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## material - 2
NAME = 'AS4-3502'
temp = [2, NAME]
E11, E22, E33 =  144.78e9, 9.58e9, 9.58e9
G12, G13, G23 =  4.785e9, 4.785e9, 3.4214e9
NU12, NU13, NU23 = 0.31, 0.31, 0.4
A11, A22, A33 = 8.629e-6, 26.15E-6, 26.15E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## material - 3
NAME = 'Glass/epoxy'
temp = [3, NAME]
E11, E22, E33 =  44.73e9, 12.76e9, 12.76e9
G12, G13, G23 =  5.8e9, 5.8e9, 4.49e9
NU12, NU13, NU23 = 0.297, 0.297, 0.4209
A11, A22, A33 = 8.6e-6, 22.1E-6, 22.1E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## material - 4
NAME = 'Silenka'
temp = [4, NAME]
E11, E22, E33 =  45.763e9, 16.167e9, 16.167e9
G12, G13, G23 =  5.8626e9, 5.8626e9, 5.7415e9
```

```
NU12, NU13, NU23 = 0.2786, 0.2786, 0.40799
A11, A22, A33 = 8.629e-6, 26.152E-6, 26.152E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## material - 5
NAME = 'T300'
temp = [5, NAME]
E11, E22, E33 =  138e9, 11e9, 11e9
G12, G13, G23 =  5.5e9, 5.5e9, 3.9286e9
NU12, NU13, NU23 = 0.28, 0.28, 0.4
A11, A22, A33 = 8.6e-6, 22.1E-6, 22.1E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## material - 6
NAME = 'HIGHS'
temp = [6, NAME]
E11, E22, E33 =  41.7e9, 13e9, 13e9
G12, G13, G23 =  3.4e9, 3.4e9, 4.6429e9
NU12, NU13, NU23 = 0.3, 0.3, 0.4
A11, A22, A33 = 8.6e-6, 22.1E-6, 22.1E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## if you need to add more materials then add before this line.
## end of database
del mat_name[mat_name.index(0)]

##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
GO = 0
print ' ---------- creating the laminate ---------------'

if(theta == 90):
## seeding controls
    cdmm = c_d/1000

    if (load_case == 3):
        if(ratio == 5):
            if (cdmm == 0.16):
                seed_1 = 60
            elif (cdmm == 0.32):
                seed_1 = 30
            elif (cdmm == 0.48):
                seed_1 =  20
            elif (cdmm == 0.64):
                seed_1 =  15
        else:
            if (cdmm == 0.4):
                seed_1 = 24
            elif (cdmm == 0.8):
                seed_1 = 12
            elif (cdmm == 1.2):
                seed_1 =  8
            elif (cdmm == 1.6):
                seed_1 =  6
```

```
    else:
        if(ratio == 5):
            if (cdmm == 0.16):
                seed_1 = 40
            elif (cdmm == 0.32):
                seed_1 = 20
            elif (cdmm == 0.48):
                seed_1 = 15
            elif (cdmm == 0.64):
                seed_1 = 10
        else:
            if (cdmm == 0.4):
                seed_1 = 16
            elif (cdmm == 0.8):
                seed_1 = 8
            elif (cdmm == 1.2):
                seed_1 = 6
            elif (cdmm == 1.6):
                seed_1 = 4

    seed_5 = int(4 * ratioo) ## seeding for uncracked layers thickness
    if(load_case == 3):
        seed_6 = int(4  * ratio) ## seeding for cracked layers.
    else:
        seed_6 = int(4  * ratio)

## length of the layer.
    a = thickness[i]/zeta # crack spacing and breadth of the layer
    b = a/10.0
## NOTE: Here 'b' is ARBITRARY
    exec open('C:/model_ninety.py')
    ## again the path needs to be specified.
    print 'Input file created.'
    print '                 '
    print 'Opening the Input file'
    if(load_case == 3):
        loadcase_3_mag_1 = 3E6 * 2 * b * ped  ## for inner and centered
plies.
        loadcase_3_mag_2 = 3E6 * a * b  ## for outer plies.
        print '        '
        print 'This is for out-of-plane 23 SHEAR Loading only'
        exec open('C:/bc_ninety_23.py')
    else:
        print '   '
        print 'This is for INPLANE loading'
        exec open('C:/bc_ninety.py')
    GO = 1

##_____________________________________________________________________________
if (GO == 1):
    print '               '
    print 'Runnning the File'
    laminatemodel = mdb.JobFromInputFile(name = 'Run_case_2', inputFileName =
'runfile.inp')
    laminatemodel.submit()
    laminatemodel.waitForCompletion()
```

```
    statuss = laminatemodel.status
    print 'THE ANALYSIS IS', statuss

    if(statuss == COMPLETED):
        AA = session.openOdb('C:/Krishna/MTC_ninety/Run_case_2.odb')
        session.viewports['Viewport: 1'].setValues(displayedObject=AA)
        session.viewports['Viewport: 1'].odbDisplay.setDeformedVariable(
variableLabel='U')
        session.viewports['Viewport: 1'].odbDisplay.setPrimaryVariable(
            variableLabel='S', outputPosition=INTEGRATION_POINT,
refinement=(COMPONENT, 'S11'))

else:
    print 'Only Modeling'
```

Code for building the model: ***model_ninety.py***

```
###
#-------------------------------------------------------
Dx, Cx, Ax, Bx = -a/2.0,  a/2.0, -a/2.0,  a/2.0
Dy, Cy, Ay, By =  b/2.0,  b/2.0, -b/2.0, -b/2.0

Ex, Fx, Gx, Hx = -a/2.0,  a/2.0, -a/2.0,  a/2.0
Ey, Fy, Gy, Hy =    0.0,    0.0, -b/4.0, -b/4.0

Ix, Jx, Kx, Lx = -a/2.0,  a/2.0, -a/4.0,    0.0
Iy, Jy, Ky, Ly =  b/4.0,  b/4.0, -b/4.0, -b/4.0

Mx, Nx, Ox, Px =  a/4.0,  a/4.0,  a/4.0,    0.0
My, Ny, Oy, Py = -b/4.0,    0.0,  b/4.0,  b/4.0

Qx, Qy = -a/4.0, b/4.0

Rx, Sx, Tx, Ux = -a/4.0,    0.0, -a/4.0,    0.0
Ry, Sy, Ty, Uy =    0.0,    0.0, -b/2.0, -b/2.0

Vx, Wx, Xx, Yx =  a/4.0,  a/4.0,    0.0, -a/4.0
Vy, Wy, Xy, Yy = -b/2.0,  b/2.0,  b/2.0,  b/2.0


### mid points of the corners

ATx, ATy  = (Ax + Tx)/2.0, (Ay + Ty)/2.0
TUx, TUy  = (Tx + Ux)/2.0, (Ty + Uy)/2.0
UVx, UVy  = (Vx + Ux)/2.0, (Vy + Uy)/2.0
VBx, VBy  = (Vx + Bx)/2.0, (Vy + By)/2.0
BHx, BHy  = (Hx + Bx)/2.0, (Hy + By)/2.0
HFx, HFy  = (Hx + Fx)/2.0, (Hy + Fy)/2.0
FJx, FJy  = (Jx + Fx)/2.0, (Jy + Fy)/2.0
JCx, JCy  = (Jx + Cx)/2.0, (Jy + Cy)/2.0
CWx, CWy  = (Wx + Cx)/2.0, (Wy + Cy)/2.0
WXx, WXy  = (Wx + Xx)/2.0, (Wy + Xy)/2.0
XYx, XYy  = (Yx + Xx)/2.0, (Yy + Xy)/2.0
YDx, YDy  = (Yx + Dx)/2.0, (Yy + Dy)/2.0
DIx, DIy  = (Ix + Dx)/2.0, (Iy + Dy)/2.0
IEx, IEy  = (Ix + Ex)/2.0, (Iy + Ey)/2.0
EGx, EGy  = (Gx + Ex)/2.0, (Gy + Ey)/2.0
```

```
GAx, GAy  = (Gx + Ax)/2.0, (Gy + Ay)/2.0

QIx, QIy  = (Ix + Qx)/2.0, (Iy + Qy)/2.0
REx, REy  = (Rx + Ex)/2.0, (Ry + Ey)/2.0
KGx, KGy  = (Gx + Kx)/2.0, (Gy + Ky)/2.0

LKx, LKy  = (Lx + Kx)/2.0, (Ly + Ky)/2.0
SRx, SRy  = (Sx + Rx)/2.0, (Ry + Sy)/2.0
PQx, PQy  = (Px + Qx)/2.0, (Py + Qy)/2.0

MLx, MLy  = (Mx + Lx)/2.0, (My + Ly)/2.0
NSx, NSy  = (Nx + Sx)/2.0, (Ny + Sy)/2.0
OPx, OPy  = (Ox + Px)/2.0, (Oy + Py)/2.0

HMx, HMy  = (Hx + Mx)/2.0, (Hy + My)/2.0
FNx, FNy  = (Fx + Nx)/2.0, (Fy + Ny)/2.0
JOx, JOy  = (Jx + Ox)/2.0, (Jy + Oy)/2.0

YQx, YQy  = (Yx + Qx)/2.0, (Yy + Qy)/2.0
QRx, QRy  = (Qx + Rx)/2.0, (Qy + Ry)/2.0
RKx, RKy  = (Rx + Kx)/2.0, (Ry + Ky)/2.0
KTx, KTy  = (Kx + Tx)/2.0, (Ky + Ty)/2.0

XPx, XPy  = (Xx + Px)/2.0, (Xy + Py)/2.0
PSx, PSy  = (Px + Sx)/2.0, (Py + Sy)/2.0
SLx, SLy  = (Sx + Lx)/2.0, (Sy + Ly)/2.0
LUx, LUy  = (Lx + Ux)/2.0, (Ly + Uy)/2.0

WOx, WOy  = (Wx + Ox)/2.0, (Wy + Oy)/2.0
ONx, ONy  = (Ox + Nx)/2.0, (Oy + Ny)/2.0
NMx, NMy  = (Nx + Mx)/2.0, (Ny + My)/2.0
MVx, MVy  = (Mx + Vx)/2.0, (My + Vy)/2.0

##-----------------------------------------------------------------
#-----------------------------------------------------------------
lammodel = mdb.Model(name='Laminate Model')

def lami_part():
    print' Creating the Extruded Part'
# coordinates of the verticess
    rectcd = ((a/2.0,b/2.0), (-a/2.0,b/2.0), (-a/2.0,-b/2.0), (a/2.0,-b/2.0),
(0.0,0.0))
    s = lammodel.Sketch(name='lamin', sheetSize = 200.0)
# creating the rectangle.
    s.rectangle(point1=rectcd[1], point2=rectcd[3])
    hexpart = lammodel.Part(name='lami', dimensionality=THREE_D,
      type=DEFORMABLE_BODY)
    hexpart.BaseSolidExtrude(sketch=s, depth=ped)
# viewport options
    myViewport = session.Viewport(name='Viewport for Laminate Model',
        origin=(10, 10), width=150, height=100)
    myViewport.setValues(displayedObject=hexpart)
    myViewport.partDisplay.setValues(renderStyle=SHADED)
    print'Part created from Extrusion'
#-----------------------------------------------------------------
def lami_parti():
    p2 = lammodel.parts['lami']
```

```
    for inte in range(len(layup)-1):
        increm = thick_location[inte]
        temp_6 = p2.DatumPlaneByThreePoints(point1=(0.0,0.0,increm),
point2=(a/2.0,0.0,increm), point3=(0.0,b/2.0,increm))
        datams = temp_6.id
        call = p2.cells
        ccall = call.findAt((0.,0.,thick_midpoints[inte]))
        p2.PartitionCellByDatumPlane(cells=ccall,
datumPlane=p2.datums[datams])

    print 'The cells are partitioned for layers'
#-------------------------------------------------------------------
def lami_parti_2():
    p2 = lammodel.parts['lami']
    c = p2.cells
    f = p2.faces
    e = p2.edges
# creating the partition for seam in various layers
    temp_6 = p2.DatumPlaneByThreePoints(point1=(0.0,-b/2.0,ped),
point2=(0.0,-b/2.0,0.0), point3=(0.0,b/2.0,ped))
    datum_center = temp_6.id
    call = p2.cells
    p2.PartitionCellByDatumPlane(cells=call,
datumPlane=p2.datums[datum_center])

#-------------------------------------------------------------------
def lami_parti_3():
    p2 = lammodel.parts['lami']
    c = p2.cells
    f = p2.faces
    e = p2.edges

    temp_6 = p2.DatumPlaneByThreePoints(point1=(0.0,b/4.0,ped),
point2=(0.0,b/4.0,0.0), point3=(a/2.0,b/4.0,ped))
    datum_center = temp_6.id
    call = p2.cells
    p2.PartitionCellByDatumPlane(cells=call,
datumPlane=p2.datums[datum_center])

    temp_6 = p2.DatumPlaneByThreePoints(point1=(0.0,-b/4.0,ped),
point2=(0.0,-b/4.0,0.0), point3=(a/2.0,-b/4.0,ped))
    datum_center = temp_6.id
    call = p2.cells
    p2.PartitionCellByDatumPlane(cells=call,
datumPlane=p2.datums[datum_center])

    temp_6 = p2.DatumPlaneByThreePoints(point1=(-a/4.0,-b/4.0,ped), point2=(-
a/4.0,-b/4.0,0.0), point3=(-a/4.0,b/4.0,ped))
    datum_center = temp_6.id
    call = p2.cells
    p2.PartitionCellByDatumPlane(cells=call,
datumPlane=p2.datums[datum_center])

    temp_6 = p2.DatumPlaneByThreePoints(point1=(a/4.0,b/4.0,ped),
point2=(a/4.0,b/4.0,0.0), point3=(a/4.0,-b/4.0,ped))
    datum_center = temp_6.id
    call = p2.cells
```

```
    p2.PartitionCellByDatumPlane(cells=call,
datumPlane=p2.datums[datum_center])

    temp_6 = p2.DatumPlaneByThreePoints(point1=(a/2.0,0.0,ped),
point2=(a/2.0,0.0,0.0), point3=(0.0,0.0,ped))
    datum_center = temp_6.id
    call = p2.cells
    p2.PartitionCellByDatumPlane(cells=call,
datumPlane=p2.datums[datum_center])

## ------------------------------------------------------------
## material properties of the plies.
## material - 1
def lami_mater():
    num_mat = 0
## creating lamina material
    for inte in range(len(mat_name)):
        algraphiteepoxy=lammodel.Material(name=mat_name[inte][1])
        propes = (mat_name[inte][2], mat_name[inte][3], mat_name[inte][4],
                  mat_name[inte][5], mat_name[inte][6], mat_name[inte][7],
                  mat_name[inte][8], mat_name[inte][9], mat_name[inte][10])
        algraphiteepoxy.Elastic(table=(propes, ), type=ENGINEERING_CONSTANTS)
        CTEprop = (mat_name[inte][11], mat_name[inte][12],
mat_name[inte][13])
        algraphiteepoxy.Expansion(type=ORTHOTROPIC, table=(CTEprop,))
        num_mat = num_mat + 1
    print num_mat, '- Materials created'
#------------------------------------------------------------
#------------------------------------------------------------
def lami_section_1():
    p2 = lammodel.parts['lami']
    c = p2.cells

    for inte in range(len(layup)):
        floc_3 = thick_midpoints[inte]
        theta_1 = layup[inte]
        name_1 = 'CS_'+str(inte)

        if(theta < 0):
            x1 = abs(a/2.0)
            y1 = 0.0
            x2 = 0.0
            y2 = abs(b/2.0)
        else:
            x1 = a/2.0
            y1 = 0.0
            x2 = 0.0
            y2 = b/2.0

        y1p = x1 * sin(theta_1*pi/180) + y1 * cos(theta_1*pi/180)
        x1p = x1 * cos(theta_1*pi/180) - y1 * sin(theta_1*pi/180)

        y2p = x2 * sin(theta_1*pi/180) + y2 * cos(theta_1*pi/180)
        x2p = x2 * cos(theta_1*pi/180) - y2 * sin(theta_1*pi/180)

        CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN,
origin=(0.,0.,floc_3),
```

```
        point1=(x1p,y1p,floc_3), point2=(x2p,y2p,floc_3), name=name_1)
        csind = CS.id

        c_cells = c.findAt(((0.0,b/2.0,floc_3),), ((0.0,-b/2.0,floc_3),))
        regi = regionToolset.Region(cells=c_cells)

        name_2 = 'layer_'+str(theta_1)+ '_' + str(inte)
        floc_4 = materia[inte]
        section_2 = lammodel.HomogeneousSolidSection(name=name_2, material =
mat_name[floc_4 - 1][1])
        p2.SectionAssignment(region=regi, sectionName=name_2)
        p2.assignMaterialOrientation(region = regi,
localCsys=p2.datums[csind])

    print 'Coordinate systems created and material orientations assigned'
# ------------------------------------------------------------------------
# ------------------------------------------------------------------------
def lami_seam():
    p2 = lammodel.parts['lami']
    c, f, e = p2.cells, p2.faces, p2.edges
    floc = thick_midpoints[loc_ply_1 - 1]
    loc = len(thickness) - loc_ply_1
    floc_2 = thick_midpoints[loc]
    faces = f.findAt(((0., 0.0, floc),),((0.0, 0.0, floc_2),))
    regi = regionToolset.Region(faces=faces)
    p2.engineeringFeatures.assignSeam(regions = regi)
    print 'seams are created'

# ------------------------------------------------------------------
def lami_mesh_1():
# assigning mesh controls
    p2 = lammodel.parts['lami']
    c, e, f = p2.cells, p2.edges, p2.faces

    seed_1b = 1
    for inte in range(len(layup)+1):
        if (inte == 0):
            increm = 0
        else:
            increm = thick_location[inte-1]

        edges = e.findAt(((ATx,ATy,increm),), ((TUx,TUy,increm),),
((UVx,UVy,increm),),
                         ((VBx,VBy,increm),),  ((CWx,CWy,increm),),
                         ((WXx,WXy,increm),), ((XYx,XYy,increm),),
((YDx,YDy,increm),),)

        p2.seedEdgeByNumber(edges=edges, number=seed_1, constraint=FIXED)

        edges = e.findAt(((HMx,HMy,increm),), ((MLx,MLy,increm),),
((LKx,LKy,increm),),
                         ((KGx,KGy,increm),), ((FNx,FNy,increm),),
((NSx,NSy,increm),),
                         ((SRx,SRy,increm),), ((REx,REy,increm),),
((JOx,JOy,increm),),
                         ((OPx,OPy,increm),), ((PQx,PQy,increm),),
((QIx,QIy,increm),),)
```

```
        p2.seedEdgeByNumber(edges=edges, number=seed_1, constraint=FIXED)

        edges = e.findAt(((YQx,YQy,increm),), ((QRx,QRy,increm),),
((RKx,RKy,increm),),
                         ((KTx,KTy,increm),), ((XPx,XPy,increm),),
((PSx,PSy,increm),),
                         ((SLx,SLy,increm),), ((LUx,LUy,increm),),
((WOx,WOy,increm),),
                         ((ONx,ONy,increm),), ((NMx,NMy,increm),),
((MVx,MVy,increm),),
                         ((DIx,DIy,increm),), ((IEx,IEy,increm),),
((EGx,EGy,increm),),
                         ((BHx,BHy,increm),), ((HFx,HFy,increm),),
((FJx,FJy,increm),),
                         ((JCx,JCy,increm),),((GAx,GAy,increm),),)
        p2.seedEdgeByNumber(edges=edges, number=seed_1b, constraint=FIXED)


    for inte in range(len(layup)):
        increm = thick_midpoints[inte]

        if(inte==(loc_ply_1-1)):
            edges = e.findAt(((Ax, Ay, increm),), ((Bx, By, increm),), ((Cx,
Dy, increm),),
                             ((Dx, Dy, increm),), ((Ex, Ey, increm),), ((Fx,
Fy, increm),),
                             ((Gx, Gy, increm),), ((Hx, Hy, increm),), ((Ix,
Iy, increm),),
                             ((Jx, Jy, increm),), ((Kx, Ky, increm),), ((Lx,
Ly, increm),),
                             ((Mx, My, increm),), ((Nx, Ny, increm),), ((Ox,
Oy, increm),),
                             ((Px, Py, increm),), ((Rx, Ry, increm),), ((Sx,
Sy, increm),),
                             ((Tx, Ty, increm),), ((Ux, Uy, increm),), ((Vx,
Vy, increm),),
                             (( Wx, Wy, increm),), ((Xx, Xy, increm),), ((Yx,
Yy, increm),), )
            p2.seedEdgeByNumber(edges=edges, number=seed_6, constraint=FIXED)
        elif(inte ==(nol -i -1)):
            edges = e.findAt(((Ax, Ay, increm),), ((Bx, By, increm),), ((Cx,
Dy, increm),),
                             ((Dx, Dy, increm),), ((Ex, Ey, increm),), ((Fx,
Fy, increm),),
                             ((Gx, Gy, increm),), ((Hx, Hy, increm),), ((Ix,
Iy, increm),),
                             ((Jx, Jy, increm),), ((Kx, Ky, increm),), ((Lx,
Ly, increm),),
                             ((Mx, My, increm),), ((Nx, Ny, increm),), ((Ox,
Oy, increm),),
                             ((Px, Py, increm),), ((Rx, Ry, increm),), ((Sx,
Sy, increm),),
                             ((Tx, Ty, increm),), ((Ux, Uy, increm),), ((Vx,
Vy, increm),),
                             (( Wx, Wy, increm),), ((Xx, Xy, increm),), ((Yx,
Yy, increm),), )
            p2.seedEdgeByNumber(edges=edges, number=seed_6, constraint=FIXED)
```

```
        else:
            edges = e.findAt(((Ax, Ay, increm),), ((Bx, By, increm),), ((Cx,
Dy, increm),),
                             ((Dx, Dy, increm),), ((Ex, Ey, increm),), ((Fx,
Fy, increm),),
                             ((Gx, Gy, increm),), ((Hx, Hy, increm),), ((Ix,
Iy, increm),),
                             ((Jx, Jy, increm),), ((Kx, Ky, increm),), ((Lx,
Ly, increm),),
                             ((Mx, My, increm),), ((Nx, Ny, increm),), ((Ox,
Oy, increm),),
                             ((Px, Py, increm),), ((Rx, Ry, increm),), ((Sx,
Sy, increm),),
                             ((Tx, Ty, increm),), ((Ux, Uy, increm),), ((Vx,
Vy, increm),),
                             (( Wx, Wy, increm),), ((Xx, Xy, increm),), ((Yx,
Yy, increm),), )
            p2.seedEdgeByNumber(edges=edges, number=seed_5, constraint=FIXED)
##
# hiding the datum planes.. not supressing
    session.viewports['Viewport for Laminate
Model'].partDisplay.geometryOptions.setValues(datumPlanes=OFF)
##assigning the elements and meshing the model
    elemType1 = mesh.ElemType(elemCode=C3D8I, elemLibrary=STANDARD)
    cell = p2.cells
    cells = (cell, )
    elements = (elemType1)
    p2.setElementType(regions=cells , elemTypes=(elemType1, ))

    c = p2.cells
    cells = (c,)
    p2.generateMesh(regions=cells)
    view1 = session.Viewport(name='Mesh View of Laminate Model',
origin=(0.,0.))
    view1.setValues(displayedObject=p2)
    session.viewports['Mesh View of Laminate
Model'].partDisplay.setValues(mesh=ON)
    view1.assemblyDisplay.meshOptions.setValues(meshTechnique=ON,
meshVisibleEdges=EXTERIOR)
    session.viewports['Mesh View of Laminate
Model'].partDisplay.geometryOptions.setValues(datumPlanes=OFF)
##    session.viewports['Mesh View of Laminate Model'].maximize()

# ----------------------------------------------------------------------
def lami_assembly():
    p2 = lammodel.parts['lami']
    lamasseme = lammodel.rootAssembly
    laminsta = lamasseme.Instance(name = 'sublami', part = p2, dependent =
ON)

#----------------------------------------------------------------------
def lami_step():
    p4 = lammodel.rootAssembly
    p2 = lammodel.parts['lami']
    lammodel.StaticStep(name='Step-1', previous='Initial')
#----------------------------------------------------------------------
```

```
def lami_load():
    p4 = lammodel.rootAssembly
    inst = p4.instances['sublami']
    v = inst.vertices
    v_1 = v.findAt(((a/2.0,b/2.0,0.0),))
    region = regionToolset.Region(vertices=v_1)
    lammodel.ConcentratedForce(name='Load', createStepName='Step-1',
region=region, cf1=1000)
#-------------------------------------------------------------------
def lami_bc():
    p4 = lammodel.rootAssembly
    vv = p4.instances['sublami']
    v =vv.vertices
    v_1 = v.findAt(((a/2.0,b/2.0,ped),))
    region = regionToolset.Region(vertices=v_1)
    lammodel.DisplacementBC(name='BC1', createStepName='Initial',
region=region,
            u1=SET, u2=UNSET, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET,
            amplitude=UNSET, localCsys=None)
#-------------------------------------------------------------------
def lami_job():
    p4 = lammodel.parts['lami']
    hexjob = mdb.Job(name='Lami_Job',model=lammodel)
    hexjob.writeInput()
##    del mdb.models['Model-1']

# -------------------------------------------------------------------
def lami_assembly():
    p2 = lammodel.parts['lami']
    lamasseme = lammodel.rootAssembly
    laminsta = lamasseme.Instance(name = 'sublami', part = p2, dependent =
ON)


#-----------------------------------------------------------
from abaqus import *
from abaqusConstants import *
from section import*
import part
import regionToolset
import displayGroupMdbToolset as dgm
import material
import section
import sketch
import assembly
import step
import interaction
import load
import mesh
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import job
import math


lami_part()
```

```
lami_parti()
lami_mater()
lami_section_1()
lami_parti_2()
lami_seam()
lami_parti_3()
lami_assembly()
lami_mesh_1()
lami_step()
lami_load()
lami_bc()
lami_job()
```

Code for imposing the boundary conditions: ***bc_ninety.py***

```
# this program writes the boundary conditions for the
# *.inp file.
#
#
##  Note: this program requires the information from run.py
output_1 = open('runfile.inp', 'w')
output_2 = open('runfile_2.inp', 'w')
output_3 = open('element_sets.inp', 'w')

##_____________________________________________________________________________
##_____________________________________________________________________________
##-----------------------------------------------------------------------------

def reading_inp_file():
    global num_node, inte, intee, num_elem, inte_el
    global x, y, z, num_n, xa, xb, ya, yb, za, zb, xab, yab, zab
    global node1, node2, node3, node4, node5, node6, node7, node8, num_el
    global tem, tm, nola, crack_layer, thickness, crack_angle, outer_layer
    global num_ori, bc3
    x, y, z, num_n = [0], [0], [0], [0]
    node1, node2, node3, node4= [0], [0], [0], [0]
    node5, node6, node7, node8 = [0], [0], [0], [0]
    num_el = [0]
    inte, intee = 0, 0
    endfile, node_flag, elem_flag = 0, 0, 0 # logical operators
    num_node, num_elem = 0, 0 # initializing nodes and elements
    num_ori = 0

##  variables from run_lam_4_2.py
    nola = nol # number of layers
    crack_layer = loc_ply_1
    thickness = ped
    crack_angle = layup[loc_ply_1-1]
##    print crack_angle

    outer_layer = 0 ## the cracked layer is not an outer layer if the value
is one.

    input = open('Lami_Job.inp', 'r')
    strg = input.readlines()

    while inte in range(len(strg)):
```

```
strg_2 = strg[inte]

if(strg_2[1:8]=='Element'):
    output_1.write(strg_2)
    print 'Reading Elements'
    int_el = inte
    while(elem_flag != 1):
        int_el = int_el + 1
        strg_3 = strg[int_el]
        if(strg_3[0]=='*'):
            elem_flag = 1
            strg_temp = '** End of Elements\n'
            output_1.write(strg_temp)

        else:
            num_elem = num_elem + 1
            elem_reading(na=num_elem, stt= strg_3)
            output_1.write(strg_3)
            num_el = num_el + [num_elem]
            node1 = node1 + [nod1]
            node2 = node2 + [nod2]
            node3 = node3 + [nod3]
            node4 = node4 + [nod4]
            node5 = node5 + [nod5]
            node6 = node6 + [nod6]
            node7 = node7 + [nod7]
            node8 = node8 + [nod8]
            inte = int_el
            nn = num_elem
    del num_el[num_el.index(0)]
    del node1[node1.index(0)]
    del node2[node2.index(0)]
    del node3[node3.index(0)]
    del node4[node4.index(0)]
    del node5[node5.index(0)]
    del node6[node6.index(0)]
    del node7[node7.index(0)]
    del node8[node8.index(0)]
    face_pairs()
elif(strg_2[1:5] == 'Node'):
    output_1.write(strg_2)
    print 'Reading Nodes'
    intee = inte
    while (node_flag != 1):
        intee = intee + 1
        strg_2 = strg[intee]
        if(strg_2[0]=='*'):
            node_flag = 1
            strg_temp = '** End of Nodes\n'
            output_1.write(strg_temp)
        else:
            num_node = num_node + 1
            node_reading(n=num_node, st=strg_2)
            num_n = num_n + [num_node]
            x = x + [x1]
            y = y + [y1]
            z = z + [z1]
```

```
                    nn = num_node
                    strrg = '   '+str(num_n[nn])+','+'   '+ str(x[nn]) +
','+'   ' + str(y[nn]) + ','+'   ' + str(z[nn]) + '\n'
                    output_1.write(strrg)
                    inte = intee
            del x[x.index(0)]
            del y[y.index(0)]
            del z[z.index(0)]
            del num_n[num_n.index(0)]
            xa, xb, ya, yb, za, zb = max(x), min(x), max(y), min(y), max(z),
min(z)
            xab = (xa + xb)/2.0
            yab = (ya + yb)/2.0
            zab = (za + zb)/2.0
            cal_shf()
            node_swap(nt = num_n, xt = x, yt = y, zt = z)

        elif(strg_2[0:2]=='**'):
            output_1.write(strg[inte])

        elif(strg_2[0:9]=='*Boundary'):
            rigid_body_modes()
            inte = inte + 1

        elif(strg_2[0:8]=='*Restart'):
            print 'Eliminating Rigid Body Modes and writing Loads'
#           loads()
            output_1.write(strg_2)

        elif(strg_2[0:6]=='*Cload'):
            print 'Writing Loads'
            loads()
            inte = inte + 1

        elif(strg_2[0:13]=='*End Instance'):
            print 'Writing Constraints'
            if(lumped_inner == 1):
                corner_constraints_lump()
                edge_constraints_lump()
                face_constraints_lump()
                center_constraints_lump()
            else:
                corner_constraints()
                edge_constraints()
                face_constraints()
                center_constraints()

            output_1.write(strg_2)

        elif(strg_2[0:6] == '*Elset'):
            output_1.write(strg_2)
            output_3.write(strg_2)
            intel = inte
            flag = 0
            while(flag == 0):
                intel = intel + 1
                strg_5 = strg[intel]
```

```
        if(strg_5[0] != '*'):
            output_1.write(strg_5)
            output_3.write(strg_5)
        else:
            flag = 1
            inte = intel - 1

elif(strg_2[0:14]=='*Solid Section'):
    output_1.write(strg_2)
    output_3.write(strg_2)
    intel = inte
    flag = 0
    while(flag == 0):
        intel = intel + 1
        strg_5 = strg[intel]
        if(strg_5[0] != '*'):
            output_1.write(strg_5)
            output_3.write(strg_5)
        else:
            flag = 1
            inte = intel - 1

elif(strg_2[1:7]=='Orient'):
    output_1.write(strg_2)
    output_3.write(strg_2)
    intel = inte
    flag = 0
    num_ori = num_ori + 1
    while(flag == 0):
        intel = intel + 1
        strg_5 = strg[intel]
        if(strg_5[0] != '*'):
            output_1.write(strg_5)
            output_3.write(strg_5)
        else:
            flag = 1
            inte = intel - 1

elif(strg_2[1:13]=='End Assembly'):
    strgg = '*Nset, nset=n_set_1, instance=sublami \n'
    output_1.write(strgg)
    strgg = str(c_1)+'\n'
    output_1.write(str(strgg))
    strgg = '*Nset, nset=n_set_2, instance=sublami \n'
    output_1.write(strgg)
    strgg = str(c_2)+'\n'
    output_1.write(strgg)
  output_1.write('*Nset, nset=n_set_3, instance=sublami \n')
    strgg = str(c_1_2)+'\n'
    output_1.write(strgg)
  output_1.write('*Nset, nset=n_set_4, instance=sublami \n')
  strgg = str(c_4)+'\n'
    output_1.write(strgg)
    output_1.write('*Nset, nset=n_set_5, instance=sublami \n')
  strgg = str(bodyc)+'\n'
    output_1.write(strgg)
    output_1.write('*Nset, nset=n_set_6, instance=sublami \n')
```

```
          strgg = str(c_1_1)+'\n'
            output_1.write(strgg)
            output_1.write('*Nset, nset=n_set_7, instance=sublami \n')
          strgg = str(c_6_x)+'\n'
            output_1.write(strgg)
            output_1.write('*Nset, nset=n_set_8, instance=sublami \n')
          strgg = str(c_6_y)+'\n'
            output_1.write(strgg)

            output_1.write('*Nset, nset=n_set_9, instance=sublami, internal,
generate \n')
          strgg = str(1)+','+'  '+ str(num_node)+','+'  '+str(1)+'\n'
            output_1.write(strgg)

            output_1.write('*Nset, nset=n_set_10, instance=sublami \n')
          strgg = str(c_14m)+'\n'
            output_1.write(strgg)

            output_1.write('*Nset, nset=n_set_11, instance=sublami \n')
          strgg = str(c_23m)+'\n'
            output_1.write(strgg)

            if(lumped_inner == 1):
                output_1.write('*Nset, nset=n_set_12, instance=sublami \n')
                strgg = str(c_21)+'\n'
                output_1.write(strgg)
                output_1.write('*Nset, nset=n_set_14, instance=sublami \n')
                strgg = str(c_22)+'\n'
                output_1.write(strgg)
                output_1.write('*Nset, nset=n_set_15, instance=sublami \n')
                strgg = str(bc2)+'\n'
                output_1.write(strgg)
                output_1.write('*Nset, nset=n_set_16, instance=sublami \n')
                strgg = str(bc3)+'\n'
                output_1.write(strgg)

          output_1.write(strg_2)
          print 'Nsets for rigid body modes and loads - created'

      elif(strg_2[1:14]=='Output, field'):
            output_1.write(strg_2)
            output_1.write('*Element output,position=centroid \n')
            strgg = 'Evol, S \n'
            output_1.write(strgg)

        else:
            output_1.write(strg[inte])
        inte = inte + 1
    else:
        print 'End of File'

    output_1.close()
    output_2.close()
    output_3.close()
##_____________________________________________________________________________
##-----------------------------------------------------------------------------
def cal_shf():
```

```
    global shf
    temp_x, temp_y, temp_z  = [0], [0], [0]
    for j in range(len(x)):
        tt = abs(x[j])/10000.0
        tt = str(tt)
        temp_x = temp_x + [int(len(tt))]

        tt = abs(y[j])/10000.0
        tt = str(tt)
        temp_y = temp_y + [int(len(tt))]

        tt = abs(z[j])/10000.0
        tt = str(tt)
        temp_z = temp_z + [int(len(tt))]

    del temp_x[temp_x.index(0)]
    del temp_y[temp_y.index(0)]
    del temp_z[temp_z.index(0)]

    tx_i, ty_i = temp_x.index(max(temp_x)), temp_y.index(max(temp_y))
    tz_i = temp_z.index(max(temp_z))

    tx_max = x[tx_i]
    ty_max = y[ty_i]
    tz_max = z[tz_i]
    tt = [tx_max, ty_max, tz_max]
    ttt = [len(str(tx_max)), len(str(ty_max)), len(str(tz_max))]
    tttt = str(tt[ttt.index(max(ttt))])
    tttt = tttt.replace('.', ' ')
    tttt = tttt.split()
    t5 = tttt[1]
    t5 = len(t5)
    sh = '1E-'+str(t5-3)
    shf = float(sh)

##---------------------------------------------------------------------------
def node_reading(n, st):
    global num_n1, x1, y1, z1
    num_n1 = n
    x1, y1, z1 = 0., 0., 0.
    n_ck = 0

    strg_n = st.replace(',', ' ')
    strg_n = strg_n.split()
    n_k = int(strg_n[0])
    x1 = float(strg_n[1])
    y1 = float(strg_n[2])
    z1 = float(strg_n[3])

#---------------------------------------------------------------------------
def elem_reading(na, stt):
    global nod1, nod2, nod3, nod4, nod5, nod6, nod7, nod8
    nod1, nod2, nod3, nod4, nod5, nod6, nod7, nod8  = 0, 0, 0, 0, 0, 0, 0, 0

    strg_n = stt.replace(',', ' ')
    strg_n = strg_n.split()
    n_k = int(strg_n[0])
```

```
    nod1 = int(strg_n[1])
    nod2 = int(strg_n[2])
    nod3 = int(strg_n[3])
    nod4 = int(strg_n[4])
    nod5 = int(strg_n[5])
    nod6 = int(strg_n[6])
    nod7 = int(strg_n[7])
    nod8 = int(strg_n[8])

#-------------------------------------------------------------------------
def node_swap(nt, xt, yt, zt):
## this subroutine detects the center nodes, corner nodes, edge nodes and
face nodes.
    global tem, tm
    global c_1, c_2, c_3, c_4, c_1_1, c_1_2, bodyc, bc2, bc3
    global c135, c136, c146, c145, c235, c236, c246, c245
    global c146_2, c236_2, c145_2, c235_2
    global ct_35, ct_36, ct_45, ct_46
    global e_35, e_36, e_45, e_46
    global e_15, e_16, e_25, e_26
    global e_13, e_14, e_23, e_24
    global f_1, f_2, f_3, f_4, shf
    global c_6_x, c_6_y, c_14m, c_23m
    global c_11, c_12, c_21, c_22, c_21
    global ck_z_1, ck_z_2, ck_z_3, ck_z_4
    global e_15c_11, e_15c_22, e_16c_11, e_16c_22
    global e_25c_11, e_25c_22, e_26c_11, e_26c_22

    ck_z_1 = thick[loc_ply_1 - 1]
    ck_z_2 = thick[loc_ply_1]
    locaa = nol - loc_ply_1 + 1
    ck_z_3 = thick[locaa - 1]
    ck_z_4 = thick[locaa]

    j  = 0
    c_1, c_2, c_3, c_4 = 0,0,0,0
    c_11, c_12, c_21, c_22 = 0, 0, 0, 0
    c_1_1, c_1_2 = 0,0
    c135, c136, c146, c145 = 0,0,0,0
    c235, c236, c246, c245 = 0,0,0,0
    c_14m, c_23m = 0 , 0
    c146_2, c145_2, c235_2, c236_2 = 0,0,0,0
    e_15c_11, e_15c_22, e_25c_11, e_25c_22 = 0, 0, 0, 0
    e_16c_11, e_16c_22, e_26c_11, e_26c_22 = 0, 0, 0, 0
    trip_c146, trip_c236, trip_c145, trip_c235 = 0, 0, 0, 0
    trip_c1511, trip_c1611 = 0, 0
    trip_c2511, trip_c2611 = 0, 0

    while j in range(len(nt)):
        tx, ty, tz = x[j], y[j], z[j]

        if((abs(tx-xab)<= shf)and(abs(ty-yb)<=shf)and(abs(tz-zab)<=shf)):
            c_1 = nt[j]
            print 'Face center - 1', c_1
        elif((abs(xab-tx)<=shf)and(abs(ty-ya)<=shf)and(abs(tz-zab)<=shf)):
            c_2 = nt[j]
            print 'Face center - 2', c_2
```

```
        elif((abs(xab-tx)<=shf)and(abs(ty-ya)<=shf)and(abs(tz-ck_z_1)<=shf)):
            c_21 = nt[j]
            print 'Face center - 2 1', c_21
        elif((abs(tx-xa)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-zab)<=shf)):
          c_3 = nt[j]
            print 'Face center - 3', c_3
        elif((abs(tx-xb)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-zab)<=shf)):
            c_4 = nt[j]
          print 'Face center - 4', c_4
      elif((abs(xa-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(zb-tz)<=shf)):
          c135 = nt[j]
          print 'Corner 135', c135
      elif((abs(xa-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(za-tz)<=shf)):
          c136 = nt[j]
            print 'Corner 136', c136

      elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(za-tz)<=shf)and
(trip_c146 == 0)):
          c146 = nt[j]
          trip_c146 = 1
            print 'Corner 146', c146
      elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(za-tz)<=shf) and
(trip_c146 == 1)):
          c146_2 = nt[j]
            print 'Corner 146 - 2 ', c146_2
      elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(zb-tz)<=shf) and
(trip_c145 == 0)):
          c145 = nt[j]
          trip_c145 = 1
            print 'Corner 145', c145
        elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(zb-tz)<=shf) and
(trip_c145 == 1)):
          c145_2 = nt[j]
            print 'Corner 145 - 2', c145_2
      elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(zb-tz)<=shf) and
(trip_c235 == 0)):
          c235 = nt[j]
          trip_c235 = 1
            print 'Corner 235', c235
        elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(zb-tz)<=shf) and
(trip_c235 == 1)):
          c235_2 = nt[j]
            print 'Corner 235 - 2', c235_2
      elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(za-tz)<=shf) and
(trip_c236 == 0)):
          c236 = nt[j]
          trip_c236 = 1
            print 'Corner 236', c236
      elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(za-tz)<=shf) and
(trip_c236 == 1)):
          c236_2 = nt[j]
            print 'Corner 236 - 2', c236_2

      elif((abs(xb-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(za-tz)<=shf)):
          c246 = nt[j]
          print 'Corner 246', c246
      elif((abs(xb-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(zb-tz)<=shf)):
```

```
          c245 = nt[j]
            print 'Corner 245', c245
        elif((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-zb)<=shf)):
            c_1_1 = nt[j]
        elif((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-za)<=shf)):
            c_1_2 = nt[j]
        elif((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-zab)<=shf)):
            bodyc = nt[j]

        elif((abs(tx-xb)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-za)<=shf)):
            c_6_x = nt[j]
        elif((abs(xab-tx)<=shf)and(abs(ty-ya)<=shf)and(abs(tz-za)<=shf)):
            c_6_y = nt[j]

        elif((abs(xb-tx)<=shf)and(abs(ty-yb)<=shf)and(abs(tz-zab)<=shf)):
            c_14m = nt[j]
        elif((abs(xa-tx)<=shf)and(abs(ty-ya)<=shf)and(abs(tz-zab)<=shf)):
            c_23m = nt[j]

        if(lumped_inner == 1):
            if((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-
ck_z_4)<=shf)):
                bc2 = nt[j]
                print 'Body center - 2', bc2
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf)and(abs(tz-
ck_z_1)<=shf)):
                c_11 = nt[j]
                print 'Face center - 31 ', c_11
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf)and(abs(tz-
ck_z_4)<=shf)):
                c_12 = nt[j]
                print 'Face center - 32 ', c_12

            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_1)<=shf)):
                c_21 = nt[j]
                print 'Face center - 41 ', c_21
            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_4)<=shf)):
                c_22 = nt[j]
                print 'Face center - 42 ', c_22

            elif((abs(tx-xb)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-zab)<=shf)):
                bc3 = nt[j]
                print 'New BC - ', bc3

      if(outerlayer == 1):
            if((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf) and (abs(tz-
ck_z_1)<=shf) and (trip_c1511 == 0)):
                e_15c_11 = nt[j]
                trip_c1511 = 1
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf) and (abs(tz-
ck_z_1)<=shf) and (trip_c1511 == 1)):
                e_15c_22 = nt[j]
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf)and(abs(tz-
ck_z_4)<=shf) and (trip_c1611 == 0)):
                e_16c_11 = nt[j]
```

```
                trip_c1611 = 1
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf)and(abs(tz-
ck_z_4)<=shf) and (trip_c1611 == 1)):
                e_16c_22 = nt[j]

            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_1)<=shf) and (trip_c2511 == 0)):
                e_25c_11 = nt[j]
                trip_c2511 = 1
            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_1)<=shf) and (trip_c2511 == 1)):
                e_25c_22 = nt[j]
            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_4)<=shf) and (trip_c2611 == 0)):
                e_26c_11 = nt[j]
                trip_c2611 = 1
            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_4)<=shf) and (trip_c2611 == 1)):
                e_26c_22 = nt[j]

        j = j + 1

    if((outerlayer == 0) and (lumped_inner == 0)):
        tm = [c135, c136, c146, c145, c235, c236, c246, c245]
        tem = [c_1, c_2, c_3, c_4]

    elif((outerlayer == 1) and (lumped_inner == 0)):
        tm = [c135, c136, c146, c145, c235, c236, c246, c245,
              e_15c_11, e_25c_11,  e_16c_11, e_26c_11, e_25c_22, e_26c_22,
e_15c_22,  e_16c_22]
        tem = [c_1, c_2, c_3, c_4]

    elif(lumped_inner == 1):
        tm = [c135, c136, c146, c145, c235, c236, c246, c245,
              c_11, c_12, c_21, c_22]
        tem = [c_3, c_4]

    if(0 in tm):
        print 'Zero in tm'
    else:
        print 'TM is complete '

##    print tm
    print 'Center nodes and Corner nodes are identified'
##_____________________________________________________________________________
##-----------------------------------------------------------------------------
## subroutine to detect the edge node 35 36 45 46

    ct_35, ct_36, ct_45, ct_46 = 0, 0, 0, 0
    ct_15, ct_16, ct_25, ct_26 = 0, 0, 0, 0
    ct_13, ct_14, ct_23, ct_24 = 0, 0, 0, 0

    j = 0
    front, fronta, frontb, frontc = 0, 0, 0, 0
    frontd, fronte, frontf, frontg = 0, 0, 0, 0
    fronth, fronti, frontj, frontk = 0, 0, 0, 0
```

```
    e_35, e_36, e_45, e_46 = [0], [0], [0], [0]
    e_15, e_16, e_25, e_26 = [0], [0], [0], [0]
    e_13, e_14, e_23, e_24 = [0], [0], [0], [0]

    while j in range(len(nt)):
        tx, ty, tz = x[j], y[j], z[j]
        check = nt[j]
        front, fronta, frontb, frontc  = 0, 0, 0, 0
        frontd, fronte, frontf, frontg = 0, 0, 0, 0
        fronth, fronti, frontj, frontk = 0, 0, 0, 0

        if check in tem:
            front, fronta, frontb, frontc  = 1, 1, 1, 1
            frontd, fronte, frontf, frontg = 1, 1, 1, 1
            fronth, fronti, frontj, frontk = 1, 1, 1, 1
        if check in tm:
            front, fronta, frontb, frontc  = 1, 1, 1, 1
            frontd, fronte, frontf, frontg = 1, 1, 1, 1
            fronth, fronti, frontj, frontk = 1, 1, 1, 1

##       EDGE 35
      if((abs(tx-xa)<=shf)and(abs(tz-zb)<=shf)):
          if(front!=1):
                ct_35 = ct_35 + 1
            e_35 = e_35 + [check]

##     EDGE 36
      elif((abs(tx-xa)<=shf)and(abs(tz-za)<=shf)):
            if(fronta!=1):
                ct_36 = ct_36 + 1
                e_36 = e_36 + [check]

##      EDGE 45
        elif( (abs(tx-xb)<=shf)and(abs(tz-zb)<=shf) ):
            if(frontb!=1):
                ct_45 = ct_45 + 1
                e_45 = e_45 + [check]

##      EDGE 46
        elif((abs(tx-xb)<=shf)and(abs(tz-za)<=shf)):
            if(frontc!=1):
                ct_46 = ct_46 + 1
                e_46 = e_46 + [check]

##      EDGE 15
        elif((abs(ty-yb)<=shf)and(abs(tz-zb)<=shf)):
            if(frontd!=1):
                ct_15 = ct_15 + 1
            e_15 = e_15 + [check]

##      EDGE 25
      elif((abs(ty-ya)<=shf)and(abs(tz-zb)<=shf)):
            if(fronte!=1):
            ct_25 = ct_25 + 1
            e_25 = e_25 + [check]

##      EDGE 16
```

```
      elif((abs(ty-yb)<=shf)and(abs(tz-za)<=shf)):
            if(frontf!=1):
            ct_16 = ct_16 + 1
            e_16 = e_16 + [check]

##      EDGE 26
      elif((abs(ty-ya)<=shf)and(abs(tz-za)<=shf)):
          if(frontg!=1):
            ct_26 = ct_26 + 1
            e_26 = e_26 + [check]

##      EDGE 13
      elif((abs(tx-xa)<=shf)and(abs(ty-yb)<=shf)):
            if(fronth!=1):
                ct_13 = ct_13 + 1
                e_13 = e_13 + [check]

##      EDGE 14
        elif((abs(tx-xb)<=shf)and(abs(ty-yb)<=shf)):
            if(fronti!=1):
                ct_14 = ct_14 + 1
                e_14 = e_14 + [check]

##      EDGE 23
        elif((abs(tx-xa)<=shf)and(abs(ty-ya)<=shf)):
            if(frontj!=1):
                ct_23 = ct_23 + 1
                e_23 = e_23 + [check]

##      EDGE 24
        elif((abs(tx-xb)<=shf)and(abs(ty-ya)<=shf)):
            if(frontk!=1):
                ct_24 = ct_24 + 1
                e_24 = e_24 + [check]

        j = j + 1
    del e_35[e_35.index(0)]
    del e_36[e_36.index(0)]
    del e_45[e_45.index(0)]
    del e_46[e_46.index(0)]
    del e_15[e_15.index(0)]
    del e_16[e_16.index(0)]
    del e_25[e_25.index(0)]
    del e_26[e_26.index(0)]
    del e_13[e_13.index(0)]
    del e_23[e_23.index(0)]
    del e_14[e_14.index(0)]
    del e_24[e_24.index(0)]
    print 'Edge nodes are identified'
##-----------------------------------------------------------------------------
## nodes on the edges are identified. These nodes have to be paired for
writing
## the constraint equations.  the node pairs are based on the boundary
conditions.
    global e_1626, e_1525, e_3646, e_3545
    global ct_e_1626, ct_e_1525, ct_e_3545, ct_e_3646
    k = 0
```

```
    m = 0
    e_1626, e_1525, e_3545, e_3646 = [0], [0], [0], [0]
    ct_e_1626, ct_e_1525, ct_e_3646, ct_e_3545 = 0, 0, 0, 0
    if(ct_16 != ct_26):
            print 'The edges nodes are not compatible 16 26 -- check the
code'
    while k in range(len(e_16)):
        check1 = e_16[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_26)):
            check2 = e_26[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tx1-tx2)<=shf):
                ct_e_1626 = ct_e_1626 + 2
                e_1626 = e_1626 + [check1]
                e_1626 = e_1626 + [check2]
            m = m + 1
        k = k + 1
    del e_1626[e_1626.index(0)]
    if((ct_e_1626/ct_16 != 2) and (ct_e_1626/ct_26 != 2)):
        print '  Error in identifying edge pairs -  Edges 16 and 26'
    else:
        print '  Edge pairs 16 and 26 are identified'
##-------------------------------------------------
    k, m = 0, 0
    if(ct_15 != ct_25):
            print 'The edges nodes are not compatible 15 25 -- check the
code'
    while k in range(len(e_15)):
        check1 = e_15[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_25)):
            check2 = e_25[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tx1-tx2)<=shf):
                ct_e_1525 = ct_e_1525 + 2
                e_1525 = e_1525 + [check1]
                e_1525 = e_1525 + [check2]
            m = m + 1
        k = k + 1
    del e_1525[e_1525.index(0)]
    if((ct_e_1525/ct_15 != 2) and (ct_e_1525/ct_25 != 2)):
        print '  Error in identifying edge pairs -  Edges 15 and 25'
    else:
        print '  Edge pairs 15 and 25 are identified'
##-------------------------------------------------
    k, m = 0, 0
    if(ct_35 != ct_45):
            print 'The edges nodes are not compatible  35 45 -- check the
code'
    while k in range(len(e_35)):
        check1 = e_35[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_45)):
```

```
            check2 = e_45[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_3545 = ct_e_3545 + 2
                e_3545 = e_3545 + [check1]
                e_3545 = e_3545 + [check2]
            m = m + 1
        k = k + 1
    del e_3545[e_3545.index(0)]
    if((ct_e_3545/ct_35 != 2) and (ct_e_3545/ct_45 != 2)):
        print '  Error in identifying edge pairs -  Edges 35 and 45'
    else:
        print '  Edge pairs 35 and 45 are identified'
##-----------------------------------------------------
    k, m = 0, 0
    if(ct_36 != ct_46):
            print 'The edges nodes are not compatible  36 46 -- check the
code'
    while k in range(len(e_36)):
        check1 = e_36[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_46)):
            check2 = e_46[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_3646 = ct_e_3646 + 2
                e_3646 = e_3646 + [check1]
                e_3646 = e_3646 + [check2]
            m = m + 1
        k = k + 1
    del e_3646[e_3646.index(0)]
    if((ct_e_3646/ct_36 != 2) and (ct_e_3646/ct_46 != 2)):
        print '  Error in identifying edge pairs -  Edges 36 and 46'
    else:
        print '  Edge pairs 36 and 46 are identified'
##-----------------------------------------------------
    global ct_e_1424, ct_e_2324, ct_e_1324
    global e_1424, e_2324, e_1324
    e_1424, e_2324, e_1324 = [0], [0], [0]

    ct_e_1424, ct_e_2324, ct_e_1324 = 0, 0, 0
    k, m = 0, 0
    if(ct_14 != ct_24):
            print 'The edges nodes are noe_2324t compatible  14 24 -- check
the code'
    while k in range(len(e_14)):
        check1 = e_14[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_24)):
            check2 = e_24[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tz1-tz2)<=shf):
                ct_e_1424 = ct_e_1424 + 2
                e_1424 = e_1424 + [check1]
                e_1424 = e_1424 + [check2]
```

```
            m = m + 1
        k = k + 1
    del e_1424[e_1424.index(0)]
    if((ct_e_1424/ct_14 != 2) and (ct_e_1424/ct_24 != 2)):
        print '  Error in identifying edge pairs -  Edges 14 and 24 Uncracked
Region'
    else:
        print '  Edge pairs 14 and 24 are identified - Uncracked Region'
##-------------------------------------------------
    k, m = 0, 0
    if(ct_23 != ct_24):
            print 'The edges nodes are not compatible  23 24 -- check the
code'
    while k in range(len(e_23)):
        check1 = e_23[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_24)):
            check2 = e_24[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tz1-tz2)<=shf):
                ct_e_2324 = ct_e_2324 + 2
                e_2324 = e_2324 + [check1]
                e_2324 = e_2324 + [check2]
            m = m + 1
        k = k + 1
    del e_2324[e_2324.index(0)]
    if((ct_e_2324/ct_23 != 2) and (ct_e_2324/ct_24 != 2)):
        print '  Error in identifying edge pairs -  Edges 23 and 24 Uncracked
Region'
    else:
        print '  Edge pairs 23 and 24 are identified - Uncracked Region'
##-------------------------------------------------
    k, m = 0, 0
    if(ct_13 != ct_24):
            print 'The edges nodes are not compatible  13 24 -- check the
code'
    while k in range(len(e_13)):
        check1 = e_13[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_24)):
            check2 = e_24[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if((abs(tz1-tz2)<=shf) and (abs(ty1 + ty2)<= shf)):
                ct_e_1324 = ct_e_1324 + 2
                e_1324 = e_1324 + [check1]
                e_1324 = e_1324 + [check2]
            m = m + 1
        k = k + 1
    del e_1324[e_1324.index(0)]
    if((ct_e_1324/ct_13 != 2) and (ct_e_1324/ct_24 != 2)):
        print '  Error in identifying edge pairs -  Edges 13 and 24 Uncracked
Region'
    else:
        print '  Edge pairs 13 and 24 are identified - Uncracked Region'
##-------------------------------------------------------------------------
```

```
##-----------------------------------------------------------------------------
def face_pairs():
## nodes on faces are recognized now. we do not need face 5 and face 6 nodes.
    global ct_f_1, ct_f_2, ct_f_3, ct_f_4
    global f_1c, f_2c, ct_f_1c, ct_f_2c, f_3, f_4, f_1, f_2
    ct_f_1, ct_f_2, ct_f_3, ct_f_4 = 0, 0, 0, 0
    f_1, f_2, f_3, f_4 = [0], [0], [0], [0]
    f_1c, f_2c = [0], [0]
    ct_f_1c, ct_f_2c = 0, 0
    j = 0

    while j in range(len(num_n)):
        tx, ty, tz = x[j], y[j], z[j]
        check = num_n[j]
        frontl, frontm, frontn, fronto = 0, 0, 0, 0

        if check in tm:     frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in tem:    frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_35:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_36:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_45:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_46:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_15:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_25:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_16:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_26:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_13:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_23:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_14:   frontl, frontm, frontn, fronto = 1, 1, 1, 1
        if check in e_24:   frontl, frontm, frontn, fronto = 1, 1, 1, 1

        if(abs(ty-yb)<=shf):

            if((outerlayer == 0) and (lumped_inner == 0)):
                if( ((tz>ck_z_1) and (tz<ck_z_2) and (abs(tx-xab)<=shf) and
not (abs(tz-ck_z_1)<=shf) and not(abs(tz-ck_z_2)<=shf))
                 or ((tz<ck_z_4) and (tz>ck_z_3) and (abs(tx-xab)<=shf) and
not (abs(tz-ck_z_3)<=shf) and not(abs(tz-ck_z_4)<=shf))):
                    if(frontl != 1):
                        ct_f_1c = ct_f_1c + 1
                        f_1c = f_1c + [check]
                else:
                    if(frontl != 1):
                        ct_f_1 = ct_f_1 + 1
                        f_1 = f_1 + [check]

            elif((outerlayer == 0) and (lumped_inner == 1)):
                if( (tz>ck_z_1) and (tz<ck_z_4) and (abs(tx-xab)<=shf) and
not((abs(tz-ck_z_1)<=shf) or(abs(tz-ck_z_4)<=shf)) ):
                    if(frontl != 1):
                        ct_f_1c = ct_f_1c + 1
                        f_1c = f_1c + [check]
                else:
                    if(frontl != 1):
                        ct_f_1 = ct_f_1 + 1
                        f_1 = f_1 + [check]
```

```
            elif((outerlayer == 1) and (lumped_inner == 0)):
                if( ( (tz>ck_z_1) and (tz<ck_z_2) and (abs(tx-xab)<=shf) and
not(abs(tz-ck_z_2)<=shf) )
                 or ( (tz<ck_z_4) and (tz>ck_z_3) and (abs(tx-xab)<=shf) and
not(abs(tz-ck_z_3)<=shf) ) ):
                    if(frontl != 1):
                        ct_f_1c = ct_f_1c + 1
                        f_1c = f_1c + [check]
                else:
                    if(frontl != 1):
                        ct_f_1 = ct_f_1 + 1
                        f_1 = f_1 + [check]

        elif(abs(ty-ya)<=shf):

            if((outerlayer == 0) and (lumped_inner == 0)):
                if( ((tz>ck_z_1) and (tz<ck_z_2) and (abs(tx-xab)<=shf) and
not (abs(tz-ck_z_1)<=shf) and not(abs(tz-ck_z_2)<=shf))
                 or ((tz<ck_z_4) and (tz>ck_z_3) and (abs(tx-xab)<=shf) and
not (abs(tz-ck_z_3)<=shf) and not(abs(tz-ck_z_4)<=shf))):
                    if(frontm != 1):
                        ct_f_2c = ct_f_2c + 1
                        f_2c = f_2c + [check]
                else:
                    if(frontm != 1):
                        ct_f_2 = ct_f_2 + 1
                        f_2 = f_2 + [check]

            elif((outerlayer == 0) and (lumped_inner == 1)):
                if( (tz>ck_z_1) and (tz<ck_z_4) and (abs(tx-xab)<=shf) and
not((abs(tz-ck_z_1)<=shf) or(abs(tz-ck_z_4)<=shf)) ):
                    if(frontm != 1):
                        ct_f_2c = ct_f_2c + 1
                        f_2c = f_2c + [check]
                else:
                    if(frontm != 1):
                        ct_f_2 = ct_f_2 + 1
                        f_2 = f_2 + [check]

            elif((outerlayer == 1) and (lumped_inner == 0)):
                if( ( (tz>ck_z_1) and (tz<ck_z_2) and (abs(tx-xab)<=shf) and
not(abs(tz-ck_z_2)<=shf) )
                 or ( (tz<ck_z_4) and (tz>ck_z_3) and (abs(tx-xab)<=shf) and
not(abs(tz-ck_z_3)<=shf) ) ):
                    if(frontm != 1):
                        ct_f_2c = ct_f_2c + 1
                        f_2c = f_2c + [check]
                else:
                    if(frontm != 1):
                        ct_f_2 = ct_f_2 + 1
                        f_2 = f_2 + [check]

        elif(abs(tx-xa)<=shf):
            if(frontn != 1):
                ct_f_3 = ct_f_3 + 1
                f_3 = f_3 + [check]
```

```
        elif(abs(tx-xb)<=shf):
             if(fronto != 1):
                ct_f_4 = ct_f_4 + 1
                f_4 = f_4 + [check]

        j = j + 1

    if(outerlayer == 1):
        f_1c = f_1c + [e_15c_11]
        f_1c = f_1c + [e_15c_22]
        f_1c = f_1c + [e_16c_11]
        f_1c = f_1c + [e_16c_22]

        f_2c = f_2c + [e_25c_11]
        f_2c = f_2c + [e_25c_22]
        f_2c = f_2c + [e_26c_11]
        f_2c = f_2c + [e_26c_22]

    del f_1[f_1.index(0)]
    del f_2[f_2.index(0)]
    del f_3[f_3.index(0)]
    del f_4[f_4.index(0)]

    del f_1c[f_1c.index(0)]
    del f_2c[f_2c.index(0)]

    print 'Face nodes are identified'
##-------------------------------------------------------------------------
## this routine is for identifying the node pairs in all the combinations of
edge pairs,
## face pairs, corner node pairs, face center pairs.
##-------------------------------------------------------------------------
global ct_f_12, ct_f_34, fp_12, fp_34

    j, k = 0, 0
    fp_12, fp_34 = [0], [0]
    ct_f_12, ct_f_34 = 0, 0

    while j in range(len(f_1)):
        if(ct_f_1 != ct_f_2):
            print 'Face pairs F1 and F2 are not matched'
        check1 = f_1[j]
        tx1, ty1, tz1 = x[check1 - 1],y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_2)):
            check2 = f_2[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tx1 - tx2) <= shf) and (abs(tz1 - tz2) <= shf) and
(abs(ty1 + ty2) <= shf)):
                ct_f_12 = ct_f_12 + 2
                fp_12 = fp_12 + [check1]
                fp_12 = fp_12 + [check2]
            k = k + 1
        j = j + 1
    del fp_12[fp_12.index(0)]
    if((ct_f_12/ct_f_1 != 2) and (ct_f_12/ct_f_2 != 2)):
        print '  Error in identifying face pairs -  Face 1 and Face 2'
```

```
    else:
        print '  Face pairs F1 and F2 are identified'
##-----------------------------------------------------
    j, k = 0,0
    while j in range(len(f_3)):
        if(ct_f_3 != ct_f_4):
            print 'Face pairs F1 and F2 are not matched'
        check1 = f_3[j]
        tx1, ty1, tz1 = x[check1 - 1],y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_4)):
            check2 = f_4[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tx1 + tx2) <= shf) and (abs(tz1 - tz2) <= shf) and
(abs(ty1 - ty2) <= shf)):
                ct_f_34 = ct_f_34 + 2
                fp_34 = fp_34 + [check1]
                fp_34 = fp_34 + [check2]
            k = k + 1
        j = j + 1
    del fp_34[fp_34.index(0)]
    if((ct_f_34/ct_f_3 != 2) and (ct_f_34/ct_f_4 != 2)):
        print '  Error in identifying face pairs -  Face 3 and Face 4'
    else:
        print '  Face pairs F3 and F4 are identified'
##-------------------------------------------------------------------------
##-----------------------------------------------------
    global e_1c_el, e_1c_n, e_2c_el, e_2c_n
    e_1c_el, e_1c_n, e_2c_el, e_2c_n = [0], [0], [0], [0]

    k, m  = 0, 0
    e_1c_n, e_1c_el = [0], [0]
    while k in range(len(f_1c)):
        check = f_1c[k]
        m  = 0
        while m in range(len(node1)):
            check1 = node1[m]
            check2 = node2[m]
            check3 = node3[m]
            check4 = node4[m]
            check5 = node5[m]
            check6 = node6[m]
            check7 = node7[m]
            check8 = node8[m]
            if check in e_1c_n:
                pass
            else:
                if((check == check1) or (check == check2) or (check ==
check3) or (check == check4)
                   or (check == check5) or (check == check6) or (check ==
check7) or (check == check8)):
                    check9 = num_el[m]
                    e_1c_el = e_1c_el + [check9]
                    e_1c_n = e_1c_n + [check]
            m = m + 1
        k = k + 1
    del e_1c_el[e_1c_el.index(0)]
```

```
    del e_1c_n[e_1c_n.index(0)]
##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
    global e_1c_BOBp, e_1c_BBp
##    from math import acosFace nodes are identified
##    pi = acos(-1.0)
    k, m = 0, 0
    e_1c_BOBp, e_1c_BBp = [0], [0]
    while k in range(len(e_1c_el)):
        check = e_1c_el[k]
        check2 = e_1c_n[k]
        t1, t2, t3, t4 = node1[check - 1], node2[check - 1], node3[check -
1], node4[check - 1]
        t5, t6, t7, t8 = node5[check - 1], node6[check - 1], node7[check -
1], node8[check - 1]
        tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1], x[t4 - 1]
        tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1], x[t8 - 1]
        ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1], y[t4 - 1]
        ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1], y[t8 - 1]
        tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7 + tx8)
        tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7 + ty8)
##        from math  import atan
##        tempxy = atan(tempy/tempx) * 180 / pi
        if(tempx > 0.0 ):
            e_1c_BBp = e_1c_BBp + [check2]
        elif(tempx < 0.0):
            e_1c_BOBp  = e_1c_BOBp  + [check2]
        k = k + 1

    del e_1c_BOBp[e_1c_BOBp.index(0)]
    del e_1c_BBp[e_1c_BBp.index(0)]

##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
    k, m  = 0, 0
    while k in range(len(f_2c)):
        check = f_2c[k]
        m  = 0
        while m in range(len(node1)):
            check1 = node1[m]
            check2 = node2[m]
            check3 = node3[m]
            check4 = node4[m]
            check5 = node5[m]
            check6 = node6[m]
            check7 = node7[m]
            check8 = node8[m]
            if check in e_2c_n:
                pass
            else:
                if((check == check1) or (check == check2) or (check ==
check3) or (check == check4)
                   or (check == check5) or (check == check6) or (check ==
check7) or (check == check8)):
                    check9 = num_el[m]
                    e_2c_el = e_2c_el + [check9]
                    e_2c_n = e_2c_n + [check]
```

```
            m = m + 1
        k = k + 1
    del e_2c_el[e_2c_el.index(0)]
    del e_2c_n[e_2c_n.index(0)]
##--------------------------------------------------------------------------
    k, m = 0, 0
    global e_2c_DODp, e_2c_DDp, pi
    e_2c_DODp, e_2c_DDp = [0], [0]
    while k in range(len(e_2c_el)):
        check = e_2c_el[k]
        check2 = e_2c_n[k]
        t1, t2, t3, t4 = node1[check - 1], node2[check - 1], node3[check -
1], node4[check - 1]
        t5, t6, t7, t8 = node5[check - 1], node6[check - 1], node7[check -
1], node8[check - 1]
        tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1], x[t4 - 1]
        tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1], x[t8 - 1]
        ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1], y[t4 - 1]
        ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1], y[t8 - 1]
        tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7 + tx8)
        tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7 + ty8)
        tempxy = atan(tempy/tempx) * 180 / pi
        if(tempx < 0.0 ):
            e_2c_DODp = e_2c_DODp + [check2]
        elif(tempx > 0.0 ):
            e_2c_DDp  = e_2c_DDp  + [check2]

        k = k + 1
    del e_2c_DODp[e_2c_DODp.index(0)]
    del e_2c_DDp[e_2c_DDp.index(0)]
##--------------------------------------------------------------------------
##
    global f_12c_BDp, ct_f_12c
    f_12c_BDp, ct_f_12c = [0], 0
    k,m = 0,0
    while k in range(len(e_1c_BBp)):
        check = e_1c_BBp[k]
        tx1, ty1, tz1 = x[check-1], y[check-1], z[check-1]
        m = 0
        while m in range(len(e_2c_DDp)):
            check2 = e_2c_DDp[m]
            tx2, ty2, tz2 = x[check2-1], y[check2-1], z[check2-1]
            if(abs(tz1-tz2)<=shf):
                ct_f_12c = ct_f_12c + 2
                f_12c_BDp = f_12c_BDp + [check]
                f_12c_BDp = f_12c_BDp + [check2]
            m = m + 1
        k = k + 1
    del f_12c_BDp[f_12c_BDp.index(0)]
    if((ct_f_12c/len(e_2c_DDp) != 2) and (ct_f_12c/len(e_1c_BBp) != 2)):
        print '  Error in identifying face pairs -  1 C and 2 C '
    else:
        print '  Face pairs 1 C and 2 C are identified '
##--------------------------------------------------------------------------
##--------------------------------------------------------------------------
##
    global f_12c_BODp, ct_f_12cd
```

```
    f_12c_BODp, ct_f_12cd = [0], 0
    k,m = 0,0
    while k in range(len(e_1c_BOBp)):
        check = e_1c_BOBp[k]
        tx1, ty1, tz1 = x[check-1], y[check-1], z[check-1]
        m = 0
        while m in range(len(e_2c_DODp)):
            check2 = e_2c_DODp[m]
            tx2, ty2, tz2 = x[check2-1], y[check2-1], z[check2-1]
            if(abs(tz1-tz2)<=shf):
                ct_f_12cd = ct_f_12cd + 2
                f_12c_BODp = f_12c_BODp + [check]
                f_12c_BODp = f_12c_BODp + [check2]
            m = m + 1
        k = k + 1
    del f_12c_BODp[f_12c_BODp.index(0)]
    if((ct_f_12cd/len(e_2c_DODp) != 2) and (ct_f_12cd/len(e_1c_BOBp) != 2)):
        print '  Error in identifying face pairs -  1 C and 2 C '
    else:
        print '  Face pairs 1 C and 2 C are identified '
##-------------------------------------------------------------------------
def face_constraints():
    k,m = 0,0
    while k in range(len(fp_12)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 1 and 2 are written'
##-----------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_34)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
```

```
            strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 3 and 4 are written'
##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_12c_BDp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BDp[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BDp[k+1])+',' +  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 3 and 4 are written'

##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_12c_BODp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k+1])+',' +  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 3 and 4 are written'
##---------------------------------------------------------------------------
##---------------------------------------------------------------------------
def corner_constraints():
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(3) + '\n'
```

```
    output_1.write(strgg)
    strgg = ' ' + str(c146)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(3) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c236)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(4) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c136)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(3) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c145)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c245)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
```

```
        strgg = str(3) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c235)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c245)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(4) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c245)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    print '  Corner Constraints are written'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def edge_constraints():
    k,m = 0,0
    while k in range(len(e_1626)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1626[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1626[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 16 and 26'

    k,m = 0,0
    while k in range(len(e_1525)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
```

```
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1525[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1525[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 15 and 25'

    k,m = 0,0
    while k in range(len(e_3545)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3545[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3545[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 35 and 45'

    k,m = 0,0
    while k in range(len(e_3646)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 36 and 46'
```

```
    k,m = 0,0
    while k in range(len(e_1424)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1424[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1424[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 14 and 24'

    k,m = 0,0
    while k in range(len(e_2324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 23 and 24'

    k,m = 0,0
    while k in range(len(e_1324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1324[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1324[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
```

```
            strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 13 and 24'

##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def center_constraints():
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_1)+',' +  '+ str(m+1)+','+  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_3)+',' +  '+ str(m+1)+','+  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    print '  Face Center Constraints are written'
##-----------------------------------------------------------------------------
## CODE STARTS FOR LUMPED INNER LAYERS WITH ZERO ORIENTATION
##-----------------------------------------------------------------------------
def corner_constraints_lump():
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(4) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c146)+',' +  '+ str(m+1)+','+  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c246)+',' +  '+ str(m+1)+','+  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_21)+',' +  '+ str(m+1)+','+  '+str(1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_22)+',' +  '+ str(m+1)+','+  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1
```

```
m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(3) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c236)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(5) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c136)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(4) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c145)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c245)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(3) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c235)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
```

```
        strgg = ' ' + str(c245)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(5) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c245)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    print '  Lumped - Corner Constraints are written'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def edge_constraints_lump():
    k,m = 0,0
    while k in range(len(e_1626)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1626[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1626[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 16 and 26'

    k,m = 0,0
    while k in range(len(e_1525)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
```

```
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1525[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1525[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 15 and 25'

    k,m = 0,0
    while k in range(len(e_3545)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3545[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3545[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 35 and 45'

    k,m = 0,0
    while k in range(len(e_3646)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
```

```
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 36 and 46'

    k,m = 0,0
    while k in range(len(e_1424)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1424[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1424[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 14 and 24'

    k,m = 0,0
    while k in range(len(e_2324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 23 and 24'

    k,m = 0,0
    while k in range(len(e_1324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(5) + '\n'
            output_1.write(strgg)
```

```
            strgg = ' ' + str(e_1324[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1324[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +  '+ str(m+1)+','+  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +  '+ str(m+1)+','+  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 13 and 24'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def center_constraints_lump():
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_3)+',' +  '+ str(m+1)+','+  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_11)+',' +  '+ str(m+1)+','+  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_22)+',' +  '+ str(m+1)+','+  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_12)+',' +  '+ str(m+1)+','+  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_21)+',' +  '+ str(m+1)+','+  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1
```

```
    print '  Lumped - Face Center Constraints are written'
##-------------------------------------------------------------------------
def face_constraints_lump():
    k,m = 0,0
    while k in range(len(fp_12)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Face Constraints for faces 1 and 2 are written'
##-----------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_34)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Face Constraints for faces 3 and 4 are written'
##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_12c_BDp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
```

```
            strgg = ' ' + str(f_12c_BDp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BDp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Face Constraints for faces 3 and 4 are written'

##_____________________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_12c_BODp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Face Constraints for faces 3 and 4 are written'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def rigid_body_modes():
    strgg ='** Boundary conditions written\n'
    output_1.write(strgg)
    strgg='*Boundary\n'
    output_1.write(strgg)

    if(lumped_inner == 1):
        if((load_case == 1) or (thermal == 1)):
            strgg = ' n_set_4'+','+' '+str(2) + ','+ ' '+ str(3)+'\n'
            output_1.write(strgg)
            strgg = ' n_set_7'+','+' '+str(2) + ','+ ' '+ str(2)+'\n'
            output_1.write(strgg)
            strgg = ' n_set_3'+','+' '+str(1) + ','+ ' '+ str(2)+'\n'
            output_1.write(strgg)
            strgg = ' n_set_15'+','+' '+str(1) + ','+ ' '+ str(1)+'\n'
```

```
            output_1.write(strgg)

        elif(load_case == 2):
            strgg = ' n_set_12'+','+' '+str(1) + ','+ ' '+ str(2)+'\n'
            output_1.write(strgg)
            strgg = ' n_set_14'+','+' '+str(1) + ','+ ' '+ str(2)+'\n'
            output_1.write(strgg)
            strgg = ' n_set_15'+','+' '+str(1) + ','+ ' '+ str(1)+'\n'
            output_1.write(strgg)
            strgg = ' n_set_15'+','+' '+str(3) + ','+ ' '+ str(3)+'\n'
            output_1.write(strgg)

    else:
        strgg = ' n_set_5'+','+' '+str(1) + ','+ ' '+ str(3)+'\n'
        output_1.write(strgg)
        strgg = ' n_set_2'+','+' '+str(1) + ','+ ' '+ str(1)+'\n'
        output_1.write(strgg)
        strgg = ' n_set_2'+','+' '+str(3) + ','+ ' '+ str(3)+'\n'
        output_1.write(strgg)
        if(outerlayer == 1):
            strgg = ' n_set_4'+','+' '+str(3) + ','+ ' '+ str(3)+'\n'
            output_1.write(strgg)
        else:
            strgg = ' n_set_3'+','+' '+str(1) + ','+ ' '+ str(1)+'\n'
            output_1.write(strgg)
    print 'Rigid body modes are written'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def loads():
    if(mechanical == 1):
        strgg = '*Boundary \n'
        output_1.write(strgg)
        if((load_case == 1) and (lumped_inner == 0)):
            strgg = ' n_set_4'+','+' '+ str(1) + ',,'+ ' '+ str(-1e-6)+'\n'
            output_1.write(strgg)
        elif((load_case == 2) and (lumped_inner == 0)):
            strgg = ' n_set_4'+','+' '+ str(2) + ',,'+ ' '+ str(-1e-6)+'\n'
            output_1.write(strgg)
        elif((load_case == 1) and (lumped_inner == 1)):
        ## This part is completely done and corrected.
            strgg = ' n_set_4'+','+' '+ str(1) + ',,'+ ' '+ str(-1e-6)+'\n'
            output_1.write(strgg)
        elif((load_case == 2) and (lumped_inner == 1)):
        ## This part is completely done and corrected.
            strgg = ' n_set_4'+','+' '+ str(2) + ',,'+ ' '+ str(1e-6)+'\n'
            output_1.write(strgg)

    ## This part is not required as the thermal properties do
    ## do not change.
    if(thermal == 1):
        strgg = '*Temperature \n'
        output_1.write(strgg)
        strgg = ' n_set_9' + ','+' '+ str(theta_range)+'\n'
        output_1.write(strgg)
        print 'Thermal Loading Only'
```

```
##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
from abaqus import *
from abaqusConstants import *
from section import*
import math
reading_inp_file()
```

Code for file: ***run_ninety_2.py***

```
# this program writes the boundary conditions for the
# *.inp file.
#
#
##  Note: this program requires the information from run.py
output_1 = open('runfile.inp', 'w')
output_2 = open('runfile_2.inp', 'w')
output_3 = open('element_sets.inp', 'w')

##_____________________________________________________________________________
##_____________________________________________________________________________
##-------------------------------------------------------------------------
def reading_inp_file():
    global num_node, inte, intee, num_elem, inte_el
    global x, y, z, num_n, xa, xb, ya, yb, za, zb, xab, yab, zab
    global node1, node2, node3, node4, node5, node6, node7, node8, num_el
    global tem, tm, nola, crack_layer, thickness, crack_angle, outer_layer
    global num_ori, bc3, bc4, num_n
    x, y, z, num_n = [0], [0], [0], [0]
    node1, node2, node3, node4= [0], [0], [0], [0]
    node5, node6, node7, node8 = [0], [0], [0], [0]
    num_el = [0]
    inte, intee = 0, 0
    endfile, node_flag, elem_flag = 0, 0, 0 # logical operators
    num_node, num_elem = 0, 0 # initializing nodes and elements
    num_ori = 0

    nola = nol # number of layers
    crack_layer = loc_ply_1
    thickness = ped
    crack_angle = layup[loc_ply_1-1]

    outer_layer = 0 ## the cracked layer is not an outer layer if the value
is one.

    input = open('Lami_Job.inp', 'r')
    strg = input.readlines()

    while inte in range(len(strg)):
        strg_2 = strg[inte]

        if(strg_2[1:8]=='Element'):
##            output_1.write(strg_2)
            print 'Reading Elements'
            int_el = inte
            while(elem_flag != 1):
```

```
                int_el = int_el + 1
                strg_3 = strg[int_el]
                if(strg_3[0]=='*'):
                    elem_flag = 1
##                    strg_temp = '** End of Elements in the file\n'
##                    output_1.write(strg_temp)

                else:
                    num_elem = num_elem + 1
                    elem_reading(na=num_elem, stt= strg_3)
##                    output_1.write(strg_3)
                    num_el = num_el + [num_elem]
                    node1 = node1 + [nod1]
                    node2 = node2 + [nod2]
                    node3 = node3 + [nod3]
                    node4 = node4 + [nod4]
                    node5 = node5 + [nod5]
                    node6 = node6 + [nod6]
                    node7 = node7 + [nod7]
                    node8 = node8 + [nod8]
                    inte = int_el
                    nn = num_elem
            del num_el[num_el.index(0)]
            del node1[node1.index(0)]
            del node2[node2.index(0)]
            del node3[node3.index(0)]
            del node4[node4.index(0)]
            del node5[node5.index(0)]
            del node6[node6.index(0)]
            del node7[node7.index(0)]
            del node8[node8.index(0)]

            if(gap_elements == 1) :
                gap_element()
            nodes_elements() ## writing the nodes and elements.
            if(gap_elements == 1) :
                gap_elements_written()
            face_pairs()

        elif(strg_2[1:5] == 'Node'):
##            output_1.write(strg_2)
            print 'Reading Nodes'
            intee = inte
            while (node_flag != 1):
                intee = intee + 1
                strg_2 = strg[intee]
                if(strg_2[0]=='*'):
                    node_flag = 1
##                    strg_temp = '** End of Nodes\n'
##                    output_1.write(strg_temp)
                else:
                    num_node = num_node + 1
                    node_reading(n=num_node, st=strg_2)
                    num_n = num_n + [num_node]
                    x = x + [x1]
                    y = y + [y1]
                    z = z + [z1]
```

```
                    nn = num_node
##                    strrg = '   '+str(num_n[nn])+','+'   '+ str(x[nn]) +
','+'   ' + str(y[nn]) + ','+'   ' + str(z[nn]) + '\n'
##                    output_1.write(strrg)
                    inte = intee
            del x[x.index(0)]
            del y[y.index(0)]
            del z[z.index(0)]
            del num_n[num_n.index(0)]
            xa, xb, ya, yb, za, zb = max(x), min(x), max(y), min(y), max(z),
min(z)
            xab = (xa + xb)/2.0
            yab = (ya + yb)/2.0
            zab = (za + zb)/2.0
            cal_shf()
            node_swap(nt = num_n, xt = x, yt = y, zt = z)

        elif(strg_2[0:2]=='**'):
            output_1.write(strg[inte])

        elif(strg_2[0:9]=='*Boundary'):
            rigid_body_modes()
            inte = inte + 1

        elif(strg_2[0:8]=='*Restart'):
            print 'Eliminating Rigid Body Modes and writing Loads'
            output_1.write(strg_2)

        elif(strg_2[0:6]=='*Cload'):
            print 'Writing Loads'
            loads()
            inte = inte + 1

        elif(strg_2[0:13]=='*End Instance'):

            print 'Writing Constraints'

            if(lumped_inner == 1):
                corner_constraints_lump()
                edge_constraints_lump()
                face_constraints_lump()
                center_constraints_lump()

            elif(outerlayer == 1):
                corner_constraints_outer()
                edge_constraints_outer()
                face_constraints_outer()
                center_constraints_outer()

            else:
                corner_constraints()
                edge_constraints()
                face_constraints()
                center_constraints()

            if(outerlayer == 1) and (crack_surfaces == 1):
                print ' SURFACE CONTACT Surfaces ARE WRITTEN'
```

```
                contact_surfaces()
            output_1.write(strg_2)

## this is to include the surface contact.
##        elif(strg_2[0:9]=='*End Part'):


        elif(strg_2[0:6] == '*Elset'):
            output_1.write(strg_2)
            output_3.write(strg_2)
            intel = inte
            flag = 0
            while(flag == 0):
                intel = intel + 1
                strg_5 = strg[intel]
                if(strg_5[0] != '*'):
                    output_1.write(strg_5)
                    output_3.write(strg_5)
                else:
                    flag = 1
                    inte = intel - 1

        elif(strg_2[0:14]=='*Solid Section'):
            output_1.write(strg_2)
            output_3.write(strg_2)
            intel = inte
            flag = 0
            while(flag == 0):
                intel = intel + 1
                strg_5 = strg[intel]
                if(strg_5[0] != '*'):
                    output_1.write(strg_5)
                    output_3.write(strg_5)
                else:
                    flag = 1
                    inte = intel - 1

        elif(strg_2[1:7]=='Orient'):
            output_1.write(strg_2)
            output_3.write(strg_2)
            intel = inte
            flag = 0
            num_ori = num_ori + 1
            while(flag == 0):
                intel = intel + 1
                strg_5 = strg[intel]
                if(strg_5[0] != '*'):
                    output_1.write(strg_5)
                    output_3.write(strg_5)
                else:
                    flag = 1
                    inte = intel - 1

        elif(strg_2[1:13]=='End Assembly'):

            strgg = '*Nset, nset=n_set_1, instance=sublami \n'
            output_1.write(strgg)
```

```
            strgg = str(c_1)+'\n'
            output_1.write(str(strgg))
            strgg = '*Nset, nset=n_set_2, instance=sublami \n'
            output_1.write(strgg)
            strgg = str(c_2)+'\n'
            output_1.write(strgg)
          output_1.write('*Nset, nset=n_set_3, instance=sublami \n')
            strgg = str(c_5)+'\n'
            output_1.write(strgg)
          output_1.write('*Nset, nset=n_set_4, instance=sublami \n')
          strgg = str(c_4)+'\n'
            output_1.write(strgg)
            output_1.write('*Nset, nset=n_set_5, instance=sublami \n')
          strgg = str(bodyc)+'\n'
            output_1.write(strgg)
            output_1.write('*Nset, nset=n_set_6, instance=sublami \n')
          strgg = str(c_6)+'\n'
            output_1.write(strgg)
            output_1.write('*Nset, nset=n_set_7, instance=sublami \n')
          strgg = str(c_6_x)+'\n'
            output_1.write(strgg)
            output_1.write('*Nset, nset=n_set_8, instance=sublami \n')
          strgg = str(c_6_y)+'\n'
            output_1.write(strgg)

            output_1.write('*Nset, nset=n_set_9, instance=sublami, internal,
generate \n')
          strgg = str(1)+','+'  '+ str(num_node)+','+'  '+str(1)+'\n'
            output_1.write(strgg)

            output_1.write('*Nset, nset=n_set_10, instance=sublami \n')
          strgg = str(c_14m)+'\n'
            output_1.write(strgg)

            output_1.write('*Nset, nset=n_set_11, instance=sublami \n')
          strgg = str(c_23m)+'\n'
            output_1.write(strgg)

            if(lumped_inner == 1):
                output_1.write('*Nset, nset=n_set_12, instance=sublami \n')
                strgg = str(c_21)+'\n'
                output_1.write(strgg)
                output_1.write('*Nset, nset=n_set_14, instance=sublami \n')
                strgg = str(c_22)+'\n'
                output_1.write(strgg)
                output_1.write('*Nset, nset=n_set_15, instance=sublami \n')
                strgg = str(bc2)+'\n'
                output_1.write(strgg)
                output_1.write('*Nset, nset=n_set_16, instance=sublami \n')
                strgg = str(bc3)+'\n'
                output_1.write(strgg)
                output_1.write('*Nset, nset=n_set_17, instance=sublami \n')
                strgg = str(bc4)+'\n'
                output_1.write(strgg)

            if(outerlayer == 1):
                output_1.write('*Nset, nset=n_set_15, instance=sublami \n')
```

```
                strgg = str(bc2)+'\n'
                output_1.write(strgg)
                output_1.write('*Nset, nset=n_set_61, instance=sublami \n')
                strgg = str(c_61)+'\n'
                output_1.write(strgg)
                output_1.write('*Nset, nset=n_set_62, instance=sublami \n')
                strgg = str(c_62)+'\n'
                output_1.write(strgg)



          output_1.write(strg_2)
          if(outerlayer == 1) and (crack_surfaces == 1):
                print ' SURFACE CONTACT CONSTRAINTS ARE WRITTEN'
                crack_contact()


          print 'Nsets for rigid body modes and loads - created'

        elif(strg_2[1:14]=='Output, field'):
              if(gap_elements == 1) or (crack_surfaces == 1):
                  strgg = '*PRINT, CONTACT = YES\n'
                  output_1.write(strgg)
              output_1.write(strg_2)
              output_1.write('*Element output,position=centroid \n')
              strgg = 'Evol, S \n'
              output_1.write(strgg)

          else:
              output_1.write(strg[inte])
          inte = inte + 1
      else:
          print 'End of File'

      output_1.close()
      output_2.close()
      output_3.close()
##_____________________________________________________________________________
##-----------------------------------------------------------------------------
def cal_shf():
    global shf
    temp_x, temp_y, temp_z  = [0], [0], [0]
    for j in range(len(x)):
        tt = abs(x[j])/10000.0
        tt = str(tt)
        temp_x = temp_x + [int(len(tt))]

        tt = abs(y[j])/10000.0
        tt = str(tt)
        temp_y = temp_y + [int(len(tt))]

        tt = abs(z[j])/10000.0
        tt = str(tt)
        temp_z = temp_z + [int(len(tt))]

    del temp_x[temp_x.index(0)]
```

```
    del temp_y[temp_y.index(0)]
    del temp_z[temp_z.index(0)]

    tx_i, ty_i = temp_x.index(max(temp_x)), temp_y.index(max(temp_y))
    tz_i = temp_z.index(max(temp_z))

    tx_max = x[tx_i]
    ty_max = y[ty_i]
    tz_max = z[tz_i]
    tt = [tx_max, ty_max, tz_max]
    ttt = [len(str(tx_max)), len(str(ty_max)), len(str(tz_max))]
    tttt = str(tt[ttt.index(max(ttt))])
    tttt = tttt.replace('.', ' ')
    tttt = tttt.split()
    t5 = tttt[1]
    t5 = len(t5)
    sh = '1E-'+str(t5-3)
    shf = float(sh)

##---------------------------------------------------------------------------
def node_reading(n, st):
    global num_n1, x1, y1, z1
    num_n1 = n
    x1, y1, z1 = 0., 0., 0.
    n_ck = 0

    strg_n = st.replace(',', ' ')
    strg_n = strg_n.split()
    n_k = int(strg_n[0])
    x1 = float(strg_n[1])
    y1 = float(strg_n[2])
    z1 = float(strg_n[3])

#---------------------------------------------------------------------------
def elem_reading(na, stt):
    global nod1, nod2, nod3, nod4, nod5, nod6, nod7, nod8
    nod1, nod2, nod3, nod4, nod5, nod6, nod7, nod8  = 0, 0, 0, 0, 0, 0, 0, 0

    strg_n = stt.replace(',', ' ')
    strg_n = strg_n.split()
    n_k = int(strg_n[0])
    nod1 = int(strg_n[1])
    nod2 = int(strg_n[2])
    nod3 = int(strg_n[3])
    nod4 = int(strg_n[4])
    nod5 = int(strg_n[5])
    nod6 = int(strg_n[6])
    nod7 = int(strg_n[7])
    nod8 = int(strg_n[8])

#---------------------------------------------------------------------------
def node_swap(nt, xt, yt, zt):
## this subroutine detects the center nodes, corner nodes, edge nodes and
face nodes.
    global tem, tm
    global c_1, c_2, c_3, c_4, c_5, c_6, c_1_1, c_1_2, bodyc, bc2, bc3, bc4
    global c135, c136, c146, c145, c235, c236, c246, c245
```

```
    global c146_2, c236_2, c145_2, c235_2
    global ct_35, ct_36, ct_45, ct_46
    global e_35, e_36, e_45, e_46
    global e_15, e_16, e_25, e_26
    global e_13, e_14, e_23, e_24
    global f_1, f_2, f_3, f_4, shf
    global c_6_x, c_6_y, c_14m, c_23m
    global c_11, c_12, c_21, c_22, c_21, c_61, c_62, c_51, c_52
    global ck_z_1, ck_z_2, ck_z_3, ck_z_4
    global e_15c_11, e_15c_22, e_16c_11, e_16c_22
    global e_25c_11, e_25c_22, e_26c_11, e_26c_22

    ck_z_1 = thick[loc_ply_1 - 1]
    ck_z_2 = thick[loc_ply_1]
    locaa = nol - loc_ply_1 + 1
    ck_z_3 = thick[locaa - 1]
    ck_z_4 = thick[locaa]

    j  = 0
    c_1, c_2, c_3, c_4, c_5, c_6 = 0,0,0,0,0,0
    c_11, c_12, c_21, c_22 = 0, 0, 0, 0
    c_1_1, c_1_2 = 0,0
    c135, c136, c146, c145 = 0,0,0,0
    c235, c236, c246, c245 = 0,0,0,0
    c_14m, c_23m = 0 , 0
    c146_2, c145_2, c235_2, c236_2 = 0,0,0,0
    e_15c_11, e_15c_22, e_25c_11, e_25c_22 = 0, 0, 0, 0
    e_16c_11, e_16c_22, e_26c_11, e_26c_22 = 0, 0, 0, 0
    trip_c146, trip_c236, trip_c145, trip_c235 = 0, 0, 0, 0
    trip_c1511, trip_c1611 = 0, 0
    trip_c2511, trip_c2611 = 0, 0
    trip_c61 = 0
    trip_c51 = 0

    while j in range(len(nt)):
        tx, ty, tz = x[j], y[j], z[j]

        if((abs(tx-xab)<= shf)and(abs(ty-yb)<=shf)and(abs(tz-zab)<=shf)):
            c_1 = nt[j]
            print 'Face center - 1', c_1
        elif((abs(xab-tx)<=shf)and(abs(ty-ya)<=shf)and(abs(tz-zab)<=shf)):
            c_2 = nt[j]
            print 'Face center - 2', c_2
        elif((abs(tx-xa)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-zab)<=shf)):
          c_3 = nt[j]
            print 'Face center - 3', c_3
        elif((abs(tx-xb)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-zab)<=shf)):
            c_4 = nt[j]
          print 'Face center - 4', c_4

      elif((abs(tx-xab)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-zb)<=shf) and
(outerlayer == 0)):
            c_5 = nt[j]
            print 'Face center - 5', c_5
        elif((abs(tx-xab)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-za)<=shf) and
(outerlayer == 0)):
            c_6 = nt[j]
```

```
            print 'Face center - 6', c_6

      elif((abs(xab-tx)<=shf)and(abs(ty-ya)<=shf)and(abs(tz-ck_z_1)<=shf)):
            c_21 = nt[j]
            print 'Face center - 2 1', c_21

      elif((abs(xa-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(zb-tz)<=shf)):
          c135 = nt[j]
          print 'Corner 135', c135
      elif((abs(xa-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(za-tz)<=shf)):
          c136 = nt[j]
            print 'Corner 136', c136

      elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(za-tz)<=shf)and
(trip_c146 == 0)):
          c146 = nt[j]
          trip_c146 = 1
            print 'Corner 146', c146
      elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(za-tz)<=shf) and
(trip_c146 == 1)):
          c146_2 = nt[j]
            print 'Corner 146 - 2 ', c146_2
      elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(zb-tz)<=shf) and
(trip_c145 == 0)):
          c145 = nt[j]
          trip_c145 = 1
            print 'Corner 145', c145
        elif((abs(xb-tx)<=shf)and(abs(yb-ty)<=shf)and(abs(zb-tz)<=shf) and
(trip_c145 == 1)):
          c145_2 = nt[j]
            print 'Corner 145 - 2', c145_2
      elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(zb-tz)<=shf) and
(trip_c235 == 0)):
          c235 = nt[j]
          trip_c235 = 1
            print 'Corner 235', c235
        elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(zb-tz)<=shf) and
(trip_c235 == 1)):
          c235_2 = nt[j]
            print 'Corner 235 - 2', c235_2
      elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(za-tz)<=shf) and
(trip_c236 == 0)):
          c236 = nt[j]
          trip_c236 = 1
            print 'Corner 236', c236
      elif((abs(xa-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(za-tz)<=shf) and
(trip_c236 == 1)):
          c236_2 = nt[j]
            print 'Corner 236 - 2', c236_2

      elif((abs(xb-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(za-tz)<=shf)):
          c246 = nt[j]
          print 'Corner 246', c246
      elif((abs(xb-tx)<=shf)and(abs(ya-ty)<=shf)and(abs(zb-tz)<=shf)):
          c245 = nt[j]
            print 'Corner 245', c245
##        elif((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-zb)<=shf)):
```

```
##            c_1_1 = nt[j]
##         elif((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-za)<=shf)):
##            c_1_2 = nt[j]
        elif((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-zab)<=shf)):
            bodyc = nt[j]

        elif((abs(tx-xb)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-za)<=shf)):
            c_6_x = nt[j]
        elif((abs(xab-tx)<=shf)and(abs(ty-ya)<=shf)and(abs(tz-za)<=shf)):
            c_6_y = nt[j]

        elif((abs(xb-tx)<=shf)and(abs(ty-yb)<=shf)and(abs(tz-zab)<=shf)):
            c_14m = nt[j]
        elif((abs(xa-tx)<=shf)and(abs(ty-ya)<=shf)and(abs(tz-zab)<=shf)):
            c_23m = nt[j]
## the lumped zero layers include the center node.  Two nodes of equal
distance from the
## center node need to be taken for the two center nodes on the two opposite
faces.
## c_11, c_12, c_21, c_22.

        if(lumped_inner == 1):
            if((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-
ck_z_4)<=shf)):
                bc2 = nt[j]
                print 'Body center - 2', bc2

            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf)and(abs(tz-
ck_z_1)<=shf)):
                c_11 = nt[j]
                print 'Face center - 31 ', c_11
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf)and(abs(tz-
ck_z_4)<=shf)):
                c_12 = nt[j]
                print 'Face center - 32 ', c_12

            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_1)<=shf)):
                c_21 = nt[j]
                print 'Face center - 41 ', c_21
            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_4)<=shf)):
                c_22 = nt[j]
                print 'Face center - 42 ', c_22

            elif((abs(tx-xab)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-
ck_z_1)<=shf)):
                bc3 = nt[j]
                print 'New BC - ', bc3
            elif((abs(tx-xb)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-zab)<=shf)):
                bc4 = nt[j]
                print 'New BC - ', bc4

      if(outerlayer == 1):
            if((abs(xab-tx)<=shf)and(abs(ty-yab)<=shf)and(abs(tz-
ck_z_3)<=shf)):
                bc2 = nt[j]
```

```
                print 'Body center - 2', bc2
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf) and (abs(tz-
ck_z_1)<=shf) and (trip_c1511 == 0)):
                e_15c_11 = nt[j]
                trip_c1511 = 1
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf) and (abs(tz-
ck_z_1)<=shf) and (trip_c1511 == 1)):
                e_15c_22 = nt[j]
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf)and(abs(tz-
ck_z_4)<=shf) and (trip_c1611 == 0)):
                e_16c_11 = nt[j]
                trip_c1611 = 1
            elif((abs(tx-xab)<=shf)and(abs(yb-ty)<=shf)and(abs(tz-
ck_z_4)<=shf) and (trip_c1611 == 1)):
                e_16c_22 = nt[j]

            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_1)<=shf) and (trip_c2511 == 0)):
                e_25c_11 = nt[j]
                trip_c2511 = 1
            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_1)<=shf) and (trip_c2511 == 1)):
                e_25c_22 = nt[j]
            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_4)<=shf) and (trip_c2611 == 0)):
                e_26c_11 = nt[j]
                trip_c2611 = 1
            elif((abs(tx-xab)<=shf)and(abs(ya-ty)<=shf)and(abs(tz-
ck_z_4)<=shf) and (trip_c2611 == 1)):
                e_26c_22 = nt[j]

## the faces 5 and 6 have cracks and so have two nodes.
            elif((abs(tx-xb/2.0)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-
zb)<=shf)):
                c_51 = nt[j]
                trip_c51 = 1
                print 'Face center - 5 - 1 -', c_51
            elif((abs(tx-xa/2.0)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-
zb)<=shf)):
                c_52 = nt[j]
                print 'Face center - 5 - 2 -', c_52

            elif((abs(tx-xb/2.0)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-
za)<=shf)):
                c_61 = nt[j]
                trip_c61 = 1
                print 'Face center - 6 - 1 -', c_61

            elif((abs(tx-xa/2.0)<=shf)and(abs(yab-ty)<=shf)and(abs(tz-
za)<=shf)):
                c_62 = nt[j]
                print 'Face center - 6 - 2 -', c_62

        j = j + 1

    if((outerlayer == 0) and (lumped_inner == 0)):
        tm = [c135, c136, c146, c145, c235, c236, c246, c245]
```

```
        tem = [c_1, c_2, c_3, c_4, c_5, c_6]

    elif((outerlayer == 1) and (lumped_inner == 0)):
        tm = [c135, c136, c146, c145, c235, c236, c246, c245,
              e_15c_11, e_25c_11,  e_16c_11, e_26c_11, e_25c_22, e_26c_22,
e_15c_22,  e_16c_22]
        tem = [c_1, c_2, c_3, c_4, c_51, c_52, c_61, c_62]

    elif(lumped_inner == 1):
        tm = [c135, c136, c146, c145, c235, c236, c246, c245,
              c_11, c_12, c_21, c_22]
        tem = [c_3, c_4, c_5, c_6]

    if(0 in tm):
        print 'Zero in tm'
    else:
        print 'TM is complete '

##    print tm
    print 'Center nodes and Corner nodes are identified'
##_____________________________________________________________________________
##-----------------------------------------------------------------------------
## subroutine to detect the edge node 35 36 45 46

    ct_35, ct_36, ct_45, ct_46 = 0, 0, 0, 0
    ct_15, ct_16, ct_25, ct_26 = 0, 0, 0, 0
    ct_13, ct_14, ct_23, ct_24 = 0, 0, 0, 0

    j = 0
    front, fronta, frontb, frontc = 0, 0, 0, 0
    frontd, fronte, frontf, frontg = 0, 0, 0, 0
    fronth, fronti, frontj, frontk = 0, 0, 0, 0

    e_35, e_36, e_45, e_46 = [0], [0], [0], [0]
    e_15, e_16, e_25, e_26 = [0], [0], [0], [0]
    e_13, e_14, e_23, e_24 = [0], [0], [0], [0]

    while j in range(len(nt)):
        tx, ty, tz = x[j], y[j], z[j]
        check = nt[j]
        front, fronta, frontb, frontc  = 0, 0, 0, 0
        frontd, fronte, frontf, frontg = 0, 0, 0, 0
        fronth, fronti, frontj, frontk = 0, 0, 0, 0

        if check in tem:
            front, fronta, frontb, frontc  = 1, 1, 1, 1
            frontd, fronte, frontf, frontg = 1, 1, 1, 1
            fronth, fronti, frontj, frontk = 1, 1, 1, 1
        if check in tm:
            front, fronta, frontb, frontc  = 1, 1, 1, 1
            frontd, fronte, frontf, frontg = 1, 1, 1, 1
            fronth, fronti, frontj, frontk = 1, 1, 1, 1

##       EDGE 35
      if((abs(tx-xa)<=shf)and(abs(tz-zb)<=shf)):
          if(front!=1):
                ct_35 = ct_35 + 1
```

```
            e_35 = e_35 + [check]

##     EDGE 36
      elif((abs(tx-xa)<=shf)and(abs(tz-za)<=shf)):
            if(fronta!=1):
                ct_36 = ct_36 + 1
                e_36 = e_36 + [check]

##      EDGE 45
        elif( (abs(tx-xb)<=shf)and(abs(tz-zb)<=shf) ):
            if(frontb!=1):
                ct_45 = ct_45 + 1
                e_45 = e_45 + [check]

##      EDGE 46
        elif((abs(tx-xb)<=shf)and(abs(tz-za)<=shf)):
            if(frontc!=1):
                ct_46 = ct_46 + 1
                e_46 = e_46 + [check]

##      EDGE 15
        elif((abs(ty-yb)<=shf)and(abs(tz-zb)<=shf)):
            if(frontd!=1):
                ct_15 = ct_15 + 1
            e_15 = e_15 + [check]

##      EDGE 25
      elif((abs(ty-ya)<=shf)and(abs(tz-zb)<=shf)):
            if(fronte!=1):
            ct_25 = ct_25 + 1
            e_25 = e_25 + [check]

##      EDGE 16
      elif((abs(ty-yb)<=shf)and(abs(tz-za)<=shf)):
            if(frontf!=1):
            ct_16 = ct_16 + 1
            e_16 = e_16 + [check]

##      EDGE 26
      elif((abs(ty-ya)<=shf)and(abs(tz-za)<=shf)):
          if(frontg!=1):
            ct_26 = ct_26 + 1
            e_26 = e_26 + [check]

##      EDGE 13
      elif((abs(tx-xa)<=shf)and(abs(ty-yb)<=shf)):
            if(fronth!=1):
                ct_13 = ct_13 + 1
                e_13 = e_13 + [check]

##      EDGE 14
        elif((abs(tx-xb)<=shf)and(abs(ty-yb)<=shf)):
            if(fronti!=1):
                ct_14 = ct_14 + 1
                e_14 = e_14 + [check]

##      EDGE 23
```

```
        elif((abs(tx-xa)<=shf)and(abs(ty-ya)<=shf)):
            if(frontj!=1):
                ct_23 = ct_23 + 1
                e_23 = e_23 + [check]

##      EDGE 24
        elif((abs(tx-xb)<=shf)and(abs(ty-ya)<=shf)):
            if(frontk!=1):
                ct_24 = ct_24 + 1
                e_24 = e_24 + [check]

        j = j + 1
    del e_35[e_35.index(0)]
    del e_36[e_36.index(0)]
    del e_45[e_45.index(0)]
    del e_46[e_46.index(0)]
    del e_15[e_15.index(0)]
    del e_16[e_16.index(0)]
    del e_25[e_25.index(0)]
    del e_26[e_26.index(0)]
    del e_13[e_13.index(0)]
    del e_23[e_23.index(0)]
    del e_14[e_14.index(0)]
    del e_24[e_24.index(0)]
    print 'Edge nodes are identified'
##-----------------------------------------------------------------------
## nodes on the edges are identified. These nodes have to be paired for
writing
## the constraint equations.  the node pairs are based on the boundary
conditions.
    global e_1626, e_1526, e_2526, e_3646, e_3546, e_4546
    global ct_e_1626, ct_e_1526, ct_e_3546, ct_e_3646, ct_e_4546, ct_e_2526
    k = 0
    m = 0
    e_1626, e_1526, e_3546, e_3646, e_4546, e_2526 = [0], [0], [0], [0], [0],
[0]
    ct_e_1626, ct_e_1526, ct_e_3646, ct_e_3546, ct_e_4546, ct_e_2526 = 0, 0,
0, 0, 0, 0
    if(ct_16 != ct_26):
            print 'The edges nodes are not compatible 16 26 -- check the
code'
    while k in range(len(e_16)):
        check1 = e_16[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_26)):
            check2 = e_26[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tx1-tx2)<=shf):
                ct_e_1626 = ct_e_1626 + 2
                e_1626 = e_1626 + [check1]
                e_1626 = e_1626 + [check2]
            m = m + 1
        k = k + 1
    del e_1626[e_1626.index(0)]
    if((ct_e_1626/ct_16 != 2) and (ct_e_1626/ct_26 != 2)):
        print '  Error in identifying edge pairs -  Edges 16 and 26'
```

```
    else:
        print '  Edge pairs 16 and 26 are identified'
##-------------------------------------------------
    k, m = 0, 0
    if(ct_15 != ct_26):
            print 'The edges nodes are not compatible 15 25 -- check the
code'
    while k in range(len(e_15)):
        check1 = e_15[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_26)):
            check2 = e_26[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tx1-tx2)<=shf):
                ct_e_1526 = ct_e_1526 + 2
                e_1526 = e_1526 + [check1]
                e_1526 = e_1526 + [check2]
            m = m + 1
        k = k + 1
    del e_1526[e_1526.index(0)]
    if((ct_e_1526/ct_15 != 2) and (ct_e_1526/ct_26 != 2)):
        print '  Error in identifying edge pairs -  Edges 15 and 26'
    else:
        print '  Edge pairs 15 and 26 are identified'
##-------------------------------------------------
    k, m = 0, 0
    if(ct_25 != ct_26):
            print 'The edges nodes are not compatible 15 25 -- check the
code'
    while k in range(len(e_25)):
        check1 = e_25[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_26)):
            check2 = e_26[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tx1-tx2)<=shf):
                ct_e_2526 = ct_e_2526 + 2
                e_2526 = e_2526 + [check1]
                e_2526 = e_2526 + [check2]
            m = m + 1
        k = k + 1
    del e_2526[e_2526.index(0)]
    if((ct_e_2526/ct_25 != 2) and (ct_e_2526/ct_26 != 2)):
        print '  Error in identifying edge pairs -  Edges 25 and 26'
    else:
        print '  Edge pairs 25 and 26 are identified'
##-------------------------------------------------
    k, m = 0, 0
    if(ct_35 != ct_46):
            print 'The edges nodes are not compatible  35 45 -- check the
code'
    while k in range(len(e_35)):
        check1 = e_35[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
```

```
        while m in range(len(e_46)):
            check2 = e_46[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_3546 = ct_e_3546 + 2
                e_3546 = e_3546 + [check1]
                e_3546 = e_3546 + [check2]
            m = m + 1
        k = k + 1
    del e_3546[e_3546.index(0)]
    if((ct_e_3546/ct_35 != 2) and (ct_e_3546/ct_46 != 2)):
        print '  Error in identifying edge pairs -  Edges 35 and 46'
    else:
        print '  Edge pairs 35 and 46 are identified'
##-----------------------------------------------------
    k, m = 0, 0
    if(ct_36 != ct_46):
            print 'The edges nodes are not compatible  36 46 -- check the
code'
    while k in range(len(e_36)):
        check1 = e_36[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_46)):
            check2 = e_46[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_3646 = ct_e_3646 + 2
                e_3646 = e_3646 + [check1]
                e_3646 = e_3646 + [check2]
            m = m + 1
        k = k + 1
    del e_3646[e_3646.index(0)]
    if((ct_e_3646/ct_36 != 2) and (ct_e_3646/ct_46 != 2)):
        print '  Error in identifying edge pairs -  Edges 36 and 46'
    else:
        print '  Edge pairs 36 and 46 are identified'
##-----------------------------------------------------
    k, m = 0, 0
    if(ct_45 != ct_46):
            print 'The edges nodes are not compatible  45 46 -- check the
code'
    while k in range(len(e_45)):
        check1 = e_45[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_46)):
            check2 = e_46[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(ty1-ty2)<=shf):
                ct_e_4546 = ct_e_4546 + 2
                e_4546 = e_4546 + [check1]
                e_4546 = e_4546 + [check2]
            m = m + 1
        k = k + 1
    del e_4546[e_4546.index(0)]
    if((ct_e_4546/ct_45 != 2) and (ct_e_4546/ct_46 != 2)):
```

```
        print '  Error in identifying edge pairs -  Edges 45 and 46'
    else:
        print '  Edge pairs 45 and 46 are identified'
##-----------------------------------------------------
##-----------------------------------------------------
    global ct_e_1424, ct_e_2324, ct_e_1324
    global e_1424, e_2324, e_1324
    e_1424, e_2324, e_1324 = [0], [0], [0]

    ct_e_1424, ct_e_2324, ct_e_1324 = 0, 0, 0
    k, m = 0, 0
    if(ct_14 != ct_24):
            print 'The edges nodes are noe_2324t compatible  14 24 -- check
the code'
    while k in range(len(e_14)):
        check1 = e_14[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_24)):
            check2 = e_24[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tz1-tz2)<=shf):
                ct_e_1424 = ct_e_1424 + 2
                e_1424 = e_1424 + [check1]
                e_1424 = e_1424 + [check2]
            m = m + 1
        k = k + 1
    del e_1424[e_1424.index(0)]
    if((ct_e_1424/ct_14 != 2) and (ct_e_1424/ct_24 != 2)):
        print '  Error in identifying edge pairs -  Edges 14 and 24 Uncracked
Region'
    else:
        print '  Edge pairs 14 and 24 are identified'
##-----------------------------------------------------
    k, m = 0, 0
    if(ct_23 != ct_24):
            print 'The edges nodes are not compatible  23 24 -- check the
code'
    while k in range(len(e_23)):
        check1 = e_23[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_24)):
            check2 = e_24[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if(abs(tz1-tz2)<=shf):
                ct_e_2324 = ct_e_2324 + 2
                e_2324 = e_2324 + [check1]
                e_2324 = e_2324 + [check2]
            m = m + 1
        k = k + 1
    del e_2324[e_2324.index(0)]
    if((ct_e_2324/ct_23 != 2) and (ct_e_2324/ct_24 != 2)):
        print '  Error in identifying edge pairs -  Edges 23 and 24 Uncracked
Region'
    else:
        print '  Edge pairs 23 and 24 are identified'
```

```
##-------------------------------------------------
    k, m = 0, 0
    if(ct_13 != ct_24):
            print 'The edges nodes are not compatible  13 24 -- check the
code'
    while k in range(len(e_13)):
        check1 = e_13[k]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(e_24)):
            check2 = e_24[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if((abs(tz1-tz2)<=shf) and (abs(ty1 + ty2)<= shf)):
                ct_e_1324 = ct_e_1324 + 2
                e_1324 = e_1324 + [check1]
                e_1324 = e_1324 + [check2]
            m = m + 1
        k = k + 1
    del e_1324[e_1324.index(0)]
    if((ct_e_1324/ct_13 != 2) and (ct_e_1324/ct_24 != 2)):
        print '  Error in identifying edge pairs -  Edges 13 and 24 Uncracked
Region'
    else:
        print '  Edge pairs 13 and 24 are identified '
##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
def nodes_elements():
    strgg = '*Node\n'
    output_1.write(strgg)
    nn = 0
    while nn in range(len(num_n)):
        strrg = '   '+str(num_n[nn])+','+'   '+ str(x[nn]) + ','+'   ' +
str(y[nn]) + ','+'   ' + str(z[nn]) + '\n'
        output_1.write(strrg)
        nn = nn + 1

    strgg = '*Element, type=C3D8I\n'
    output_1.write(strgg)
    nn = 0
    while nn in range(len(num_el)):
        strrg = '   '+str(num_el[nn])+','+'   '+ str(node1[nn]) + ','+'   ' +
str(node2[nn]) + ','+'   '
        strrg = strrg + str(node3[nn])+','+'   ' + str(node4[nn])+','+'   '
+str(node5[nn])+','+'   '
        strrg = strrg +str(node6[nn]) +','+'   '+str(node7[nn])+','+'   '
+str(node8[nn])+','+'   ' +'\n'
        output_1.write(strrg)
        nn = nn + 1

##-------------------------------------------------------------------------
def face_pairs():
## nodes on faces are recognized now. we do not need face 5 and face 6 nodes.
    global ct_f_1, ct_f_2, ct_f_3, ct_f_4, ct_f_5, ct_f_6, pi
    global f_1c, f_2c, ct_f_1c, ct_f_2c, f_3, f_4, f_1, f_2, f_5c, f_6c,
ct_f_5c, ct_f_6c
    ct_f_1, ct_f_2, ct_f_3, ct_f_4, ct_f_5, ct_f_6 = 0, 0, 0, 0, 0, 0
    f_1, f_2, f_3, f_4, f_5, f_6 = [0], [0], [0], [0], [0], [0]
```

```
    f_1c, f_2c = [0], [0]
    f_5c, f_6c = [0], [0]
    ct_f_1c, ct_f_2c = 0, 0
    ct_f_5c, ct_f_6c = 0, 0
    j = 0

    while j in range(len(num_n)):
        tx, ty, tz = x[j], y[j], z[j]
        check = num_n[j]
        frontl, frontm, frontn, fronto = 0, 0, 0, 0
        front_5, front_6 = 0, 0

        if check in tm:     frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in tem:    frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_35:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_36:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_45:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_46:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_15:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_25:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_16:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_26:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_13:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_23:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_14:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1
        if check in e_24:   frontl, frontm, frontn, fronto, front_5, front_6
= 1, 1, 1, 1, 1, 1

        if(abs(ty-yb)<=shf):

            if((outerlayer == 0) and (lumped_inner == 0)):
                if( ((tz>ck_z_1) and (tz<ck_z_2) and (abs(tx-xab)<=shf) and
not (abs(tz-ck_z_1)<=shf) and not(abs(tz-ck_z_2)<=shf))
                 or ((tz<ck_z_4) and (tz>ck_z_3) and (abs(tx-xab)<=shf) and
not (abs(tz-ck_z_3)<=shf) and not(abs(tz-ck_z_4)<=shf))):
                    if(frontl != 1):
                        ct_f_1c = ct_f_1c + 1
                        f_1c = f_1c + [check]
                else:
                    if(frontl != 1):
                        ct_f_1 = ct_f_1 + 1
                        f_1 = f_1 + [check]

            elif((outerlayer == 0) and (lumped_inner == 1)):
```

```
                if( (tz>ck_z_1) and (tz<ck_z_4) and (abs(tx-xab)<=shf) and
not((abs(tz-ck_z_1)<=shf) or(abs(tz-ck_z_4)<=shf)) ):
                    if(frontl != 1):
                        ct_f_1c = ct_f_1c + 1
                        f_1c = f_1c + [check]
                else:
                    if(frontl != 1):
                        ct_f_1 = ct_f_1 + 1
                        f_1 = f_1 + [check]

            elif((outerlayer == 1) and (lumped_inner == 0)):
                if( ( (tz>ck_z_1) and (tz<ck_z_2) and (abs(tx-xab)<=shf) and
not(abs(tz-ck_z_2)<=shf) )
                 or ( (tz<ck_z_4) and (tz>ck_z_3) and (abs(tx-xab)<=shf) and
not(abs(tz-ck_z_3)<=shf) ) ):
                    if(frontl != 1):
                        ct_f_1c = ct_f_1c + 1
                        f_1c = f_1c + [check]
                else:
                    if(frontl != 1):
                        ct_f_1 = ct_f_1 + 1
                        f_1 = f_1 + [check]

        elif(abs(ty-ya)<=shf):

            if((outerlayer == 0) and (lumped_inner == 0)):
                if( ((tz>ck_z_1) and (tz<ck_z_2) and (abs(tx-xab)<=shf) and
not (abs(tz-ck_z_1)<=shf) and not(abs(tz-ck_z_2)<=shf))
                 or ((tz<ck_z_4) and (tz>ck_z_3) and (abs(tx-xab)<=shf) and
not (abs(tz-ck_z_3)<=shf) and not(abs(tz-ck_z_4)<=shf))):
                    if(frontm != 1):
                        ct_f_2c = ct_f_2c + 1
                        f_2c = f_2c + [check]
                else:
                    if(frontm != 1):
                        ct_f_2 = ct_f_2 + 1
                        f_2 = f_2 + [check]

            elif((outerlayer == 0) and (lumped_inner == 1)):
                if( (tz>ck_z_1) and (tz<ck_z_4) and (abs(tx-xab)<=shf) and
not((abs(tz-ck_z_1)<=shf) or(abs(tz-ck_z_4)<=shf)) ):
                    if(frontm != 1):
                        ct_f_2c = ct_f_2c + 1
                        f_2c = f_2c + [check]
                else:
                    if(frontm != 1):
                        ct_f_2 = ct_f_2 + 1
                        f_2 = f_2 + [check]

            elif((outerlayer == 1) and (lumped_inner == 0)):
                if( ( (tz>ck_z_1) and (tz<ck_z_2) and (abs(tx-xab)<=shf) and
not(abs(tz-ck_z_2)<=shf) )
                 or ( (tz<ck_z_4) and (tz>ck_z_3) and (abs(tx-xab)<=shf) and
not(abs(tz-ck_z_3)<=shf) ) ):
                    if(frontm != 1):
                        ct_f_2c = ct_f_2c + 1
                        f_2c = f_2c + [check]
```

```
            else:
                if(frontm != 1):
                    ct_f_2 = ct_f_2 + 1
                    f_2 = f_2 + [check]

    elif(abs(tx-xa)<=shf):
        if(frontn != 1):
            ct_f_3 = ct_f_3 + 1
            f_3 = f_3 + [check]

    elif(abs(tx-xb)<=shf):
         if(fronto != 1):
            ct_f_4 = ct_f_4 + 1
            f_4 = f_4 + [check]

    elif(abs(tz-zb)<=shf):
        if(outerlayer != 1):
            if(front_5 != 1):
                ct_f_5 = ct_f_5 + 1
                f_5 = f_5 + [check]
        else:
            if(front_5 != 1):
                if(abs(tx-xab)<=shf):
                    ct_f_5c = ct_f_5c +1
                    f_5c = f_5c + [check]
                else:
                    ct_f_5 = ct_f_5 + 1
                    f_5 = f_5 + [check]

    elif(abs(tz-za) <= shf):
        if(outerlayer != 1):
            if(front_6 != 1):
                ct_f_6 = ct_f_6 + 1
                f_6 = f_6 + [check]
        else:
            if(front_6 != 1):
                if(abs(tx-xab)<=shf):
                    ct_f_6c = ct_f_6c + 1
                    f_6c = f_6c + [check]
                else:
                    ct_f_6 = ct_f_6 + 1
                    f_6 = f_6 + [check]

    j = j + 1

del f_1[f_1.index(0)]
del f_2[f_2.index(0)]
del f_3[f_3.index(0)]
del f_4[f_4.index(0)]
del f_5[f_5.index(0)]
del f_6[f_6.index(0)]

del f_1c[f_1c.index(0)]
del f_2c[f_2c.index(0)]

if(outerlayer == 1):
    del f_5c[f_5c.index(0)]
```

```
        del f_6c[f_6c.index(0)]

    print 'Face nodes are identified'
##-------------------------------------------------------------------------
## this routine is for identifying the node pairs in all the combinations of
edge pairs,
## face pairs, corner node pairs, face center pairs.
##-------------------------------------------------------------------------
    global ct_f_12, ct_f_34, fp_12, fp_34, ct_f_56, fp_56

    j, k = 0, 0
    fp_12, fp_34, fp_56 = [0], [0], [0]
    ct_f_12, ct_f_34, ct_f_56  = 0, 0, 0

    while j in range(len(f_1)):
        if(ct_f_1 != ct_f_2):
            print 'Face pairs F1 and F2 are not matched'
        check1 = f_1[j]
        tx1, ty1, tz1 = x[check1 - 1],y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_2)):
            check2 = f_2[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tx1 - tx2) <= shf) and (abs(tz1 - tz2) <= shf) and
(abs(ty1 + ty2) <= shf)):
                ct_f_12 = ct_f_12 + 2
                fp_12 = fp_12 + [check1]
                fp_12 = fp_12 + [check2]
            k = k + 1
        j = j + 1
    del fp_12[fp_12.index(0)]
    if((ct_f_12/ct_f_1 != 2) and (ct_f_12/ct_f_2 != 2)):
        print '  Error in identifying face pairs -  Face 1 and Face 2'
    else:
        print '  Face pairs F1 and F2 are identified'
##-----------------------------------------------------
    j, k = 0,0
    while j in range(len(f_3)):
        if(ct_f_3 != ct_f_4):
            print 'Face pairs F1 and F2 are not matched'
        check1 = f_3[j]
        tx1, ty1, tz1 = x[check1 - 1],y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_4)):
            check2 = f_4[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tx1 + tx2) <= shf) and (abs(tz1 - tz2) <= shf) and
(abs(ty1 - ty2) <= shf)):
                ct_f_34 = ct_f_34 + 2
                fp_34 = fp_34 + [check1]
                fp_34 = fp_34 + [check2]
            k = k + 1
        j = j + 1
    del fp_34[fp_34.index(0)]
    if((ct_f_34/ct_f_3 != 2) and (ct_f_34/ct_f_4 != 2)):
        print '  Error in identifying face pairs -  Face 3 and Face 4'
    else:
```

```
        print '  Face pairs F3 and F4 are identified'
##-------------------------------------------------
    j, k = 0,0
    while j in range(len(f_5)):
        if(ct_f_5 != ct_f_6):
            print 'Face pairs F5 and F6 are not matched'
        check1 = f_5[j]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        k = 0
        while k in range(len(f_6)):
            check2 = f_6[k]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if( (abs(tx1 - tx2) <= shf)  and (abs(ty1 - ty2) <= shf) ):
                ct_f_56 = ct_f_56 + 2
                fp_56 = fp_56 + [check1]
                fp_56 = fp_56 + [check2]
            k = k + 1
        j = j + 1
    del fp_56[fp_56.index(0)]
    if((ct_f_56/ct_f_5 != 2) and (ct_f_56/ct_f_6 != 2)):
        print '  Error in identifying face pairs -  Face 5 and Face 6'
    else:
        print '  Face pairs F5 and F6 are identified'
##-----------------------------------------------------------------------------
##-------------------------------------------------------
    global e_1c_el, e_1c_n, e_2c_el, e_2c_n
    e_1c_el, e_1c_n, e_2c_el, e_2c_n = [0], [0], [0], [0]

    k, m  = 0, 0
    e_1c_n, e_1c_el = [0], [0]
    while k in range(len(f_1c)):
        check = f_1c[k]
        m  = 0
        while m in range(len(node1)):
            check1 = node1[m]
            check2 = node2[m]
            check3 = node3[m]
            check4 = node4[m]
            check5 = node5[m]
            check6 = node6[m]
            check7 = node7[m]
            check8 = node8[m]
            if check in e_1c_n:
                pass
            else:
                if((check == check1) or (check == check2) or (check ==
check3) or (check == check4)
                   or (check == check5) or (check == check6) or (check ==
check7) or (check == check8)):
                    check9 = num_el[m]
                    e_1c_el = e_1c_el + [check9]
                    e_1c_n = e_1c_n + [check]
            m = m + 1
        k = k + 1
    del e_1c_el[e_1c_el.index(0)]
    del e_1c_n[e_1c_n.index(0)]
##-----------------------------------------------------------------------------
```

```
##-------------------------------------------------------------------------
global e_1c_BOBp, e_1c_BBp
##    from math import acosFace nodes are identified
    pi = acos(-1.0)
    k, m = 0, 0
    e_1c_BOBp, e_1c_BBp = [0], [0]
    while k in range(len(e_1c_el)):
        check = e_1c_el[k]
        check2 = e_1c_n[k]
        t1, t2, t3, t4 = node1[check - 1], node2[check - 1], node3[check -
1], node4[check - 1]
        t5, t6, t7, t8 = node5[check - 1], node6[check - 1], node7[check -
1], node8[check - 1]
        tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1], x[t4 - 1]
        tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1], x[t8 - 1]
        ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1], y[t4 - 1]
        ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1], y[t8 - 1]
        tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7 + tx8)
        tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7 + ty8)
        if(tempx > 0.0 ):
            e_1c_BBp = e_1c_BBp + [check2]
        elif(tempx < 0.0):
            e_1c_BOBp  = e_1c_BOBp  + [check2]
        k = k + 1

    del e_1c_BOBp[e_1c_BOBp.index(0)]
    del e_1c_BBp[e_1c_BBp.index(0)]

##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
    k, m  = 0, 0
    while k in range(len(f_2c)):
        check = f_2c[k]
        m  = 0
        while m in range(len(node1)):
            check1 = node1[m]
            check2 = node2[m]
            check3 = node3[m]
            check4 = node4[m]
            check5 = node5[m]
            check6 = node6[m]
            check7 = node7[m]
            check8 = node8[m]
            if check in e_2c_n:
                pass
            else:
                if((check == check1) or (check == check2) or (check ==
check3) or (check == check4)
                   or (check == check5) or (check == check6) or (check ==
check7) or (check == check8)):
                    check9 = num_el[m]
                    e_2c_el = e_2c_el + [check9]
                    e_2c_n = e_2c_n + [check]
            m = m + 1
        k = k + 1
    del e_2c_el[e_2c_el.index(0)]
    del e_2c_n[e_2c_n.index(0)]
```

```
##-------------------------------------------------------------------------
    k, m = 0, 0
    global e_2c_DODp, e_2c_DDp
    e_2c_DODp, e_2c_DDp = [0], [0]
    while k in range(len(e_2c_el)):
        check = e_2c_el[k]
        check2 = e_2c_n[k]
        t1, t2, t3, t4 = node1[check - 1], node2[check - 1], node3[check -
1], node4[check - 1]
        t5, t6, t7, t8 = node5[check - 1], node6[check - 1], node7[check -
1], node8[check - 1]
        tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1], x[t4 - 1]
        tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1], x[t8 - 1]
        ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1], y[t4 - 1]
        ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1], y[t8 - 1]
        tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7 + tx8)
        tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7 + ty8)
        tempxy = atan(tempy/tempx) * 180 / pi
        if(tempx < 0.0 ):
            e_2c_DODp = e_2c_DODp + [check2]
        elif(tempx > 0.0 ):
            e_2c_DDp  = e_2c_DDp  + [check2]

        k = k + 1
    del e_2c_DODp[e_2c_DODp.index(0)]
    del e_2c_DDp[e_2c_DDp.index(0)]
##-------------------------------------------------------------------------
##
    global f_12c_BDp, ct_f_12c
    f_12c_BDp, ct_f_12c = [0], 0
    k,m = 0,0
    while k in range(len(e_1c_BBp)):
        check = e_1c_BBp[k]
        tx1, ty1, tz1 = x[check-1], y[check-1], z[check-1]
        m = 0
        while m in range(len(e_2c_DDp)):
            check2 = e_2c_DDp[m]
            tx2, ty2, tz2 = x[check2-1], y[check2-1], z[check2-1]
            if(abs(tz1-tz2)<=shf):
                ct_f_12c = ct_f_12c + 2
                f_12c_BDp = f_12c_BDp + [check]
                f_12c_BDp = f_12c_BDp + [check2]
            m = m + 1
        k = k + 1
    del f_12c_BDp[f_12c_BDp.index(0)]
    if((ct_f_12c/len(e_2c_DDp) != 2) and (ct_f_12c/len(e_1c_BBp) != 2)):
        print '  Error in identifying face pairs -  1 C and 2 C '
    else:
        print '  Face pairs 1 C and 2 C BD are identified '
##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
##
    global f_12c_BODp, ct_f_12cd
    f_12c_BODp, ct_f_12cd = [0], 0
    k,m = 0,0
    while k in range(len(e_1c_BOBp)):
        check = e_1c_BOBp[k]
```

```
        tx1, ty1, tz1 = x[check-1], y[check-1], z[check-1]
        m = 0
        while m in range(len(e_2c_DODp)):
            check2 = e_2c_DODp[m]
            tx2, ty2, tz2 = x[check2-1], y[check2-1], z[check2-1]
            if(abs(tz1-tz2)<=shf):
                ct_f_12cd = ct_f_12cd + 2
                f_12c_BODp = f_12c_BODp + [check]
                f_12c_BODp = f_12c_BODp + [check2]
            m = m + 1
        k = k + 1
    del f_12c_BODp[f_12c_BODp.index(0)]
    if((ct_f_12cd/len(e_2c_DODp) != 2) and (ct_f_12cd/len(e_1c_BOBp) != 2)):
        print '  Error in identifying face pairs -  1 C and 2 C '
    else:
        print '  Face pairs 1 C and 2 C BOD are identified '
##-------------------------------------------------------------------------
    if (outerlayer == 1):
## the faces 5 and 6 have nodes on the crack.  Therefore have to be divided.
        global e_5c_el, e_5c_n, e_6c_el, e_6c_n
        e_5c_el, e_5c_n, e_6c_el, e_6c_n = [0], [0], [0], [0]

        k, m  = 0, 0
        e_5c_n, e_5c_el = [0], [0]
        while k in range(len(f_5c)):
            check = f_5c[k]
            m  = 0
            while m in range(len(node1)):
                check1 = node1[m]
                check2 = node2[m]
                check3 = node3[m]
                check4 = node4[m]
                check5 = node5[m]
                check6 = node6[m]
                check7 = node7[m]
                check8 = node8[m]
                if check in e_5c_n:
                    pass
                else:
                    if((check == check1) or (check == check2) or (check ==
check3) or (check == check4)
                       or (check == check5) or (check == check6) or (check ==
check7) or (check == check8)):
                        check9 = num_el[m]
                        e_5c_el = e_5c_el + [check9]
                        e_5c_n = e_5c_n + [check]
                m = m + 1
            k = k + 1
        del e_5c_el[e_5c_el.index(0)]
        del e_5c_n[e_5c_n.index(0)]
##-------------------------------------------------------------------------
        global e_5c_BOBp, e_5c_BBp
        k, m = 0, 0
        e_5c_BOBp, e_5c_BBp = [0], [0]
        while k in range(len(e_5c_el)):
            check = e_5c_el[k]
            check2 = e_5c_n[k]
```

```
            t1, t2, t3, t4 = node1[check - 1], node2[check - 1], node3[check
- 1], node4[check - 1]
            t5, t6, t7, t8 = node5[check - 1], node6[check - 1], node7[check
- 1], node8[check - 1]
            tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1], x[t4 - 1]
            tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1], x[t8 - 1]
            ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1], y[t4 - 1]
            ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1], y[t8 - 1]
            tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7 + tx8)
            tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7 + ty8)
            if(tempx > xab ):
                e_5c_BBp = e_5c_BBp + [check2]
            elif(tempx < xab):
                e_5c_BOBp  = e_5c_BOBp  + [check2]
            k = k + 1

        del e_5c_BOBp[e_5c_BOBp.index(0)]
        del e_5c_BBp[e_5c_BBp.index(0)]


##-------------------------------------------------------------------------
        k, m  = 0, 0
        e_6c_n, e_6c_el = [0], [0]
        while k in range(len(f_6c)):
            check = f_6c[k]
            m  = 0
            while m in range(len(node1)):
                check1 = node1[m]
                check2 = node2[m]
                check3 = node3[m]
                check4 = node4[m]
                check5 = node5[m]
                check6 = node6[m]
                check7 = node7[m]
                check8 = node8[m]
                if check in e_6c_n:
                    pass
                else:
                    if((check == check1) or (check == check2) or (check ==
check3) or (check == check4)
                       or (check == check5) or (check == check6) or (check ==
check7) or (check == check8)):
                        check9 = num_el[m]
                        e_6c_el = e_6c_el + [check9]
                        e_6c_n = e_6c_n + [check]
                m = m + 1
            k = k + 1
        del e_6c_el[e_6c_el.index(0)]
        del e_6c_n[e_6c_n.index(0)]
##-------------------------------------------------------------------------
        k, m = 0, 0
        global e_6c_DODp, e_6c_DDp
        e_6c_DODp, e_6c_DDp = [0], [0]
        while k in range(len(e_6c_el)):
            check = e_6c_el[k]
            check2 = e_6c_n[k]
```

```
            t1, t2, t3, t4 = node1[check - 1], node2[check - 1], node3[check
- 1], node4[check - 1]
            t5, t6, t7, t8 = node5[check - 1], node6[check - 1], node7[check
- 1], node8[check - 1]
            tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1], x[t4 - 1]
            tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1], x[t8 - 1]
            ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1], y[t4 - 1]
            ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1], y[t8 - 1]
            tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7 + tx8)
            tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7 + ty8)
            tempxy = atan(tempy/tempx) * 180 / pi
            if(tempx < 0.0 ):
                e_6c_DODp = e_6c_DODp + [check2]
            elif(tempx > 0.0 ):
                e_6c_DDp  = e_6c_DDp  + [check2]

            k = k + 1
        del e_6c_DODp[e_6c_DODp.index(0)]
        del e_6c_DDp[e_6c_DDp.index(0)]
##-------------------------------------------------------------------------
        global f_56c_BDp, ct_f_56c
        f_56c_BDp, ct_f_56c = [0], 0
        k,m = 0,0
        while k in range(len(e_5c_BBp)):
            check = e_5c_BBp[k]
            tx1, ty1, tz1 = x[check-1], y[check-1], z[check-1]
            m = 0
            while m in range(len(e_6c_DDp)):
                check2 = e_6c_DDp[m]
                tx2, ty2, tz2 = x[check2-1], y[check2-1], z[check2-1]
                if(abs(ty1-ty2)<=shf):
                    ct_f_56c = ct_f_56c + 2
                    f_56c_BDp = f_56c_BDp + [check]
                    f_56c_BDp = f_56c_BDp + [check2]
                m = m + 1
            k = k + 1
        del f_56c_BDp[f_56c_BDp.index(0)]
        if((ct_f_56c/len(e_6c_DDp) != 2) and (ct_f_56c/len(e_5c_BBp) != 2)):
            print '  Error in identifying face pairs -  5 C and 6 C BD '
        else:
            print '  Face pairs 5 C and 6 C BD are identified '
##-------------------------------------------------------------------------
        global f_56c_BODp, ct_f_56cd
        f_56c_BODp, ct_f_56cd = [0], 0
        k,m = 0,0
        while k in range(len(e_5c_BOBp)):
            check = e_5c_BOBp[k]
            tx1, ty1, tz1 = x[check-1], y[check-1], z[check-1]
            m = 0
            while m in range(len(e_6c_DODp)):
                check2 = e_6c_DODp[m]
                tx2, ty2, tz2 = x[check2-1], y[check2-1], z[check2-1]
                if(abs(ty1-ty2)<=shf):
                    ct_f_56cd = ct_f_56cd + 2
                    f_56c_BODp = f_56c_BODp + [check]
                    f_56c_BODp = f_56c_BODp + [check2]
                m = m + 1
```

```
            k = k + 1
        del f_56c_BODp[f_56c_BODp.index(0)]
        if((ct_f_56cd/len(e_6c_DODp) != 2) and (ct_f_56cd/len(e_5c_BOBp) !=
2)):
            print '  Error in identifying face pairs -  5 C and 6 C BOD '
        else:
            print '  Face pairs 5 C and 6 C BOD are identified '
##-----------------------------------------------------------------------------
    global e_25_AOAp, e_26_AOAp, e_25_AAp, e_16_AAp
    global e_15_AOAp, e_15_AAp, e_16_AOAp,  e_16_AAp
    global e_1626, e_1526, e_2526

    if(outerlayer == 1):
        trip_e_15c_11, trip_e_15c_22 = 0, 0
        trip_e_16c_11, trip_e_16c_22 = 0, 0
        trip_e_25c_11, trip_e_25c_22 = 0, 0
        trip_e_26c_11, trip_e_26c_22 = 0, 0

        m = 0
        while m in range(len(num_el)):
            check1 = node1[m]
            check2 = node2[m]
            check3 = node3[m]
            check4 = node4[m]
            check5 = node5[m]
            check6 = node6[m]
            check7 = node7[m]
            check8 = node8[m]
            chec = [check1, check2, check3, check4, check5, check6, check7,
check8]

            if(trip_e_15c_11==0):
                if e_15c_11 in chec:
                    trip_e_15c_11 = 1
                    c_146_el = num_el[m]
                    t1, t2, t3, t4 = node1[c_146_el - 1], node2[c_146_el -
1], node3[c_146_el - 1], node4[c_146_el - 1]
                    t5, t6, t7, t8 = node5[c_146_el - 1], node6[c_146_el -
1], node7[c_146_el - 1], node8[c_146_el - 1]
                    tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1],
x[t4 - 1]
                    tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1],
x[t8 - 1]
                    ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1],
y[t4 - 1]
                    ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1],
y[t8 - 1]
                    tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7
+ tx8)
                    tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7
+ ty8)
                    if(tempx < 0.0):
                        e_15_AOAp = e_15c_11
                        e_15_AAp = e_15c_22
                    else:
                        e_15_AOAp = e_15c_22
                        e_15_AAp = e_15c_11
```

```
            if(trip_e_16c_11==0):
                if e_16c_11 in chec:
                    trip_e_16c_11 = 1
                    c_146_el = num_el[m]
                    t1, t2, t3, t4 = node1[c_146_el - 1], node2[c_146_el -
1], node3[c_146_el - 1], node4[c_146_el - 1]
                    t5, t6, t7, t8 = node5[c_146_el - 1], node6[c_146_el -
1], node7[c_146_el - 1], node8[c_146_el - 1]
                    tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1],
x[t4 - 1]
                    tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1],
x[t8 - 1]
                    ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1],
y[t4 - 1]
                    ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1],
y[t8 - 1]
                    tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7
+ tx8)
                    tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7
+ ty8)
                    if(tempx < 0.0):
                        e_16_AOAp = e_16c_11
                        e_16_AAp = e_16c_22
                    else:
                        e_16_AOAp = e_16c_22
                        e_16_AAp = e_16c_11

            if(trip_e_25c_11==0):
                if e_25c_11 in chec:
                    trip_e_25c_11 = 1
                    c_146_el = num_el[m]
                    t1, t2, t3, t4 = node1[c_146_el - 1], node2[c_146_el -
1], node3[c_146_el - 1], node4[c_146_el - 1]
                    t5, t6, t7, t8 = node5[c_146_el - 1], node6[c_146_el -
1], node7[c_146_el - 1], node8[c_146_el - 1]
                    tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1],
x[t4 - 1]
                    tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1],
x[t8 - 1]
                    ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1],
y[t4 - 1]
                    ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1],
y[t8 - 1]
                    tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7
+ tx8)
                    tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7
+ ty8)
                    if(tempx < 0.0):
                        e_25_AOAp = e_25c_11
                        e_25_AAp = e_25c_22
                    else:
                        e_25_AOAp = e_25c_22
                        e_25_AAp = e_25c_11

            if(trip_e_26c_11==0):
                if e_26c_11 in chec:
```

```
                    trip_e_26c_11 = 1
                    c_146_el = num_el[m]
                    t1, t2, t3, t4 = node1[c_146_el - 1], node2[c_146_el -
1], node3[c_146_el - 1], node4[c_146_el - 1]
                    t5, t6, t7, t8 = node5[c_146_el - 1], node6[c_146_el -
1], node7[c_146_el - 1], node8[c_146_el - 1]
                    tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1],
x[t4 - 1]
                    tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1],
x[t8 - 1]
                    ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1],
y[t4 - 1]
                    ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1],
y[t8 - 1]
                    tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7
+ tx8)
                    tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7
+ ty8)
                    if(tempx < 0.0):
                        e_26_AOAp = e_26c_11
                        e_26_AAp = e_26c_22
                    else:
                        e_26_AOAp = e_26c_22
                        e_26_AAp = e_26c_11

            m = m + 1

## the crack nodes on edges are added so that no additional equations are
required.
        e_2526 = e_2526 + [e_25_AOAp, e_26_AOAp, e_25_AAp, e_26_AAp]
        e_1526 = e_1526 + [e_15_AOAp, e_26_AOAp, e_15_AAp, e_26_AAp]
        e_1626 = e_1626 + [e_16_AOAp, e_26_AOAp, e_16_AAp, e_26_AAp]
##-------------------------------------------------------------------------
def face_constraints():
    k,m = 0,0
    while k in range(len(fp_12)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 1 and 2 are written'
##-----------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_34)):
```

```
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 3 and 4 are written'
##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_12c_BDp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BDp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BDp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 1  BD and 2 are written'

##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_12c_BODp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
```

```
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 1 BOD and 2 are written'
##-----------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_56)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_56[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_56[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_6)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 5 and 6 are written'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def corner_constraints():
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(3) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c146)+',' +  '+ str(m+1)+','+  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c246)+',' +  '+ str(m+1)+','+  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(3) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c236)+',' +  '+ str(m+1)+','+  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c246)+',' +  '+ str(m+1)+','+  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
```

```
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(3) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c245)+',' +  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_6)+',' +  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)

    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(4) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c145)+',' +  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_6)+',' +  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(4) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c235)+',' +  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_6)+',' +  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(5) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c135)+',' +  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+'  '+str(2.0) + '\n'
```

```
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_6)+',' +  '+ str(m+1)+','+  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(4) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c136)+',' +  '+ str(m+1)+','+  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c246)+',' +  '+ str(m+1)+','+  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(2.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1


    print '  Corner Constraints are written - No outer layers'
#-------------------------------------------------------------------------------
##-------------------------------------------------------------------------------
def edge_constraints():
    k,m = 0,0
    while k in range(len(e_1626)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1626[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1626[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 16 and 26'

    k,m = 0,0
    while k in range(len(e_1526)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
```

```
            strgg = ' ' + str(e_1526[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1526[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_6)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 15 and 26'

    k,m = 0,0
    while k in range(len(e_2526)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2526[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2526[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_6)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 25 and 26'

    k,m = 0,0
    while k in range(len(e_3546)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3546[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3546[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_6)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
```

```
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 35 and 46'

    k,m = 0,0
    while k in range(len(e_4546)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_4546[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_4546[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 45 and 46'

    k,m = 0,0
    while k in range(len(e_3646)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 36 and 46'

    k,m = 0,0
    while k in range(len(e_1424)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1424[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
```

```
            strgg = ' ' + str(e_1424[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 14 and 24'

    k,m = 0,0
    while k in range(len(e_2324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 23 and 24'

    k,m = 0,0
    while k in range(len(e_1324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1324[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1324[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 13 and 24'

##-----------------------------------------------------------------------
##-----------------------------------------------------------------------
```

```
def center_constraints():
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    print '  Face Center Constraints are written'
##-----------------------------------------------------------------------------
## CODE STARTS FOR LUMPED INNER LAYERS WITH ZERO ORIENTATION
##-----------------------------------------------------------------------------
def corner_constraints_lump():
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(4) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c146)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1
```

```
m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(3) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c236)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(5) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c136)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(3) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c245)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(5) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c145)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
```

```
        strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(4) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c235)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(6) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    print '  Lumped - Corner Constraints are written'
##-------------------------------------------------------------------------
##-------------------------------------------------------------------------
def edge_constraints_lump():
    k,m = 0,0
    while k in range(len(e_1626)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
```

```
            strgg = ' ' + str(e_1626[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1626[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +  '+ str(m+1)+','+  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +  '+ str(m+1)+','+  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 16 and 26'

    k,m = 0,0
    while k in range(len(e_1526)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(5) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1526[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1526[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +  '+ str(m+1)+','+  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +  '+ str(m+1)+','+  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_6)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 15 and 26'

    k,m = 0,0
    while k in range(len(e_2526)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2526[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2526[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
```

```
            strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 25 and 26'

    k,m = 0,0
    while k in range(len(e_3546)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3546[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3546[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 35 and 46'

    k,m = 0,0
    while k in range(len(e_4546)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_4546[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_4546[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 45 and 46'

    k,m = 0,0
    while k in range(len(e_3646)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
```

```
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 36 and 46'

    k,m = 0,0
    while k in range(len(e_1424)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1424[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1424[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +  '+ str(m+1)+','+  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +  '+ str(m+1)+','+  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 14 and 24'

    k,m = 0,0
    while k in range(len(e_2324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
```

```
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 23 and 24'

    k,m = 0,0
    while k in range(len(e_1324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(5) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1324[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1324[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Edge constraints are written for 13 and 24'
##---------------------------------------------------------------------------
##---------------------------------------------------------------------------
def center_constraints_lump():
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_11)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1
```

```
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_12)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_5)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    print '  Lumped - Face Center Constraints are written'
##-------------------------------------------------------------------------
def face_constraints_lump():
    k,m = 0,0
    while k in range(len(fp_12)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Face Constraints for faces 1 and 2 are written'
##-------------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_34)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
```

```
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Face Constraints for faces 3 and 4 are written'
##_____________________________________________________________________
k, m = 0, 0
    while k in range(len(f_12c_BDp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BDp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BDp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Face Constraints for faces 3 and 4 are written'

##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_12c_BODp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_21)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
```

```
            strgg = ' ' + str(c_22)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Lumped - Face Constraints for faces 3 and 4 are written'
##-----------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_56)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_56[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_56[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_6)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 5 and 6 are written'
##-----------------------------------------------------------------------
## constraint equations for outerlayer 90 degree failure.
## CODE STARTS FOR OUTER LAYER.
def face_constraints_outer():
    k,m = 0,0
    while k in range(len(fp_12)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_12[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 1 and 2 are written'
##-----------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_34)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
```

```
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_34[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 3 and 4 are written'
##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_12c_BDp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BDp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BDp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 1  BD and 2 are written'

##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_12c_BODp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_12c_BODp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
```

```
    print '  Face Constraints for faces 1 BOD and 2 are written'
##-----------------------------------------------------
    k,m = 0,0
    while k in range(len(fp_56)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_56[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(fp_56[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 5 and 6 are written'
##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_56c_BDp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_56c_BDp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_56c_BDp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 5 C BD and 6 C are written'
##_____________________________________________________________________
    k, m = 0, 0
    while k in range(len(f_56c_BODp)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
```

```
            output_1.write(strgg)
            strgg = ' ' + str(f_56c_BODp[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(f_56c_BODp[k+1])+',' +'  '+ str(m+1)+','+'
'+str(-1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Face Constraints for faces 5 C BOD and 6 C are written'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def center_constraints_outer():
    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_1)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_3)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_51)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    m = 0
    while m in range(3):
```

```
        strgg = '*Equation\n'
        output_1.write(strgg)
        strgg = str(2) + '\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_52)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
        output_1.write(strgg)
        strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
        output_1.write(strgg)
        m = m + 1

    print '  Face Center Constraints are written'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def edge_constraints_outer():
    k,m = 0,0
    while k in range(len(e_1626)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1626[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1626[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 16 and 26'

    k,m = 0,0
    while k in range(len(e_1526)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(5) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1526[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1526[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
```

```
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 15 and 26'

    k,m = 0,0
    while k in range(len(e_2526)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2526[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2526[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 25 and 26'

    k,m = 0,0
    while k in range(len(e_3546)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(5) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3546[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3546[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 35 and 46'

    k,m = 0,0
```

```
    while k in range(len(e_4546)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_4546[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_4546[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 45 and 46'

    k,m = 0,0
    while k in range(len(e_3646)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_3646[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 36 and 46'

    k,m = 0,0
    while k in range(len(e_1424)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1424[k])+',' +'  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1424[k+1])+',' +'  '+ str(m+1)+','+'  '+str(-
1.0) + '\n'
```

```
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 14 and 24'

    k,m = 0,0
    while k in range(len(e_2324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(3) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_2324[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 23 and 24'

    k,m = 0,0
    while k in range(len(e_1324)):
        m = 0
        while m in range(3):
            strgg = '*Equation\n'
            output_1.write(strgg)
            strgg = str(4) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1324[k])+',' +  '+ str(m+1)+','+'
'+str(1.0)+'\n'
            output_1.write(strgg)
            strgg = ' ' + str(e_1324[k+1])+',' +  '+ str(m+1)+','+  '+str(-
1.0) + '\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_2)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            strgg = ' ' + str(c_4)+',' +  '+ str(m+1)+','+  '+str(2.0) +
'\n'
            output_1.write(strgg)
            m = m + 1
        k = k + 2
    print '  Edge constraints are written for 13 and 24'

##-----------------------------------------------------------------------
##-----------------------------------------------------------------------
def corner_constraints_outer():
    m = 0
```

```
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(3) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c146)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(3) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c236)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(4) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c245)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(5) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c145)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
```

```
    strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(5) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c235)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(6) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c135)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_61)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_62)+',' +'  '+ str(m+1)+','+'  '+str(1.0) + '\n'
    output_1.write(strgg)
    m = m + 1

m = 0
while m in range(3):
    strgg = '*Equation\n'
    output_1.write(strgg)
    strgg = str(4) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c136)+',' +'  '+ str(m+1)+','+'  '+str(1.0)+'\n'
    output_1.write(strgg)
    strgg = ' ' + str(c246)+',' +'  '+ str(m+1)+','+'  '+str(-1.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_2)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    strgg = ' ' + str(c_4)+',' +'  '+ str(m+1)+','+'  '+str(2.0) + '\n'
    output_1.write(strgg)
    m = m + 1
```

```
    print '  Corner Constraints are written - No outer layers'
#-----------------------------------------------------------------------------
def contact_surfaces():
    global crack_1A_elem, crack_1B_elem
    global crack_2A_elem, crack_2B_elem

    crack_1A_elem, crack_2A_elem = [], []
    crack_1B_elem, crack_2B_elem = [], []

    j = 0
    while j in range(len(num_el)):
        check = num_el[j]
        t1, t2, t3, t4 = node1[check - 1], node2[check - 1], node3[check -
1], node4[check - 1]
        t5, t6, t7, t8 = node5[check - 1], node6[check - 1], node7[check -
1], node8[check - 1]
        tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1], x[t4 - 1]
        tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1], x[t8 - 1]
        ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1], y[t4 - 1]
        ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1], y[t8 - 1]
        tz1, tz2, tz3, tz4 = z[t1 - 1], z[t2 - 1], z[t3 - 1], z[t4 - 1]
        tz5, tz6, tz7, tz8 = z[t5 - 1], z[t6 - 1], z[t7 - 1], z[t8 - 1]
        check1 = [tx1, tx2, tx3, tx4, tx5, tx6, tx7, tx8]
        tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7 + tx8)
        tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7 + ty8)
        tempz = 1/8.0 * (tz1 + tz2 + tz3 + tz4 + tz5 + tz6 + tz7 + tz8)
        ck_z_2a = ck_z_2 - crack_layer_thickness / seed_6
        ck_z_3a = ck_z_3 + crack_layer_thickness / seed_6
        ck_z_1a = ck_z_1 + crack_layer_thickness / seed_6
        ck_z_4a = ck_z_4 - crack_layer_thickness / seed_6
        ck_y_1a = 3.0*b/8.0
        if (tempx < xab) and (tempz > ck_z_1a) and (tempz < ck_z_2a) and
(abs(tempy)<ck_y_1a):
            if ck_z_2 in check1:
                pass
            else:
                if xab in check1:
                    crack_element_faces(cck2=check1)
                    crack_1A_elem = crack_1A_elem + [check, face_number]
        elif(tempx > xab) and (tempz > ck_z_1a) and (tempz < ck_z_2a) and
(abs(tempy)<ck_y_1a):
            if ck_z_2 in check1:
                pass
            else:
                if xab in check1:
                    crack_element_faces(cck2=check1)
                    crack_1B_elem = crack_1B_elem + [check, face_number]
        elif (tempx < xab) and (tempz > ck_z_3a) and (tempz < ck_z_4a) and
(abs(tempy)<ck_y_1a):
            if ck_z_3 in check1:
                pass
            else:
                if xab in check1:
                    crack_element_faces(cck2=check1)
                    crack_2A_elem = crack_2A_elem + [check, face_number]
```

```
        elif(tempx > xab) and (tempz > ck_z_3a) and (tempz < ck_z_4a) and
(abs(tempy)<ck_y_1a):
            if ck_z_3 in check1:
                pass
            else:
                if xab in check1:
                    crack_element_faces(cck2=check1)
                    crack_2B_elem = crack_2B_elem + [check, face_number]
        j = j + 1

    ## writing the element based surfaces.

    j = 0
    strgg = '*Surface, TYPE=ELEMENT, NAME=CRACK1A, internal\n'
    output_1.write(strgg)
    while j in range(len(crack_1A_elem)):
        strgg = str(crack_1A_elem[j])+',   '+crack_1A_elem[j+1]+'\n'
        output_1.write(strgg)
        j = j + 2

    j = 0
    strgg = '*Surface, TYPE=ELEMENT, NAME=CRACK1B, internal\n'
    output_1.write(strgg)
    while j in range(len(crack_1B_elem)):
        strgg = str(crack_1B_elem[j])+',   '+crack_1B_elem[j+1]+'\n'
        output_1.write(strgg)
        j = j + 2


    j = 0
    strgg = '*Surface, TYPE=ELEMENT, NAME=CRACK2A, internal\n'
    output_1.write(strgg)
    while j in range(len(crack_2A_elem)):
        strgg = str(crack_2A_elem[j])+',   '+crack_2A_elem[j+1]+'\n'
        output_1.write(strgg)
        j = j + 2

    j = 0
    strgg = '*Surface, TYPE=ELEMENT, NAME=CRACK2B, internal\n'
    output_1.write(strgg)
    while j in range(len(crack_2B_elem)):
        strgg = str(crack_2B_elem[j])+',   '+crack_2B_elem[j+1]+'\n'
        output_1.write(strgg)
        j = j + 2


def crack_contact():
    strgg = '*CONTACT PAIR, INTERACTION = CRACK1AB,  SMALL SLIDING,'
    strgg = strgg + 'TYPE = SURFACE TO SURFACE\n'
    output_1.write(strgg)
    strgg = 'sublami_CRACK1A,  sublami_CRACK1B\n'
    output_1.write(strgg)
    strgg = '*SURFACE INTERACTION, NAME=CRACK1AB\n'
    output_1.write(strgg)
##    strgg = '*SURFACE BEHAVIOR, PENALTY\n'
##    output_1.write(strgg)
    strgg = '*FRICTION\n'
```

```
    output_1.write(strgg)
    strgg = str(0.4)+'\n'
    output_1.write(strgg)
    strgg = '*CONTACT PAIR, INTERACTION = CRACK2AB,  SMALL SLIDING,'
    strgg = strgg + 'TYPE = SURFACE TO SURFACE\n'
    output_1.write(strgg)
    strgg = 'sublami_CRACK2B,  sublami_CRACK2A\n'
    output_1.write(strgg)
    strgg = '*SURFACE INTERACTION, NAME=CRACK2AB\n'
    output_1.write(strgg)
##    strgg = '*SURFACE BEHAVIOR, PENALTY\n'
##    output_1.write(strgg)
    strgg = '*FRICTION\n'
    output_1.write(strgg)
    strgg = str(0.4)+'\n'
    output_1.write(strgg)


##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def crack_element_faces(cck2):
    global face_number
    cck21, cck22, cck23, cck24 = cck2[0], cck2[1], cck2[2], cck2[3]
    cck25, cck26, cck27, cck28 = cck2[4], cck2[5], cck2[6], cck2[7]
    if (cck21 == xab) and (cck22 == xab) and (cck25 == xab) and (cck26 ==
xab):
        face_number = 'S3'
    elif (cck21 == xab) and (cck22 == xab) and (cck23 == xab) and (cck24 ==
xab):
        face_number = 'S1'
    elif (cck25 == xab) and (cck26 == xab) and (cck27 == xab) and (cck28 ==
xab):
        face_number = 'S2'
    elif (cck22 == xab) and (cck23 == xab) and (cck26 == xab) and (cck27 ==
xab):
        face_number = 'S4'
    elif (cck23 == xab) and (cck24 == xab) and (cck27 == xab) and (cck28 ==
xab):
        face_number = 'S5'
    elif (cck21 == xab) and (cck24 == xab) and (cck25 == xab) and (cck28 ==
xab):
        face_number = 'S6'
#-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def gap_element():
    global crack_1A_elem, crack_1B_elem
    global crack_2A_elem, crack_2B_elem
    global crack_1A_node, crack_1B_node
    global crack_2A_node, crack_2B_node
    global crack_1AB_nodes, crack_2AB_nodes

    crack_1A_elem, crack_2A_elem = [], []
    crack_1B_elem, crack_2B_elem = [], []
    crack_1A_node, crack_2A_node = [], []
    crack_1B_node, crack_2B_node = [], []
    crack_1AB_nodes, crack_2AB_nodes = [], []
    j = 0
```

```
    while j in range(len(num_el)):
        check = num_el[j]
        t1, t2, t3, t4 = node1[check - 1], node2[check - 1], node3[check -
1], node4[check - 1]
        t5, t6, t7, t8 = node5[check - 1], node6[check - 1], node7[check -
1], node8[check - 1]
        tx1, tx2, tx3, tx4 = x[t1 - 1], x[t2 - 1], x[t3 - 1], x[t4 - 1]
        tx5, tx6, tx7, tx8 = x[t5 - 1], x[t6 - 1], x[t7 - 1], x[t8 - 1]
        ty1, ty2, ty3, ty4 = y[t1 - 1], y[t2 - 1], y[t3 - 1], y[t4 - 1]
        ty5, ty6, ty7, ty8 = y[t5 - 1], y[t6 - 1], y[t7 - 1], y[t8 - 1]
        tz1, tz2, tz3, tz4 = z[t1 - 1], z[t2 - 1], z[t3 - 1], z[t4 - 1]
        tz5, tz6, tz7, tz8 = z[t5 - 1], z[t6 - 1], z[t7 - 1], z[t8 - 1]
        check1 = [tx1, tx2, tx3, tx4, tx5, tx6, tx7, tx8]
        ncheck = [t1, t2, t3, t4, t5, t6, t7, t8]
        tempx = 1/8.0 * (tx1 + tx2 + tx3 + tx4 + tx5 + tx6 + tx7 + tx8)
        tempy = 1/8.0 * (ty1 + ty2 + ty3 + ty4 + ty5 + ty6 + ty7 + ty8)
        tempz = 1/8.0 * (tz1 + tz2 + tz3 + tz4 + tz5 + tz6 + tz7 + tz8)
        ck_z_2a = ck_z_2 - crack_layer_thickness / seed_6
        ck_z_3a = ck_z_3 + crack_layer_thickness / seed_6
        ck_z_1a = ck_z_1 + crack_layer_thickness / seed_6
        ck_z_4a = ck_z_4 - crack_layer_thickness / seed_6
        ck_y_1a = 3.0*b/8.0
        if (tempx < xab) and (tempz > ck_z_1a) and (tempz < ck_z_2a) and
(abs(tempy)<ck_y_1a):

            if xab in check1:
                crack_1A_elem = crack_1A_elem + [check]
                m = 0
                for m in range(len(check1)):
                    if (abs(check1[m]-xab)<=shf) and (ncheck[m] not in
crack_1A_node):
                        crack_1A_node = crack_1A_node + [ncheck[m]]

        elif(tempx > xab) and (tempz > ck_z_1a) and (tempz < ck_z_2a) and
(abs(tempy)<ck_y_1a):

            if xab in check1:
                crack_1B_elem = crack_1B_elem + [check]
                m = 0
                for m in range(len(check1)):
                    if (abs(check1[m]-xab)<=shf) and (ncheck[m] not in
crack_1B_node):
                        crack_1B_node = crack_1B_node + [ncheck[m]]

        elif (tempx < xab) and (tempz > ck_z_3a) and (tempz < ck_z_4a) and
(abs(tempy)<ck_y_1a):
            if xab in check1:
                crack_2A_elem = crack_2A_elem + [check]
                m = 0
                for m in range(len(check1)):
                    if (abs(check1[m]-xab)<=shf) and (ncheck[m] not in
crack_2A_node):
                        crack_2A_node = crack_2A_node + [ncheck[m]]

        elif(tempx > xab) and (tempz > ck_z_3a) and (tempz < ck_z_4a) and
(abs(tempy)<ck_y_1a):
            if xab in check1:
```

```
                crack_2B_elem = crack_2B_elem + [check]
                m = 0
                for m in range(len(check1)):
                    if (abs(check1[m]-xab)<=shf) and (ncheck[m] not in
crack_2B_node):
                        crack_2B_node = crack_2B_node + [ncheck[m]]

        j = j + 1

    j = 0
    while j in range(len(crack_1A_node)):
        check1 = crack_1A_node[j]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(crack_1B_node)):
            check2 = crack_1B_node[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if (abs(ty1-ty2)<=shf) and (abs(tz1-tz2)<=shf):
##                x[check1 - 1]= x[check1 - 1] - a/(seed_1*512)
##                x[check2 - 1]= x[check2 - 1] + a/(seed_1*512)
                crack_1AB_nodes = crack_1AB_nodes + [check2, check1]
            m = m + 1
        j = j + 1

    j = 0
    while j in range(len(crack_2A_node)):
        check1 = crack_2A_node[j]
        tx1, ty1, tz1 = x[check1 - 1], y[check1 - 1], z[check1 - 1]
        m = 0
        while m in range(len(crack_2B_node)):
            check2 = crack_2B_node[m]
            tx2, ty2, tz2 = x[check2 - 1], y[check2 - 1], z[check2 - 1]
            if (abs(ty1-ty2)<=shf) and (abs(tz1-tz2)<=shf):
##                x[check1 - 1]= x[check1 - 1] - a/(seed_1*512)
##                x[check2 - 1]= x[check2 - 1] - a/(seed_1*512)
                crack_2AB_nodes = crack_2AB_nodes + [check2, check1]
            m = m + 1
        j = j + 1

def gap_elements_written():
    strgg = '** The GAP elements are written here\n'
    output_1.write(strgg)
    strgg = '*Element, type=GAPUNI\n'
    output_1.write(strgg)

    j = 0
    jj = 0
    while j in range(len(crack_1AB_nodes)/2):
        strgg = str(num_elem+j+1)+',   '+str(crack_1AB_nodes[jj])+ ',
'+str(crack_1AB_nodes[jj+1])+'\n'
        output_1.write(strgg)
        jj = jj + 2
        j = j + 1

    k = j

    strgg = '*Element, type=GAPUNI\n'
```

```
    output_1.write(strgg)
    j = 0
    jj = 0
    while j in range(len(crack_2AB_nodes)/2):
        strgg = str(num_elem+k+j+1)+',   '+str(crack_2AB_nodes[jj])+ ',
'+str(crack_2AB_nodes[jj+1])+'\n'
        output_1.write(strgg)
        jj = jj + 2
        j = j + 1

    kk = j

    strgg = '*Elset, elset=CRACK1AB, internal, generate\n'
    output_1.write(strgg)
    strgg = str(num_elem+1)+',   '+str(num_elem+k)+',   '+str(1)+'\n'
    output_1.write(strgg)
    strgg = '*Elset, elset=CRACK2AB, internal, generate\n'
    output_1.write(strgg)
    strgg = str(num_elem+k+1)+',   '+str(num_elem+2*k)+',   '+str(1)+'\n'
    output_1.write(strgg)

    print ' GAP ELEMENT PROPERTIES ARE WRITTEN'
    strgg = '*GAP, ELSET = CRACK1AB\n'
    strgg = strgg + '0.0, -1.0, 0.0, 0.0\n'
    output_1.write(strgg)
    strgg = '*GAP, ELSET = CRACK2AB\n'
    strgg = strgg + '0.0, -1.0, 0.0, 0.0\n'
    output_1.write(strgg)

#-------------------------------------------------------------------------------
##-------------------------------------------------------------------------------
def rigid_body_modes():
    strgg ='** Boundary conditions written\n'
    output_1.write(strgg)
    strgg='*Boundary\n'
    output_1.write(strgg)

    if(lumped_inner == 1):
##        strgg = ' n_set_6'+','+' '+str(1) + ','+ ' '+ str(3)+'\n'
##        output_1.write(strgg)
##        strgg = ' n_set_7'+','+' '+str(2) + ','+ ' '+ str(2)+'\n'
##        output_1.write(strgg)
##        strgg = ' n_set_8'+','+' '+str(3) + ','+ ' '+ str(3)+'\n'
##        output_1.write(strgg)

        strgg = ' n_set_4'+','+' '+str(1) + ','+ ' '+ str(3)+'\n'
        output_1.write(strgg)
        strgg = ' n_set_7'+','+' '+str(2) + ','+ ' '+ str(2)+'\n'
        output_1.write(strgg)
        strgg = ' n_set_6'+','+' '+str(2) + ','+ ' '+ str(3)+'\n'
        output_1.write(strgg)

    else:
        strgg = ' n_set_5'+','+' '+str(1) + ','+ ' '+ str(3)+'\n'
        output_1.write(strgg)
        strgg = ' n_set_2'+','+' '+str(3) + ','+ ' '+ str(3)+'\n'
        output_1.write(strgg)
```

```
        strgg = ' n_set_2'+','+' '+str(1) + ','+ ' '+ str(1)+'\n'
        output_1.write(strgg)

        if(outerlayer == 1):
            strgg = ' n_set_4'+','+' '+str(3) + ','+ ' '+ str(3)+'\n'
            output_1.write(strgg)
        else:
            strgg = ' n_set_6'+','+' '+str(1) + ','+ ' '+ str(1)+'\n'
            output_1.write(strgg)

    print 'Rigid body modes are written'
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
def loads():
    if((mechanical == 1) and (lumped_inner == 1)):
##        strgg= '*Cload \n'
##        output_1.write(strgg)
##        strgg = ' n_set_4'+','+' '+ str(3) + ','+ ' '+ str(-
loadcase_3_mag_1)+'\n'
##        output_1.write(strgg)

        strgg= '*Cload \n'
        output_1.write(strgg)
        strgg = ' n_set_6'+','+' '+ str(1) + ','+ ' '+ str(-
loadcase_3_mag_2)+'\n'
        output_1.write(strgg)

##        strgg= '*Boundary \n'
##        output_1.write(strgg)
##        strgg = ' n_set_4'+','+' '+ str(3) + ',,'+ ' '+ str(1e-5)+'\n'
##        output_1.write(strgg)
    elif((mechanical == 1) and (lumped_inner == 0) and (outerlayer == 0)):
        strgg= '*Cload \n'
        output_1.write(strgg)
        strgg = ' n_set_4'+','+' '+ str(3) + ','+ ' '+ str(-
loadcase_3_mag_1)+'\n'
        output_1.write(strgg)
        print 'Mechanical Loading only '

    elif((mechanical ==1) and (outerlayer ==1)):
        strgg= '*Cload \n'
        output_1.write(strgg)
        strgg = ' n_set_61'+','+' '+ str(1) + ','+ ' '+
str(loadcase_3_mag_2)+'\n'
        output_1.write(strgg)
        strgg = ' n_set_62'+','+' '+ str(1) + ','+ ' '+
str(loadcase_3_mag_2)+'\n'
        output_1.write(strgg)
        print 'Mechanical Loading only '

##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
from abaqus import *
from abaqusConstants import *
from section import*
import math
reading_inp_file()
```

APPENDIX C

PYTHON CODE FOR T-JOINT MODEL

In this appendix, a computer code written in *Python* language, for generating a composite T-joint is listed. The program is intended to run in ABAQUS Version 6.6. The entire script is run by executing the statement "**execfile('Joint_two_defects_composite_A.py')**" in an ABAQUS command window. The script file has all the required parameters to build the model. This includes material definition, laminate properties, and the joint dimensions. Note that this code does not completely generate the model as automating continuum shell generation is not available with ABAQUS Version 6.6. Also, the boundary conditions to eliminate rigid body modes need to be imposed after the model is generated.

Program: ***Joint_two_defects_composite_A.py***

```
## A JOINT THAT HAS DEFECTS ON THE BASE PLATE. OTHER DEFECTS
## CAN BE CREATED.

from abaqus import *
from abaqusConstants import *
import math
global ED, DT
##-------------------------------------------------------------
## CAUTION: There are a lot of input parameters involved.
##-------------------------------------------------------------
## information of the laminate that applies to the base plate,
## vertical plate and the clips.
## loading
theta_temp = -414
## theta_temp - temperature loading range.
layup_ge = [-45, 0, 45, 90, 90, 45, 0, -45]
##layup_ge = [0, 0, 0, 0, 0, 0, 0, 0]

temp = []
for ij in range(6):
    temp = temp + layup_ge
layup_ge= temp
nol = len(layup_ge)
laminate_thickness = 0.12
ply_t = laminate_thickness/nol
ply_alt = 0.004 ## thickness of the titanium ply.
ply_al = ply_alt + 0.01 ## thickness of the adhesive layer.

layup_thic = [ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t]
temp = []
for ij in range(6):
    temp = temp + layup_thic
layup_thic = temp

layup_material = [0, 0, 0, 0, 0, 0, 0, 0]
temp = []
for ij in range(6):
    temp = temp + layup_material
```

```
layup_material = temp

clip_material = [0, 0, 0, 0, 0, 0, 0, 0]
temp = []
for ij in range(6):
    temp = temp + clip_material
clip_material = temp
## nos = number of sections
nos = 9

## this is addded to reduce the thickness of the vertical plate.

layup_ge_v =    [-45, 0, 45, 90, 90, 45, 0, -45]
layup_ge_v_ch = [-45, 0, 45, 90, 90, 45, 0, -45]

temp = []
for ij in range(6):
    if(ij == 0):
        temp = temp + layup_ge_v_ch
    else:
        temp = temp + layup_ge_v
layup_ge_v = temp

layup_thic_v =    [ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t]
layup_thic_v_ch = [ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t]
temp = []
for ij in range(6):
    if(ij == 0):
        temp = temp + layup_thic_v_ch
    else:
        temp = temp + layup_thic_v
layup_thic_v = temp

layup_material_v =    [0, 0, 0, 0, 0, 0, 0, 0]
layup_material_v_ch = [0, 0, 0, 0, 0, 0, 0, 0]
temp = []
for ij in range(6):
    if(ij == 0):
        temp = temp + layup_material_v_ch
    else:
        temp = temp + layup_material_v
layup_material_v = temp



## this is addded to reduce the thickness of the BASE plate.

layup_ge_b =    [-45, 0, 45, 90, 90, 45, 0, -45]
layup_ge_b_ch = [-45, 0, 45, 90, 90, 45, 0, -45]

temp = []
for ij in range(6):
    if(ij == 6):
        temp = temp + layup_ge_b_ch
    else:
        temp = temp + layup_ge_b
layup_ge_b = temp
```

```
layup_thic_b =    [ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t]
layup_thic_b_ch = [ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t]
temp = []
for ij in range(6):
    if(ij == 6):
        temp = temp + layup_thic_b_ch
    else:
        temp = temp + layup_thic_b
layup_thic_b = temp

layup_material_b =    [0, 0, 0, 0, 0, 0, 0, 0]
layup_material_b_ch = [0, 0, 0, 0, 0, 0, 0, 0]
temp = []
for ij in range(6):
    if(ij == 6):
        temp = temp + layup_material_b_ch
    else:
        temp = temp + layup_material_b
layup_material_b = temp

## this is added to delete some of the layers from the composite in
## one of the clips. the layer removed is described at the end of this.
layup_ge_ch = [-45, 0, 45, 90, 90, 45, 0, -45]
len_layup_ch = len(layup_ge_ch)
layup_ge_2  = [-45, 0, 45, 90, 90, 45, 0, -45]
temp = []
for ij in range(6):
    if (ij == 6) :
        temp = temp + layup_ge_ch
    else:
        temp = temp + layup_ge_2
layup_ge_ch= temp
nol_ch = len(layup_ge_ch)
laminate_thickness_ch = 0.12 - 0.12/nol * 2 * len_layup_ch ## two layers
removed.
ply_t_ch = laminate_thickness_ch/nol_ch

layup_thic_ch = [ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t]
layup_thic_2 =  [ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t, ply_t]
temp = []
for ij in range(6):
    if (ij == 6):
        temp = temp + layup_thic_ch
    else:
        temp = temp + layup_thic_2
layup_thic_ch = temp

layup_material_ch = [0, 0, 0, 0, 0, 0, 0, 0]
layup_material_2 =  [0, 0, 0, 0, 0, 0, 0, 0]
temp = []
for ij in range(6):
    if (ij == 6):
        temp = temp + layup_material_ch
    else:
        temp = temp + layup_material_2
layup_material_ch = temp
```

```
## material database
mat_name = [0]

## NOTE:-- 1 psi = 6.895 k Pa
## material - 1
NAME = 'Graphite'
temp = [1, NAME]
E11, E22, E33 =  3E7, 1.5E6, 1.5E6
NU12, NU13, NU23 = 0.3, 0.3, 0.6
G12, G13, G23 =  4.E4, 4.E4, E22/(2*(1+NU23))
A11, A22, A33 = -2e-7, 1.5e-5, 1.5e-5
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## material - 2
NAME = 'Adhesive'
temp = [2, NAME]
E11, E22, E33 =  0.3E6, 0.3E6, 0.3E6
G12, G13, G23 =  0.1064E6, 0.1064E6, 0.1064E6
NU12, NU13, NU23 = 0.41, 0.41, 0.41
A11, A22, A33 = 42e-6, 42e-6, 42e-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

#### material - 3
NAME = 'Titaniumlayer'
temp = [3, NAME]
EE = 116E9/6895.0
GG = EE/(2*(1+0.36))
E11, E22, E33 =  EE, EE, EE
G12, G13, G23 =  GG, GG, GG
NU12, NU13, NU23 = 0.36, 0.36, 0.36
A11, A22, A33 = 4.72E-6, 4.72E-6, 4.72E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]
##
#### material - 4
NAME = 'Defect' ## for now we use Aluminum
temp = [3, NAME]
EE = 72.36737E9/6895.0
GG = EE/(2*(1+0.33))
E11, E22, E33 =  EE, EE, EE
G12, G13, G23 =  GG, GG, GG
NU12, NU13, NU23 = 0.33, 0.33, 0.33
A11, A22, A33 = 13E-6, 13E-6, 13E-6
temp = temp + [E11, E22, E33, NU12, NU13, NU23, G12, G13, G23, A11, A22, A33]
mat_name = mat_name + [temp]

## if you need to add more materials then add before this line.
del mat_name[mat_name.index(0)]
## end of databa
##-----------------------------------------------------------------------------

## information of the BASE plate.
base_length = 6.0 ## note the units are in inches.
```

```
base_breadth = 4.5

a = base_length
b = base_breadth

base_composite = 1 ## 0 - homogeneous material, 1 - composite
if(base_composite == 0):
    base_material = 0
    base_thick = 0.12  ## the thickness of the laminate.
else:
## base plate composite lay-up.
    base_layup = layup_ge_b
    base_material = layup_material_b
    base_t = layup_thic_b
    base_thick = 0
    for ij in range(len(base_t)):
        base_thick = base_thick + base_t[ij]

## NOTE THAT LAYUP IS FROM THE BOTTOM OF THE LAMINATE
## IN THE DIRECTION OF THE SWEEP FOR THE EDGE.
##-------------------------------------------------------------
## information of the WEB plate.
web_length = 3.5
web_breadth = 3.5

a1 = web_length
b1 = web_breadth

yspacing = 0.5 ## this is the vertical distance from which the
## the displacements are measured.

dist_web_base = 0.125 ## distance btwn the base of the web plate and
## the base plate.
## position of the bolts..
bolt_x = 4.5
bolt_y = 3.0

web_composite = 1 ## 0 - homogeneous material, 1 - composite

if(web_composite == 0):
    web_material = 0
    web_thick = 1.802
else:
## base plate composite lay-up.
    web_layup = layup_ge_v
    web_material = layup_material_v
    web_t = layup_thic_v
    web_thick = 0
    for ij in range(len(web_t)):
        web_thick = web_thick + web_t[ij]
##-------------------------------------------------------------

## information of the clips
clip_len = 1.125
clip_breadth = 0.8
clip_depth = 2.75
clip_radi = 0.120  ## average of the range.
```

```
clip_composite = 1 ## 0 - homogeneous material, 1 - composite
if(clip_composite == 0):
    clip_material = 0
    clip_thick = 1.802
else:
## base plate composite lay-up.
    clip_layup = layup_ge
    clip_material = clip_material
    clip_t = layup_thic
    clip_thick = 0
    for ij in range(len(clip_t)):
        clip_thick = clip_thick + clip_t[ij]

cx1, cy1 = clip_len, 0.0
cx2, cy2 = clip_len - clip_thick, 0.0
cx3, cy3 = cx2, -clip_breadth + clip_thick + clip_radi
cx4, cy4 = clip_len - clip_thick - clip_radi, - clip_breadth + clip_thick
cx5, cy5 = 0.0, -clip_breadth + clip_thick
cx6, cy6 = 0.0, -clip_breadth
cx7, cy7 = cx4, cy6
cx8, cy8 = clip_len, cy3
cx0, cy0 = clip_len-clip_thick-clip_radi, - clip_breadth + clip_thick +
clip_radi

## changed clip information by taking out the layers.
## information of the clips
clip_len_ch = 1.125
clip_breadth_ch = 0.8
clip_depth_ch = 2.75
clip_radi_ch = 0.12  ## average of the range.
clip_composite_ch = 1 ## 0 - homogeneous material, 1 - composite
if(clip_composite_ch == 0):
    clip_material_ch = 0
    clip_thick_ch = 1.802
else:
## base plate composite lay-up.
    clip_layup_ch = layup_ge_ch
    clip_material_ch = layup_material_ch
    clip_t_ch = layup_thic_ch
    clip_thick_ch = 0
    for ij in range(len(clip_t_ch)):
        clip_thick_ch = clip_thick_ch + clip_t_ch[ij]

cx1_ch, cy1_ch = clip_len_ch, 0.0
cx2_ch, cy2_ch = clip_len_ch - clip_thick_ch, 0.0
cx3_ch, cy3_ch = cx2_ch, -clip_breadth_ch + clip_thick_ch + clip_radi_ch
cx4_ch, cy4_ch = clip_len_ch - clip_thick_ch - clip_radi_ch, -
clip_breadth_ch + clip_thick_ch
cx5_ch, cy5_ch = 0.0, -clip_breadth_ch + clip_thick_ch
cx6_ch, cy6_ch = 0.0, -clip_breadth_ch
cx7_ch, cy7_ch = cx4_ch, cy6_ch
cx8_ch, cy8_ch = clip_len_ch, cy3_ch
cx0_ch, cy0_ch = clip_len_ch-clip_thick_ch-clip_radi_ch, - clip_breadth_ch +
clip_thick_ch + clip_radi_ch


##-------------------------------------------------------------
```

```
## information of the adhesives
adh_1_length = 2.75
adh_1_length_ch = 3.35*2.0/3.0
adh_1_thick = 0.010
adh_1_breadth = clip_breadth-clip_thick-clip_radi
adh_1_material = 1

a2 = adh_1_length
b2 = adh_1_breadth
## adh_1 is on the vertical plate.

adh_2_length = 2.75
adh_2_length_ch = 3.350*2.0/3.0
adh_2_thick = 0.010
adh_2_breadth = clip_len-clip_thick-clip_radi
adh_2_material = 1

a3 = adh_2_length
b3 = adh_2_breadth
## the adhesive are assumed to be of HOMOGENEOUS material

## the following is the information for incorporating L Shape Aluminum
## layer over the exposed vertical plate.
## ply_al = thickness of the aluminum layer.
## Ls - L-Shape aluminum layer.
Ls_length = web_length
Ls_breadth = web_breadth - clip_len - adh_1_thick + base_thick  # the exposed
area.

Lsa = 2.25*2.0
Lsb = 2.25
LShape_material = 2

##------------------------------------------------------------------------

###information of defect_1
defect_1a = 0.5
defect_1b = 0.5
adh_defect = adh_1_thick ## adhesive thickness for the defects.
d_1_t_al = 0.025
d_1_t = adh_defect + d_1_t_al # adhesive thickness
d_1x = defect_1a/2.0
d_1y = defect_1b/2.0
defect_1_material = 3

# information for defect_2
defect_2a = 0.5
defect_2b = 0.5
d_2_t_al = 0.025
d_2_t = adh_defect + d_2_t_al # adhesive thickness
d_2x = defect_2a/2.0
d_2y = defect_2b/2.0
defect_2_material = 3

##------------------------------------------------------------------------
##-------------------------------------------------
jointmodel = mdb.Model(name='Joint Model')
```

```
##-----------------------------------------------------
def base_part():
    global cxa, cya, cxb, cyb, cxc, cyc, cxd, cyd
    print' Creating the Base Plate'
    # coordinates of the verticess
    rectcd = ((a/2.0,b/2.0), (-a/2.0,b/2.0), (-a/2.0,-b/2.0), (a/2.0,-b/2.0),
(0.0,0.0))
    s = jointmodel.Sketch(name='J-BasePlate', sheetSize = 200.0)
    # creating the rectangle.
    s.rectangle(point1=rectcd[1], point2=rectcd[3])
    jointpart = jointmodel.Part(name='Base Plate', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=base_thick)

    print ' Creating the Web Plate'
    # coordinates of the verticess
    rectcd = ((a1/2.0,b1/2.0), (-a1/2.0,b1/2.0), (-a1/2.0,-b1/2.0), (a1/2.0,-
b1/2.0), (0.0,0.0))
    s = jointmodel.Sketch(name='J-WebPlate', sheetSize = 200.0)
    s.rectangle(point1=rectcd[1], point2=rectcd[3])
    jointpart = jointmodel.Part(name='Web Plate', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=web_thick)

    print '  Creating the additional Aluminum L-Shape layer'
    ## creating additional Aluminum layers on the Web Plate.
    rectcd = ((-Lsa/2.0, Lsb/2.0), (0, Lsb/2.0), (0, -Lsb/2.0), (-Lsa/2.0, -
Lsb/2.0))
    s = jointmodel.Sketch(name='L-Shape', sheetSize = 200.0)
    s.Line(point1 = rectcd[0], point2 = rectcd[1])
    s.Line(point1 = rectcd[1], point2 = rectcd[2])
    s.Line(point1 = rectcd[2], point2 = rectcd[3])
    s.Line(point1 = rectcd[3], point2 = rectcd[0])
    jointpart = jointmodel.Part(name='LShape', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=ply_al)

    print ' Creating adhesive bonds'
    ## coordinates for the adhesives
    rectcd = ((a2/2.0,b2/2.0), (-a2/2.0,b2/2.0), (-a2/2.0,-b2/2.0), (a2/2.0,-
b2/2.0), (0.0,0.0))
    s = jointmodel.Sketch(name='J-Adhesive-1', sheetSize = 200.0)
    s.rectangle(point1=rectcd[1], point2=rectcd[3])
    jointpart = jointmodel.Part(name='Adhesive - 1', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=adh_1_thick)

    ## coordinates for the adhesives
    rectcd = ((a3/2.0,b3/2.0), (-a3/2.0,b3/2.0), (-a3/2.0,-b3/2.0), (a3/2.0,-
b3/2.0), (0.0,0.0))
    s = jointmodel.Sketch(name='J-Adhesive-2', sheetSize = 200.0)
    s.rectangle(point1=rectcd[1], point2=rectcd[3])
    jointpart = jointmodel.Part(name='Adhesive - 2', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=adh_2_thick)

    print ' Creating the First Clip - Layup changed'
```

```
    ## coordinates for the clips.
    rectcd_ch = ((cx1_ch,cy1_ch), (cx2_ch,cy2_ch), (cx3_ch,cy3_ch),
(cx4_ch,cy4_ch), (cx5_ch,cy5_ch),
                (cx6_ch,cy6_ch), (cx7_ch,cy7_ch), (cx8_ch,cy8_ch),
(cx0_ch,cy0_ch))
    s = jointmodel.Sketch(name='J-Clip2', sheetSize = 200.0)
    s.Line(point1 = rectcd_ch[7], point2 = rectcd_ch[2])
    s.ArcByCenterEnds(center = (cx0_ch, cy0_ch), point1 = rectcd_ch[2],
point2 = rectcd_ch[3])
    s.Line(point1 = rectcd_ch[3], point2 = rectcd_ch[6])
    s.ArcByCenterEnds(center = rectcd_ch[8], point1 = rectcd_ch[7], point2 =
rectcd_ch[6])
    jointpart = jointmodel.Part(name='Clip-2', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=clip_depth_ch)

    s = jointmodel.Sketch(name='J-Clip1', sheetSize = 200.0)
    s.Line(point1 = rectcd_ch[0], point2 = rectcd_ch[1])
    s.Line(point1 = rectcd_ch[1], point2 = rectcd_ch[2])
    s.Line(point1 = rectcd_ch[2], point2 = rectcd_ch[7])
    s.Line(point1 = rectcd_ch[7], point2 = rectcd_ch[0])
    jointpart = jointmodel.Part(name='Clip-1', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=clip_depth_ch)

    s = jointmodel.Sketch(name='J-Clip3', sheetSize = 200.0)
    s.Line(point1 = rectcd_ch[3], point2 = rectcd_ch[4])
    s.Line(point1 = rectcd_ch[4], point2 = rectcd_ch[5])
    s.Line(point1 = rectcd_ch[5], point2 = rectcd_ch[6])
    s.Line(point1 = rectcd_ch[6], point2 = rectcd_ch[3])
    jointpart = jointmodel.Part(name='Clip-3', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=clip_depth_ch)

    print ' Creating the second Clip'
    ## coordinates for the clips.
    rectcd = ((cx1,cy1), (cx2,cy2), (cx3,cy3), (cx4,cy4), (cx5,cy5),
              (cx6,cy6), (cx7,cy7), (cx8,cy8), (cx0,cy0))
    s = jointmodel.Sketch(name='J-Clip22', sheetSize = 200.0)
    s.Line(point1 = rectcd[7], point2 = rectcd[2])
    s.ArcByCenterEnds(center = (cx0, cy0), point1 = rectcd[2], point2 =
rectcd[3])
    s.Line(point1 = rectcd[3], point2 = rectcd[6])
    s.ArcByCenterEnds(center = rectcd[8], point1 = rectcd[7], point2 =
rectcd[6])
    jointpart = jointmodel.Part(name='Clip-22', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=clip_depth)

    s = jointmodel.Sketch(name='J-Clip12', sheetSize = 200.0)
    s.Line(point1 = rectcd[0], point2 = rectcd[1])
    s.Line(point1 = rectcd[1], point2 = rectcd[2])
    s.Line(point1 = rectcd[2], point2 = rectcd[7])
    s.Line(point1 = rectcd[7], point2 = rectcd[0])
    jointpart = jointmodel.Part(name='Clip-12', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=clip_depth)
```

```
    s = jointmodel.Sketch(name='J-Clip32', sheetSize = 200.0)
    s.Line(point1 = rectcd[3], point2 = rectcd[4])
    s.Line(point1 = rectcd[4], point2 = rectcd[5])
    s.Line(point1 = rectcd[5], point2 = rectcd[6])
    s.Line(point1 = rectcd[6], point2 = rectcd[3])
    jointpart = jointmodel.Part(name='Clip-32', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=clip_depth)

    print 'Creating defect - 1'
    ## coordinates of defect -1
    rectcd = ((-d_1x,d_1y), (d_1x,-d_1y))
    s = jointmodel.Sketch(name='Defect-1', sheetSize = 200.0)
    s.rectangle(point1=rectcd[0], point2=rectcd[1])
    jointpart = jointmodel.Part(name='defect_1', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=d_1_t)

    print 'Creating defect - 2'
    ## coordinates of defect - 2
    rectcd = ((-d_2x,d_2y), (d_2x,-d_2y))
    s = jointmodel.Sketch(name='Defect-2', sheetSize = 200.0)
    s.rectangle(point1=rectcd[0], point2=rectcd[1])
    jointpart = jointmodel.Part(name='defect_2', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
    jointpart.BaseSolidExtrude(sketch=s, depth=d_2_t)

    jointmodel.convertAllSketches()
#-------------------------------------------------------------
def joint_section():
    global cx24m, cy24m
    p2 = jointmodel.parts['Base Plate']
    c,f,e = p2.cells, p2.faces, p2.edges
    CP = p2.DatumPlaneByThreePoints(point1 =(0.0, b/2.0, 0.0), point2 = (0.0,
-b/2.0, clip_depth),
                                    point3 = (0.0, -b/2.0, 0.0))
    face = f.findAt((0.0, 0.0, 0.0),)
    datums = CP.id
    face = f.findAt((0.0, -b/2.0,base_thick/2.0), )
    datums = CP.id
    name_2 = 'Base_section'
    regi=regionToolset.Region(cells=c)
    if(base_composite == 1):
        layup = [0]
        for ij in range(len(base_t)):
            k = base_material[ij]

            name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=base_t[ij], orientAngle=base_layup[ij],
numIntPts=5)
            layup = layup + [name1]
        del layup[layup.index(0)]
        section_2 = jointmodel.CompositeShellSection(name=name_2,
layup=layup)
        CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN, origin=(-
a/2.0,-b/2.0,0),
```

```
            point1=(a/2.0,-b/2.0,0.0), point2=(-a/2.0,b/2.0,0.0),
name='basecsys')
        csind = CS.id
    else:
        CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN, origin=(-
a/2.0,-b/2.0,0),
            point1=(a/2.0,-b/2.0,0.0), point2=(-a/2.0,b/2.0,0.0),
name='basecsys')
        csind = CS.id
        k = base_material
        section_2 = jointmodel.HomogeneousSolidSection(name=name_2, material
= mat_name[k][1])
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind],
axis=AXIS_3)
    p2.SectionAssignment(region=regi, sectionName=name_2)
##-----------------------------------------------------------------------------
    p2 = jointmodel.parts['Web Plate']
    c,f,e = p2.cells, p2.faces, p2.edges
    name_2 = 'Web_section'
    regi=regionToolset.Region(cells=c)
    if(web_composite == 1):
        c = p2.cells
        layup = [0]
        for ij in range(len(web_t)):
            k = web_material[ij]

            name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=web_t[ij], orientAngle=web_layup[ij],
numIntPts=5)
            layup = layup + [name1]
        del layup[layup.index(0)]
        section_2 = jointmodel.CompositeShellSection(name=name_2,
layup=layup)
        CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN, origin=(-
a1/2.0,-b1/2.0,0),
            point1=(a1/2.0,-b1/2.0,0.0), point2=(-a1/2.0,b1/2.0,0.0),
name='webcsys')
        csind = CS.id
    else:
        CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN, origin=(-
a1/2.0,-b1/2.0,0),
            point1=(a1/2.0,-b1/2.0,0.0), point2=(-a1/2.0,b1/2.0,0.0),
name='webcsys')
        csind = CS.id
        k = web_material
        section_2 = jointmodel.HomogeneousSolidSection(name=name_2, material
= mat_name[k][1])
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind],
axis=AXIS_3)
    p2.SectionAssignment(region=regi, sectionName=name_2)
#-----------------------------------------------------------------------------
    p2 = jointmodel.parts['LShape']
    c,f,e = p2.cells, p2.faces, p2.edges
    name_2 = 'LShape_section'
    regi=regionToolset.Region(cells=c)
    CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN, origin=(-a1/2.0,-
b1/2.0,0),
```

```
            point1=(a1/2.0,-b1/2.0,0.0), point2=(-a1/2.0,b1/2.0,0.0),
name='LShapesys')
    csind = CS.id
    k = LShape_material
    layup = [0]
    name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=ply_alt, orientAngle=0, numIntPts=5)
    layup = layup + [name1]
    k = adh_1_material
    name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=adh_1_thick, orientAngle=0, numIntPts=5)
    layup = layup + [name1]

    del layup[layup.index(0)]
    section_2 = jointmodel.CompositeShellSection(name=name_2, layup=layup)
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind],
axis=AXIS_3)
    p2.SectionAssignment(region=regi, sectionName=name_2)
##-----------------------------------------------------------------------
    p2 = jointmodel.parts['Adhesive - 1']
    c,f,e = p2.cells, p2.faces, p2.edges
    CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN, origin=(a2/2.0,-
b2/2.0,0),
        point1=(a2/2.0,b2/2.0,0.0), point2=(-a2/2.0,-b2/2.0,0.0),
name='adh1csys')
    csind = CS.id
    name_2 = 'Adhesive_1_section'
    regi = regionToolset.Region(cells = c)
    k = adh_1_material
    layup = [0]
    name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=adh_1_thick, orientAngle=0, numIntPts=5)
    layup = layup + [name1]
    del layup[layup.index(0)]
    section_2 = jointmodel.CompositeShellSection(name=name_2, layup=layup)
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind],
axis=AXIS_3)
    p2.SectionAssignment(region=regi, sectionName=name_2)
    p2 = jointmodel.parts['Adhesive - 2']
    c,f,e = p2.cells, p2.faces, p2.edges
    CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN, origin=(a2/2.0,-
b2/2.0,0),
        point1=(a2/2.0,b2/2.0,0.0), point2=(-a2/2.0,-b2/2.0,0.0),
name='adh2csys')
    csind = CS.id
    name_2 = 'Adhesive_2_section'
    regi = regionToolset.Region(cells = c)
    k = adh_2_material
    layup = [0]
    name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=adh_2_thick, orientAngle=0, numIntPts=5)
    layup = layup + [name1]
    del layup[layup.index(0)]
    section_2 = jointmodel.CompositeShellSection(name=name_2, layup=layup)
    p2.SectionAssignment(region=regi, sectionName=name_2)
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind],
axis=AXIS_3)
```

```
##-----------------------------------------------------------------------------
##-----------------------------------------------------------------------------
    ## CHANGED FOR THE CHANGED LAYUP
    p2 = jointmodel.parts['Clip-3']
    c,f,e = p2.cells, p2.faces, p2.edges
    CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN,
origin=((cx6_ch+cx5_ch)*0.5,(cy6_ch+cy5_ch)*0.5,0.0),
        point1=((cx6_ch+cx5_ch)*0.5,(cy6_ch+cy5_ch)*0.5,clip_depth_ch),
point2=((cx7_ch+cx4_ch)*0.5,(cy4_ch+cy7_ch)*0.5,0.0), name='clip1csys')
    csind = CS.id
    name_2 = 'clip_section_1'
    regi=regionToolset.Region(cells=c)
    if(clip_composite_ch == 1):
        c = p2.cells
        layup_ch = [0]
        for ij in range(len(clip_t_ch)):
            k = clip_material[ij]
            name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=clip_t_ch[ij], orientAngle=clip_layup_ch[ij],
numIntPts=5)
            layup_ch = layup_ch + [name1]
        del layup_ch[layup_ch.index(0)]
        section_2 = jointmodel.CompositeShellSection(name=name_2,
layup=layup_ch)

    else:
        k = clip_material_ch
        section_2 = jointmodel.HomogeneousSolidSection(name=name_2, material
= mat_name[k][1])
    p2.SectionAssignment(region=regi, sectionName=name_2)
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind],
axis=AXIS_3)

    p2 = jointmodel.parts['Clip-1']
    c,f,e = p2.cells, p2.faces, p2.edges
    CS2 = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN,
origin=((cx3_ch+cx8_ch)*0.5,(cy3_ch+cy8_ch)*0.5,0.0),
        point1=((cx3_ch+cx8_ch)*0.5,(cy3_ch+cy8_ch)*0.5,clip_depth_ch),
point2=((cx1_ch+cx2_ch)*0.5,(cy1_ch+cy2_ch)*0.5,0.0), name='clip2csys')
    csind2 = CS2.id
    name_2 = 'clip_section_2'
    regi=regionToolset.Region(cells=c)
    if(clip_composite == 1):
        c = p2.cells
        layup_ch = [0]
        for ij in range(len(clip_t_ch)):
            k = clip_material_ch[ij]

            name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=clip_t_ch[ij], orientAngle=clip_layup_ch[ij],
numIntPts=5)
            layup_ch = layup_ch + [name1]
        del layup_ch[layup_ch.index(0)]
        section_2 = jointmodel.CompositeShellSection(name=name_2,
layup=layup_ch)

    else:
```

```
        k = clip_material_ch
        section_2 = jointmodel.HomogeneousSolidSection(name=name_2, material
= mat_name[k][1])
    p2.SectionAssignment(region=regi, sectionName=name_2)
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind2],
axis=AXIS_3)
## partitioning start.
    p2 = jointmodel.parts['Clip-2']
    c,f,e = p2.cells, p2.faces, p2.edges
    theta = 90/nos
    th = 0
    for ij in range(nos - 1):
        th = theta * (ij+1)
        xc = (clip_radi_ch+clip_thick_ch/2.0)*cos(th*pi/180)
        yc = (clip_radi_ch+clip_thick_ch/2.0)*sin(th*pi/180)
        xcc = clip_len_ch - clip_thick_ch - clip_radi_ch
        ycc = clip_breadth_ch - clip_thick_ch - clip_radi_ch
        CP = p2.DatumPlaneByThreePoints(point1 =(xcc, -ycc, 0.0), point2 =
(xcc, -ycc, clip_depth),
                                        point3 = (xc+xcc, -yc-ycc, 0.0))
        c = p2.cells
        cell =c.findAt((xcc+xc, -ycc-yc, 0.0),)
        datums = CP.id
        p2.PartitionCellByDatumPlane(cells = cell,
datumPlane=p2.datums[datums])
    theta = 90/nos
    th = 0
    for ij in range(nos):
        th = theta * (ij+1)
        xc = clip_radi_ch*cos(th*pi/180)
        yc = clip_radi_ch*sin(th*pi/180)
        xcc = clip_len_ch - clip_thick_ch - clip_radi_ch
        ycc = clip_breadth_ch - clip_thick_ch - clip_radi_ch
        thet = theta*(2.0*ij+1.0)*0.5
        cx24m = xcc + (clip_radi_ch+clip_thick_ch/2.0)*cos(thet*pi/180)
        cy24m = ycc + (clip_radi_ch+clip_thick_ch/2.0)*sin(thet*pi/180)

        cx87m = cx24m +
(clip_radi_ch+clip_thick_ch/2.0)*sin(thet*pi/180)*tan(thet*pi/180)
        CS3 = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN,
origin=(cx24m,-cy24m,0.0),
            point1=(cx24m,-cy24m,clip_depth), point2=(cx87m,-ycc,0.0),
name='clip3csys')
        csind3 = CS3.id
        name_2 = 'clip_section_3'
        cels = c.findAt((cx24m, -cy24m, 0.0),)
        cel = cels.index
        regi=regionToolset.Region(cells=c[cel:cel+1])
        if(clip_composite_ch == 1):
            c = p2.cells
            layup_ch = [0]
            for ij in range(len(clip_t_ch)):
                k = clip_material_ch[ij]

                name1 = section.SectionLayer(material=mat_name[k][1],
                        thickness=clip_t_ch[ij],
orientAngle=clip_layup_ch[ij], numIntPts=5)
```

```
                layup_ch = layup_ch + [name1]
            del layup_ch[layup_ch.index(0)]
            section_2 = jointmodel.CompositeShellSection(name=name_2,
layup=layup_ch)

        else:
            k = clip_material
            section_2 = jointmodel.HomogeneousSolidSection(name=name_2,
material = mat_name[k][1])
        p2.SectionAssignment(region=regi, sectionName=name_2)
        p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind3],
axis=AXIS_3)

    theta = 90/nos
    for ij in range(nos):
        th = theta * (ij+1)
        xc = clip_radi_ch*cos(th*pi/180)
        yc = clip_radi_ch*sin(th*pi/180)
        xcc = clip_len_ch - clip_thick_ch - clip_radi_ch
        ycc = clip_breadth_ch - clip_thick_ch - clip_radi_ch
        thet = theta*(2.0*ij+1.0)*0.5
        cx24m = xcc + clip_radi_ch*cos(thet*pi/180)
        cy24m = ycc + clip_radi_ch*sin(thet*pi/180)

        p2 = jointmodel.parts['Clip-2']
        c,f,e = p2.cells, p2.faces, p2.edges
        cell =c.findAt((cx24m, -cy24m, 0.0),)
        celli = cell.index
        p2.setMeshControls(regions=c[celli:celli+1], technique=SWEEP,
algorithm=MEDIAL_AXIS)
        thetaa = theta*ij
        xce, yce = (clip_radi_ch + clip_thick_ch/2.0)*cos(thetaa*pi/180),
(clip_radi_ch + clip_thick_ch/2.0)*sin(thetaa*pi/180)
        edge = e.findAt((xce+xcc, -ycc-yce, 0.0),)
        ein = edge.index
        p2.setSweepPath(region=c[celli], edge=edge, sense=FORWARD)
        elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)

        p2.seedEdgeByNumber(edges=(edge,), number = 1, constraint=FIXED)
    p2.seedPart(size=0.2, deviationFactor=1.8)
##    p2.seedPart(size=0.1, deviationFactor=0.1)
##--------------------------------------------------------------------------
##--------------------------------------------------------------------------
    p2 = jointmodel.parts['Clip-32']
    c,f,e = p2.cells, p2.faces, p2.edges
    CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN,
origin=((cx7+cx4)*0.5,(cy7+cy4)*0.5,0.0),
        point1=((cx7+cx4)*0.5,(cy7+cy4)*0.5,clip_depth),
point2=((cx5+cx6)*0.5,(cy5+cy6)*0.5,0.0), name='clip1csys')
    csind = CS.id
    name_2 = 'clip_section_12'
    regi=regionToolset.Region(cells=c)
    if(clip_composite == 1):
        c = p2.cells
        layup = [0]
        for ij in range(len(clip_t)):
            k = clip_material[ij]
```

```
            name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=clip_t[ij], orientAngle=clip_layup[ij],
numIntPts=5)
            layup = layup + [name1]
        del layup[layup.index(0)]
        section_2 = jointmodel.CompositeShellSection(name=name_2,
layup=layup)

    else:
        k = clip_material
        section_2 = jointmodel.HomogeneousSolidSection(name=name_2, material
= mat_name[k][1])
    p2.SectionAssignment(region=regi, sectionName=name_2)
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind],
axis=AXIS_3)

    p2 = jointmodel.parts['Clip-12']
    c,f,e = p2.cells, p2.faces, p2.edges
    CS2 = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN,
origin=((cx1+cx2)*0.5,(cy1+cy2)*0.5,0.0),
        point1=((cx1+cx2)*0.5,(cy1+cy2)*0.5,clip_depth),
point2=((cx3+cx8)*0.5,(cy3+cy8)*0.5,0.0), name='clip2csys')
    csind2 = CS2.id
    name_2 = 'clip_section_22'
    regi=regionToolset.Region(cells=c)
    if(clip_composite == 1):
        c = p2.cells
        layup = [0]
        for ij in range(len(clip_t)):
            k = clip_material[ij]
            name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=clip_t[ij], orientAngle=clip_layup[ij],
numIntPts=5)
            layup = layup + [name1]
        del layup[layup.index(0)]
        section_2 = jointmodel.CompositeShellSection(name=name_2,
layup=layup)
    else:
        k = clip_material
        section_2 = jointmodel.HomogeneousSolidSection(name=name_2, material
= mat_name[k][1])
    p2.SectionAssignment(region=regi, sectionName=name_2)
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind2],
axis=AXIS_3)
    p2 = jointmodel.parts['Clip-22']
    c,f,e = p2.cells, p2.faces, p2.edges
    theta = 90/nos
    th = 0
    for ij in range(nos - 1):
        th = theta * (ij+1)
        xc = clip_radi*cos(th*pi/180)
        yc = clip_radi*sin(th*pi/180)
        xcc = clip_len - clip_thick - clip_radi
        ycc = clip_breadth - clip_thick - clip_radi
        CP = p2.DatumPlaneByThreePoints(point1 =(xcc, -ycc, 0.0), point2 =
(xcc, -ycc, clip_depth),
                                        point3 = (xc+xcc, -yc-ycc, 0.0))
```

```
        c = p2.cells
        cell =c.findAt((xcc+xc, -ycc-yc, 0.0),)
        datums = CP.id
        p2.PartitionCellByDatumPlane(cells = cell,
datumPlane=p2.datums[datums])
    theta = 90/nos
    th = 0
    for ij in range(nos):
        th = theta * (ij+1)
        xc = clip_radi*cos(th*pi/180)
        yc = clip_radi*sin(th*pi/180)
        xcc = clip_len - clip_thick - clip_radi
        ycc = clip_breadth - clip_thick - clip_radi
        thet = theta*(2.0*ij+1.0)*0.5
        cx24m = xcc + (clip_radi+clip_thick/2.0)*cos(thet*pi/180)
        cy24m = ycc + (clip_radi+clip_thick/2.0)*sin(thet*pi/180)

        cy87m = ycc + (clip_radi+clip_thick/2.0)/sin(thet*pi/180)
        CS3 = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN,
origin=(cx24m,-cy24m,0.0),
            point1=(cx24m,-cy24m,clip_depth), point2=(xcc,-cy87m,0.0),
name='clip3csys')
        csind3 = CS3.id
        name_2 = 'clip_section_3'
        cels = c.findAt((cx24m, -cy24m, 0.0),)
        cel = cels.index
        regi=regionToolset.Region(cells=c[cel:cel+1])
        if(clip_composite == 1):
            c = p2.cells
            layup = [0]
            for ij in range(len(clip_t)):
                k = clip_material[ij]

                name1 = section.SectionLayer(material=mat_name[k][1],
                        thickness=clip_t[ij], orientAngle=clip_layup[ij],
numIntPts=5)
                layup = layup + [name1]
            del layup[layup.index(0)]
            section_2 = jointmodel.CompositeShellSection(name=name_2,
layup=layup)
        else:
            k = clip_material
            section_2 = jointmodel.HomogeneousSolidSection(name=name_2,
material = mat_name[k][1])
        p2.SectionAssignment(region=regi, sectionName=name_2)
        p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind3],
axis=AXIS_3)
    theta = 90/nos
    for ij in range(nos):
        th = theta * (ij+1)
        xc = clip_radi*cos(th*pi/180)
        yc = clip_radi*sin(th*pi/180)
        xcc = clip_len - clip_thick - clip_radi
        ycc = clip_breadth - clip_thick - clip_radi
        thet = theta*(2.0*ij+1.0)*0.5
        cx24m = xcc + clip_radi*cos(thet*pi/180)
        cy24m = ycc + clip_radi*sin(thet*pi/180)
```

```
        p2 = jointmodel.parts['Clip-22']
        c,f,e = p2.cells, p2.faces, p2.edges
        cell =c.findAt((cx24m, -cy24m, 0.0),)
        celli = cell.index
        p2.setMeshControls(regions=c[celli:celli+1], technique=SWEEP,
algorithm=MEDIAL_AXIS)
        thetaa = theta*ij
        xce, yce = (clip_radi + clip_thick/2.0)*cos(thetaa*pi/180),
(clip_radi + clip_thick/2.0)*sin(thetaa*pi/180)
        edge = e.findAt((xce+xcc, -ycc-yce, 0.0),)
        ein = edge.index
        p2.setSweepPath(region=c[celli], edge=edge, sense=REVERSE)
        elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
        p2.seedEdgeByNumber(edges=(edge,), number = 1, constraint=FIXED)
    p2.seedPart(size=0.2)
##-----------------------------------------------------------------------
    p2 = jointmodel.parts['defect_1']
    c,f,e = p2.cells, p2.faces, p2.edges
    CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN, origin=(d_1x,-
d_1y,0),
        point1=(d_1x,d_1y,0.0), point2=(-d_1x,-d_1y,0.0), name='d1csys')
    csind = CS.id
    name_2 = 'Defect_1_section'
    regi = regionToolset.Region(cells = c)
    k = adh_1_material
    layup = [0]
    name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=adh_defect, orientAngle=0, numIntPts=5)
    layup = layup + [name1]
    k = defect_1_material
    name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=d_1_t_al, orientAngle=0, numIntPts=5)
    layup = layup + [name1]
    del layup[layup.index(0)]
    section_2 = jointmodel.CompositeShellSection(name=name_2, layup=layup)
    p2.SectionAssignment(region=regi, sectionName=name_2)
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind],
axis=AXIS_3)
    p2 = jointmodel.parts['defect_2']
    c,f,e = p2.cells, p2.faces, p2.edges
    CS = p2.DatumCsysByThreePoints(coordSysType=CARTESIAN, origin=(d_2x,-
d_2y,0),
        point1=(d_2x,d_2y,0.0), point2=(-d_2x,-d_2y,0.0), name='d2csys')
    csind = CS.id
    name_2 = 'Defect_2_section'
    regi = regionToolset.Region(cells = c)
    k = adh_1_material
    layup = [0]
    name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=adh_defect, orientAngle=0, numIntPts=5)
    layup = layup + [name1]
    k = defect_1_material
    name1 = section.SectionLayer(material=mat_name[k][1],
                    thickness=d_2_t_al, orientAngle=0, numIntPts=5)
    layup = layup + [name1]
    del layup[layup.index(0)]
    section_2 = jointmodel.CompositeShellSection(name=name_2, layup=layup)
```

```
    p2.SectionAssignment(region=regi, sectionName=name_2)
    p2.assignMaterialOrientation(region=regi,localCsys=p2.datums[csind],
axis=AXIS_3)
##-------------------------------------------------------------------------
##-----------------------------------------------------
def joint_sweeppaths():
    p2=jointmodel.parts['Base Plate']
    c, e, v = p2.cells, p2.edges, p2.vertices
    cellin = c.findAt((0.0,0.0,0.0),)
    celli = cellin.index
    p2.setMeshControls(regions=c[celli:celli+1], technique=SWEEP,
algorithm=MEDIAL_AXIS)
    edge = e.findAt((a/2.0, b/2.0, base_thick/3.0),)
    p2.setSweepPath(region=c[celli], edge=edge, sense=FORWARD)
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    p2.seedPart(size=0.2, deviationFactor=0.1)
    p2.seedEdgeByNumber(edges=(edge,), number = 1, constraint=FIXED)
    p2=jointmodel.parts['Web Plate']
    c, e, v = p2.cells, p2.edges, p2.vertices
    cellin = c.findAt((0.0,0.0,0.0),)
    celli = cellin.index
    p2.setMeshControls(regions=c[celli:celli+1], technique=SWEEP,
algorithm=MEDIAL_AXIS)
    edge = e.findAt((a1/2.0, b1/2.0, base_thick/3.0),)
    p2.setSweepPath(region=c[celli], edge=edge, sense=FORWARD)
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    p2.seedPart(size=0.2, deviationFactor=0.1)
    p2.seedEdgeByNumber(edges=(edge,), number = 1, constraint=FIXED)
##---------------------------------------
    p2 = jointmodel.parts['Clip-3']
    c,f,e = p2.cells, p2.faces, p2.edges
    cellin = c.findAt((cx4_ch,cy4_ch,0.0),)
    celli = cellin.index
    p2.setMeshControls(regions=c[celli:celli+1], technique=SWEEP,
algorithm=MEDIAL_AXIS)
    edge = e.findAt((cx4_ch, (cy4_ch+cy7_ch)*0.5, 0.0),)
    p2.setSweepPath(region=c[celli], edge=edge, sense=FORWARD)
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    p2.seedPart(size=0.2, deviationFactor=0.1)
    p2.seedEdgeByNumber(edges=(edge,), number = 1, constraint=FIXED)
    p2 = jointmodel.parts['Clip-1']
    c,f,e = p2.cells, p2.faces, p2.edges
    cellin = c.findAt((cx1_ch,cy1_ch,0.0),)
    celli = cellin.index
    p2.setMeshControls(regions=c[celli:celli+1], technique=SWEEP,
algorithm=MEDIAL_AXIS)
    edge = e.findAt(((cx1_ch+cx2_ch)*0.5, cy1_ch, 0.0),)
    p2.setSweepPath(region=c[celli], edge=edge, sense=FORWARD)
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    p2.seedPart(size=0.2, deviationFactor=0.1)
    p2.seedEdgeByNumber(edges=(edge,), number = 1, constraint=FIXED)
##---------------------------------------
    p2 = jointmodel.parts['Clip-32']
    c,f,e = p2.cells, p2.faces, p2.edges
    cellin = c.findAt((cx4,cy4,0.0),)
    celli = cellin.index
```

```
    p2.setMeshControls(regions=c[celli:celli+1], technique=SWEEP,
algorithm=MEDIAL_AXIS)
    edge = e.findAt((cx4, (cy4+cy7)*0.5, 0.0),)
    p2.setSweepPath(region=c[celli], edge=edge, sense=REVERSE)
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    p2.seedPart(size=0.2, deviationFactor=0.1)
    p2.seedEdgeByNumber(edges=(edge,), number = 1, constraint=FIXED)
    p2 = jointmodel.parts['Clip-12']
    c,f,e = p2.cells, p2.faces, p2.edges
    cellin = c.findAt((cx1,cy1,0.0),)
    celli = cellin.index
    p2.setMeshControls(regions=c[celli:celli+1], technique=SWEEP,
algorithm=MEDIAL_AXIS)
    edge = e.findAt(((cx1+cx2)*0.5, cy1, 0.0),)
    p2.setSweepPath(region=c[celli], edge=edge, sense=REVERSE)
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    p2.seedPart(size=0.2, deviationFactor=0.1)
    p2.seedEdgeByNumber(edges=(edge,), number = 1, constraint=FIXED)
##-------------------------------------------------
    p2 = jointmodel.parts['Adhesive - 1']
    c,f,e = p2.cells, p2.faces, p2.edges
    cellin = c.findAt((0.0,0.0,0.0),)
    celli = cellin.index
    edge = e.findAt((a2/2.0, b2/2.0, adh_1_thick/2.0),)
    p2.setSweepPath(region=c[celli], edge=edge, sense=FORWARD)
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    p2.seedPart(size=0.14, deviationFactor=0.1)
    p2 = jointmodel.parts['Adhesive - 2']
    c,f,e = p2.cells, p2.faces, p2.edges
    cellin = c.findAt((0.0,0.0,0.0),)
    celli = cellin.index
    edge = e.findAt((a3/2.0, b3/2.0, adh_1_thick/2.0),)
    p2.setSweepPath(region=c[celli], edge=edge, sense=FORWARD)
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    p2.seedPart(size=0.14, deviationFactor=0.1)
##-------------------------------------------------
    p2 = jointmodel.parts['LShape']
    c,f,e = p2.cells, p2.faces, p2.edges
    cellin = c.findAt((0.0,0.0,0.0),)
    celli = cellin.index
    edge = e.findAt((0.0, Lsb/2.0, ply_al/2.0),)
    p2.setSweepPath(region=c[celli], edge=edge, sense=FORWARD)
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    p2.seedPart(size=0.14, deviationFactor=0.1)
##-------------------------------------------------
    ## defect element assignment and meshing
    p2 = jointmodel.parts['defect_1']
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    cell9 = p2.cells
    cels = (cell9,)
    p2.setElementType(regions=cels , elemTypes=(elemType1, ))
    p2.seedPart(size=0.05, deviationFactor=0.1)
    ## defect element assignment and meshing
    p2 = jointmodel.parts['defect_2']
    elemType1 = mesh.ElemType(elemCode=SC8R, elemLibrary=STANDARD)
    cell9 = p2.cells
    cels = (cell9,)
```

```
    p2.setElementType(regions=cels , elemTypes=(elemType1, ))
    p2.seedPart(size=0.05, deviationFactor=0.1)
##------------------------------------------------------------
#------------------------------------------------------------
def clip_partition():
    global xyx, xyy
    p2=jointmodel.parts['Base Plate']
    c = p2.cells
    f = p2.faces
    x = a/2.0
    s = jointmodel.Sketch(name='BasePlatep', sheetSize = 200.0)
    rectcd = ((bolt_x/2.0,bolt_y/2.0), (-bolt_x/2.0,-bolt_y/2.0))
    s.rectangle(point1=rectcd[0], point2=rectcd[1])
    faces = f.findAt((0.0, 0.0, 0.0),)
    p2.PartitionFaceBySketch(faces= faces, sketch=s)
#------------------------------------------------------------
# material properties of the plies.
## material - 1
def base_mater():
    num_mat = 0
## creating lamina material
    for inte in range(len(mat_name)):
        algraphiteepoxy=jointmodel.Material(name=mat_name[inte][1])
        propes = (mat_name[inte][2], mat_name[inte][3], mat_name[inte][4],
                  mat_name[inte][5], mat_name[inte][6], mat_name[inte][7],
                  mat_name[inte][8], mat_name[inte][9], mat_name[inte][10])
        algraphiteepoxy.Elastic(table=(propes, ), type=ENGINEERING_CONSTANTS)
        CTEprop = (mat_name[inte][11], mat_name[inte][12],
mat_name[inte][13])
        algraphiteepoxy.Expansion(type=ORTHOTROPIC, table=(CTEprop,))
        num_mat = num_mat + 1
    print num_mat, '- Materials created'

##------------------------------------------------------------
##
def joint_instance():
    p2 = jointmodel.parts['Base Plate']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'baseinstance', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['baseinstance']
    p2.rotateAboutAxis(axisPoint=(a/2.0, 0.0, 0.0), axisDirection=(-a/2.0,
0.0,
    0.0), angle=90.0)
    p2 = jointmodel.parts['Web Plate']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'webinstance', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['webinstance']
    y = b1/2.0 + base_thick + dist_web_base##+ clip_thick ##+ base_thick ##
web plate moved up.
    ynew = y
    p2.translate(vector=(0.0,y,-web_thick/2.0))
    p2 = jointmodel.parts['LShape']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'Lshapeinstance', part = p2,
dependent = ON)
```

```
    p2 = jointasseme.instances['Lshapeinstance']
    y = Lsb/2.0 + base_thick + clip_len + adh_1_thick
    y = ynew + b1/2.0-Lsb/2.0
    x = a1/2.0
    p2.translate(vector=(x,y,-web_thick/2.0-ply_al))
## base plate adhesive translation and rotation
    p2 = jointmodel.parts['Adhesive - 2']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'adhinstance-2-1', part = p2,
                                      dependent = ON)
    p2=jointasseme.instances['adhinstance-2-1']
    y,z = adh_2_breadth/2.0+base_thick, adh_1_thick + clip_len +
web_thick/2.0
    p2.translate(vector=(0,y,z))

    x1, y1, z1 = adh_1_length/2.0,base_thick,z
    p2.rotateAboutAxis(axisPoint=(x1, y1, z1), axisDirection=(-
adh_1_length,0.0,0.0), angle=90.0)

    p2 = jointmodel.parts['Adhesive - 2']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'adhinstance-2-2', part = p2,
                                      dependent = ON)
    p2=jointasseme.instances['adhinstance-2-2']
    y,z = adh_2_breadth/2.0+base_thick, adh_1_thick + clip_len +
web_thick/2.0 + adh_2_thick
    p2.translate(vector=(0,y,-z))

    x1,y1,z1 = adh_1_length/2.0, base_thick, adh_1_thick + clip_len +
web_thick/2.0
    x2,y2,z2 = -adh_1_length,0.0,0.0
    p2.rotateAboutAxis(axisPoint=(x1,y1,-z1), axisDirection=(x2,y2,z2),
angle=-90.0)
## clip translation and rotation.
    p2 = jointmodel.parts['Clip-1']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'clipinstance-1-1', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['clipinstance-1-1']
    x,y,z = 0, clip_breadth, 0.0
    p2.rotateAboutAxis(axisPoint=(0, 0, 0.0), axisDirection=(x,y,z),
angle=90.0)

    x,y = -adh_2_length/2.0,clip_breadth+base_thick+adh_2_thick
    z = adh_1_thick+web_thick/2.0+clip_len
    p2.translate(vector=(x,y,z))

    p2 = jointmodel.parts['Clip-2']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'clipinstance-2-1', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['clipinstance-2-1']
    x,y,z = 0, clip_breadth, 0.0
    p2.rotateAboutAxis(axisPoint=(0, 0, 0.0), axisDirection=(x,y,z),
angle=90.0)

    x,y = -adh_2_length/2.0,clip_breadth+base_thick+adh_2_thick
```

```
    z = adh_1_thick+web_thick/2.0+clip_len
    p2.translate(vector=(x,y,z))
    p2 = jointmodel.parts['Clip-3']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'clipinstance-3-1', part = p2,
                                      dependent = ON )
    p2 = jointasseme.instances['clipinstance-3-1']
    x,y,z = 0, clip_breadth, 0.0
    p2.rotateAboutAxis(axisPoint=(0, 0, 0.0), axisDirection=(x,y,z),
angle=90.0)
    x,y = -adh_2_length/2.0,clip_breadth+base_thick+adh_2_thick
    z = adh_1_thick+web_thick/2.0+clip_len
    p2.translate(vector=(x,y,z))
## adhesive 1-1 on the web section translate only
    p2 = jointmodel.parts['Adhesive - 1']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'adhinstance-1-1', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['adhinstance-1-1']
    y = -adh_1_breadth/2.0+adh_2_thick+clip_breadth+base_thick
    z = web_thick/2.0
    p2.translate(vector=(0.0,y,z))
## adhesive 1-2 on the web section translate only
    p2 = jointmodel.parts['Adhesive - 1']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'adhinstance-1-2', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['adhinstance-1-2']
    y = -adh_1_breadth/2.0 + adh_2_thick + clip_breadth + base_thick
    z = web_thick/2.0 + adh_1_thick
    p2.translate(vector=(0.0,y,-z))
    jointasseme.DatumPointByCoordinate(coords=(0.0, 0.0, 0.0))
##-----------------------------------------------------------------------
    p2 = jointmodel.parts['Clip-12']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'clipinstance-1-2', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['clipinstance-1-2']
    x,y,z = 0, clip_breadth, 0.0
    p2.rotateAboutAxis(axisPoint=(0, 0, 0.0), axisDirection=(x,y,z), angle=-
90.0)
    x,y = adh_2_length/2.0,clip_breadth+base_thick+adh_2_thick
    z = adh_1_thick+web_thick/2.0+clip_len
    p2.translate(vector=(x,y,-z))
    p2 = jointmodel.parts['Clip-22']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'clipinstance-2-2', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['clipinstance-2-2']
    x,y,z = 0, clip_breadth, 0.0
    p2.rotateAboutAxis(axisPoint=(0, 0, 0.0), axisDirection=(x,y,z), angle=-
90.0)
    x,y = adh_2_length/2.0,clip_breadth+base_thick+adh_2_thick
    z = adh_1_thick+web_thick/2.0+clip_len
    p2.translate(vector=(x,y,-z))
    p2 = jointmodel.parts['Clip-32']
    jointasseme = jointmodel.rootAssembly
```

```
    base_insta = jointasseme.Instance(name = 'clipinstance-3-2', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['clipinstance-3-2']
    x,y,z = 0, clip_breadth, 0.0
    p2.rotateAboutAxis(axisPoint=(0, 0, 0.0), axisDirection=(x,y,z), angle=-
90.0)
    x,y = adh_2_length/2.0,clip_breadth+base_thick+adh_2_thick
    z = adh_1_thick+web_thick/2.0+clip_len
    p2.translate(vector=(x,y,-z))
## defect - 1 instances translate
## clip defects
    p2 = jointmodel.parts['defect_1']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'defect-1-1', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['defect-1-1']
    z = web_thick/2.0
    y = b1 + base_thick + clip_thick - Lsb/2.0
    p2.translate(vector=(0.0,y,z))
    p2 = jointmodel.parts['defect_2']
    jointasseme = jointmodel.rootAssembly
    base_insta = jointasseme.Instance(name = 'defect-1-2', part = p2,
                                      dependent = ON)
    p2 = jointasseme.instances['defect-1-2']
    z = web_thick/2.0
    y = b1 + base_thick + clip_thick - Lsb/2.0
    p2.translate(vector=(2*defect_2a,y,z))
## hiding the datum planes.. not supressing
    session.viewports['Viewport: 1'].setValues(displayedObject=jointasseme)
    session.viewports['Viewport:
1'].assemblyDisplay.geometryOptions.setValues(
        datumPlanes=OFF)
##-------------------------------------------------------------
def joint_tie_constraints():
    jointasseme = jointmodel.rootAssembly
    p2 = jointasseme.instances['baseinstance']
    f = p2.faces
    xm = 0.0
    zm = web_thick/2.0 + adh_1_thick + (clip_thick + clip_radi +
clip_len)/2.0
    ym = base_thick
# first base adhesive (on left in isometric view) and base plate
    s1 = f.findAt((xm,ym,zm),)
    s11in = s1.index
    region1=regionToolset.Region(side1Faces=f[s11in:s11in+1])
    p3 = jointasseme.instances['adhinstance-2-1']
    f2 = p3.faces
    s2 = f2.findAt((xm,ym,zm),)
    s12in = s2.index
    region2=regionToolset.Region(side1Faces=f2[s12in:s12in+1])
    jointmodel.Tie(name='Constraint-1',master=region1, slave=region2,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
# second base adhesive (on right in isometric view) and base plate
    s3 = f.findAt((xm,ym,-zm),)
    s11in = s3.index
    region1=regionToolset.Region(side1Faces=f[s11in:s11in+1])
```

```
    p4 = jointasseme.instances['adhinstance-2-2']
    f3 = p4.faces
    s3 = f3.findAt((xm,ym,-zm),)
    s13in = s3.index
    region2=regionToolset.Region(side1Faces=f3[s13in:s13in+1])
    jointmodel.Tie(name='Constraint-2',master=region1, slave=region2,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
# first adhesive and second clip
    xs = 0.0
    ys = base_thick+adh_2_thick
    zs = zm
    p5 = jointasseme.instances['clipinstance-3-1']
    f5 = p5.faces
    s5 = f5.findAt((xs, ys, zs),)
    s15in = s5.index
    region1 = regionToolset.Region(side1Faces=f5[s15in:s15in+1])
    s6 = f2.findAt((xs, ys, zs),)
    s16in = s6.index
    region2 = regionToolset.Region(side1Faces=f2[s16in:s16in+1])
    jointmodel.Tie(name='Constraint-3',master=region1, slave=region2,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
# second adhesive and second clip
    p7 = jointasseme.instances['clipinstance-3-2']
    f7 = p7.faces
    s7 = f7.findAt((xs, ys, -zs),)
    s17in = s7.index
    region1 = regionToolset.Region(side1Faces=f7[s17in:s17in+1])
    s6 = f3.findAt((xs, ys, -zs),)
    s16in = s6.index
    region2 = regionToolset.Region(side1Faces=f3[s16in:s16in+1])
    jointmodel.Tie(name='Constraint-4',master=region1, slave=region2,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
# left clip and top left adhesive
    p5 = jointasseme.instances['clipinstance-1-1']
    f5 = p5.faces
    xsm = 0.0
    ysm = base_thick + adh_2_thick + (clip_radi + clip_thick +
clip_breadth)/2.0
    zsm = web_thick/2.0 + adh_1_thick
    s8 = f5.findAt((xsm, ysm, zsm),)
    s18in = s8.index
    region2 = regionToolset.Region(side1Faces=f5[s18in:s18in+1])
    p9 = jointasseme.instances['adhinstance-1-1']
    f9 = p9.faces
    s9 = f9.findAt((xsm, ysm, zsm),)
    s19in = s9.index
    region1 = regionToolset.Region(side1Faces=f9[s19in:s19in+1])
    jointmodel.Tie(name='Constraint-5',master=region2, slave=region1,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
# right clip and top right adhesive
    p7 = jointasseme.instances['clipinstance-1-2']
    f7 = p7.faces
    s10 = f7.findAt((xsm, ysm, -zsm),)
```

```
    s20in = s10.index
    region2 = regionToolset.Region(side1Faces=f7[s20in:s20in+1])
    p11 = jointasseme.instances['adhinstance-1-2']
    f11 = p11.faces
    s11 = f11.findAt((xsm, ysm, -zsm),)
    s21in = s11.index
    region1 = regionToolset.Region(side1Faces=f11[s21in:s21in+1])
    jointmodel.Tie(name='Constraint-6',master=region2, slave=region1,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
# top left adhesive and webplate
    xms = 0.0
    yms = base_thick + adh_2_thick + (clip_radi + clip_thick +
clip_breadth)/2.0
    zms = web_thick/2.0
    f12 = jointasseme.instances['webinstance'].faces
    s12 = f12.findAt((xms, yms, zms),)
    s22in = s12.index
    region1 = regionToolset.Region(side1Faces=f12[s22in:s22in+1])
    s13 = f9.findAt((xms, yms, zms),)
    s23in = s13.index
    region2 = regionToolset.Region(side1Faces=f9[s23in:s23in+1])
    jointmodel.Tie(name='Constraint-7',master=region1, slave=region2,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
# top right adhesive and webplate
    s14 = f12.findAt((xms, yms, -zms),)
    s24in = s14.index
    region1 = regionToolset.Region(side1Faces=f12[s24in:s24in+1])
    s15 = f11.findAt((xms, yms, -zms),)
    s25in = s15.index
    region2 = regionToolset.Region(side1Faces=f11[s25in:s25in+1])
    jointmodel.Tie(name='Constraint-8',master=region1, slave=region2,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
## aluminum layer and weplate
    xms = web_length/4.0
    yms = web_length/2.0
    zms = -web_thick/2.0
    f12 = jointasseme.instances['webinstance'].faces
    s12 = f12.findAt((xms, yms, zms),)
    s22in = s12.index
    region1 = regionToolset.Region(side1Faces=f12[s22in:s22in+1])
    f9a = jointasseme.instances['Lshapeinstance'].faces
    s13 = f9a.findAt((xms, yms, zms),)
    s23in = s13.index
    region2 = regionToolset.Region(side1Faces=f9a[s23in:s23in+1])
    jointmodel.Tie(name='Constraint-7a',master=region1, slave=region2,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
##-----------------------------------------------------------------------------
def joint_tie_constraint_2():
    jointasseme = jointmodel.rootAssembly
    p1 = jointasseme.instances['clipinstance-1-1']
    f1 = p1.faces

    p2 = jointasseme.instances['clipinstance-2-1']
```

```
    f2 = p2.faces

    p3 = jointasseme.instances['clipinstance-3-1']
    f3 = p3.faces
    xm1, ym1 = 0.0, base_thick+adh_2_thick+clip_thick_ch/2.0
    zm1 =  web_thick/2.0+adh_1_thick+clip_thick_ch+clip_radi_ch
    s2 = f2.findAt((xm1, ym1, zm1),)
    s2in = s2.index
    region1 = regionToolset.Region(side1Faces=f2[s2in:s2in+1])

    s3 = f3.findAt((xm1, ym1, zm1),)
    s3in = s3.index
    region2 = regionToolset.Region(side1Faces=f3[s3in:s3in+1])

    jointmodel.Tie(name='Constraint-12',master=region2, slave=region1,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
    xm2, ym2 = 0.0, base_thick+adh_2_thick+clip_radi_ch+clip_thick_ch
    zm2 = web_thick/2.0 + adh_1_thick+clip_thick_ch/2.0
    s2 = f2.findAt((xm2, ym2, zm2),)
    s2in = s2.index
    region1 = regionToolset.Region(side1Faces=f2[s2in:s2in+1])
    s3 = f1.findAt((xm2, ym2, zm2),)
    s3in = s3.index
    region2 = regionToolset.Region(side1Faces=f1[s3in:s3in+1])
    jointmodel.Tie(name='Constraint-14',master=region2,    slave=region1,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
    p1 = jointasseme.instances['clipinstance-1-2']
    f1 = p1.faces
    p2 = jointasseme.instances['clipinstance-2-2']
    f2 = p2.faces
    p3 = jointasseme.instances['clipinstance-3-2']
    f3 = p3.faces
    xm1, ym1 = 0.0, base_thick+adh_2_thick+clip_thick/2.0
    zm1 =  web_thick/2.0+adh_1_thick+clip_thick+clip_radi
    s2 = f2.findAt((xm1, ym1, -zm1),)
    s2in = s2.index
    region1 = regionToolset.Region(side1Faces=f2[s2in:s2in+1])
    s3 = f3.findAt((xm1, ym1, -zm1),)
    s3in = s3.index
    region2 = regionToolset.Region(side1Faces=f3[s3in:s3in+1])
    jointmodel.Tie(name='Constraint-15',master=region2, slave=region1,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
    xm2, ym2 = 0.0, base_thick+adh_2_thick+clip_radi+clip_thick
    zm2 = web_thick/2.0 + adh_1_thick+clip_thick/2.0
    s2 = f2.findAt((xm2, ym2, -zm2),)
    s2in = s2.index
    region1 = regionToolset.Region(side1Faces=f2[s2in:s2in+1])
    s3 = f1.findAt((xm2, ym2, -zm2),)
    s3in = s3.index
    region2 = regionToolset.Region(side1Faces=f1[s3in:s3in+1])
    jointmodel.Tie(name='Constraint-16',master=region2, slave=region1,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
#-----------------------------------------------------------------------
```

```
def defect_constraints():
    jointasseme = jointmodel.rootAssembly
    p2 = jointasseme.instances['defect-1-1']
    c,f = p2.cells, p2.faces
    z = (web_thick/2.0)
    y = b1 + base_thick + clip_thick - Lsb/2.0
    s1 = f.findAt((0.0,y,z),)
    s11in = s1.index
    side = f[s11in:s11in+1]
    region2 = regionToolset.Region(side1Faces = side)
    p3 = jointasseme.instances['webinstance']
    c3,f3 = p3.cells, p3.faces
    s2 = f3.findAt((0.0,y,z),)
    s21in = s2.index
    side = f3[s21in:s21in+1]
    region1 = regionToolset.Region(side1Faces = side)
    jointmodel.Tie(name='Constraint-22',master=region1, slave=region2,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
    p2 = jointasseme.instances['defect-1-2']
    c,f = p2.cells, p2.faces
    z = (web_thick/2.0)
    y = b1 + base_thick + clip_thick - Lsb/2.0
    s1 = f.findAt((2*defect_2a,y,z),)
    s11in = s1.index
    side = f[s11in:s11in+1]
    region2 = regionToolset.Region(side1Faces = side)
    p3 = jointasseme.instances['webinstance']
    c3,f3 = p3.cells, p3.faces
    s2 = f3.findAt((2*defect_2a,y,z),)
    s21in = s2.index
    side = f3[s21in:s21in+1]
    region1 = regionToolset.Region(side1Faces = side)
    jointmodel.Tie(name='Constraint-23',master=region1, slave=region2,
                   positionToleranceMethod=COMPUTED, adjust=ON,
                   tieRotations=ON, thickness=ON)
##-----------------------------------------------------------------------------
def joint_step():
    jointmodel.StaticStep(name='Step-1', previous='Initial')
##-----------------------------------------------------------------------------
def joint_load():
    jointasseme = jointmodel.rootAssembly
    region =(jointasseme.instances['webinstance'].cells,
             jointasseme.instances['clipinstance-1-1'].cells,
             jointasseme.instances['clipinstance-2-1'].cells,
             jointasseme.instances['clipinstance-3-1'].cells,
             jointasseme.instances['clipinstance-1-2'].cells,
             jointasseme.instances['clipinstance-2-2'].cells,
             jointasseme.instances['clipinstance-3-2'].cells,
             jointasseme.instances['baseinstance'].cells,
             jointasseme.instances['adhinstance-1-1'].cells,
             jointasseme.instances['adhinstance-1-2'].cells,
             jointasseme.instances['adhinstance-2-1'].cells,
             jointasseme.instances['adhinstance-2-2'].cells,
             jointasseme.instances['Lshapeinstance'].cells,
             jointasseme.instances['defect-1-1'].cells,
             jointasseme.instances['defect-1-2'].cells,)
```

```
    mdb.models['Joint Model'].Temperature(name='Field-1',
createStepName='Step-1',
    region=region, crossSectionDistribution=CONSTANT_THROUGH_THICKNESS,
magnitudes=(theta_temp, ))
#-------------------------------------------------------------
##-------------------------------------------------------------
def joint_job():
    global statuss
    jointjob = mdb.Job(name='Joint_Job',model=jointmodel)
#-------------------------------------------------------------
from abaqus import *
from abaqusConstants import *
from section import*
import part
import regionToolset
import displayGroupMdbToolset as dgm
import material
import section
import sketch
import assembly
import step
import interaction
import load
import mesh
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import job
import math

base_part()
base_mater()
joint_section()
joint_sweeppaths()
clip_partition()  ## if the parts are partitioned, the orientation is lost
for some reason.
joint_instance()
joint_tie_constraints()
joint_tie_constraint_2()
defect_constraints()
joint_step()
joint_load()
joint_job()

print 'NOTE THAT THE FOLLOWING NEED TO BE DONE MANUALLY'
print '1. ASSIGNING CONTINUUM SHELLS FOR THE PARTS IN THE MESH MODULE'
print '2. ASSIGNING SWEEP DIRECTIONS FOR THE ADHESIVES, DEFECTs, AND Ti
LAYER'
print '2. MESHING THE INDIVIDUAL PARTS'
print '3. BOUNDARY CONDITIONS FOR THE MODEL'
```

APPENDIX D

DEGRADATION MODEL REVIEW REFERENCES

This appendix contains the references cited by Akula and Garnich (2007a,b) in their review of the literature on various degradation models developed for laminates of unidirectional polymer composite materials. The review was restricted to time independent material behavior and models associated with FE implementation for progressive failure analysis.

[D.1]. Sendeckyj GP. A brief survey of empirical multi-axial strength criteria for composites. Composite Materials: Testing and Design (Second Conference), ASTM STP 497, American Society for Testing and Materials, 1972. p. 41-51.

[D.2]. Wu EM. Phenomenological anisotropic failure criterion. In: Sendeckyj GP, editor. Mechanics of Composite Materials, composite materials series, Vol. 2, 1974. p. 353-431.

[D.3]. Soni SR. A new look at commonly used failure theories in composite laminates. Proceedings of the 24th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, Lake Tahoe, Nevada, USA, May 2-4, 1983. p. 171-175.

[D.4]. Soni SR. A comparative study of failure envelopes in composite laminates. J Reinf Plast Comp 1983;2(1):34-42.

[D.5]. Tsai SW. A survey of macroscopic failure criteria for composite materials. J Reinf Plast Comp 1984;3(1):40-62.

[D.6]. Rowlands RE. Strength (failure) theories and their experimental correlation. In: Sih GC, Skudra AM, editors. Failure mechanics of composites, handbook of composites, 1985. p. 71-125.

[D.7]. Nahas MN. Survey of failure and post-failure theories of laminated fiber-reinforced composites. J Compos Tech Res 1986;8(4):138-153.

[D.8]. Fan W. On phenomenological anisotropic failure criteria. Compos Sci Technol 1987;28(4):269-278.

[D.9]. Labossiere P, Neale KW. Macroscopic failure criteria for fibre-reinforced composite materials. Solid Mech Arch 1987;12(2):439-450.

[D.10]. Thom H. A review of the biaxial strength of fibre-reinforced plastics. Compos Part A-Appl S 1998;29(8): 869-886.

[D.11]. Federico P. A study of failure criteria of fibrous composite materials. NASA/CR-2001-210661.

[D.12]. Soden PD, Hinton MJ, Kaddour AS. A comparison of predictive capabilities of current failure theories for composite laminates. Compos Sci Technol 1998;58(7):1225-1254.

[D.13]. Hinton MJ, Kaddour AS, Soden PD. A comparison of the predictive capabilities of current failure theories for composite laminates, judged against experimental evidence. Compos Sci Technol 2002;62(12-13):1725-1797.

[D.14]. Kaddour AS, Hinton MJ, Soden PD. A comparison of the predictive capabilities of current failure theories for composite laminates: Additional contributions. Compos Sci Technol 2004;64(3-4):449-476.

[D.15]. Hinton MJ, Kaddour AS, Soden PD. A further assessment of the predictive capabilities of current failure theories for composite laminates: comparison with experimental evidence. Compos Sci Technol 2004;64(3-4):549-588.

[D.16]. Icardi U, Locatto S, Longo A. Assesment of recent theories for predicting failures of composite laminates. Appl Mech Rev 2007;60(2):76-86.

[D.17]. Hill R. *Mathematical Theory of Plasticity*. Oxford, Clarendon Press, New York, 1950.

[D.18]. Azzi VD, Tsai SW. Anisotropic strength of composites. Exp Mech 1965;5(9):283-288.

[D.19]. Hoffman O. The brittle strength of orthotropic materials. J Compos Mater 1967;1(2):200-206.

[D.20]. Chamis CC. Failure criteria for filamentary composites. Composite Materials: Testing and Design, ASTM STP 460, American Society for Testing and Materials, 1969. p. 336-351.

[D.21]. Tsai SW, Wu EM. A general theory of strength for anisotropic materials. J Compos Mater 1971;5(1):58-80.

[D.22]. Sandhu RS. Nonlinear behavior of unidirectional and angle ply laminates. J Aircraft 1974;13(2):104-111.

[D.23]. Hashin Z, Rotem A. A fatigue failure criterion for fiber reinforced materials. J Compos Mater 1973;7(4):448-464.

[D.24]. Hashin Z. Failure criteria for unidirectional fiber composites. J Appl Mech 1980;47(2):329-334.

[D.25]. Lee JD. Three-dimensional finite element analysis of damage accumulation in composite laminates. In Fracture of Composite Materials, Proceedings of the 2nd USA-USSR Symposium, Bethlehem, Pennsylvania, USA, 1982, pp. 291-306.

[D.26]. Christensen RM. Stress based yield/failure criteria for fiber composites. Int J Solids Struct 1997;34(5):529-543.

[D.27]. Mayes JS, Hansen AC. A multicontinuum failure analysis of composite structural laminates. Mech Compos Mater Struct 2001;8(4):249-262.

[D.28]. Hahn HT, Williams JG. Compression failure mechanisms in unidirectional composites. Composite Materials: Testing and Design, ASTM STP 893, American Society for Testing and Materials, Philadelphia, Pennsylvania, USA, 1986. p. 115-139.

[D.29]. Eason TG, Ochoa OO. Modeling progressive damage in composites: A shear deformable element for ABAQUS. Compos Struct 1996;34(2):119-128.

[D.30]. Ye L. Role of matrix resin in delamination onset and growth in composite laminates. Compos Sci Technol 1988;33(4):257-277.

[D.31]. Yamada SE, Sun CT. Analysis of laminate strength and its distribution. J Compos Mater 1978;12(3):275-284.

[D.32]. Chang FK, Scott RA, Springer GS. Failure strength of nonlinearly elastic composite laminates containing a pin loaded hole. J Compos Mater 1984;18(5):464-477.

[D.33]. Lessard LB. Bearing failure of composite pinned joints using progressive damage modeling. Proceedings of the 8th International Conference on Composite Materials, Honolulu, Hawaii, USA, July 15-19, 1991. p. 9.E.1-9.E.10.

[D.34]. Griffin OH Jr. Evaluation of finite-element software packages for stress analysis of laminated composites. Compos Technol Rev 1982;4(4):136-141.

[D.35]. Saarela O. Computer programs for mechanical analysis and design of polymer matrix components. Prog Polym Science (Oxford) 1994;19(1):171-301.

[D.36]. ABAQUS Version 6.6, ABAQUS Inc. Rhode Island, USA, 2006. .

[D.37]. Ochoa OO, Reddy JN. *Finite Element Analysis of Composite Laminates*, Kluwer Academic Publishers, 1992.

[D.38]. Robbins DH Jr, Reddy JN. Modelling of thick composites using a layer-wise laminate theory. Int J Numer Meth Eng 1993;36(4):655-677.

[D.39]. Reddy JN. On refined computational models of composite laminates. Int J Numer Meth Eng 1989;27(2):361-382.

[D.40]. Reddy JN. On the generalization of displacement-based laminate theories. Appl Mech Rev 1989;42(11): S213-S222.

[D.41]. Hyer MW. *Stress Analysis of Fiber-Reinforced Composite Materials*. New York: McGraw-Hill, 1998.

[D.42]. Herakovich CT. *Mechanics of Fibrous Composites.* New York: John Wiley & Sons, 1998.

[D.43]. Jones RM. *Mechanics of Composite Materials*. Second Edition, Taylor and Francis, 1999.

[D.44]. Chang FK, Chang YK. A progressive damage model for laminated composites containing stress concentrations. J Compos Mater 1987;21(9):834-855.

[D.45]. Chou PC, Carleone J, Hsu CM. Elastic constants of layered media. J Compos Mater 1972;6(1):80-93.

[D.46]. Sun CT, Li S. Three-dimensional effective elastic constants for thick laminates. J Compos Mater 1988;22(7):629-639.

[D.47]. Al-Khalil MFS, Soden PD. Theoretical through-thickness elastic constants for filament-wound tubes. Int J Mech Sci 1994;36(1):49-62.

[D.48]. Lempriere BM. Poisson's ratio in orthotropic materials. AIAA J 1968;6(11):2226-2227.

[D.49]. Reddy YS, Reddy JN. Three-dimensional finite element progressive failure analysis of composite laminates under axial extension. J Compos Tech Res 1993;15(2):73-87.

[D.50]. Vinson JR, Sierakowski RL. *The Behavior of structures composed of composite materials*, Mechanics of structural systems, 5, Baker & Taylor, 1986.

[D.51]. Chen WH, Lee SS. Numerical and experimental failure analysis of composite laminates with bolted joints under bending loads. J Compos Mater 1995;29(1):15-36.

[D.52]. Lee JD. Three dimensional finite element analysis of damage accumulation in composite laminate. Comput Struct 1982;15(3):335-350.

[D.53]. Lee JD. Three dimensional finite element analysis of layered fiber-reinforced composite materials. Comput Struct 1980;12(3):319-339.

[D.54]. Labossiere P, Neale KW, Neglo K. Damage accumulation in fibre-reinforced composite laminates. T Can Soc Mech Eng 1988;12(3):133-137.

[D.55]. Hwang WC, Sun CT. Failure analysis of laminated composites by using iterative three-dimensional finite element method. Comput Struct 1989;33(1):41-47.

[D.56]. Chang FK, Springer GS. The strengths of fiber reinforced composite bends. J Compos Mater 1986;20(1):30-45.

[D.57]. Nagesh. Finite-element analysis of pressure vessels with progressive degradation. Defence Sci J 2003;53(1):75-86.

[D.58]. Tsau LR, Plunkett R. Finite element analysis of progressive failure for laminated FRP plates with inplane loading. Eng Fract Mech 1993;45(4):529-546.

[D.59]. Tsau LR, Chang YH, Tsao FL. The design of optimal stacking sequence for laminated FRP plates with inplane loading. Comput Struct 1995;55(4):565-580.

[D.60]. Tsau LR, Liu CH. A comparison between two optimization methods on the stacking sequence of fiber-reinforced composite laminate. Comput Struct 1995;55(3):515-525.

[D.61]. Tsai SW. *Composites design*, Think Composites, Dayton, Ohio, 1986.

[D.62]. Hu FZ, Soutis C, Edge EC. Interlaminar stresses in composite laminates with a circular hole. Compos Struct 1997;37(2):223-232.

[D.63]. Pachajoa ME, Frances MK, Lee JD. Stress and failure analysis of composite structures. Eng Fract Mech 1995;50(5-6):883-902.

[D.64]. McCarthy CT, McCarthy MA, Lawlor VP. Progressive damage analysis of multi-bolt composite joints with variable bolt-hole clearances. Compos Part B-Eng 2005;36(4):290-305.

[D.65]. Camanho PP, Matthews FL. A progressive damage model for mechanically fastened joints in composite laminates. J Compos Mater 1999;33(24):2248-2280.

[D.66]. Nuismer RJ, Tan SC. Constitutive relations of a cracked composite lamina. J Compos Mater 1988;22(4):306-321.

[D.67]. Tan SC, Nuismer RJ. A theory for progressive matrix cracking in composite laminates. J Compos Mater 1989;23(10):1029-1047.

[D.68]. O'Higgins RM, Padhi GS, McCarthy MA, McCarthy CT. Experimental and Numerical Study of the Open-Hole Tensile Strength of Carbon/Epoxy Composites. Mech Compos Mater 2004;40(4):269-278.

[D.69]. Apalak ZG, Apalak MK, Genc MS. Progressive damage modeling of an adhesively bonded composite single lap joint under flexural loads at the mesoscale level. J Reinf Plast Comp 2007;26(9):903-953.

[D.70]. Kermanidis Th, Labeas G, Tserpes KI, Pantelakis Sp. Finite element modeling of damage accumulation in bolted composite joints under incremental tensile loading. Proceedings of the 3rd ECCOMAS Congress, Barcelona, Spain, 11-14 September, 2000.

[D.71]. Tserpes KI, Papanikos P, Kermanidis Th. A three-dimensional progressive damage model for bolted joints in composite laminates subjected to tensile loading. Fatigue Fract Eng M 2001;24(10):663-675.

[D.72]. Shokrieh MM, Lessard LB, Poon C. Three-dimensional progressive failure analysis of pin/bolt loaded composite laminates. 83rd meeting of the AGARD SMP on bolted joints in polymeric composites, Florence, Italy, 2-3 September, 1996. p. 7.1-7.10.

[D.73]. Tserpes KI, Labeas G, Papanikos P, Kermanidis Th. Strength prediction of bolted joints in graphite/epoxy composite laminates, Compos Part B-Eng. 2002;33(7):521-529.

[D.74]. Ireman T, Ranvik T, Eriksson I. On damage development in mechanically fastened composite laminates. Compos Struct 2000;49(2):151-171

[D.75]. Mayes JS. Multicontinuum failure analysis of composite structural laminates. PhD Dissertation, University of Wyoming, Department of Mechanical Engineering, 1999.

[D.76]. Mayes JS, Hansen AC. Composite laminate failure analysis using multicontinuum theory. Compos Sci Technol 2004;64(3-4):379-394.

[D.77]. Mayes JS, Hansen AC. A comparison of multicontinuum theory based failure simulation with experimental results. Compos Sci Technol 2004;64(3-4):517-527.

[D.78]. Waszczak JP, Cruse TA. Failure mode and strength predictions of anisotropic bolt bearing specimens. J Compos Mater 1971;5(3):421-425.

[D.79]. Sciuva MD, Icardi U, Villani M. Failure analysis of composite laminates under large deflection. Compos Struct 1998;40(3-4):239-255.

[D.80]. Pal P, Ray C. Progressive failure analysis of laminated composite plates by finite element method. J Reinf Plast Comp 2002;21(16):1505-1513.

[D.81]. Prusty BG. Progressive failure analysis of laminated unstiffened and stiffened composite panels. J Reinf Plast Comp 2005;24(6):633-642.

[D.82]. Lee YJ, Chen WH. Failure process and bolted joint strength of composite laminates. J Chin Soc Mech Eng Trans Chin Inst Eng C 1988;9(3):169-182.

[D.83]. Chen WH, Lee YJ. The effects of clearance and pin elasticity on the bearing strength of composite laminates. J Chin Soc Mech Eng Trans Chin Inst Eng C 1990;11(2):147-157.

[D.84]. Kim ZG, Hong CS, Kim CG. Postbuckling analysis of stringer-stiffened composite laminated cylindrical panels. J Reinf Plast Comp 1995;14(8):827-846.

[D.85]. Kong CW, Lee IC, Kim CG, Hong CS. Postbuckling and failure of stiffened composite panels under axial compression. Compos Struct 1998;42(1):13-21.

[D.86]. Kong CW, Hong CS, Kim CG. Postbuckling strength of composite plate with a hole. J Reinf Plast Comp 2001;20(6):466-481.

[D.87]. Hyer MW, Wolford GF, Knight NF Jr. Damage initiation and progression in internally pressurized noncircular composite cylinders. Proceedings of the 44th AIAA./ASME/ASCE/AHS

Structures, Structural Dynamics, and Materials Conference, Norfolk, Virginia, USA, April 7-10, 2003.

[D.88]. Chen JK, Sun CT, Chang CI. Failure analysis of a graphite/epoxy laminate subjected to combined thermal and mechanical loading. J Compos Mater 1985;19(5):408-423.

[D.89]. Gummadi LNB, Palazotto AN. Progressive failure analysis of composite cylindrical shells considering large rotations. Compos Part B-Eng 1998;29(5):547-563.

[D.90]. Kweon JH, Hong CS, Lee IC. Postbuckling compressive strength of graphite/epoxy laminated cylindrical panels loaded in compression. AIAA J 1995;33(2):217-222.

[D.91]. Kweon JH. Post-failure analysis of composite cylindrical panels under compression. J Reinf Plast Comp 1998;17(18):1665-1681.

[D.92]. Kweon JH. Crippling analysis of composite stringers based on complete unloading method. Comput Struct 2002;80(27-30):2167-2175.

[D.93]. Lee IC, Kim CG, Hong CS. Buckling and postbuckling behavior of stiffened composite panels loaded in compression, AIAA J 1997;35(1):202-204.

[D.94]. Sun XK, Du SY, Wang GD. Bursting problem of filament wound composite pressure vessels. Int J Pres Ves Pip 1999;76(1):55-59.

[D.95]. Wang J, Callus PJ, Bannister MK. Experimental and numerical investigation of the tension and compression strength of un-notched and notched quasi-isotropic laminates. Compos Struct 2004;64(3/4):297-306.

[D.96]. Kam TY, Sher HF, Chao TN, Chang RR. Predictions of deflection and first-ply failure load of thin laminated composite plates via the finite element approach. Int J Solids Struct 1996;33(3):375-398.

[D.97]. Sandhu RS, Sendeckyj GP, Gallo RL. Modeling of the failure process in notched laminates. Proceedings of the IUTAM symposium, Blacksburg, Virginia, USA, 1983. p. 179-189.

[D.98]. Griffis CA, Nemes JA, Stonesifer FR, Chang CI. Degradation in strength of laminated composites subjected to intense heating and mechanical loading. J Compos Mater 1986;20(3):216-235.

[D.99]. Engelstad SP, Reddy JN, Knight NF Jr. Postbuckling response and failure prediction of graphite-epoxy plates loaded in compression. AIAA J 1992;30(8):2106-2113.

[D.100]. Singh SB, Kumar A. Postbuckling response and failure of symmetric laminates under in-plane shear. Compos Sci Technol 1998;58(12):1949-1960.

[D.101]. Singh SB, Kumar A. Postbuckling response and strength of laminates under combined in-plane loads. Compos Sci Technol 1999;59(5):727-736.

[D.102]. Ganesan R, Zhang D. Progressive failure analysis of composite laminates subjected to in-plane compressive and shear loadings. Sci Eng Compos Mater 2004;11(2-3):79-102.

[D.103]. Pandey AK, Reddy JN. A post first-ply failure analysis of composite laminates. Proceedings of the AIAA/ASME/ASCE/AHS/ASC 28th Structures, Structural Dynamics, and Material Conference, Monterey, Canada, 1987. p. 788-797.

[D.104]. Gotsis PK, Chamis CC, Minnetyan L. Prediction of composite laminate fracture: micromechanics and progressive fracture. Compos Sci Technol 1998;58(7):1137-1149.

[D.105]. Gotsis PK, Chamis CC, Minnetyan L. Application of progressive fracture analysis for predicting failure envelopes and stress-strain behaviors of composite laminates: a comparison with experimental results. Compos Sci Technol 2002;62(12-13):1545-1559.

[D.106]. Minnetyan L, Chamis CC, Murthy PLN. Structural durability of a composite pressure vessel. J Reinf Plast Comp 1992;11(11):1251-1269.

[D.107]. Gotsis PK, Chamis CC, Minnetyan L. Computational simulation of damage progression of composite thin shells subjected to mechanical loads. Theor Appl Fract Mech 1996;2(3)5: 211-224.

[D.108]. Gotsis PK, Chamis CC, Minnetyan L. Progressive fracture of fiber composite build-up structures. J Reinf Plast Comp 1997;16(2):182-198.

[D.109]. Minnetyan L, Chamis CC. Progressive fracture of composite cylindrical shells subjected to external pressure. J Compos Tech Res 1997;19(2):65-71.

[D.110]. Minnetyan L, Rivers JM, Chamis CC, Murthy PLN. Discontinuously stiffened composite panel under compressive loading. J Reinf Plast Comp 1995;14(1):85-98.

[D.111]. Chamis CC, Gotsis PK, Minnetyan L. Damage tolerance of composite pressurized shells. Proceedings of the 37th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference and Exhibit, Salt Lake City, Utah, USA, April 15-17, 1996. p. 2112-2121.

[D.112]. Chamis CC, Gotsis PK, Minnetyan L. Progressive fracture and damage tolerance of composite pressure vessels, J Adv Mater 1998;30(1):22-26.

[D.113]. Huang D, Minnetyan L. Damage progression in carbon-fiber reinforced plastic I-Beams. J Compos Constr 1998;2(1):38-45.

[D.114]. Huang D, Minnetyan L. Progressive fracture of stitched stiffened composite shear panels in the postbuckling range. J Reinf Plast Comp 2001;20(18):1617-1632.

[D.115]. Averill RC. A Micromechanics-based progressive failure model for laminated composite structures. Proceedings of the 33rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference and Exhibit, Washington, DC, 1992. p. 2898-2904.

[D.116]. Kim Y, Davalos JF, Barbero EJ. Progressive failure analysis of laminated composite beams. J Compos Mater 1996;30(5):536-560.

[D.117]. Davalos JF, Kim Y. A layer-wise formulation for progressive failure analysis of laminated composite beams. Proceedings of the Fourth Materials Engineering Conference, Washington, District of Colombia, USA, November 10-14, 1996. p. 1150-1159.

[D.118]. Greif R, Chapon E. Investigation of successive failure modes in graphite/epoxy laminated composite beams. J Reinf Plast Comp 1993;12(5):602-621.

[D.119]. Yang L. Reliability of composite laminates. Mech Struct Mach 1988;16(4):523-536.

[D.120]. Zhao H, Zhanjun G. Reliability analysis of composite laminates by enumerating significant failure modes. J Compos Mater 1995;14(5):427-444.

[D.121]. Thomas DJ, Wetherhold RC. Reliability analysis of composite laminates with load sharing. J Compos Mater 1991;25(11):1459-1475.

[D.122]. Max Yen SC, Chu TC, Teh KT, Wright MA, Stumpff P. Diagnosis of the subsequent failure mechanisms of composite laminates. Int J Offshore Polar 1994;4(3):248-256.

[D.123]. Kang TJ, Cho YJ, Youn JR, Chung K, Lee KW. Prediction of the ultimate strength of composite laminates under in-plane loading using a probabilistic approach. Polym Polym Compos 2001;9(5):339-343.

[D.124]. Tsai SW, Azzi VD. Strength of laminated composite materials. AIAA J 1966;4(2):296-301.

[D.125]. Chiu KD. Ultimate strengths of laminated composites. J Compos Mater 1969;3(3):578-582.

[D.126]. Chou SC, Orringer O, Rainey JH. Post-failure behavior of laminates –I. No stress concentrations. J Compos Mater 1976;10(4):371-381.

[D.127]. Eckold GC, Leadbetter D, Soden PD, Griggs PR. Lamination theory in prediction of failure envelopes for filament wound materials subjected to biaxial loading. Composites 1978;9(4):243-246.

[D.128]. Kaddour AS, Soden PD, Hinton MJ. Failure of ±55 degree filament wound glass/epoxy composite tubes under biaxial compression. J Compos Mater. 1998;32(18):1618-1645.

[D.129]. Hinton MJ, Soden PD, Kaddour AS. Strength of composite laminates under biaxial loads. Appl Compos Mater 1996;3(3):151-162.

[D.130]. Schneider W, Bardenheier R. Versagenskriterien fur Kunstsoffe, Teil II: Experimentelle Ergebnisse. J Mater Technol 1975;6:339-348.

[D.131]. Huang ZM. A bridging model prediction of the ultimate strength of composite laminates subjected to biaxial loads. Compos Sci Technol 2004;64(3-4):395-448.

[D.132]. Huang ZM. Correlation of the bridging model predictions of the biaxial failure strengths of fibrous laminates with experiments, Compos Sci Technol 2004;64(3-4):529-548.

[D.133]. Huang ZM. A unified micromechanical model for the mechanical properties of two constituent composite materials. part V: laminate strength, J Thermoplast Compos 2000;13(3):190-206.

[D.134]. Huang ZM. Ultimate strength of a composite cylinder subjected to three-point bending: Correlation of beam theory with experiment. Compos Struct 2004;63(3-4):439-445.

[D.135]. Huang ZM. Flexural strength of a composite cylinder incorporated with thermal residual stresses. Mater Sci Eng 2004;A366(2):367-380.

[D.136]. Hinton MJ. Predicting failure in composite laminates: the background to the exercise. Compos Sci Technol 1998;58(7):1001-1010.

[D.137]. Soden PD, Hinton MJ, Kaddour AS. Lamina properties, lay-up configurations and loading conditions for a range of fibre-reinforced composite laminates. Compos Sci Technol 1998;58(7): 1011-1022.

[D.138]. Hinton MJ, Kaddour AS, Soden PD. Evaluation of failure prediction in composite laminates: background to 'part B' of the exercise. Compos Sci Technol 2002;62(12-13):1481-1488.

[D.139]. Soden PD, Hinton MJ, Kaddour AS. Biaxial test results for strength and deformation for a range of e-glass and fibre reinforced composite laminates: failure exercise benchmark data. Compos Sci Technol 2002;62(12-13):1489-1514.

[D.140]. Hinton MJ, Kaddour AS, Soden PD. Evaluation of failure prediction in composite laminates: background to 'part C' of the exercise. Compos Sci Technol 2004;64(3-4):321-327.

[D.141]. Soden PD, Kaddour AS, Hinton MJ. Recommendations for designers and researchers resulting from the world-wide failure exercise. Compos Sci Technol 2004;64(3-4):589-604.

[D.142]. Harik VM. Optimization of structural designs for a safe failure pattern: layered material systems. Mater Des 2001;22(4):317-324.

[D.143]. Bogetti TA, Hoppel CPR, Harik VM, Newill JF, Burns BP. Predicting the nonlinear response and progressive failure of composite laminates. Compos Sci Technol 2004;64(3-4):329-342.

[D.144]. Bogetti TA, Christopher PRH, Harik VM, Newill JF, Burns PB. Predicting the nonlinear response and failure of composite laminates: Correlation with experimental evidence. Compos Sci Technol 2004;64(3-4):477-485.

[D.145]. Liu K, Tsai SW. A progressive quadratic failure criterion for a laminate. Compos Sci Technol 1998;58(7):1023-1032.

[D.146]. Kuraishi A, Tsai SW, Liu KS. A progressive quadratic failure criterion, part B. Compos Sci Technol 2002;62(12-13):1683-1695.

[D.147]. Wolfe WE, Butalia TS. A strain-energy based failure criterion for non-linear analysis of composite laminates subjected to biaxial loading. Compos Sci Technol 1998;58(7):1107-1124.

[D.148]. Liu X, Mahadevan S. Ultimate strength failure probability estimation of composite structures. J Reinf Plast Comp 2000;19(5):403-426.

[D.149]. Mahadevan S, Liu X. Probabilistic analysis of composite structure ultimate strength. AIAA J 2002;40(7):1408-1414.

[D.150]. Sluimer GM. Ultimate performance of GRP-laminates under in-plane biaxial loading. Heron 1998;43(4):199-225.

[D.151]. Puck A. Zum deformationsverhalten und bruchmechanismus von glasfaser/kunstoff. Kunstoffe Bd 1965;55:913

[D.152]. Tan SC. A progressive failure model for composite laminates containing openings. J Compos Mater 1991;25(5):556-577.

[D.153]. Tan SC, Perez J. Progressive failure of laminated composites with a hole under compressive loading. J Reinf Plast Comp 1993;12(10):1043-1057.

[D.154]. Kwon YW, Berner JM. Micromechanics model for damage and failure analysis of laminated fibrous composites. Eng Fract Mech 1995;52(2):231-242.

[D.155]. Davila CG, Ambur DR, McGowan DM. Analytical prediction of damage growth in notched composite panels loaded in compression. J Aircraft 2000;37(5):898-905.

[D.156]. Ambur DR, Jaunky N, Hilburger MW. Progressive failure studies of stiffened panels subjected to shear loading. Compos Struct 2004;65(2):129-142.

[D.157]. Ambur DR, Jaunky N, Hillburger MW, Davila CG. Progressive failure analyses of compression-loaded composite curved panels with and without cutouts. Compos Struct 2004;65(2):143-155.

[D.158]. Xiao Y, Ishikawa T. Bearing failure in bolted composite joints: analytical tools development. Adv Compos Mater 2003;11(4):375-391.

[D.159]. Xiao Y, Ishikawa T. Bearing strength and failure behavior of bolted composite joints, part II: modeling and simulation. Compos Sci Technol 2005;65(7-8): 1032-1043.

[D.160]. Kim SJ, Hwang JS, Kim JH. Progressive failure analysis of pin-loaded laminated composites using penalty finite element method. AIAA J 1998;36(1):75-80.

[D.161]. Feih S, Shercliff HR. Composite failure prediction of single-L joint under bending. Compos Part A-Appl S, 2005;36(3):381-395.

[D.162]. Feih S, Shercliff HR. Quality assessment of curved composite components in peel joint structures. Compos Part A-Appl S 2005;36(3):397-408.

[D.163]. Chang FK, Lessard LB, Tang JM. Compression response of laminated composites containing an open hole. Sampe Quart 1988;19(4):46-51.

[D.164]. Chang FK, Lessard LB. Damage tolerance of laminated composites containing an open hole and subjected to compressive loadings: part I – analysis. J Compos Mater 1991;25(1):2-43.

[D.165]. Xie D, Biggers SB Jr. Postbuckling analysis with progressive damage modeling in tailored laminated plates and shells with a cutout. Compos Struct 2003;59(2):199-216.

[D.166]. Lessard LB, Shokrieh MM. Two-dimensional modeling of composite pinned-joint failure. J Compos Mater 1995;29(5):671-697.

[D.167]. Zarco-Gonzalez JC, Fellows NA, Durodola JF. A step-size independent method for finite element modeling of damage in composites. Compos Sci Techol 2004;64(10-11):1679-1689.

[D.168]. Okutan B. The effects of geometric parameters on the failure strength for pin-loaded multi-directional fiber-glass reinforced epoxy laminate. Compos Part B-Eng 2002;33(8):567-578.

[D.169]. Okutan B, Karakuzu R. The failure strength for pin-loaded multi-directional fiber-glass reinforced epoxy laminate. J Compos Mater 2002;36(24):2695-2712.

[D.170]. Okutan B, Karakuzu R. The strength of pinned joints in laminated composites. Compos Sci Technol 2003;63(6):893-905.

[D.171]. Kang TJ, Lee KW. Strength prediction for mechanical joints in laminated composite plate using progressive failure analysis. Polym Polym Compos 1997;5(5):343-351.

[D.172]. Lee KW, Kang TJ. Strength prediction of mechanical joints in laminated composite plates. Key Eng Mat 2000;183-187:1165-1170.

[D.173]. Dano ML, Gendron G, Picard A. Stress and failure analysis of mechanically fastened joints in composite laminates. Compos Struct 2000;50(3):287-296.

[D.174]. Padhi GS, Shenoi RA, Moy SSJ, Hawkins GL. Progressive failure and ultimate collapse of laminated composite plates in bending. Compos Struct 1998;40(3/4):277-291.

[D.175]. Kress G, Siau M, Ermanni P. Iterative solution methods for damage progression analysis. Compos Struct 2005;69(1):21-33.

[D.176]. Tolson S, Zabaras N. Finite element analysis of progressive failure in laminated composite plates. Comput Struct 1991;38(3):361-376.

[D.177]. Spottswood SM, Palazotto AN. Progressive failure analysis of a composite shell. Compos Struct 2001;53(1):117-131.

[D.178]. Ochoa OO, Engblom JJ. Analysis of progressive failure in composites. Compos Sci Technol 1987;28(2):87-102.

[D.179]. Engblom JJ, Ochoa OO. Finite element formulation including interlaminar stress calculations. Comput Struct 1986;23(2):241-249.

[D.180]. Greszczuk LB. Microbuckling failure of circular fiber-reinforced composites. AIAA J 1975;13(10):1311-1318.

[D.181]. Singh SB, Kumar A, Iyengar NGR. Progressive failure of symmetrically laminated plates under uni-axial compression. Struct Eng Mech 1997;5(4):433-450.

[D.182]. Singh SB, Kumar A, Iyengar NGR. Progressive failure of symmetric laminates under in-plane shear: I-Positive shear. Struct Eng Mech 1998;6(2):143-159.

[D.183]. Singh SB, Kumar A, Iyengar NGR. Progressive failure of symmetric laminates under in-plane shear: II-Negative Shear. Struct Eng Mech 1998;6(7):757-772.

[D.184]. Sleight DW. Progressive failure analysis methodology for laminated composite structures, NASA/TP–1999–209107.

[D.185]. Sleight DW, Knight NF Jr, Wang JT. Evaluation of a progressive damage methodology for laminated composite structures. Proceedings of the 38th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Kissimmee, Florida, USA, April 7-10, 1997. p.2257-2272.

[D.186]. Sleight DW, Lotts CG. Application of interface technology in progressive failure analysis of composite panels. Proceedings of the 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Denver, Colorado, USA, April 22-25, 2002.

[D.187]. Wang JT, Lotts CG, Sleight DW. Analysis of discrete-source damage progression in a tensile stiffened composite panel. Proceedings of the 40th AIAA/ASME/ASCE/AHS/ASC

Structures, Structural Dynamics, and Materials Conference, St. Louis, Missouri, USA, April 12-15, 1999.

[D.188]. Hyer MW, Wolford GF. Microcrack progression in internally pressurized elliptical composite cylinders. Proceedings of the 17th Annual Technical Conference of the American Society for Composites, Boca Raton, Florida, USA, 2002.

[D.189]. Hyer MW, Wolford GF. Progressive failure analysis of internally pressurized noncircular composite cylinders. Proceedings of the 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Denver, Colorado, USA, April 22-25, 2002. p. 1522-1532.

[D.190]. Knight NF Jr, Rankin CC, Brogan FA. STAGS computational procedure for progressive failure analysis of laminated composite structures, Int J Nonlinear Mech 2002;37(4-5):833-849.

[D.191]. Mahadevan S, Liu X, Xiao Q. A probabilistic progressive failure model of composite laminates. J Reinf Plast Comp 1997;16(11):1020-1038.

[D.192]. Liu X, Mahadevan S. System reliability of composite laminates. Proceedings of the 38th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference, Kissimmee, Florida, USA, 1997. p. 2877-2884.

[D.193]. Chen J, Wang X, Luo C. Reliability analysis of FRP laminates plates with consideration of both initial imperfection and failure sequence. Acta Mech Solida Sin 2002;15(3):227-235.

[D.194]. Mazzeranghi A, Vanghi D. A post-failure model for composite laminates based on phenomenologic aspects of damage. Compos Struct 1994;28:323-332.

[D.195]. Reddy YSN, Moorthy CMD, Reddy JN. Non-linear progressive failure analysis of laminated composite plates. Int J Non-Linear Mech 1995;30(5):629-649.

[D.196]. Chang FK, Kutlu Z. Strength and response of cylindrical composite shells subjected to out-of-plane loadings. J Compos Mater 1989;23(1):11-31.

[D.197]. Chang FK, Kutlu Z. Study on the crushing response of cylindrical composite shells subjected to transverse loading. J Compos Technol Res 1990;12(4):239-244.

[D.198]. Tsai SW, Hahn HT. *Introduction to Composite Materials*. Technomic Publishing Co., Inc. Westport, CT, 1980.

[D.199]. Chang FK, Chang KY. Post-failure analysis of bolted composite joints in tension or shear-out mode failure. J Compos Mater 1987;21(9):809-833.

[D.200]. Joo SG, Hong CS, Kim CG. Free edge effect on the post failure behavior of composite laminates under tensile loading. J Reinf Plast Compos 2001;20(3):191-221.

[D.201]. Joo SG, Hong CS. Progressive failure analysis of composite laminates using 3-D finite element method. Key Eng Mater 2000;183-187:535-540.

[D.202]. Sun HT, Chang FK, Qing X. The response of composite joints with bolt-clamping loads, part I: model development. J Compos Mater 2002;36(1):47-67.

[D.203]. Sun HT, Chang FK, Qing X. The response of composite joints with bolt-clamping loads, part II: model verification. J Compos Mater 2002;36(1):69-92.

[D.204]. Qing X, Chang FK, Starnes J. Damage tolerance of notched composite laminates with reinforcing strips. J Compos Mater 2003;37(2):111-128.

[D.205]. Qing X, Sun HT, Dagba L, Chang FK. Damage-tolerance-based design of bolted composite joints. In: Grant P, Rousseau CQ, editors. Composite Structures: Theory and Practice, American Society for Testing and Materials, West Conshocken, Pennsylvania, USA, 2000. p. 243-272.

[D.206]. Shahid I, Chang FK. An accumulative damage model for tensile and shear failures of laminated composite plates. J Compos Mater 1995;29(7):926-981.

[D.207]. Idelsohn S, Laschet G, Nyssen C. Pre- and post-degradation analysis of composite materials with different moduli in tension and compression. Comput Methods Appl Mech Eng 1982;30(2):133-149.

[D.208]. Sandhu RS, Gallo RL, Sendeckyj GP. Initiation and accumulation of damage in composite laminates. In: Daniel IM, editor. Composite Materials: Testing and Design, ASTM STP 787, American Society for Testing and Materials, 1982, p. 163-182.

[D.209]. Lin WP, Hu HT. Nonlinear analysis of fiber-reinforced composite laminates subjected to uniaxial tensile load. J Compos Mater 2002;36(12):1429-1450.

[D.210]. Lin WP, Hu HT. Parametric study on the failure of fiber-reinforced composite laminates under biaxial tensile load. J Compos Mater 2002;36(12):1481-1503.

[D.211]. Moas E, Griffin OH Jr. Progressive failure analysis of laminated composites structures. 38th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Kissimmee, Florida, USA, April 7-10, 1997. p. 2246-2256.

[D.212]. Petit PH, Waddoups ME. A method of predicting the nonlinear behavior of laminated composites. J Compos Mater 1969;3(1):2-19.

[D.213]. Grimes GC, Whitney JM. Degradation of graphite/epoxy composite materials because of load induced micromechanical damage. SAMPE Q 1974;5(4):1-13.

[D.214]. Craddock JN. Behavior of composite laminates after first-ply-failure. Compos Struct 1985;3(2):187-200.

[D.215]. Nahas MN. Yield and ultimate strengths of fibre composite laminates. Compos Struct 1986;6(4):283-294.

[D.216]. Hahn HT, Kiusalaas J, Burns BP, Bogetti TA. Constitutive modeling of composite laminates with progressive ply cracking. Proceedings of the 1992 ASME International Computers in Engineering Conference and Exposition, San Francisco, California, USA, August 2-6, 1992. p. 185-193.

[D.217]. Lee JW, Daniel IM. Progressive transverse cracking of crossply composite laminates. J Compos Mater 1990;24(11):1225-1243.

[D.218]. Zinoviev PA, Grigoriev SV, Lebedeva OV, Tairova LP. The strength of multilayered composites under a plane-stress state. Compos Sci Technol 1998;58(7):1209-1223.

[D.219]. Zinoviev PA, Lebedeva OV, Tairova LP. A coupled analysis of experimental and theoretical results on the deformation and failure of laminated composites under a state of plane stress. Compos Sci Technol 2002;62(12-13):1711-1723.

[D.220]. Kim YW, Hong CS. Progressive failure model for the analysis of laminated composites based on finite element approach. J Reinf Plast Compos 1992;11(10):1078-1092.

[D.221]. Lim SG, Hong CS. Prediction of transverse cracking and stiffness reduction in cross-ply laminated composites. J Compos Mater 1989;23(7):695-713.

[D.222]. Chang KY, Liu S, Chang FK. Damage tolerance of laminated composites containing an open hole and subjected to tensile loadings. J Compos Mater 1991;25(3):274-301.

[D.223]. Shahid I, Chang FK. Modeling of failure and response to laminated composites subjected to in-plane loads. Proceedings of the Workshop on Computational Methods for Failure Analysis and Life Prediction, Hampton, Virginia, USA, October 14-15, 1992. p. 83-120.

[D.224]. Shahid I, Sun HT, Chang FK. Predicting scaling effect on the notched strength of prepreg and fiber tow-placed laminated composites. J Compos Mater 1995;29(8):1063-1095.

[D.225]. Hung CL, Chang FK. Bearing failure of bolted composite joints. Part II: Model and verification. J Compos Mater 1996;30(12):1359-1400.

[D.226]. Hung CL, Chang FK. Strength envelope of bolted composite joints under bypass loads. J Compos Mater 1996;30(13):1402-1435.

[D.227]. Kwon YW, Berner JM. Matrix damage of fibrous composites: effects of thermal residual stresses and layer sequences. Comput Struct 1997;64(1-4):375-382.

[D.228]. Kwon YW, Berner J. Analysis of matrix damage evolution in laminated composite plates. Eng Fract Mech 1994;48(6):811-817.

[D.229]. Swanson SR, Christoforou AP. Response of quasi-isotropic carbon/epoxy laminates to biaxial stress. J Compos Mater 1986;20(5):457-471.

[D.230]. Swanson SR, Christoforou AP. Progressive failure in carbon/epoxy laminates under biaxial stress. J Eng Mater Technol Trans ASME 1987;109(1):12-16.

[D.231]. Hwang TK, Hong CS, Kim CG. Probabilistic deformation and strength prediction for a filament wound pressure vessel. Compos Part B:Eng 2003;34(5):481-497.

[D.232]. Edge EC. Stress-based Grant-Sanders method for predicting failure of composite laminates. Compos Sci Technol 1998;58(7):1033-1041.

[D.233]. Edge EC. A comparison of theory and experiment for stress-based Grant-Sanders method. Compos Sci Technol 2002;62(12-13):1571-1589.

[D.234]. Rotem A. Prediction of laminate failure with Rotem failure criterion. Compos Sci Technol 1998;58(7):1083-1094.

[D.235]. Rotem A. The Rotem failure criterion: theory and practice. Compos Sci Technol 2002;62(12-13):1663-1671.

[D.236]. Sun CT, Tao J. Prediction of failure envelopes and stress/strain behavior of composite laminates. Compos Sci Technol 1998;58(7):1125-1136.

[D.237]. Sun CT, Tao J, Kaddour AS. Prediction of failure envelopes and stress-strain behavior of composite laminates: comparison with experimental results. Compos Sci Technol 2002;62(12-13):1673-1682.

[D.238]. Tao XJ, Sun CT. Effect of matrix cracking on stiffness of composite laminates. Mech Compos Mater Struct 1996;3(3):225-239.

[D.239]. Puck A, Schürmann H. Failure analysis of FRP laminates by means of physically based phenomenological models. Compos Sci Technol 1998;58(7):1045-1067.

[D.240]. Puck A, Schürmann H. Failure analysis of FRP laminates by means of physically based phenomenological models – part B. Compos Sci Technol 2002;62(12-13):1633-1662.

[D.241]. Cuntze RG, Freund A. The predictive capability of failure mode concept-based strength criteria for multidirectional laminates. Compos Sci Technol 2004;64(3-4):343-377.

[D.242]. Cuntze RG. The predictive capability of failure mode concept-based strength criteria for multi-directional laminates: part B. Compos Sci Technol 2004;64(3-4):487-516.

[D.243]. Knops M, Bogle C. Gradual failure in fibre/polymer laminates. Compos Sci Technol 2006;66(5):616-625.

[D.244]. Puck A. Festigkeitsberechnung and Glasfaser/Kunststoff-Laminaten bei zusannengesetzter beanspruchung: bruchhypothesen und schichtweise bruchanalyse, Kunststoffe, German Plastics 55, 1969, p. 18-9.

[D.245]. Hult J. Introduction and general overview. In: Krajcinovic D and Lemaitre J, editors. Continuum Damage Mechanics: Theory and Applications, Springer-Verlag, New York, 1987. p. 1-36.

[D.246]. Lemaitre J. *A Course on Damage Mechanics*. Second Edition, Springer-Verlag, Germany, 1996.

[D.247]. Voyiadjis GZ, Kattan PI. *Damage Mechanics*, Taylor & Francis, 2005.

[D.248]. Murakami S, Kamiya K. Constitutive and damage evolution equations of elastic-brittle materials based on irreversible thermodynamics. Int J Mech Sci 1997;39(4):473-486.

[D.249]. Germain P, Nguyen QS, Suquet P. Continuum thermodynamics. J Appl Mech-Trans ASME 1983;50(4B):1010-1020.

[D.250]. Lemaitre J, Chaboche JL. *Mechanics of Solid Materials*, Cambridge Press, 1990.

[D.251]. Lawrence ME. Introduction to the Mechanics of a Continuous Medium, Prentice-Hall, 1969.

[D.252]. Renard J, Favre JP, Jeggy Th. Influence of transverse cracking on ply behavior: Introduction of a characteristic damage variable. Compos Sci Technol 1993;46(1):29-37.

[D.253]. Thionnet A, Lesne P-M, Renard J. Using characteristic damage variables to simulate degradations and their evolutions in composite materials - comparison between experimental and numerical simulations. 14th Annual Energy-Sources Technology Conference and Exhibition, Houston, Texas, 1991, p. 105-111.

[D.254]. Renard J, Thionnet A. Meso-macro approach to predict the damage evolution of transverse cracking in laminated composite. Winter Annual Meeting of the American Society of Mechanical Engineers, Anaheim, California, USA, November 8-13, 1992. p. 31-39.

[D.255]. Thionnet A, Renard J. Meso-macro approach to transverse cracking in laminated composites using Talreja's model. Compos Eng 1993;3(9):851-871.

[D.256]. Renard J, Thionnet A. Damage in composites: From physical mechanisms to modelling. Compos Sci Technol 2006;66(5):642-646.

[D.257]. Nguyen BN. Three-dimensional modeling of damage in laminated composites containing a central hole. J Compos Mater 1997;31(17):1672-1693.

[D.258]. Nguyen BN. A three-dimensional modeling of transverse matrix cracking in laminated composites. Key Eng Mater 1997;127-131(2):1117-1126.

[D.259]. Nguyen BN. Damage modeling of laminated composites by the use of multilayer volume elements. Compos Sci and Tech 1998;58(6):891-905.

[D.260]. Laws N, Dvorak GJ, Hejazi M. Stiffness changes in unidirectional composites caused by crack systems. Mech Mater 1983;2(2):123-137.

[D.261]. Dvorak GJ, Laws N, Hejazi M. Analysis of progressive matrix cracking in composite laminates I. Thermoelastic properties of a ply with cracks. J Compos Mater 1985;19(3):216-234.

[D.262]. Phillips EA, Herakovich CT, Graham LL. Damage development in composites with large stress gradients. Compos Sci Technol 2001;61(15):2169-2182.

[D.263]. Allix O, Ladaveze P, Gilletta D, Ohayon R. A damage prediction method for composite structures. Int J Numer Methods Eng 1989;27(2):271-283.

[D.264]. Herakovich CT. *Mechanics of Fibrous Composites*. John Wiley and Sons Inc, New York, 1998.

[D.265]. Flesher ND, Herakovich CT. Predicting delamination in composite structures. Compos Sci Technol 2006;66(6):745-754.

[D.266]. Coats TW, Harris CE. A progressive damage methodology for residual strength predictions of notched composite panels. J Compos Mater 1999;33(23):2193-2224.

[D.267]. Coats TW, Harris CE. A Damage-Dependent Finite Element Analysis for Fiber-Reinforced Composite Laminates. In: International Conference on Computational Engineering Sciences, 1998.

[D.268]. Allen DH, Harris CE, Groves SE. A thermomechanical constitutive theory for elastic composites with distributed damage – I. Theoretical development. Int J Solids Struct 1987;23(9):1301-1318.

[D.269]. Allen DH, Harris CE, Groves SE. Thermo-mechanical constitutive theory for elastic composites with distributed damage – II. Application to matrix cracking in laminated composites. Int J Solids Struct 1987;23(9):1319-1338.

[D.270]. Allen DH, Harris CE, Groves SE. Damage modelling in laminated composites. Proceedings of the IUTAM/ICM Symposium on Yielding, Damage, and Failure of Anisotropic Solids, Villard-de-Lans, France, August 24-28, 1987. p. 535.

[D.271]. Tay TE, Lam KY, Cen Z. Analysis of composite structures with distributed and localized damage by the finite-element method. Compos Struct 1997;37(2):135-143.

[D.272]. Tay TE, Lim EH. Analysis of composite laminates with transverse cracks. Compos Struct 1996;34(4):419-426.

[D.273]. Tay TE, Lim EH, Tay AAO. Analysis of stresses in cross-ply laminates containing distributed transverse cracks. Finite Elem Anal Des 1994;18(1-3):301-308.

[D.274]. Lim EH, Tay TE. Stiffness loss of composite laminates with transverse cracks under mode I and mode III loading. Int J Damage Mech 1996;5(2):190-215.

[D.275]. Lim EH, Tay TE. Analysis of stiffness loss in cross-ply composite laminates. Compos Struct 1993;25(1-4):419-423.

[D.276]. Kennedy TC, Nahan MF. A simple nonlocal damage model for predicting failure of notched laminates. Compos Struct 1996;35(2):229-236.

[D.277]. Kennedy TC, Nahan MF. A simple nonlocal damage model for predicting failure in a composite shell containing a crack. Compos Struct 1997;39(1-2):85-91.

[D.278]. Nahan MF, Kennedy TC. A nonlocal damage theory for laminated plate with application to aircraft damage tolerance. J Reinf Plast Compos 2000;19(5):354-363.

[D.279]. Engblom JJ, Yang Q, Abdul-samad N, Havelka J. Residual property predictions for laminated composite structures: FE based internal state variable approach. Proceedings of the 33rd Structures, Dynamics and Materials Conference, Dallas, Texas, USA, 1992, p. 21-28.

[D.280]. Yang Q, Engblom JJ. Finite element based sub-laminate damage model for intraply cracking. J Reinf Plast Compos 1995;14(3):233-254.

[D.281]. Engblom JJ. Modelling the effects of intraply cracking in composite laminates at the sublaminate level. American Society of Mechanical Engineers, Pressure Vessels and Piping Division, vol. 121, 1987, p. 105-110.

[D.282]. Talreja R. A continuum mechanics characterization of damage in composite materials. Proc R Soc Lond A, Math Phys Sci 1985;399(1817):195-216.

[D.283]. Talreja R. Transverse cracking and stiffness reduction in composite laminates. J Compos Mater 1985;19(4):355-375.

[D.284]. Talreja R. Modeling of damage development in composites using internal variables concepts. Winter Annual Meeting of the American Society of Mechanical Engineers, Boston, Massachusetts, USA, 1987. p. 11-16.

[D.285]. Talreja R. Damage characterization by internal variables. In: Talreja R, editors. Damage Mechanics of Composite Materials, Composite Materials Series, Volume 5, 1994. p. 53-78.

[D.286]. Talreja R, Yalvac S, Yats LD, Wetters DG. Transverse cracking and stiffness reduction in cross ply laminates of different matrix toughness. J Compos Mater 1992;26(11):1644-1663.

[D.287]. Talreja R. Internal variable damage mechanics of composite materials. Proceedings of the IUTAM/ICM Symposium on Yielding, Damage, and Failure of Anisotropic Solids, Villard-de-Lans, France, August 24-28, 1987. p. 509-533.

[D.288]. Goyal VK, Jaunky NR, Johnson ER, Ambur DR. Intralaminar and interlaminar progressive failure analyses of composite panels with circular cutouts. Compos Struct 2004;64(1):91-105.

[D.289]. Schipperen JHA. An anisotropic damage model for the description of transverse matrix cracking in a graphite-epoxy laminate. Compos Struct 2001;53(3):295-299.

[D.290]. Isometsa J, Sjoling SG. A continuum damage mechanics model for fiber reinforced composites. Int J Damage Mech 1999;8(1):2-17.

[D.291]. Yang F, Chow CL. Progressive damage of unidirectional graphite/epoxy composites containing a circular hole. J Compos Mater 1998;32(6):504-525.

[D.292]. Chow CL, Yang F. Inelastic finite element analysis of fiber reinforced laminates damage. Proceedings of the 1994 International Mechanical Engineering Congress and Exposition, Chicago Illinois, USA, November 6-11, 1994. p. 111-135.

[D.293]. Chow CL, Yang F. A simple model for brittle composite lamina with damage. J Reinf Plast Compos 1992;11(3):222-242.

[D.294]. Chow CL, Yang F. On one-parameter description of damage state for brittle material. Eng Fract Mech 1991;40(2):335-343.

[D.295]. Chow CL, Yang F. Three-dimensional inelastic stress analysis of center notched composite laminates with damage. Int J Damage Mech 1997;6(1):23-50.

[D.296]. Chow CL, Yang F, Asundi A. A three-dimensional analysis of symmetric composite laminates with damage. Int J Damage Mech 1993;2(3):229-245.

[D.297]. Chow CL, Yang F, Asundi A. A method for nonlinear damage analysis for anisotropic materials and its application to thin composite materials. Int J Damage Mech 1992;1(3):347-366.

[D.298]. Chatterjee SN. Damage, stiffness loss, and failure in composite structures. In: Grant P, Rousseau C Q, editors. Composite Structures: Theory and Practice, ASTM STP 1383, American Society for Testing and Materials, West Conshohocken, Pennsylvania, USA, 2000. p. 452-469.

[D.299]. Turon A, Costa J, Maimi P, Trias D, Mayugo JA. A progressive damage model for unidirectional fibre-reinforced composites based on fibre fragmentation. Part I: Formulation. Compos Sci Techol 2005;65(13):2039-2048.

[D.300]. Barbero EJ, Lonetti P. Damage model for composites defined in terms of available data. Mech Compos Mater Struct 2001;8(4):299-315.

[D.301]. Barbero EJ, Vivo LD. A constitutive model for elastic damage in fiber-reinforced PMC laminae. Int J Damage Mech 2001;10(1):73-93.

[D.302]. Barbero EJ, Lonetti P. An inelastic damage model for fiber reinforced laminates. J Compos Mater 2002;36(8):941-962.

[D.303]. Barbero EJ, Abdelal GF, Caceres A. A micromechanics approach for damage modeling of polymer matrix composites. Compos Struct 2005;67(4):427-436.

[D.304]. Ladeveze P, La Dantec E. Damage modelling of the elementary ply for laminated composites. Compos Sci Technol 1992;43(3):257-267.

[D.305]. Ladeveze P. A damage computational method for composite structures. Comput Struct 1992;44(1/2): 79-87.

[D.306]. Allix O, Bahlouli N, Cluzel C, Perret L. Modelling and identification of temperature dependent mechanical behavior of the elementary ply in carbon/epoxy laminates. Compos Sci Technol 1996;56(7):883-888.

[D.307]. Li S, Reid SR, Soden PD. A continuum damage model for transverse matrix cracking in laminated fibre-reinforced composites. Philos Trans R Soc Lond A, Math Phys Sci 1998;356(1746):2379-2412.

[D.308]. Matzenmiller A, Lubliner J, Taylor RL. A constitutive model for anisotropic damage in fiber-composites. Mech Mater 1995;20(2):125-152.

[D.309]. Hammer VB. Optimization of fibrous laminates undergoing progressive damage. Int J Numer Methods Eng 2000;48(9):1265-1284.

[D.310]. Hammer VB, Pedersen P. On an orthotropic model for progressive degradation. Compos Struct 1999;46(3):217-228.