Characterizing the Mechanical Properties of Composite Materials Using Tubular Samples

By
Robert Hansbrough Carter

Dissertation Submitted to the Faculty of the
Virginia Polytechnic Institute and State University
In Partial Fulfillment of the Requirements for the Degree of:

Doctorate of Philosophy
In
Materials Engineering Science

Dr. Kenneth Reifsnider, Chair
Dr. William Curtin
Dr. Scott Case
Dr. Michael Hyer
Dr. Jack Lesko
Dr. Steve Kampe

July 16, 2001
Blacksburg, Virginia

Keywords: Composites, mechanical characterization, tubes, nonlinear regression

Characterizing the Mechanical Properties of Composite Materials Using Tubular Samples

By

Robert Hansbrough Carter

ABSTRACT

Application of composite materials to structures has presented the need for engineering analysis and modeling to understand the failure mechanisms. Unfortunately, composite materials, especially in a tubular geometry, present a situation where it is difficult to generate simple stress states that allow for the characterization of the ply-level properties. The present work focuses on calculating the mechanical characteristics, both on a global and local level, for composite laminate tubes. Global responses to axisymmetric test conditions (axial tension, torsion, and internal pressure) are measured on sections of the material. New analysis techniques are developed to use the global responses to calculate the ply level properties for tubular composite structures. Error analyses are performed to illustrate the sensitivity of the nonlinear regression methods to error in the experimental data. Ideal test matrices are proposed to provide the best data sets for improved accuracy of the property estimates. With these values, the stress and strain states can be calculated through the thickness of the material, enabling the application of failure criteria, and the calculation of failure envelopes.

ACKNOWLEDGEMENTS

The author wishes to acknowledge those that have been involved with this process:

- Dr. Kenneth Reifsnider, my advisor at the completion of this degree, for all his time, advice, and patience over the last several years.
- Dr. William Curtin, for serving as my advisor when I entered into the PhD program and getting me started on this project.
- Dr. Scott Case, for his help with all of my work, answering numerous questions, and taking the time to serve on my committee.
- Dr. Michael Hyer, Dr. Jack Lesko, and Dr. Steve Kampe – for their time and effort as members of my committee.
- Xinyu Huang, for his contribution on the McDermott filter research.
- Mac McCord, for making many of the test fixtures, which worked well even with the short notice given and without a good explanation of what was needed.
- Dr. Howard Halverson and Blair Russell, for their help in developing my experimental procedures and practices.
- Shelia Collins, Beverly Williams, Jan Doran, and the other departmental secretaries for their help, organization, and advice on how everything is done within the departments.
- Dr. Eric Johnson, for the use of the Aerospace Department's MTS frame that simplified testing and data analysis.
- Dr. Ted Bessman and Kent Probst, for providing the oxide/SiC tubes from Oak Ridge.
- Bill Patterson and Honeywell Advanced Composites, for supplying the SiC/SiC materials.
- Dr. Rich Wagner and McDermott Technologies, for supplying the hot gas candle filters.
- Dave Simmons and the staff of the machine shop, for the high quality work on the loading fixtures used in this research.

- Dr. Walter H. Carter, my father, for his help with the statistical analyses, and the constant support and motivation he has provided.
- Judy Carter, my mother, for her incredible patience hearing my father and I talk incessantly about my research, and the love and support she has provided.
- The students of the MRG, for their help and company (and lunch) over the last several years.
- Niki Parker and my friends at Virginia Tech for providing a lot of good times and putting up with so much.
- Mike Whaley and Mark Wilson, for giving me the opportunity to have a lot of fun and to tend bar in my spare time at TOTS.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Utilization of materials for any application requires knowledge of the mechanical properties, more specifically the engineering material constants describing the elastic behavior of the material. For isotropic materials, only three independent engineering constants (E, G or ν, and α) are needed to describe the elastic response of the material to mechanical and thermal loads. With the advent of more advanced composite materials, many of which cannot be described as isotropic, more constants are needed to describe the behavior. For most composite materials, 12 engineering constants (E_1, E_2, E_3, G_{12}, G_{13}, G_{23}, ν_{12}, ν_{13}, ν_{23}, α_1, α_2, and α_3) are required. Measuring these values requires the testing of numerous samples with several different test procedures. These procedures are designed to generate a constant or nearly constant state of stress throughout the material, reducing the number of elastic constants needed to describe the deformation for the given loading condition to the smallest number possible. This is usually done by using simple geometries, usually planar samples, to measure 1 to 3 engineering constants simultaneously (as in an longitudinal modulus test – E_1, ν_{12}, and ν_{13} can be measured simultaneously). In an ideal situation, with the proper test matrix and procedures, all the material constants can be calculated, given enough tests and sample materials.

A problem arises if the situation is not ideal, and, for example, the samples cannot be made into planar geometries. With a change in geometry from planar to cylindrical, the behavior cannot be described by using a few elastic constants at a time. For a composite tube (a laminated, anisotropic system), load/displacement relations become geometrically nonlinear, turning nearly all displacements into a function of several elastic constants. It becomes difficult, if not impossible, to produce a loading condition that will generate a simple stress state, which can be described by a small number of elastic constants. Characterization of the mechanical response is limited to the response of the structure as a whole, and not that of the constituent material (ply-level properties of the composite).

With thermoset and thermoplastic composite materials, the properties are generally well documented. The use of “prepreg” materials allows for the fabrication of test samples from the same material as the tubular part. The problem arises when the fabrication method does not allow for the development of traditional test samples to characterize the material, as is the case

with many new ceramic matrix composite materials. Recent work at the Materials Response Group at Virginia Tech has been on characterizing several composite materials fabricated with new procedures that were developed for tubular samples that do not have planar counterparts for characterization [1-5]. The projects have been complicated by the fact that the materials are fabricated in tubular geometries, with each tube being in a separate "batch", posing the possibility of different properties for each sample. Moreover, the properties through the thickness and sometimes in the plane of the surface may not be constant, i.e., the materials may be point-wise non-uniform. The processes were not able to fabricate identical materials to characterize the elastic properties, and the global response would not provide enough information to develop accurate models for use in later applications.

The problem has been further complicated by a lack of literature reporting on methods to characterize the elastic properties of composite materials in tubular geometries. There exists a large discrepancy in the approaches and procedures.

1.1 Objectives

There are three primary objectives for this dissertation. They address the need for experimental and analytical methods for characterizing the mechanical properties of composite tubes.

1. **Develop an experimental test matrix to measure the mechanical characteristics of composite tubes**. By this, it is intended that the global response (response of the tube as a whole) of the sample be determined by a series of tests. Axial tension, compression, torsion, and internal pressure tests were chosen to characterize the material, both in elastic response and for strength.
2. **Develop/refine an analysis for calculating the ply-level properties for the composite from the data found by the first objective**. Currently, no systematic method exists for calculating the ply-level engineering constants (E_1, E_2, E_3, G_{12}, ν_{12}, ν_{13}, and ν_{23}) for a composite tube. A method, initially utilized with limited success, has been modified, and used to calculate the engineering constants from the results of the first objective.

3. **Determine failure modes and calculate failure envelopes for this material**. With the values from the second objective, the stress and strain in the material can be calculated at the failure loads measured from the results of the first objective. With the state of stress at failure known for different loading conditions, failure envelopes can be calculated.

1.2 Materials

The focus of this work is on characterizing the elastic response of different composite tubes. There are three composite materials and one control sample used in this investigation. The first material is a candidate hot gas filter supplied by McDermott Technologies, Inc. of Lynchburg, Virginia. The second material is an alumina fiber reinforced silicon carbide (SiC) composite material supplied by Oak Ridge National Laboratory (ORNL). The third composite material is a silicon carbide reinforced silicon carbide composite from Honeywell Advanced Composites, of Wilmington, Delaware. The control sample is an AISI Type 304 stainless steel, chosen because its properties are well documented and should be similar to the properties of the silicon carbide materials.

In the following sections the background and fabrication procedures of the different materials will be explored. The McDermott material is not a structural composite material and has a specific application, so a little more background is given. The ORNL tubes were developed in a scale up of a new fabrication technique for ceramic composite materials, and sent to Virginia Tech for characterization of the mechanical properties. The description will focus less on potential applications but more on the fabrication and testing. A limited background will be given on the tubes from Honeywell since the materials were donated and much of the fabrication information is proprietary.

1.2.1 McDermott Technologies Hot Gas filter

An effort by the Department of Energy to improve efficiency and reduce emissions of fossil fuel based power plants has focused on development of improved plant design and technology. One area of this emphasis is directed at replacing the multiple processes currently

used to remove particulate from hot gas streams. Coal combustion generates large amounts of particulate from the noncombustible constituents and incomplete combustion products. These particulates must be removed to prevent excess wear on the gas turbines and to meet environmental emission standards.

Recent advances in the design of Pressurized Fluidized Bed Combustion (PFBC) and Integrated Gasification Combined Cycle (IGCC) technologies have utilized hot gas filters to remove particulate. It is believed the improvements will lead to operating efficiencies in excess of 50%, compared to 35% for conventional technologies. Increases in efficiency and economic savings will be due to the introduction of new combustion processes and plant designs that use hot gas filters. The filters would work in conjunction with or in place of conventional particulate removal systems, such as high efficiency cyclones and electrostatic precipitators, to remove particulate from the hot gas stream and meet environmental emission standards [6-20]. Benefits of the filters are: higher operating temperatures and efficiency, lower downstream particulate concentrations (especially at the smaller particle sizes), the resulting reduction in wear on downstream components, a reduction in complexity for hot gas cleaning, and reduced maintenance of plant components [6,8-12]. Currently, multiple high efficiency cyclones remove greater than 98% of the particulate, but they cause a drop in pressure and cannot operate at optimum temperature ranges. The small amount of particulate that is not removed erodes and fouls the blades in the turbines and other downstream components, requiring the use of less efficient, more damage tolerant equipment. The new designs use the hot gas filters directly in the hot gas stream, operating at higher temperatures and showing a lower pressure drop that in the conventional design. The filters remove more particulate (>99.8%), eliminating the need for cyclones or electrostatic precipitators, and allowing for the use of high efficiency turbines. A schematic of the Power Systems Development Facility (PSDF), located in Wilsonville, AL, with four different filtration units utilizing hot gas candle filters is shown in Figure 1-1 and Figure 1-2 [12]. On the left side of Figure 1-1, coal and limestone are fed into a crusher and sent to the two parts of the facility. The upper branch is a second generation PFBC, while the lower branch is a gasification system. Each branch uses hot gas filters to remove particulate from the different hot gas streams. Figure 1-2 contains the schematic of the hot gas filtration units (highlighted regions in Figure 1-1). The hot gas from the combustion of the fuel (coal or other fossil fuels) enters through the side of the housing, passes through the filter from the outer surface to inner region,

depositing any particulate, and exits the top of the assembly. As the deposit, or cake, thickens, the filter begins to clog, reducing the airflow. Periodic cleaning is required to maintain capacity. A back pulse cleaning cycle, which consists of a pulse of air forced through the filter in reverse, is used when the airflow through the filters falls below a set level. The pulse breaks the particulate cake formation, as seen in schematic diagram located in Figure 1-3. The particulate material falls to the bottom of the unit, and is removed.

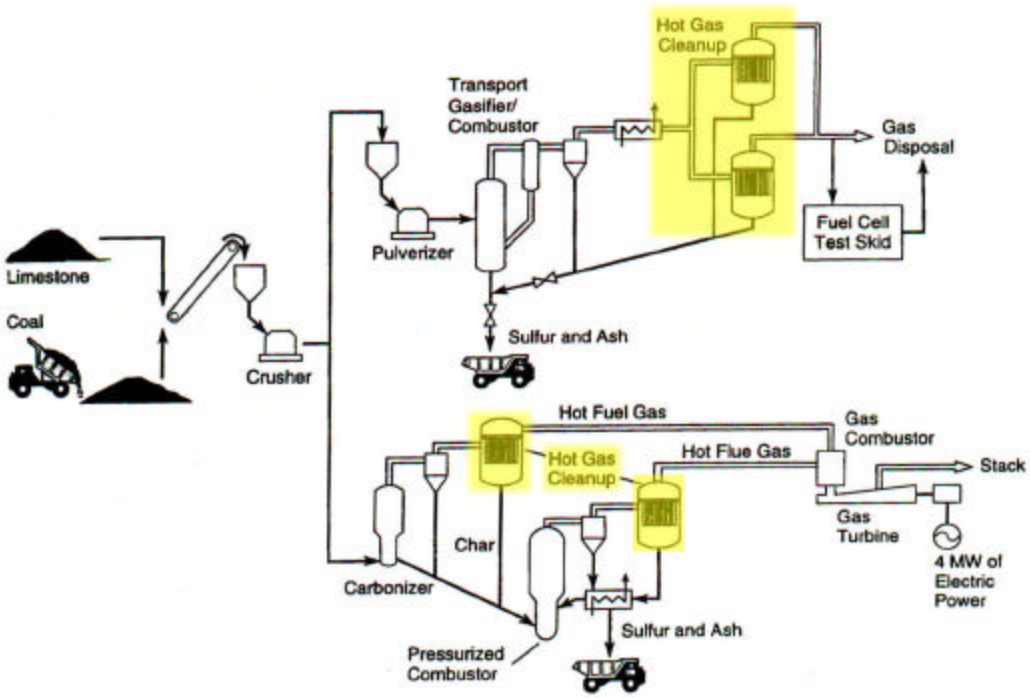


Figure 1-1. Schematic of a second generation PFBC facility (note the different Hot Gas Cleanup areas are highlighted) [Ref. #12, Smith and Ahmadi, 1998 – Reprinted with permission from Elseveir Science Publishing].

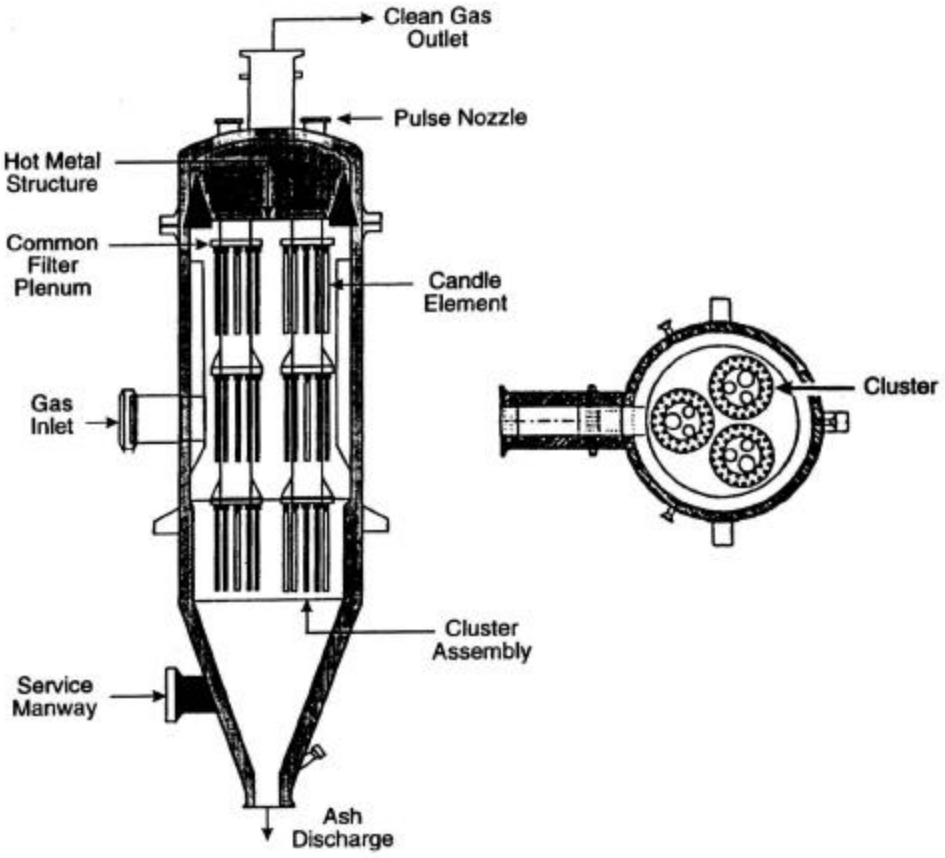


Figure 1-2. Schematic of Candle filter assembly [Ref. #12, Smith and Ahmadi, 1998 – Reprinted with permission from Elseveir Science Publishing].

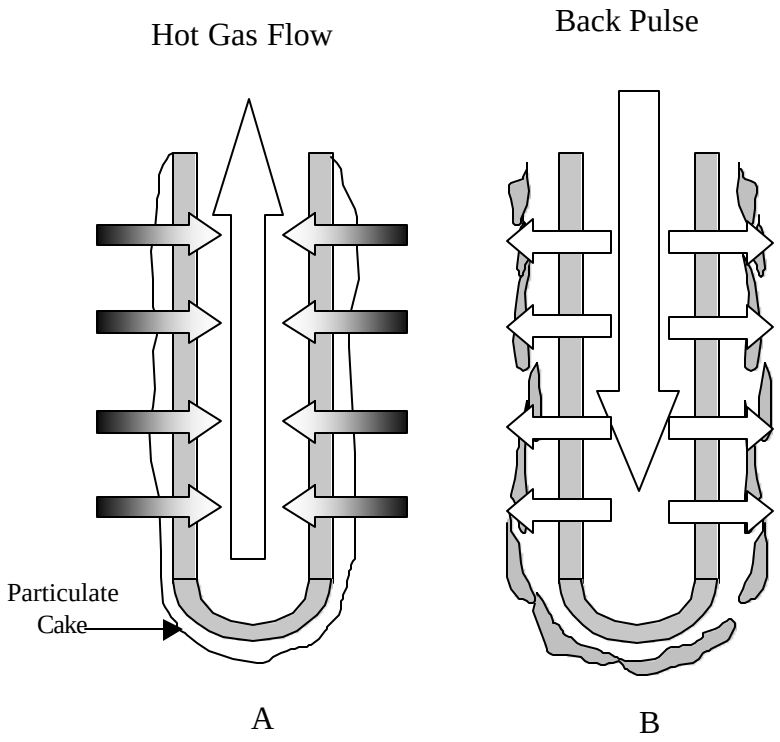


Figure 1-3. Filter operation and a back pulse cleaning cycle. A)Clogged Filter. B)Back Pulse.

In-service conditions for the filters are listed in Table 1-I. The ranges in the particle size and dust loading are due to different designs or operating conditions. Some designs use a cyclone prior to the filtration to reduce the frequency of filter cleaning cycles. It is uncertain if this will be used in the plant designs, since some research points to improved performance without the cyclone. The cyclone removes the larger particulate, leaving the finer particles for the filter. The smaller particulate imbeds in the filter more and develops a cake that is more difficult to remove, decreasing the effectiveness of the cleaning cycles.

Table 1-I. Operating Condition for Hot Gas Filter[8,9]

Temperature (ºC)	Pressure (MPa)	Dust Loading (ppm)	Mean Particle Size (ìm)	Pressure Drop (kPa)
650 to 900	0.98 to 1.1	500 to 18000	1.3 to 22	28-41

Current materials and designs of the filters should allow for long operating lives, but application of these materials to actual service conditions has lead to some premature failures. Due to the difficulty in observing the filters in-service, many of the reasons for failure of the materials are attributed to conditions that are an educated guess as to actual conditions at failure. A majority of the failures in the literature have been attributed to plant upset conditions - variations, some very rapid and extreme, in the plant conditions. The startup of the combustors and turbine trip conditions result in large pressure and temperature gradients, and have been attributed to failure of the materials [10,11].

Failures during normal plant conditions have been attributed to thermal shock resulting from the back pulse cleaning cycles and ash bridging. The air pulse used in the cleaning cycle is cooler than the hot gas stream (in some cases the air is near ambient temperatures). The very rapid influx of the cool gas generates large thermal gradients within the filter. Many monolithic filters do not survive this event, while the composite materials usually fair better. The cleaning of the filters is an important process to the operation of the plant. It is needed to maintain proper airflow and to maintain the filters. Without proper or complete cleaning, the cake builds and can break the filters. A number of reasons for this breakage have been proposed in the literature. The most common observed condition is ash bridging, where incomplete cleaning of the filters leads to an ash buildup, which eventually connects two adjacent filters. This connection places added stresses on the filters, breaking them from the tube sheet near the flange. In some cases, the filters were bent by the ash bridge and ultimately broken, while in other conditions falling sections of ash buildup were blamed for breaking the filters.

Application of the filters to test facilities has yielded excellent filtration performance, but they have experienced several mechanical failures. These failures have been attributed to excessive stresses induced during plant upset conditions, thermal shock, and insufficient strength for certain in-service conditions. In order to improve the design of the materials and develop methods for life predictions, the baseline material properties need to be measured. With composite tubes, there is no way to measure the engineering constants directly, due to the complexity of the geometry. This body of work was developed to address these issues.

1.2.2 Fabrication of the McDermott Technologies' Hot Gas Filter

The candidate design from McDermott Technologies, Inc. (MTI), of Lynchburg, VA, is an advanced oxide-oxide ceramic composite. It consists of two different aluminum oxide fibers wound and deposited around a round mandrel and bonded together to form a highly porous structure. The structural component is Nextel 610 fibers (>99% alumina) wound onto a porous mandrel while chopped a Saffil fiber (95 – 97% alumina, 3 –5% silica) slurry is applied. The water from the slurry is removed by applying a vacuum to the mandrel, drawing it through the green structure. The two components are bonded together by an alumina bond agent deposited using sol-gel processing. The reaction process is as follows:

$$AlOOH \rightarrow g - Al_2O_3 \rightarrow d - Al_2O_3 \rightarrow q - Al_2O_3 \rightarrow a - Al_2O_3 \qquad (1.1)$$

The oxyhydroxide of aluminum, or boehmite, transforms to alumina between 450 and 580 C. This deposits alumina throughout the filter, bonding the Saffil and Nextel fibers, imparting some structural integrity. The final product is typically a filter 1.5 m in length, 5 cm inner diameter and 6 cm outer diameter, with an approximate porosity of 70% [11]. The average dimensions of the McDermott samples are in Table 1-II.

Table 1-II. Dimensions of the McDermott Technologies hot gas filter samples

Sample	Thermal Exposures	Wall Thickness (mm)	Outer Diamter (mm)	Inner Diameter (mm)	Cross-sectional Area (mm^2)	Polar Moment of Inertia (mm^4)
7-2-28	0	4.85	59.41	49.80	831.80	6.24E+05
7-5-13	103	4.93	58.96	49.11	836.52	6.16E+05
7-6-12	104	5.06	59.41	49.29	863.95	6.44E+05
7-6-16	105	5.39	59.98	49.19	924.76	6.96E+05

Inspection of the tubes revealed that the outer surface winding angles were not ±45º. The outer fiber tows were closer to ±50º. The reason for the change in the winding angles was found in the thesis by George [21]. When a filament winding procedure is programmed using constant winding parameters, namely the transverse movement rate, x', and the mandrel rotation rate, θ',

adjustments are not made for the change in thickness for each added layer. For the first layer of the composite the rates were chosen so that the transverse length is equal to the circumferential length of fiber, which will give a ±45º structure. With each layer the radius is increased, changing the circumferential length of fiber per unit time. This is illustrated in Figure 1-4, which is a 2-D projection of the surface of the tube, with Rθ being an arc length of the circumference and X being distance along the axis.

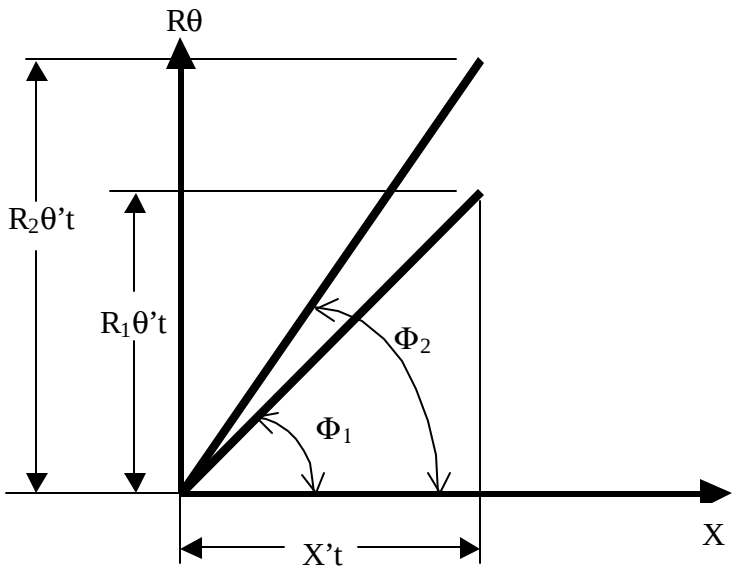


Figure 1-4. Relationship for ply angle and winding parameters

From this, the winding angle, ϕ, is:

$$f = tan^{-1}\left(\frac{Rq'}{x'}\right) \tag{1.2}$$

For a ±45º winding angle, the transverse rate is equal to the product of the initial radius, R_i, and the rotation rate:

$$x' = R_i q' \tag{1.3}$$

By combining these two, an expression can be found for the winding angle as a function of the radius for a ±45º structure.

$$f = tan^{-1}\left(\frac{R}{R_i}\right) \tag{1.4}$$

Measuring the angles of the tows from the etchings of the inner and outer surfaces of the McDermott filters confirms this behavior. The inner surfaces were found to be ±45º, while the outer surfaces were ~±50º, which matches the calculated value of ±50.5º.

Several samples of the McDermott filters were characterized. The initial work was to determine the effect of the back pulse cleaning cycle on the mechanical properties of the material. To simulate the cleaning cycle's thermal shock event, the samples were placed in a furnace, operating at 870ºC, and exposed to an ambient temperature air pulse. The samples tested were in the as-fabricated condition as well as after being subjected to 1, 10, and 100 thousand cleaning cycles.

1.2.3 Al_2O_3/SiC Tube from ORNL

The materials from Oak Ridge National Laboratory are part of a project aimed at developing advanced ceramic composite materials for use in high temperature applications for future fossil fuel based power plants. Fabrication of these materials has proved to be difficult and costly, so research is being performed to develop new fabrication procedures. One such fabrication technique is chemical vapor infiltration (CVI) in which the matrix material is deposited into the fiber performs. This allows for the formation of several different matrix and fiber systems without requiring the high processing temperatures seen in other processes. Unfortunately, standard CVI procedures require infiltration times that can be too long to be considered economically feasible.

Development of new processes has resulted in greatly reduced infiltration times, overcoming many of the problems of previous CVI techniques. The previous methods, such as isothermal/isobaric CVI, rely upon diffusion processes to deposit the matrix material. Low deposition rates were used to prevent large density gradients caused by the outer surfaces becoming fully dense, and not permitting infiltration to the inner portions. This results in long deposition times or high porosity. Researchers at Oak Ridge National Laboratory (ORNL) have developed a new method to overcome the problems of long process times or large density gradients [22-25]. The forced flow-thermal gradient process (FCVI) utilizes a temperature gradient to change the deposition rates from the inner preform to the outer surface. The

deposition times are reduced from a period of weeks to that of only hours, allowing the formation of nearly dense composites without density gradients.

The process has been used to fabricate planar samples, and it has been scaled-up to produce samples with a tubular geometry. To determine the effectiveness of the process, the completed materials have been sent out for material characterization. Much of the previous mechanical testing of composites made using the FCVI technique has been limited to relatively small planar samples. These test methods include flexure tests and some limited axial testing [26-28]. With the larger sample size and different sample geometry, different tests need to be performed to characterize the mechanical properties of tubular samples. The Materials Response Group at Virginia Tech has performed testing on previous tubes supplied by ORNL and Babcock and Wilcox (now McDermott Technologies, Inc., Lynchburg, VA) [29-31]. That work focused on the mechanical properties of Nicalon/SiC and alumina/alumina ceramic composites of various designs and lay-ups. The materials were fabricated by different methods, ranging from using forced flow CVI for the Nicalon/SiC [31] to sol-gel processing to deposit an alumina matrix for the alumina/alumina materials [29,30] The mechanical properties for these materials are listed in Table 1-III.

Table 1-III. Reported properties from previous tube samples

Material	Axial Stiffness Tension (GPa)	Shear Stiffness (GPa)	Axial Strength (MPa)	Shear Strength (MPa)
Nicalon/SiC Braided tube	---	127	---	---
Nicalon/SiC Cloth wrapped	~275	94	---	---
Almax/Al_20_3	54-61	41-43	41.4, 44.5	56.54

1.2.3.1 Fabrication

The composite tubes used in this investigation were fabricated using the forced-flow, thermal gradient chemical vapor infiltration technique developed at Oak Ridge [22-24]. The preforms consisted of eight to ten Nextel 610 (Nextel 312 for sample CVI-1173) braided sleeves stretched over a polyethylene mandrel. The green preform was infiltrated with a small amount of Borden Durite resin to provide some structural support prior to the silicon carbide infiltration.

The preform was compressed by aluminum tube sections and allowed to cure. The cured preform was trimmed to a 35.5-cm length prior to the SiC infiltration via the FCVI process. The FCVI process temperature was 1200°C with a gas flow rate of 5 slm of hydrogen and 1 slm of methyltrichlorosilane. The samples achieved 80 to 90% of theoretical densities in about 36 hours. Once processing was completed, the ends of the samples were removed to leave a 30-cm long sample. The final tube properties are listed in Table 1-IV. It should be noted that in this study, no fugitive carbon layer was deliberately deposited to improve composite toughness. A small amount may be present from the decomposition of the resin used to rigidize the preform, though none was observed in the initial inspection of the tubes.

Table 1-IV. Properties of the Nextel/SiC composite tubes

	CVI 1173	**CVI 1216**	**CVI 1219**
Process Time(hours)	36	43	36
Density (g/cm^3)	3.00	2.88	3.08
% Theoretical Density	87	81.8	80.3
Fiber Type	Nextel 312	Nextel 610	Nextel 610
Fiber Volume Fraction (%)	50	37.5	32.2
Number of layers	10	10	8

A cross section of the CVI-1173 tube can be seen in Figure 1-5. The light gray layer on the inner surface is a SiC layer deposited during the FCVI process. This layer was less than 1 mm thick at the ends of the tube, and approached 4 mm in thickness in some cross sections taken near the middle of the sample.

Figure 1-5: Cross section of sample CVI 1173

This change in dimension creates problems in accurately calculating the stress in the material. The cross sectional area and polar moment of inertia used in this study were calculated from the average values of the outer diameter and tube thickness at each end of the sample. The average dimensions for the samples are in Table 1-V. The changes in the dimensions of Sample CVI 1219, shown in the last two columns, are due to changes made to the sample during testing. An excess layer of SiC was deposited on the inner surface, and, when the material strength exceeded the load capacity of the MTS system, it was returned ORNL and the SiC layer was removed. The wall thickness values for the sample after milling exhibits large variations, which is due to the inner surface not being concentric with the outer.

Table 1-V. Dimensions of the samples

In mm	**CVI 1173-1**	**CVI 1173-2**	**CVI 1216**	**CVI 1219** (as-received)	**CVI 1219** (after milling)
Outer Diameter	59.7±0.2	59.7±0.2	59.3±0.06	59.2±0.1	59.1±0.2
Thickness	6.9±0.7	6.8±0.4	7.6±0.6	7.9±1.8	4.8±0.4
Inner Diameter	45.8	46.1	43.9	43.4	49.7
Area (10^{-4} m^2)	11.48	11.24	13.09	14.99	8.14

1.2.4 SiC/SiC Tube from Honeywell

The material donated by Honeywell is a SiC/SiC composite material utilizing Ceramic Grade Nicalon fibers to reinforce an enhanced SiC matrix deposited by chemical vapor infiltration [32]. A layer of pyrocarbon was deposited on the interface to improve the composite toughness and damage tolerance. The geometry and mechanical properties are in Table 1-VI and Table 1-VII. Three samples, each roughly 11.4 cm (4.5 in) in length, were cut from the same tube of material, and were sent to Virginia Tech for testing.

Table 1-VI. Geometry of the Honeywell SiC/SiC material

Outer Diameter (mm)	57.78	Thickness (mm)	3.70
Inner Diameter (mm)	50.39	Area (mm^2)	2543.19
Density	2.3 g/cc	Polar Moment of Inertia (mm^4)	$4.62*10^5$
Number of Plies	6 woven plies	Fiber Vol. Fraction	0.37
Orientation	[0/90] 5 Harness Satin Weave		

Table 1-VII. Typical Room Temperature Properties of the Honeywell SiC/SiC material

Axial Stiffness, E	138 GPa (20 Msi)
Transverse Tensile Strength, Sy	59 MPa (8.5 ksi)
Ultimate Tensile Strength, UTS	228 MPa (33 ksi)
Failure Strain	0.39%

1.2.5 AISI Type 304 Stainless Steel (Control)

To test the capabilities of the analysis for a simplest case scenario, a stainless steel control tube was subjected to the same tests as the other materials. The material was chosen since its properties are well documented in the literature, it exhibits stiffness greater than that expected for most of the materials tested, and it can be expected to exhibit (near) isotropic properties. The well-documented, isotropic properties are important to the analyses developed in the later portions of this work, allowing for its use as a good control sample. The high stiffness of the material is significant for the internal pressure test methods developed in the next section. It

allows for testing at high pressures, which would be needed to fail the composite materials. The properties from the material supplier are listed in Table 1-VIII through Table 1-X [33].

Table 1-VIII. Chemical composition of AISI Type 304 Steel

AISI Type 304	Weight % Range	
Composition	Minimum	Maximum
Fe	66.345	74
Cr	18	20
Ni	8	10.5
Mn	--	2
Si	--	1
C	--	0.08
P	--	0.045
S	--	0.03

Table 1-IX. Mechanical properties of AISI Type 304 Steel

Tensile Modulus, E	193-200 GPa	28.565 Msi
Shear Modulus, G	86 GPa	12.47 Msi
Yield Strength, $ó_o$	215 MPa	31.175
Tensile Strength, $ó_{UTS}$	505 MPa	73.225
Elongation, ε_{max}	70%	

Table 1-X. Geometry of the control sample tube

Average Outer Diameter	2.366±0.01 in	60.1±0.23 mm
Thickness, t	0.11±0.0	2.79±0.0 mm
Cross Sectional Area, A	0.78 in^2	5.03e-4 m^2
Polar Moment of Inertia, J	0.99 in^4	4.14e-7 m^4

1.3 Geometric Conventions

For the remainder of this body of work certain orientations and naming conventions will be used. The focus of this work is on composite tubes; therefore, the cylindrical coordinate system will be used to describe the geometry, as illustrated in Figure 1-6. Due to the different levels of this analysis, there will be two different classifications: the global and material levels. The

global level refers to the response of the tube as a whole, without the differentiation for the composite structure, and will be labeled with x, r, and/or θ. For example, the axial stiffness of the material is calculated as the axial force divided by the cross sectional area of the tube and the axial strain response (labeled E_x). It does not consider any of the composite material structure or orientation, but is intended to describe how the tube responds as a whole. The second classification, the material level, will be used to describe the response on the individual ply level. This will be affected by the orientation and ply number for the composite, and will be used for the composite analyses. The material will be described with a 1, 2 and/or 3 direction, with 1 being along the fibers, 2 perpendicular to the fibers (but in plane of the ply), and 3 perpendicular to the ply.

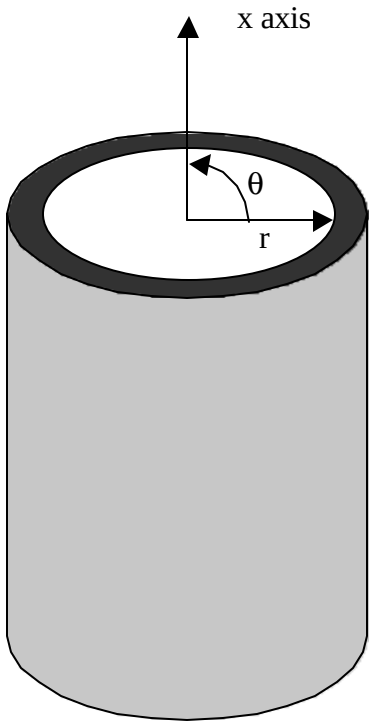


Figure 1-6. Global Coordinate System

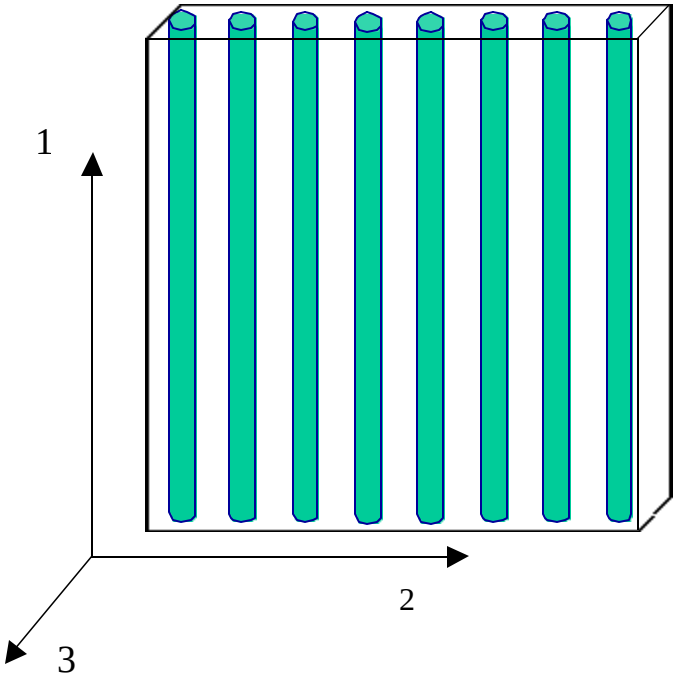


Figure 1-7. Material Coordinate System

1.4 Selection of Test Methods

Much of the background of this work has already been described in previous works by the author and others in the Materials Response Group at Virginia Tech [1-5, 21, 29-31]. The focus of this work is to develop a series of procedures and analyses to gain the most insight into the ply-level material properties of a laminated composite tube. The initial procedures were developed for characterizing the properties of the McDermott filter materials. The materials from Oak Ridge National Laboratory and Honeywell came later, but were composite tubes and could be characterized by the same methods with some minor modifications and improvements to the procedures, to accommodate the vastly different material properties of the samples.

For the candle filters, there are several different test programs to determine the different candidate materials and designs for use in the next generation power facilities. A few investigators have performed tests on the materials to determine which are the best for these applications. Papers by Singh, *et al.*, and Alvin, *et al.*, have reported work that characterized the performance in terms of property retention after thermal shock conditions and actual service

exposure [34-36]. Little of the literature has focused on the mechanical properties of filter materials. Many have used a variety of experimental procedures to obtain qualitative information about the material, with fewer measuring quantitative results. Tests, such as C-ring and O-ring tension and compression, have been performed without being able to evaluate any material properties from the results. The O-ring and C-ring tests apply a large, complex state of stress to a small section of the material – making analysis difficult. Several sources performed these tests to characterize changes in the bulk properties with exposure to thermal shock or high temperature exposure. Other tests have been performed that allow for some simplifying assumptions in the analysis. Internal pressure tests, using either a pressurized bladder or compressed elastomer plugs, have been used to find some modulus and strength values. These analyses do not account for the composite structure to calculate the material values. Finally, a few researchers have used sonic and ultrasonic time of flight measurements to obtain values for the axial modulus of the structures. These measurements use the relationship between stiffness and wave velocity to calculate stiffness values from time of flight measurements for sound waves.

The test methods used in the present work were chosen to generate axisymmetric load conditions which would allow for comparison of the results to an elasticity theory model. The analysis by George has been improved and employed to find estimates of the ply-level properties [21]. Because of this, procedures that generate axisymmetric load conditions were employed.

1.5 Analytical models

A thesis by George addressed the need for an analysis that would calculate the ply level properties from the global response of the composite tube [21]. The work developed a nonlinear regression routine that would find the best elastic constants to fit the experimentally observed strain results. The model used expressions derived using elasticity theory to calculate the stresses and strains for a given axisymmetric load condition. The derivation of this work was done by Rousseau and Hyer [37,38]. The work is for the general case, where the layers of the composite are orthotropic. For a few special conditions, there are certain solutions that cause the equations describing the material behavior to fail. In the first part of this section the general solution will be detailed, with the two degenerate conditions explained thereafter.

1.5.1 General Elasticity Solution ("Forward" Solution)

The derivation illustrated is from the work by Rousseau, *et al* [37,38]. The following expressions can be used to calculate the stresses, strains and displacements for a composite cylinder under axisymmetric loading conditions (axial tension, compression, torsion, internal and external pressure, and uniform temperature change – singly or in combination). By applying the strain-displacement relations to the displacement equations, the different strain components can be calculated through the thickness of the tube. The strain values are related to the stress values by the constitutive equations.

Displacement Equations – u = axial displacement, v = tangential displacement and w = radial displacement

$$
\begin{aligned}
u(x) &= e^o x \\
v(x,r) &= g^o x r \\
w(r) &= A_1 r^l + A_2 r^{-l} + \Gamma e^o r + \Omega g^o r^2 + \Psi r \Delta T \\
l &= \sqrt{\frac{\bar{C}_{22}}{\bar{C}_{33}}} \\
\Gamma &= \left(\frac{\bar{C}_{12} - \bar{C}_{13}}{\bar{C}_{33} - \bar{C}_{22}} \right) \\
\Omega &= \left(\frac{\bar{C}_{26} - 2\bar{C}_{36}}{4\bar{C}_{33} - \bar{C}_{22}} \right) \\
\Psi &= \left(\frac{\sum (\bar{C}_{i2} - \bar{C}_{i3}) a_i}{\bar{C}_{33} - \bar{C}_{22}} \right)
\end{aligned}
\tag{1.5}
$$

Strain-Displacement Equations

$$
\begin{aligned}
e_x &= \frac{\partial u}{\partial x} \quad & g_{rq} &= \frac{\partial v}{\partial r} - \frac{v}{r} \\
e_q &= \frac{w}{r} \quad & g_{xr} &= \frac{\partial u}{\partial r} \\
e_r &= \frac{\partial w}{\partial r} \quad & g_{xq} &= \frac{\partial v}{\partial x}
\end{aligned}
\tag{1.6}
$$

which for the general case expand to:

$$
\begin{aligned}
e_r &= l\,A_1 r^{l-1} - l\,A_2 r^{-l-1} + \Gamma e^o + 2\Omega g^o r + \Psi \Delta T \\
e_q &= A_1 r^{l-1} + A_2 r^{-l-1} + \Gamma e^o + \Omega g^o r + \Psi \Delta T \\
e_x &= e^o \\
g_{qr} &= 0 \\
g_{rx} &= 0 \\
g_{xq} &= g^o r
\end{aligned}
\tag{1.7}
$$

Constitutive Equations

$$
\begin{bmatrix} s_x \\ s_q \\ s_r \\ g_{qr} \\ g_{xr} \\ g_{xq} \end{bmatrix} =
\begin{bmatrix}
\overline{C}_{11} & \overline{C}_{12} & \overline{C}_{13} & 0 & 0 & \overline{C}_{16} \\
\overline{C}_{12} & \overline{C}_{22} & \overline{C}_{23} & 0 & 0 & \overline{C}_{26} \\
\overline{C}_{13} & \overline{C}_{23} & \overline{C}_{33} & 0 & 0 & \overline{C}_{36} \\
0 & 0 & 0 & \overline{C}_{44} & \overline{C}_{45} & 0 \\
0 & 0 & 0 & \overline{C}_{45} & \overline{C}_{55} & 0 \\
\overline{C}_{16} & \overline{C}_{26} & \overline{C}_{36} & 0 & 0 & \overline{C}_{66}
\end{bmatrix}
\begin{bmatrix} e_x \\ e_q \\ e_r \\ g_{qr} \\ g_{xr} \\ g_{xq} \end{bmatrix}
\tag{1.8}
$$

At this point the expressions for stress, strain and displacement can be derived in terms of the transformed material stiffness matrix ($\overline{C}_{ij}$), ε^0, γ^0, A_1 and A_2. Since the elastic properties and geometric considerations are known for a given structure, the $\overline{C}_{ij}$ values can be calculated for each ply. The remaining unknown terms - ε^0, γ^0, A_1 and A_2 – need to be found using the boundary conditions. For a laminated structure composed of N layers, there will be 2N+2

unknowns – ε^0, γ^0, N A_1's, and N A_2's. The first boundary condition is a relation between the applied axial force and the axial stress. For a tube in axial tension, the applied axial force, F_x, must be equal to the sum of the integrals of the axial stress through the thickness. The same applies to a tube with an applied torque, T_x, the sum of the integrals of the shear stress multiplied by the radial position must equal the applied torque. The expressions for these boundary conditions are:

$$\begin{aligned} F_x &= 2p\sum_{k=1}^{N}\int_{r_{k-1}}^{r_k} s_x^{(k)} r\,dr \\ T_x &= 2p\sum_{k=1}^{N}\int_{r_{k-1}}^{r_k} t_{xq}^{(k)} r^2\,dr \end{aligned} \tag{1.9}$$

This gives two equations towards the 2N+2 unknowns. Two more come from the pressurized cylinder condition. The pressure at the inner and outer surfaces must be equal and opposite to the applied pressures. These can be equated as:

$$\begin{aligned} -p_i &= s_r^1(R_i) \\ -p_o &= s_r^N(R_o) \end{aligned} \tag{1.10}$$

where p_i and p_o are in the applied internal and external pressures, R_i and R_o are the inner and outer radii, and the superscript on the stress expression is the layer of the material. The last two requirements for the tube are that of continuity of traction and displacements at ply interfaces, or:

$$\begin{aligned} w^{(k)}(r_k) &= w^{(k+1)}(r_k) \\ s_r^{(k)}(r_k) &= s_r^{(k+1)}(r_k) \end{aligned} \tag{1.11}$$

This gives the last 2(N-1) equations needed to solve for the unknowns. Simultaneously solving the above equations will give the 2N+2 unknowns for the displacement, strain and stress equations.

1.5.2 Degenerate Solutions

By examining Equation (1.5), there are two conditions that cause zero to appear in the denominator in the expression for Ã, namely the conditions where $\bar{C}_{22} = \bar{C}_{33}$ or $\bar{C}_{22} = 4\bar{C}_{33}$. The

first of these conditions occurs when the material is isotropic or transversely isotropic with a 0 degree orientation. The second condition has not been associated with an easily explained physical situation, but is included for completeness. In both of the degenerate conditions, the expressions for both u and v remain the same as before, but that for w changes [39].

For the case of $\bar{C}_{22} = \bar{C}_{33}$ (ë=1), w becomes:

$$
\begin{aligned}
w(r) &= A_1 r^{l} + A_2 r^{-l} + \Gamma e^{o} r \ln(r) + \Omega g^{o} r^{2} + \Psi \Delta T r \ln(r) \\
l &= \sqrt{\frac{\bar{C}_{22}}{\bar{C}_{33}}} = 1 \\
\Gamma &= \left(\frac{\bar{C}_{12} - \bar{C}_{13}}{2\bar{C}_{33}} \right) \\
\Omega &= \left(\frac{\bar{C}_{26} - 2\bar{C}_{36}}{4\bar{C}_{33} - \bar{C}_{22}} \right) \\
\Psi &= \left(\frac{\sum (\bar{C}_{i2} - \bar{C}_{i3}) a_i}{2\bar{C}_{33}} \right)
\end{aligned}
\tag{1.12}
$$

For the case of $\bar{C}_{22} = 4\bar{C}_{33}$ (ë=2), w becomes:

$$
\begin{aligned}
w(r) &= A_1 r^{l} + A_2 r^{-l} + \Gamma e^{o} r + \Omega g^{o} r^{2} (4\ln(r) - 1) + \Psi r \Delta T \\
l &= \sqrt{\frac{\bar{C}_{22}}{\bar{C}_{33}}} = 2 \\
\Gamma &= \left(\frac{\bar{C}_{12} - \bar{C}_{13}}{3\bar{C}_{33}} \right) \\
\Omega &= \left(\frac{\bar{C}_{26} - 2\bar{C}_{36}}{16\bar{C}_{33}} \right) \\
\Psi &= \left(\frac{\sum (\bar{C}_{i2} - \bar{C}_{i3}) a_i}{3\bar{C}_{33}} \right)
\end{aligned}
\tag{1.13}
$$

A more detailed derivation of these expressions can be found in Appendix A. Since the value of ë varies from ply to ply, the choice of the expression for w, and the resultant expressions for the boundary conditions, must be made for each layer.

1.6 Inverse Solution

The elasticity or Forward Solution calculates the stresses and strains generated by an applied load given the elastic properties of the materials. This is a useful model, but it requires the knowledge of the in-plane and some out-of-plane constants. Complete sets of these values are not known for many materials and are, in general, difficult to calculate. An added complication is encountered for tubular materials that do not have a planar counterpart for determining the elastic constants, which was the case for the materials used in this body of work. Both the MTI filter and ORNL materials were fabricated in processes designed for tubular geometry, therefore creating planar samples for testing was not possible. For these laminate structures with varied ply orientations, the geometric effects do not allow for the generation of a simple stress state to estimate the material stiffnesses. A methodology was developed by George to calculate the properties from the tubular samples by nonlinear regression analysis of the strain data [12]. In this body of work the analysis is called the Inverse solution to the elasticity theory derived by Hyer and Rousseau [37,38]. This is not a closed form solution for calculating the material properties, but is an analysis that relies on nonlinear regression techniques to calculate a set of material parameters that minimize an error function. A simple schematic of the method is located in Figure 1-8. The analysis uses the Forward solution, as the core of the program. The user inputs the geometric properties needed to describe the sample. The start or "guess" values for the material properties are entered, as are the experimentally recorded strain and loads. It should be noted that the applied loads (axial force, axial torque, and internal pressure) are entered and not the calculated stresses (axial stress and shear stress). With this information, the forward solution calculates a predicted strain response at the same load levels as the experimental results. The difference between the measured and calculated values is determined. The error function, the sum of squared errors, is minimized through a number of iterations until a minimum is found or the program exceeds set parameters. The termination of the procedures relies upon declared convergence criteria, basically a series of values or conditions that determine whether a solution has been found or that a reasonable amount of time or iterations has been exceeded.

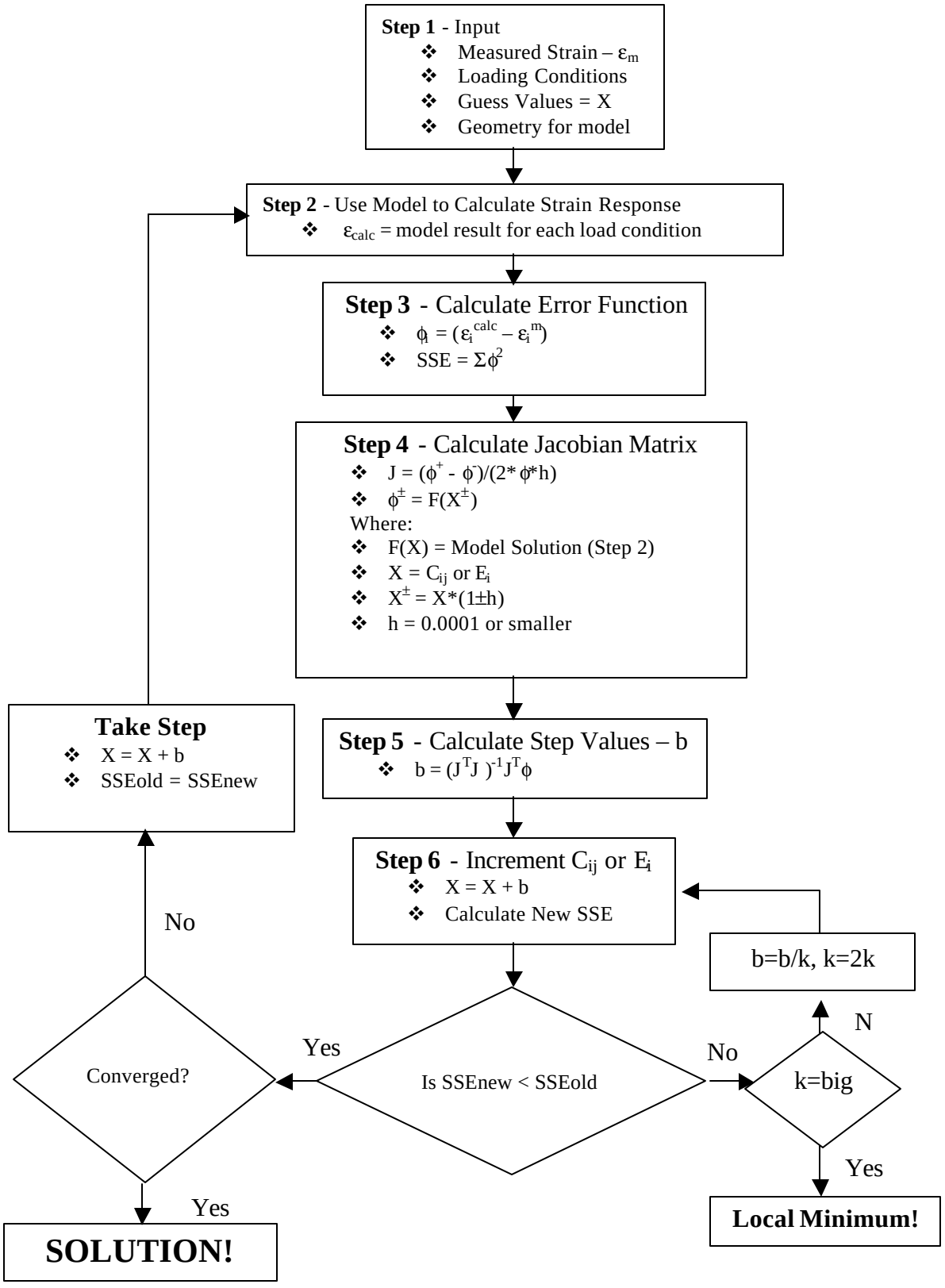


Figure 1-8. Schematic of nonlinear regression analysis

The method employed in the thesis by George is termed a Newtonian method, which utilizes the partial derivatives to calculate an incremental step. The equation for the step values, b_i, are:

$$\begin{aligned} E_i^{k+1} &= E_i^k + b_i \\ b &= \left(J^T J\right)^{-1} J^T f \\ f_i &= \left(e_i^{calc} - e_i^{\exp}\right) \end{aligned} \tag{1.14}$$

where E_i are the different elastic constants (E_1, E_2, E_3, G_{12}, ν_{12}, ν_{13}, and ν_{23}), the superscript k is the interation count, ϕ_i is the difference between each of the calculated and measured strain values, and J is the Jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial E_1} & \cdots & \frac{\partial f_1}{\partial n_{23}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial E_1} & \cdots & \frac{\partial f_N}{\partial n_{23}} \end{bmatrix} \tag{1.15}$$

where ϕ is the difference between the observed and predicted values, defined in (1.14). The size of the ϕ vector depends upon the number of different loading conditions and number of different strain measurements. For this work, the value N in the Jacobian is set to be three times the number of different loading conditions, since there are three strain values used per load (ε_x, ε_θ, $\gamma_{x\theta}$). The derivative values are approximated, since deriving the exact expressions would be exceedingly complicated to implement into computer codes. The values are approximated by:

$$\frac{\partial f_j}{\partial X_i} = \frac{f_j^+ - f_j^-}{2 X_i h} \tag{1.16}$$

where $\phi^{\pm}$ are the ϕ values calculated with the X_i ($X_i = E_i$) changed to $X_i(1 \pm h)$, and h is a small value ($h \ll 1$).

With each iteration, the procedure evaluates whether the new values are decreasing the error function, or

$$SSE(X_i^{k+1}) < SSE(X_i^k) \tag{1.17}$$

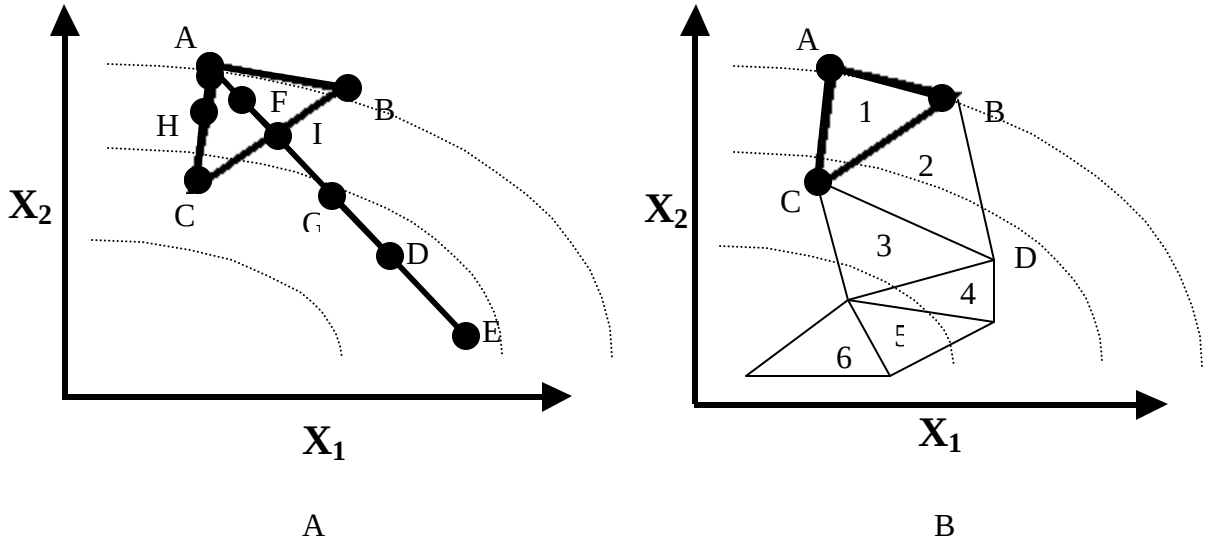
If the error decreases, the values are incremented by the step values, and the process proceeds to the next iteration. If the error is not decreased, the program begins a routine the decreases the

size of the step values, such as, the step size is cut in half until a decrease is achieved. A set number of repeats is usually set, after which, the program declares a local minimum since a small step does not decrease the error, and terminates.

The program by George was tested using strain data generated by the Forward Solution and input into the Inversion program with erroneous start values. After several iterations, the program found the correct values for the elastic constants, minimizing the error function. Attempts to apply this procedure to experimental results met with limited success. Only axial tension and torsion were used to generate the strain data sets. Due to the limited amount of data and the requirement that the system be overdefined (more data points than unknown values – a condition needed for any regression work), two of the Poisson ratios were set to be constant since there was insufficient data to optimize all values. The number of experimental data points fell to 6, limiting the number of variables to 5.

Initial work using Newtonian methods to search for a minimum was problematic. As the analysis neared a solution, the step values would become large, sending the solution to an undesirable set of values. To validate the Newtonian method and to investigate the unusual behavior, a second minimization approach has been used to minimize the error function. The Nelder-Mead Downhill Simplex method was employed [40-42]. A simplex can be thought of as a shape with N+1 vertices, where N is the dimension of the problem (N variables – for a 2-D problem, the simplex would be a triangle). The Nelder-Mead method evaluates the function at each vertex, and it would move the simplex away from larger values by reflecting, extending, contracting, or shrinking. A schematic of this behavior for a 2 variable system is illustrated in Figure 1-9 and Table 1-XI [41]. The simplex moves in a downhill fashion, in that it moves from the high values to the lower ones, changing its shape to move more efficiently towards a minimum. It is termed a "direct" method in that it does not use derivatives of the objective function as the Newton method does. This provides advantages in that it would not be adversely affected by the topography of the solution, such as large gradients. It is not an efficient process, requiring a large number of iterations to locate a minimum, but is easily handled by a PC – usually taking only a few seconds to a few minutes. This was on the same order as the Newtonian method, if not quicker, but used more iterations (several thousand for NM vs. less than 100 for the Newtonian method). The difference in efficiency between this method and gradient methods is in the operation. The Nelder-Mead method moves to decreasing values by

moving away from higher values rather than moving in a straight line towards a minimum. The algorithm for the Nelder-Mead method was taken from Numerical Recipes [42, pages 408-412], and applied to minimize the sum of squared errors from the Forward Solution.



A B

Figure 1-9. Schematic of the Nelder-Mead Simplex Method for two variables. A) Potential actions of the simplex (Table 1-XI). B) Movement of the simplex through 2-D space.

Table 1-XI. The different potential actions of the Nelder-Mead method for the simplex illustrated in Figure 1-9 A.

Starting Simplex	Action	Resulting Simplex
ABC	Reflect	BCE
	Reflect and Extend	BCF
	Contract	BCG
	Reflect and Contract	BCH
	Shrink	A'B'C

Another advantage of the Nelder-Mead method is the ease of which constraints can be applied to minimize undesirable minimums from being found [40,41]. A simple constraint is to require all elastic constants to be positive and within acceptable bounds (solutions exist

containing negative values or, say, Poisson's ratios much larger than 1). If during the execution of the program, a vertex lands on a value violating a constraint, the associated error value is made very large (very large means larger than any other error value). This will cause the simplex to move away from that value, and toward more meaningful solutions.

The development of these procedures allows for an added benefit in characterizing the strength of the materials under multi-axial stress conditions. The testing of composite tubes allows for the generation of complex stress states without the need for different orientations of the composite laminates by application of differing levels of axial load, torque, and internal pressure. The added benefit of no edge effects, as seen in planar coupons, should allow better characterization of different failure conditions. The elastic solution described allows for the calculation of the stress state in the material at all points through the thickness. It is not expected that the results in this research will allow for research to be done on multi-axial failure theory or even the calculation of failure envelopes for the different materials tested, due to an insufficient number of available sample failures, but it does provide a framework that would allow for both of these if more data could be obtained. The test procedures will allow for the generation of complex load states, and the analysis should determine the elastic properties of the material. The elasticity model will allow for the calculation of the stress states for the materials at each of the failure levels.

2 Experimental Procedures

As was noted in the first section, the goal of this work is to gain as much information as possible about the mechanical properties of composite tubes from a few tests. The procedures were chosen so that they would generate an axisymmetric load condition, so that the data could be used in a non-linear regression analysis. Due to the different properties of each material and some improvements made as the testing was conducted, there are differences in the procedures used for each sample, which will be described in this section.

2.1 Test Facilities

All the tests described were performed using MTS servohydraulic load frames. System control and the load profiles used the TESTSTAR II software from MTS. The initial tests conducted on the MTI materials used an external data acquisition system. A computer with an 8-channel data acquisition board was used to record six channels from the sample, and 2 (load and displacement) channels from the MTS controller. Later tests utilized a MTS frame with extended data acquisition capabilities, allowing for the recording of all the pertinent information (up to 14 external channels plus all internal load and displacement readings) during the tests by the TESTSTAR II software. This was advantageous since now all the controls and data were handled internally, greatly simplifying testing.

For the samples that fit directly into the frame (ORNL and the control samples), special collets were used to grip the samples. Due to the large size of the samples, two 63.5 mm (2.400 in) round collets were developed to grip directly on the outer surface of the sample. For other samples that either couldn't support the large compressive stresses in the collets or would not fit, different load fixtures were developed to transfer load from the frame to the sample. These were designed to fit into the 38.1 mm (1.500 in.) round collets from MTS.

2.2 Test Procedures

During the testing of the different materials, three different procedures were used for the axial tests (axial tension, compression and torsion), while only one was used for the internal pressure tests. Most were made to accommodate the different material properties of the different specimens, but the changes in the axial procedures for the material from Honeywell was due to improved data acquisition accommodations. The specifics of the procedures and rationale behind the changes are listed in the following sections.

2.2.1 MTI Hot Gas Filter Axial Tests

Due to the low strength of the material, other methods, such as gripping the sample in the MTS grips, was not possible. Due to the large number of samples expected from MTI, a fixture that would be interchangeable from sample to sample was needed. To accommodate this, the MTI materials were pin loaded for the axial and torque tests, and used a test fixture to transfer the load from the MTS frame to the pins placed through the ends of the tube. The axial samples were 18.5 cm (7.25 inches) in length with a pin spacing of ~12 cm (5 inches). A schematic of the axial test samples is in Figure 2-1. The effects of the pinholes and concentrated load points were removed by potting the end in epoxy. This locally strengthened the material and distributed the load from the pin. Initially, the end plug used a bonded ceramic foam plug to distribute the load away from the pins, with the hope of doing tests at elevated temperatures. During some of the early strength tests, failure occurred by cracks propagating from the pinhole and winding around the sample. The plug was not removing the stress concentrations at the hole, so the ends were cast in epoxy. With the new configuration, the sample experienced gage section failures.

To cast the ends, the outer surface was wrapped in plastic to prevent leaking, and filled with the liquid epoxy to a level even with the loading pins. The epoxy hardened level with the top of the pin, with a small amount of wicking into the sample (as can be seen as the dark region on the sample above the pin level in Figure 2-1). Dissection of failed samples revealed the epoxy permeated only as far as is visible on the outer surface (no permeation into the gage section). This gave an approximate gage section of 10cm (4 in).

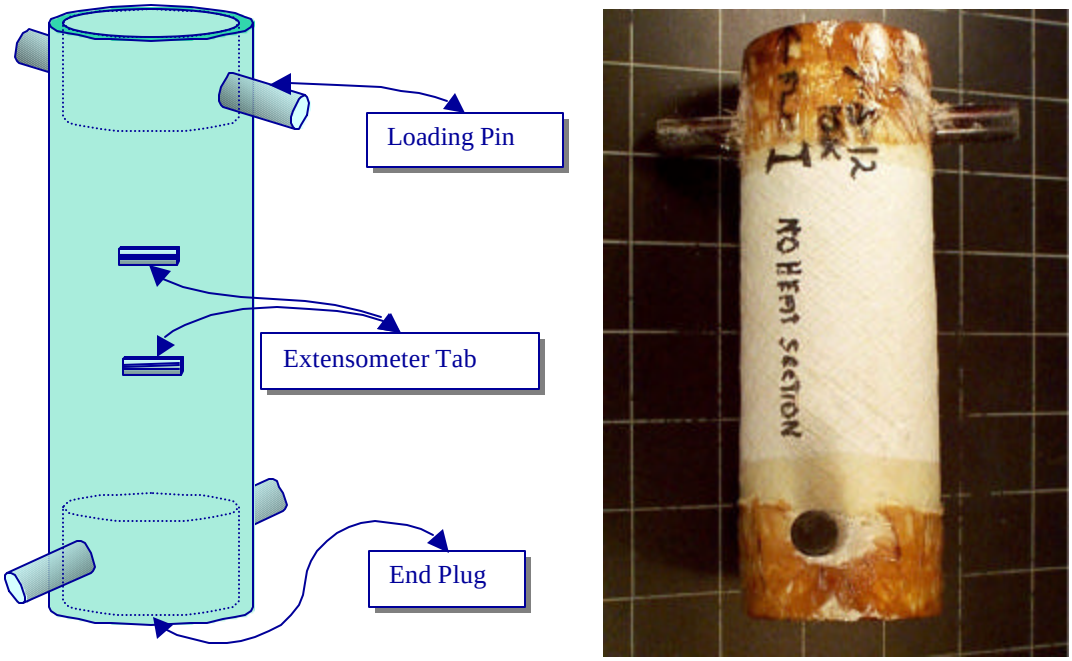


Figure 2-1. McDermott filter axial test sample.

Strain measurements were made using both extensometers and strain gage rosettes. The axial stiffness calculations were based on extensometer measurements, while the rosettes were used for the shear strain values. Strain gage data were on the order of 15% stiffer than the extensometer measurements for the axial tensile strain, and is believed to be due to the adhesive permeating and locally reinforcing the matrix. It has been shown that the axial stiffness is predominantly controlled by the matrix properties, while the torsional stiffness is more closely related to that of the reinforcing fiber properties (and experiences little to no influence from the local reinforcement) [2]. This is expected since the material is a ±45° woven structure, which would allow the fibers to bear the applied torque. The load ranges for elastic testing were ±445N (±100 lb) in tension/compression and ± 22.5N-m (±200in-lb) in torque. The tests were run under load control, at a rate of 45 N/sec (10 lb/sec) in tension, and a rate of 2.5 N-m/sec (20 in-lb/sec) for the torsion tests. The tensile strength tests were run in displacement control at a rate of 0.64 mm/sec (0.025 in/sec) to a maximum range of 6.4 mm (0.25 in.) for the early samples, and a maximum range of 25 mm (1 in.) for the later tests.

2.2.2 Al_2O_3/SiC from ORNL and Control Sample Axial Tests

The test procedures for the ORNL and Control samples were identical. Both materials exhibited sufficient strength to survive direct gripping into the MTS collets. In order for the samples to fit, the outer diameter needed to be 2.400±0.001 inches (tolerance listed for the MTS grips). Both materials were smaller than 2.4 inches, so the outer surfaces were coated with epoxy and machined down to tolerance. The initial tests with the SiC samples revealed that the material, if left unsupported, would be damaged by the grip pressure and cause grip induced failures. In order to provide some support steel inserts were made that closely matched the inner surface of the materials (Figure 2-3). With the inserts bonded in place by epoxy, the outer surface could be built up with the potting epoxy and machined to tolerance. This procedure was repeated for the remaining SiC materials from ORNL and the steel control sample.

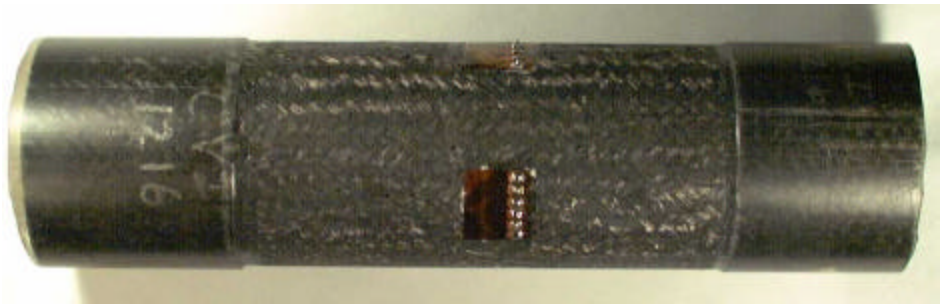


Figure 2-2. Sample CVI 1216 axial sample

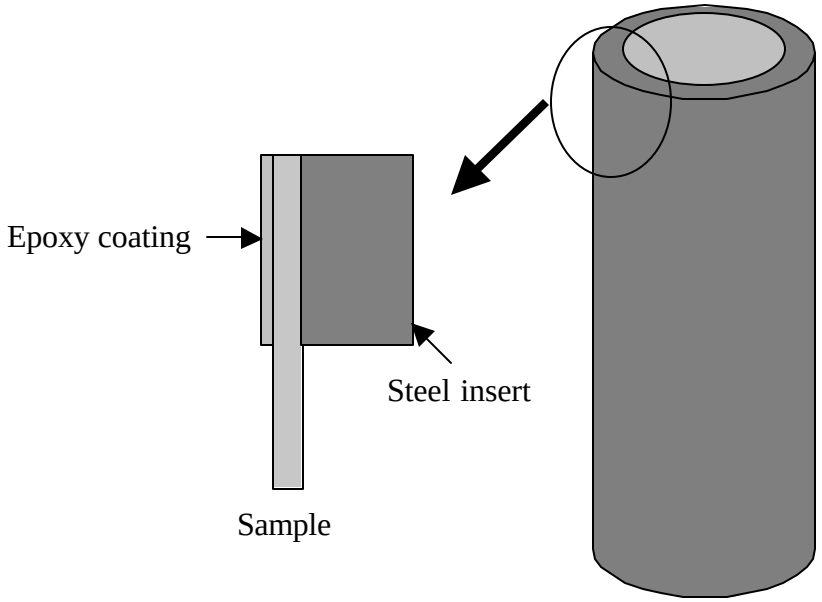


Figure 2-3. Schematic of ORNL and Control axial samples

2.2.3 Honeywell Axial Tests

The samples from Honeywell were tested with a different approach from the other materials. The diameter of this material was sufficient to allow access to the inner surfaces for strain gaging, which provides more information for the analyses used later in this work. This precluded the other approaches to loading the samples, since they would not allow the wires from the strain gages to exit the sample and reach data acquisition boards. In order to allow access to the inner surfaces, caps were made to fit on each end of the materials, as shown in Figure 2-4 through Figure 2-6. The fixtures had a groove cut into each end that would fit the sample and a small gap for an epoxy layer. Approximately seven to ten thousandths of an inch was allowed on each side of the sample for the epoxy layer. Four holes drilled through the end caps allowed the strain gage lead wires to exit the sample. Shims were used to center the tube in the caps when the epoxy was placed in the sample, to assure a uniform epoxy thickness. Figure 2-7 contains photographs of a sample from Honeywell.

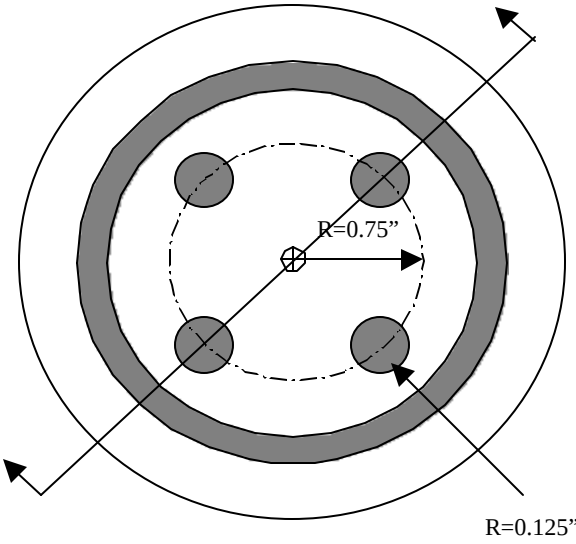


Figure 2-4. Top view of Honeywell loading fixture

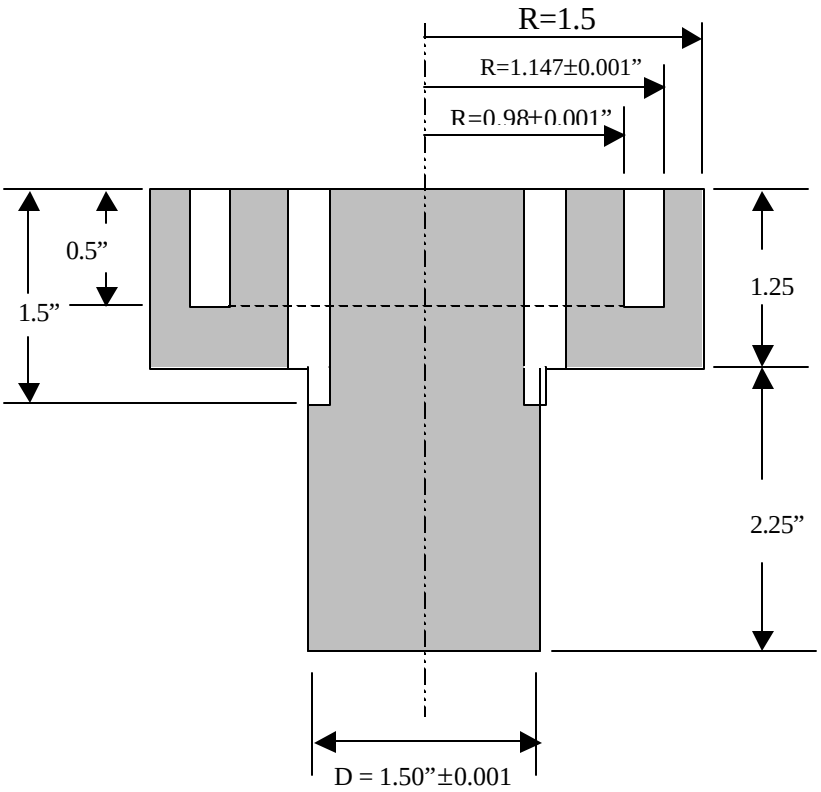
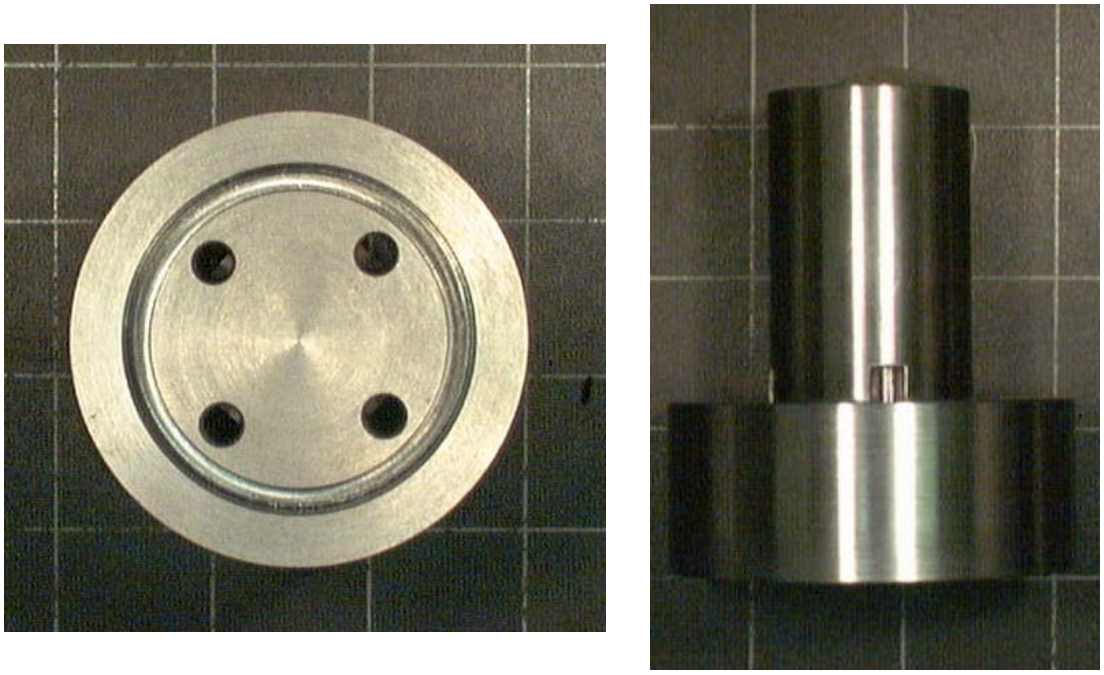


Figure 2-5. Cut-away of the Honeywell fixture



Top View Side View

Figure 2-6. Photographs of the Honeywell fixture

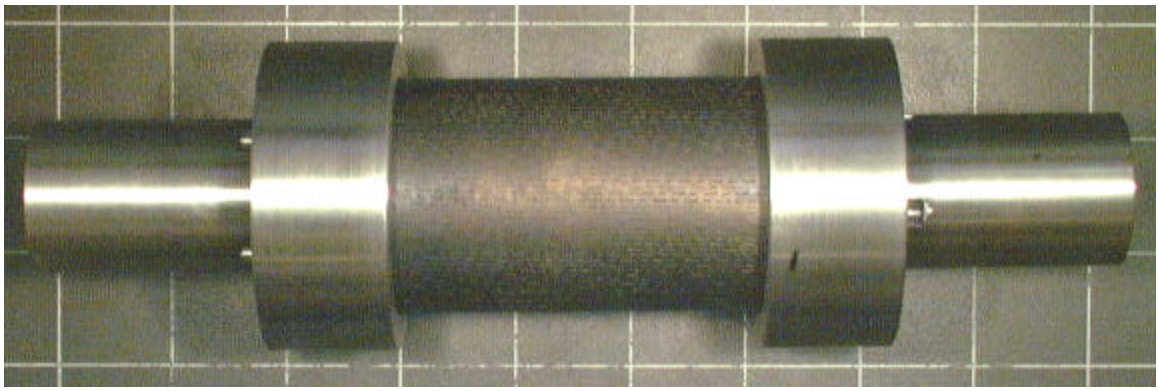


Figure 2-7. Honeywell axial sample mounted in loading fixture

2.2.4 Internal Pressure Testing

Internal pressure testing is a common method used for characterizing composite tubes [1-4,34-36,44]. The method generates stress in the material primarily in the hoop direction of the sample. This provides more information on the properties of the material when used with axial tests. In this work, the compressed rubber plug technique as described in Singh, *et al*, was utilized for the internal pressure tests [34,35]. A schematic of the test is found in Figure 2-8. Poisson expansion resulting from compression of a rubber plug generates pressure on the inner surface.

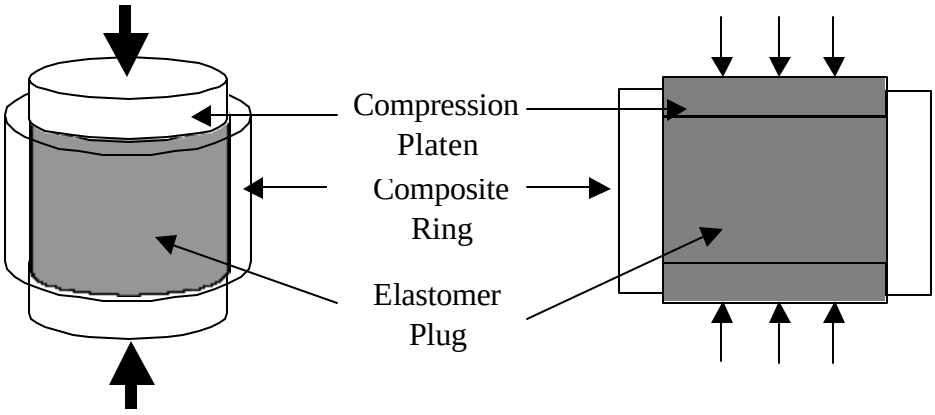


Figure 2-8: Schematic of internal pressure test with a cut-away

The pressure generated at the inner surface of the sample is related to the applied stress in the material. To analyze this situation we start with the Hooke's Law expression for the strain in the x direction (the axis of the cylinder), and simplify by applying two conditions: $s_r = s_q$ in this axisymmetric condition where $ó_r$ is the internal pressure to the sample [43, pages 383-384]. Using these conditions in the Hooke's Law expression

$$e_x = \frac{1}{E_P}\left[s_x - n\left(s_r + s_q\right)\right] \tag{2.1}$$

we obtain

$$e_x = \frac{1}{E_P}[s_x - 2n s_r] \tag{2.2}$$

For the internal pressure test, $\sigma_r = -P_i$, which allows for the internal pressure calculation:

$$P_i = \frac{E_p e_x - s_x}{2n_p} \tag{2.3}$$

Where P_i is the internal pressure, σ_x, E_p, ε_x and ν_p are the compressive stress, Young's modulus, axial strain, and Poisson's ratio for the plug material. This expression is derived from an elasticity solution for an isotropic, linear elastic material. For an incompressible, linear elastic material (í = 0.5), the expression in Equation (2.3) simplifies to:

$$P_i = E_p e_x - s_x \tag{2.4}$$

or

$$P_i = s_x^E - s_x \tag{2.5}$$

where s_x^E is the elastic stress for a given ε (Eε for linear elastic materials under small deformations). It is important to note that due to the large deformations used for these tests (30 to 60% strain, on average), the engineering values for stress and strain are no longer correct, and true stress and strain values should be used for this procedure. These values are found from the engineering stress and strain values by:

$$\begin{aligned} \tilde{e} &= \ln(1+e) \\ \tilde{s} &= s\,(1+e) \end{aligned} \tag{2.6}$$

where $\tilde{s}$ and $\tilde{e}$ are the true stress and strain and σ and ε are the engineering stress and strain [45, page 162] A plot of the stress-strain behavior of Dow Corning Silastic silicone rubber for a compression test is included in Figure 2-9. The two lines are the engineering and true stress strain curves. Linear regression analysis of the true stress/strain values yields a slope of 2.05 MPa (298 psi) and an $R^2 > 0.99$. Finding the Young's modulus in this way yields a different stress strain relation:

$$E = \frac{\tilde{s}}{\tilde{e}} = \frac{s\,(1+e)}{\ln(1+e)} \tag{2.7}$$

or

$$s_x^E = \frac{E\ln(1+e_x)}{1+e_x} \tag{2.8}$$

By combining Equation (2.5) and (2.8), the expression for calculating the pressure from the stress measurements for the internal pressure test procedure is:

$$P_i = \frac{E_p \ln(1+e_x)}{1+e_x} - s_x \tag{2.9}$$

The σ_x value is the applied stress to the plug and $\aa_x$ is the engineering strain.

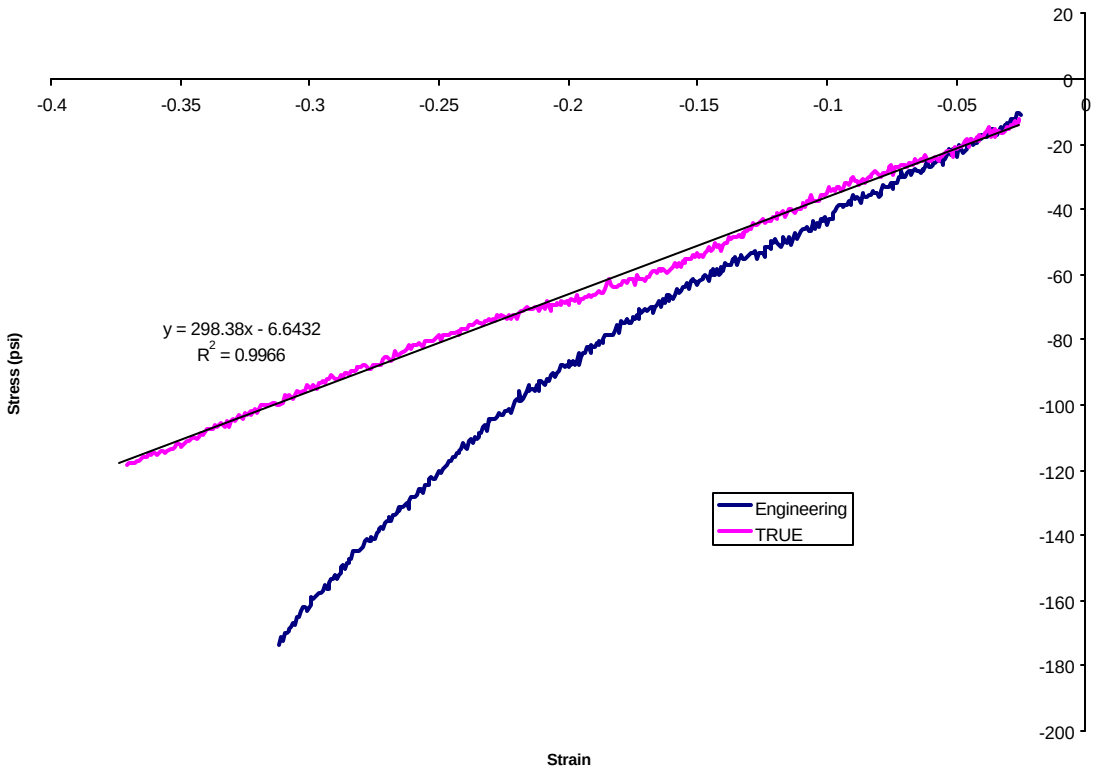


Figure 2-9. Compressive stress/strain curve for the Silastic plug. The curved line is the enginerring stress/strain and the other line is the true stress/strain plot with a best-fit line. The equation beside the best-fit line is its slope and R^2 value.

If the elastomer used for the plug is incompressible (i.e. has a Poisson's ratio of 0.5), and the test begins with the plug in intimate contact with the inner surface of the sample, a hydrostatic state is generated with the application of pressure. By this, the internal pressure would equal the stress generated by the applied axial force. For the tests performed in this work, the plug diameter was smaller than the inner diameter of the sample. This required a certain stress state in the material for contact to be made with the inner surface. By using Equations (2.5) and (2.8), the contact stress can be subtracted out to leave the pressure value.

Other research has been conducted independently on this procedure, using similar procedures and materials at Oak Ridge National Laboratory (46). Instead of viewing the Silastic materials as a linear elastic material, hyperelastic theory is used to describe the material response. This theory accounts for the nonlinear stress/strain behavior exhibited during large deformations of elastomeric compounds. There are several equations used to describe the behavior, but, for the Silastic compound, the Mooney-Rivlin equations are adequate to describe the deformation. The resulting expression to describe the deformation is:

$$s_x^o = 2\left(l - \frac{1}{l^2}\right)\left(C_1 + \frac{C_2}{l}\right) \tag{2.10}$$

where σ^o is the stress value using the original cross sectional area, C_1 and C_2 are constants fit to the data of an unconstrained compression test, and ë=1+å [47, pages 309-312]. Again, the pressure term is calculated by subtracting the contact stress value from the measured axial stress, as shown in Equation(2.5).

To verify if this method is accurate in determining the internal pressure, the stainless steel control sample was used since it allows for the calculation of the pressure by different methods. For the test, four strain gage rosettes were placed around the sample recording the hoop strain so the pressure could be calculated from the equations for a Lamé cylinder solution [43, pages 68-71]. The internal pressure was calculated using the linear elastic and hyperelastic equations, and they were checked by a pressure value found using a Lamé cylinder. The hoop stress at the outer surface by the Lamé cylinder solution is:

$$s_q = \frac{2r_i^2 P_i}{r_o^2 - r_i^2} \tag{2.11}$$

where r_i and r_o are the inner and outer radii and P_i is the internal pressure. Rearranging (2.11) to solve for the pressure value gives:

$$P_i = \frac{E_S e_H \left(r_o^2 - r_i^2\right)}{2r_i^2} \tag{2.12}$$

with E_S being the Young's modulus of the steel and ε_H is the hoop strain. Figure 2-10 shows the pressure as a function of time during the test. The three values are nearly superimposed onto each other.

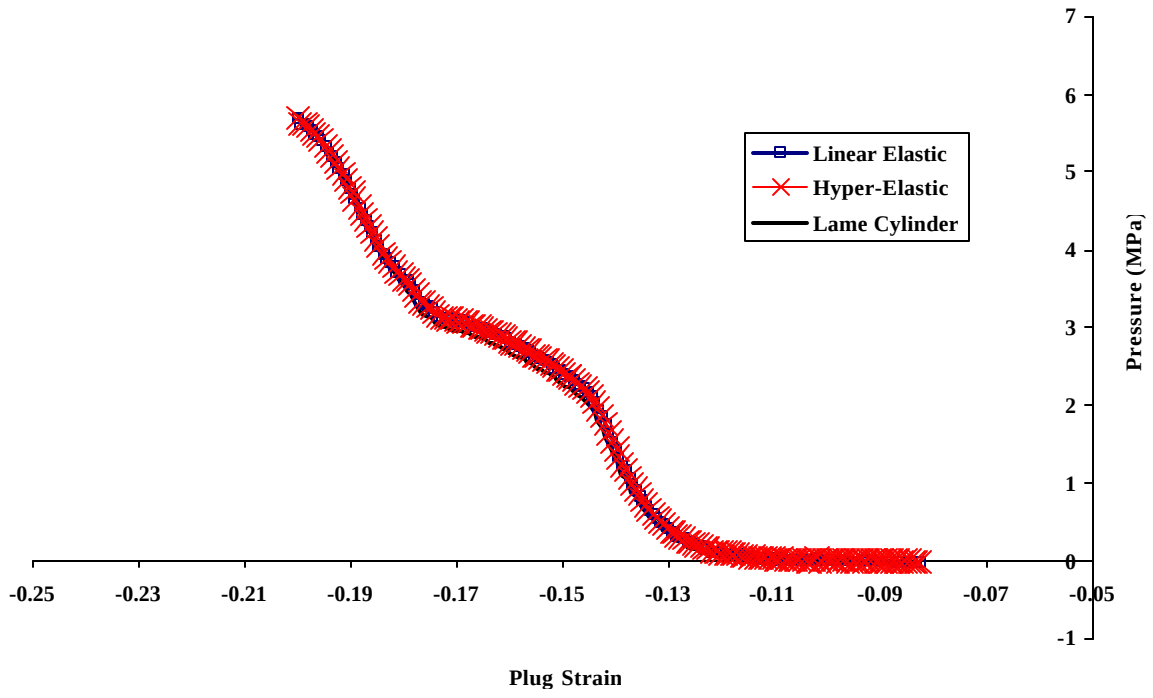


Figure 2-10. Results from the internal pressure test for the control sample. The different lines represent the pressure values found using the Linear Elastic, Hyper-Elastic and Lame cylinder solutions

2.2.4.1 Internal Pressure Tests for the McDermott Filters

For the MTI materials, the tests were conducted using a Dow Silastic T-2 silicon rubber cylinder. The Silastic liquid was cast into cylinders that were nearly the same height as the sample and slightly smaller than the inner diameter. This allowed for the initial contact between the compressed plug and the inner surface to occur when the plug was shorter than the sample, in order to prevent early failure in the tube due to edge effects. The tests were conducted with a layer of lubricant between the elastomer plug and the sample to prevent frictional loading of the sample during the compression. This condition would create a biaxial loading condition that couldn't be accounted for in the analysis of the material properties. Due to the porous nature of the MTI materials, the lubricant layer was sandwiched between to polymer sheets (Saran Wrap). This sandwich prevented the lubricant from penetrating the material and potentially affecting the tests. That sample did not exhibit any movement during the tests (this is not the case for all materials, as is described in the next section). Figure 2-11 is of a MTI sample during an internal pressure test. The sample is positioned with the plug being the same length of the sample, but is slightly smaller than the inner diameter. This positioning allows the plug to be slightly shorter than the sample when contact is made, preventing the loads to be generated at the edge of the sample, where edge effects would affect the outcome of the test by a premature failure that would not be indicative of the material strength.

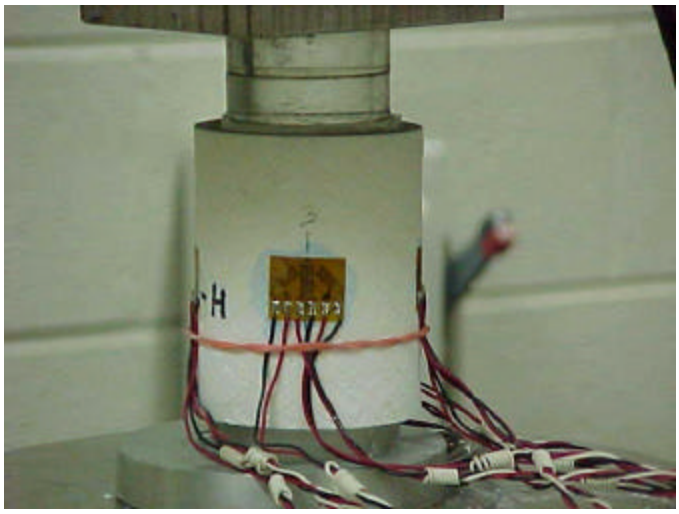


Figure 2-11. Internal pressure test for the McDermott filter

2.2.4.2 Internal Pressure Tests for the ORNL, Honeywell, and Control Samples

The elastomer compound used for these tests was Dow Corning Silastic silicone rubber (Silastic T-2). Initial work with this material was unsuccessful, due to the material failing by shearing out around the compression platens at high pressures. Later attempts used a urethane rubber compound, which exhibited better shear properties. Difficulties arose with accurately predicting the pressure generated due to the nonlinear elastic and viscoelastic properties of the urethane. The Silastic compounds were re-examined and found to exhibit linear elastic behavior with no appreciable viscoelastic effects. The compression platens were made to match the inner diameter of the sample more closely, which eliminated problems with the shear deformations and failure of the plug.

A high-pressure lubricant is applied to the plug, plunger, and inner surface of the sample to minimize compressive loads generated by friction and generate a more uniform internal pressure. After each test, the plug and sample are inspected for any damage associated with the test. To date, the tests have not been run to a level sufficient to damage the SiC composites, or to yield the steel. By improving the fit of the compression platen to the inner surface of the sample should allow for increased pressure ranges. This would keep the stress in the plug to a true hydrostatic compressive stress, instead of the unpressurized area around the edges where the large shear stresses develop. The presence of the gap for the experiments in this study was due to the roughness of the inner surface for many of the samples. Because of this, the platens were made smaller than the inner diameter to prevent contact with the sample during the test. The platens used in this study were either plexiglass (for the MTI samples) or aluminum, so that if contact was made the platen material was softer than that of the sample and would not damage it. Again, the plugs were cast slightly longer than the sample with a smaller diameter, so that when the plug made contact with the sample, it would be shorter and not apply load directly to the edges. For these materials, a small frame was made to hold the samples in place so that the platens would be positioned properly to prevent loading to the edges of the samples

2.3 Data Reduction

With samples of this geometry, off-axis loading is a problem that has to be considered. Small sample misalignments can generate significant stress/strain variations around the sample. To remove the elastic contributions of the bending, strain was measured at four, equally spaced locations around the sample. By averaging the strain values, the variations created by off-axis loading may be removed. Figure 2-12 is a schematic of the variations to the strain values due to the bending contributions.

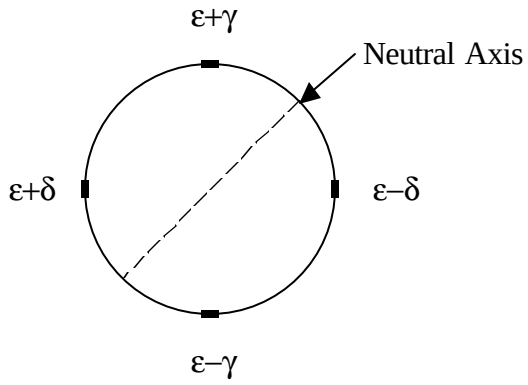


Figure 2-12. Schematic of the elastic off-axis loading response for the tubes.

Due to the number of strain measurements, at times all the channels could not be read simultaneously, so the tests were repeated with changes made to the location of the extensometer or active strain gage. When it was time to reduce the strain data, the data from the different locations would not be directly comparable since the load profiles would vary due to system noise and control. To change the data to the same load profile, the individual stress/strain curves were fitted using a 4^{th} to 6^{th} order polynomial regression line (6^{th} order is the highest order fit available in Microsoft Excel), depending on which best represented the data. Once the data were represented it could be calculated at the same values for averaging. When all the data could be recorded simultaneously, the values could be averaged without having to represent the data with best-fit lines. An example of this is in Figure 2-15 and Figure 2-16. The offset of the average line is due to the amplifiers for the strain gages being "zeroed" with a small load applied to the

sample. Due to the off-axis conditions and low loads for the filter materials, it was difficult to have this value equal zero. To correct this, the intercept value was dropped, shifting the data back to a line passing through the origin.

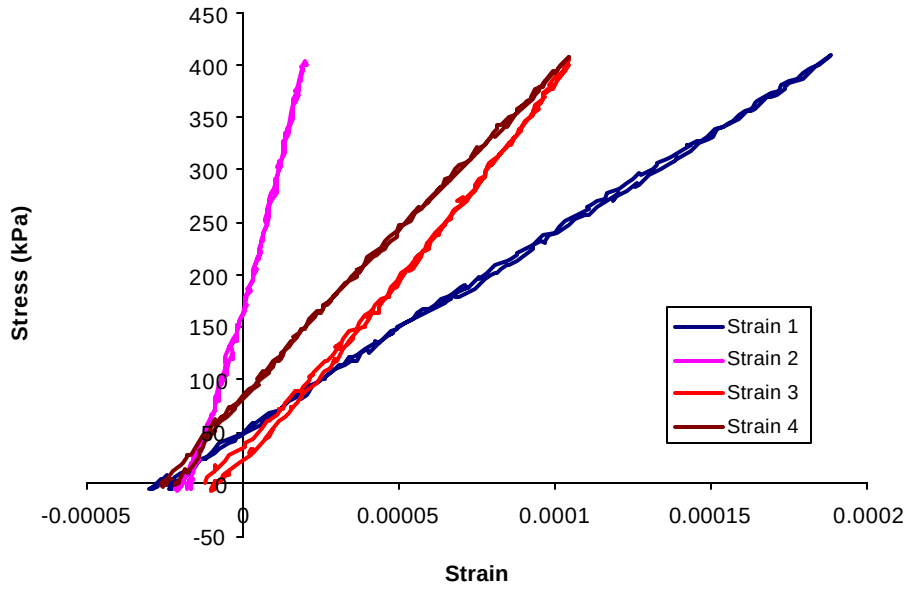


Figure 2-13. Axial tension results from 4 locations on the sample

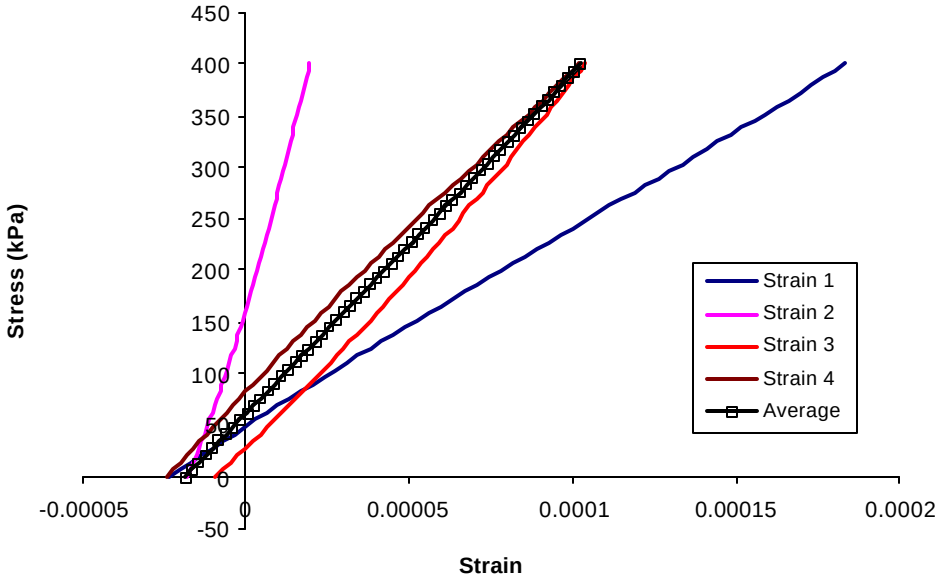


Figure 2-14. Best fit lines of the data in Figure 2-13 with the average result

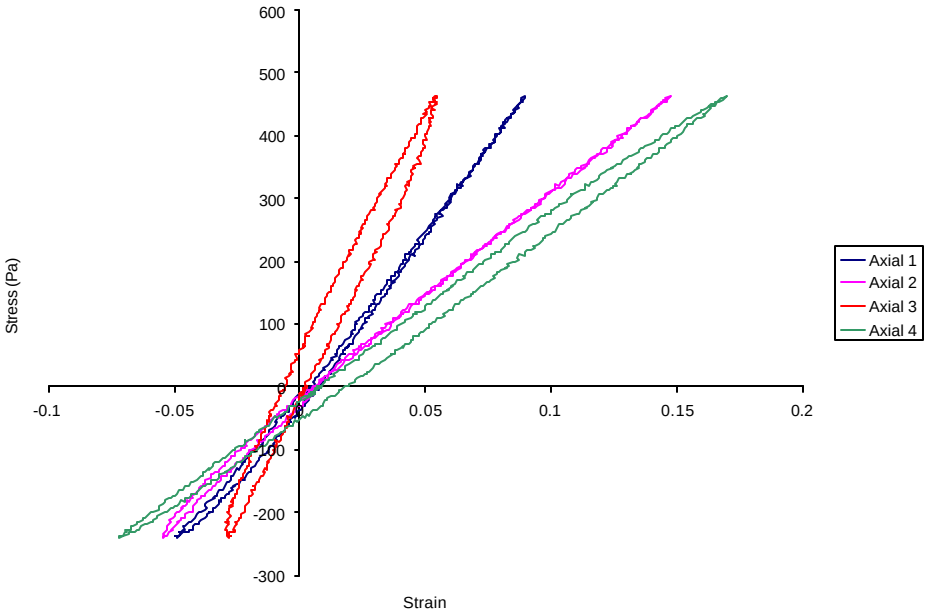


Figure 2-15. Axial strain measurements from the four gages around a McDermott filter sample.

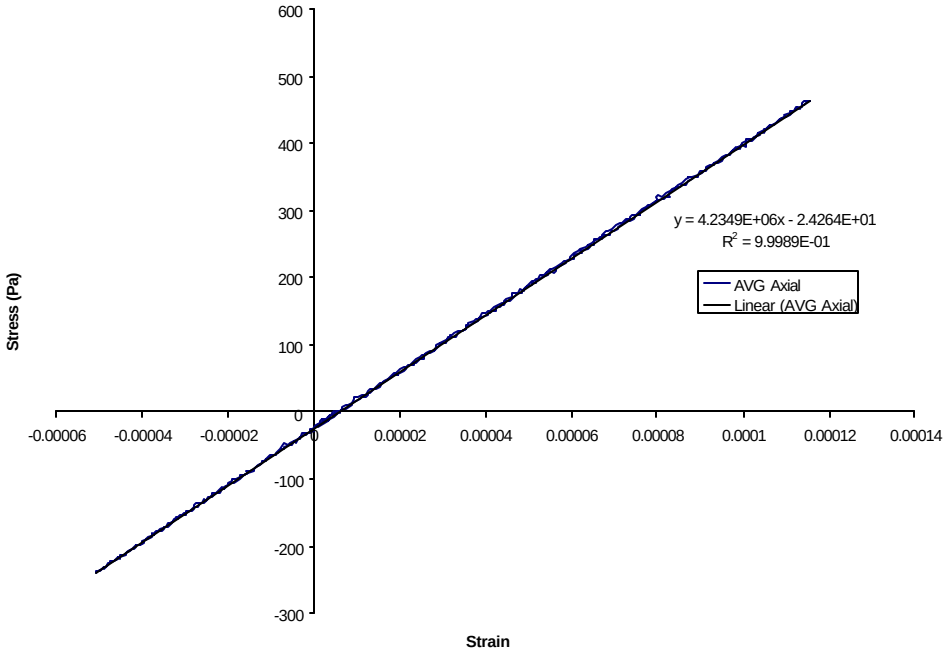


Figure 2-16. The averaged values for the axial strain values in Figure 2-15

2.3.1 Property calculation

With the data reduced to the average value for each strain component, values for the stiffnesses can be calculated. For each material, the global stiffness values were calculated using the nominal stress field, which is the applied load divided by the area for axial tension/compression, and shear stress at the outer surface divided by the measured shear strain for the axial torsion tests. The shear stress value was calculated by the expression for torsion of a cylinder:

$$t = \frac{Tr_o}{J}$$
$$J = \frac{1}{2}p\left(r_o^4 - r_i^4\right) \tag{2.13}$$

where τ is the shear stress, T is the applied torque, J is the polar moment of inertia for a cylinder, and r_o and r_i are the outer and inner radii.

The results for the internal pressure tests can be presented in two fashions, relating to either the pressure or estimated hoop stress. The pressure values are straight forward in that they relate the changes in the strain to that of the change in pressure. The hoop stress values are termed estimates since they are calculated from the internal pressure value through a Lamé cylinder model [43, pages 68-71]. The model used is for a homogeneous, isotropic material, which is inconsistent with the samples used in this study. The hoop stress values from this model are given by:

$$s_q = \frac{2r_i^2 P_i}{r_o^2 - r_i^2} \tag{2.14}$$

Linear regression of the experimental data was used to calculate the stiffness values. A line with two degrees of freedom (y=mx+b) was used to remove the offset values created by not having the zero for the strain gage amplifiers match the zero for the load cells (Figure 2-16 does not pass through the origin). Once a slope value is found, the offset value was discarded, shifting the resulting response back through the origin.

3 Experimental Results

The mechanical characteristics of the materials were measured using the procedures detailed in the previous section. Reported values will be those of the axial, shear, and hoop stiffness (estimated from the internal pressure tests). The slopes of the other strain terms will be included, since they are important to the analysis developed in the following section. The values are given with the standard deviation on the significant stress directions except for the internal pressure test results since the test is run only once on a sample (except for ORNL CVI 1219 – one repeat was performed). Tensile strength values are reported for the McDermott filter materials and some of the ORNL samples. The strength of one of the ORNL samples and the Honeywell material exceeded the capacity of the equipment or loading fixtures, so the plots are given with the largest load condition reached. The internal pressure burst values are only for the McDermott filters, since the elastomer plug experienced shear failures at pressures below the strength of the sample for the other materials. The plots for each of the internal pressure tests are for the largest pressure reached. Many exhibit nonlinear behavior that is attributed to either the plug failure process or frictional loading of the sample.

3.1 McDermott Technologies Candle Filter

Typical stress strain curves for axial tension and torsion of the McDermott filters are shown in Figure 3-1 and Figure 3-2. The axial and hoop strain lines in Figure 3-2 exhibit a large deviation from linearity between zero and 200 kPa. This nonlinearity is due to the pin loading fixture shifting as the load is applied, and this behavior was seen in several of the tests. It contaminates the data and renders the slope values meaningless, but was easily overcome by removing those values from the data set. By doing this, a good fit could be found, as can be seen in the low scatter in the axial and torsional stiffness results in Table 3-I, and plotted in Figure 3-3 and Figure 3-4. The average stiffness and standard deviation results are the calculated from the values from repeated tests on the same sample. The average slopes of the different strain components are listed in Table 3-II (important for use in the nonlinear regression analysis).

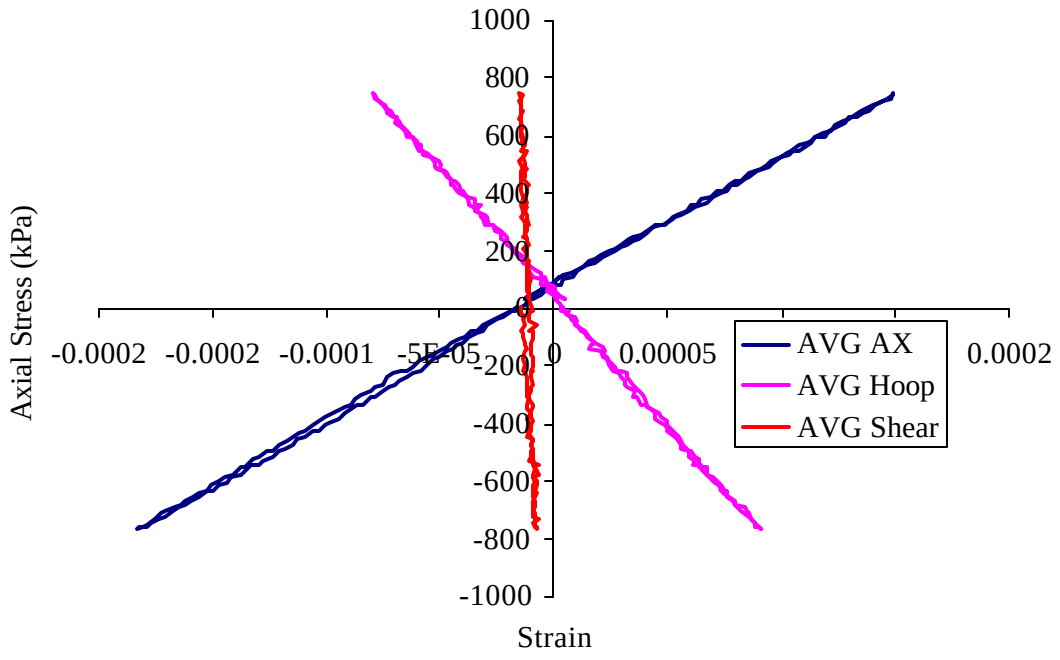


Figure 3-1. Typical axial tension/compression test results for McDermott filter

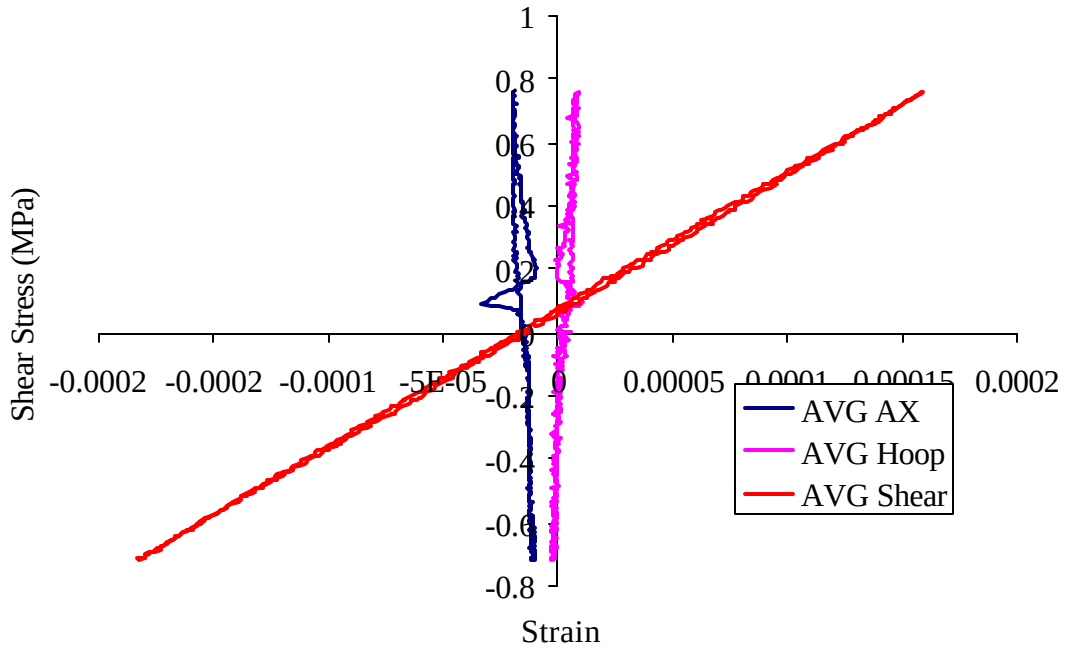


Figure 3-2. Typical torsion test result for the McDermott filter

Table 3-I. Axial and shear stiffness for the McDermott filter samples

Tube ID	Cycles	Tensile Stiffness (GPa)	STD DEV. %	Shear Stiffness (GPa)	STD DEV. %
7-2-28-3	0	3.300±0.055	1.669	7.581±0.183	2.420
7-2-28-4	0	3.378±0.074	2.183	7.855±0.216	2.755
7-5-13-C	10^3	3.441±0.004	0.125	--	--
7-5-13-E	10^3	3.324±0.101	3.033	7.853±0.084	1.064
7-5-13-NH	10^3	3.504±0.064	1.820	6.970	--
7-6-12-A	10^4	4.323±0.102	2.364	7.516±0.179	2.378
7-6-12-C	10^4	3.498±0.039	1.127	8.661±0.127	1.469
7-6-12-E	10^4	3.458±0.012	0.358	7.961±0.122	1.533
7-6-12-NH	10^4	3.244	--	8.567±0.081	0.950
7-6-16-C	10^5	3.848±0.061	1.574	9.428±0.106	1.124
7-6-16-A	10^5	3.260±0.029	0.895	8.986±0.058	0.642

Table 3-II. Slopes of the strain responses for different loading conditions – average values for strain gage data

Sample	Axial Strain GPa	Hoop Strain GPa	Shear Strain GPa	Poisson's ratio
7-6-12				
Axial Stress	4.45	-8.07	-224.46	0.55
Shear Stress	285.15	-430.01	9.04	
Internal Pressure	-1.13	1.26	29.24	1.11
7-6-16				
Axial Stress	4.40	-8.53	162.80	0.52
Shear Stress	-272.96	748.69	8.16	
Internal Pressure	-1.07	1.26	-89.67	1.18

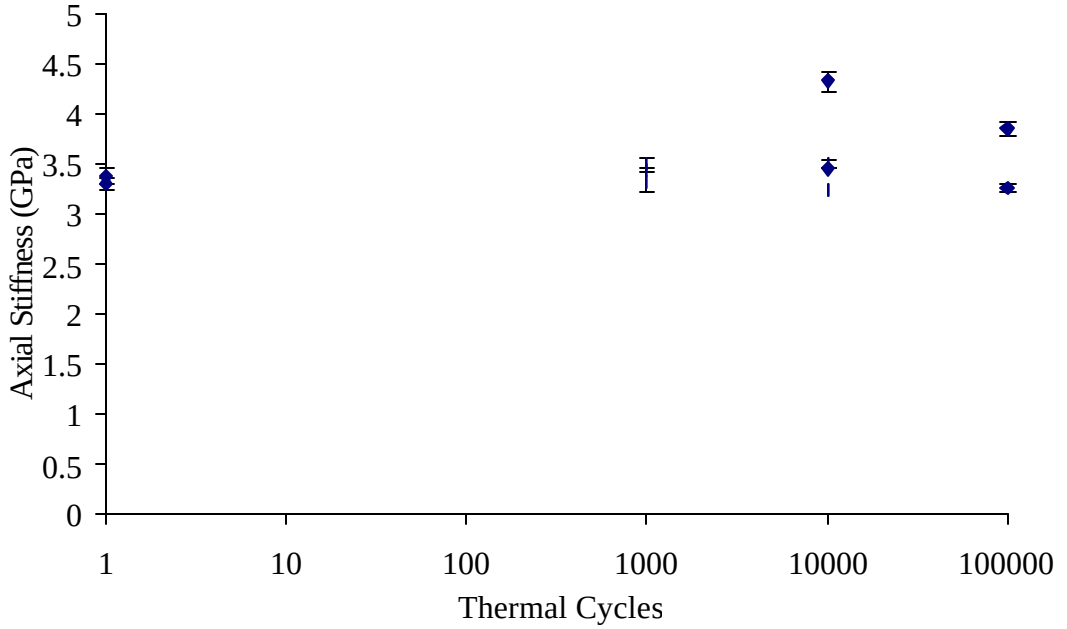


Figure 3-3. Axial stiffness after exposure to simulated back pulses.

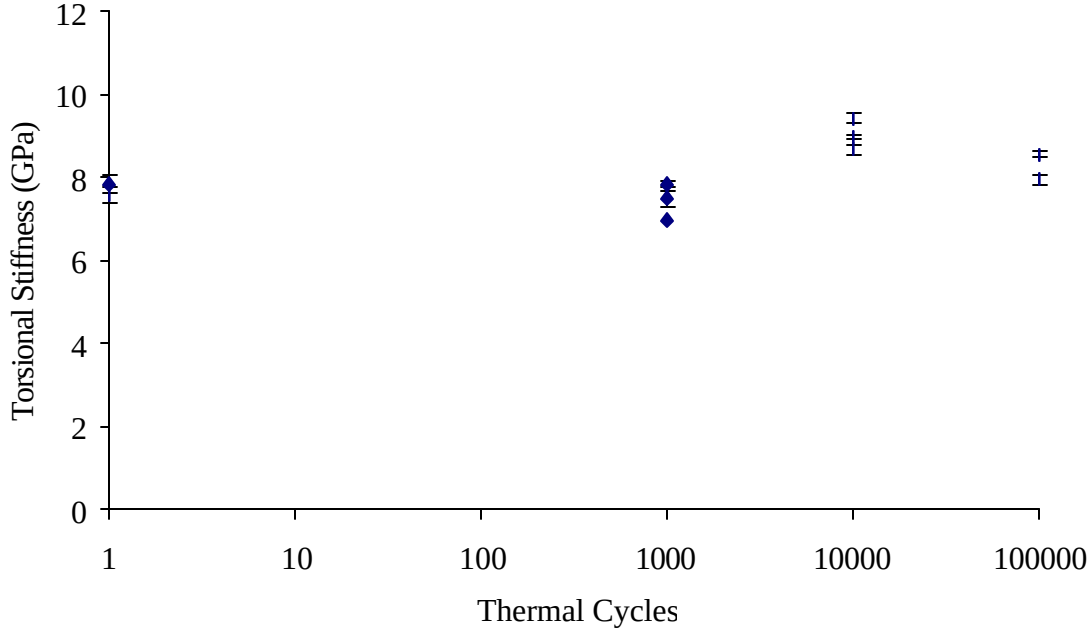


Figure 3-4. Torsional stiffness after exposure to simulated back pulses

Neither the axial stiffness nor the shear stiffness are adversely affected by repeated thermal shock of the simulated back pulse cleaning cycles. The variations in the torsional stiffness are probably batch-induced variations as suggested by the way the data are clustered (each tube is an independent "batch" – and only one tube was subjected to the different thermal cycling). A plot of the effect of the back pulse cleaning cycle on the tensile strength is in Figure 3-5. The figure contains data for the samples that experienced gage section failures with exposure to the 1 to 100 thousand cycle samples. The data for the as-fabricated tubes were not included since it used the ceramic foam end plug, which exhibited crack initiation from the pin holes (not a gage section failure).

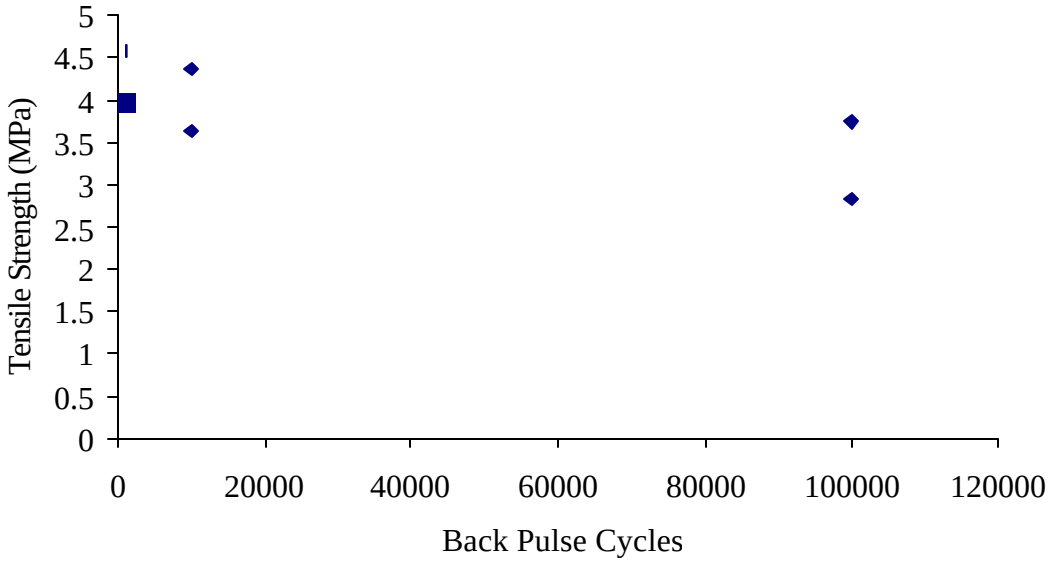


Figure 3-5. Tensile strength with back pulse exposure. The square data point is the sample that was not exposed to high temperature.

The decrease in tensile strength is on the order of 20%, but with the small number of samples that experienced gage section failure and significant scatter in the results, it is difficult to attribute this to changes in the material. No changes in the failure mode were observed. Failure begins with the formation of a shear band that follows a fiber tow around the sample. The fiber tows remain intact with the majority of the deformation occurring in the matrix material (schematic illustration in the results for the internal pressure test - Figure 3-12). Occasionally,

the shear band would lead to a large crack that would permeate through the wall of the tube, but would be bridged by the transverse tows, which would remain intact. This was common in the materials that exhibited failure due to crack initiation at the loading pin. For most failures, the damage was more evenly dispersed throughout the material, as seen in Figure 3-7. The load-strain curve for the tensile strength test for Sample 7-6-16-C is found in Figure 3-6. The strain values plotted are calculated using the displacement measured by the MTS frame, since the strain gage values are not useful past the initial failure event. The gage length was 10.2 cm with a final displacement of 2.54 cm (25% strain).

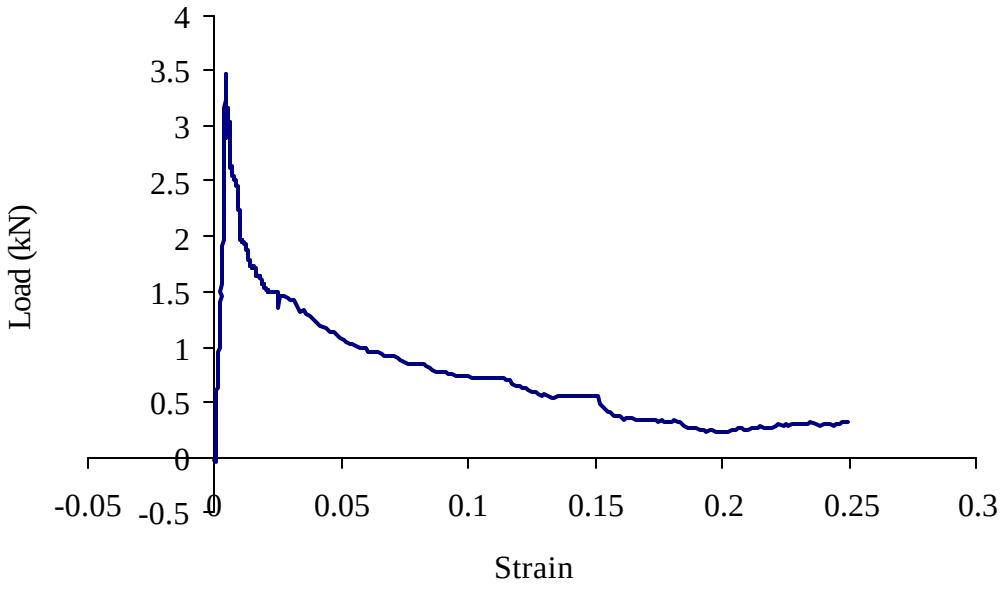


Figure 3-6. Tensile strength plot for Sample 7-6-16-C. Sample strain is calculated using a gage length of 10.2 cm.

A photograph of the failed sample is in Figure 3-7. The fiber tows are intact with the matrix material having failed.

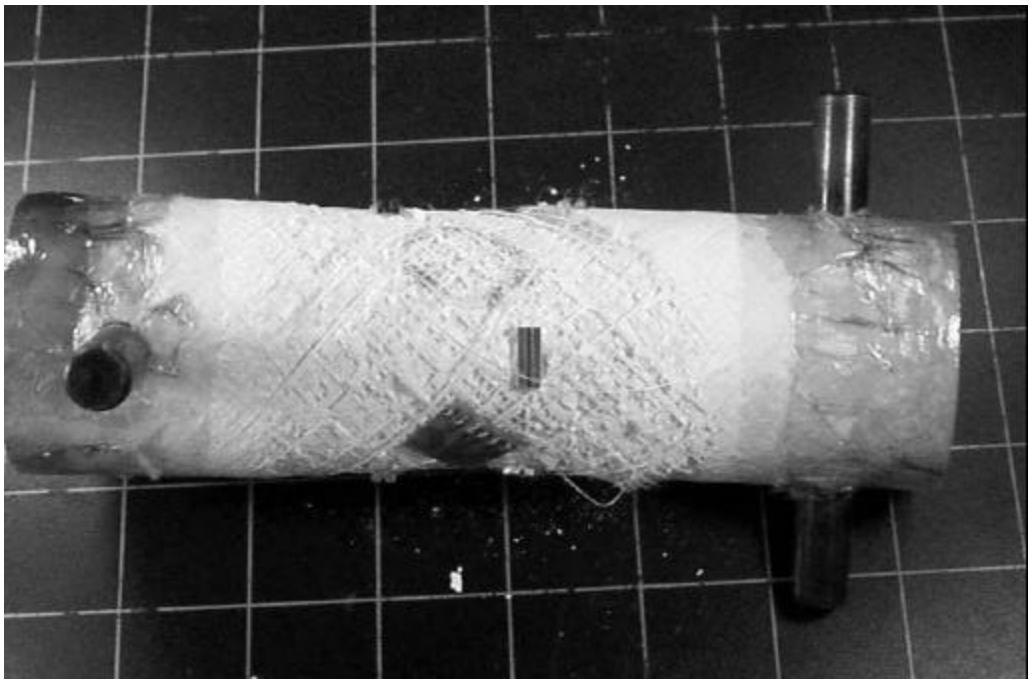


Figure 3-7. Sample 7-6-16-C after tensile strength test.

3.1.1 Internal Pressure Tests

A plot of the strain response during an internal pressure test for the McDermott filters is in Figure 3-8. The sample length used for these experiments was 7.6 cm, since below this sample length the burst pressure decreases significantly with decreases in length. When compared to a 25 cm sample pressurized using a water-filled bladder, this procedure yields comparable results, as can be seen in Figure 3-9 [36]. This may be expected since the material is extremely porous, and requires a certain length to remove the edge effects. By exploring the values for the 7.6 cm samples, the effect of back pulse exposure is negligible, as shown in Figure 3-10.

Slopes of the strain response are in Table 3-III. The values are the slopes of the pressure/strain plots, as seen in Figure 3-8. Included is the estimate of the hoop stiffness found using the Lamé equation described in the Materials and Procedures section. It is higher than the observed values for the axial stiffness. For a ±45º laminate, one would expect the axial stiffness

and hoop stiffness to be the same by symmetry, but the McDermott materials exhibited a variation in the winding angle, and the fibers are oriented at 50 degrees on the outer surface. Due to this, one would expect the hoop stiffness to be higher than the axial.

Table 3-III. Slopes of the pressure/strain response and the Lamé cylinder estimate of the hoop stiffness

Sample	Axial Strain GPa	Hoop Strain GPa	Shear Strain GPa	Hoop Strain/Hoop Stress GPa
7-6-12	-1.13	1.26	29.24	5.53
7-6-16	-1.07	1.26	-89.67	5.19

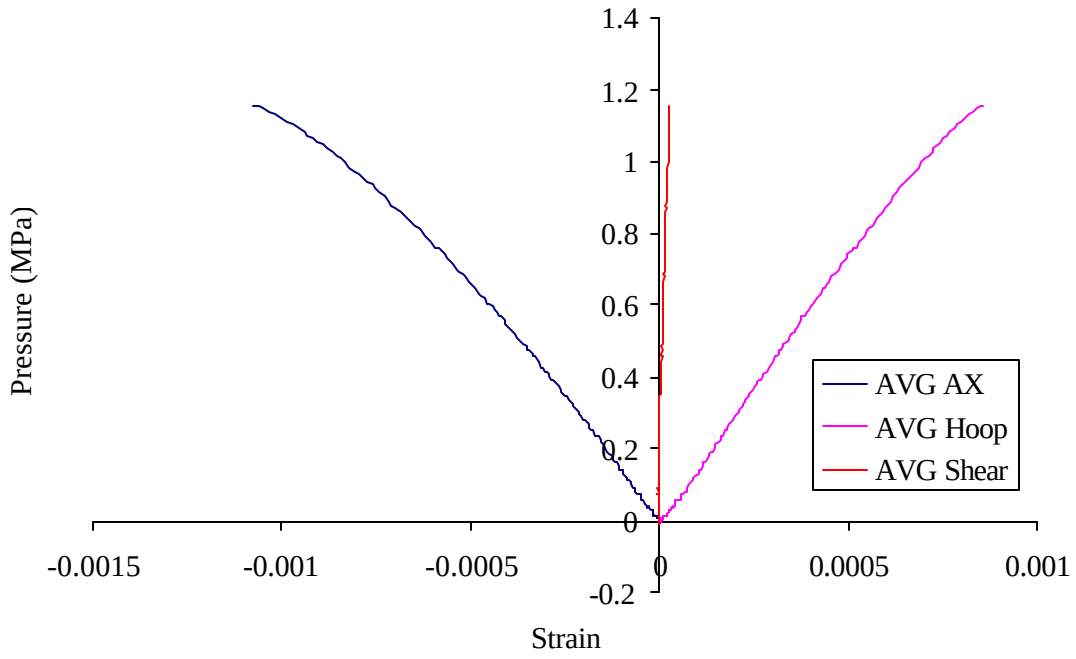


Figure 3-8. Typical internal pressure test for the McDermott filters

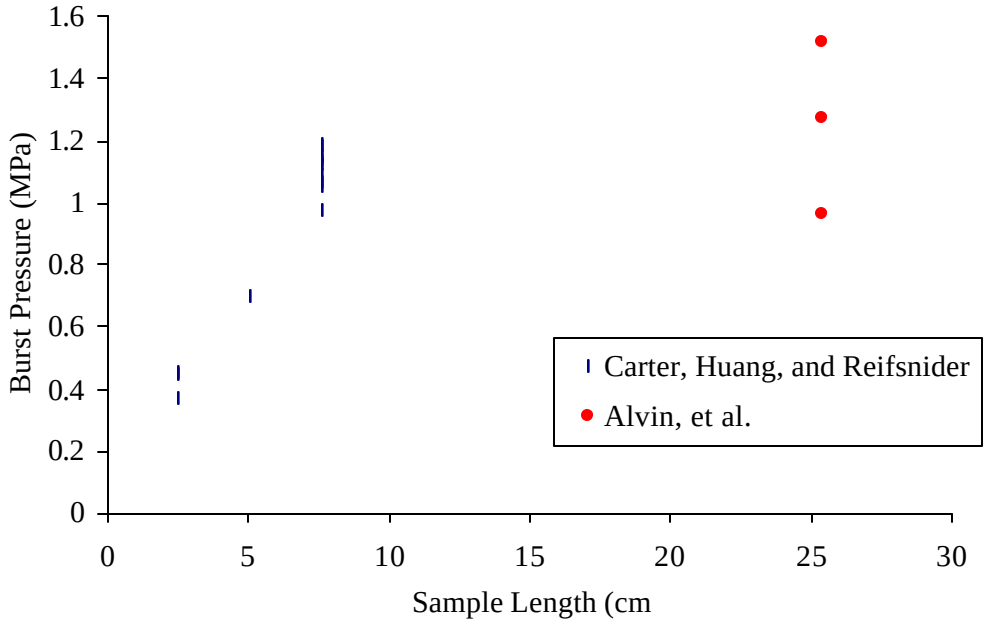


Figure 3-9. Burst pressures vs. sample length for the internal pressure tests of the McDermott filters. The round data points were found using a water filled bladder (Alvin, *et al*. [36])

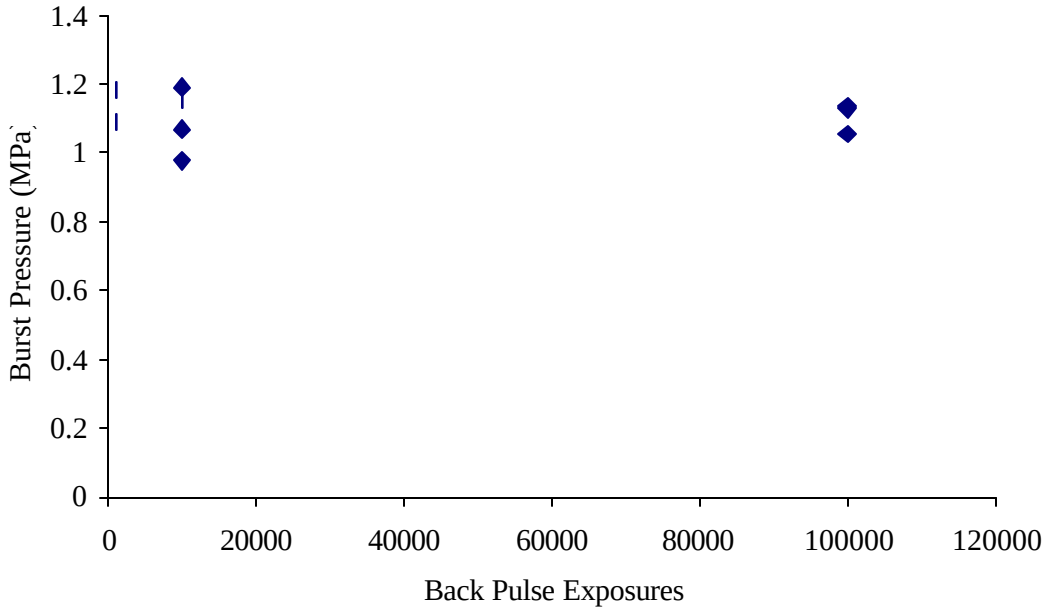


Figure 3-10. Burst pressure vs. back pulse cycle count for the 7.6cm samples

The material does not exhibit a significant change due to the simulated service conditions. As in the axial tension tests, the material did not exhibit catastrophic failure, but underwent a gradual increase in damage throughout the sample. There were no preferred failure sites or large amounts of crack initiation and propagation. A burst sample at different points of an internal pressure burst test is seen in Figure 3-11.

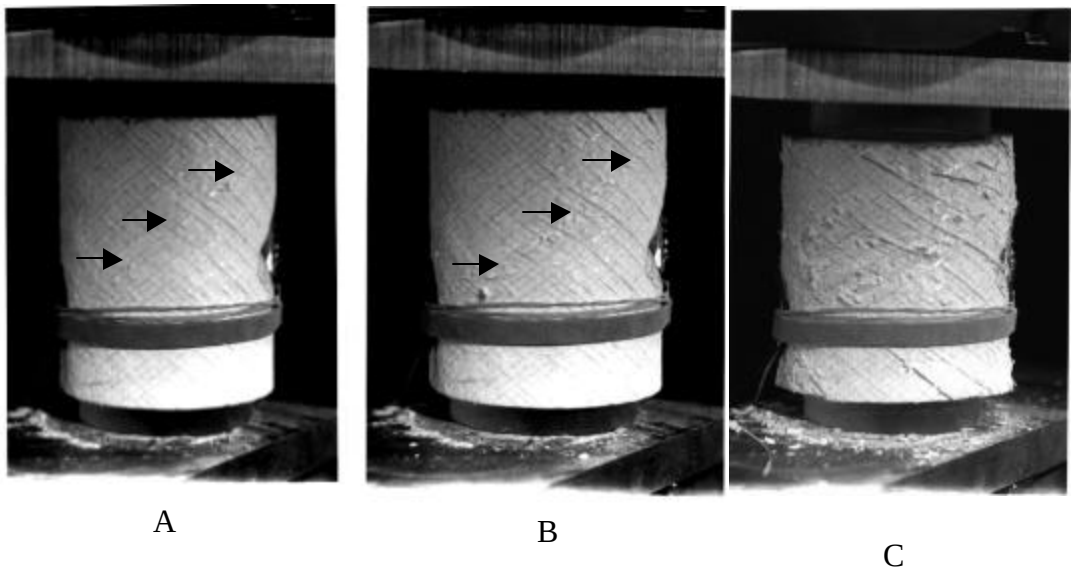


Figure 3-11. Failure of filter during an internal pressure burst test. A and B) Shear band formation C) Large scale deformation

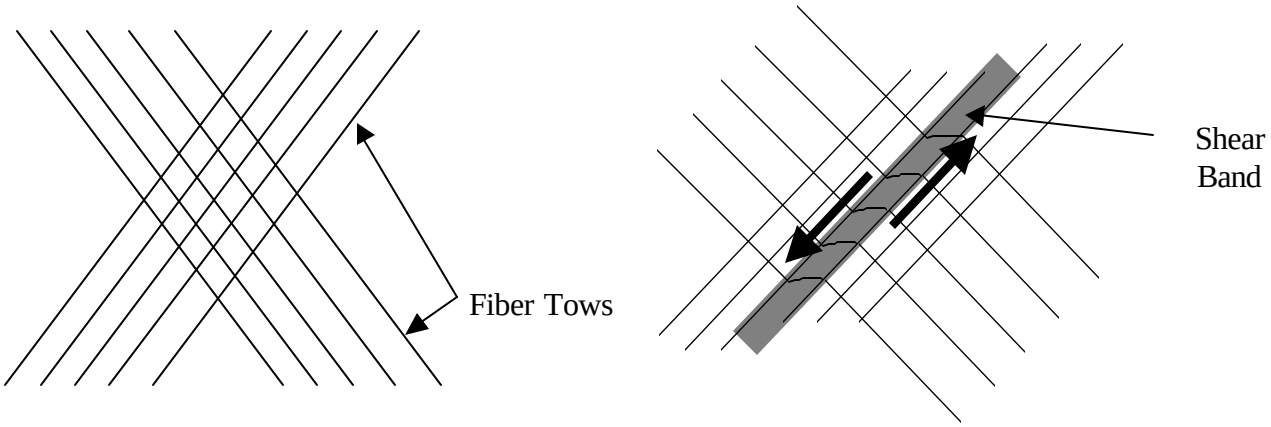


Figure 3-12. Schematic of shear band evolution. Normal tow structure (left) and shear band deformation (right).

Figure 3-11A shows the sample in the early stages of failure. A shear band has formed and is located along the tows at the end of the arrows. In Figure 3-11B, the shear band has become more prominent. The surface tows can be seen as they change orientation and debond from the filter. This behavior is described in the schematic deformations in Figure 3-12. The fiber structure on the right is the undeformed filter structure. Once strength of the material is exceeded, the structure deforms in a region between to adjacent fiber tows. The fiber tows remain intact, while the matrix material begins to crumble and fall off, and can be seen lying around the sample. Figure 3-11C is the same sample at large-scale deformation. The fiber tows are still intact, but are subject to large displacements and have debonded from the surface.

3.2 Fossil SiC/SiC materials

Typical axial stress/strain curves for the tubes from ORNL are located in Figure 3-13 (axial tension/compression) and Figure 3-14 (axial torsion), with the measured mechanical properties listed in Table 3-IV. The two sets of elastic values listed for both CVI 1216 and 1219 are due to the changes made in the samples after the first series of tests, where neither sample failed in tension. For both samples, the SiC layer on the inner surface of the tube was relatively thick (on the order of 4-mm thick for CVI 1219), increasing the cross-sectional area, thereby decreasing the stress in the material during testing. Due to this, the strength of the sample exceeded the load capacity of the MTS system (246 kN). The steel inserts were removed by burning off the epoxy layer by placing the sample in a furnace at 400ºC for 1 hour. The power was turned off, and the system was allowed to cool over night before the sample was removed. The epoxy burned off leaving the samples coated in charred epoxy resin and soot, but free of the inserts. The SiC layer on CVI 1216 was too thin for effective machining without damaging the sample, so it was repotted and returned to testing.

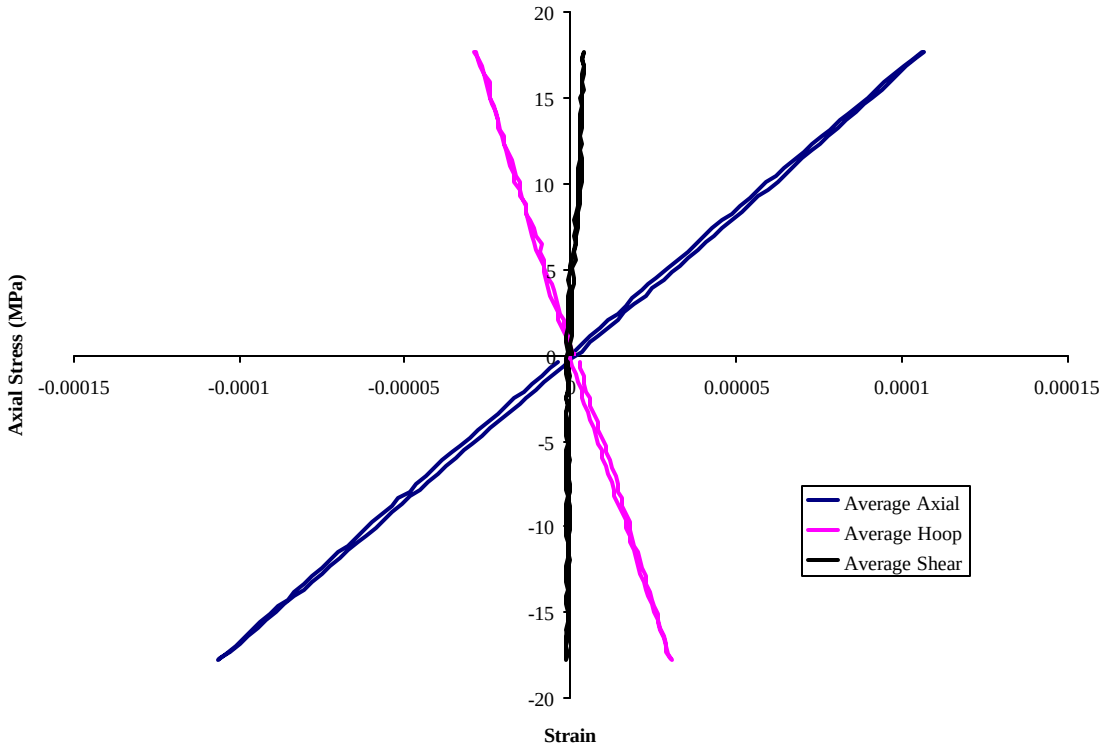


Figure 3-13: Typical tensile test for sample CVI 1219

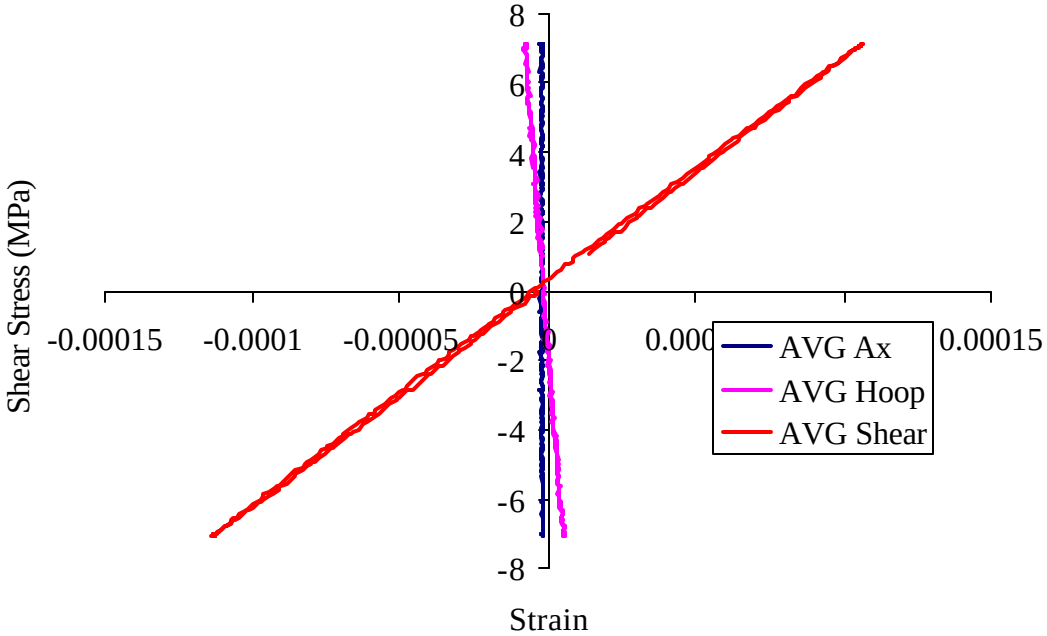


Figure 3-14. Typical torsion test for sample CVI 1219

CVI 1219 was returned to ORNL, and the SiC layer was machined off the inner surface. For Sample CVI 1216 – AR is for As-Received and AHT is for After Heat Treatment. For Sample CVI 1219 – AR is for As-Received and AM for After-Milling.

Table 3-IV: Mechanical Properties of the Nextel/SiC composite tubes

Tube ID	Axial Stiffness (GPa)	Torsional Stiffness (GPa)	Poisson Ratio	Tensile Strength (MPa)
1173-1	172.5±0.9	66.1±2.1	0.25	31.1
1173-2	165.2±0.6	65.7±0.1	0.24	68.1
1216 AR	168.1±1.1	62.8±1.0	0.20	>171.5
1216 AHT	167.5±4.6	80.17±1.2	0.27	>197.4 Axial + 43.5 Shear
1219 AR	142.1±0.9	65.1±0.5	0.31	>137
1219 AM	148.8±1.3	67.3±0.5	0.52	200.2

The axial stiffness for CVI 1216 was not affected by the thermal exposure to remove the epoxy, while the torsional stiffness increased by 28%. The torsional tests have been repeated, with little variation in the observed modulus. Investigation into what structural changes may have occurred has not been conducted at this point. The properties for CVI 1219 increased slightly with the removal of the SiC inner coating.

The stress/strain curves for the tensile strength tests of the CVI 1173 samples are in Figure 3-15. The tensile strength values for the two 1173 samples are not indicative of the material since both exhibited grip induced failures. Both 1173 samples exhibited a small amount of nonlinear behavior, as can be seen in the departure from the CVI 1216 line. The fracture surface exhibited some fiber pullout, as can be seen in Figure 3-21. CVI 1219 failed in the gage section, and the tensile strength plot is in Figure 3-16. An extension of the linear behavior of the material is included to illustrate the nonlinear behavior. The value is believed to be lower than what would be expected for the material, since the sample was changed by the machining process. The inner and outer surfaces were not concentric, creating wall thickness variations around the sample.

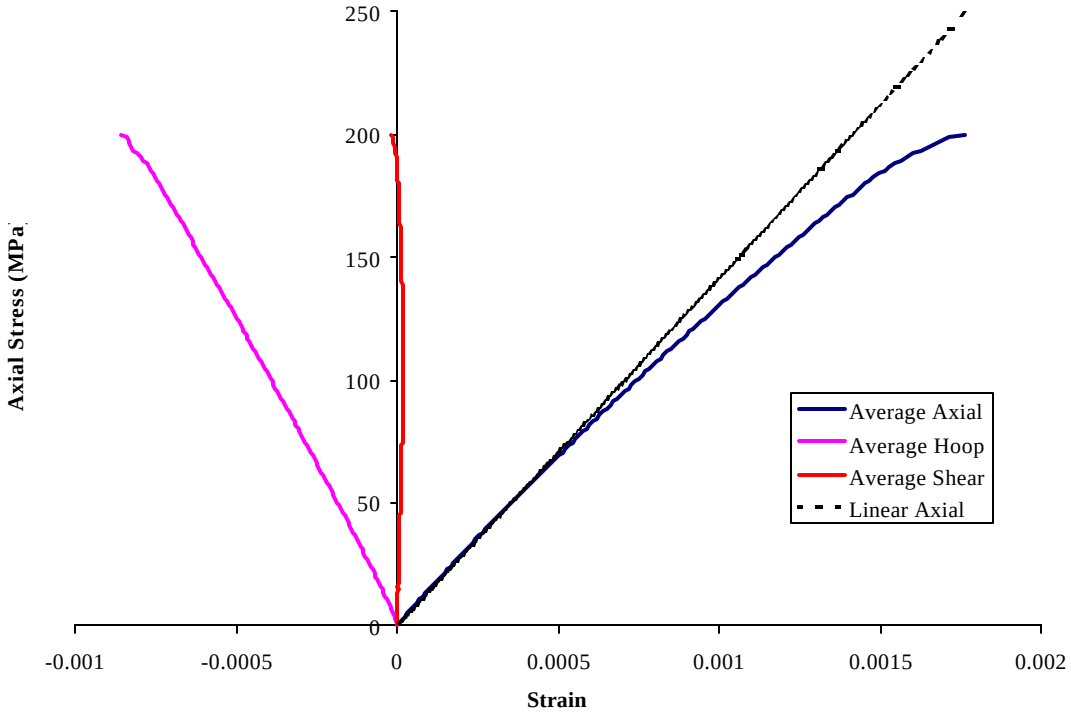


Figure 3-15: Stress/ Axial Strain curves for tensile strength tests

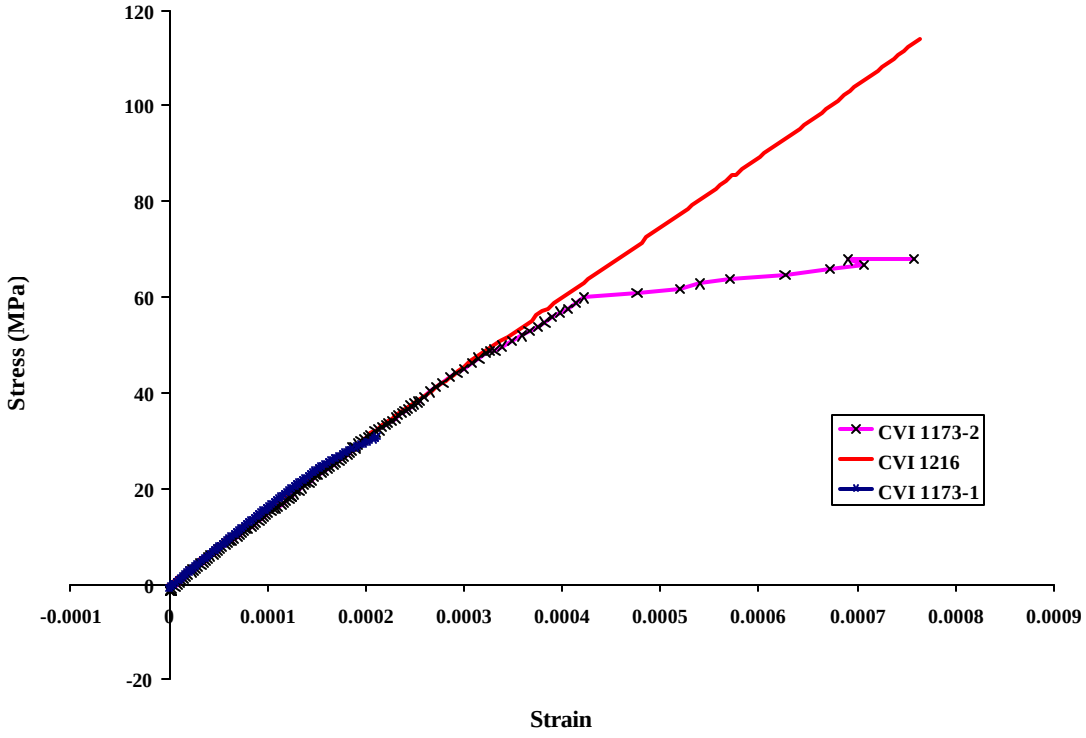


Figure 3-16. Tensile Strength Plot for CVI 1219.

During the testing of samples 1216 and 1219 AR, the load profile reached the system limit before failure. The values listed are the largest nominal stress values applied before the 246kN (55 kip) limit or prior to slippage of the sample. In order to test CVI 1216 and 1219, the grip pressure was boosted to 38 MPa, from the normal setting of 20 MPa, to prevent slipping at the higher loads. For CVI 1216 AHT, the load profile was altered so that torque would be applied when the axial load limit was reached. The load profile for CVI 1216 is in Figure 3-17. The axial force reaches 246 kN and an applied torque of 1422 N-m before the sample slipped out of the grips of the MTS frame. Later attempts to repeat the test to a higher level failed due to wear on the epoxy grip surface.

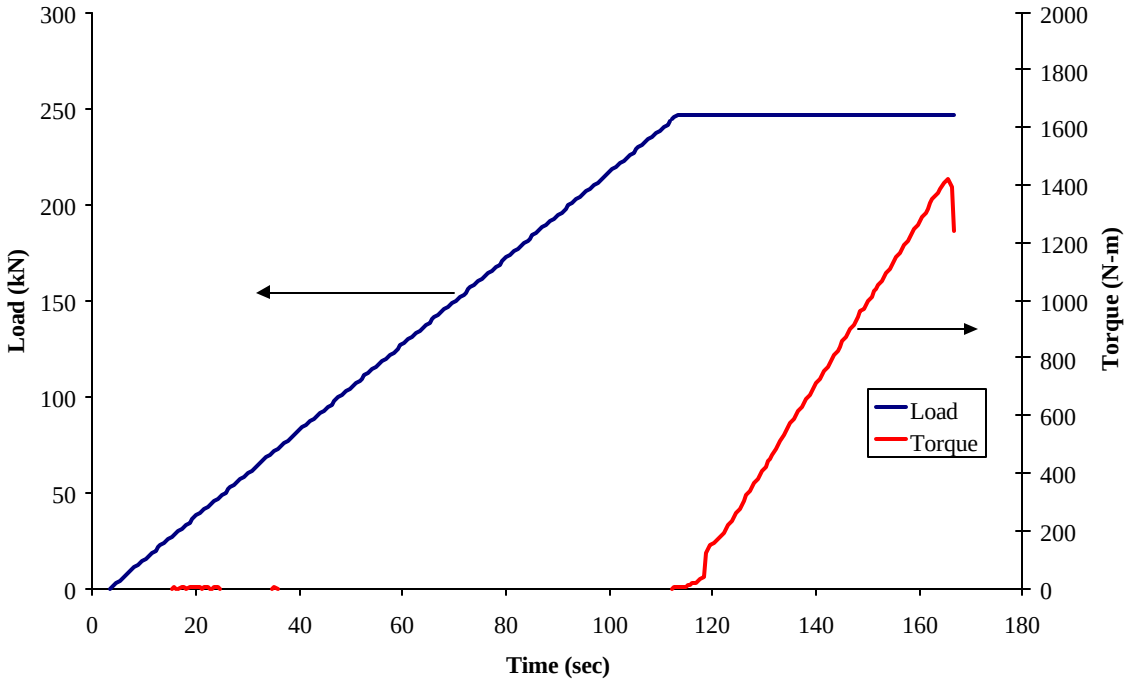


Figure 3-17. Load Profile for CVI 1216

Figure 3-18 and Figure 3-19 are of the strain response to the different sections of the loading profile. Figure 3-18 has the pure tension results, while 2-8 contains the result for the torque ramp with a constant 246 kN axial load.

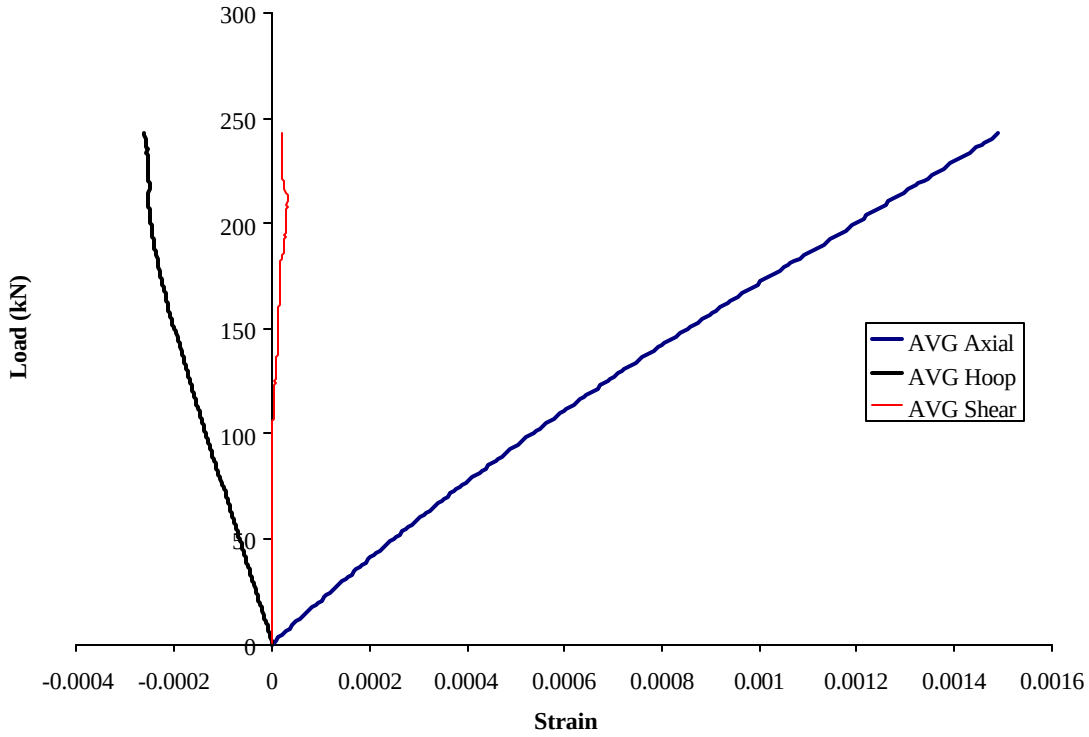


Figure 3-18. Strain response for the tension only portion of Figure 2-6 – sample did not fail

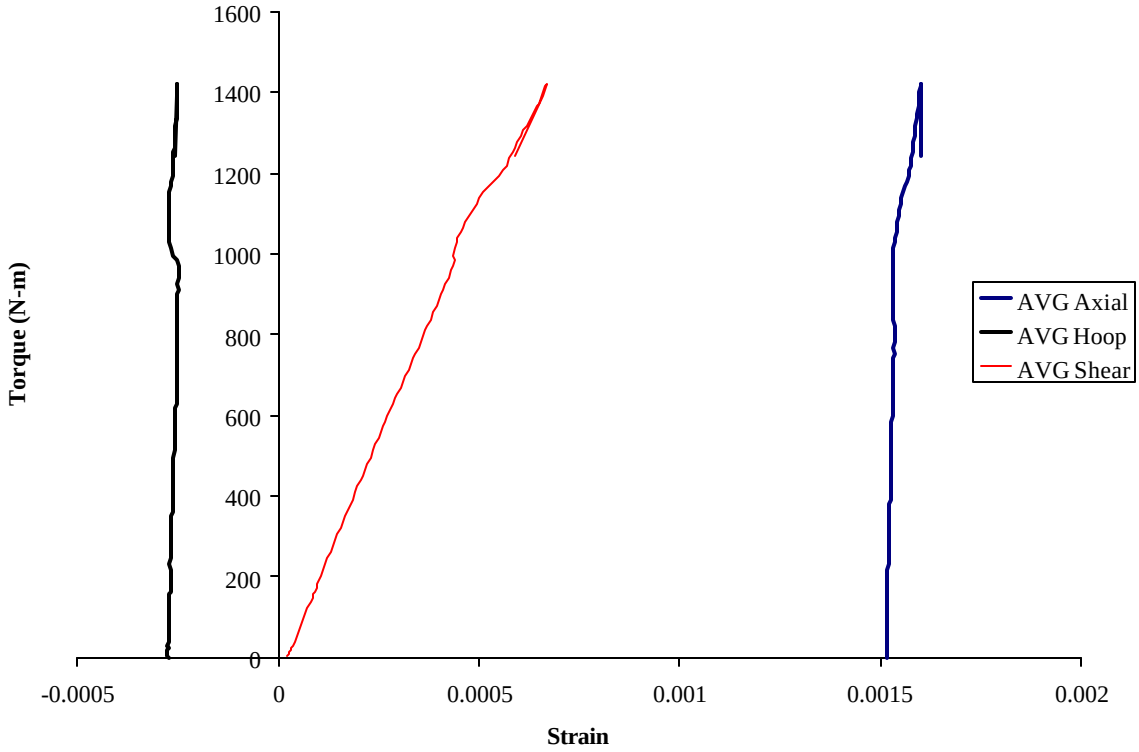


Figure 3-19. Strain response for the torque ramp with 246kN load – sample did not fail

Photographs of the failed of samples CVI 1173-1 and CVI 1219 are in Figure 3-20 through Figure 3-23. The first three images show the grip-induced failure of the CVI 1173-1. The failure originated in the region above the grip area, as can be seen in Figure 3-20 and Figure 3-21. Figure 3-22 and Figure 3-23 are of CVI 1219. All the samples exhibited fiber pullout, while CVI 1219 had delamination and pullout (highlighted in Figure 3-22).

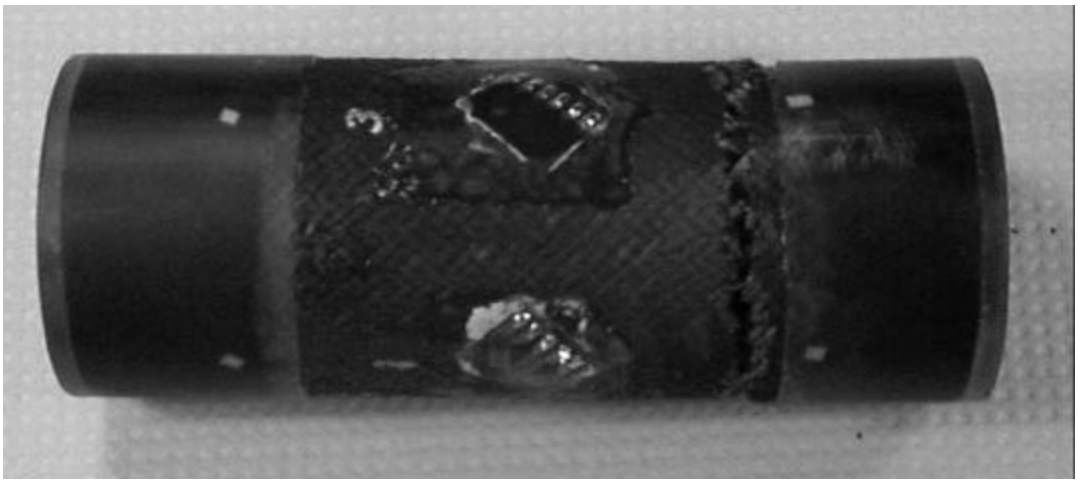
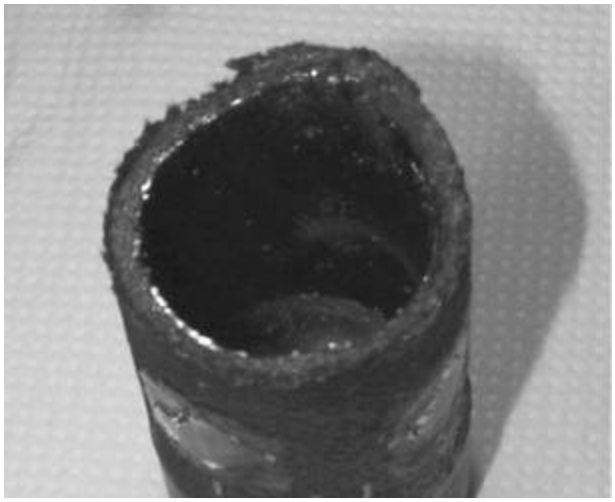


Figure 3-20. Grip induced failure of CVI 1173-1



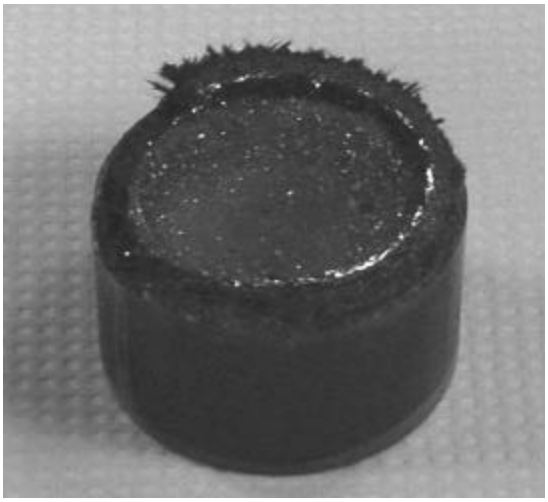


Figure 3-21: Fractured tensile strength specimen CVI 1173-1

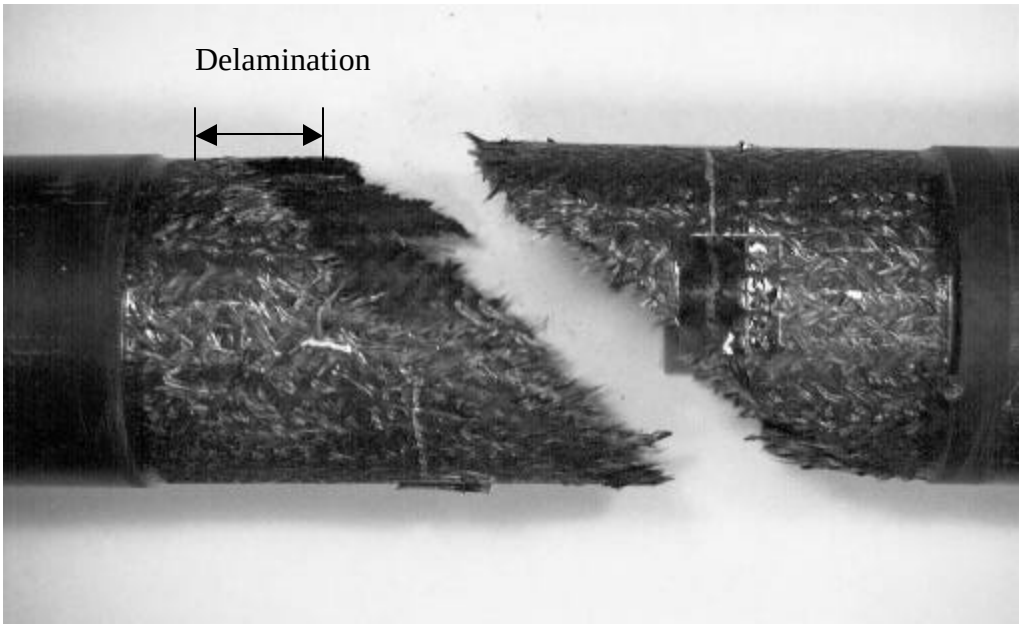


Figure 3-22. Sample CVI 1219 with large amounts of fiber pullout and delamination

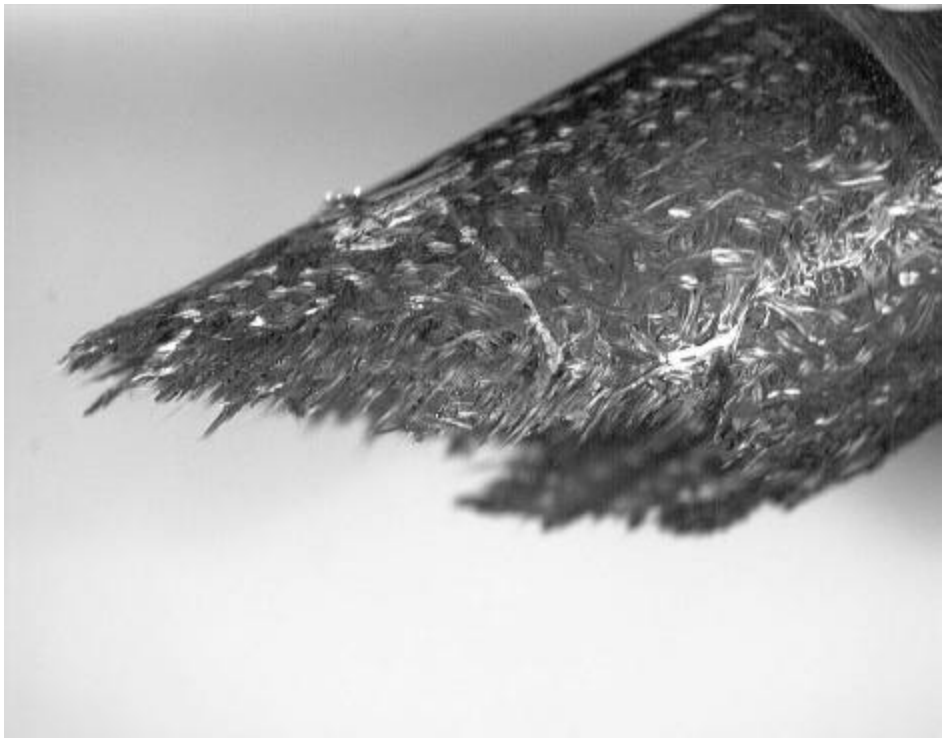


Figure 3-23. Sample CVI 1219 failure surface with pullout

3.2.1 Internal Pressure Tests Results

The results of the internal pressure tests are in Table 3-V. The values are the slopes found by linear regression for the data, as shown in Figure 3-24 (Pressure/strain). The strains become nonlinear between 10 and 12 MPa due to frictional loading and shear failure of the plug. The slope values were found by fitting the data below 8 MPa, to avoid the nonlinear affects.

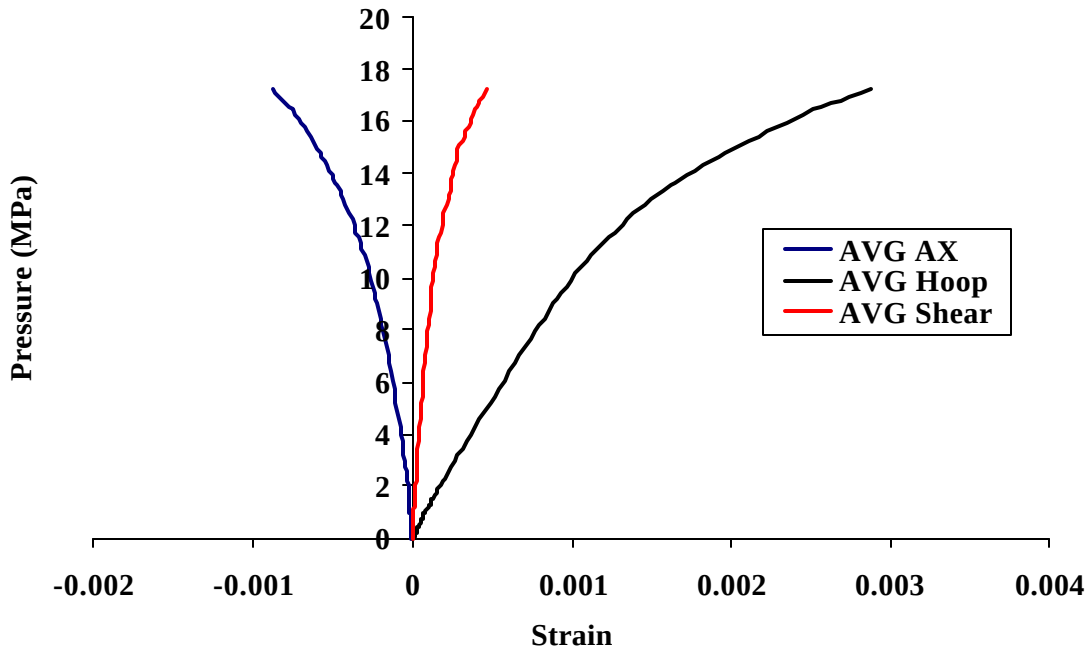


Figure 3-24. Measured strain vs. internal pressure for an internal pressure test – the nonlinear behavior above 10 MPa is due to plug failure.

Table 3-V. Slopes of the best-fit data for CVI 1219

AVG Values	Axial (GPa)	Hoop (GPa)	Shear (GPa)	Poisson's ratio
Pressure/Strain	-65.50±1.21	11.75±0.72	94.51±7.20	0.18
Hoop Stress/Strain	-343.89	61.71	496.17	0.17

3.3 Honeywell Sample

Typical strain response plots for the axial tension and torsion tests for the Honeywell materials are in Figure 3-25 and Figure 3-26. A plot of the strain response of the internal pressure test is found in Figure 3-27. The axial and shear strain components exhibit nonlinear behavior and become more negative throughout the test. The nonlinear response is believed to be due to frictional loading by the plug at the higher pressures. By taking the slope of the data at low loads, the affects of the loading are avoided, and the performance of the material due to pure internal pressure can be recorded. The elastic properties for the Honeywell materials are in Table 3-VI.

Table 3-VI. Elastic properties for the Honeywell materials

Value	Axial Strain (GPa)	Hoop Strain (GPa)	Shear Strain (GPa)	Poisson's Ratio
Axial Stress (Inner)	135.13±1.97	$-1.1*10^3$	$-1.5*10^3$	0.12
Axial Stress (Outer)	133.04±0.77	-888	-412	0.15
Shear Stress (Inner)	$16.2*10^6$	$-23.9*10^3$	48.40±0.32	--
Shear Stress (Outer)	$-55.2*10^3$	$-2.2*10^3$	48.40±0.25	--
Pressure	712	20.26	-504	-0.03
Hoop Stress	$4.5*10^3$	128.87	$-3.2*10^3$	-0.03

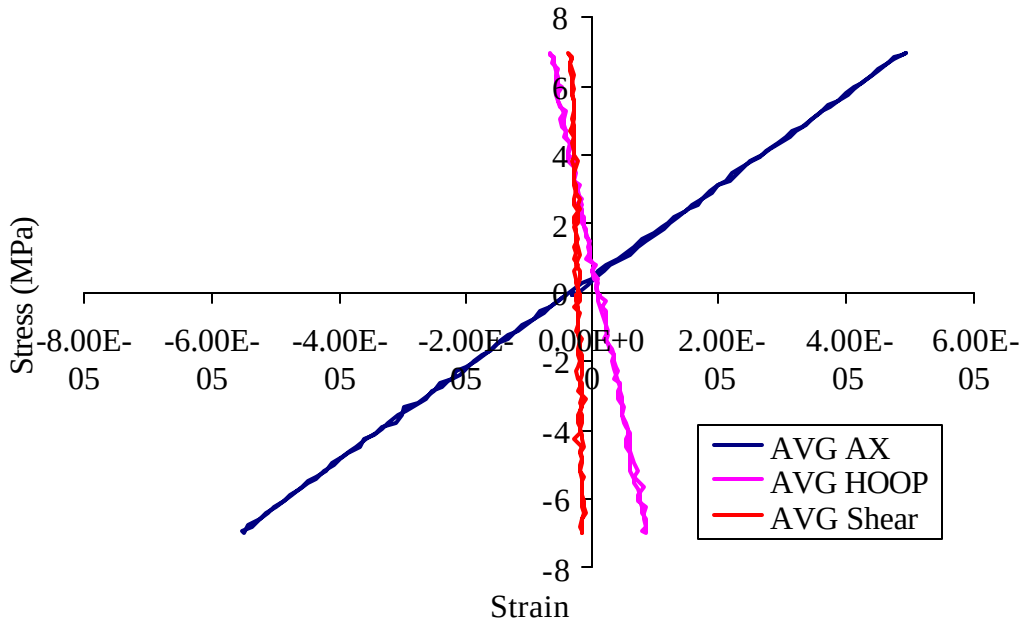


Figure 3-25. Typical Axial tension test for the SiC/SiC tubes from Honeywell

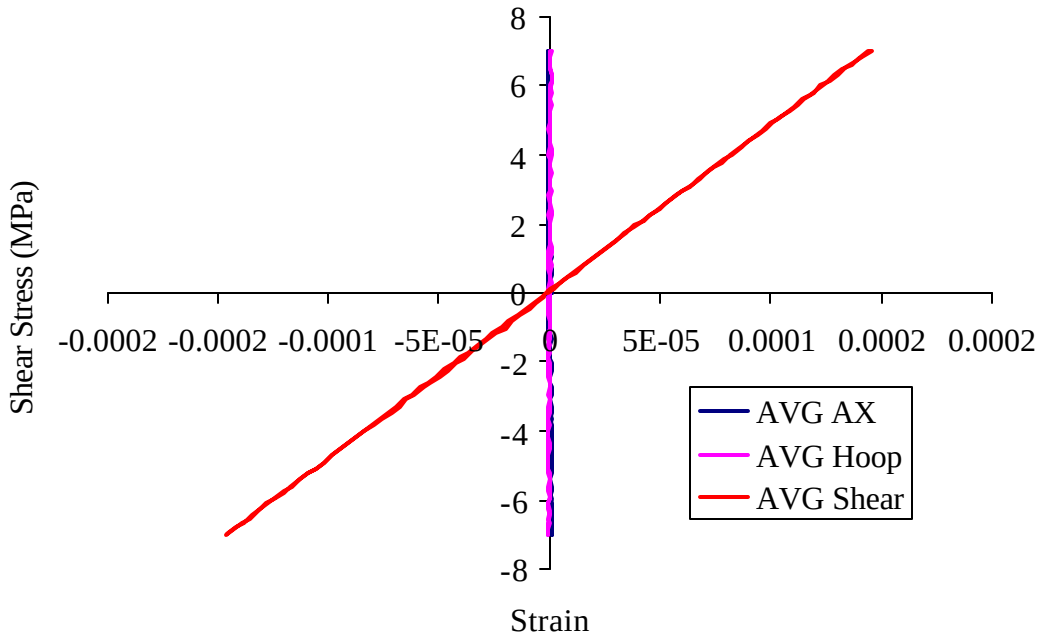


Figure 3-26. Typical torsion response for the SiC/SiC tube from Honeywell

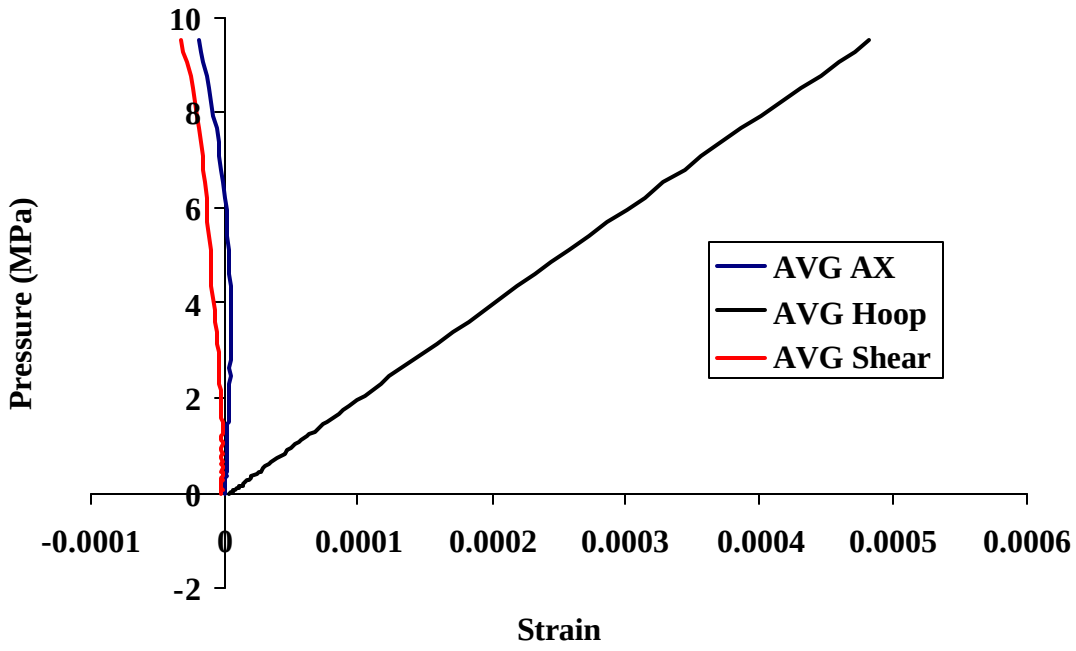


Figure 3-27. Internal pressure test for Honeywell material

The plot of the tensile strength test is in Figure 3-28. The material exhibits a near perfect bilinear behavior due to the 90-degree plies failing in transverse tension around 80 MPa. The test ran to a maximum of 144 MPa, when the epoxy bond in the loading fixture (see Figure 2-5) failed and the sample pulled out (not allowing for measurement of the ultimate tensile strength).

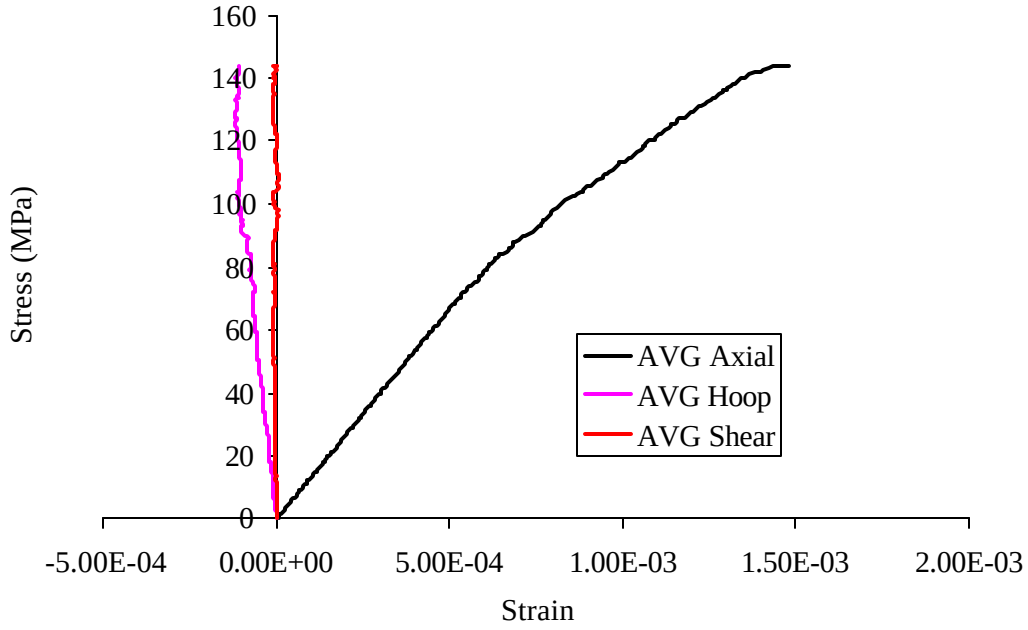


Figure 3-28. Tensile strength test for the Honeywell sample. The sample did not fail – at the maximum load the sample pulled out to the loading fixture

3.4 Control Sample

The elastic response of the steel tube to the different test conditions is listed in Table 3-VII, with the typical stress/strain plots in Figure 3-29 through Figure 3-31. The axial and hoop moduli are within the range listed for this material (193-200 GPa). The shear modulus is below the published value of 86 GPa, but the measured value follows the predicted value for an isotropic material, giving some doubt about the published value.

$$G = \frac{E}{2(1+n)} = 75.4\text{GPa} \qquad (3.1)$$

Table 3-VII. Slopes of the strain responses for the control material

Load Condition	Axial Strain (GPa)	Hoop Strain (GPa)	Shear Strain (GPa)	Poisson's Ratio
Axial Stress	**192.1±2.2**	-700	$-25.5*10^3$	0.274±0.005
Shear Stress	$21.2*10^3$	$-9.73*10^5$	**73.5±0.6**	--
Pressure	-104.32	21.01	-147.54	0.20
Hoop Stress	-968	**194.95**	$-1.37*10^3$	0.20

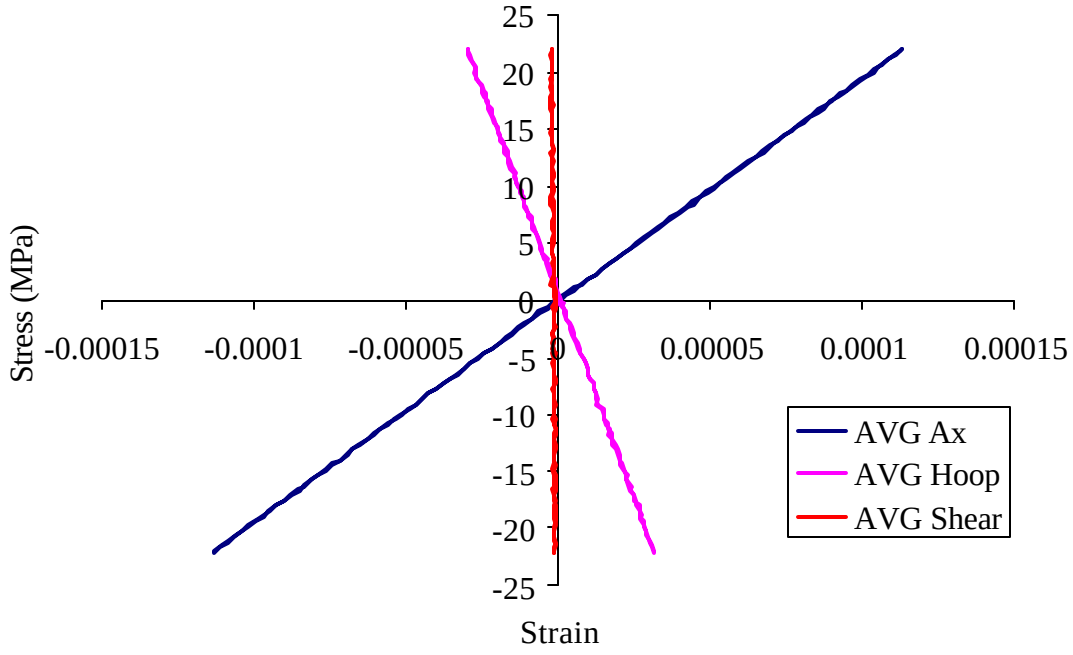


Figure 3-29. Axial tension test for AISI Type 304 steel

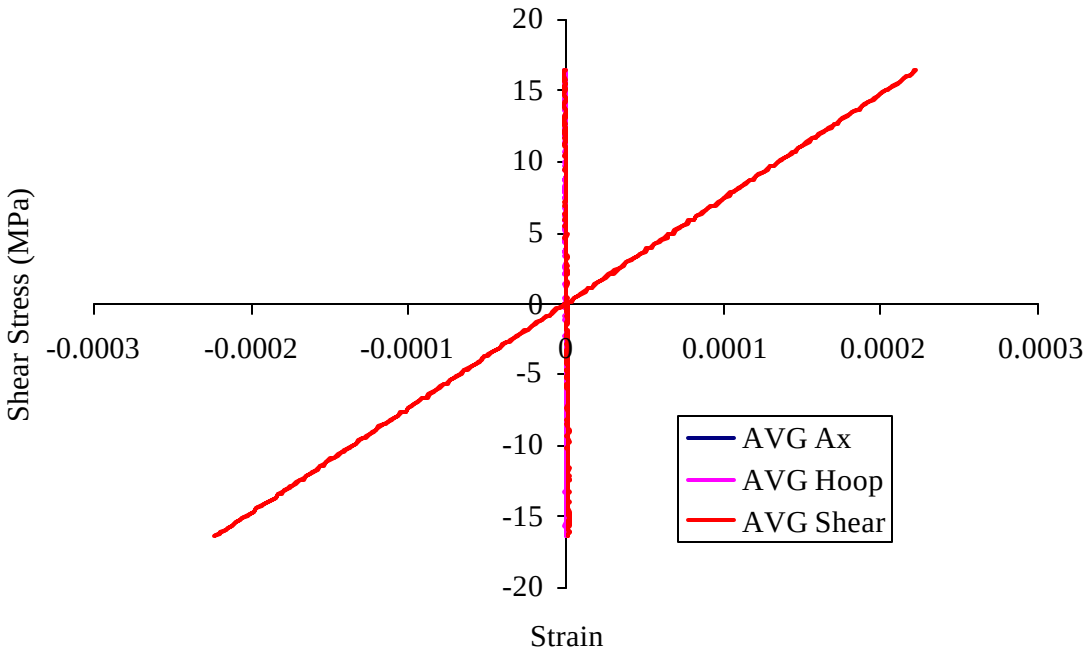


Figure 3-30. Axial torsion test results for AISI Type 304 steel

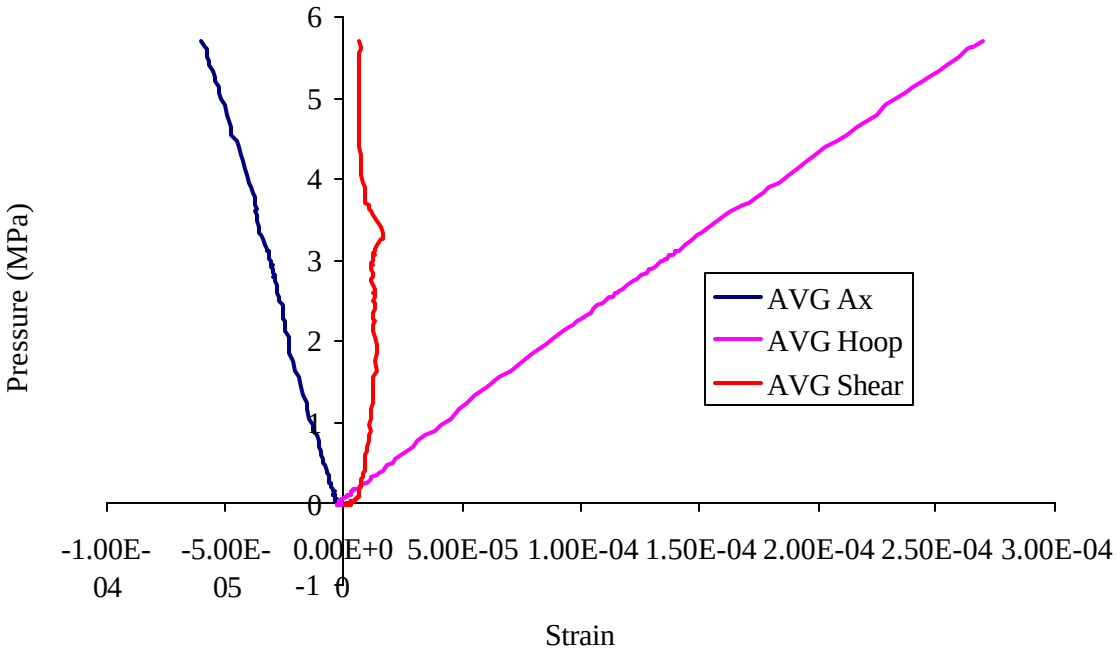


Figure 3-31. Internal pressure results for AISI Type 304 steel

3.5 Discussion of Experimental Results

The first objective for this work was to develop experimental procedures to measure the mechanical properties of the composite tubes from several different sources. The procedures chosen were selected since they would subject the sample to a near-constant stress state over the entire gage section. This provides better insight into the properties than several of the other tests proposed in the literature, such as the O-ring or C-ring tests. The strains are measurable and (ideally) uniform throughout the gage section, allowing for more accurate determination of the material properties.

The axial tension/compression and torsion tests were straightforward in their experimental procedures. Complications arose due to sample misalignment, but could be compensated for with the use of 4 gages around the sample. By averaging the values, the bending contributions were removed and accurate measurement of the tube stiffness could be obtained. The different methods for loading the samples were successful in transferring load from the MTS frame to the samples. The pin loading approach for the filter materials is not ideal, since there were several failures due to the pinholes in the samples. The low strength of the material prevented the use of the other loading approaches, but once the end was potted in epoxy, it experienced gage section failures. The two other approaches, directly gripping the sample and the end caps, were more desirable. Sample misalignment was reduced for both methods to that measured in the filter materials. The end caps used for the Honeywell material worked very well, in that the misalignment was reduced, if not completely removed. The one drawback was the bond failure that prevented the measurement of the ultimate tensile strength. This was due to the fact that the depth of the groove was not machined to the specifications in the diagram. The groove could only be milled to between 7 and 10 mm in depth due to the small width. With a deeper channel, it is likely that the fixture would survive past the tensile strength of the sample.

Values for the axial strength were found for the McDermott filter materials, but not for the Honeywell or ORNL materials. Sample CVI 1219 failed in axial tension after the silicon carbide coating on the inner surface was removed. The strength of the material is expected to be higher than the measured value due to the presence of thickness variations. The milling process removed some of the composite material from the inner surface, cutting into the woven layers, and possibly inducing damage within the material, which would be expected to reduce strength.

Even though the ultimate strength could not be measured for the Honeywell material, the strain response reveals the transverse strength of the composite to be around 80 MPa.

The procedures for the internal pressure tests were developed during this research. This work was done independently of the research at ORNL that developed similar methods. The internal pressure results are in agreement with the values calculated from the hoop response of the control sample, and supported by the results from the axial tests for the McDermott and Honeywell samples. The hoop stiffness values are similar to thee axial stiffness values, as would be expected from the symmetry of the material (or isotropy for the control). The oxide/SiC composites from ORNL did not exhibit this behavior. Several factors may contribute to the deviation, but damage incurred during other tests and due to the removal of the SiC layer are more likely. The samples had been used for testing with the urethane plugs. Pressures estimated to be in excess of 10 MPa were applied to these materials. It is uncertain if any matrix cracking was present due to the previous loading or machining process to remove the SiC layer (which was removed from the internal pressure test sample as well). Both of these factors would decrease the stiffness, due to the increased compliance of the cracked material.

The responses of the materials at low pressures were in agreement with what is to be expected. This is not necessarily the case at higher pressures. The strain responses for many of the samples exhibited deviations from linearity, as can be seen in the axial and shear strain responses in Figure 3-27. These are not believed to be due to damage in the material, but more from frictional loading of the sample. Selecting a better high-pressure lubricant can reduce this problem. Also, the tests were not conducted to pressures sufficient to fail the materials. The Silastic plug would shear out around the compression platens prior to failure of the material. A compression platen that more closely matches the inner diameter of the test sample should alleviate this problem, since it would generate a pure, or at least, nearly pure hydrostatic load condition. The large shear stresses would not be present without the spacing between the platen and the sample. A better fitting platen was not developed in this work since the surface roughness of the samples required some space to prevent the platen from directly loading the sample in compression.

4 Forward and Inverse Solutions

As was described in the Introduction, a nonlinear regression technique has been utilized to calculate estimates of the elastic constants for an orthotropic composite tube. The work by George was successful in minimizing the error in data sets calculated by the Forward solution, but was not able to find good estimates for the elastic properties for experimental data sets. Since that work, several alterations have been made to the original idea, adding functionality to the analysis, in the hopes of improving its usefulness. Also, a separate optimization analysis was developed that works the same problem with a different approach. Instead of using a Newtonian or gradient-based regression technique, this new method utilizes the Nelder-Mead simplex method.

The changes to the analysis from the work of George and the new methods utilized will be described in this section. They will be validated by simulated test data found using the Forward solution. Each of the methods developed for this work will be given a data set of accurate strain values, and allowed to search for a solution given the same set of start values, which are significantly different from the solution set. The analysis will be applied to the experimental data and the results will be given. To understand the influence of error on the final solutions, the more efficient methods will be given strain data sets with random error introduced. This will allow for the effects of different forms of error on the experimental data to be illustrated, and give some information of the level of accuracy needed in experimental result for these methods to find an accurate inverse solution. Also, the variations of the strain response due to changes in the individual elastic constants are investigated to determine which tests are the most effective in generating the most useful data sets.

Extensions of the research methods proposed in this body of work are the development and application of suitable multi-axial failure theories and increasing the scope of the analysis to find thermal and hygroscopic expansion constants. The experimental results provide information on the elastic properties and strength under several different loading conditions. The Forward solution allows for the calculation of the stress state through the thickness for the failure loads. With enough variation in the failure conditions, sufficient data would be available to calculate failure envelopes for these materials. A brief framework and example of applying a multiaxial

failure criterion will be given using the results from the McDermott filter data, since it was the only material with failures under different loading conditions (Pi, Fx, Fx+Tx).

4.1 Inverse Solution

The first alteration made to the analysis was the use of the Levenberg-Marquardt compromise to stabilize the performance of the nonlinear regression analysis. This is a small change to the operation of the Newtonian method, which decreases sensitivity to poorly formed Jacobian matrices. The next change was to the value being optimized. The original program optimized the elastic properties, where as a new approach alters the values of the stiffness matrix (C_{ij}) directly, reducing the nonlinearity of the system of equations. Another added feature allows for the user to reduce the number of active parameters (E_i values) from the seven to any value desired. This is significant since it allows for any number of variables to be set constant while the remaining values are optimized. The final change was to increase the amount of data available to the analysis by allowing data from the inner surface to be included.

The method used to change the E_i or C_{ij} values is a Newtonian or gradient method. For the Newtonian methods, the changes to the elastic constants are calculated using the error vector, Ö, and the Jacobian matrix, J.

$$X^{i+1} = X^{i} + \left(J^{T} J\right)^{-1} J^{T} \Phi \tag{4.1}$$

where X is the variable (E_i or C_{ij}), X^{i+1} is the new, adjusted value, X^i is the previous value, and ϕ is the difference between the measured and predicted strain value. This method requires that the gradient is defined and continuous at all points, and that the Jacobian matrix is well formed (non-singular and not ill conditioned or collinear).

A problem with this method arises with a near singular Jacobian matrix due to collinearity in the data. The step value can become extremely large, causing the parameters to go into an undesirable region of parameter space. The Levenberg-Marquardt compromise was developed to reduce the sensitivity of the analysis to this condition [48, pages 80-81, and 49, pages 624-627]. Diagonalizing the inverted section of (4.1) by adding a small value stabilizes the matrix and reduces sensitivity to abrupt changes in the gradients. There are several methods for changing the matrix, but the method employed in this research changes (4.1) to:

$$X^{i+1} = X^i + \left(J^T J + kD \right)^{-1} J^T \Phi \tag{4.2}$$

Where D is the diagonal values of the $J^T J$ matrix and k is a scaling factor (chosen so k<<1). This is the Marquardt method, where the diagonal terms are multiplied by a factor of (1+k). The increment values range from the original Newtonian step value (k=0), to what would be the step value for the method of steepest descent (k). The steepest descent uses infinitely small steps calculated from each gradient, allowing for the most direct path to a minimum, but is not feasible due to the increased number of computations for the small steps. The flow chart for the program using the Levenberg-Marquardt compromise is found in Figure 4-1.

With the addition of the scaling term k, the problem has become more complex since now this term must be maintained. The size of the k value must be changed to maintain the best optimization step size. As k increases, the step size decreases, and conversely, as k decreases, the step size increases. The method employed in this analysis is not computationally efficient, but it is efficient in choosing the best step value. It evaluates the error function for three different k values (step sizes) and chooses the step that results in the largest decrease. This roughly triples the number of evaluations of the error function, increasing the computation time and decreasing efficiency. The initial k value is chosen to be small (k=0.0001), and the first step is calculated. To take the next step, three values are chosen for k; k/10, k, and 10k, and the error functions are evaluated. The step that produces the smallest error is chosen for the step. If the smallest error of the three steps is not smaller than the previous error value, then the program goes into an interval-reducing scheme (seen in the lower right portion of Figure 4-1). The value of k is multiplied by ten (k, 10k, 100k), reducing the step size, and is repeated (10k, 100k, 1000k, etc…) until the step reduces the error function. If the value of k exceeds a set large value, the program stops, and the data inspected to see if it is a local minimum or a bad set of values.

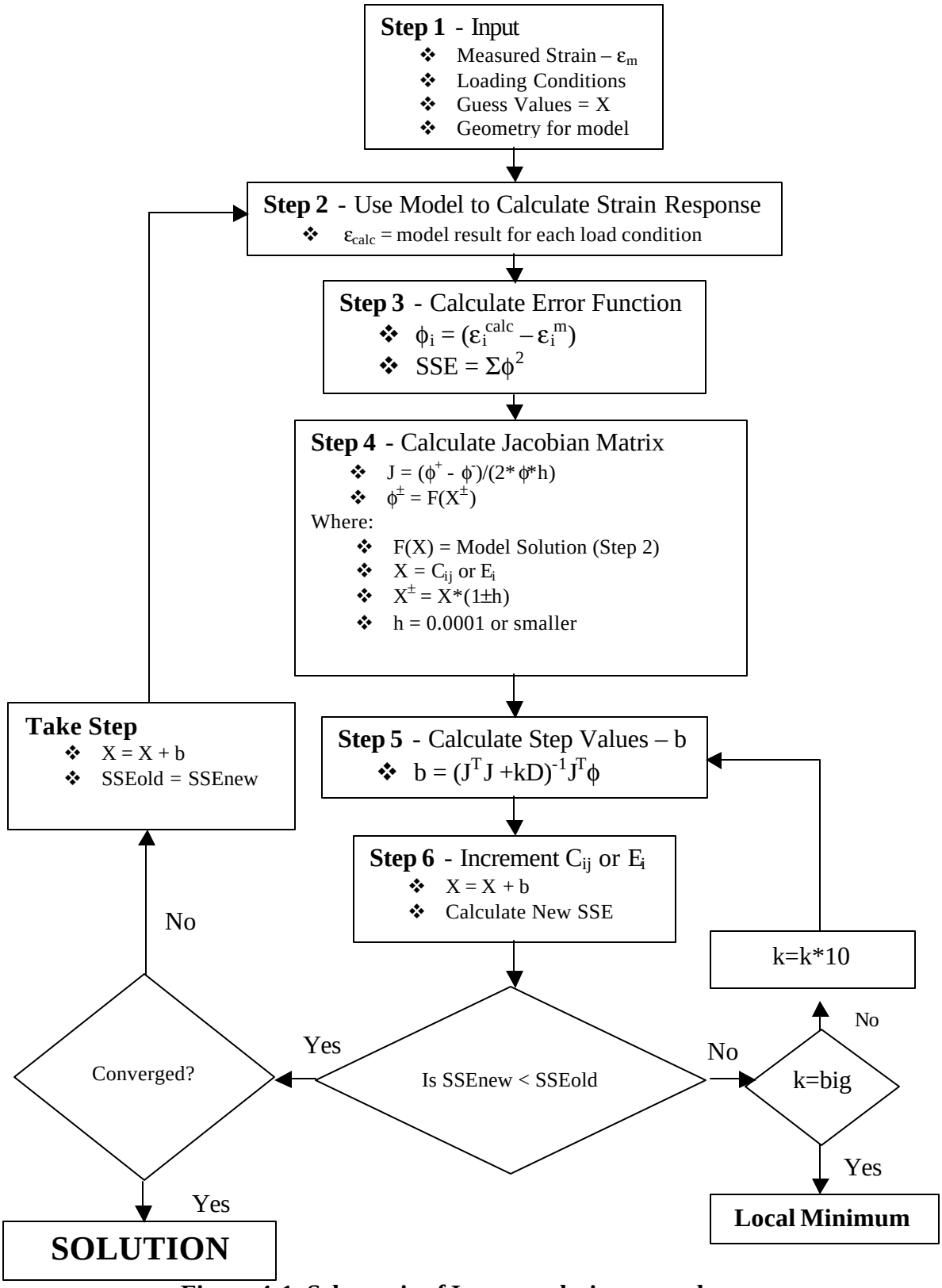


Figure 4-1. Schematic of Inverse solution procedure

Another addition to the program is the ability to optimize on the elastic constants (E_1, E_2, E_3…) or on the stiffness matrix values (C_{ij}). This is significant in that it reduces the nonlinearity of the equations by removing the nonlinear conversion from E_i to C_{ij}, as can be seen in (4.3).

$$
\begin{aligned}
C_{11} &= \frac{1-\nu_{23}\nu_{32}}{E_2 E_3 \Delta} & C_{23} &= \frac{\nu_{32}+\nu_{12}\nu_{31}}{E_1 E_3 \Delta} \\
C_{12} &= \frac{\nu_{12}+\nu_{32}\nu_{13}}{E_1 E_3 \Delta} & C_{33} &= \frac{1-\nu_{12}\nu_{21}}{E_1 E_2 \Delta} \\
C_{13} &= \frac{\nu_{13}+\nu_{12}\nu_{23}}{E_1 E_2 \Delta} & C_{44} &= G_{23} \\
C_{22} &= \frac{1-\nu_{13}\nu_{31}}{E_1 E_3 \Delta} & C_{55} &= G_{13} \\
 & & C_{66} &= G_{12}
\end{aligned}
\qquad (4.3)
$$

$$
\Delta = \frac{1-\nu_{12}\nu_{21}-\nu_{23}\nu_{32}-\nu_{31}\nu_{13}-2\nu_{21}\nu_{32}\nu_{13}}{E_1 E_2 E_3}
$$

The more terms that can be removed from the nonlinear aspects of the problem, the better the linear step value will improve the estimates. In the first several steps of the Forward solution (Step 2 in Figure 4-1), the elastic constants are used to calculate the stiffness values (C_{ij}). Since there is only one set of stiffness values for the material, they can be used for the optimization values. This is not possible for the transformed stiffness matrices since they are dependant upon the geometry of the each layer and not just the material constants (for an N ply composite – 1 set of C_{ij} values but up to N sets of $\bar{C}_{ij}$ values). The disadvantage of optimizing the stiffness matrix is that it precludes another process where the number of constants being optimized is reduced from the seven original values to a set input by the user ranging from 1 to 7 active parameters.

A concern with this analysis is the small amount of data available to optimize seven elastic parameters. Early results with this process would find some reproducible solutions, but the out-of-plane values would return far outside acceptable ranges. These variations are due to the experimental error in the limited data sets. There are two possible ways to handle the limited data sets; one is to reduce the number of variables, and the other is to increase the amount of data. By decreasing the number of active parameters, or fixing certain elastic constants to set values and optimizing on the remaining values, the user can control variables that tend to unrealistic values, and find values for the others. An example of this would be to set the out-of-

plane values to be constant and optimize on the remaining in-plane values. To reduce the active parameters, the input data includes the number of active parameters and information on the values. With this information the Jacobian matrix is reduced in size, including the terms applying to the active values. The program operates in the same fashion, but only increments the variables defined as active, leaving the other terms constant and equal to the start value. The routines for making these changes can be found in the Jacobian routines found in the source codes in the Appendices.

The second method is to increase the amount of data available to the analysis. One means of this is to input values for the strain response from the inner surface of the sample. This increases the total amount of data by doubling the measurements for each test. It will reduce the number of possible solutions, since the values need to predict the strain response on the two surfaces. Also, it should give more information on the out-of-plane properties from the internal pressure tests, since the largest out-of-plane stress state is at the inner surface. The limitation of actually measuring the strain of the inner surface accurately with pressure applied is beyond this work. The elastomer compression method does not allow for these measurements.

The selection of convergence criteria is important to the success of the analysis. A balance must be made between declaring convergence and terminating the program, and allowing the system to find a true minimum. If the criteria are chosen too loosely, the program declares a minimum before a true minimum is found. Conversely, we wish to stop the analysis from wasting computation time making insignificant changes to the estimates or continuing when the estimates are well outside acceptable values. Several convergence criteria are employed to determine convergence or to prevent the system from running for excessive periods of time. From the flow chart in Figure 4-1, there are three locations for terminating the procedure; if k becomes too large, the analysis exceeds a fixed number of iterations, or if the solution meets the convergence criteria. The user of the program sets the maximum number of iterations, so that program terminates if no solution has been found in a reasonable amount of time. This is not a convergence criterion, since it does not inspect the data itself, but merely prevents the program from running for extended periods of time unchecked. There are two main criteria employed in this program to check for significant change in the estimates of the elastic properties. The first is that the program continues as long as one of active elastic constants changes by more that 0.01% over 5 iterations, and the second is if the scaling factor, k, exceeds a

set large value. Usually, when k becomes large ($k>10^6$), the step sizes become small enough that the program hits the other criterion, in that the changes to the active parameters become very small. If the criterion is not reached, the step sizes are very large, indicating the potential that the solution has entered an undesirable section of parameter space. The program stops and reports which criterion was reached, which allows for the user to inspect the solution, and choose a suitable course of action (i.e. determine if a genuine solution has been found or to change the guess values and repeat the analysis).

4.1.1 Verification of Inversion Methods

To verify if the different methods calculate the best elastic properties, test runs were conducted using each. A fictitious sample was created using the typical values for an S-2 Glass/Epoxy composite (50, page 14). The properties and geometry values are in Table 4-I and Table 4-II. The Forward solution was used to calculate the strains for a basic set of load conditions, listed in Table 4-III. The values of the loads were chosen to be similar to those seen in experiments. The calculated strain response, listed in Table 4-IV, are listed for the different loading conditions and surfaces (I=inner or O=outer).

Table 4-I. Elastic Constants for a S-2/Epoxy composite

Elastic Properties	**GPa (Msi)**
$\mathbf{E_1}$	43.5 (6.31)
$\mathbf{E_2}$	11.5 (1.67)
$\mathbf{E_3}$	11.5 (1.67)
$\mathbf{G_{12}}$	3.45 (0.5)
$\mathbf{n_{12}}$	0.27
$\mathbf{n_{13}}$	0.4
$\mathbf{n_{23}}$	0.4

Table 4-II. Geometric Constants for two fictitious composite tubes

Tube ID	Inner Radius mm (in)	Outer Radius mm (in)	Number of Plies	Orientation	Ply Thickness mm (in)
1	25.4 (1.0)	27.9 (1.1)	20	$[45/-45]_{10}$	0.127 (0.005)
2	25.4 (1.0)	27.9 (1.1)	20	$[0/90]_{10}$	0.127 (0.005)

Table 4-III. Loading Conditions for the tubes

Load #	P_i MPa (ksi)	F_x kN (lbf.)	T_x N-m (in-lbf)
1	10 (1.45)	0	0
2	0	5 (1125)	0
3	0	0	100 (885)
4	0	5 (1125)	100 (885)
5	10 (1.45)	5 (1125)	100 (885)

Table 4-IV. Strain response for the tubes using the loading conditions from Table 4-III

		Tube 1			Tube 2		
Load #	Surface	ε_x	ε_θ	$\gamma_{x\theta}$	ε_x	ε_θ	$\gamma_{x\theta}$
1	O	-5.5713E-03	8.4283E-03	-5.9773E-06	-3.348E-04	3.440E-03	-1.177E-18
2	O	1.0406E-03	-6.4180E-04	2.0183E-06	4.251E-04	-5.393E-05	2.215E-20
3	O	1.4434E-06	-4.2934E-07	7.4009E-04	1.584E-20	-9.157E-20	2.671E-03
4	O	1.0421E-03	-6.4223E-04	7.4211E-04	4.251E-04	-5.393E-05	2.671E-03
5	O	-4.5292E-03	7.7860E-03	7.3613E-04	9.022E-05	3.386E-03	2.671E-03
1	I	-5.5713E-03	9.4626E-03	-5.4339E-06	-3.348E-04	3.977E-03	-1.070E-18
2	I	1.0406E-03	-6.8795E-04	1.8348E-06	4.251E-04	-4.135E-05	2.014E-20
3	I	1.4434E-06	-5.2785E-07	6.7281E-04	1.584E-20	-1.039E-19	2.428E-03
4	I	1.0421E-03	-6.8848E-04	6.7464E-04	4.251E-04	-4.135E-05	2.428E-03
5	I	-4.5292E-03	8.7741E-03	6.6921E-04	9.022E-05	3.935E-03	2.428E-03

The first check of the analyses is to input a perfect data set (one with no error included in the 5 significant digits listed above) and give bad start values. The results for the different optimization methods given the same start values are listed in Table 4-V and Table 4-VI. The different methods are the Levenberg-Marquardt compromise (LM) and the Nelder-Mead simplex method (NM) optimizing the stiffness matrix values (C_{ij}) and the elastic constants, (E_i). Plots of the Sum of Square Errors (SSE) at each iteration are located in Figure 4-2 and Figure 4-3. The values of the SSE are on a logarithmic scale, starting at an initial value between 10^{-2} and 10^{-3}, and plateau at the solution at 10^{-15}.

Table 4-V. Results for Tube 1. LM – Levenberg-Marquardt, NM – Nelder-Mead

Properties	Start	LM – C_{ij}	LM – E_i	NM - C_{ij}	NM - E_i
E_1 (GPa)	6.90	43.61	43.61	46.20	46.75
E_2 (GPa)	3.45	11.40	11.40	8.48	7.93
E_3 (GPa)	3.45	11.52	11.52	8.00	10.89
G_{12} (GPa)	0.69	3.45	3.45	3.45	3.45
ν_{12}	0.3	0.273	0.273	0.357	0.385
ν_{13}	0.3	0.402	0.402	0.443	0.627
ν_{23}	0.3	0.398	0.398	0.257	0.171
Final SSE		5.98E-15	5.98E-15	1.34E-11	2.18E-11
Iterations		14	62		
Computation time		1.04 Minutes	3.1 minutes	8.8 minutes	44 seconds

Table 4-VI. Results for Tube 2, LM – Levenberg-Marquardt, NM – Nelder-Mead

Properties	Start	LM – C_{ij}	LM – E_i	NM - C_{ij}	NM - E_i
E_1 (GPa)	6.90	42.42	43.86	27.92	47.64
E_2 (GPa)	3.45	12.66	11.14	27.24	7.38
E_3 (GPa)	3.45	11.52	11.49	10.83	6.09
G_{12} (GPa)	0.69	3.45	3.45	3.45	3.43
ν_{12}	0.3	0.246	0.279	0.114	0.365
ν_{13}	0.3	0.385	0.402	0.640	-0.002
ν_{23}	0.3	0.421	0.395	0.195	1.054
Final SSE		1.53E-15	1.53E-15	5.37E-12	4.65E-08
Iterations		14	15		
Computation time		60 seconds	1.21 minutes	50 seconds	8.4 minutes

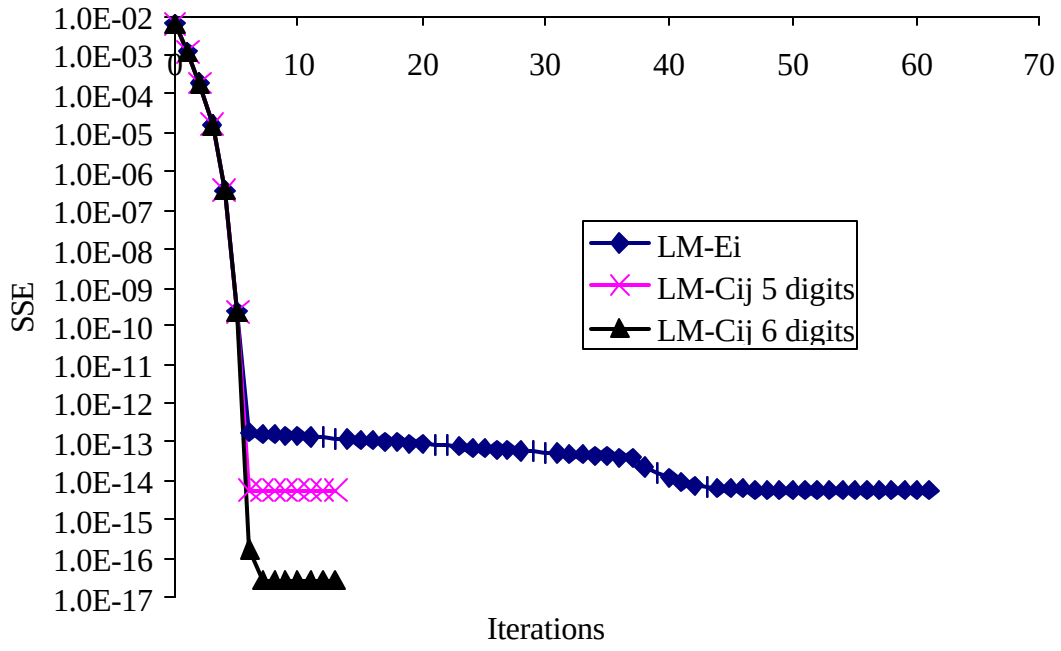


Figure 4-2. Sum of Square Errors for Tube 1 for the two different Newtonian methods.

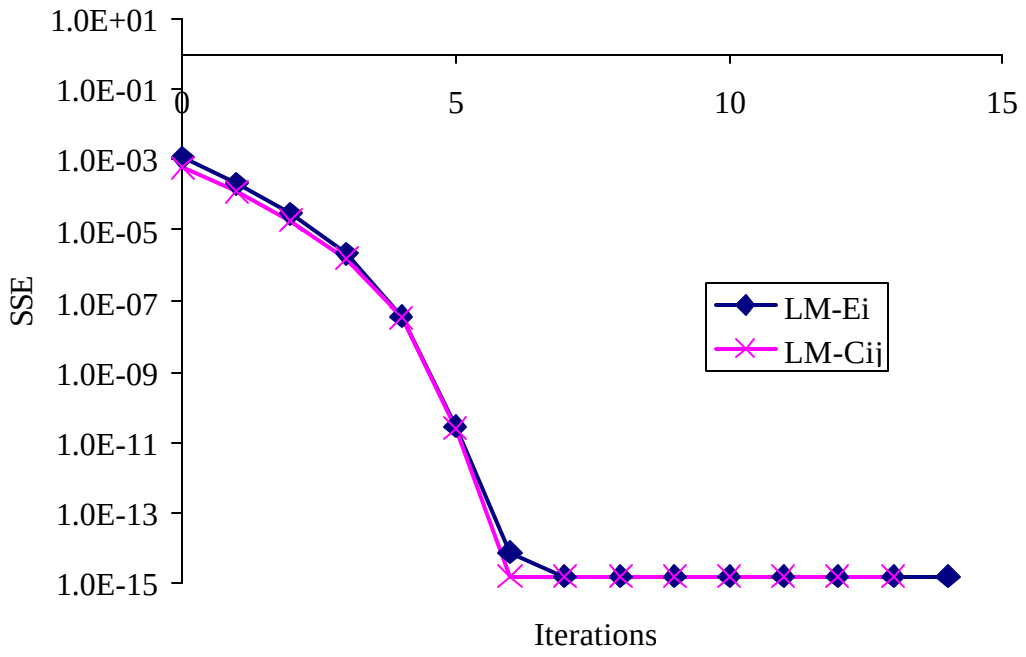


Figure 4-3. Sum of Square Errors for Tube 2 for the two different Newtonian methods.

Upon inspection of the results, the two Newtonian methods are more accurate, finding the estimates of the elastic constants to within a few percent of the true values (the accuracy can be increased by including a larger number of significant digits in the strain data – as can be seen in the two LM-C_{ij} lines in Figure 4-2 – one contains 5 significant digits and found a minimum at 10^{-15} and the other has 6 digits and terminated with a final SSE of 10^{-17}). The Nelder-Mead methods improved the values, but not to the same level as the other two approaches.

An interesting side effect of this analysis was found using both of the optimization methods; a second solution exists for the Tube 2 geometry. For the Levenberg-Marquardt compromise on the elastic constants, a second solution to the problem was found by changing the start values (Table 4-VII). With the [0/90] structure of the tube, a second solution is an average of the E_1 and E_2 values. It has the same residual error, meaning without knowing the correct answer, it would be impossible to determine which solution is the correct with some other information to differentiate them.

Table 4-VII. The two solutions for Tube 2 found using different start values

Properties	Start 1	LM – E_i	Start 2	LM – E_i
E_1 (GPa)	6.90	42.42	6.90	27.92
E_2 (GPa)	3.45	12.66	6.90	27.44
E_3 (GPa)	3.45	11.52	6.90	11.38
G_{12} (GPa)	0.69	3.45	0.69	3.45
ν_{12}	0.3	0.246	0.2	0.112891
ν_{13}	0.3	0.385	0.3	0.171464
ν_{23}	0.3	0.421	0.3	0.674271
Final SSE		1.53E-15		1.53E-15
Iterations		14		15
Computation time		60 seconds		1.2 minutes

4.2 Application to Experimental Results

The data described in Section 2 of this paper has been reduced for input into the analysis by fitting the linear elastic portions of the data with a best-fit line. The slopes of the data were used to calculate the strain response at the maximum load ranges for each of the tests. The same numbers of data points were included from each test, so the results would not be biased towards one set of data.

There is a discrepancy between the model used and the experimental composite samples. The model was designed to describe the deformations in a laminated structure composed of orthotropic layers of various orientations. The experimental samples are a laminated structure with woven plies, which do not behave in a consistent fashion to unidirectional plies. To approximate the behavior of the woven plies, each layer is modeled as two cross-ply layers.

Different data sets were created from each of the different materials tested. The inputs for each of the optimization techniques will be listed (geometric input values), and the output solutions. Tables of the start and output values with final error are given, and graphs of the solutions will be presented to illustrate the fit to the experimental results.

Selection of start values is important for the operation of the analysis. Poor start values can cause the program to converge to an incorrect solution, or at least cause the system to waste time in searching for values. A good estimate of the elastic properties is always the best start value, since the closer the start is to the solution, the more likely it is to find it. This seems obvious – if one already has a good idea of the properties, this analysis isn't necessary. This procedure has the capability of starting with bad start values and finding the right solution, but

the further away the start, the greater the possibility that the solution that is found is an alternate solution – as was seen with the results for Tube 2 in Table 4-VII. Experience with start values seems to indicate that conservative estimates work better. That is to say that starting smaller than the probable solution appears to work better than picking values known to be larger. The percentage change between using smaller or larger values of the material constants is best illustrated by examining their behavior. The difference value is found by subtracting the new strain response from that of the original, or:

$$e_i^{\exp} - e_i^{calc} \rightarrow e_i^{\exp} - \frac{s}{E_i} \propto 1 - \frac{1}{E_i} \tag{4.4}$$

where ε is the strain value, *exp* and *calc* refer to the experimental and calculated values, and E_i are the elastic properties. Since both the experimental strain and the stress values are constant, the change in the strain response should follow a (1-1/X) behavior (as shown in Equation (4.4)). Consequently, the sum of square errors should be proportional to $(1\text{-}1/X)^2$. Since all the independent variations to the different material values would be too numerous to include in this report, a plot of the variation of the SSE as a function of a constant times all of the material properties if found in Figure 4-4. The plot of the $(1\text{-}1/X)^2$ line is not to scale, but shown to illustrate the similar behavior. The small variations to the error values above one would be susceptible to change by the introduction of error in the measured values, and could result in the formation of a local minima that is not the ideal solution. The large variations to the values below one would be more tolerant of error effects, leading to a better chance of finding a minimum.

For each of the experimental results, several start values will be used for each material, and will include a set of values smaller than the anticipated values and a set of "good" estimates. The small values are found by simply choosing a number sufficiently smaller than the expected values, anywhere from a few percent to orders of magnitude, depending on the performance of the analysis. The good values are any value that can be considered a decent estimate of the expected value. For the unknown properties of the composite materials, micromechanics models will be used to calculate an estimate from the constituent properties.

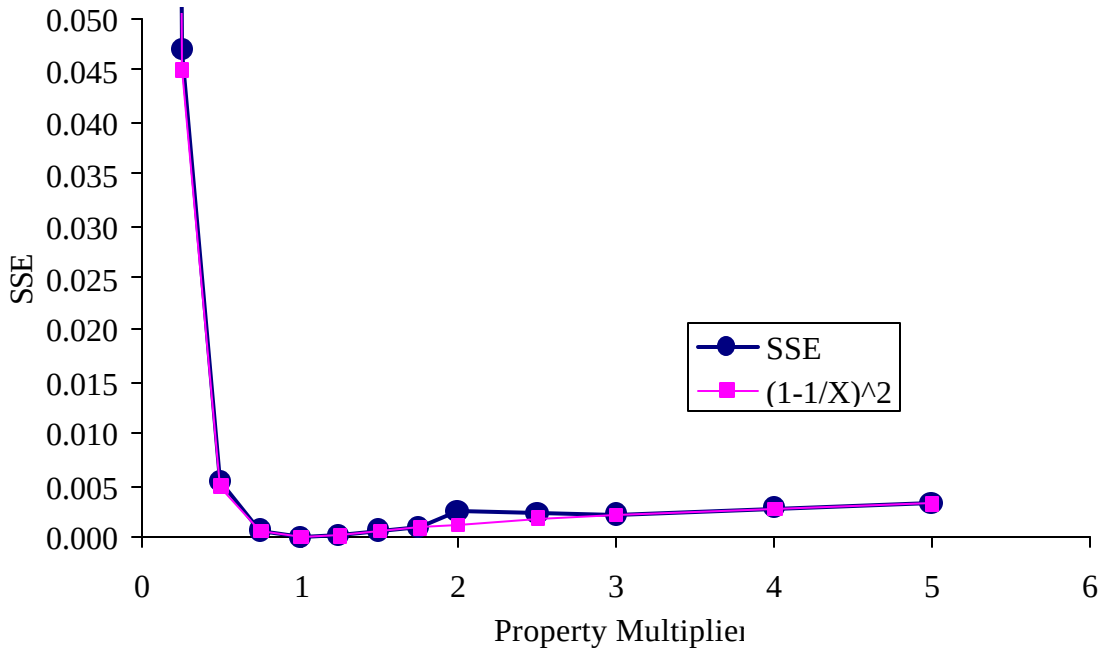


Figure 4-4. The error function for Thick Tube 2 plotted against the value of the multiplier on the elastic properties (2 = 2*Ei, i=1..7).

4.2.1 McDermott Technologies Hot Gas Filter

The geometric input values for the McDermott filter material are in Table 4-VIII. The winding angle variations follow the expression derived in the Introduction, and are calculated at every layer interfacial radius. The experimental load and strain response values are listed in Table 4-IX. The values were calculated from the average responses of all the tests, and since they are linear, only one value from each test is needed (the model passes through the origin giving a second point for a linear response).

Table 4-VIII. Geometric inputs for MTI

Inner radius- cm (in)	2.46 (0.968)
Outer radius - cm (in)	3.00 (1.18)
Number of Layers	36
Layer Thickness - cm (in)	0.015 (0.006)
Orientation	45º to 50º

Table 4-IX. Experimental Applied Loads and Measured Strains for the MTI materials

Loads				Strain		
P_o (MPa)	P_i (kPa)	Axial Load (kN)	Torque (N-m)	ε_x	ε_θ	$\gamma_{x\theta}$
0	690	0	0	-0.00064	0.000546	-7.69E-06
0	0	0.445	0	0.00011	-5.65E-05	2.96E-06
0	0	0	11.3	-1.78E-06	6.51E-07	5.97E-05

The Levenberg-Marquardt method optimizing the E_i values was used to find the best-fit elastic properties; the three different start values used are listed in Table 4-X. The first is a small start value, where the values should be smaller than the expected properties. The second is a set of start values calculated from a micromechanics approach developed by Huang [3,51]. The final set is found with the out-of-plane values set as constants. Plots of the experimentally observed strain response versus the predicted strain response for Solution Set #1 are in Figure 4-5 through Figure 4-7.

Table 4-X. Solutions for the McDermott Hot Gas Filter 7-6-16 with 10^5 simulated back-pulse cleaning cycles.

	Set 1		Set 2		Set 3	
Properties GPa (Msi)	Input	Output	Input	Output	Input	Output
E_1	0.69 (0.1)	26.75 (3.88)	29.65 (4.3)	26.75 (3.88)	29.65 (4.30)	35.03 (5.08)
E_2	0.07 (0.01)	8.62 (1.25)	2.30 (0.333)	8.62 (1.25)	2.30 (0.333)	10.62 (1.54)
E_3	0.07 (0.01)	0.10 (0.015)	2.30 (0.333)	0.10 (0.015)	0.42 (0.061)	0.42 (0.061)
G_{12}	0.28 (0.04)	1.26 (0.183)	1.15 (0.167)	1.26 (0.183)	1.15 (0.167)	1.14 (0.165)
ν_{12}	0.2	0.149	0.01	0.149	0.15	0.699
ν_{13}	0.15	-26.2	0.01	-26.2	0.01	0.01
ν_{23}	0.1	1.32	0.01	1.32	0.01	0.01
Error (SSE)	3.66E-5	1.34E-11	9.21E-9	1.34E-11	2.67E-9	9.70E-10

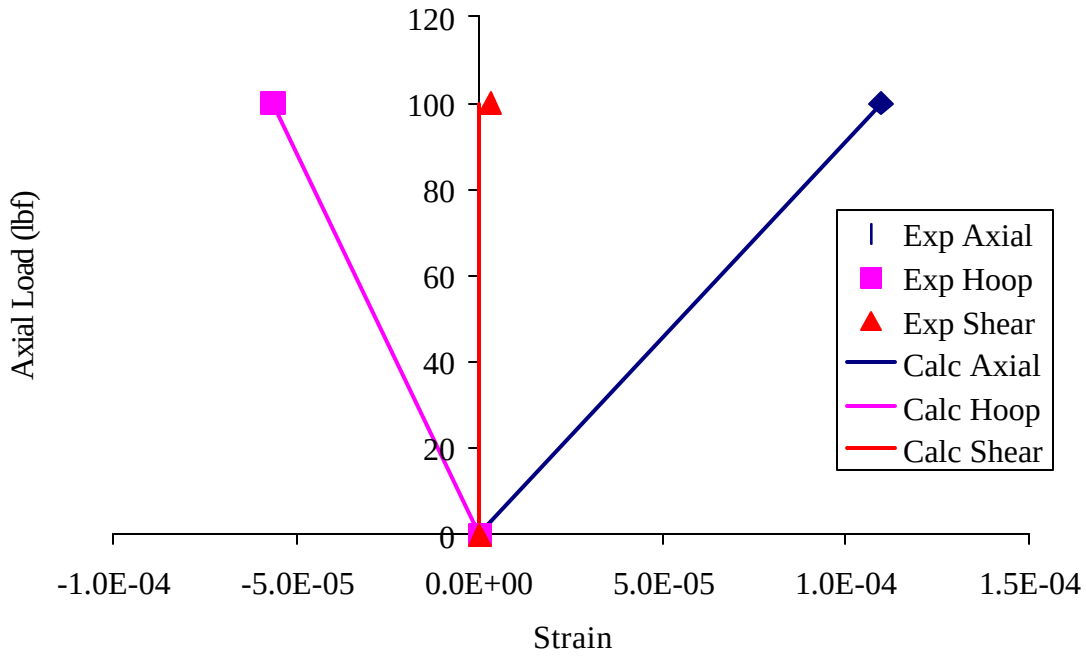


Figure 4-5. Axial tension test response – experimental data are represented by data points and the model predictions are lines

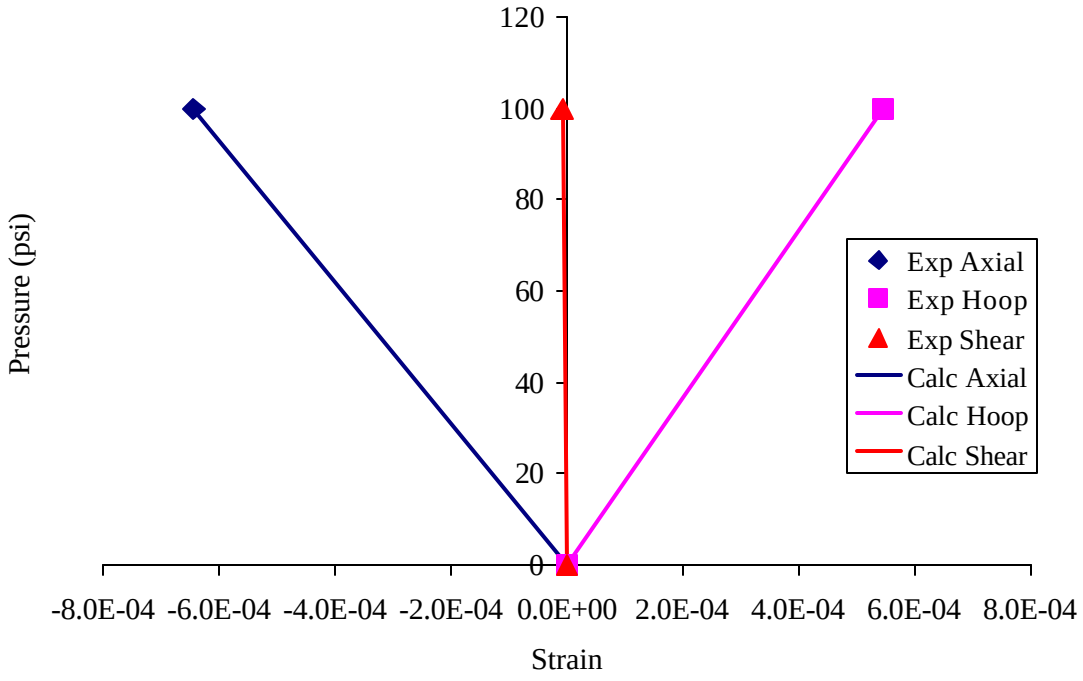


Figure 4-6. Internal pressure test response – experimental data are represented by data points and the model predictions are lines

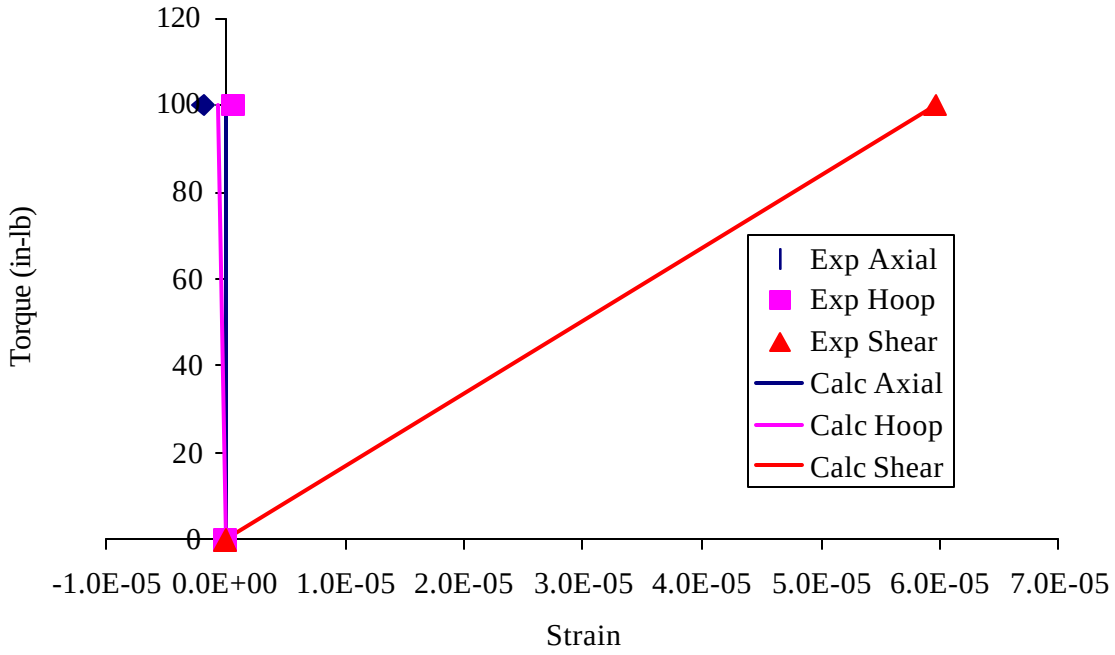


Figure 4-7. Axial torsion test response – experimental data are represented by data points and the model predictions are lines

The best-fit values accurately describe the response of the material, in that the lines for each of the strain values are very close to the experimental data points. The values for the in-plane responses are good, but the out-of-plane values are outside expected ranges.

4.2.2 ORNL Al_20_3/SiC Tubes

The geometric inputs for CVI 1219 are in Table 4-XI. The experimental inputs, loads and strains, are in Table 4-XII.

Table 4-XI. Geometric inputs for CVI 1219

Inner radius- cm (in)	2.5 (0.98)
Outer radius - cm (in)	2.95 (1.16)
Number of Layers	16
Layer Thickness - cm (in)	0.47 (0.187)
Orientation	±45

Table 4-XII. Experimental Applied Loads and Measured Strains

Loads				Strain		
P_o (MPa)	P_i (MPa)	Axial Load (kN)	Torque (N-m)	ε_x	ε_θ	$\gamma_{x\theta}$
0	4.14	0	0	-6.2E-05	0.000337	4.63E-05
0	6.21	0	0	-9.4E-05	0.000506	6.94E-05
0	0	-22.2	0	-0.00018	9.51E-05	-6.64E-07
0	0	22.2	0	1.84E-04	-9.51E-05	6.64E-07
0	0	0	-215	-4.44E-06	3.99E-06	-1.56E-04
0	0	0	215	4.44E-06	-3.99E-06	1.56E-04

The results for different sets of start values are in Table 4-XIII. The values in Set 1 are rough estimates of the measured stiffness values listed in Table 3-IV and Table 3-V, while those in Set 2 are rough estimates calculated using micromechanics models for E_1 and E_2:

$$E_1 = V_f E_f + (1 - V_f) E_m \tag{4.5}$$

$$\frac{1}{E_2} = \frac{1 - \sqrt{V_f}}{E_m} + \frac{\sqrt{V_f}}{E_f \sqrt{V_f} + \left(1 - \sqrt{V_f}\right) E_m} \tag{4.6}$$

where E_f and E_m are the modulus of the fiber and the matrix and V_f is the fiber volume fraction [39]. The modulus values used for the SiC and the Nextel 610 fiber are 414 and 372 GPa[1], respectively, and a fiber volume fraction of 0.32 was used. The input values used in Set 3 were solutions from the Nelder-Mead approach using the values from Set 1 as start values. As can be seen, the Nelder-Mead solution is not a minima, since the Newtonian method found a different set of values.

[1] Values taken from reports on the 3M website

Table 4-XIII. Input and output values for CVI 1219

	Set 1		Set 2		Set 3	
Properties GPa (Msi)	Input	Output	Input	Output	Input	Output
E_1	*172 (25.0)*	208 (30.1)	*414 (60.0)*	175 (25.5)	*199 (28.8)*	254 (36.9)
E_2	*76 (11.0)*	108 (15.7)	*276 (40.0)*	141 (20.4)	*53 (7.67)*	59 (8.59)
E_3	*69 (10.0)*	4964(720.0)	*276 (40.0)*	1855 (269.0)	*77 (11.2)*	155 (22.5)
G_{12}	*62 (9.0)*	65 (9.49)	*131 (19.0)*	66 (9.55)	*33 (4.84)*	62 (9.02)
ν_{12}	*0.15*	0.27	*0.1*	0.21	*-0.01*	0.38
ν_{13}	*0.15*	-0.13	*0.15*	-0.07	*0.23*	-0.67
ν_{23}	*0.25*	0.13	*0.25*	0.07	*0.63*	0.58
Error (SSE)		7.66E-09		7.59E-09		8.27E-09

The output values are different for each set of starting values. The only value that is consistent is the shear modulus value. The other in-plane values (1-2 values) are calculated with a range of values that are acceptable, while the out-of-plane values are not. The through-thickness stiffness (E_3) ranges in value from 155 GPa to 4964 GPa (22.5 Msi to 720 Msi), while the two Poisson's ratios (ν_{13} and ν_{23}) range from –0.67 to 0.58. These values are well outside the expected ranges for these values (E_3 E_2 and ν_{12} ν_{13}), and at this time no start values have been found that return all values in the expected ranges. This is not indicative that analysis has failed to minimize the error function. Figure 4-8 through Figure 4-10 are the graphs of the experimental data plotted with the calculated results. In each graph, the experimental strain measurements are plotted as data points, while the model predictions are the lines. In all cases the model matches the data very closely, with the possible exception of the shear strain values in Figure 4-10.

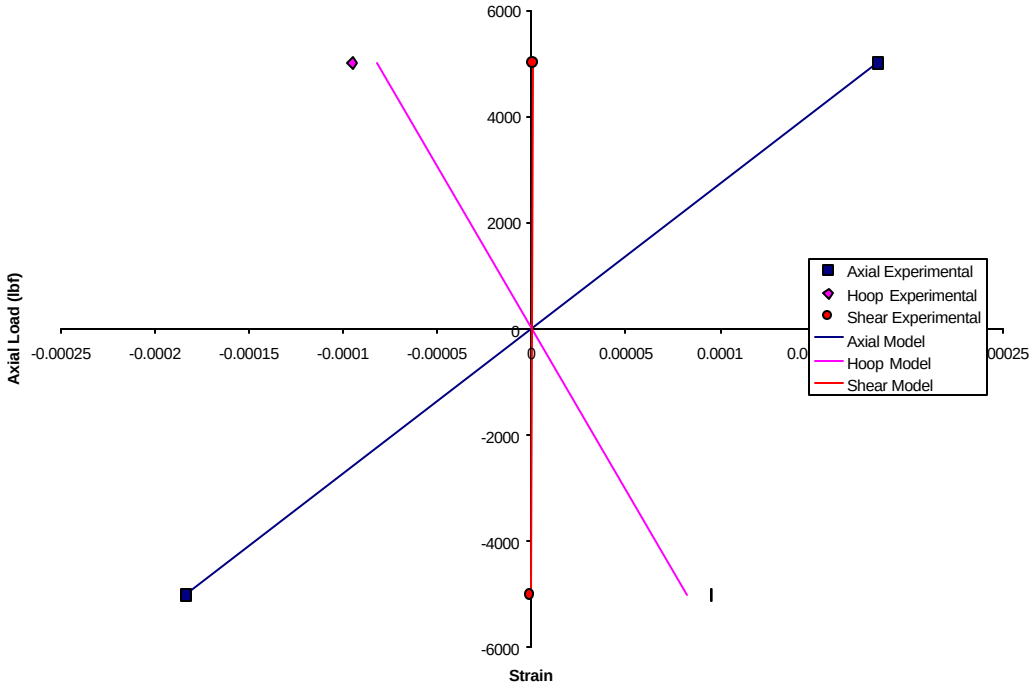


Figure 4-8. Axial tension and compression results – the data points are the experimentally observed values and the lines are the model predictions

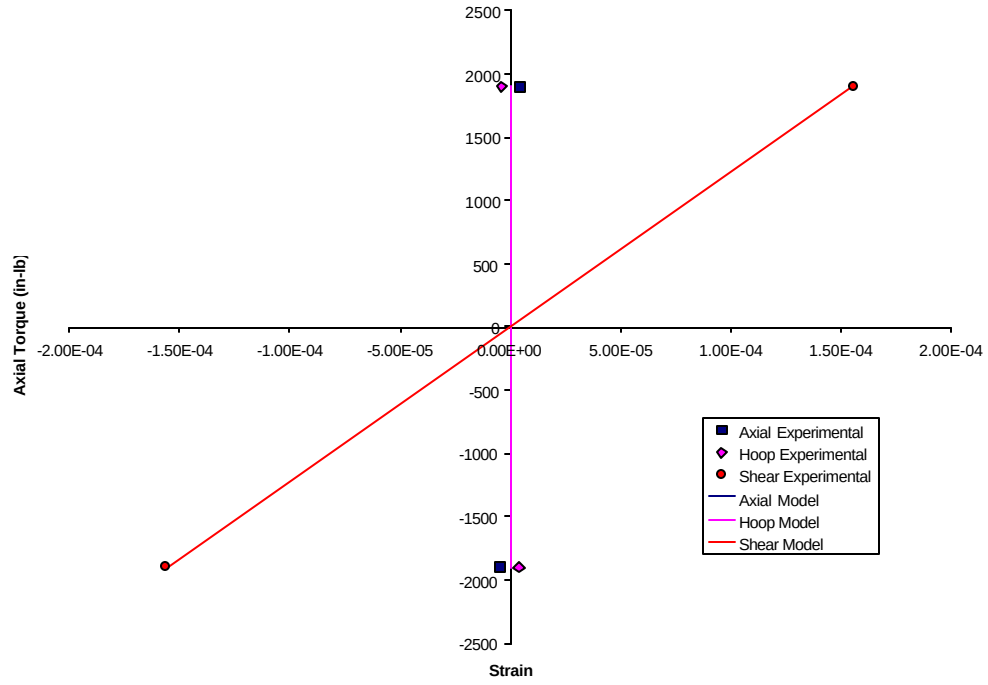


Figure 4-9. Axial torsion results– the data points are the experimentally observed values and the lines are the model predictions

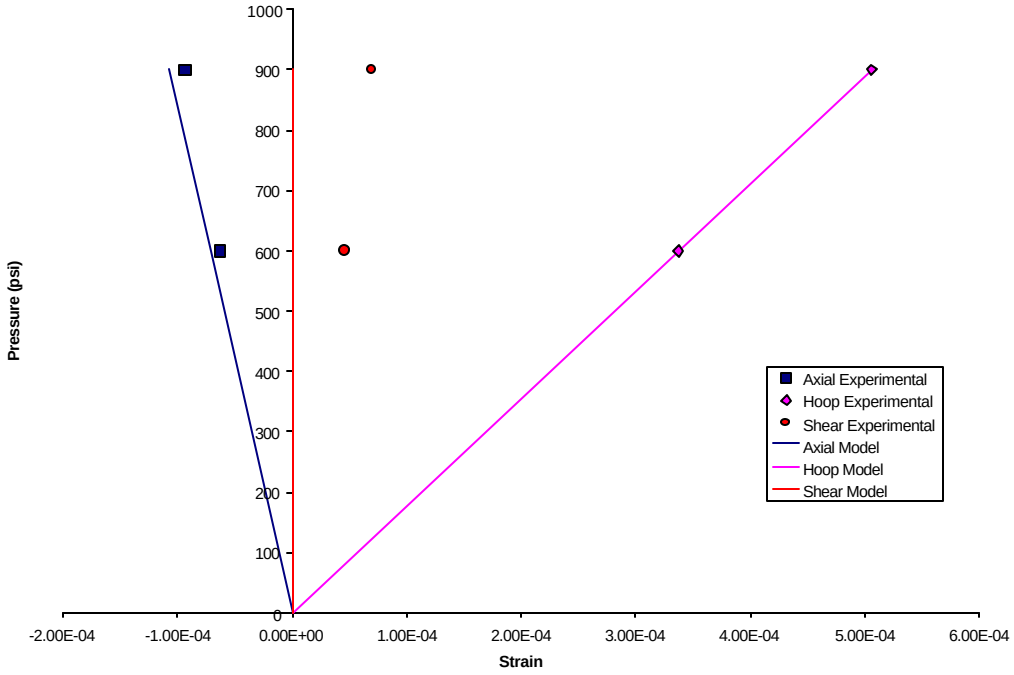


Figure 4-10. Internal pressure results – the data points are the experimentally observed values and the lines are the model predictions

4.2.3 Honeywell SiC/SiC Tubes

The geometric inputs for the Honeywell tubes are in Table 4-XIV. The applied loads and strain responses are in Table 4-XV. Unlike the data for the McDermott filters and the ORNL tubes, the responses include data from the inner surface and combined load conditions (Fx and Tx).

Table 4-XIV. Geometric inputs for the Honeywell tubes

Inner radius- cm (in)	2.51 (0.99)
Outer radius - cm (in)	2.90 (1.14)
Number of Layers	24
Layer Thickness - cm (in)	0.015 (0.006)
Orientation	[0/90]

Table 4-XV. Experimental Applied Loads and Measured Strains for the Honeywell materials

Loads				Strain			
P_o (MPa)	P_i (MPa)	Axial Load (kN)	Torque (N-m)	Surface	ε_x	ε_θ	$\gamma_{x\theta}$
0.0	3447.5	0.00	0.0	o	4.841E-06	1.702E-04	-6.836E-06
0.0	0.0	4.48	0.0	o	5.310E-05	-7.955E-06	-1.715E-05
0.0	0.0	0.00	113.0	o	-1.283E-07	-3.173E-06	1.463E-04
0.0	0.0	4.48	0.0	i	5.166E-05	-6.477E-06	-4.582E-06
0.0	0.0	0.00	113.0	i	3.793E-10	-2.570E-07	1.270E-04
0.0	0.0	2.23	56.2	o	2.609E-05	-4.160E-06	7.231E-05
0.0	0.0	-2.26	-56.0	o	-2.543E-05	3.541E-06	-7.445E-05
0.0	0.0	2.27	-55.8	o	2.693E-05	-4.049E-06	-7.371E-05
0.0	0.0	-2.24	56.1	o	-2.744E-05	4.484E-06	7.210E-05
0.0	0.0	2.23	55.9	i	2.788E-05	-5.629E-06	-6.044E-05
0.0	0.0	-2.24	-56.3	i	-2.991E-05	6.486E-06	6.172E-05
0.0	0.0	2.27	-56.5	i	2.187E-05	2.139E-07	6.636E-05
0.0	0.0	-2.22	55.4	i	-2.549E-05	1.750E-06	-6.428E-05

The results for the three sets of start values are listed in Table 4-XVI. As was seen in the results for Tube 2, there exist two different solutions. Set 2 and 3 find similar solutions, where the Set 1 values for E_1 and E_2 are the average of those in Set 2.

Table 4-XVI. Results for the Honeywell tubes

	Set 1		Set 2		Set 3	
Properties GPa (Msi)	Input	Output	Input	Output	Input	Output
E_1	137.90 (20.0)	134.45 (19.5)	206.85 (30.0)	199.27 (28.9)	2.07 (0.3)	201.33 (29.2)
E_2	137.90 (20.0)	129.63 (18.8)	68.95 (10.0)	67.09 (9.73)	0.69 (0.1)	73.78 (10.7)
E_3	137.90 (20.0)	373.02 (54.1)	68.95 (10.0)	436.45 (6.33)	0.69 (0.1)	181.34 (26.3)
G_{12}	48.27 (7.0)	84.12 (12.2)	62.06 (9.0)	82.74 (12.0)	0.62 (0.09)	84.12 (12.2)
ν_{12}	0.2	0.048	0.1	0.085	0.1	0.101
ν_{13}	0.29	0.600	0.25	0.676	0.25	1.053
ν_{23}	0.29	0.596	0.25	0.396	0.25	0.658
Error (SSE)		5.09E-08		5.10E-08		5.09E-08

The out-of-plane values again are outside of expected ranges, but are not outrageously so, as for the McDermott ORNL materials. The analysis was repeated setting the out-of-plane values as constants (number of active parameters (NAP) = 4), with E_3 set approximately equal to E_2 from the Set 2 solution and the Poisson ratios set to 0.2 and 0.5.

Table 4-XVII. Results for Honeywell tubes. Set 1 NAP =7, Set 2 and 3 NAP=4 (In-plane values only)

	Set 1		Set 2		Set 3	
Properties GPa (Msi)	Input	Output	Input	Output	Input	Output
E_1	206.9 (30.0)	199.3 (28.9)	206.9 (30.0)	250.3 (36.3)	206.9 (30.0)	245.5 (35.6)
E_2	68.95 (10.0)	66.33 (9.62)	68.95 (10.0)	2.75 (0.399)	68.95 (10.0)	17.24 (2.5)
E_3	68.95 (10.0)	413.0 (60.0)	68.95 (10.0)	68.95 (10.0)	68.95 (10.0)	68.95 (10.0)
G_{12}	84.12 (12.2)	84.12 (12.2)	84.12 (12.2)	84.12 (12.2)	84.12 (12.2)	84.12 (12.2)
ν_{12}	0.1	0.086	0.1	2.217	0.1	0.37
ν_{13}	0.2	0.694	0.2	0.200	0.5	0.50
ν_{23}	0.2	0.404	0.2	0.200	0.5	0.50
Error (SSE)		5.09E-08		5.10E-08		5.09E-08

With the reduction of active parameters, the out-of-plane properties are set as constants and the remaining values are optimized. By examining the solutions for Set 2 and Set 3, one notices the in-plane values change to compensate for the lack of contribution from the out-of-plane values.

4.2.4 Control Sample

The geometric inputs for the control sample are in Table 4-XVIII. The experimental load conditions and strain response is in Table 4-XIX. As with the Honeywell material, the experimental results contain combined axial load and torque, giving more information for the analysis, but without access to the inner surface, the strain response is only from the outer surface.

Table 4-XVIII. Geometric inputs for the Control Sample

Inner radius- cm (in)	2.73 (1.07)
Outer radius - cm (in)	3.00 (1.18)
Number of Layers	1
Layer Thickness - cm (in)	0.28 (0.11)
Orientation	0

Table 4-XIX. Experimental Applied Loads and Measured Strains for the Control Sample

Loads				Strain			
P_o (MPa)	P_i (MPa)	Axial Load (kN)	Torque (N-m)	Surface	ε_x	ε_θ	$\gamma_{x\theta}$
0.0	2758.0	0.00	0.0	o	-2.692E-05	1.281E-04	1.267E-05
0.0	0.0	44.35	0.0	o	4.601E-04	-1.284E-04	-1.036E-05
0.0	0.0	0.00	418.1	o	-1.769E-06	-1.320E-06	4.194E-04
0.0	0.0	4.48	55.7	o	4.468E-05	-1.277E-05	5.353E-05
0.0	0.0	3.37	112.5	o	3.333E-05	-1.026E-05	1.094E-04
0.0	0.0	4.51	112.5	o	4.495E-05	-1.355E-05	1.093E-04
0.0	0.0	4.52	-112.5	o	4.556E-05	-1.256E-05	-1.124E-04
0.0	0.0	8.92	112.8	o	9.073E-05	-2.253E-05	1.136E-04
0.0	0.0	-8.91	-113.2	o	-9.119E-05	2.761E-05	-1.098E-04
0.0	0.0	-8.93	111.8	o	-9.119E-05	2.662E-05	1.128E-04
0.0	0.0	8.96	-112.0	o	9.124E-05	-2.167E-05	-1.092E-04

With the isotropy (or at least near-isotropy) of the metal, estimates of the properties were entered for Set 1. For the second data set, the out-of-plane values were set to be constant, and the process repeated.

Table 4-XX. Results for the control sample Set 1 NAP =7, Set 2 NAP=4 (In-plane values only).

	Set 1		Set 2	
Properties GPa (Msi)	Input	Output	Input	Output
E_1	186.2 (27.0)	190.85 (27.7)	186.17 (27.0)	190.99 (27.7)
E_2	172.38 (25.0)	194.44 (28.2)	172.38 (25.0)	199.96 (29.0)
E_3	172.38 (25.0)	-4467.27 (-648)	193.06 (28.0)	193.06 (28.0)
G_{12}	68.95 (10.0)	72.81 (10.6)	68.95 (10.0)	73.09 (10.6)
ν_{12}	0.25	0.273	0.25	0.272584
ν_{13}	0.3	1.066	0.27	0.27
ν_{23}	0.3	6.575E+09	0.27	0.27
Error (SSE)		4.165E-10		4.70E-10

4.3 Error Sensitivity

As with any regression analysis, linear or nonlinear, the quality of the estimates for the variables depends upon the quality and quantity of the data. When the data have a large amount of error or scatter, it is advantageous to increase the amount of data to improve the fit. For the experimental work done here, there is a limit to the amount of data due to the limits of the model and experimental procedures. To illustrate the effects of different forms of error on the

regression analysis, simulations using the elasticity model were developed, and the strain results adjusted to illustrate two different forms of error. One is called a proportional error, which is a straight percentage variation to the strain. The second will be the addition of a fixed level of error, which adversely affects the smaller strain measurements more than the large values. This is analogous to signal noise affects for the different measurements employed. For both conditions the RAND() function from Microsoft Excel will be utilized to calculate the error value. It generates a flat distribution between 0 and 1 (all values between 1 and 0 have the same probability).

This does not address several other forms of error that could be expected in the experimental data. This assumes that all the tests used to generate the results are perfectly done. This is to say that all tests generate the exact same amount of error in the data, and it is distributed randomly throughout all the values. For the axial tension and torsion tests this is probably not a bad assumption since they are performed at the same time using the same MTS system controls, but it might not be applicable to the internal pressure tests. The pressure calculated by the elastomer plug method here matches the data well for the control sample, but with the rougher surfaces of the composite materials there is probably a small axial load generated. The load would change the axial and hoop strain results, biasing the data.

A bonus to the recording of the data was the use of 4 rosettes to record the stain. This was primarily to remove elastic bending contributions due to specimen misalignment, but has the added benefit of reducing several sources of experimental error. Random noise and orientation variations will be reduced since the stains are the average of four gages. Another method of reducing experimental error is the averaging of the slopes for the repeated tests on each sample. For each test, the data are recorded, a best-fit line is found and the slope is recorded, and the slopes are averaged.

4.3.1 Proportional Error

As was described earlier, a proportionate error was included to test for sensitivity to error. This generated an error such that all strains were within fixed percentage range of the correct strain values. Microsoft Excel worksheets were used to tabulate the data, and the random

number function, RAND, was used to generate the distribution. Since the RAND function is a distribution between 0 and 1, by shifting the values by 1-2RAND, as seen in Equation (4.7), will give a new distribution from 1 to –1. This will allow for the median value to correspond to the original strain value, and the distribution to be between the percentage factor, %δ, or [ε(1−δ) ≤ ε' ≤ ε(1+δ)].

$$e' = e\left(1 + \%d\left(1 - 2\,RAND\right)\right) \tag{4.7}$$

Error values of 0.1, 1, and 10% will be used for this illustration, and are listed in Table 4-XXI through Table 4-XXVI. The range for the percentage error should be excess of what could be seen in experimental results (0.1% is better than what can be expected and 10% is the accepted upper limit). In each table are the load condition (see Table 4-III), the surface of the readings, the altered strain values, and the percentage change from the values in Table 4-IV.

Table 4-XXI. Strain values for Tube 1 with 0.1% random error

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_q	γ_{xq}	ε_x	ε_q	γ_{xq}
1	o	-5.573E-03	8.433E-03	-5.982E-06	-0.028	-0.058	-0.086
2	o	1.041E-03	-6.421E-04	2.017E-06	-0.035	-0.051	0.082
3	o	1.443E-06	-4.296E-07	7.403E-04	0.013	-0.054	-0.030
4	o	1.042E-03	-6.424E-04	7.426E-04	-0.010	-0.032	-0.065
5	o	-4.532E-03	7.785E-03	7.366E-04	-0.069	0.010	-0.058
1	i	-5.569E-03	9.470E-03	-5.437E-06	0.036	-0.075	-0.052
2	i	1.041E-03	-6.877E-04	1.836E-06	-0.068	0.038	-0.061
3	i	1.444E-06	-5.277E-07	6.724E-04	-0.064	0.019	0.065
4	i	1.042E-03	-6.884E-04	6.749E-04	0.038	0.010	-0.042
5	i	-4.528E-03	8.776E-03	6.698E-04	0.020	-0.025	-0.084

Table 4-XXII. Strain values for Tube 1 with 1.0% random error

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_q	γ_{xq}	ε_x	ε_q	γ_{xq}
1	o	-5.582E-03	8.399E-03	-5.979E-06	-0.196	0.345	-0.031
2	o	1.035E-03	-6.367E-04	2.023E-06	0.503	0.793	-0.227
3	o	1.431E-06	-4.303E-07	7.435E-04	0.824	-0.225	-0.459
4	o	1.040E-03	-6.406E-04	7.369E-04	0.239	0.254	0.703
5	o	-4.564E-03	7.813E-03	7.367E-04	-0.762	-0.348	-0.080
1	i	-5.540E-03	9.505E-03	-5.400E-06	0.557	-0.449	0.632
2	i	1.033E-03	-6.914E-04	1.848E-06	0.687	-0.495	-0.726
3	i	1.436E-06	-5.245E-07	6.689E-04	0.510	0.637	0.574
4	i	1.032E-03	-6.838E-04	6.745E-04	0.930	0.673	0.016
5	i	-4.515E-03	8.690E-03	6.759E-04	0.304	0.959	-0.994

Table 4-XXIII. Strain values for Tube 1 with 10.0% random error

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_q	γ_{xq}	ε_x	ε_q	γ_{xq}
1	o	-6.023E-03	8.225E-03	-5.873E-06	-8.105	2.412	1.743
2	o	1.102E-03	-6.014E-04	1.956E-06	-5.932	6.293	3.111
3	o	1.576E-06	-4.638E-07	7.649E-04	-9.210	-8.022	-3.357
4	o	1.079E-03	-6.520E-04	7.638E-04	-3.570	-1.514	-2.920
5	o	-4.947E-03	8.304E-03	7.105E-04	-9.230	-6.648	3.475
1	i	-5.025E-03	9.161E-03	-5.698E-06	9.803	3.191	-4.852
2	i	1.054E-03	-7.058E-04	1.901E-06	-1.294	-2.595	-3.596
3	i	1.429E-06	-4.803E-07	6.396E-04	0.996	9.012	4.932
4	i	1.094E-03	-7.318E-04	6.897E-04	-5.025	-6.297	-2.236
5	i	-4.407E-03	8.825E-03	6.310E-04	2.704	-0.583	5.708

Table 4-XXIV. Strain values for Tube 2 with 0.1% random error

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_q	γ_{xq}	ε_x	ε_q	γ_{xq}
1	o	-3.351E-04	3.438E-03	-1.177E-18	-0.082	0.041	0.000
2	o	4.250E-04	-5.391E-05	2.216E-20	0.004	0.043	-0.017
3	o	1.586E-20	-9.148E-20	2.672E-03	-0.098	0.095	-0.029
4	o	4.252E-04	-5.396E-05	2.672E-03	-0.021	-0.057	-0.046
5	o	9.027E-05	3.384E-03	2.673E-03	-0.052	0.042	-0.093
1	i	-3.351E-04	3.980E-03	-1.070E-18	-0.065	-0.095	0.021
2	i	4.247E-04	-4.136E-05	2.016E-20	0.085	-0.030	-0.084
3	i	1.586E-20	-1.040E-19	2.429E-03	-0.089	-0.088	-0.031
4	i	4.251E-04	-4.137E-05	2.426E-03	-0.002	-0.065	0.083
5	i	9.023E-05	3.935E-03	2.429E-03	-0.008	0.022	-0.031

Table 4-XXV. Strain values for Tube 2 with 1.0% random error

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_q	γ_{xq}	ε_x	ε_q	γ_{xq}
1	o	-3.364E-04	3.422E-03	-1.177E-18	-0.452	0.503	0.039
2	o	4.248E-04	-5.445E-05	2.215E-20	0.054	-0.968	0.034
3	o	1.572E-20	-9.080E-20	2.660E-03	0.747	0.833	0.402
4	o	4.219E-04	-5.361E-05	2.680E-03	0.753	0.594	-0.353
5	o	8.948E-05	3.411E-03	2.696E-03	0.821	-0.732	-0.951
1	i	-3.368E-04	3.947E-03	-1.072E-18	-0.585	0.748	-0.161
2	i	4.222E-04	-4.112E-05	2.029E-20	0.678	0.550	-0.728
3	i	1.588E-20	-1.048E-19	2.440E-03	-0.239	-0.877	-0.498
4	i	4.221E-04	-4.165E-05	2.432E-03	0.687	-0.720	-0.172
5	i	8.944E-05	3.931E-03	2.420E-03	0.863	0.108	0.315

Table 4-XXVI. Strain values for Tube 2 with 10.0% random error

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_q	γ_{xq}	ε_x	ε_q	γ_{xq}
1	o	-3.564E-04	3.126E-03	-1.123E-18	-6.448	9.124	4.558
2	o	4.350E-04	-4.862E-05	2.284E-20	-2.334	9.856	-3.101
3	o	1.597E-20	-8.885E-20	2.875E-03	-0.799	2.969	-7.651
4	o	4.038E-04	-5.851E-05	2.632E-03	5.008	-8.488	1.469
5	o	8.173E-05	3.242E-03	2.693E-03	9.408	4.232	-0.821
1	i	-3.446E-04	4.284E-03	-1.083E-18	-2.917	-7.727	-1.260
2	i	4.523E-04	-4.330E-05	1.972E-20	-6.411	-4.730	2.072
3	i	1.676E-20	-1.074E-19	2.609E-03	-5.808	-3.369	-7.473
4	i	3.986E-04	-3.955E-05	2.480E-03	6.234	4.352	-2.163
5	i	8.194E-05	3.846E-03	2.242E-03	9.178	2.269	7.675

For each of the error values, the strains are input with the same starting values used in Table 4-V and Table 4-VI. The number of iterations was limited to 1000 (except for one run of Tube 2 data)

Table 4-XXVII. Effect of the different forms of Error on the property estimate using LM-E_i

Correct Values		Tube 1			Tube 2		
	GPa	0.10%	1.0%	10.0%	0.10%	1.0%	10.0%
E_1	43.51	43.71	12.62	0.14	43.99	44.06	Singular
E_2	11.51	11.31	42.06	48.27	10.96	11.24	Matrix
E_3	11.51	15.17	4.08	0.01	10.89	1.46	
G_{12}	3.45	3.45	3.45	3.45	3.45	3.44	
ν_{12}	0.27	0.278	0.069	-0.006	0.284	0.241	
ν_{13}	0.40	0.437	-0.368	-3.466	0.469	-2.132	
ν_{23}	0.40	0.426	0.831	-9.072	0.338	2.184	
Final SSE		4.54E-11	1.21E-08	9.79E-07	1.67E-11	1.96E-09	
Iterations		11	67	1000 (max)	15	15	
Time (minutes)		1.17	3.41	61.42	1.28	1.15	

Table 4-XXVIII. Effect of the different forms of error on the property estimates using LM-C_{ij}

	Correct	Tube 1			Tube 2		
	Values	0.10%	1.0%	10.0%	0.10%	1.0%	10.0%
E_1	43.51	43.71	12.62	-2.62	41.85	64.05	-140.66
E_2	11.51	11.31	42.06	51.64	13.24	-1.28	197.89
E_3	11.51	15.17	4.08	-0.35	10.89	-0.34	-36.96
G_{12}	3.45	3.45	3.45	3.45	3.45	3.44	3.39
ν_{12}	0.27	0.278	0.069	-0.006	0.235	-2.427	0.021
ν_{13}	0.40	0.437	-0.368	-3.487	0.441	3.230	1.993
ν_{23}	0.40	0.426	0.831	-1.380	0.376	2.016	1.205
Final SSE		4.54E-11	1.21E-08	9.79E-07	1.67E-11	1.96E-09	2.08E-07
Iterations		16	17	18	48	319	1500 (max)
Time (minutes)		1.15	1.34	1.19	2.5	13.14	60.6

The values of the final solution quickly deviate from the correct values with the introduction of error in the data. Since this data are a random distribution around the correct value, a larger data sets should remove this error, as can be seen in Table 4-XXIX. A data set of 100 different load conditions (formed using the same method as for the smaller sets of 10 loading conditions) was input into the analysis and given the same start values. The solutions are more accurate, and will continue to improve as the data set is increased.

Table 4-XXIX. Inversion results using a larger data set, 100 Loading Conditions – LM-E_i

	Correct	**Tube 1**		
	Values	**0.10%**	**1.0%**	**10.0%**
E_1	43.51	43.78	46.27	15.31
E_2	11.51	11.24	8.48	38.41
E_3	11.51	13.51	7.72	2.28
G_{12}	3.45	3.45	3.45	3.42
ν_{12}	0.27	0.277265	0.367	0.063
ν_{13}	0.40	0.455982	0.358	-1.00
ν_{23}	0.40	0.359598	0.370	1.64
Final SSE		2.02E-09	1.97E-07	1.89E-05
Iterations		21	60	54
Time (minutes)		14.8	32.4	35.7

4.3.2 Additive or Noise Error

A more realistic form of error would be noise introduced in the readings of the strain and load from the strain gage amplifiers or load cells. This type of error wouldn't be proportional to

the incoming signal, but a type that would affect the smaller measurements more than the large. To simulate this, the strain values will be shifted by adding a small, random value, instead of by multiplying by a percentage term as used in the previous section. This value will be a much larger percentage change to small values but a smaller change to the larger values, as random noise would do to experimental measurements. The RAND function was used to generate a random scaling factor such that the range would be $[\varepsilon-\delta \le \varepsilon' \le \varepsilon+\delta]$ or:

$$e' = e + d\left(1 - 2RAND\right) \tag{4.8}$$

The values for delta will range from $1*10^{-6}$ to $1*10^{-4}$, or 1 to 100 microstrain. The new strain data sets are listed in Table 4-XXX through Table 4-XXXV. The percentage change from the original data, as listed in Table 4-IV, is usually small except for the conditions that contain a near-zero strain component (i.e. shear strain in axial tension).

Table 4-XXX. Strain values for Tube 1 with $d=10^{-6}$

		Strain Values			Percent Change to Original		
Load	**Surface**	**e_x**	**e_q**	**g_{xq}**	**e_x**	**e_q**	**g_{xq}**
1	o	-5.571E-03	8.428E-03	-6.333E-06	0.010	0.000	-5.943
2	o	1.041E-03	-6.425E-04	2.822E-06	-0.082	-0.102	-39.8
3	o	1.415E-06	-1.345E-06	7.409E-04	1.950	-213.4	-0.109
4	o	1.043E-03	-6.423E-04	7.428E-04	-0.086	-0.010	-0.096
5	o	-4.529E-03	7.786E-03	7.362E-04	0.013	-0.003	-0.015
1	i	-5.571E-03	9.463E-03	-5.607E-06	0.005	-0.003	-3.189
2	i	1.040E-03	-6.877E-04	8.371E-07	0.049	0.037	54.4
3	i	1.944E-06	-3.566E-07	6.723E-04	-34.7	32.4	0.070
4	i	1.042E-03	-6.894E-04	6.750E-04	0.024	-0.126	-0.058
5	i	-4.529E-03	8.774E-03	6.682E-04	-0.002	0.002	0.149

Table 4-XXXI. Strain values for Tube 1 with $d=10^{-5}$

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_θ	$\gamma_{x\theta}$	ε_x	ε_θ	$\gamma_{x\theta}$
1	o	-5.562E-03	8.435E-03	-3.532E-06	0.171	-0.079	40.9
2	o	1.034E-03	-6.415E-04	1.888E-06	0.628	0.055	6.462
3	o	-3.610E-06	4.890E-06	7.395E-04	350.1	1239.0	0.085
4	o	1.044E-03	-6.448E-04	7.515E-04	-0.191	-0.405	-1.263
5	o	-4.528E-03	7.783E-03	7.280E-04	0.029	0.038	1.101
1	i	-5.577E-03	9.464E-03	-2.675E-06	-0.096	-0.018	50.77
2	i	1.048E-03	-6.864E-04	-3.269E-06	-0.757	0.231	278.2
3	i	9.853E-06	3.122E-06	6.769E-04	-582.6	691.5	-0.612
4	i	1.048E-03	-6.977E-04	6.833E-04	-0.553	-1.344	-1.277
5	i	-4.533E-03	8.771E-03	6.625E-04	-0.090	0.036	0.999

Table 4-XXXII. Strain values for Tube 1 with $\delta=10^{-4}$

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_θ	$\gamma_{x\theta}$	ε_x	ε_θ	$\gamma_{x\theta}$
1	o	-5.541E-03	8.432E-03	7.869E-05	0.540	-0.041	1416.4
2	o	9.698E-04	-5.523E-04	-2.938E-05	6.805	13.95	1555.5
3	o	3.643E-05	-4.908E-05	7.329E-04	-2423.8	-11330.8	0.977
4	o	9.637E-04	-6.350E-04	8.355E-04	7.519	1.128	-12.583
5	o	-4.477E-03	7.695E-03	8.080E-04	1.155	1.168	-9.763
1	i	-5.599E-03	9.384E-03	-9.803E-05	-0.489	0.833	-1704.0
2	i	1.040E-03	-6.701E-04	3.606E-05	0.060	2.594	-1865.5
3	i	-8.301E-05	-7.084E-05	7.105E-04	5851.1	-13320.3	-5.597
4	i	1.007E-03	-6.218E-04	7.476E-04	3.337	9.689	-10.81
5	i	-4.542E-03	8.832E-03	6.870E-04	-0.279	-0.657	-2.660

Table 4-XXXIII. Strain values for Tube 2 with $\delta=10^{-6}$

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_θ	$\gamma_{x\theta}$	ε_x	ε_θ	$\gamma_{x\theta}$
1	o	-3.349E-04	3.441E-03	-3.161E-07	0.00	-0.02	-2.69E+13
2	o	4.260E-04	-5.463E-05	6.046E-07	-0.22	-1.29	-2.73E+15
3	o	-4.929E-07	1.057E-07	2.670E-03	3.11E+15	1.15E+14	0.01
4	o	4.253E-04	-5.472E-05	2.672E-03	-0.06	-1.47	-0.04
5	o	9.051E-05	3.386E-03	2.671E-03	-0.32	0.00	-0.01
1	i	-3.340E-04	3.976E-03	-8.325E-07	0.25	0.01	-7.78E+13
2	i	4.257E-04	-4.166E-05	-2.519E-07	-0.14	-0.76	1.25E+15
3	i	1.271E-07	-6.901E-07	2.428E-03	-8.02E+14	-6.64E+14	-0.01
4	i	4.249E-04	-4.160E-05	2.428E-03	0.04	-0.61	0.00
5	i	9.049E-05	3.935E-03	2.429E-03	-0.30	0.01	-0.04

Table 4-XXXIV. Strain values for Tube 2 with $\delta=10^{-5}$

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_θ	$\gamma_{x\theta}$	ε_x	ε_θ	$\gamma_{x\theta}$
1	o	-3.327E-04	3.449E-03	5.931E-06	0.63	-0.26	5.04E+14
2	o	4.315E-04	-5.254E-05	4.454E-06	-1.51	2.58	-2.01E+16
3	o	-7.945E-06	5.296E-06	2.669E-03	5.02E+16	5.78E+15	0.08
4	o	4.170E-04	-5.864E-05	2.671E-03	1.90	-8.73	-0.01
5	o	9.945E-05	3.383E-03	2.674E-03	-10.23	0.07	-0.11
1	i	-3.294E-04	3.976E-03	-8.699E-06	1.63	0.02	-8.13E+14
2	i	4.220E-04	-3.992E-05	-3.899E-06	0.72	3.46	1.94E+16
3	i	3.641E-06	1.310E-06	2.434E-03	-2.30E+16	1.26E+15	-0.26
4	i	4.312E-04	-3.150E-05	2.427E-03	-1.45	23.81	0.04
5	i	9.766E-05	3.933E-03	2.425E-03	-8.24	0.05	0.13

Table 4-XXXV. Strain values for Tube 2 with $\delta=10^{-4}$

		Strain Values			Percent Change to Original		
Load	Surface	ε_x	ε_θ	$\gamma_{x\theta}$	ε_x	ε_θ	$\gamma_{x\theta}$
1	o	-2.886E-04	3.523E-03	-6.711E-05	13.81	-2.41	-5.70E+15
2	o	3.411E-04	3.899E-05	-8.576E-05	19.76	172.29	3.87E+17
3	o	-3.790E-05	6.313E-05	2.641E-03	2.39E+17	6.89E+16	1.13
4	o	4.682E-04	-1.161E-04	2.762E-03	-10.14	-115.34	-3.42
5	o	1.824E-04	3.378E-03	2.669E-03	-102.13	0.22	0.05
1	i	-2.429E-04	4.056E-03	-7.699E-05	27.47	-2.00	-7.20E+15
2	i	4.626E-04	-7.879E-05	-5.107E-05	-8.82	-90.55	2.54E+17
3	i	5.289E-05	-7.744E-05	2.499E-03	-3.34E+17	-7.45E+16	-2.94
4	i	3.394E-04	-2.317E-05	2.378E-03	20.15	43.97	2.05
5	i	1.580E-04	3.980E-03	2.423E-03	-75.09	-1.12	0.19

Table 4-XXXVI. The effects of the different levels of error on the estimates using LM-E_i

	Correct	Tube 1			Tube 2		
	Values	1.00E-06	1.00E-05	1.00E-04	1.00E-06	1.00E-05	1.00E-04
E_1	43.51	45.02	48.40	0.32	43.99	Singular	52.06
E_2	11.51	9.86	5.72	27.58	10.96	Matrix	2.30
E_3	11.51	11.72	12.96	0.70	16.00		80.67
G_{12}	3.45	3.45	3.45	3.46	3.45		3.43
ν_{12}	0.27	0.316	0.549	0.075	0.287		1.166
ν_{13}	0.40	0.392	0.165	-0.623	0.721		0.802
ν_{23}	0.40	0.400	0.547	4.747	0.164		0.171
Final SSE		7.24E-12	8.25E-10	8.15E-08	6.27E-12		8.85E-08
Iterations		18	11	1000	13		315
Time (minutes)		1.45	1.07	41.97	1.20		13.67

Table 4-XXXVII. The effects of the different levels of error on the estimates using LM-C_{ij}

	Correct	Tube 1			Tube 2		
	Values	1.00E-06	1.00E-05	1.00E-04	1.00E-06	1.00E-05	1.00E-04
E_1	43.51	45.02	48.40	-26.06	37.92	48.40	-26.06
E_2	11.51	9.86	5.72	80.67	17.31	5.72	80.67
E_3	11.51	11.72	12.96	4.51	15.86	12.96	4.51
G_{12}	3.45	3.45	3.45	3.46	3.45	3.45	3.46
ν_{12}	0.27	0.316	0.549	0.050	0.182	0.549	0.050
ν_{13}	0.40	0.392	0.165	-1.285	0.616	0.165	-1.285
ν_{23}	0.40	0.400	0.547	1.742	0.272	0.547	1.742
Final SSE		7.24E-12	8.25E-10	8.14E-08	6.27E-12	8.25E-10	8.14E-08
Iterations		18	15	18	76	44	18
Time (minutes)		1.45	1.16	1.38	3.64	2.24	1.38

Table 4-XXXVIII Solutions found using larger data sets (100 Load Conditions) Using LM-Ei

	Correct	Tube 1		
	Values	**1.00E-06**	**1.00E-05**	**1.00E-04**
E_1	43.51	43.73	44.06	26.55
E_2	11.51	11.28	10.89	28.75
E_3	11.51	11.52	10.62	3.85
G_{12}	3.45	3.45	3.45	3.45
ν_{12}	0.27	0.276	0.286445	0.106
ν_{13}	0.40	0.406	0.422087	1.10
ν_{23}	0.40	0.394	0.382472	-0.454
Final SSE		2.054E-10	9.94E-09	9.21E-07
Iterations		22	16	47
Time (minutes)		16.1	12.96	25.2

The effects of the different types of error in the data are insignificant for the small error amount and severe for the larger two, in some cases causing the estimates to grow so large that they caused the program to crash. However, the effects of random error will be decreased with increasing data set sizes, as can be seen in the improved estimates in Table 4-XXXVIII. With an increasing amount of data, the random distributions average out to a more correct value, increasing the accuracy of the estimate.

4.3.3 Effect of error on estimating the different constants

While the data used in the previous section contained error randomly distributed in a range with the median being the accurate value, experimental data may contain error that bias the data such that the average is no longer accurate. The effects of the bias would cause the solution to change from the true values. Larger data sets would not improve the final solutions, since the data itself is no longer consistent with the desired result. By investigating the different experimental results and those from the simulated error effects, one can notice that the different constants are more or less sensitive to error. That is, the out-of-plane values begin to vary with the introduction of a small amount of error, while the in plane properties are, by comparison, relatively unaffected. It would be beneficial to estimate the scale of the contribution of each variable, and its variations, to the strain response of the tube. To accomplish this task, the

elasticity solution was used to calculate the strain response for the two tube geometries listed in Table 4-II, but with an added set of geometries of tubes with a wall thickness two and a half times thicker, as listed in Table 4-XXXIX. Table 4-XL contains the different load profiles for the tubes, which have been shifted to generate the same stress state at the outer surface for a more direct comparison of the strain responses.

Table 4-XXXIX. Geometry of the four different composite simulations

Tube ID	Inner Radius mm (in)	Outer Radius mm (in)	Number of Plies	Orientation	Ply Thickness mm (in)
Thin 1	25.4 (1.0)	27.9 (1.1)	20	$[45/-45]_{10}$	0.127 (0.005)
Thick 1	25.4 (1.0)	31.75 (1.25)	20	$[45/-45]_{10}$	0.3125 (0.0125)
Thin 2	25.4 (1.0)	27.9 (1.1)	20	$[0/90]_{10}$	0.127 (0.005)
Thick 2	25.4 (1.0)	31.75 (1.25)	20	$[0/90]_{10}$	0.3125 (0.0125)

Table 4-XL. Applied loads for each tube – thick values calculated to generate the same stress state as for the thin tubes

	Po	Pi	Fx	Tx	Po	Pi	Fx	Tx
	MPa	MPa	kN	N-m	MPa	MPa	kN	N-m
1	0.0	34.5	0.00	0.0	0.0	92.3	0.00	0.0
2	0.0	0.0	44.80	0.0	0.0	0.0	120.00	0.0
3	0.0	0.0	0.00	1130.0	0.0	0.0	0.00	3088.4
4	0.0	0.0	44.80	1130.0	0.0	0.0	120.00	3088.4
5	0.0	34.5	44.80	1130.0	0.0	92.3	120.00	3088.4

For each of the four tubes listed, the strain profiles were calculated for the original elastic properties, and repeated with each constant individually reduced by 50% (tables for each of the variations are in Appendix B). This should illustrate the contribution of each constant, and give an idea of the sensitivity of the strain measurements needed to find accurate estimates. Each of the tables contains the differences between the strain values found using the normal material properties and those with the reduced properties. The resulting percentage change is listed to give a gage for the significance of the variation. It is important to look at both values, since in many cases the largest percentage changes are on strains that are too small to be measured experimentally.

Table 4-XLI. Strain response for the different loading conditions for Thin Tube 1

Standard Response				
Load	Surface	Axial	Hoop	Shear
1	I	-1.92E-02	3.26E-02	-1.87E-05
1	o	-1.92E-02	2.91E-02	-2.06E-05
2	I	9.25E-03	-6.12E-03	1.63E-05
2	o	9.25E-03	-5.70E-03	1.79E-05
3	I	1.63E-05	-5.96E-06	7.60E-03
3	o	1.63E-05	-4.85E-06	8.36E-03
4	I	9.27E-03	-6.12E-03	7.62E-03
4	o	9.27E-03	-5.71E-03	8.38E-03
5	I	-9.95E-03	2.65E-02	7.60E-03
5	o	-9.95E-03	2.34E-02	8.36E-03

Table 4-XLII. Strain deviation from the Standard Response for the thin Tube 1 with $E_1 = E_1/2$

	Surface	Percentage Change from Standard			Deviation from Standard		
Load		Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-15.65	10.22	-76.25	-3.01E-03	-3.33E-03	-1.43E-05
1	o	-15.65	9.42	-76.25	-3.01E-03	-2.74E-03	-1.57E-05
2	I	9.41	-15.65	-33.42	-8.71E-04	-9.57E-04	5.45E-06
2	o	9.41	-13.74	-33.42	-8.71E-04	-7.84E-04	6.00E-06
3	I	-33.42	-76.25	76.65	5.45E-06	-4.55E-06	-5.83E-03
3	o	-33.42	-77.11	76.65	5.45E-06	-3.74E-06	-6.41E-03
4	I	9.34	-15.71	76.42	-8.65E-04	-9.62E-04	-5.82E-03
4	o	9.34	-13.79	76.42	-8.65E-04	-7.87E-04	-6.40E-03
5	I	-38.94	16.20	76.80	-3.87E-03	-4.29E-03	-5.84E-03
5	o	-38.94	15.10	76.79	-3.87E-03	-3.53E-03	-6.42E-03

Table 4-XLIII. Strain deviation from the Standard Response for the thin Tube 1 with $E_3 = E_3/2$

	Surface	Percentage Change from Standard			Deviation from Standard		
Load		Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.00	0.29	10.98	-9.00E-07	-9.37E-05	2.06E-06
1	o	0.00	-0.15	10.98	-9.00E-07	4.31E-05	2.26E-06
2	I	0.00	0.00	-0.05	-1.00E-08	-2.60E-07	7.90E-09
2	o	0.00	0.00	-0.05	-1.00E-08	2.40E-07	8.80E-09
3	I	-0.05	10.98	0.00	7.90E-09	6.55E-07	-3.00E-08
3	o	-0.05	-12.87	0.00	7.90E-09	-6.24E-07	-3.00E-08
4	I	0.00	0.01	0.00	0.00E+00	4.00E-07	-2.00E-08
4	o	0.00	-0.01	0.00	0.00E+00	-3.80E-07	-2.00E-08
5	I	-0.01	0.35	-0.03	-8.20E-07	-9.33E-05	2.04E-06
5	o	-0.01	-0.18	-0.03	-8.20E-07	4.27E-05	2.24E-06

The out-of-plane elastic constants, E_3, ν_{13}, and ν_{23}, do not contribute significantly for the thin materials. A 50% drop in any of these results in a 5% change, at best, to the observed strain response for all the measurable strain components. Some of the values listed Table 4-XLIII illustrate a percentage change of greater than 5%, but inspecting the difference value reveals these are changes too small to measure (10^{-6} strain or smaller). For the value of E_3, the change does not alter any of the measurable strain components by even 1%. Even for the thicker materials, which should be influenced by the out-of-plane properties more than the thin, these variations only grow to range between 1 and 5%. By comparison, the in-plane properties are more responsive. The changes in many of the strain responses are on the order of 30 to 75% for a 50% decrease in the E_1 value, and this behavior is similar for the other in-plane constants.

These results represent the changes due to a 50% drop in the elastic properties, and they can not be assumed to be symmetric about the value. The values listed in Table 4-XLIV and Table 4-XLV show the changes in the strain response for changes in E_3 from $E_3/2$ to $2*E_3$ (-50% to +100%). The variations are not symmetric, and agree with the (1-1/X) behavior in that the larger values do not alter the response as greatly as the smaller values do. The largest effect of being within a range of -50 to +200% of the accurate value is seen as a variation of only 3.61% to –1.83% change in the hoop strain response on the inner surface of an internal pressure test (which is the largest observable change).

Table 4-XLIV. Strain deviation from the Standard Response for the Thick Tube 2 with $E_3=E_3/2$

	Surface	Percentage Change from Standard			Deviation from Standard		
Load		Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.65	3.61	-0.29	5.79E-06	-6.14E-04	-1.09E-20
1	o	0.65	-1.94	-0.29	5.79E-06	2.29E-04	-1.37E-20
2	I	0.00	0.65	-0.03	-3.00E-08	1.84E-06	5.40E-23
2	o	0.00	-0.27	-0.03	-3.00E-08	-1.46E-06	6.70E-23
3	I	-0.03	-0.29	0.00	5.50E-23	-3.56E-21	0.00E+00
3	o	-0.03	0.32	0.00	5.50E-23	2.90E-21	0.00E+00
4	I	0.00	0.65	0.00	-3.00E-08	1.84E-06	0.00E+00
4	o	0.00	-0.27	0.00	-3.00E-08	-1.46E-06	0.00E+00
5	I	-0.20	3.66	0.00	5.76E-06	-6.13E-04	0.00E+00
5	o	-0.20	-2.02	0.00	5.76E-06	2.27E-04	0.00E+00

Table 4-XLV. Strain deviation from the Standard Response for the Thick Tube 2 with $E_3=2*E_3$

Load	Surface	Percentage Change from Standard			Deviation from Standard		
		Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-0.33	-1.83	0.15	-2.95E-06	3.11E-04	5.56E-21
1	o	-0.33	0.99	0.15	-2.95E-06	-1.17E-04	6.96E-21
2	I	0.00	-0.33	0.02	1.00E-08	-9.39E-07	-2.70E-23
2	o	0.00	0.14	0.02	1.00E-08	7.44E-07	-3.40E-23
3	I	0.02	0.15	0.00	-2.70E-23	1.81E-21	0.00E+00
3	o	0.02	-0.16	0.00	-2.70E-23	-1.48E-21	0.00E+00
4	I	0.00	-0.33	0.00	1.00E-08	-9.39E-07	0.00E+00
4	o	0.00	0.14	0.00	1.00E-08	7.44E-07	0.00E+00
5	I	0.10	-1.85	0.00	-2.94E-06	3.10E-04	0.00E+00
5	o	0.10	1.03	0.00	-2.94E-06	-1.16E-04	0.00E+00

The plot of the sensitivity of the tube response to changes in the elastic properties is illustrated in Figure 4-11. This is the plot of the hoop strain variation for a combined loading condition (#5) for the Thick Tube 2 sample, and it was chosen since it gives the largest variation in all the strain responses. The variations to in-plane values cause a large change in the strain response, while the out-of-plane property, E_3 in this case, only creates large variations as the value tends to zero.

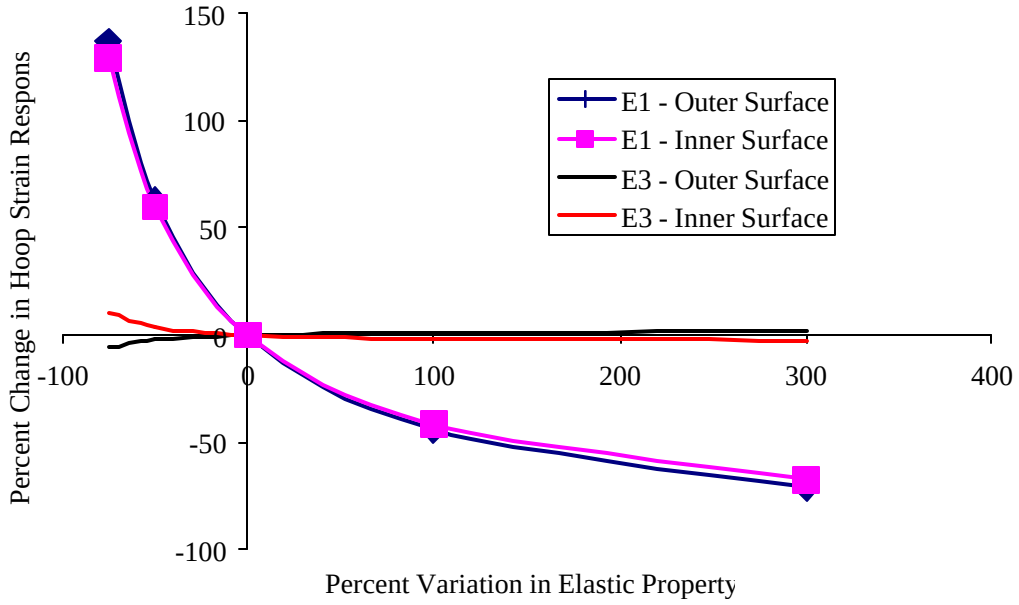


Figure 4-11. Effect on the Hoop Strain response by variations to E1 and E3 for the Thick Tube 2 Load Condition 5 (Pi, Fx, Tx simultaneously).

By switching the axes and zooming in on the origin of Figure 4-11, as seen in Figure 4-12, the plot illustrates the range of the elastic values that describe a given strain response within an error range. For an accuracy of ±5%, the variation of the in-plane values is roughly symmetric about the origin, and small. This illustrates that the in-plane values can be calculated with decent accuracy given small but non-trivial amount of error, since a small range of the in-plane elastic constant describe that range of the strain value. The out-of-plane properties do not exhibit the same behavior. They are not symmetric about the origin and exhibit large variations in values for small amounts of error. This requires a very high degree of accuracy to find an accurate estimate of the elastic constant value, since variation range of ±1 to 2% can result in property variation range well in excess of 100%.

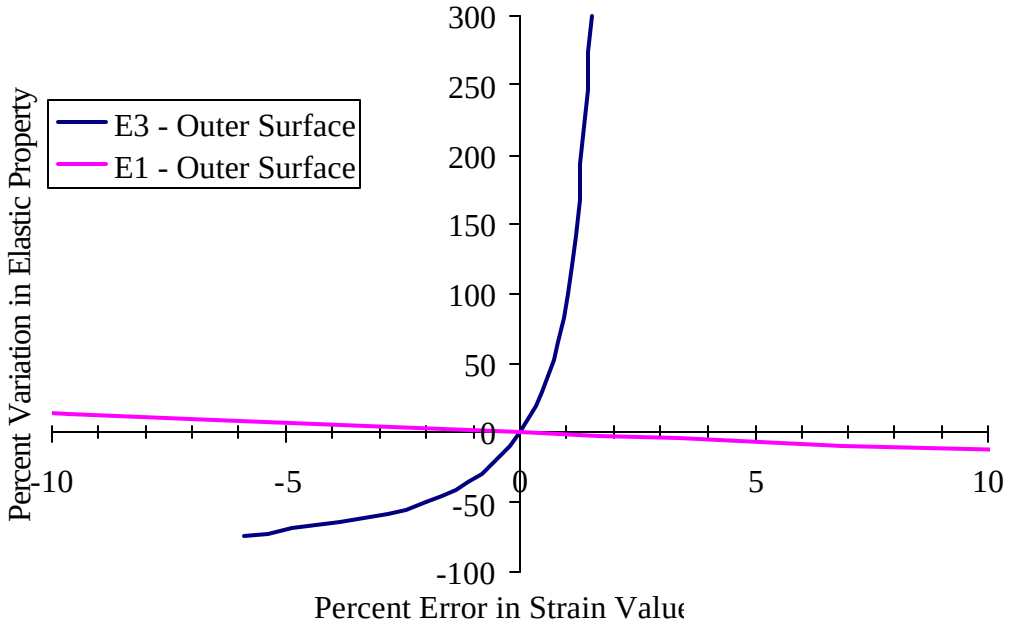


Figure 4-12. Variation in the properties given error in the strain values

These graphs give a range of values that fall within a small percentage variation of the measured strain response. This evaluates the variation of only one value at a time with in-plane strain measurements, but it gives insight into appropriate test selection. By choosing tests that result in responses for the out-of-plane properties similar to the in-plane values in Figure 4-13, a

test matrix can be created that will yield better results for all the elastic constants. There are no significant responses for the out-of-plane values using the test matrix and strain values described in the experimental procedures. By changing the test matrix to include internal and external pressure with measured values for radial strain, the E_3 contribution is significant, as can be seen in Figure 4-14 (in this, the plots for the hoop strain from Figure 4-11 are included to give a reference), and would allow for improved calculation of E_3 and the other out-of-plane values.

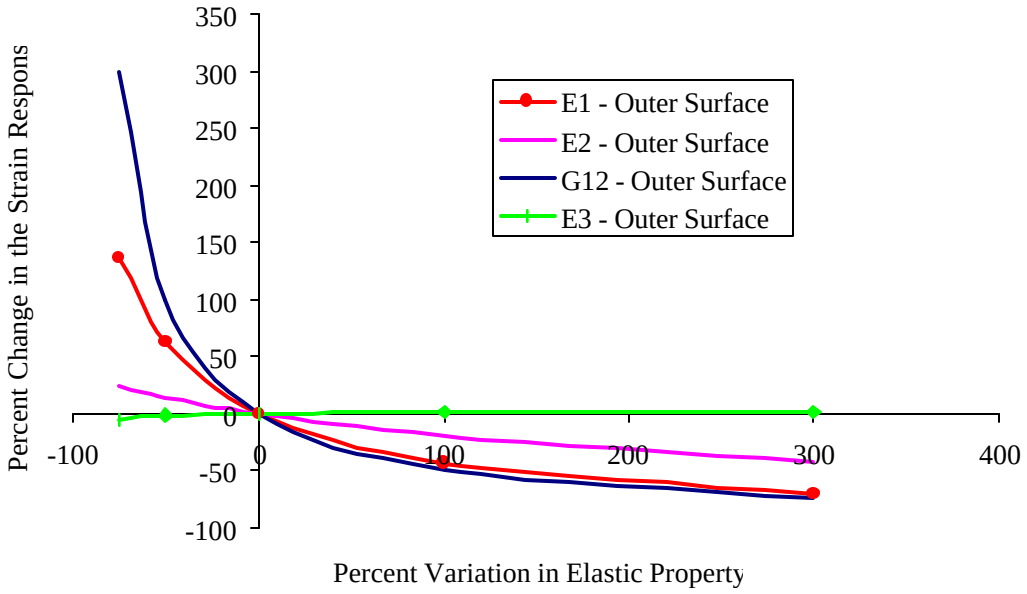


Figure 4-13. Variation to the strain responses with changes to the elastic properties. All values are for the hoop strain response to load condition #5, except the shear modulus line is for the shear strain response.

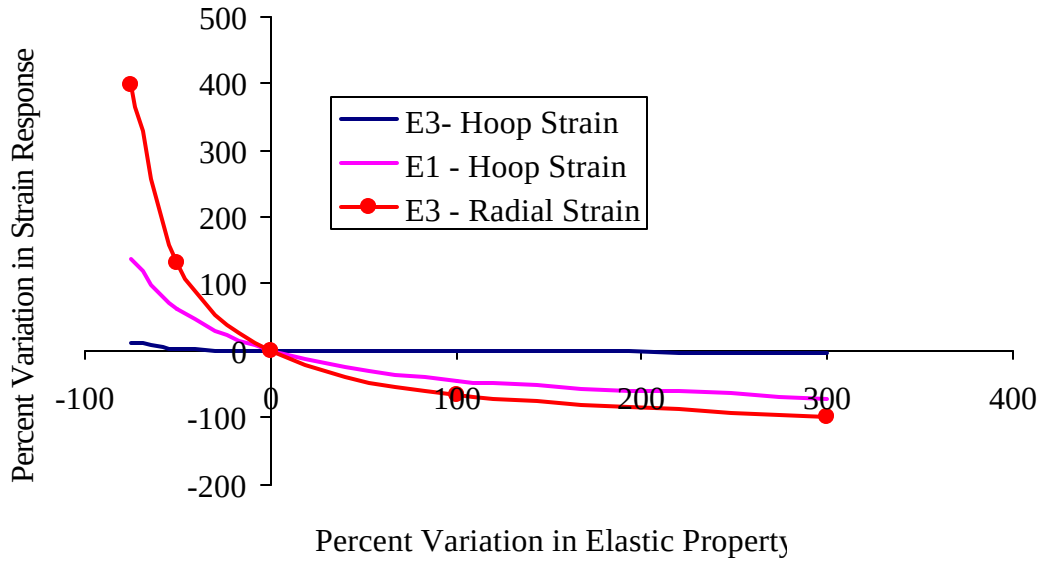


Figure 4-14. Variation in the strain response with changes to the elastic properties for Thick Tube 2. The E3 line with the large response is for a loading condition of combined internal and external pressure.

4.4 Discussion of the Analytical Results

Four different approaches to optimizing the elastic properties for a set of experimentally observed strain values have been illustrated. The Nelder-Mead routines used for this research were easily implemented, but they were not as accurate in finding the minimum of the error function. The Newtonian methods found the minimum with good accuracy when given data sets free from error. Both methods were usually able to find solutions in less than five minutes, but there were exceptions where calculations ran substantially longer. The computation time for the LM-C_{ij} program was shorter than that of the LM-E_i program, since the LM-C_{ij} program was more efficient and took fewer steps to find a minimum. The analyses performed on many of the simulations and experimental were performed using the LM-E_i code for reproducibility when the number of active parameters was reduced.

The presence of a second solution that accurately describes the strain data for the Tube 2 geometry presents an undesirable situation. Without extra knowledge or other criteria, no means is available to distinguish which solution contains the true values. This is possible with other

tube structures, but it is believed that this ply lay-up should be avoided. The symmetry of the lay-up is identical to the loading conditions, so that for each condition a pure loading condition is generated (axial tension – all 0º plies are in perfect longitudinal tension and the 90º plies are in transverse tension – for torsion tests all plies are in the same state of stress – internal pressure is the same as the axial tension just rotated by 90º). While the symmetry of the ±45º presents the same condition, the loading conditions are not simple as that for of [0/90] lay-up. The change in the stress state from the [0/90] to the [±45] lay-up appears to be sufficient to remove the second solution. When the values of the second solution for Tube 2 are entered as the start values for Tube 1 the program still calculates the best estimates to be the correct values. It is unknown at this point if there are other solutions within acceptable property ranges for the geometries used for this work, but it is clear that the [0/90] structure is not ideal for sample geometries for this procedure.

Application of the optimization methods to the experimental results illustrates the need for better experimental accuracy and a larger test matrix. The out-of-plane estimates were well outside acceptable ranges for most of the experimental results, and most of the values varied with different start values. This variation precludes declaring the solutions for all of the elastic constants as the true values. The small data sets for the McDermott filter and ORNL tubes are not the best for calculating the values. Larger data sets should be used for this method, since with more data points the sensitivity of the optimization method to random error is decreased. Data from the inner surface should also be included, again to increase the amount of data for the analysis and require the solution to meet the strain response at both surfaces. The results for the Honeywell material and the control sample are closer to acceptable data values giving some credence to the larger data sets work better (and that the changes to the experimental procedures were improvements). The in-plane results for the steel sample are within a few percent of the accepted values, but the out-of-plane values are clearly wrong. The results for the Honeywell sample exhibit two solutions, as was seen with the Tube 2 example. The in-plane properties exhibit small variation in the different solutions, giving the impression that they may be close to the accurate values.

The ability for the user to change the number of active parameters is attractive, but has its drawbacks. The good quality of being able to set the out-of-plane response to constant values and allowing the minimum to be found using only the in-plane constants is offset by the large

variations this can cause in the results. As with the control sample, the elastic constants for the material were calculated with good accuracy when the out-of-plane values were removed from the program. This was not the case for the Honeywell material. The change by reducing the active parameters caused large changes in the elastic constants. For the control sample, the out-of-plane constants were chosen with good accuracy by the isotropy of the sample, keeping them consistent with the expected results. The values for the Honeywell material, these values were not known, so if the values are inconsistent with the true properties then the data could cause significant variation in the in-plane values.

Investigations into the effects of error on the different value estimates illustrated that the in-plane values are more tolerant of error in the strain data than the out-of-plane values. Inspection of the strain variations for the test matrix used for this work proved that the level of accuracy needed to accurately describe the out-of-plane values is beyond the level of experimental procedure. Different test methods are needed, and the capability to measure the radial strain response is necessary to accurately determine all the elastic constants. The radial strain needs to be recorded, since in each of the tests to date, the out-of-plane values are indirectly measured by observing the in-plane strain response. This indirect measurement requires a high level of accuracy to calculate the out-of-plane values. A test of combined internal and external pressure generates a radial stress condition that would allow for the direct observation of the out-of-plane values.

Up to this point, this discussion has focused on creating a good set of data to minimize the sensitivity to the experimental error, but little has been done to document whether an apparently good solution is "good". This is presented since the "bad" solutions are easily identifiable since the values are outside acceptable values, but what of the solutions that are not so outrageously incorrect. All of the constants will be altered by error in the data, but with a proper test matrix, including the radial strain meaurement, the variations to the engineering values will be of the same magnitude as the experimental error (<10%) instead of the large variations seen in this work. As with the materials used in this research, in many cases it may not be possible to check the values against published values (the material has truly unknown properties), so methods to verify the results are needed. There are a few methods to check for a good solution which can be described here and a few that will have to be developed in later work. The easiest check of the data is to predict the strain response to a new load condition (one that was not entered into the

analysis). If the prediction is accurate, it is likely that the estimated elastic parameters are good. Another means of checking the solution is to inspect the stress and strain profiles predicted for the different loading conditions. Non-typical responses would give indication to a poor solution. Both of these methods are qualitative in nature, in that they do not give a numerical measurement of the quality of the fit, but merely provide an illustration of good versus bad values. Further statistical analyses are needed to generate standard deviations or confidence intervals on the estimates.

4.5 Application to Failure Envelopes

A useful aspect of the models used in this approach are the multi-axial stress states that are present during testing, and the ability to control and model them through the thickness of the materials. The presents a useful means to conduct failure analysis of the materials, with controlled application of multi-axial stress states.

Unfortunately, of all the materials tested for this body of work, only the McDermott candle filters failed in sufficient number to perform this analysis. The other materials did not fail in all test procedures; therefore not generating failure loads in different loading conditions. This is problematic since the material properties for the McDermott materials have only been calculated by the inversion technique developed here, and the out-of-plane values are well outside acceptable values. This does not defeat the effectiveness of this research, but merely shows that without more data to reduce error in the values or published results to compare against, the error is unknown and believed to be significant. This section of the work is performed more for illustrative purposes than to determine the failure properties of the material. This example is merely to illustrate that with a complete test matrix to characterize the elastic properties, and a sufficient number of samples to fail under a wide variety of combined load conditions, the failure envelope could be calculated.

The failure events observed in the McDermott materials can be generally cast into two different forms: shear band formation and large single crack formation. The large crack formation was observed in samples that failed due to crack initiation at the loading pins, and is not believed to be indicative of the materials strength. The shear band formation was visible in all the internal pressure failures and tensile failures, and it will be used as the failure mechanism.

As was illustrated earlier, the fiber tows do not fail, but it is the matrix material between fiber tows that is responsible for the failure. For this illustration, a Tsai-Hill criterion will be used to describe the failure properties of the McDermott Technologies filter material. The failure will be assumed to be near the outer surface since the outer surface contained more of the bond agent, and should be the strongest area of the material. Also, this allows for simplification to an in-plane failure criterion, since the stress in the 3-direction is zero at the outer surface. The Tsai-Hill criterion for in-plane failures is:

$$\frac{s_1^2}{X^2} - \frac{s_1 s_2}{X^2} + \frac{s_2^2}{Y^2} + \frac{t_{12}^2}{S^2} = 1 \tag{4.9}$$

where X, Y and S are the axial, transverse and shear strength, respectively [47, pages 314-316, 52, pages 76-80]. For this simple model and with limited data, the values of X and Y will not be changed for different tensile and compressive strengths, or $X_T=X_C=X$ and $Y_T=Y_C=Y$. The values of the failure stresses are low and the tensile strength of the fiber tows should be large, so the criterion can be simplified to:

$$\frac{s_2^2}{Y^2} + \frac{t_{12}^2}{S^2} = 1 \tag{4.10}$$

for matrix controlled failure. The ply-level stresses at failure are calculated by entering the values for the elastic constants found using the analysis (listed in Table 4-XLVI) and the failure loading conditions into the Forward solution. The stress values in Table 4-XLVII are for the stress value at the outer surface, since it is assumed that is the location where failure occurs.

Table 4-XLVI. The elastic properties of the McDermott filter material

Set 3	
Properties - GPa	
E_1	35.0
E_2	10.6
E_3	0.42
G_{12}	1.14
ν_{12}	0.70
ν_{13}	0.01
ν_{23}	0.01

Table 4-XLVII. The experimental failure loads and resulting stress states at the outer surface of the McDermott filter

Applied Load				Stress Generated at Outer Surface			
P_o	P_i (MPa)	F_x (kN)	T_x (N-m)	$ó_1$ (MPa)	$ó_2$ (MPa)	$ô_{12}$ (MPa)	$ó_3$ (MPa)
0.00	0.00	2.56	0.00	1.04	2.02	2.66	-1.5E-17
0.00	0.00	3.50	0.00	1.42	2.76	3.64	-2E-16
0.00	1.05	0.00	0.00	2.35	-0.54	-4.34	5.85E-17
0.00	1.14	0.00	0.00	2.54	-0.58	-4.68	6.87E-17
0.00	1.13	0.00	0.00	2.51	-0.57	-4.63	2.66E-16
0.00	0.00	3.48	-44.45	3.88	3.26	3.35	3.36E-16

Enough data exists to apply the ply-level stresses and the Tsai-Hill criterion to calculate the values for Y and S. A nonlinear regression routine in the MathCAD software was used to find the best parameters to fit the model. The results are in Table 4-XLVIII are for the solution to both Equation (4.9) and (4.10). The plot of the surface of the failure envelope with the experimental stresses is found in Figure 4-15.

Table 4-XLVIII. Tsai-Hill strength values for the McDermott filter

	2 Variables (Y and S)	3 Variables (X, Y, and S)
X (MPa)	--	1.04E+05
Y (MPa)	4.27	4.25
S (MPa)	4.55	4.56

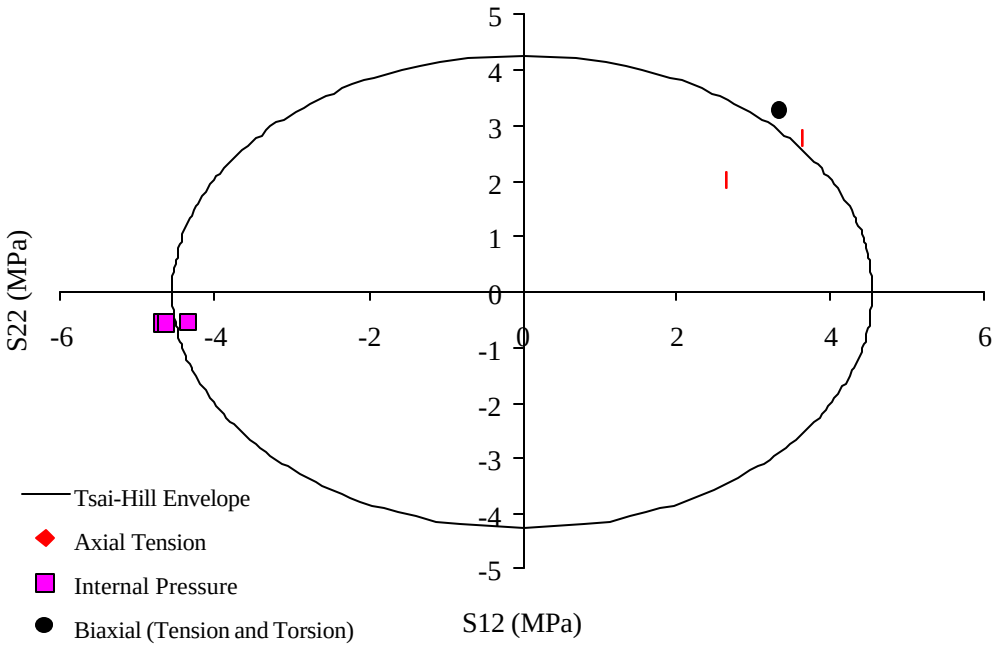


Figure 4-15. Tsai-Hill failure envelope for the McDermott filter

This demonstration of calculating a failure envelope was made with large assumptions and is not intended to represent the failure performance of the McDermott filter material. It has been included since the method is a direct extension of this research and provided a powerful tool for characterizing not only the elastic properties but also the strength of the material under multiaxial stress conditions.

5 Summary and Conclusions

This body of work has addressed each of the three research objectives listed in the Introduction. Experimental procedures have been developed, the regression analysis has been improved and sensitivity of the method has documented, and a framework for developing failure analyses in composite tubes has been presented. In this section, a brief summary of each of the major sections is given, and the final conclusions of this thesis are presented. A few ideas are presented in the Future Work section.

5.1 Summary of Experimental Results

Experimental procedures were developed for characterizing the elastic response and the strength of composite tubes. Different loading fixtures were developed for applying different axisymmetric load conditions to the axial samples. The fixture used for the Honeywell sample generated the simplest stress state since it was not exposed to the large compressive stresses by the MTS collets as was the case for the ORNL materials, or the effects of the stress concentrations found in the MTI filters. The failure of the fixture at loads below the ultimate tensile strength of the sample can be overcome by increasing the depth of the groove, thereby increasing the epoxy bond area and strength. The internal pressure test procedure developed here is sufficient for testing composite materials for internal pressure results. The method did not generate sufficient pressure to fail the SiC ceramics, but it is believed that improved fit between the compression platen and the inner surface of the sample will increase the pressure limit to that exceeding the strength of the samples.

5.2 Summary of Nonlinear Regression Analysis

Several improvements to the original programs developed by George have been implemented. The new analyses can process more data from each test and are more stable in operation. The Newtonian and Nelder-Mead Simplex methods reduce the error

in a set of initial guesses, and output a set of estimates of the material properties that best describe the data. The Newtonian methods are more accurate for this application than the Nelder-Mead Simplex methods. The Levenberg-Marquardt method of optimizing the elastic constants has been applied to the experimental results. The best-fit values are varied, and many of the out-of-plane responses are returned outside acceptable ranges. This behavior is attributed to limited data set sizes, which increase the sensitivity to experimental error and biases in the data. By examining the effects of the variation of individual elastic constants on the strain response, illustrations of the change to the different strain responses for variations to the in-plane and out-of-plane constants have been developed. The plots illustrate that the procedures developed in the experimental section require an unattainable level of accuracy to determine the out-of-plane constants. The results indicate that recording the radial strain response is necessary for this procedure to work for all the values.

5.3 Conclusions

In the attempt to address the three objectives stated for this dissertation, a series of conclusions were found for the experimental and analytical methodologies presented:

1. The elastomer compression method for internal pressure testing is successful. The pressure calculated matches that observed in the control samples, and is supported by the experimental results for the composite tubes. It is not ideal for use for generating data for the inversion techniques since it does not allow access to the inner surface and a small axial compressive stress is present (which would introduce an unpredicted change in the strain data – essentially error).
2. A test matrix must include axial tension, torsion, and internal pressure, at a minimum, to calculate values for the elastic constants, but using only these values requires extreme accuracy in the load and strain measurements.
3. The [0/90] architecture should be avoided for use in the experimental samples. A second “false” solution exists that perfectly describes the strain response.

4. The Newtonian regression methods were more accurate than the Nelder-Mead Simplex method for calculating the material property estimates.
5. Larger data sets are more tolerant of random error in the experimental results.
6. Out-of-plane property prediction will improve with the direct measurement of out-of-plane responses. The methods used in this work only estimate the out-of-plane values from the in-plane responses. This means including radial strain values, and ideally, a combined state of internal and external pressure.
7. A recommended test matrix includes internal and external pressure, axial tension, compression, and axial torsion with the ability to apply those singly and in combination while recording axial, hoop, radial, and shear strain from both surfaces. This would give an ideal test data set for characterizing all seven of the elastic constants.

5.4 Future Work

There have been several areas in this research where future work is needed to develop these methods to the point of being a useful tool to the engineer. The first is the testing of the proposed test matrix where strain measurements (axial, hoop, radial, and shear) can be made on both surfaces for a tube subjected to axial tension, torsion, and internal and external pressure. Unfortunately, executing that test matrix was not possible during this research. The thickness change or radial strain measurement procedures would have to be developed and applied to the samples under the test matrix conditions. The other major development to this work requires developing methods for quantifying error in the solution values for an unknown material. This work has shown methods believed to reduce the sensitivity of the analysis so that the results have a similar level of error as the data, but it does not relate any information on the scatter in the values. Methods for calculating the standard deviation for the estimates given error levels in the data would be beneficial.

A number of logical extensions of this work could be easily pursued with small changes to the analyses and experimental procedures. A test matrix with all the different

axisymmetric load conditions and measurement of the four strain components can be used to optimize not only the elastic constants but also the thermal and hygroscopic expansion constants. The thermal and hygroscopic expansion conditions fall within the axisymmetric conditions required of the elasticity model and analysis as long as they are performed so that they are uniform for the entire sample. A simple change to the equations in the computer program would allow for the system to optimize on the seven elastic, three thermal, and three hygroscopic properties.

6 References

1.R. H. Carter, X. Huang, and K. L. Reifsnider, "Experimental Study of a Ceramic Hot Gas Candle Filter,"*Ceramic Engineering and Science Proceedings*, **20** [3], pp. 153-160, 1999.

2.X. Huang, R. H. Carter, and K. L. Reifsnider, "Experimental Study and Modeling of a Novel Ceramic Composite Hot Gas Candle Filter Material," *Ceramic Engineering and Science Proceedings*, **19** [3], 1998.

3.X. Huang, R. H. Carter, and K. L. Reifsnider, "Modeling Ceramic Composite Hot Gas Candle Filter Material Using Energy Method," *Ceramic Engineering and Science Proceedings*, **20** [3], pp. 145-152, 1999.

4.R. H. Carter, X. Huang, and K. L. Reifsnider, "Effects of Thermal Shock on the Mechanical Properties of a Hot Gas Candle Filter, "*Ceramic Engineering and Science Proceedings*, **21** [3], pp. 57-64, 2000.

5.R. H. Carter, S. Case, K. Reifsnider, K. Probst, and T. Besmann, "Mechanical Properties of Alumina fiber/SiC Matrix Composite Tubes," *Ceramic Engineering and Science Proceedings*, **21** [3], pp. 575-579, 2000.

6. T. E. Lippert, M. A. Alvin, G. J. Bruck, J. Isaksson, R. A. Dennis, and R. A. Brown, , "Testing of the Westinghouse Hot Gas Filter at Ahlstrom Pyropower Corporation," Proceedings of the International Conference on Fluidized Bed Combustion, 1991, ASME, New York, NY, pp 251-261.

7.R. R. Judkins, D. P. Stinton, R. G. Smith, E. M. Fischer, J. H. Eaton, B. L. Weaver, J. L. Kahnke, D. J. Pysher, "Development of Ceramic Composite Hot-Gas Filters," ASME publication 95-GT-305, ASME, New York, NY, 1995.

8.M. J. Mudd and M. W. Durner, "American Electric Power's PFBC Hot Gas Clean Up Test Program," Proceedings of the International Conference on Fluidized Bed Combustion, 1991, ASME, New York, NY, pp 953-957.

9.T. E. Lippert, "Field Test Results Obtained on the Integrated Operation of Ceramic Barrier Filters in PFBC Test Facilities," Proceedings of the 1993 International Joint Power Generation Conference, ASME, vol. 16, pp. 39-55, 1993.

10. Wagner, R. A, "Tough Hot Gas Filter Developed," Power Engineering, vol. 102, No. 8, pp. 46-50, 1998.

11. R. A. Wagner, "Ceramic Hot Gas Filter Development," Presented at the Advanced Coal-Based Power and Environmental Systems Conference, Morgantown, WV, July 21-23, 1997.

12. D. H. Smith and G. Ahmadi, "Problems and Progress in Hot-Gas Filtration for Pressurized Fluidized Bed Combustion (PFBC) and Integrated Gasification Combined Cycle (IGCC)." Aerosol Science and Technology, vol. 29, pp. 163-169, 1998.

13. T. E. Lippert, M. A. Alvin, D. M. Bachovchin, G. B. Haldipur, R. A. Newby, and E. E. Smeltzer, "Development and Commercialization of Hot Gas Filtration Systems," Proceedings of the International Conference on Fluidized Bed Combustion, 1991, ASME, New York, NY, pp 1073-1079.

14. M. Durst, M. Muller, M. R. Schnell, and A. R. Wagner, "Performance of Rigid Ceramic Filter Elements in an 8,000 Hour Durability Test at High Temperatures," Proceedings of the International Conference on Fluidized Bed Combustion, 1991, ASME, New York, NY, pp 1081-1086.

15. T. E. Lippert, R. A. Newby, M. A. Alvin, D. M. Bachovchin, G. J. Bruck, and E. E. Smeltzer, "Development of Hot Gas Cleaning Systems for Advanced, Coal-Based Gas Turbine Cycles." Transactions of the ASME, vol. 115, pp. 658-664, 1993.

16. J. E. Oakey, T. Lowe, R. Morrell, R. A. Brown, and J. Stringer, "Experiences with the EPRI Hot Gas Filter in the Grimethorpe Topping Cycle Project," Proceedings of the International Conference on Fluidized Bed Combustion, 1995, ASME, New York, NY, pp 1199-1209.

17. M. J. Mudd and J. D. Hoffman, "Operating Experience from the Tidd PFBC Hot Gas Clean Up Program," Proceedings of the International Conference on Fluidized Bed Combustion, 1995, ASME, New York, NY, pp 263-277.

18. J. Stringer, A. J. Leitch, R. K. and Clark, "The EPRI Hot Gas Filter Pilot Plant: What Worked, What Broke and Where Do We Go Now?" Proceedings of the International Conference on Fluidized Bed Combustion, 1991, ASME, New York, NY, pp 971-984.

19. P. S. Weitzel, "Study of the Application of an Advanced Ceramic Tube Filter in a Commercial PFBC Plant," Proceedings of the International Conference on Fluidized Bed Combustion, 1991, ASME, New York, NY, pp 33-40.

20. D. M. Hudson, A. N. Twigg, R. K. Clark, P. Holbrow, and A. J. Leitch, "Durability of Ceramic Filtration Systems for Particulate Removal at High Temperatures," Proceedings of the International Conference on Fluidized Bed Combustion, 1991, ASME, New York, NY, pp 295-301.

21. E. George, Master's Thesis: " A Method for the Ply-Level Elastic Characterization of Composite Materials Using Thick Tubular Angle-Ply Specimens," Virginia Tech, Blacksburg, VA, 1993.

22. T. Besmann, B. W. Sheldon, R. A Lowden, and D. P. Stinton, "Vapor-Phase Fabrication and Properties of Continous-Filament Ceramic Composites," *Science*, 253, pp 1104-1109, 1991.

23. K. J. Probst, T. M. Besmann, D. P. Stinton, T. J. Anderson, and T. L. Starr, "Recent Advances in Forced-Flow, Thermal Gradient CVI for Refractory Composites," *Surface Coatings and Technology*, 120-121, pp. 250-258, 1999.

24. K. J. Probst, T. M. Besmann, J. C. McLaughlin, T. J. Anderson, and T. L. Starr, "Development of a Scaled-Up Chemical Vapor Infiltration System for Tubular Geometries," *Materials at High Temperatures*, 16[4], pp. 201-205, 1999.

25. B. L. Weaver, R. A. Lowden, J. C. Laughlin, D. P. Stinton, T. M. Besmann and O. J. Schwarz, "Nextel/SiC Composites Fabricated Using Forced Chemical Vapor Infiltration," *Ceramic Engineering and Science Proceedings*, 14[9-10], pp. 1008-1015, 1993.

26. K. R. Vaidyanathan, J. Sankar, A. D. Kelkar and B. Weaver," Mechanical Properties of Nextel 312 Fiber-Reinforced SiC Matrix Composites in Tension," *Ceramic Engineering and Science Proceedings*, 15[4], pp. 251-261, 1994.

27. K. R. Vaidyanathan, J. Sankar, A. D. Kelkar and J. Narayan, "Investigation of Mechanical Properties of Chemically Vapor Infiltrated (CVI) Ceramic Matrix Composites," *Ceramic Engineering and Science Proceedings*, 15[4], pp. 281-291, 1994.

28. K. R. Vaidyanathan, J. Sankar, A. D. Kelkar, D. P. Stinton and M. H. Headinger, "Investigation of Mechanical Properties of Chemically Vapor Infiltrated Ceramic Matrix Composites Under Pure Tension, " *Ceramic Engineering and Science Proceedings*, 14[9-10], pp. 1016-1027, 1993.

29. K. Liao, E. George, and K. L. Reifsnider, "Characterization of Ceramic Matrix Composite Tubes Under Uniaxial/Biaxial Monotonic and Cyclic Loading," Proceedings of the 1995 Symposium on Multiaxial Fatigue and Deformation Testing Techniques, ASM STP 1280, pp. 224-240, 1997.

30. K. Liao, M. Elahi, and K. L. Reifsnider, "Uniaxial and Multiaxial Fatigue of Ceramic Composite Tubes," *Ceramic Engineering and Science Proceedings*, 16[4], pp. 559-569, 1995.

31. K. L. Reifsnider, W. W. Stinchcomb, L. Olesksuk, and S. S. Lee, "Investigation of Properties and Performance of Ceramic Composite Components," Proceedings of the Eigth Annual Conference on Fossil Energy Materials, May 10-12, 1994, Oak Ridge, TN, pp. 19-28, 1994.

32. Personal communications with Phil Patterson at Honeywell

33. Data sheet from OnlineMetals.com

34. J. P. Singh, M. Sutaria, and W. Bielke, "Thermal Shock Behavior of Advanced Ceramic/Composite Hot-Gas Filters" *Ceramic Engineering and Science Proceedings,* **18**[3], pp.719-727, 1997.

35. J. P. Singh, S. Majumdar,M. Sutaria, and W. Bielke, "Thermal Shock Behavior of Advanced Ceramic/Composite Hot-Gas Filters," Argonne National Laboratory Report, ANL/FE-97/01, March, 1997.

36. M. A. Alvin, T. E. Lippert, E. E. Smeltzer, and G. J. Bruck, "Advanced Hot Gas Filter Performance and Characterization," Presented at the Advanced Coal-Based Power and Environmental Systems Conference, Morgantown, WV, July 21-23, 1998.

37. C. Rousseau, Master's Thesis: " Stresses and Deformations in Angle-Ply Composite Tubes," Virginia Tech, Blacksburg, VA, 1987.

38. C. Rousseau, M. Hyer, and S. Tompkins, "Stresses and Deformations in Angle-Ply Composite Tubes," CCMS-87-04, Virginia Tech, Blacksburg, VA, 1987.

39. Personal communications with Dr. Scott Case, Assistant Professor, Department of Engineering Science and Mechanics, Virginia Tech.

40. Personal communications with Dr. Walter H. Carter, Professor and Chairman, Department of Biostatistics, Virginia Commonwealth University.

41. D. M. Olsson and L. S. Nelson, "The Nelder-Mead Simplex Procedure for Function Minimization," *Technometrics,* Vol. 17, No. 1, pp. 45-51, 1975.

42. W. H. Press, S. A. Teulkolsky, W. T. Vetterling, and B. P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, Second Edition, Cambridge University Press, New York, 1992.

43. S. P. Timoshenko and J. N. Goodier, Theory of Elasticity: Third Edition, McGraw-Hill Book Co., New York, 1970.

44. L. Chuck and G. Graves, "Hoop Tensile Strength and Fracture Behavior of Continuous Fiber Ceramic Composite (CFCC) Tubes from Ambient to Elevated Temperatures," *Journal of Composites Technology & Research*, Vol. 19, No. 3, pp. 184-190, 1997.

45. N. E. Dowling, Mechanical Behavior of Materials, Prentice-Hall, Inc., Englewood Cliffs, NJ, pp. 162-164, 1993.

46. Personal Communications with Dr. E. Lara-Curzio, Oak Ridge National Laboratory, January, 2001.

47. M. Ward and D. W. Hadley, An Introduction to the Mechanical Properties of Solid Polymers, John Wiley and Sons, New York, 1993.

48. D. M. Bates and D. G. Watts, Nonlinear Regression Analysis and Its Application, John Wiley and Sons, New York, 1988.

49. G. A. F. Seber and C. J. Wild, Nonlinear Regression, John Wiley and Sons, New York, 1989.

50. C. T. Herakovich, Mechanics of Fibrous Composites, John Wiley and Sons, New York, 1998.

51. X. Huang, R. Carter, K. Reifsnider, " Mechanical Properties of a Hot Gas Candle Filter Material, " Center for Composite Materials and Structures Report #CCMS-99-02, Virginia Tech, Blacksburg, VA, 1998.

52. R. M. Jones, Mechanics of Composite Materials, Taylor and Francis, 1975.

7 Appendix A – Elasticity Solutions

The work by Hyer and Rousseau derived the general solution for a composite tube consisting of orthotropic material layers. The displacement equations are the solution for a composite tube, consisting of orthotropic layers oriented at an angle to the axis of the tube, subjected to axisymmetric loading conditions. The general solution is given as:

$$
\begin{aligned}
&u(x) = e^o x \\
&v(x,r) = g^o xr \\
&w(r) = A_1 r^l + A_2 r^{-l} + \Gamma e^o r + \Omega g^o r^2 + \Psi r \Delta T \\
&l = \sqrt{\frac{\bar{C}_{22}}{\bar{C}_{33}}} \\
&\Gamma = \left(\frac{\bar{C}_{12} - \bar{C}_{13}}{\bar{C}_{33} - \bar{C}_{22}} \right) \\
&\Omega = \left(\frac{\bar{C}_{26} - 2\bar{C}_{36}}{4\bar{C}_{33} - \bar{C}_{22}} \right) \\
&\Psi = \left(\frac{\sum (\bar{C}_{i2} - \bar{C}_{i3}) a_i}{\bar{C}_{33} - \bar{C}_{22}} \right)
\end{aligned}
\tag{7.1}
$$

By examination of the equations for the Γ, Ω, and Ψ terms, two conditions create a condition where the denominator goes to zero, namely when $\bar{C}_{22} = \bar{C}_{33}$ (λ = 1) and $\bar{C}_{22} = 4\bar{C}_{33}$ (λ = 2). For both of these conditions, a degenerate solution can be found by starting with the Equilibrium condition equations.

$$
\begin{aligned}
&\frac{\partial s_x}{\partial x} + \frac{1}{r}\frac{\partial t_{xq}}{\partial q} + \frac{\partial t_{xr}}{\partial r} + \frac{1}{r} t_{xr} = 0 \\
&\frac{\partial t_{xq}}{\partial x} + \frac{1}{r}\frac{s_q}{\partial q} + \frac{t_{rq}}{\partial r} + \frac{2}{r} t_{rq} = 0 \\
&\frac{\partial t_{xr}}{\partial x} + \frac{1}{r}\frac{\partial t_q}{\partial q} + \frac{\partial s_r}{\partial r} + \frac{s_r - s_q}{r} = 0
\end{aligned}
\tag{7.2}
$$

The constraints imposed by the axisymmetric conditions and boundary equations, the Equilibrium equations simplify to:

$$\begin{aligned} \frac{\partial t_{xr}}{\partial r} + \frac{1}{r} t_{xr} &= 0 \\ \frac{t_{rq}}{\partial r} + \frac{2}{r} t_{rq} &= 0 \\ \frac{\partial s_r}{\partial r} + \frac{s_r - s_q}{r} &= 0 \end{aligned} \tag{7.3}$$

The strain displacement relations and constitutive equations are needed to translate the displacements to strains and stresses that can be used to solve the equations.

$$\begin{aligned} e_x &= \frac{\partial u}{\partial x} & g_{rq} &= \frac{1}{r}\left(\frac{\partial w}{\partial q} - v + r\frac{\partial v}{\partial r}\right) \\ e_q &= \frac{w}{r} & g_{xr} &= \frac{\partial u}{\partial r} + \frac{\partial w}{\partial x} \\ e_r &= \frac{\partial w}{\partial r} & g_{xq} &= \frac{\partial v}{\partial x} + \frac{1}{r}\frac{\partial u}{\partial q} \end{aligned} \tag{7.4}$$

After several steps of work in Hyer and Rousseau, it was found that:

$$\begin{aligned} u &= e_x^o x \\ v &= g^o xr \end{aligned} \tag{7.5}$$

The degenerate conditions arise from the expression used to find w(r):

$$\frac{\partial s_r}{\partial r} + \frac{s_r - s_q}{r} = 0 \tag{7.6}$$

The values for σ_r and σ_θ are:

$$\begin{aligned} s_r &= \bar{C}_{13}\varepsilon_x + \bar{C}_{23}\varepsilon_q + \bar{C}_{33}\varepsilon_r + \bar{C}_{36}\gamma_{xq} \\ s_q &= \bar{C}_{12}\varepsilon_x + \bar{C}_{22}\varepsilon_q + \bar{C}_{23}\varepsilon_r + \bar{C}_{26}\gamma_{xq} \end{aligned} \tag{7.7}$$

which expand to:

$$\begin{aligned} s_r &= \bar{C}_{13}\varepsilon_x^o + \bar{C}_{23}\frac{w}{r} + \bar{C}_{33}\frac{dw}{dr} + \bar{C}_{36}\gamma^o r \\ s_q &= \bar{C}_{12}\varepsilon_x^o + \bar{C}_{22}\frac{w}{r} + \bar{C}_{23}\frac{dw}{dr} + \bar{C}_{26}\gamma^o r \end{aligned} \tag{7.8}$$

Collecting for w, the expression for the general solution is:

$$\bar{C}_{33}\left(\frac{d^2w}{dr^2} + \frac{1}{r}\frac{dw}{dr}\right) - \bar{C}_{22}\frac{w}{r^2} = \left(\bar{C}_{26} - 2\bar{C}_{36}\right)\gamma^o + \left(\frac{\bar{C}_{12} - \bar{C}_{13}}{r}\right)\varepsilon_x^o \tag{7.9}$$

which is listed in Equation (7.1). In the first of the two degenerate conditions, λ=1, the expression becomes:

$$\bar{C}_{33}\left(\frac{d^2w}{dr^2} + \frac{1}{r}\frac{dw}{dr} - \frac{w}{r^2}\right) = \left(\bar{C}_{26} - 2\bar{C}_{36}\right)\gamma^o + \left(\frac{\bar{C}_{12} - \bar{C}_{13}}{r}\right)\varepsilon_x^o \tag{7.10}$$

which solves to:

$$
\begin{aligned}
w(r) &= A_1 r^{l} + A_2 r^{-l} + \Gamma e^{o} r \ln(r) + \Omega g^{o} r^{2} + \Psi \Delta T r \ln(r) \\
l &= \sqrt{\frac{\bar{C}_{22}}{\bar{C}_{33}}} = 1 \\
\Gamma &= \left(\frac{\bar{C}_{12} - \bar{C}_{13}}{2\bar{C}_{33}} \right) \\
\Omega &= \left(\frac{\bar{C}_{26} - 2\bar{C}_{36}}{4\bar{C}_{33} - \bar{C}_{22}} \right) \\
\Psi &= \left(\frac{\sum (\bar{C}_{i2} - \bar{C}_{i3}) a_i}{2\bar{C}_{33}} \right)
\end{aligned}
\tag{7.11}
$$

The second degenerate condition is:

$$
\bar{C}_{33} \left(\frac{d^2 w}{dr^2} + \frac{1}{r} \frac{dw}{dr} - 4 \frac{w}{r^2} \right) = \left(\bar{C}_{26} - 2\bar{C}_{36} \right) g^{o} + \left(\frac{\bar{C}_{12} - \bar{C}_{13}}{r} \right) e_x^{o} \tag{7.12}
$$

which solves to:

$$
\begin{aligned}
w(r) &= A_1 r^{l} + A_2 r^{-l} - \Gamma e^{o} r + \Omega g^{o} r^{2} (4\ln(r) - 1) + \Psi r \Delta T \\
l &= \sqrt{\frac{\bar{C}_{22}}{\bar{C}_{33}}} = 2 \\
\Gamma &= \left(\frac{\bar{C}_{12} - \bar{C}_{13}}{3\bar{C}_{33}} \right) \\
\Omega &= \left(\frac{\bar{C}_{26} - 2\bar{C}_{36}}{16\bar{C}_{33}} \right) \\
\Psi &= \left(\frac{\sum (\bar{C}_{i2} - \bar{C}_{i3}) a_i}{3\bar{C}_{33}} \right)
\end{aligned}
\tag{7.13}
$$

With these expressions, the different programs used to calculate the stress and strain responses check the value of λ and use the appropriate expressions.

8 Appendix B – Elastic Response for Varied Elastic Properties

After the analysis is run on data sets with error incorporated, certain properties seem less sensitive to error. The in plane values usually came back with small amounts variation in comparison to the out of plane properties. To investigate this and the effect that thickness may play on the calculations, the strain responses for four different tube geometries will be calculated with different variations of elastic properties. In each condition a single elastic constant will be reduced to one-half of its original value. The strain response to the same loading conditions will illustrate the sensitivity to error for that value by illustrating the level of accuracy needed to get within 50% of the correct value. Two of the geometries have been altered to increase the wall thickness of the material. To keep the loads in similar ranges, the loads have been scaled to generate the same stress state in the material, as the thinner samples experienced.

Table 8-I. Elastic Constants for the composite material

Elastic Properties	**GPa (Msi)**
$\mathbf{E_1}$	43.5 (6.31)
$\mathbf{E_2}$	11.5 (1.67)
$\mathbf{E_3}$	11.5 (1.67)
$\mathbf{G_{12}}$	3.45 (0.5)
$\mathbf{n_{12}}$	0.27
$\mathbf{n_{13}}$	0.4
$\mathbf{n_{23}}$	0.4

Table 8-II. Geometry of the four different composite simulations

Tube ID	Inner Radius mm (in)	Outer Radius mm (in)	Number of Plies	Orientation	Ply Thickness mm (in)
Thin 1	25.4 (1.0)	27.9 (1.1)	20	$[45/-45]_{10}$	0.127 (0.005)
Thick 1	25.4 (1.0)	31.75 (1.25)	20	$[45/-45]_{10}$	0.3125 (0.0125)
Thin 2	25.4 (1.0)	27.9 (1.1)	20	$[0/90]_{10}$	0.127 (0.005)
Thick 2	25.4 (1.0)	31.75 (1.25)	20	$[0/90]_{10}$	0.3125 (0.0125)

Table 8-III. Applied loads for each tube – thick values calculated to generate the same stress state as for the thin tubes

	Thin Tube Loading Conditions				Thick Tube Loading Conditions			
	Po	Pi	Fx	Tx	Po	Pi	Fx	Tx
Load Case	psi	psi	lbf	In-lbf	psi	psi	lbf	In-lbf
1	0	5000	0	0	0	13393	0	0
2	0	0	10000	0	0	0	26786	0
3	0	0	0	10000	0	0	0	27331
4	0	0	10000	10000	0	0	26786	27331
5	0	5000	10000	10000	0	13393	26786	27331

8.1 Thin Tube 1

Table 8-IV. Strain Response for the different loading conditions for Thin Tube 1

Standard Response				
Load	Surface	Axial	Hoop	Shear
1	I	-1.92E-02	3.26E-02	-1.87E-05
1	o	-1.92E-02	2.91E-02	-2.06E-05
2	I	9.25E-03	-6.12E-03	1.63E-05
2	o	9.25E-03	-5.70E-03	1.79E-05
3	I	1.63E-05	-5.96E-06	7.60E-03
3	o	1.63E-05	-4.85E-06	8.36E-03
4	I	9.27E-03	-6.12E-03	7.62E-03
4	o	9.27E-03	-5.71E-03	8.38E-03
5	I	-9.95E-03	2.65E-02	7.60E-03
5	o	-9.95E-03	2.34E-02	8.36E-03

Table 8-V. Strain deviation from the Standard Response for E1/2

E1/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-15.65	10.22	-76.25	-3.01E-03	-3.33E-03	-1.43E-05
1	o	-15.65	9.42	-76.25	-3.01E-03	-2.74E-03	-1.57E-05
2	I	9.41	-15.65	-33.42	-8.71E-04	-9.57E-04	5.45E-06
2	o	9.41	-13.74	-33.42	-8.71E-04	-7.84E-04	6.00E-06
3	I	-33.42	-76.25	76.65	5.45E-06	-4.55E-06	-5.83E-03
3	o	-33.42	-77.11	76.65	5.45E-06	-3.74E-06	-6.41E-03
4	I	9.34	-15.71	76.42	-8.65E-04	-9.62E-04	-5.82E-03
4	o	9.34	-13.79	76.42	-8.65E-04	-7.87E-04	-6.40E-03
5	I	-38.94	16.20	76.80	-3.87E-03	-4.29E-03	-5.84E-03
5	o	-38.94	15.10	76.79	-3.87E-03	-3.53E-03	-6.42E-03

Table 8-VI. Strain deviation from the Standard Response for E2/2

E2/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-5.37	3.42	0.13	-1.03E-03	-1.12E-03	2.37E-08
1	o	-5.37	3.31	0.13	-1.03E-03	-9.61E-04	2.60E-08
2	I	3.28	-5.37	32.35	-3.03E-04	-3.29E-04	-5.28E-06
2	o	3.28	-4.85	32.35	-3.03E-04	-2.77E-04	-5.80E-06
3	I	32.35	0.13	6.78	-1.07E-05	4.56E-06	5.31E-03
3	o	32.35	4.09	6.78	-5.28E-06	1.99E-07	-5.67E-04
4	I	3.33	-5.37	6.83	-3.09E-04	-3.29E-04	-5.21E-04
4	o	3.33	-4.85	6.83	-3.09E-04	-2.77E-04	-5.73E-04
5	I	-13.49	5.45	6.85	-1.34E-03	-1.45E-03	-5.21E-04
5	o	-13.49	5.30	6.85	-1.34E-03	-1.24E-03	-5.73E-04

Table 8-VII. Strain deviation from the Standard Response for E3/2

E3/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.00	0.29	10.98	-9.00E-07	-9.37E-05	2.06E-06
1	o	0.00	-0.15	10.98	-9.00E-07	4.31E-05	2.26E-06
2	I	0.00	0.00	-0.05	-1.00E-08	-2.60E-07	7.90E-09
2	o	0.00	0.00	-0.05	-1.00E-08	2.40E-07	8.80E-09
3	I	-0.05	10.98	0.00	7.90E-09	6.55E-07	-3.00E-08
3	o	-0.05	-12.87	0.00	7.90E-09	-6.24E-07	-3.00E-08
4	I	0.00	0.01	0.00	0.00E+00	4.00E-07	-2.00E-08
4	o	0.00	-0.01	0.00	0.00E+00	-3.80E-07	-2.00E-08
5	I	-0.01	0.35	-0.03	-8.20E-07	-9.33E-05	2.04E-06
5	o	-0.01	-0.18	-0.03	-8.20E-07	4.27E-05	2.24E-06

Table 8-VIII. Strain deviation from the Standard Response for G12/2

G12/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	129.81	80.22	197.32	2.49E-02	-2.62E-02	3.70E-05
1	o	129.81	81.88	197.32	2.49E-02	-2.38E-02	4.07E-05
2	I	81.77	129.81	68.75	-7.56E-03	7.94E-03	-1.12E-05
2	o	81.77	126.51	68.75	-7.56E-03	7.22E-03	-1.23E-05
3	I	68.75	197.32	0.00	-1.12E-05	1.18E-05	-2.00E-08
3	o	68.75	220.56	0.00	-1.12E-05	1.07E-05	-2.00E-08
4	I	81.75	129.88	0.15	-7.57E-03	7.95E-03	-1.12E-05
4	o	81.75	126.59	0.15	-7.57E-03	7.23E-03	-1.23E-05
5	I	174.60	68.76	-0.34	1.74E-02	-1.82E-02	2.57E-05
5	o	174.60	70.95	-0.34	1.74E-02	-1.66E-02	2.83E-05

Table 8-IX. Strain deviation from the Standard Response for nu12/2

Nu12/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-1.95	1.19	-9.23	-3.74E-04	-3.88E-04	-1.73E-06
1	o	-1.95	1.25	-9.23	-3.74E-04	-3.64E-04	-1.90E-06
2	I	1.25	-1.95	-2.70	-1.16E-04	-1.19E-04	4.41E-07
2	o	1.25	-1.96	-2.70	-1.16E-04	-1.12E-04	4.85E-07
3	I	-2.70	-9.23	-4.60	4.41E-07	-5.51E-07	3.50E-04
3	o	-2.70	-9.15	-4.60	4.41E-07	-4.44E-07	3.85E-04
4	I	1.24	-1.96	-4.60	-1.15E-04	-1.20E-04	3.50E-04
4	o	1.24	-1.96	-4.60	-1.15E-04	-1.12E-04	3.85E-04
5	I	-4.92	1.92	-4.59	-4.90E-04	-5.08E-04	3.49E-04
5	o	-4.92	2.04	-4.59	-4.90E-04	-4.76E-04	3.83E-04

Table 8-X. . Strain deviation from the Standard Response for nu13/2

Nu13/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.29	-0.38	21.69	5.55E-05	1.25E-04	4.06E-06
1	o	0.29	0.00	21.69	5.55E-05	4.00E-07	4.47E-06
2	I	0.00	0.29	3.32	4.50E-07	1.77E-05	-5.41E-07
2	o	0.00	-0.30	3.32	4.50E-07	-1.71E-05	-5.95E-07
3	I	3.32	21.69	0.00	-5.49E-07	6.39E-07	3.00E-08
3	o	3.32	25.18	0.00	-5.41E-07	1.22E-06	0.00E+00
4	I	0.00	0.31	0.01	-1.00E-07	1.90E-05	-5.40E-07
4	o	0.00	-0.28	0.01	-1.00E-07	-1.59E-05	-5.90E-07
5	I	0.56	-0.54	-0.05	5.55E-05	1.44E-04	3.52E-06
5	o	0.56	0.07	-0.05	5.55E-05	-1.55E-05	3.87E-06

Table 8-XI. Strain deviation from the Standard Response for nu23/2

Nu23/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.34	-0.43	-32.85	6.57E-05	1.40E-04	-6.16E-06
1	o	0.34	-0.01	-32.85	6.57E-05	4.10E-06	-6.77E-06
2	I	-0.01	0.34	3.93	5.30E-07	2.09E-05	-6.41E-07
2	o	-0.01	-0.36	3.93	5.30E-07	-2.03E-05	-7.05E-07
3	I	3.93	-32.85	0.00	-6.41E-07	-1.96E-06	-1.00E-08
3	o	3.93	-44.37	0.00	-6.41E-07	-2.15E-06	-2.00E-08
4	I	0.00	0.31	0.01	-1.10E-07	1.90E-05	-6.50E-07
4	o	0.00	-0.39	0.01	-1.10E-07	-2.25E-05	-7.10E-07
5	I	0.66	-0.60	0.09	6.56E-05	1.59E-04	-6.81E-06
5	o	0.66	0.08	0.09	6.56E-05	-1.84E-05	-7.49E-06

8.2 Thick Tube 1

Table 8-XII. Strain Response for the different loading conditions for Thick Tube 1

Standard Response				
Load	Surface	Axial	Hoop	Shear
1	I	-2.01E-02	3.85E-02	-4.58E-05
1	o	-2.01E-02	2.90E-02	-5.73E-05
2	I	9.24E-03	-6.41E-03	3.51E-05
2	o	9.24E-03	-5.45E-03	4.39E-05
3	I	3.58E-05	-1.49E-05	6.69E-03
3	o	3.58E-05	-9.17E-06	8.36E-03
4	I	9.27E-03	-6.43E-03	6.73E-03
4	o	9.27E-03	-5.46E-03	8.41E-03
5	I	-1.09E-02	3.21E-02	6.68E-03
5	o	-1.09E-02	2.36E-02	8.35E-03

Table 8-XIII. Strain deviation from the Standard Response for E1/2

E1/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-16.82	11.31	-75.74	-3.39E-03	-4.36E-03	-3.47E-05
1	o	-16.82	9.56	-75.74	-3.39E-03	-2.78E-03	-4.34E-05
2	I	9.52	-16.82	-33.32	-8.79E-04	-1.08E-03	1.17E-05
2	o	9.52	-12.38	-33.32	-8.79E-04	-6.75E-04	1.46E-05
3	I	-33.32	-75.74	76.65	1.19E-05	-1.13E-05	-5.13E-03
3	o	-33.32	-77.80	76.65	1.19E-05	-7.13E-06	-6.41E-03
4	I	9.35	-16.96	76.08	-8.67E-04	-1.09E-03	-5.12E-03
4	o	9.35	-12.49	76.08	-8.67E-04	-6.82E-04	-6.40E-03
5	I	-39.17	16.97	77.12	-4.26E-03	-5.45E-03	-5.15E-03
5	o	-39.17	14.66	77.12	-4.26E-03	-3.46E-03	-6.44E-03

Table 8-XIV. Strain deviation from the Standard Response for E2/2

E2/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-5.67	3.51	-1.82	-1.14E-03	-1.35E-03	-8.34E-07
1	o	-5.67	3.46	-1.82	-1.14E-03	-1.01E-03	-1.04E-06
2	I	3.33	-5.67	32.48	-3.07E-04	-3.63E-04	-1.14E-05
2	o	3.33	-4.47	32.48	-3.07E-04	-2.44E-04	-1.42E-05
3	I	32.48	-1.82	6.78	-2.36E-05	1.10E-05	4.67E-03
3	o	32.48	7.66	6.78	-1.16E-05	7.02E-07	-5.67E-04
4	I	3.44	-5.66	6.91	-3.19E-04	-3.63E-04	-4.65E-04
4	o	3.44	-4.45	6.91	-3.19E-04	-2.43E-04	-5.81E-04
5	I	-13.44	5.35	6.97	-1.46E-03	-1.72E-03	-4.66E-04
5	o	-13.44	5.30	6.97	-1.46E-03	-1.25E-03	-5.82E-04

Table 8-XV. Strain deviation from the Standard Response for E3/2

E3/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-0.05	1.48	23.05	-1.10E-05	-5.72E-04	1.06E-05
1	o	-0.05	-0.80	23.05	-1.10E-05	2.33E-04	1.32E-05
2	I	0.00	-0.05	-0.26	-1.10E-07	-3.48E-06	9.14E-08
2	o	0.00	0.05	-0.26	-1.10E-07	2.96E-06	1.14E-07
3	I	-0.26	23.05	0.00	9.33E-08	3.43E-06	-1.30E-07
3	o	-0.26	-33.50	0.00	9.33E-08	-3.07E-06	-1.60E-07
4	I	0.00	0.00	0.00	-2.00E-08	-5.00E-08	-3.00E-08
4	o	0.00	0.00	0.00	-2.00E-08	-1.20E-07	-4.00E-08
5	I	-0.10	1.78	-0.16	-1.09E-05	-5.72E-04	1.05E-05
5	o	-0.10	-0.99	-0.16	-1.09E-05	2.32E-04	1.32E-05

Table 8-XVI. Strain deviation from the Standard Response for G12/2

G12/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	131.21	76.96	183.99	2.64E-02	-2.97E-02	8.43E-05
1	o	131.21	81.77	183.99	2.64E-02	-2.37E-02	1.05E-04
2	I	81.15	131.21	68.15	-7.50E-03	8.41E-03	-2.39E-05
2	o	81.15	123.52	68.15	-7.50E-03	6.74E-03	-2.99E-05
3	I	68.15	183.99	0.00	-2.44E-05	2.74E-05	-8.00E-08
3	o	68.15	239.19	0.00	-2.44E-05	2.19E-05	-1.00E-07
4	I	81.10	131.33	0.36	-7.52E-03	8.44E-03	-2.40E-05
4	o	81.10	123.72	0.36	-7.52E-03	6.76E-03	-3.00E-05
5	I	173.98	66.08	-0.90	1.89E-02	-2.12E-02	6.03E-05
5	o	173.98	72.05	-0.90	1.89E-02	-1.70E-02	7.54E-05

Table 8-XVII. Strain deviation from the Standard Response for nu12/2

Nu12/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-1.93	1.11	-9.24	-3.88E-04	-4.29E-04	-4.24E-06
1	o	-1.93	1.28	-9.24	-3.88E-04	-3.70E-04	-5.29E-06
2	I	1.26	-1.93	-2.68	-1.17E-04	-1.24E-04	9.40E-07
2	o	1.26	-1.96	-2.68	-1.17E-04	-1.07E-04	1.18E-06
3	I	-2.68	-9.24	-4.60	9.59E-07	-1.38E-06	3.08E-04
3	o	-2.68	-9.05	-4.60	9.59E-07	-8.30E-07	3.85E-04
4	I	1.25	-1.95	-4.59	-1.16E-04	-1.25E-04	3.09E-04
4	o	1.25	-1.97	-4.59	-1.16E-04	-1.07E-04	3.86E-04
5	I	-4.64	1.72	-4.56	-5.04E-04	-5.53E-04	3.05E-04
5	o	-4.64	2.03	-4.56	-5.04E-04	-4.78E-04	3.81E-04

Table 8-XVIII. Strain deviation from the Standard Response for nu13/2

Nu13/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.66	-0.91	19.86	1.32E-04	3.49E-04	9.10E-06
1	o	0.66	-0.01	19.86	1.32E-04	2.00E-06	1.14E-05
2	I	-0.03	0.66	3.30	2.44E-06	4.20E-05	-1.16E-06
2	o	-0.03	-0.72	3.30	2.44E-06	-3.91E-05	-1.45E-06
3	I	3.30	19.86	0.00	-1.27E-06	-4.74E-07	1.40E-07
3	o	3.30	28.20	0.00	-1.18E-06	2.59E-06	1.00E-08
4	I	-0.01	0.70	0.02	1.25E-06	4.50E-05	-1.15E-06
4	o	-0.01	-0.67	0.02	1.25E-06	-3.65E-05	-1.43E-06
5	I	1.23	-1.23	-0.12	1.33E-04	3.94E-04	7.95E-06
5	o	1.23	0.15	-0.12	1.33E-04	-3.45E-05	9.95E-06

Table 8-XIX. Strain deviation from the Standard Response for nu23/2

Nu23/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.77	-0.95	-26.95	1.55E-04	3.66E-04	-1.24E-05
1	o	0.77	-0.08	-26.95	1.55E-04	2.22E-05	-1.54E-05
2	I	-0.03	0.77	3.88	2.90E-06	4.94E-05	-1.36E-06
2	o	-0.03	-0.85	3.88	2.90E-06	-4.61E-05	-1.70E-06
3	I	3.88	-26.95	0.00	-1.39E-06	-4.01E-06	-6.00E-08
3	o	3.88	-54.52	0.00	-1.39E-06	-5.00E-06	-7.00E-08
4	I	-0.02	0.71	0.02	1.50E-06	4.54E-05	-1.42E-06
4	o	-0.02	-0.94	0.02	1.50E-06	-5.11E-05	-1.77E-06
5	I	1.44	-1.28	0.21	1.57E-04	4.11E-04	-1.38E-05
5	o	1.44	0.12	0.21	1.57E-04	-2.90E-05	-1.72E-05

8.3 Thin Tube 2

Table 8-XX. Strain Response for the different loading conditions for Thin Tube 2

Standard Response				
Load	Surface	Axial	Hoop	Shear
1	I	-1.15E-03	1.37E-02	-3.69E-18
1	o	-1.15E-03	1.19E-02	-4.06E-18
2	I	3.78E-03	-3.68E-04	1.79E-19
2	o	3.78E-03	-4.79E-04	1.97E-19
3	I	1.79E-19	-1.17E-18	2.74E-02
3	o	1.79E-19	-1.03E-18	3.02E-02
4	I	3.78E-03	-3.68E-04	2.74E-02
4	o	3.78E-03	-4.79E-04	3.02E-02
5	I	2.62E-03	1.33E-02	2.74E-02
5	o	2.62E-03	1.14E-02	3.02E-02

Table 8-XXI. Strain deviation from the Standard Response for E1/2

E1/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	199.01	64.84	-38.38	2.30E-03	-8.89E-03	-1.42E-18
1	o	199.01	65.71	-38.38	2.30E-03	-7.79E-03	-1.56E-18
2	I	65.59	199.01	24.48	-2.48E-03	7.31E-04	-4.38E-20
2	o	65.59	156.79	24.48	-2.48E-03	7.52E-04	-4.82E-20
3	I	24.48	-38.38	0.00	-4.38E-20	-4.51E-19	0.00E+00
3	o	24.48	-38.00	0.00	-4.38E-20	-3.93E-19	0.00E+00
4	I	65.59	199.01	0.00	-2.48E-03	7.31E-04	0.00E+00
4	o	65.59	156.79	0.00	-2.48E-03	7.52E-04	0.00E+00
5	I	6.87	61.15	0.00	-1.80E-04	-8.16E-03	0.00E+00
5	o	6.87	61.87	0.00	-1.80E-04	-7.04E-03	0.00E+00

Table 8-XXII. Strain deviation from the Standard Response for E2/2

E2/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-49.44	11.83	14.89	-5.71E-04	-1.62E-03	5.49E-19
1	o	-49.44	11.97	14.89	-5.71E-04	-1.42E-03	6.04E-19
2	I	11.84	-49.44	-107.62	-4.47E-04	-1.82E-04	1.93E-19
2	o	11.84	-29.83	-107.62	-4.47E-04	-1.43E-04	2.12E-19
3	I	-107.62	14.89	0.00	2.36E-19	6.26E-19	0.00E+00
3	o	-107.62	14.11	0.00	1.93E-19	1.46E-19	0.00E+00
4	I	11.84	-49.44	0.00	-4.47E-04	-1.82E-04	0.00E+00
4	o	11.84	-29.83	0.00	-4.47E-04	-1.43E-04	0.00E+00
5	I	38.80	13.52	0.00	-1.02E-03	-1.80E-03	0.00E+00
5	o	38.80	13.73	0.00	-1.02E-03	-1.56E-03	0.00E+00

Table 8-XXIII. Strain deviation from the Standard Response for E3/2

E3/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.05	0.72	-0.03	5.70E-07	-9.92E-05	-1.14E-21
1	o	0.05	-0.36	-0.03	5.70E-07	4.29E-05	-1.25E-21
2	I	0.00	0.05	0.00	0.00E+00	1.81E-07	3.00E-24
2	o	0.00	-0.03	0.00	0.00E+00	-1.60E-07	4.00E-24
3	I	0.00	-0.03	0.00	3.00E-24	-3.60E-22	0.00E+00
3	o	0.00	0.03	0.00	3.00E-24	3.30E-22	0.00E+00
4	I	0.00	0.05	0.00	0.00E+00	1.81E-07	0.00E+00
4	o	0.00	-0.03	0.00	0.00E+00	-1.60E-07	0.00E+00
5	I	-0.02	0.74	0.00	5.70E-07	-9.91E-05	0.00E+00
5	o	-0.02	-0.38	0.00	5.70E-07	4.27E-05	0.00E+00

Table 8-XXIV. Strain deviation from the Standard Response for G12/2

G12/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.00	0.00	121.90	0.00E+00	0.00E+00	4.50E-18
1	o	0.00	0.00	121.90	0.00E+00	0.00E+00	4.95E-18
2	I	0.00	0.00	237.55	0.00E+00	0.00E+00	-4.25E-19
2	o	0.00	0.00	237.55	0.00E+00	0.00E+00	-4.68E-19
3	I	237.55	121.90	100.00	-4.25E-19	1.43E-18	-2.74E-02
3	o	237.55	122.62	100.00	-4.25E-19	1.27E-18	-3.02E-02
4	I	0.00	0.00	100.00	0.00E+00	0.00E+00	-2.74E-02
4	o	0.00	0.00	100.00	0.00E+00	0.00E+00	-3.02E-02
5	I	0.00	0.00	100.00	0.00E+00	0.00E+00	-2.74E-02
5	o	0.00	0.00	100.00	0.00E+00	0.00E+00	-3.02E-02

Table 8-XXV. Strain deviation from the Standard Response for nu12/2

Nu12/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-60.20	0.39	3.29	-6.95E-04	-5.31E-05	1.21E-19
1	o	-60.20	0.51	3.29	-6.95E-04	-5.99E-05	1.34E-19
2	I	0.50	-60.20	-7.45	-1.90E-05	-2.21E-04	1.33E-20
2	o	0.50	-42.66	-7.45	-1.90E-05	-2.04E-04	1.47E-20
3	I	-7.45	3.29	0.00	1.33E-20	3.87E-20	0.00E+00
3	o	-7.45	3.23	0.00	1.33E-20	3.34E-20	0.00E+00
4	I	0.50	-60.20	0.00	-1.90E-05	-2.21E-04	0.00E+00
4	o	0.50	-42.66	0.00	-1.90E-05	-2.04E-04	0.00E+00
5	I	27.22	2.06	0.00	-7.14E-04	-2.74E-04	0.00E+00
5	o	27.22	2.32	0.00	-7.14E-04	-2.64E-04	0.00E+00

Table 8-XXVI. Strain deviation from the Standard Response for nu13/2

Nu13/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	5.30	-0.87	0.28	6.12E-05	1.19E-04	1.04E-20
1	o	5.30	0.04	0.28	6.12E-05	-4.20E-06	1.15E-20
2	I	0.01	5.30	-0.15	-2.60E-07	1.95E-05	2.69E-22
2	o	0.01	-3.24	-0.15	-2.60E-07	-1.55E-05	2.96E-22
3	I	-0.15	0.28	0.00	2.66E-22	3.67E-21	0.00E+00
3	o	-0.15	-0.29	0.00	2.69E-22	-3.05E-21	0.00E+00
4	I	0.01	5.30	0.00	-2.60E-07	1.95E-05	0.00E+00
4	o	0.01	-3.24	0.00	-2.60E-07	-1.55E-05	0.00E+00
5	I	-2.32	-1.04	0.00	6.09E-05	1.39E-04	0.00E+00
5	o	-2.32	0.17	0.00	6.09E-05	-1.97E-05	0.00E+00

Table 8-XXVII. Strain deviation from the Standard Response for nu23/2

Nu23/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	5.76	-1.07	-1.05	6.65E-05	1.47E-04	-3.88E-20
1	o	5.76	-0.09	-1.05	6.65E-05	1.04E-05	-4.27E-20
2	I	0.01	5.76	-0.24	-3.10E-07	2.12E-05	4.23E-22
2	o	0.01	-4.22	-0.24	-3.10E-07	-2.02E-05	4.65E-22
3	I	-0.24	-1.05	0.00	4.23E-22	-1.24E-20	0.00E+00
3	o	-0.24	1.13	0.00	4.23E-22	1.17E-20	0.00E+00
4	I	0.01	5.76	0.00	-3.10E-07	2.12E-05	0.00E+00
4	o	0.01	-4.22	0.00	-3.10E-07	-2.02E-05	0.00E+00
5	I	-2.52	-1.26	0.00	6.62E-05	1.68E-04	0.00E+00
5	o	-2.52	0.09	0.00	6.62E-05	-9.80E-06	0.00E+00

8.4 Thick Tube 2

Table 8-XXVIII. Strain Response for the different loading conditions for Thick Tube 2

Standard Response				
Load	Surface	Axial	Hoop	Shear
1	I	-8.89E-04	1.70E-02	-3.72E-18
1	o	-8.89E-04	1.18E-02	-4.65E-18
2	I	3.78E-03	-2.83E-04	1.70E-19
2	o	3.78E-03	-5.45E-04	2.13E-19
3	I	1.74E-19	-1.21E-18	2.41E-02
3	o	1.74E-19	-9.02E-19	3.02E-02
4	I	3.78E-03	-2.83E-04	2.41E-02
4	o	3.78E-03	-5.45E-04	3.02E-02
5	I	2.89E-03	1.67E-02	2.41E-02
5	o	2.89E-03	1.13E-02	3.02E-02

Table 8-XXIX. Strain deviation from the Standard Response for E1/2

E1/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	253.58	62.63	-38.64	2.25E-03	-1.07E-02	-1.44E-18
1	o	253.58	66.40	-38.64	2.25E-03	-7.83E-03	-1.80E-18
2	I	65.48	253.58	23.05	-2.48E-03	7.17E-04	-3.92E-20
2	o	65.48	140.49	23.05	-2.48E-03	7.65E-04	-4.91E-20
3	I	23.05	-38.64	0.00	-4.00E-20	-4.67E-19	0.00E+00
3	o	23.05	-37.84	0.00	-4.00E-20	-3.41E-19	0.00E+00
4	I	65.48	253.58	0.00	-2.48E-03	7.17E-04	0.00E+00
4	o	65.48	140.49	0.00	-2.48E-03	7.65E-04	0.00E+00
5	I	7.72	59.40	0.00	-2.23E-04	-9.94E-03	0.00E+00
5	o	7.72	62.81	0.00	-2.23E-04	-7.07E-03	0.00E+00

Table 8-XXX. Strain deviation from the Standard Response for E2/2

E2/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-75.36	11.39	15.54	-6.70E-04	-1.94E-03	5.78E-19
1	o	-75.36	12.48	15.54	-6.70E-04	-1.47E-03	7.23E-19
2	I	11.89	-75.36	-105.22	-4.50E-04	-2.13E-04	1.79E-19
2	o	11.89	-22.26	-105.22	-4.50E-04	-1.21E-04	2.24E-19
3	I	-105.22	15.54	0.00	2.23E-19	6.55E-19	0.00E+00
3	o	-105.22	13.57	0.00	1.83E-19	1.22E-19	0.00E+00
4	I	11.89	-75.36	0.00	-4.50E-04	-2.13E-04	0.00E+00
4	o	11.89	-22.26	0.00	-4.50E-04	-1.21E-04	0.00E+00
5	I	38.68	12.85	0.00	-1.12E-03	-2.15E-03	0.00E+00
5	o	38.68	14.16	0.00	-1.12E-03	-1.59E-03	0.00E+00

Table 8-XXXI. Strain deviation from the Standard Response for E3/2

E3/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.65	3.61	-0.29	5.79E-06	-6.14E-04	-1.09E-20
1	o	0.65	-1.94	-0.29	5.79E-06	2.29E-04	-1.37E-20
2	I	0.00	0.65	-0.03	-3.00E-08	1.84E-06	5.40E-23
2	o	0.00	-0.27	-0.03	-3.00E-08	-1.46E-06	6.70E-23
3	I	-0.03	-0.29	0.00	5.50E-23	-3.56E-21	0.00E+00
3	o	-0.03	0.32	0.00	5.50E-23	2.90E-21	0.00E+00
4	I	0.00	0.65	0.00	-3.00E-08	1.84E-06	0.00E+00
4	o	0.00	-0.27	0.00	-3.00E-08	-1.46E-06	0.00E+00
5	I	-0.20	3.66	0.00	5.76E-06	-6.13E-04	0.00E+00
5	o	-0.20	-2.02	0.00	5.76E-06	2.27E-04	0.00E+00

Table 8-XXXII. Strain deviation from the Standard Response for G12/2

G12/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	0.00	0.00	121.47	0.00E+00	0.00E+00	4.52E-18
1	o	0.00	0.00	121.47	0.00E+00	0.00E+00	5.65E-18
2	I	0.00	0.00	235.49	0.00E+00	0.00E+00	-4.01E-19
2	o	0.00	0.00	235.49	0.00E+00	0.00E+00	-5.01E-19
3	I	235.49	121.47	100.00	-4.09E-19	1.47E-18	-2.41E-02
3	o	235.49	123.14	100.00	-4.09E-19	1.11E-18	-3.02E-02
4	I	0.00	0.00	100.00	0.00E+00	0.00E+00	-2.41E-02
4	o	0.00	0.00	100.00	0.00E+00	0.00E+00	-3.02E-02
5	I	0.00	0.00	100.00	0.00E+00	0.00E+00	-2.41E-02
5	o	0.00	0.00	100.00	0.00E+00	0.00E+00	-3.02E-02

Table 8-XXXIII. Strain deviation from the Standard Response for nu12/2

Nu12/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	-82.83	0.23	3.31	-7.36E-04	-3.96E-05	1.23E-19
1	o	-82.83	0.51	3.31	-7.36E-04	-6.06E-05	1.54E-19
2	I	0.50	-82.83	-7.04	-1.89E-05	-2.34E-04	1.20E-20
2	o	0.50	-35.79	-7.04	-1.89E-05	-1.95E-04	1.50E-20
3	I	-7.04	3.31	0.00	1.22E-20	4.00E-20	0.00E+00
3	o	-7.04	3.15	0.00	1.22E-20	2.84E-20	0.00E+00
4	I	0.50	-82.83	0.00	-1.89E-05	-2.34E-04	0.00E+00
4	o	0.50	-35.79	0.00	-1.89E-05	-1.95E-04	0.00E+00
5	I	26.09	1.64	0.00	-7.55E-04	-2.74E-04	0.00E+00
5	o	26.09	2.27	0.00	-7.55E-04	-2.55E-04	0.00E+00

Table 8-XXXIV. Strain deviation from the Standard Response for nu13/2

Nu13/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	16.81	-1.91	0.62	1.49E-04	3.25E-04	2.32E-20
1	o	16.81	0.12	0.62	1.49E-04	-1.46E-05	2.90E-20
2	I	0.03	16.81	-0.64	-1.06E-06	4.76E-05	1.08E-21
2	o	0.03	-6.37	-0.64	-1.06E-06	-3.47E-05	1.35E-21
3	I	-0.64	0.62	0.00	1.05E-21	1.11E-20	0.00E+00
3	o	-0.64	-0.74	0.00	1.11E-21	-6.66E-21	0.00E+00
4	I	0.03	16.81	0.00	-1.06E-06	4.76E-05	0.00E+00
4	o	0.03	-6.37	0.00	-1.06E-06	-3.47E-05	0.00E+00
5	I	-5.12	-2.23	0.00	1.48E-04	3.73E-04	0.00E+00
5	o	-5.12	0.44	0.00	1.48E-04	-4.93E-05	0.00E+00

Table 8-XXXV. Strain deviation from the Standard Response for nu23/2

Nu23/2		Percentage Change from Standard			Deviation from Standard		
Load	Surface	Axial	Hoop	Shear	Axial	Hoop	Shear
1	I	18.26	-2.20	-2.32	1.62E-04	3.75E-04	-8.63E-20
1	o	18.26	-0.25	-2.32	1.62E-04	2.99E-05	-1.08E-19
2	I	0.03	18.26	-0.97	-1.23E-06	5.17E-05	1.65E-21
2	o	0.03	-8.30	-0.97	-1.23E-06	-4.52E-05	2.07E-21
3	I	-0.97	-2.32	0.00	1.69E-21	-2.80E-20	0.00E+00
3	o	-0.97	2.79	0.00	1.69E-21	2.52E-20	0.00E+00
4	I	0.03	18.26	0.00	-1.23E-06	5.17E-05	0.00E+00
4	o	0.03	-8.30	0.00	-1.23E-06	-4.52E-05	0.00E+00
5	I	-5.56	-2.55	0.00	1.61E-04	4.27E-04	0.00E+00
5	o	-5.56	0.14	0.00	1.61E-04	-1.53E-05	0.00E+00

9 Appendix C. Levenberg-Marquardt Inversion – Cij

Here is the source code for the Inversion code using the Levenberg-Marquardt compromise to optimize the Cij values. Included are the C source codes for all the files. Files included:

- Inversion.cpp
- Inversion.h
- Elastic_Solution.cpp
- Elastic_Solution.h
- Data_Input.cpp
- Data_Input.h
- Matrix.cpp
- Matrix.h
- Jacobian.cpp
- Jacobian.h

9.1 Inversion.cpp

```
// Inversion Program -> Inversion.cpp

#include <stdlib.h>
#include <iostream.h>
#include <iomanip.h>

#include "Data_Input.h"
#include "Elastic_Solution.h"
#include "Inversion.h"
#include "jacobian.h"
#include <time.h>
```

```
void main()
{
      int count, convergence, i;
      int matnum, mat[K];
      double E[5][10], Enew[7], Eout[7], C[5][7], PHI[ND], SSE, B[NAP], Cij[MaxIteration][7];
      double radii[K+1], theta[K], Loads[numloads][4], Exp_Strain[numloads][3], Calculated_Strain[3],
C_Strain[numloads][3], duration;
      double C_original[7], SSEmin=1.0, E_original[7], *del, delta;
      char position[numloads];
      ofstream  CijFile;
      clock_t start, finish;
      ofstream  SSEFile;

      SSEFile.open("SSE.dat");
      CijFile.open("Cij.dat");

      delta= 0.0001;
      del=&delta;

      start = clock();

// STEP 1.  Read in all data

      matnum = Input(E, C, mat, radii, theta, Loads, Exp_Strain, position);


// Check for an inversion, search for a minimum or mapping of initial SSE values (much quicker than search)
      for (int a=0;a<7;a++)
      {
            C_original[a] = C[0][a];
            E_original[a] = E[0][a];
      }

/*****************************************************************************/

// STEP 2.  Show initial results
```

```
      CijFile << "Initial Values:"<<endl;

      CijFile << endl<< "E Values:"<< endl;
      CijFile << "\nE1"<< "\t"<< E[0][0]<<"\t";
      CijFile << "\nE2"<< "\t"<< E[0][1]<<"\t";
      CijFile << "\nE3"<< "\t"<< E[0][2]<<"\t";
      CijFile << "\nG12"<< "\t"<< E[0][3]<<"\t";
      CijFile << "\nnu12"<< "\t"<< E[0][4]<<"\t";
      CijFile << "\nnu13"<< "\t"<< E[0][5]<<"\t";
      CijFile << "\nnu23"<< "\t"<< E[0][6]<<"\t";

      CijFile << "\n\nCij Values:"<<endl;
      CijFile << "\nC11"<< "\t"<< C[0][0]<<"\t";
      CijFile << "\nC12"<< "\t"<< C[0][1]<<"\t";
      CijFile << "\nC13"<< "\t"<< C[0][2]<<"\t";
      CijFile << "\nC22"<< "\t"<< C[0][3]<<"\t";
      CijFile << "\nC23"<< "\t"<< C[0][4]<<"\t";
      CijFile << "\nC33"<< "\t"<< C[0][5]<<"\t";
      CijFile << "\nC66"<< "\t"<< C[0][6]<<"\t"<<endl;

      Calc_Increment(E, C, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI, B, &SSE,del);

      cout << endl << "\nInitial SSE = " <<SSE;
      cout << endl << "Acceptable Error Value = "<< Acceptable << endl;
      CijFile << endl << "\nInitial SSE = " <<SSE;
      SSEFile <<"Initial SSE for Guess Values\t"<<SSE<<endl;
      SSEFile <<"\n0\t"<< SSE<<"\t"<<*del;


// STEP 3.  Minimize the error function and return information on the type of solution reached

      convergence = Minimize(E, C, matnum, mat, Cij, radii,  theta, Loads,Exp_Strain, position, PHI, B,
&SSE,&count, SSEFile,del);
/*
      if (convergence==4)
      {
            *del=0;
```

```
            convergence = Minimize(E, C, matnum, mat, Cij, radii,  theta, Loads,Exp_Strain, position, PHI,
B, &SSE,&count, SSEFile,del);
      }
*/
// STEP 4.  Give information on convergence conditions

      switch(convergence)
      {
            case 0: cout<<"\nSingular matrix encountered"<<endl;
                  break;
            case 1: cout<< endl<<"Local minimum encountered"<< endl;
                  break;
            case 2: cout<< endl<<"Exceded Maximum interations"<< endl;
                  break;
            case 3: cout<< endl<<"SSE sufficiently small"<< endl;
                  break;
            case 4: cout<<endl<<"Insignificant change to values"<< endl;
                  break;
            case 5: cout << endl<<"Did not Converge"<< endl;
                  break;
      }

// STEP 5.  Output final values
      if(convergence!=0)
      {
            cout<< "\nFinal SSE Value = " << SSE << endl;
            cout<< "\nNumber of iterations = "<< count+1<< endl;
            CijFile << "\nNumber of iterations = \t"<< count+1<< endl;
            CijFile << endl<< "Final Cij Values:"<< endl;

            CijFile << "\nC11"<< "\t"<< C[0][0]<<"\t";
            CijFile << "\nC12"<< "\t"<< C[0][1]<<"\t";
            CijFile << "\nC13"<< "\t"<< C[0][2]<<"\t";
            CijFile << "\nC22"<< "\t"<< C[0][3]<<"\t";
            CijFile << "\nC23"<< "\t"<< C[0][4]<<"\t";
            CijFile << "\nC33"<< "\t"<< C[0][5]<<"\t";
            CijFile << "\nC66"<< "\t"<< C[0][6]<<"\t";
```

```
            C_to_E(&(C[0][0]),&(Enew[0]));

            CijFile << endl<<endl<< "Final E Values:"<< endl;
            CijFile << "\nE1"<< "\t"<< Enew[0]<<"\t";
            CijFile << "\nE2"<< "\t"<< Enew[1]<<"\t";
            CijFile << "\nE3"<< "\t"<< Enew[2]<<"\t";
            CijFile << "\nG12"<< "\t"<< Enew[3]<<"\t";
            CijFile << "\nnu12"<< "\t"<< Enew[4]<<"\t";
            CijFile << "\nnu13"<< "\t"<< Enew[5]<<"\t";
            CijFile << "\nnu23"<< "\t"<< Enew[6]<<"\t";

            CijFile <<"\nFinal SSE = \t" << SSE<< endl<<endl;
            for(i=0;i<7;i++)
                  E[0][i]=Enew[i];
      }
      else
      {
            CijFile<<"\nSingular Matrix\n";
      }

//Calculate the strain response for the solution values and output in table for easy plotting

      CijFile<<"\nCalculated Strains for the Final Values";
      CijFile<<"\nLoads\t\t\t\t\t"<<"Measured Strains\t\t\t\t"<<"Calculated Strains\n";
      CijFile<<"Po\t"<<"Pi\t"<<"Fx\t"<<"Tx\t\t"<<"Ax\t"<<"Hoop\t"<<"Shear\t\t"<<"Ax\t"<<"Hoop\t"<<"Shear\t"
<<"\tSurface\n";

      for(i=0 ; i<numloads ; i++)
      {

            Elastic_Solution(C, E, matnum, mat, radii, theta, &(Loads[i][0]), Calculated_Strain,
position[i]);

            for(int j=0;j<3;j++)
                  C_Strain[i][j]=Calculated_Strain[j];
            for(j=0;j<4;j++)
                  CijFile<<Loads[i][j]<<"\t";
            CijFile<<"\t";
```

```
            for(j=0;j<3;j++)
                  CijFile<<Exp_Strain[i][j]<<"\t";
            CijFile<<"\t";
            for(j=0;j<3;j++)
                  CijFile<<C_Strain[i][j]<<"\t";
            CijFile<<"\t"<<position[i]<<endl;

      }

// Calculates time used to find solution

      finish = clock();
      duration = (double)(finish - start)/CLOCKS_PER_SEC;
      if(duration>60.0)
      {
            duration=duration/60.0;
            cout << "\nCalculation Time:  "<<duration<< " minutes."<< endl;
      }
      else
      {
            cout << "\nCalculation Time:  "<<duration<< " seconds."<< endl;
      }

      for(i=0;i<count+1;i++)
      {
            CijFile <<"\nIteration #\t"<<i+1<<endl;
            CijFile << "\nC11"<< "\t"<< Cij[i][0]<<"\t";
            CijFile << "\nC12"<< "\t"<< Cij[i][1]<<"\t";
            CijFile << "\nC13"<< "\t"<< Cij[i][2]<<"\t";
            CijFile << "\nC22"<< "\t"<< Cij[i][3]<<"\t";
            CijFile << "\nC23"<< "\t"<< Cij[i][4]<<"\t";
            CijFile << "\nC33"<< "\t"<< Cij[i][5]<<"\t";
            CijFile << "\nC66"<< "\t"<< Cij[i][6]<<"\t"<<endl;

            C_to_E(&(Cij[i][0]),&(Eout[0]));

            CijFile << "\nE1"<< "\t"<< Eout[0]<<"\t";
            CijFile << "\nE2"<< "\t"<< Eout[1]<<"\t";
```

```
            CijFile << "\nE3"<< "\t"<< Eout[2]<<"\t";
            CijFile << "\nG12"<< "\t"<< Eout[3]<<"\t";
            CijFile << "\nnu12"<< "\t"<< Eout[4]<<"\t";
            CijFile << "\nnu13"<< "\t"<< Eout[5]<<"\t";
            CijFile << "\nnu23"<< "\t"<< Eout[6]<<"\t"<<endl;
      }
      CijFile.close();
}

//-------------------------------------------------------------------------------------------
/*Calc_Increment calculates the error value PHI and SSE as well as the incremental step value for Cij
*/
int Calc_Increment(double E[][10], double C[][7], int matnum, int mat[], double radii[], double theta[],
double Loads[][4], double Exp_Strain[][3], char position[], double PHI[], double B[], double *SumSquareErr,
double *del)
{
      int indx[NAP],i,crash=0;
      double J[ND][NAP], JtJ[NAP][NAP];

      *SumSquareErr = 0.0;

      for (i=0;i<NAP;i++)
      {
            for(int jj=0; jj<NAP;jj++)
                  JtJ[i][jj] = 0.0;
      }
      for(i=0;i<ND;i++)
            PHI[i] = 0.0;

      PhiSub(E, C, matnum, mat, radii,theta,Loads, Exp_Strain, position, PHI);

      for(i=0;i<ND;i++)
            *SumSquareErr += pow(PHI[i],2);     //Calculate the Sum of Square Errors

      Jacobian(E, C, matnum, mat, radii, theta, Loads, Exp_Strain, position, J);

      //Calculate ([J]t[J])^(-1)[J]t{PHI}
```

```
	for(i=0;i<NAP;i++)	// [J]t[J]
	{
		for(int j=0;j<NAP;j++)
		{
			for(int ii=0;ii<ND;ii++)
				JtJ[i][j] += J[ii][i]*J[ii][j];
		}
	}
//Levenberg-Marquardt   (JtJ+delta*D)
	for(i=0;i<NAP;i++)
		JtJ[i][i] *=(1.0 + *del);

	for(i=0;i<NAP;i++)	//[J]t{PHI}
	{
		B[i] = 0.0;
		for (int j=0;j<ND;j++)
			B[i] -=J[j][i]*PHI[j];
	}
	//SysScale(&(JtJ[0][0]),B,NAP);
	crash = LUDecomposition(&(JtJ[0][0]),NAP,indx,i);
	if(crash!=0)
	{
		cout <<"\nSingular Matrix"<<endl;
		*SumSquareErr = 1.0;
		return 1;
	}

	LUBackSub(&(JtJ[0][0]),NAP,indx,B);
	return 0;
}

//-------------------------------------------------------------------------------------------
//Function to minimize error and apply convergence criteria
int Minimize(double E[][10], double C[][7], int matnum, int mat[], double Cij[][7], double radii[], double
theta[], double Loads[][4], double Exp_Strain[][3], char position[], double PHI[], double B[], double *sse,
int *counter, ofstream SSEFile, double *del)
{
	int crash_big, crash_small, crash_del,i,j,flag,count=0;
```

```
      double CNew[5][7], CNew2[5][7],SSEnew, SSEold, del_orig, deltabig, deltasmall,minimum;
      double SSE_del=0.0, SSE_big=0.0,SSE_small=0.0, B_big[7], B_small[7], ratio[7],max=0.0,SSE_ratio;
      double CNew_big[5][7],CNew_small[5][7],SSE_del_old=0.0, SSE_big_old=0.0,SSE_small_old=0.0;
      SSEnew = 1000000;
      SSEold = 1000000;
      flag=0;
      minimum=0.0001;
      del_orig = *del;

      cout<<"\nIteration\t"<<"SSE"<<"\tdelta";

//***********Define the new C values************
      for(i=0;i<matnum;i++)
      {
            for(j=0;j<7;j++)
            {
                  CNew[i][j]=C[i][j];
                  CNew2[i][j]=C[i][j];
                  CNew_small[i][j]=C[i][j];
                  CNew_big[i][j]=C[i][j];
            }
      }
//*********Check solution near convergence********
/*    if(*del==0)
      {
            crash_del = Calc_Increment(E, C, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI,
B, &SSE_del, del);
            if (crash_del!=0)
            {
                  cout<<"Singular Matrix at solution";
                  return 1;
            }
            SSEold=SSE_del;
            for (i=0;i<7;i++)
            {
                  CNew[0][i]+=B[i];
            }
```

```
            crash_del = Calc_Increment(E, CNew, matnum, mat, radii, theta, Loads, Exp_Strain, position,
PHI, B, &SSE_del, del);
      }
*/
      for (int iteration=0;iteration <MaxIteration;iteration++)
      {
            *counter=iteration;
//          count+=1;
//          if(count==10)//periodically perturb del to make sure it is optimizing
//          {
//                *del/=10000;
//                count=0;
//          }
//***********Calculate error function and correction to values*************

            deltabig=*del*10;
            deltasmall=*del/10;
            del_orig = *del;

//Most efficient step size delta*10, 1, 0.1
//************Calculate three different B values for the different delta values************

            crash_del = Calc_Increment(E, C, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI,
B, &SSE_del, del);
            crash_big = Calc_Increment(E, C, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI,
B_big, &SSE_big, &deltabig);
            crash_small = Calc_Increment(E, C, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI,
B_small, &SSE_small, &deltasmall);

            SSEold=SSE_del;         //all SSE values are the same here - same C values input, only the B
values change

            if(SSEold<=Acceptable)
            {
                  *sse = SSEold;
                  SSEFile << endl<<iteration+1<<"\t"<<SSEold<<"\t"<<*del;
                  cout<<endl<<SSEold;
                  return 3;
```

```
            }

            if ((crash_del != 0)&&(crash_big !=0)&&(crash_small !=0))
            {
                  *sse = 1.0;
                  return 0;
            }

            for(i=0;i<7;i++)
            {
                  CNew[0][i]=C[0][i]+B[i];
                  ratio[i]=B[i];
                  CNew_big[0][i]=C[0][i]+B_big[i];
                  CNew_small[0][i]=C[0][i]+B_small[i];
            }
            if ((CNew[0][3]==CNew[0][5])||(CNew[0][3]==4*CNew[0][5]))
                  CNew[0][5]+=1;
            if ((CNew_big[0][3]==CNew_big[0][5])||(CNew_big[0][3]==4*CNew_big[0][5]))
                  CNew_big[0][5]+=1;
            if ((CNew_small[0][3]==CNew_small[0][5])||(CNew_small[0][3]==4*CNew_small[0][5]))
                  CNew_small[0][5]+=1;


            crash_del = Calc_Increment(E, CNew, matnum, mat, radii, theta, Loads, Exp_Strain, position,
PHI, B, &SSE_del, del);
            crash_big = Calc_Increment(E, CNew_big, matnum, mat, radii, theta, Loads, Exp_Strain, position,
PHI, B, &SSE_big, &deltabig);
            crash_small = Calc_Increment(E, CNew_small, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B, &SSE_small, &deltasmall);


            if ((crash_del != 0)&&(crash_big !=0)&&(crash_small !=0))
            {
                  *sse = 1.0;
                  return 0;
            }
/*          for(i=0;i<7;i++)
            {
```

```
                if(CNew[0][i]<=0.0)
                {
                      SSE_del = 1e23;
                }
                if(CNew_big[0][i]<=0.0)
                {
                      SSE_big = 1e23;
                }
                if(CNew_small[0][i]<=0.0)
                {
                      SSE_small = 1e23;
                }
          }
*/
          SSEnew= SSE_del;

          if((SSE_big<SSE_del)&&(SSE_big<SSE_small))
          {
                *del*=10;
                SSEnew= SSE_big;
                for(i=0;i<7;i++)
                {
                      CNew[0][i]=CNew_big[0][i];
                      ratio[i]=B_big[i];
                }

          }
          if((SSE_small<SSE_del)&&(SSE_small<SSE_big))
          {
                *del/=10;
                SSEnew= SSE_small;
                for(i=0;i<7;i++)
                {
                      CNew[0][i]=CNew_small[0][i];
                      ratio[i]=B_small[i];
                }
          }
//If value is below the threshold level
```

```
            if(SSEnew<=Acceptable)//if error is below an acceptable value - quit
                  {
                        for(i=0;i<7;i++)
                        {
                              C[0][i] = CNew[0][i];
                              Cij[iteration][i] = CNew[0][i];
                        }
                        *sse = SSEnew;
                        SSEFile <<endl <<iteration+1<<"\t"<<SSEnew<<"\t"<<*del;
                        cout<<endl<<SSEnew;
                        return 3;
                  }
//**********if the revision reduces SSE - take step**********
            if(SSEnew<=SSEold)
            {
                  max=0.0;
                  for(i=0;i<7;i++)
                  {
                        C[0][i] = CNew[0][i];
                        Cij[iteration][i] = CNew[0][i];
                        ratio[i]/=CNew[0][i];               //Checking the size of the steps
                        if(max<ratio[i])                    //take the largest
                              max=ratio[i];
                  }
                  SSE_ratio = SSEnew/SSEold;
                  SSEold = SSEnew;
                  *sse = SSEnew;


                  if((max<=minimum))//||((1-SSE_ratio)<=minimum))
                  {
                        flag+=1;
                  }
                  else
                  {
                        flag=0;
```

```
                  }
                  if(flag>4)
                        return 4;
            }
//***********if not - Change Delta to see if it helps - repeat until it is or quit***********
            else
            {
                  while (SSEnew>SSEold)
                  {
                        *del*=10;
                        cout<<"\nChanging Delta Value - del = "<<*del;

//***********Trying something**********
                        deltabig=*del*10;
                        deltasmall=*del/10;
                        del_orig = *del;

                        crash_del = Calc_Increment(E, C, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B, &SSE_del, del);
                        crash_big = Calc_Increment(E, C, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B_big, &SSE_big, &deltabig);
                        crash_small = Calc_Increment(E, C, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B_small, &SSE_small, &deltasmall);

                        if ((crash_del != 0)||(crash_big !=0)||(crash_small !=0))
                        {
                              *sse = 1.0;
                              return 0;
                        }


                        for(i=0;i<7;i++)
                        {
                              CNew[0][i]=C[0][i]+B[i];
                              CNew_big[0][i]=C[0][i]+B_big[i];
                              CNew_small[0][i]=C[0][i]+B_small[i];
                        }
```

```
                        crash_del = Calc_Increment(E, CNew, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B, &SSE_del, del);
                        crash_big = Calc_Increment(E, CNew_big, matnum, mat, radii, theta, Loads,
Exp_Strain, position, PHI, B, &SSE_big, &deltabig);
                        crash_small = Calc_Increment(E, CNew_small, matnum, mat, radii, theta, Loads,
Exp_Strain, position, PHI, B, &SSE_small, &deltasmall);


                        if ((crash_del != 0)||(crash_big !=0)||(crash_small !=0))
                        {
                        *sse = 1.0;
                        return 0;
                        }

                        SSEnew= SSE_del;

/*                      for(i=0;i<7;i++)
                        {
                              if(CNew[0][i]<=0.0)
                              {
                                    SSE_del = 1e23;
                              }
                              if(CNew_big[0][i]<=0.0)
                              {
                                    SSE_big = 1e23;
                              }
                              if(CNew_small[0][i]<=0.0)
                              {
                                    SSE_small = 1e23;
                              }
                        }
*/
                        if((SSE_big<SSE_del)&&(SSE_big<SSE_small))
                        {
                              SSEnew= SSE_big;
                              for(i=0;i<7;i++)
                              CNew[0][i]=CNew_big[0][i];
```

```
                        }
                        if((SSE_small<SSE_del)&&(SSE_small<SSE_big))
                        {
                              SSEnew= SSE_small;
                              for(i=0;i<7;i++)
                                    CNew[0][i]=CNew_small[0][i];

                        }

//***********Check new values now that delta is increased


                        if(SSEnew<SSEold) //if SSE is reduced after a interval halving - make changes
                        {
                              for(i=0;i<7;i++)
                              {
                                    C[0][i] = CNew[0][i];
                                    Cij[iteration][i] = C[0][i];
                              }
                              SSEold = SSEnew;
                              *sse = SSEnew;

                        }
                        if (*del>1000) //after several interval halving steps
                        {
                              *sse = SSEold;
                              SSEFile <<endl <<iteration+1<<"\t"<<SSEold<<"\t"<<*del;

                              cout<<endl<<SSEold;
                              for(i=0;i<7;i++)
                                    Cij[iteration][i] = C[0][i];
                              return 1;
                        }

                  }
                  cout<<endl<<iteration<<"\t"<<*sse<<"\t"<<*del;
            }
```

```
            SSEFile <<endl <<iteration+1<<"\t"<<*sse<<"\t"<<*del;
            cout<<endl<<iteration+1<<"\t"<<*sse<<"\t"<<*del;
      }

      *sse = SSEnew;
      SSEFile.close();
      return 2;
}

/*****************************************************************************/
//This function converts the Cij Values back to E1, E2, E3, G12, nu12, nu13, nu23

void C_to_E(double *c, double *Evalues)
{

      double S[4][4], col[4], CijMatrix[4][4];
      int i, j, indx[4], d=1;



      //Need to build the Cij matrix
      for(i=0;i<4;i++)
      {
            col[i]=0.0;
            for(j=0;j<4;j++)
            {
                  CijMatrix[i][j]=0.0;
                  S[i][j] = 0.0;
            }
      }

      CijMatrix[0][0] = c[0]; //C11
      CijMatrix[0][1] = c[1]; //C12
      CijMatrix[1][0] = c[1]; //C21
      CijMatrix[0][2] = c[2]; //C13
      CijMatrix[2][0] = c[2]; //C31
      CijMatrix[1][1] = c[3]; //C22
      CijMatrix[1][2] = c[4]; //C23
```

```
	CijMatrix[2][1] = c[4]; //C32
	CijMatrix[2][2] = c[5]; //C33
	CijMatrix[3][3] = c[6]; //C66


//Invert the Cij Matrix to get the Sij Matrix

	LUDecomposition(&CijMatrix[0][0],4,indx,d);
	for(j=0;j<4;j++)
	{
		for(i=0;i<4;i++)
			col[i] = 0.0;
		col[j] = 1.0;
		LUBackSub(&(CijMatrix[0][0]),4,indx,&col[0]);
		for(i=0;i<4;i++)
			S[i][j]=col[i];
	}

//Calculate E values from the Sij Values

	Evalues[0] = 1.0/S[0][0];
	Evalues[1] = 1.0/S[1][1];
	Evalues[2] = 1.0/S[2][2];
	Evalues[3] = 1.0/S[3][3];
	Evalues[4] = -S[0][1]*Evalues[0];
	Evalues[5] = -S[0][2]*Evalues[0];
	Evalues[6] = -S[1][2]*Evalues[1];
}
```

9.2 Inversion.h

```
//Inversion.h

// Header file for the Inverse Solution
// Modify the needed information for each different material run
#include <math.h>
#include <fstream.h>
```

```
#define K 10                        //Number of plies in the composite lay-up
#define KK 2*K+2        //Number of equations for Elastic Solution
#define dT 0                  //Temperature change for thermal strains and stresses
#define Pi acos(-1)           //Define Pi
#define numloads 3            //Number of load conditions applied (1 Fx + 1 Tx + 1 Pi = 3 !!!REQUIRED!!!)
#define NAP 7                 //Number of active parameters (number of parameters being optimized - 7 Cij
values)
#define ND  3*numloads  //Number of data points (number of strain points given - 3 strains per load
applied)
#define Acceptable 1e-015     //Convergence Criteria - SSE value that is sufficiently small
#define MaxIteration 100      //Number of iterations before quitting

int Calc_Increment(double E[][10], double C[][7],int matnum, int mat[], double radii[], double theta[],
double Loads[][4], double Exp_Strain[][3], char position[], double PHI[], double B[], double *SumSquareErr,
double *del);
int Minimize(double E[][10], double C[][7],int matnum, int mat[], double Cij[][7], double radii[], double
theta[], double Loads[][4], double Exp_Strain[][3], char position[], double PHI[], double B[], double *SSE,
int *count, ofstream SSEFile, double *del);
//void Calc_SSE(double E[][10], double C[][7], int matnum, int mat[], double radii[], double theta[],
double Loads[][4], double Exp_Strain[][3], char position[], double PHI[], double *SumSquareErr);
void C_to_E(double *c, double *Evalues);
```

9.3 Elastic Solution.cpp

```
//Elastic_Solution.cpp
#include "Elastic_Solution.h"
#include <iostream.h>
#include <fstream.h>

void Elastic_Solution(double C[][7], double E[][10], int matnum, int mat[], double r[], double th[], double
*load, double Calc_Strain[], char strainposition)
{
      int i=0,indx[KK];
      long theErr=0;
      double sc[K][4][5], bc[K][3], L[K], cbm[K][4][4], et[K][4];
      double A[KK][KK], B[KK];
```

```
      Cbar(C, th, E, matnum, mat, cbm, et);           //Transformations to create the cbar matrix

      stress(cbm, et, L, bc, sc);                     //Relations between strain, displacement and stress

      mat_stack(cbm, sc, L, bc, r, load, A, B); //Creates matrix to solve for constants Eo,Go, A1's and
A2's

      SysScale(&(A[0][0]),B,KK);                      //Scaling to increase numerical accuracy

      for (int jj=0;jj<KK;jj++)
            indx[jj] = 0;

      LUDecomposition(&(A[0][0]),KK,indx,i);    //performs PA=LU for LUBackSub

      LUBackSub(&(A[0][0]),KK,indx,B);          //Solves system PAx=PB (returns x in B)


      output(B, r, L, bc, Calc_Strain, strainposition);//Returns calculated strains
}

//*********************************************************
//***********Internal Subroutines*****************************


void Cbar(double C[][7], double theta[], double E[][10], int matnum, int mat[], double cbm[][4][4], double
et[][4])
{
 //c-bar matrix function

      int i,j,k,p;
      double m=0.0,n=0.0;

      for(i=0;i<K;i++)
            for(j=0;j<4;j++)
                  for(k=0;k<4;k++)
                        cbm[i][j][k]=0.0;

      for(i=0;i<K;i++)
```

```
	{
		m = cos(theta[i]);
		n = sin(theta[i]);

		p = mat[i];  //allows for the different material layers

		cbm[i][0][0] = C[p][0]*pow(m,4) + (2*pow(m*n,2))*(C[p][1] + 2*C[p][6]) + C[p][3]*pow(n,4);
		cbm[i][0][1] = pow(m*n,2)*(C[p][0] + C[p][3] - 4*C[p][6]) + C[p][1]*(pow(m,4) + pow(n,4));
		cbm[i][0][2] = C[p][2]*m*m + C[p][4]*n*n;
		cbm[i][0][3] = m*n*(C[p][0]*m*m - C[p][3]*n*n - (C[p][1] + 2*C[p][6])*(m*m - n*n));
		cbm[i][1][1] = C[p][0]*pow(n,4) + (2*pow(m*n,2))*(C[p][1] + 2*C[p][6]) + C[p][3]*pow(m,4);
		cbm[i][1][2] = C[p][2]*n*n + C[p][4]*m*m;
		cbm[i][1][3] = m*n*(C[p][0]*n*n - C[p][3]*m*m + (C[p][1] + 2*C[p][6])*(m*m - n*n));
		cbm[i][2][2] = C[p][5];
		cbm[i][2][3] = m*n*(C[p][2] - C[p][4]);
		cbm[i][3][3] = (C[p][0] + C[p][3] - 2*C[p][1])*pow(m*n,2) + C[p][6]*pow((m*m - n*n),2);

		for(j = 1; j<4 ; j++)
			for(k = 0; k<j ; k++)
				cbm[i][j][k] = cbm[i][k][j];  //stacks symmetric terms

		et[i][0] = (E[p][7]*m*m + E[p][8]*n*n)*dT;   //X       thermal strain
		et[i][1] = (E[p][7]*n*n + E[p][8]*m*m)*dT;   //THETA   thermal strain
		et[i][2] = E[p][9]*dT;                      //R       thermal strain
		et[i][3] = 2*m*n*(E[p][7] - E[p][8])*dT;     //X-THETA thermal strain


	}

}

void stress(double cbm[][4][4],double et[][4],double L[],double bc[][3],double sc[][4][5])
{
 //stress coefficients

	short i,j;
	double zz=0;
	int a;
```

```
for(i=0 ; i<K ; i++)
{
      a=0;
      L[i] = sqrt(fabs(cbm[i][1][1] / cbm[i][2][2]));
//    cout <<"\nL["<<i<<"]\t"<<L[i];
      if(L[i]==1.0)
            a = 1;
      if(L[i]==2.0)
            a=2;

switch(a)
      {
      case 1:  //Isotropic and Transversely Isotropic Condition cbm[i][1][1] = cbm[i][2][2]
            {
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2])/(2*cbm[i][2][2]);
//Gamma
                  bc[i][1] = (cbm[i][1][3] - 2*cbm[i][2][3])/(4*cbm[i][2][2] - cbm[i][1][1]);
//Omega
                  bc[i][2] = 1/(2*cbm[i][2][2]);
            //Psi

                  zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                  zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                  bc[i][2] *= zz;

                  for(j = 0; j<4 ; j++)
                  {

                        sc[i][j][4] = - et[i][0]*cbm[i][0][j] - et[i][1]*cbm[i][1][j];
                        sc[i][j][4] -=  et[i][2]*cbm[i][2][j] + et[i][3]*cbm[i][j][3];
                        //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                        //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                        //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                        //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                        //                              :last indx=4-> constant
```

```
                }
            }
            break;
      case 2:  //Causes the b[i][1] term to blowup in default equations
            {
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2])/(3*cbm[i][2][2]);        //Gamma
                  bc[i][1] = (cbm[i][1][3] - 2*cbm[i][2][3])/(16*cbm[i][2][2]);     //Omega
                  bc[i][2] = 1/(3*cbm[i][2][2]);
//Psi

                  zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                  zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                  bc[i][2] *= zz;

                  for(j = 0; j<4 ; j++)
                  {

                        sc[i][j][4] = - et[i][0]*cbm[i][0][j] - et[i][1]*cbm[i][1][j];
                        sc[i][j][4] -=  et[i][2]*cbm[i][2][j] + et[i][3]*cbm[i][j][3];
                        //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                        //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                        //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                        //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                        //                              :last indx=4-> constant

                  }
            }
            break;
      default:  //The General Solution for the Composite Cylinder model
            {
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2]) / (cbm[i][2][2] - cbm[i][1][1]);
//Gamma
                  bc[i][1] = (cbm[i][1][3] - 2 * cbm[i][2][3]) / (4 * cbm[i][2][2] - cbm[i][1][1]);
//Omega
                  bc[i][2] = 1 / (cbm[i][2][2] - cbm[i][1][1]);
            //Psi
```

```
                        zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                        zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                        bc[i][2] *= zz;

                        for(j = 0; j<4 ; j++)
                        {
                              sc[i][j][0] = cbm[i][0][j] + bc[i][0]*(cbm[i][1][j] + cbm[i][2][j]);
                              sc[i][j][1] = cbm[i][j][3] + bc[i][1]*(cbm[i][1][j] + 2*cbm[i][2][j]);
                              sc[i][j][2] = cbm[i][1][j] + L[i] * cbm[i][2][j];
                              sc[i][j][3] = cbm[i][1][j] - L[i] * cbm[i][2][j];
                              sc[i][j][4] = bc[i][2]*(cbm[i][1][j] + cbm[i][2][j])- et[i][0]*cbm[i][0][j];
                              sc[i][j][4] += -et[i][1]*cbm[i][1][j] - et[i][2]*cbm[i][2][j] -
et[i][3]*cbm[i][j][3];
                              //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                              //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                              //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                              //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                              //                              :last indx=4-> constant
                        }
                  }
            }
      }
}


void mat_stack(double cbm[][4][4],double sc[][4][5], double L[], double bc[][3], double r[], double *load,
double A[][KK], double B[])
{
 //subroutine for stacking the components of the linear system Ax=B

      short i, j, ic;
      double ro,ri,r2, ro2, r3, ro3, r4, ro4, b1t2, b2t2;
      int a;

      for(i = 0; i<KK ; i++)                //initialize [A],{B}
      {
            B[i] = 0.0;
```

```
            for(j = 0; j<KK ; j++)
            A[i][j] = 0.0;
      }

//next two loops stack elements associated with the integrated B.C.
      b1t2 = 0;
      b2t2 = 0;


      for(i = 0; i<K ;i++)
      {
            L[i] = sqrt(fabs(cbm[i][1][1] / cbm[i][2][2]));
            a=0;
      //set up terms for switching for the degenerate cases
            if(L[i]==1.0)
                  a = 1;
            if(L[i]==2.0)
                  a=2;

            //define radii terms for ease of programming and debugging
            ri = r[i];
            ro = r[i+1];
            r2 = pow(r[i],2);
            ro2 = pow(r[i+1],2);
            ro3 = pow(r[i+1],3);
            r3 = pow(r[i],3);
            ro4 = pow(r[i+1],4);
            r4 = pow(r[i],4);

            switch(a)
            {
            case 1:
                  {
                        A[0][0] += 0.5*(ro2-r2)*(cbm[i][0][0]);
                        A[0][0] += 0.5*((log(ro)-0.5)*ro2-(log(ri)-0.5)*r2)*bc[i][0]*cbm[i][0][1];
                        A[0][0] += 0.5*((log(ro)+0.5)*ro2-(log(ri)+0.5)*r2)*bc[i][0]*cbm[i][0][2];

                        A[0][1] += (cbm[i][0][3]+bc[i][1]*(cbm[i][0][1]+2*cbm[i][0][2]))*(ro3-r3)/3.0;
```

```
                        A[1][0] += (cbm[i][0][3]*(ro3-r3))/3.0;
                        A[1][0] += (((1.0/3.0)-log(ri))*cbm[i][1][3]+((-2.0/3.0)-
log(ri))*cbm[i][2][3])*bc[i][0]*r3/3.0;
                        A[1][0] += ((log(ro)-
(1.0/3.0))*cbm[i][1][3]+((2.0/3.0)+log(ro))*cbm[i][2][3])*bc[i][0]*ro3/3.0;
                        A[1][1] += (cbm[i][3][3]+bc[i][1]*(cbm[i][1][3]+2*cbm[i][2][3]))*(ro4-r4)/4.0;


                        b1t2 += 0.5*((log(ro)-0.5)*ro2-(log(ri)-0.5)*r2)*cbm[i][0][1]*bc[i][2];
                        b1t2 += 0.5*((log(ro)+0.5)*ro2-(log(ri)+0.5)*r2)*cbm[i][0][2]*bc[i][2];
                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;

                        b2t2 += ((3.0*log(ro)-
1.0)*cbm[i][1][3]+(3.0*log(ro)+2.0)*cbm[i][2][3])*ro3*bc[i][2]/9.0;
                        b2t2 -= ((3.0*log(ri)-
1.0)*cbm[i][1][3]+(3.0*log(ri)+2.0)*cbm[i][2][3])*r3*bc[i][2]/9.0;
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            case 2:
                  {
                        A[0][0] += (cbm[i][0][0]+bc[i][0]*(cbm[i][0][1]+cbm[i][0][2]))*(ro2-r2)/2.0;
                        A[0][1] += (((4.0*log(ro)-7.0/3.0)*ro3-(4.0*log(ri)-
7.0/3.0)*r3)*cbm[i][0][1]*bc[i][1])/3.0;
                        A[0][1] += (((4.0*log(ro)-1.0/3.0)*ro3-(4.0*log(ri)-
1.0/3.0)*r3)*2*cbm[i][0][2]*bc[i][1])/3.0;
                        A[0][1] += (ro3-r3)*cbm[i][0][3]/3.0;

                        A[1][0] += (cbm[i][0][3]+bc[i][0]*(cbm[i][1][3]+cbm[i][2][3]))*(ro3-r3)/3.0;
                        A[1][1] += (ro4-r4)*cbm[i][3][3]/4.0+((log(ro)-
0.5)*cbm[i][1][3]+2*log(ro)*cbm[i][2][3])*bc[i][1]*ro4;
                        A[1][1] -= ((0.5-log(ri))*cbm[i][1][3]-2*log(ri)*cbm[i][2][3])*bc[i][1]*r4;


                        b1t2 += (cbm[i][0][1]+cbm[i][0][2])*(ro2-r2)*(bc[i][2])/2.0;
                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;
```

```
                        b2t2 += (cbm[i][1][3]+cbm[i][2][3])*(ro3-r3)*bc[i][2]/3.0;
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            default:
                  {//A[0][i] = Fx Conditions  A[1][i] = Tx Conditions
                        A[0][0] += (sc[i][0][0]*( ro2 - r2 )) / 2.0;                              //e terms
                        A[1][0] += (sc[i][3][0]*( ro3 - r3 )) / 3.0;
                        A[0][1] += (sc[i][0][1]*( ro3 - r3 )) / 3.0;                              //g terms
                        A[1][1] += (sc[i][3][1]*( ro4 - r4 )) / 4.0;

                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;                                 //thermal
terms
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            }
      }

      ic = 0;
      for(i = 2; i<(KK-1) ; i += 2)
      {
            ic++;


      //set up terms for switching for the degenerate cases
            a=0;
            if(L[ic-1]==1.0)
                  a = 1;
            if(L[ic-1]==2.0)
                  a = 2;

            //define radii terms for ease of programming and debugging

            ri = r[ic-1];
            ro = r[ic];
            r2 = pow(r[ic-1],2);
            ro2 = pow(r[ic],2);
```

```
            ro3 = pow(r[ic],3);
            r3 = pow(r[ic-1],3);
            ro4 = pow(r[ic],4);
            r4 = pow(r[ic-1],4);

            switch(a)
            {
            case 1:
                  {
                        A[0][i] = (cbm[ic-1][0][1]+cbm[ic-1][0][2])*(r2-ro2)/2.0;
                        A[0][i+1] = (log(ro)-log(ri))*(cbm[ic-1][0][1]-cbm[ic-1][0][2]);//check 2*cbm[ic-
1][0][2]
                        A[1][i] = (ro3-r3)*(cbm[ic-1][1][3]+cbm[ic-1][2][3])/3.0;
                        A[1][i+1] = (ro-ri)*(cbm[ic-1][1][3]-cbm[ic-1][2][3]);

                  }
                  break;
            case 2:
                  {
                        A[0][i] = (cbm[ic-1][0][1]+2*cbm[ic-1][0][2])*(ro3-r3)/3.0;
                        A[0][i+1] = (2.0*cbm[i][0][2]-cbm[i][0][1])*((1.0/ro) - (1.0/ri));
                        A[1][i] = (cbm[ic-1][1][3]+2*cbm[ic-1][2][3])*(ro4-r4)/4;
                        A[1][i+1] = (cbm[ic-1][1][3]-2*cbm[ic-1][2][3])*(log(ro)-log(ri));
                  }
                  break;
            default:
                  {
                        A[0][i] = sc[ic-1][0][2]*(pow(ro, 1+L[ic-1]) - pow(ri, 1+L[ic-1]) ) / (1+L[ic-1]);
                        A[0][i+1] = sc[ic-1][0][3]*(pow(ro, 1-L[ic-1]) - pow(ri, 1-L[ic-1]) ) / (1-L[ic-
1]);
                        A[1][i] = sc[ic-1][3][2]*(pow(ro, 2+L[ic-1]) - pow(ri, 2+L[ic-1]) ) / (2+L[ic-1]);
                        A[1][i+1] = sc[ic-1][3][3]*(pow(ro, 2-L[ic-1]) - pow(ri, 2-L[ic-1]) ) / (2-L[ic-
1]);
                  }
                  break;
            }
      }
      //Applied Loads
```

```
      B[0] = (load[2] / (2 * Pi)) - b1t2;
      B[1] = (load[3] / (2 * Pi)) - b2t2;

 //next loop stacks the elements associated with the interface conditions w[i](r) = w[i+1](r) and sigma-r

      ic = 0;
      for(i = 2; i<(KK-3) ; i += 2)
      {

            //set up terms for switching for the degenerate cases
            a=0;
            if(L[ic]==1.0)
                  a = 1;
            if(L[ic]==2.0)
                  a=2;

            ic++;

            switch(a)
            {

            case 1:
                  {
                        //The sigma r conditions
                        A[i][0] = (cbm[ic-1][0][2]+bc[ic-1][0]*(cbm[ic-1][1][2]*log(r[ic])+cbm[ic-
1][2][2]*(log(r[ic])+1)));
                        A[i][0] -
=(cbm[ic][0][2]+bc[ic][0]*(cbm[ic][1][2]*log(r[ic])+cbm[ic][2][2]*(log(r[ic])+1)));
                        A[i][1] = ((cbm[ic-1][1][2]+2*cbm[ic-1][2][2])*bc[ic-1][1]+cbm[ic-1][2][3])*r[ic];
                        A[i][1] -=((cbm[ic][1][2]+2*cbm[ic][2][2])*bc[ic][1]+cbm[ic][2][3])*r[ic];

                        A[i][i] =(cbm[ic-1][1][2]+cbm[ic-1][2][2]) ;
                        A[i][i+1] = (cbm[ic-1][1][2]-cbm[ic-1][2][2])*pow(r[ic],-2);//check2*cbm[ic-
1][2][2]
                        A[i][i+2] = -(cbm[ic][1][2]+cbm[ic][2][2]);
                        A[i][i+3] = -(cbm[ic][1][2]-cbm[ic][2][2])*pow(r[ic],-2);//check2*cbm[ic-1][2][2]

                        //The w(r) conditions
```

```
                        A[i+1][0] =(bc[ic-1][0]-bc[ic][0])*r[ic]*log(r[ic]);
                        A[i+1][1] =(bc[ic-1][1]-bc[ic][1])*r[ic]*r[ic];


                        A[i+1][i] = r[ic];
                        A[i+1][i+1] = 1.0/r[ic];
                        A[i+1][i+2] = -r[ic];
                        A[i+1][i+3] = -1.0/r[ic];

                        //The thermal contributions
                        B[i] = (cbm[ic][1][2]*log(r[ic])+cbm[ic][2][2]*(log(r[ic])+1))*bc[ic][2];
                        B[i] -= (cbm[ic-1][1][2]*log(r[ic])+cbm[ic-1][2][2]*(log(r[ic])+1))*bc[ic-1][2];
                        B[i] += sc[ic][2][4] - sc[ic-1][2][4];
                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic]*log(r[ic]);
                  }
                  break;

            case 2:
                  {
                        //The sigma r conditions
                        A[i][0] = (cbm[ic-1][0][2]+bc[ic-1][0]*(cbm[ic-1][1][2]+cbm[ic-1][2][2]));
                        A[i][0] -=(cbm[ic][0][2]+bc[ic][0]*(cbm[ic][1][2]+cbm[ic][2][2]));
                        A[i][1] = (cbm[ic-1][2][3]+bc[ic-1][1]*(cbm[ic-1][1][2]*(4.0*log(r[ic])-
1)+2.0*cbm[ic-1][2][2]*(4.0*log(r[ic])+1.0)))*r[ic];
                        A[i][1] -=(cbm[ic][2][3]+bc[ic][1]*(cbm[ic][1][2]*(4.0*log(r[ic])-
1)+2.0*cbm[ic][2][2]*(4.0*log(r[ic])+1.0)))*r[ic];

                        A[i][i] =(cbm[ic-1][1][2]+2*cbm[ic-1][2][2])*r[ic] ;
                        A[i][i+1] = (cbm[ic-1][1][2]-2*cbm[ic-1][2][2])*pow(r[ic],-3);
                        A[i][i+2] = -(cbm[ic][1][2]+2*cbm[ic][2][2])*r[ic];
                        A[i][i+3] = -(cbm[ic][1][2]-2*cbm[ic][2][2])*pow(r[ic],-3);

                        //The w(r) conditions
                        A[i+1][0] =(bc[ic-1][0]-bc[ic][0])*r[ic]*log(r[ic]);
                        A[i+1][1] =(bc[ic-1][1]-bc[ic][1])*r[ic]*r[ic];


                        A[i+1][i] = r[ic];
```

```
                        A[i+1][i+1] = 1.0/r[ic];
                        A[i+1][i+2] = -r[ic];
                        A[i+1][i+3] = -1.0/r[ic];

                        //The thermal contributions
                        B[i] = (cbm[ic][1][2] + cbm[ic][2][2])*bc[ic][2]-(cbm[ic-1][1][2] + cbm[ic-
1][2][2])*bc[ic-1][2];
                        B[i] += sc[ic][2][4] - sc[ic-1][2][4];

                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic];

                  }
                  break;

            default:
                  {

                        A[i][0] = sc[ic-1][2][0] - sc[ic][2][0];                    //eo terms
                        A[i][1] = ( sc[ic-1][2][1] - sc[ic][2][1] ) * r[ic];  //go terms

                        A[i + 1][0] = ( bc[ic-1][0] - bc[ic][0] ) * r[ic];
                        A[i + 1][1] = ( bc[ic-1][1] - bc[ic][1] ) * r[ic]*r[ic];

                        A[i][i] = sc[ic-1][2][2] * pow(r[ic], (L[ic-1] - 1) );
                        A[i][i+1] = sc[ic-1][2][3] * pow(r[ic], (-L[ic-1] - 1) );
                        A[i][i+2] = -sc[ic][2][2] * pow(r[ic], (L[ic] - 1) );
                        A[i][i+3] = -sc[ic][2][3] * pow(r[ic], (-L[ic] - 1) );

                        A[i+1][i] = pow(r[ic], L[ic-1] );
                        A[i+1][i+1] = pow(r[ic], -L[ic-1] );
                        A[i+1][i+2] = -pow(r[ic], L[ic] );
                        A[i+1][i+3] = -pow(r[ic], -L[ic] );

                        B[i] = sc[ic][2][4] - sc[ic-1][2][4];
                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic];

                  }
                  break;
```

```
            }
      }

 //next lines stack the matrix elements associated with the pressure B.C.

      //The inner surface pressure conditions
            a=0;
            //set up terms for switching for the degenerate cases
            if(L[0]==1.0)
                  a = 1;
            if(L[0]==2.0)
                  a=2;
            switch(a)
            {
            case 1:
                  {
                        A[KK-2][0] =
(cbm[0][0][2]+bc[0][0]*(cbm[0][1][2]*log(r[0])+cbm[0][2][2]*(log(r[0])+1)));
                        A[KK-2][1] = ((cbm[0][1][2]+2*cbm[0][2][2])*bc[0][1]+cbm[0][2][3])*r[0];
                        A[KK-2][2] = (cbm[0][1][2]+cbm[0][2][2]);
                        A[KK-2][3] = (cbm[0][1][2]-cbm[0][2][2])*pow(r[0],-2);

                        B[KK-2] = -load[1] - (cbm[0][1][2]*log(r[0])+cbm[0][2][2]*(log(r[0])+1))*bc[0][2] -
sc[0][2][4];
                  }
                  break;
            case 2:
                  {
                        A[KK-2][0] = (cbm[0][0][2]+bc[0][0]*(cbm[0][1][2]+cbm[0][2][2]));
                        A[KK-2][1] = (cbm[0][1][2]*(4*log(r[0])-
1)+2*cbm[0][2][2]*(4*log(r[0])+1))*bc[0][1]*r[0];
                        A[KK-2][2] = (cbm[0][1][2]+2*cbm[0][2][2])*r[0];
                        A[KK-2][3] = (cbm[0][1][2]-2*cbm[0][2][2])*pow(r[0],-3);

                        B[KK-2] = -load[1] - (cbm[0][1][2]+cbm[0][2][2])*bc[0][2] - sc[0][2][4];
                  }
                  break;
            default:
```

```
                {
                        A[KK-2][0] = sc[0][2][0];
                        A[KK-2][1] = sc[0][2][1] * r[0];
                        A[KK-2][2] = sc[0][2][2] * pow(r[0], (L[0] - 1) );
                        A[KK-2][3] = sc[0][2][3] * pow(r[0], (-L[0] - 1) );

                        B[KK-2] = -load[1] - sc[0][2][4];
                }
                break;
        }
    //The outer surface pressure conditions
                a=0;
        //set up terms for switching for the degenerate cases
        if(L[K-1]==1.0)
                a = 1;
        if(L[K-1]==2.0)
                a=2;
        switch(a)
        {
        case 1:
                {
                        A[KK-2][0] = (cbm[K-1][0][2]+bc[K-1][0]*(cbm[K-1][1][2]*log(r[K])+cbm[K-
1][2][2]*(log(r[K])+1)));
                        A[KK-2][1] = ((cbm[K-1][1][2]+2*cbm[K-1][2][2])*bc[K-1][1]+cbm[K-1][2][3])*r[K];
                        A[KK-2][2] = (cbm[K-1][1][2]+cbm[K-1][2][2]);
                        A[KK-2][3] = (cbm[K-1][1][2]-cbm[K-1][2][2])*pow(r[K],-2);

                        B[KK-2] = -load[0] - (cbm[K-1][1][2]*log(r[K])+cbm[K-1][2][2]*(log(r[K])+1))*bc[K-
1][2]- sc[K-1][2][4];
                }
                break;
        case 2:
                {
                        A[KK-2][0] = (cbm[K-1][0][2]+bc[K-1][0]*(cbm[K-1][1][2]+cbm[K-1][2][2]));
                        A[KK-2][1] = (cbm[K-1][1][2]*(4*log(r[K])-1)+2*cbm[K-
1][2][2]*(4*log(r[K])+1))*bc[K-1][1]*r[K];
                        A[KK-2][2] = (cbm[K-1][1][2]+2*cbm[K-1][2][2])*r[K];
                        A[KK-2][3] = (cbm[K-1][1][2]-2*cbm[K-1][2][2])*pow(r[K],-3);
```

```
                        B[KK-2] = -load[0] - (cbm[K-1][1][2]+cbm[K-1][2][2])*bc[K-1][2]- sc[K-1][2][4];
                  }
                  break;
            default:
                  {
                        A[KK-1][0] = sc[K-1][2][0];
                        A[KK-1][1] = sc[K-1][2][1] * r[K];
                        A[KK-1][KK-2] = sc[K-1][2][2] * pow(r[K], ( L[K-1] - 1) );
                        A[KK-1][KK-1] = sc[K-1][2][3] * pow(r[K], ( -L[K-1] - 1) );

                        B[KK-1] = -load[0] - sc[K-1][2][4];
                  }
                  break;
            }
}
void output(double x[], double r[], double L[], double bc[][3], double Calc_Strain[], char strainposition)
{
      double ex=0.0, eo=0.0, gxo=0.0, w=0.0, Rc;
      int a, i;


      for (int ab=0;ab<3;ab++)
      {
            Calc_Strain[ab]=0.0;

      }

      //Inner or Outer Surface Strains?
      if(strainposition =='i'||strainposition=='I')
      {
            i=0;
            Rc=r[0];

      }
      else  //Outer Surface is the Default condition
      {
            Rc=r[K];
```

```
            i=K-1;

      }

      a=0;
      if(L[i]==1.0)
            a = 1;
      if(L[i]==2.0)
            a=2;
      switch (a)
      {
            case 1:           //Isotropic or Transversely Isotropic
                  {
                        w = bc[i][0]*Rc*log(Rc)*x[0] + bc[i][1]*x[1]*Rc*Rc + x[2*(i)+2]*Rc + x[2*(i)+3]/Rc
+ bc[i][2]*log(Rc);
                        ex = x[0];        // Ex: Epsilon x
                        eo = w/Rc;        // Eo: Epsilon theta
                        gxo = x[1]*Rc;    // Gxo: Gamma x-theta
                  }
                  break;
            case 2:           //bc[i][1] = 0
                  {
                        w = bc[i][0]*Rc*x[0] + bc[i][1]*x[1]*pow(Rc,2)*(4*log(Rc)-1) + x[2*(i)+2]*pow(Rc,
L[i]) + x[2*(i)+3]*pow(Rc, -L[i]) + bc[i][2]*Rc;
                        ex = x[0];        // Ex: Epsilon x
                        eo = w/Rc;        // Eo: Epsilon theta
                        gxo = x[1]*Rc;    // Gxo: Gamma x-theta
                  }
                  break;
            default:    //General Case
                  {
                        w = bc[i][0]*Rc*x[0] + bc[i][1]*x[1]*Rc*Rc + x[2*(i)+2]*pow(Rc, L[i]) +
x[2*(i)+3]*pow(Rc, -L[i]) + bc[i][2]*Rc;
                        ex = x[0];        // Ex: Epsilon x
                        eo = w/Rc;        // Eo: Epsilon theta
                        gxo = x[1]*Rc;    // Gxo: Gamma x-theta
                  }
                  break;
```

```
      }
            Calc_Strain[0] = ex; //Epsilon X
            Calc_Strain[1] = eo; //Epsilon Y
            Calc_Strain[2] = gxo; //Epsilon XY
}
```

9.4 Elastic Solution.h

```
//Elastic_Solution.h

#include "Inversion.h"
#include "matrix.h"
#include <iostream.h>
#include <fstream.h>
/*****************************/
/*routines internal to CCM*/
/*****************************/
void Elastic_Solution(double C[][7], double E[][10], int matnum, int mat[], double r[], double theta[],
double *load, double Calc_Strain[], char strainposition);
void Cbar(double C[][7], double theta[], double E[][10], int matnum, int mat[], double cbm[][4][4], double
et[][4]);
void stress(double cbm[][4][4], double et[][4], double L[], double bc[][3], double sc[][4][5]);
void mat_stack(double cbm[][4][4], double sc[][4][5], double L[], double bc[][3], double r[], double *load,
double A[][KK], double B[]);
void output(double x[], double r[], double L[], double bc[][3], double Calc_Strain[], char strainposition);
```

9.5 Data_Input.cpp

```
//Data_Input.cpp
```

```
#include "Data_Input.h"

ifstream inputfile, loadfile,expstrain;         //Data file containing input data (material properties,
geometry, loads)
ofstream tracking;                        //Data files for data output and intermediate values

int Input(double E[][10], double C[][7], int mat[], double radii[], double theta[], double Loads[][4],
double Exp_Strain[][3], char position[])
{
      inputfile.open("ICEinput.dat");     // Input.dat contains initial guess E's, and geometry
      loadfile.open("ICELOADS.dat");            // loads.dat contains all loading conditions (must alter
numloads value in Inversion.h)
      expstrain.open("ICEstrain.dat");// expstrain.dat contains all the measured strain values (ex, ey,
gxy) for each load condition (numload # of sets)
      tracking.open("tracking.dat");      // Opening Data files for output and tracking of values


      double temp;
      char datatype;
      int i,j,matval, matnum;

//    First input value must be the number of material layers

      inputfile>>matnum;

      for(matval=0;matval<matnum;matval++)
      {
            inputfile>>datatype;

//If the first value in the input file is 'C' then reads in Cij values
            if((datatype == 'C')||(datatype=='c'))
            {
                  for(i=0;i<7;i++)
                        inputfile>>C[matval][i];
                  for(i=7;i<10;i++)
                        inputfile>>E[matval][i];  //reads in the thermal coefficients
                  C_to_E(&(C[matval][0]),&(E[matval][0]));
            }
```

```
//If 'E' - Reads in Elastic Moduli (E1,E2,E3,G12,nu12,nu13,nu23,a1,a2,a3) IN ORDER!!!
            if((datatype == 'E')||(datatype=='e'))
            {
                  for(i=0; i<10; i++)
                        inputfile >> E[matval][i];
                  CMatrix(E,C,matnum);          //Calculates the Cij values
            }
      }

//Reads in all surface radii
      for(j=0; j<K+1; j++)
            inputfile >> radii[j];


//Reads in each ply angle (degrees) and converts to radians
      for(i=0; i<K; i++)
      {
            inputfile >> temp;
            theta[i] = temp * (Pi/180.0);
            inputfile >> mat[i];
      }

//Reads in all load and experimental strain values
      for (i=0; i<numloads; i++)                    // !!!!LOAD CONDITIONS!!!!               !!!STRAIN
CONDITIONS!!!
      {
            expstrain>>position[i];                 // position[i] = i or o for inner or outer surface
strains
            for(j=0; j<4; j++)                            // Loads[0] = External Pressure
      exp_strain[][0] = e-x
                  loadfile >> Loads[i][j];          // Loads[1] = Internal Pressure
      exp_strain[][1] = e-y
            for(int k=0;k<3;k++)                    // Loads[2] = Axial Load
      exp_strain[][2] = gamma-xy
                  expstrain>> Exp_Strain[i][k]; // Loads[3] = Torsional Load
      }
```

```
//Writes all data and stiffness values to the tracking file
      Output_data(E,matnum, mat,radii,theta,Loads, Exp_Strain, position, C);
      inputfile.close();
      loadfile.close();                                          //Close the data files
      expstrain.close();
      tracking.close();

      cout << endl;
      return matnum;
}
//-------------------------------------------------------------------------------------------
void Output_data(double E[][10],int matnum, int mat[], double radii[], double theta[], double Loads[][4],
double Exp_Strain[][3], char position[], double C[][7])
{

      tracking << "Tube properties entered:" << endl;

//Prints Elastic Moduli (E1,E2,E3,G12,nu12,nu13,nu23,a1,a2,a3)
      for(int i=0;i<matnum;i++)
      {
            tracking << "\nMaterial #"<<i+1;
            tracking << "\nE1 =\t"<<E[i][0];
            tracking << "\nE2 =\t"<<E[i][1];
            tracking << "\nE3 =\t"<<E[i][2];
            tracking << "\nG12 =\t"<<E[i][3];
            tracking << "\nnu12 =\t"<<E[i][4];
            tracking << "\nnu13 =\t"<<E[i][5];
            tracking << "\nnu23 =\t"<<E[i][6];
            tracking << "\nalpha-1 =\t"<<E[i][7];
            tracking << "\nalpha-2 =\t"<<E[i][8];
            tracking << "\nalpha-3 =\t"<<E[i][9]<<endl<<endl;
      }

      tracking <<"\n\nTube geometry entered:\n";

      for(i=0; i<K; i++)
      {
```

```
            tracking << "Ri and Ro of ply #" << i+1<< "\t"<<radii[i]<<"\t\t"<<radii[i+1]<<"\t\tAngle"<<
"\t"<< theta[i]*(180/Pi)<<"\tMaterial #\t"<<mat[i]+1<<endl;
      }
      for(int ii=0;ii<numloads;ii++)
      {
// Printing out the Load conditions to the tracking file
            tracking << "\nLoad Set #"<< ii+1<<endl;
            tracking << "\nLoading Conditions and Measured Strains:" << endl;
            tracking << "Axial Load = \t" << Loads[ii][2] << endl;
            tracking << "Torque = \t" << Loads[ii][3] << endl;
            tracking << "Internal Pressure = \t" << Loads[ii][1] << endl;
            tracking << "External Pressure = \t" << Loads[ii][0] << endl;

//Printing out the Measured Strain values for each applied load condition

            tracking << "\nSurface = ";
            if (position[ii]=='i'||position[ii]=='I')
                  tracking << "\tInner Surface Strains";
            else if (position[ii]=='o'||position[ii]=='O')
                  tracking << "\tOuter Surface Strains";
            else
                  tracking << "\tINVALID Response!";
            tracking << "\nAxial Strain (ex) =\t" << Exp_Strain[ii][0] << endl;
            tracking << "Hoop Strain (ey) =\t" << Exp_Strain[ii][1] << endl;
            tracking << "Shear Strain (gxy) =\t" << Exp_Strain[ii][2] << endl;
      }
      tracking << "\n\nInitial Cij Values:" << endl;
      for(i=0;i<matnum;i++)
      {
            tracking << "\nMaterial #"<<i+1;
            tracking << "\nC11 = \t"<<C[i][0];
            tracking << "\nC12 = \t"<<C[i][1];
            tracking << "\nC13 = \t"<<C[i][2];
            tracking << "\nC22 = \t"<<C[i][3];
            tracking << "\nC23 = \t"<<C[i][4];
            tracking << "\nC33 = \t"<<C[i][5];
            tracking << "\nC66 = \t"<<C[i][6]<<endl<<endl;
```

```
      tracking << endl;
      }
}

// Calcualtes the C Matrix Values from the input data
//---------------------------------------------------------------------------
void CMatrix(double E[][10], double C[][7], int matnum) //c matrix subroutine
{
      double v;

      for(int i=0;i<matnum;i++)
      {
            v = (1 - E[i][4] *(E[i][4] * E[i][1] / E[i][0] + 2 * E[i][6] * E[i][5] * E[i][2] / E[i][0]) -
E[i][5] * E[i][5] * E[i][2] / E[i][0] - E[i][6] * E[i][6] * E[i][2] / E[i][1]);

            C[i][0] = (1 - E[i][6] * E[i][6] * E[i][2] / E[i][1]) * E[i][0] / v;        // c11
            C[i][1] = (E[i][4] + E[i][5] * E[i][6] * E[i][2] / E[i][1]) * E[i][1] / v;     // c12
            C[i][2] = (E[i][5] + E[i][4] * E[i][6]) * E[i][2] / v;                    // c13
            C[i][3] = (1 - E[i][5] * E[i][5] * E[i][2] / E[i][0]) * E[i][1] / v;        // c22
            C[i][4] = (E[i][6] + E[i][4] * E[i][5] * E[i][1] / E[i][0]) * E[i][2] / v;     // c23
            C[i][5] = (1 - E[i][4] * E[i][4] * E[i][1] / E[i][0]) * E[i][2] / v;        // c33
            C[i][6] = E[i][3];                                                  // c66

            v=0.0;
      }
}
```

9.6 Data_Input.h

```
// Data Input.h


#include <iostream.h>
#include <iomanip.h>
#include "Inversion.h"
#include <fstream.h>
```

```
int Input(double E[][10], double C[][7], int mat[], double radii[], double theta[], double Loads[][4],
double Exp_Strain[][3], char position[]);
void Output_data(double E[][10], int matnum, int mat[], double radii[], double theta[], double Loads[][4],
double Exp_Strain[][3], char position[], double C[][7]);
void CMatrix(double E[][10], double C[][7], int matval);    //c matrix subroutine
```

9.7 Matrix.cpp

```
#include "Matrix.h"
#include <math.h>
#include <stdlib.h>
#include <iostream.h>
//Matrix.cpp


void SysScale(double *A,double *B,int n)


int LUDecomposition (double *A,int n,int *indx,int d)

void LUBackSub (double *A,int n,int *indx,double *B)

long Gauss(double *A,double *B,double *X,int n)
```

9.8 Matrix.h

```
//Matrix.h

#include <math.h>
#include "Inversion.h"
```

The following subroutines have been omitted due to copyright considerations. The appropriate subroutines can be found in Numerical Recipes (sections on LU Decomposition and Backsubstitution).

```
void SysScale(double *A,double *B,int n);
int LUDecomposition (double *A,int n,int *indx,int d);
void LUBackSub (double *A,int n,int *indx,double *B);
long Gauss(double *A,double *B,double *X,int n);
```

9.9 Jacobian.cpp

```
#include <fstream.h>
#include "jacobian.h"
#include "Elastic_Solution.h"

//Jacobian.cpp

void Jacobian(double E[][10],double C[][7], int matnum, int mat[], double r[],double th[],double
load[][4],double Exp_Strain[][3], char position[], double jaco[][NAP])
{
      int i,j;
      double h=0.00001;
      double Ctemp[5][7];
      double PHIminus[ND], PHIplus[ND];

      for(i=0;i<matnum;i++)
      {
            for(j=0; j<7; j++)
                  Ctemp[i][j] = C[i][j];
      }
```

```
      for(i=0; i<NAP; i++)
      {
            Ctemp[0][i] = C[0][i]*(1-h);
            PhiSub(E, Ctemp, matnum, mat, r, th, load, Exp_Strain, position, PHIminus);

            Ctemp[0][i] = C[0][i]*(1+h);

            PhiSub(E, Ctemp, matnum, mat, r, th, load, Exp_Strain, position, PHIplus);


            Ctemp[0][i] = C[0][i];

            for(j=0; j<ND; j++)
                  jaco[j][i] = (PHIplus[j] - PHIminus[j])/(2 * h * C[0][i]);
   }

}

void PhiSub(double E[][10],double C[][7], int matnum, int mat[], double r[],double th[],double
load[][4],double Exp_Strain[][3], char position[], double PHI[])
{
      double Calc_Strain[3];

      for(int i=0 ; i<numloads ; i++)
      {

            Elastic_Solution(C, E, matnum, mat, r, th, &(load[i][0]), Calc_Strain, position[i]);

            PHI[i*3] = Exp_Strain[i][0] - Calc_Strain[0];            // Ex
            PHI[i*3+1] = Exp_Strain[i][1] - Calc_Strain[1];          // Ey
            PHI[i*3+2] = Exp_Strain[i][2] - Calc_Strain[2];             // Gxy
      }

}
```

9.10 Jacobian.h

```
//Jacobian.h

#include "Inversion.h"  //for the parameters NAP,ND,etcÉ

void Jacobian(double E[][10],double C[][7],int matnum, int mat[], double r[],double th[],double
eload[][4],double Exp_Strain[][3], char position[], double jaco[][NAP]);
void PhiSub(double E[][10],double C[][7], int matnum, int mat[], double r[],double th[],double
eload[][4],double Exp_Strain[][3], char position[], double PHI[ND]);
```

9.11 Input Files

9.11.1 A typical load program

Po	Pi	Fx	Tx
0	400	0	0
0	0	9900	0
0	0	0	3700

9.11.2 A typical strain file

Surface	ex	eq	gxq
o	-2.692E-05	0.000128074	1.26742E-05
o	0.000460103	-0.000128363	-1.03551E-05
o	-1.76911E-06	-1.32018E-06	0.000419356

9.11.3 A typical input file

```
1                       Number of different materials
e                       Type of material data (c for Cij, e for Ei)
3.67463e+007            Ei or Cij Values
3.67463e+007
2.40486e+007
5.3941e+006
0.24
0.24
0.24
0
0                       CTE’s
0
0.979084646
0.990834146
1.002583646
1.014333146
1.026082646
1.037832146
1.049581646
1.061331146
1.073080646             Interfacial radii
1.084830146
1.096579646
1.108329146
1.120078646
1.131828146
1.143577646
1.155327146
1.167076646
```

```
45    0
-45   0
45    0                  Individual ply orientations and ply material type (dependant upon # of mat’ls)
-45   0
45    0
-45   0
45    0
-45   0
45    0
-45   0
45    0
-45   0
45    0
-45   0
```

10 Appendix C. Levenberg-Marquardt Inversion – E_i

Here is the source code for the Inversion code using the Levenberg-Marquardt compromise to optimize the E_i values. Included are the C source codes for all the files. Files included:

- Inversion.cpp
- Inversion.h
- Elastic_Solution.cpp
- Elastic_Solution.h
- Data_Input.cpp
- Data_Input.h
- Matrix.cpp
- Matrix.h
- Jacobian.cpp
- Jacobian.h

10.1 Inversion.cpp

```
// Inversion Program -> Inversion.cpp

#include <stdlib.h>
#include <iostream.h>
#include <iomanip.h>

#include "Data_Input.h"
#include "Elastic_Solution.h"
#include "Inversion.h"
#include "jacobian.h"
```

```
#include <time.h>

void main()
{
      int count, convergence, i;
      int matnum, mat[K], P[7];
      double E[5][10], PHI[ND], SSE, B[NAP], Ei[MaxIteration+1][7];
      double radii[K+1], theta[K], Loads[numloads][4], Exp_Strain[numloads][3], Calculated_Strain[3];
      double SSEmin=1.0, *del, delta, duration, C_Strain[numloads][3];
      char position[numloads];
      ofstream  EiFile;
      clock_t start, finish;
      ofstream  SSEFile;

      SSEFile.open("SSE.dat");
      EiFile.open("Eioutput.dat");

      delta= 0.0001;
      del=&delta;

      start = clock();

// STEP 1.  Read in all data

      matnum = Input(E, P, mat, radii, theta, Loads, Exp_Strain, position);


/*****************************************************************************/


// STEP 2.  Show initial results


      EiFile << "Initial Values:"<<endl;

      EiFile << endl<< "E Values:"<< endl;
      EiFile << "\nE1"<< "\t"<< E[0][0]<<"\t";
      EiFile << "\nE2"<< "\t"<< E[0][1]<<"\t";
```

```
      EiFile << "\nE3"<< "\t"<< E[0][2]<<"\t";
      EiFile << "\nG12"<< "\t"<< E[0][3]<<"\t";
      EiFile << "\nnu12"<< "\t"<< E[0][4]<<"\t";
      EiFile << "\nnu13"<< "\t"<< E[0][5]<<"\t";
      EiFile << "\nnu23"<< "\t"<< E[0][6]<<"\t";

      Calc_Increment(E, P, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI, B, &SSE,del);

      cout << endl << "\nInitial SSE = " <<SSE;
      cout << endl << "Acceptable Error Value = "<< Acceptable << endl;
      EiFile << endl << "\nInitial SSE = " <<SSE;
      SSEFile <<"Initial SSE for Guess Values\t"<<SSE<<endl;
      SSEFile <<"\n0\t"<< SSE<<"\t"<<*del;


// STEP 3.  Minimize the error function and return information on the type of solution reached

      convergence = Minimize(E, P, matnum, mat, Ei, radii,  theta, Loads,Exp_Strain, position, PHI, B,
&SSE,&count, SSEFile,del);
/*
      if (convergence==4)
      {
            *del=0;
            convergence = Minimize(E, C, matnum, mat, Cij, radii,  theta, Loads,Exp_Strain, position, PHI,
B, &SSE,&count, SSEFile,del);
      }
*/
// STEP 4.  Give information on convergence conditions

      switch(convergence)
      {
            case 0: cout<<"\nSingular matrix encountered"<<endl;
                  break;
            case 1: cout<< endl<<"Local minimum encountered"<< endl;
                  break;
            case 2: cout<< endl<<"Exceded Maximum interations"<< endl;
                  break;
            case 3: cout<< endl<<"SSE sufficiently small"<< endl;
```

```
                  break;
            case 4: cout<<endl<<"Insignificant change to values"<< endl;
                  break;
            case 5: cout << endl<<"Did not Converge"<< endl;
                  break;
      }


// STEP 5.  Output final values
      if(convergence!=0)
      {
            cout<< "\nFinal SSE Value = " << SSE << endl;
            cout<< "\nNumber of iterations = "<< count+1<< endl;
            EiFile << "\nNumber of iterations = \t"<< count+1<< endl;
            EiFile << endl<< "Final Cij Values:"<< endl;


            EiFile  << endl<<endl<< "Final E Values:"<< endl;
            EiFile  << "\nE1"<< "\t"<< E[0][0]<<"\t";
            EiFile << "\nE2"<< "\t"<< E[0][1]<<"\t";
            EiFile << "\nE3"<< "\t"<< E[0][2]<<"\t";
            EiFile << "\nG12"<< "\t"<< E[0][3]<<"\t";
            EiFile << "\nnu12"<< "\t"<< E[0][4]<<"\t";
            EiFile << "\nnu13"<< "\t"<< E[0][5]<<"\t";
            EiFile << "\nnu23"<< "\t"<< E[0][6]<<"\t";

            EiFile <<"\nFinal SSE = \t" << SSE<< endl<<endl;

      }
      else
      {
            EiFile<<"\nSingular Matrix\n";
      }



//Calculate the strain response for the solution values and output in table for easy plotting
```

```
      EiFile<<"\nCalculated Strains for the Final Values";
      EiFile<<"\nLoads\t\t\t\t\t"<<"Measured Strains\t\t\t\t"<<"Calculated Strains\n";
      EiFile<<"Po\t"<<"Pi\t"<<"Fx\t"<<"Tx\t\t"<<"Ax\t"<<"Hoop\t"<<"Shear\t\t"<<"Ax\t"<<"Hoop\t"<<"Shear\t"<
<"\tSurface\n";

      for(i=0 ; i<numloads ; i++)
      {

            Elastic_Solution(E, matnum, mat, radii, theta, &(Loads[i][0]), Calculated_Strain, position[i]);

            for(int j=0;j<3;j++)
                  C_Strain[i][j]=Calculated_Strain[j];
            for(j=0;j<4;j++)
                  EiFile<<Loads[i][j]<<"\t";
            EiFile<<"\t";
            for(j=0;j<3;j++)
                  EiFile<<Exp_Strain[i][j]<<"\t";
            EiFile<<"\t";
            for(j=0;j<3;j++)
                  EiFile<<C_Strain[i][j]<<"\t";
            EiFile<<"\t"<<position[i]<<endl;

      }

// Calculates time used to find solution

      finish = clock();
      duration = (double)(finish - start)/CLOCKS_PER_SEC;
      if(duration>60.0)
      {
            duration=duration/60.0;
            cout << "\nCalculation Time:  \t"<<duration<< "\t minutes."<< endl;
            EiFile << "\nCalculation Time:  \t"<<duration<< "\t minutes."<< endl;

      }
      else
      {
            cout << "\nCalculation Time:  \t"<<duration<< "\tseconds."<< endl;
```

```
		EiFile << "\nCalculation Time:  \t"<<duration<< "\tseconds."<< endl;
	}

	EiFile<<"\n\nIteration"<<"\tE1"<<"\tE2"<<"\tE3"<<"\tG12"<<"\tnu12"<<"\tnu13"<<"\tnu23"<<endl;
	for(i=0;i<count+1;i++)
	{

		EiFile <<i;
		EiFile <<"\t"<< Ei[i][0];
		EiFile <<"\t"<< Ei[i][1];
		EiFile <<"\t"<< Ei[i][2];
		EiFile <<"\t"<< Ei[i][3];
		EiFile <<"\t"<< Ei[i][4];
		EiFile <<"\t"<< Ei[i][5];
		EiFile <<"\t"<< Ei[i][6]<<endl;
	}
	EiFile.close();

}



//-------------------------------------------------------------------------------------------
/*Calc_Increment calculates the error value PHI and SSE as well as the incremental step value for Cij
*/
int Calc_Increment(double E[][10], int P[], int matnum, int mat[], double radii[], double theta[], double
Loads[][4], double Exp_Strain[][3], char position[], double PHI[], double B[], double *SumSquareErr, double
*del)
{
	int indx[NAP],i,crash=0;
	double J[ND][NAP], JtJ[NAP][NAP];


	*SumSquareErr = 0.0;

	for (i=0;i<NAP;i++)
	{
		for(int jj=0; jj<NAP;jj++)
```

```
                  JtJ[i][jj] = 0.0;
      }
      for(i=0;i<ND;i++)
            PHI[i] = 0.0;


      PhiSub(E, matnum, mat, radii,theta,Loads, Exp_Strain, position, PHI);

      for(i=0;i<ND;i++)
            *SumSquareErr += pow(PHI[i],2);     //Calculate the Sum of Square Errors

      Jacobian(E, P, matnum, mat, radii, theta, Loads, Exp_Strain, position, J);

      //Calculate ([J]t[J])^(-1)[J]t{PHI}

      for(i=0;i<NAP;i++)      // [J]t[J]
      {
            for(int j=0;j<NAP;j++)
            {
                  for(int ii=0;ii<ND;ii++)
                        JtJ[i][j] += J[ii][i]*J[ii][j];
            }

      }
//Levenberg-Marquardt   (JtJ+delta*D)
      for(i=0;i<NAP;i++)
            JtJ[i][i] *=(1.0 + *del);

      for(i=0;i<NAP;i++)      //[J]t{PHI}
      {
            B[i] = 0.0;
            for (int j=0;j<ND;j++)
                  B[i] -=J[j][i]*PHI[j];
      }
      //SysScale(&(JtJ[0][0]),B,NAP);
      crash = LUDecomposition(&(JtJ[0][0]),NAP,indx,i);
      if(crash!=0)
      {
```

```
            cout <<"\nSingular Matrix"<<endl;
            *SumSquareErr = 1.0;
            return 1;
      }

      LUBackSub(&(JtJ[0][0]),NAP,indx,B);
      return 0;
}


//-------------------------------------------------------------------------------------------------
//Function to minimize error and apply convergence criteria
int Minimize(double E[][10], int P[], int matnum, int mat[], double Ei[][7], double radii[], double
theta[], double Loads[][4], double Exp_Strain[][3], char position[], double PHI[], double B[], double *sse,
int *counter, ofstream SSEFile, double *del)
{
      int crash_big, crash_small, crash_del,i,j,flag,count=0;
      double ENew[5][10], ENew2[5][10],SSEnew, SSEold, del_orig, deltabig, deltasmall,minimum;
      double SSE_del=0.0, SSE_big=0.0,SSE_small=0.0, B_big[7], B_small[7], ratio[7],max=0.0,SSE_ratio;
      double ENew_big[5][10],ENew_small[5][10],SSE_del_old=0.0, SSE_big_old=0.0,SSE_small_old=0.0;
      SSEnew = 1000000;
      SSEold = 1000000;
      flag=0;
      minimum=0.0001;
      del_orig = *del;

      cout<<"\nIteration\t"<<"SSE"<<"\tdelta";

//***********Define the new E values************
      for(i=0;i<matnum;i++)
      {
            for(j=0;j<10;j++)
            {
                  ENew[i][j]=E[i][j];
                  ENew2[i][j]=E[i][j];
                  ENew_small[i][j]=E[i][j];
                  ENew_big[i][j]=E[i][j];
            }
```

```
      }
      for(i=0;i<7;i++)
            Ei[0][i]=E[0][i];
//*********Check solution near convergence********
/*    if(*del==0)
      {
            crash_del = Calc_Increment(E, C, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI,
B, &SSE_del, del);
            if (crash_del!=0)
            {
                  cout<<"Singular Matrix at solution";
                  return 1;
            }
            SSEold=SSE_del;
            for (i=0;i<7;i++)
            {
                  CNew[0][i]+=B[i];
            }

            crash_del = Calc_Increment(E, CNew, matnum, mat, radii, theta, Loads, Exp_Strain, position,
PHI, B, &SSE_del, del);
      }
*/
      for (int iteration=1;iteration <MaxIteration+1;iteration++)
      {
            *counter=iteration;
//          count+=1;
//          if(count==10)//periodically perturb del to make sure it is optimizing
//          {
//                *del/=10000;
//                count=0;
//          }
//***********Calculate error function and correction to values*************

            deltabig=*del*10;
            deltasmall=*del/10;
            del_orig = *del;
```

```
//Most efficient step size delta*10, 1, 0.1
//************Calculate three different B values for the different delta values************

            crash_del = Calc_Increment(E, P, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI,
B, &SSE_del, del);
            crash_big = Calc_Increment(E, P, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI,
B_big, &SSE_big, &deltabig);
            crash_small = Calc_Increment(E, P, matnum, mat, radii, theta, Loads, Exp_Strain, position, PHI,
B_small, &SSE_small, &deltasmall);

            SSEold=SSE_del;         //all SSE values are the same here - same C values input, only the B
values change

            if(SSEold<=Acceptable)
            {
                  *sse = SSEold;
                  SSEFile << endl<<iteration<<"\t"<<SSEold<<"\t"<<*del;
                  cout<<endl<<SSEold;
                  return 3;
            }

            if ((crash_del != 0)&&(crash_big !=0)&&(crash_small !=0))
            {
                  *sse = 1.0;
                  return 0;
            }

            for(i=0;i<NAP;i++)
            {
                  ENew[0][P[i]]=E[0][P[i]]+B[i];
                  ratio[i]=B[i];
                  ENew_big[0][P[i]]=E[0][P[i]]+B_big[i];
                  ENew_small[0][P[i]]=E[0][P[i]]+B_small[i];
            }

            crash_del = Calc_Increment(ENew, P, matnum, mat, radii, theta, Loads, Exp_Strain, position,
PHI, B, &SSE_del, del);
```

```
            crash_big = Calc_Increment(ENew_big, P, matnum, mat, radii, theta, Loads, Exp_Strain, position,
PHI, B, &SSE_big, &deltabig);
            crash_small = Calc_Increment(ENew_small, P, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B, &SSE_small, &deltasmall);


            if ((crash_del != 0)&&(crash_big !=0)&&(crash_small !=0))
            {
                  *sse = 1.0;
                  return 0;
            }

            SSEnew= SSE_del;

            if((SSE_big<SSE_del)&&(SSE_big<SSE_small))
            {
                  *del*=10;
                  SSEnew= SSE_big;
                  for(i=0;i<NAP;i++)
                  {
                        ENew[0][P[i]]=ENew_big[0][P[i]];
                        ratio[i]=B_big[i];
                  }

            }
            if((SSE_small<SSE_del)&&(SSE_small<SSE_big))
            {
                  *del/=10;
                  SSEnew= SSE_small;
                  for(i=0;i<NAP;i++)
                  {
                        ENew[0][P[i]]=ENew_small[0][P[i]];
                        ratio[i]=B_small[i];
                  }
            }
//If value is below the threshold level
```

```
            if(SSEnew<=Acceptable)//if error is below an acceptable value - quit
                  {
                        for(i=0;i<7;i++)
                        {
                              E[0][i] = ENew[0][i];
                              Ei[iteration][i] = ENew[0][i];
                        }
                        *sse = SSEnew;
                        SSEFile <<endl <<iteration<<"\t"<<SSEnew<<"\t"<<*del;
                        cout<<endl<<SSEnew;
                        return 3;
                  }
//**********if the revision reduces SSE - take step***********
            if(SSEnew<=SSEold)
            {
                  max=0.0;
                  for(i=0;i<7;i++)
                        Ei[iteration][i] = ENew[0][i];
                  for(i=0;i<NAP;i++)
                  {
                        E[0][P[i]] = ENew[0][P[i]];
                        ratio[i]/=ENew[0][P[i]];                  //Checking the size of the steps
                        if(max<ratio[i])                    //take the largest
                              max=ratio[i];
                  }
                  SSE_ratio = SSEnew/SSEold;
                  SSEold = SSEnew;
                  *sse = SSEnew;


                  if((max<=minimum))//||((1-SSE_ratio)<=minimum))
                  {
                        flag+=1;
                  }
                  else
                  {
                        flag=0;
                  }
```

```
                  if(flag>5 )
                  {
                        SSEFile <<endl <<iteration<<"\t"<<SSEnew<<"\t"<<*del;
                        return 4;
                  }
            }
//***********if not - Change Delta to see if it helps - repeat until it is or quit***********
            else
            {
                  while (SSEnew>SSEold)
                  {
                        *del*=10;
                        cout<<"\nChanging Delta Value - del = "<<*del;


                        deltabig=*del*10;
                        deltasmall=*del/10;
                        del_orig = *del;

                        crash_del = Calc_Increment(E, P, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B, &SSE_del, del);
                        crash_big = Calc_Increment(E, P, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B_big, &SSE_big, &deltabig);
                        crash_small = Calc_Increment(E, P, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B_small, &SSE_small, &deltasmall);

                        if ((crash_del != 0)||(crash_big !=0)||(crash_small !=0))
                        {
                              *sse = 1.0;
                              return 0;
                        }


                        for(i=0;i<NAP;i++)
                        {
                              ENew[0][P[i]]=E[0][P[i]]+B[i];
                              ENew_big[0][P[i]]=E[0][P[i]]+B_big[i];
                              ENew_small[0][P[i]]=E[0][P[i]]+B_small[i];
```

```
                        }


                        crash_del = Calc_Increment(ENew, P, matnum, mat, radii, theta, Loads, Exp_Strain,
position, PHI, B, &SSE_del, del);
                        crash_big = Calc_Increment(ENew_big, P, matnum, mat, radii, theta, Loads,
Exp_Strain, position, PHI, B, &SSE_big, &deltabig);
                        crash_small = Calc_Increment(ENew_small, P, matnum, mat, radii, theta, Loads,
Exp_Strain, position, PHI, B, &SSE_small, &deltasmall);


                        if ((crash_del != 0)||(crash_big !=0)||(crash_small !=0))
                        {
                        *sse = 1.0;
                        return 0;
                        }

                        SSEnew= SSE_del;

                        if((SSE_big<SSE_del)&&(SSE_big<SSE_small))
                        {
                              SSEnew= SSE_big;
                              for(i=0;i<NAP;i++)
                              ENew[0][P[i]]=ENew_big[0][P[i]];
                        }
                        if((SSE_small<SSE_del)&&(SSE_small<SSE_big))
                        {
                              SSEnew= SSE_small;
                              for(i=0;i<NAP;i++)
                                    ENew[0][P[i]]=ENew_small[0][P[i]];
                        }

//***********Check new values now that delta is increased


                        if(SSEnew<SSEold) //if SSE is reduced after a interval halving - make changes
                        {
                              for(i=0;i<7;i++)
```

```
                        {
                              E[0][i] = ENew[0][i];
                              Ei[iteration][i] = E[0][i];
                        }
                        SSEold = SSEnew;
                        *sse = SSEnew;

                  }
                  if (*del>100000000) //after several interval halving steps
                  {
                        *sse = SSEold;
                        SSEFile <<endl <<iteration<<"\t"<<SSEold<<"\t"<<*del;

                        cout<<endl<<SSEold;
                        for(i=0;i<7;i++)
                              Ei[iteration][i] = E[0][i];
                        return 1;
                  }

            }
            cout<<endl<<iteration<<"\t"<<*sse<<"\t"<<*del;
            for(i=0;i<7;i++)
            {
                  E[0][i] = ENew[0][i];
                  Ei[iteration][i] = E[0][i];
            }
      }
      SSEFile <<endl <<iteration<<"\t"<<*sse<<"\t"<<*del;
      cout<<endl<<iteration<<"\t"<<*sse<<"\t"<<*del;
}
*sse = SSEnew;
SSEFile.close();
return 2;
}

/*****************************************************************************/
//This function converts the Cij Values back to E1, E2, E3, G12, nu12, nu13, nu23
```

```
void C_to_E(double *c, double *Evalues)
{

	double S[4][4], col[4], CijMatrix[4][4];
	int i, j, indx[4], d=1;

	//Need to build the Cij matrix
	for(i=0;i<4;i++)
	{
		col[i]=0.0;
		for(j=0;j<4;j++)
		{
			CijMatrix[i][j]=0.0;
			S[i][j] = 0.0;
		}
	}

	CijMatrix[0][0] = c[0]; //C11
	CijMatrix[0][1] = c[1]; //C12
	CijMatrix[1][0] = c[1]; //C21
	CijMatrix[0][2] = c[2]; //C13
	CijMatrix[2][0] = c[2]; //C31
	CijMatrix[1][1] = c[3]; //C22
	CijMatrix[1][2] = c[4]; //C23
	CijMatrix[2][1] = c[4]; //C32
	CijMatrix[2][2] = c[5]; //C33
	CijMatrix[3][3] = c[6]; //C66


//Invert the Cij Matrix to get the Sij Matrix

	LUDecomposition(&CijMatrix[0][0],4,indx,d);
	for(j=0;j<4;j++)
	{
		for(i=0;i<4;i++)
			col[i] = 0.0;
		col[j] = 1.0;
		LUBackSub(&(CijMatrix[0][0]),4,indx,&col[0]);
```

```
            for(i=0;i<4;i++)
                  S[i][j]=col[i];
      }

//Calculate E values from the Sij Values

      Evalues[0] = 1.0/S[0][0];
      Evalues[1] = 1.0/S[1][1];
      Evalues[2] = 1.0/S[2][2];
      Evalues[3] = 1.0/S[3][3];
      Evalues[4] = -S[0][1]*Evalues[0];
      Evalues[5] = -S[0][2]*Evalues[0];
      Evalues[6] = -S[1][2]*Evalues[1];
}
```

10.2 Inversion.h

```
//Inversion.h

// Header file for the Inverse Solution
// Modify the needed information for each different material run
#include <math.h>
#include <fstream.h>
#define K 20                        //Number of plies in the composite lay-up
#define KK 2*K+2        //Number of equations for Elastic Solution
#define dT 0                  //Temperature change for thermal strains and stresses
#define Pi acos(-1)           //Define Pi
#define numloads 100          //Number of load conditions applied (1 Fx + 1 Tx + 1 Pi = 3 !!!REQUIRED!!!)
#define NAP 7                 //Number of active parameters (number of parameters being optimized - 7 Cij
values)
#define ND  3*numloads  //Number of data points (number of strain points given - 3 strains per load
applied)
#define Acceptable 1e-030     //Convergence Criteria - SSE value that is sufficiently small
#define MaxIteration 1000     //Number of iterations before quitting
```

```
int Calc_Increment(double E[][10], int P[], int matnum, int mat[], double radii[], double theta[], double
Loads[][4], double Exp_Strain[][3], char position[], double PHI[], double B[], double *SumSquareErr, double
*del);
int Minimize(double E[][10], int P[], int matnum, int mat[], double Cij[][7], double radii[], double
theta[], double Loads[][4], double Exp_Strain[][3], char position[], double PHI[], double B[], double *SSE,
int *count, ofstream SSEFile, double *del);
```

10.3 Elastic Solution.cpp

```
//Levenberg-Marquardt Ei -- Elastic_Solution.cpp
#include "Elastic_Solution.h"
#include <iostream.h>
#include <fstream.h>

void Elastic_Solution(double E[][10], int matnum, int mat[], double r[], double th[], double *load, double
Calc_Strain[], char strainposition)
{
      int i=0, j, k, indx[KK];
      long theErr=0;
      double sc[K][4][5], bc[K][3], L[K], cbm[K][4][4], et[K][4],  (*pC)[7];
      double A[KK][KK], B[KK], Cmatrix[5][7];

      for(j=0;j<matnum;j++)
            for(k=0;k<7;k++)
                  Cmatrix[j][k]=0.0;

      pC=Cmatrix;

      CMatrix(E, pC, matnum);

      Cbar(pC, th, E, matnum, mat, cbm, et);          //Transformations to create the cbar matrix

      stress(cbm, et, L, bc, sc);                     //Relations between strain, displacement and stress
```

```
      mat_stack(cbm, sc, L, bc, r, load, A, B); //Creates matrix to solve for constants Eo,Go, A1's and
A2's

      SysScale(&(A[0][0]),B,KK);                      //Scaling to increase numerical accuracy

      for (int jj=0;jj<KK;jj++)
            indx[jj] = 0;

      theErr=LUDecomposition(&(A[0][0]),KK,indx,i);   //performs PA=LU for LUBackSub
      if(theErr!=0)
      {
            cout<<"\nSingular Matrix!!!!\n";
            //put in output file for A matrix
      }

      LUBackSub(&(A[0][0]),KK,indx,B);          //Solves system PAx=PB (returns x in B)


      output(B, r, L, bc, Calc_Strain, strainposition);//Returns calculated strains

}

//*********************************************************
//***********Internal Subroutines*************************

// Calcualtes the C Matrix Values from the input data
//-------------------------------------------------------------------------------
void CMatrix(double E[][10], double (*C)[7], int matnum) //c matrix subroutine
{
      double v;

      for(int i=0;i<matnum;i++)
      {
            v = (1 - E[i][4] *(E[i][4] * E[i][1] / E[i][0] + 2 * E[i][6] * E[i][5] * E[i][2] / E[i][0]) -
E[i][5] * E[i][5] * E[i][2] / E[i][0] - E[i][6] * E[i][6] * E[i][2] / E[i][1]);

            C[i][0] = (1 - E[i][6] * E[i][6] * E[i][2] / E[i][1]) * E[i][0] / v;        // c11
```

```
            C[i][1] = (E[i][4] + E[i][5] * E[i][6] * E[i][2] / E[i][1]) * E[i][1] / v;     // c12
            C[i][2] = (E[i][5] + E[i][4] * E[i][6]) * E[i][2] / v;                  // c13
            C[i][3] = (1 - E[i][5] * E[i][5] * E[i][2] / E[i][0]) * E[i][1] / v;        // c22
            C[i][4] = (E[i][6] + E[i][4] * E[i][5] * E[i][1] / E[i][0]) * E[i][2] / v;     // c23
            C[i][5] = (1 - E[i][4] * E[i][4] * E[i][1] / E[i][0]) * E[i][2] / v;        // c33
            C[i][6] = E[i][3];                                                  // c66

            v=0.0;
      }
}


void Cbar(double (*C)[7], double theta[], double E[][10], int matnum, int mat[], double cbm[][4][4], double
et[][4])
{
 //c-bar matrix function

      int i,j,k,p;
      double m=0.0,n=0.0;

      for(i=0;i<K;i++)
            for(j=0;j<4;j++)
                  for(k=0;k<4;k++)
                        cbm[i][j][k]=0.0;

      for(i=0;i<K;i++)
      {
            m = cos(theta[i]);
            n = sin(theta[i]);

            p = mat[i];  //allows for the different material layers

            cbm[i][0][0] = C[p][0]*pow(m,4) + (2*pow(m*n,2))*(C[p][1] + 2*C[p][6]) + C[p][3]*pow(n,4);
            cbm[i][0][1] = pow(m*n,2)*(C[p][0] + C[p][3] - 4*C[p][6]) + C[p][1]*(pow(m,4) + pow(n,4));
            cbm[i][0][2] = C[p][2]*m*m + C[p][4]*n*n;
            cbm[i][0][3] = m*n*(C[p][0]*m*m - C[p][3]*n*n - (C[p][1] + 2*C[p][6])*(m*m - n*n));
            cbm[i][1][1] = C[p][0]*pow(n,4) + (2*pow(m*n,2))*(C[p][1] + 2*C[p][6]) + C[p][3]*pow(m,4);
            cbm[i][1][2] = C[p][2]*n*n + C[p][4]*m*m;
```

```
            cbm[i][1][3] = m*n*(C[p][0]*n*n - C[p][3]*m*m + (C[p][1] + 2*C[p][6])*(m*m - n*n));
            cbm[i][2][2] = C[p][5];
            cbm[i][2][3] = m*n*(C[p][2] - C[p][4]);
            cbm[i][3][3] = (C[p][0] + C[p][3] - 2*C[p][1])*pow(m*n,2) + C[p][6]*pow((m*m - n*n),2);

            for(j = 1; j<4 ; j++)
                  for(k = 0; k<j ; k++)
                        cbm[i][j][k] = cbm[i][k][j];  //stacks symmetric terms

            et[i][0] = (E[p][7]*m*m + E[p][8]*n*n)*dT;   //X       thermal strain
            et[i][1] = (E[p][7]*n*n + E[p][8]*m*m)*dT;   //THETA   thermal strain
            et[i][2] = E[p][9]*dT;                     //R       thermal strain
            et[i][3] = 2*m*n*(E[p][7] - E[p][8])*dT;     //X-THETA thermal strain
      }
}

void stress(double cbm[][4][4],double et[][4],double L[],double bc[][3],double sc[][4][5])
{
 //stress coefficients

      short i,j;
      double zz=0;
      int a;

      for(i=0 ; i<K ; i++)
      {
            a=0;
            L[i] = sqrt(fabs(cbm[i][1][1] / cbm[i][2][2]));
      //    cout <<"\nL["<<i<<"]\t"<<L[i];
            if(L[i]==1.0)
                  a = 1;
            if(L[i]==2.0)
                  a=2;

      switch(a)
            {
            case 1:  //Isotropic and Transversely Isotropic Condition cbm[i][1][1] = cbm[i][2][2]
                  {
```

```
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2])/(2*cbm[i][2][2]);
//Gamma
                  bc[i][1] = (cbm[i][1][3] - 2*cbm[i][2][3])/(4*cbm[i][2][2] - cbm[i][1][1]);
//Omega
                  bc[i][2] = 1/(2*cbm[i][2][2]);
            //Psi

                  zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                  zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                  bc[i][2] *= zz;

                  for(j = 0; j<4 ; j++)
                  {

                        sc[i][j][4] = - et[i][0]*cbm[i][0][j] - et[i][1]*cbm[i][1][j];
                        sc[i][j][4] -=  et[i][2]*cbm[i][2][j] + et[i][3]*cbm[i][j][3];
                        //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                        //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                        //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                        //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                        //                              :last indx=4-> constant
                  }
            }
            break;
      case 2:  //Causes the b[i][1] term to blowup in default equations
            {
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2])/(3*cbm[i][2][2]);        //Gamma
                  bc[i][1] = (cbm[i][1][3] - 2*cbm[i][2][3])/(16*cbm[i][2][2]);     //Omega
                  bc[i][2] = 1/(3*cbm[i][2][2]);
//Psi

                  zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                  zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                  bc[i][2] *= zz;

                  for(j = 0; j<4 ; j++)
```

```
                        {

                              sc[i][j][4] = - et[i][0]*cbm[i][0][j] - et[i][1]*cbm[i][1][j];
                              sc[i][j][4] -=  et[i][2]*cbm[i][2][j] + et[i][3]*cbm[i][j][3];
                              //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                              //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                              //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                              //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                              //                              :last indx=4-> constant

                        }
                  }
                  break;
            default:  //The General Solution for the Composite Cylinder model
                  {
                        bc[i][0] = (cbm[i][0][1] - cbm[i][0][2]) / (cbm[i][2][2] - cbm[i][1][1]);
      //Gamma
                        bc[i][1] = (cbm[i][1][3] - 2 * cbm[i][2][3]) / (4 * cbm[i][2][2] - cbm[i][1][1]);
      //Omega
                        bc[i][2] = 1 / (cbm[i][2][2] - cbm[i][1][1]);
                  //Psi

                        zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                        zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                        bc[i][2] *= zz;

                        for(j = 0; j<4 ; j++)
                        {
                              sc[i][j][0] = cbm[i][0][j] + bc[i][0]*(cbm[i][1][j] + cbm[i][2][j]);
                              sc[i][j][1] = cbm[i][j][3] + bc[i][1]*(cbm[i][1][j] + 2*cbm[i][2][j]);
                              sc[i][j][2] = cbm[i][1][j] + L[i] * cbm[i][2][j];
                              sc[i][j][3] = cbm[i][1][j] - L[i] * cbm[i][2][j];
                              sc[i][j][4] = bc[i][2]*(cbm[i][1][j] + cbm[i][2][j])- et[i][0]*cbm[i][0][j];
                              sc[i][j][4] += -et[i][1]*cbm[i][1][j] - et[i][2]*cbm[i][2][j] -
et[i][3]*cbm[i][j][3];
                              //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                              //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
```

```
                              //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                              //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                              //                              :last indx=4-> constant

                        }

                  }

            }
      }
}


void mat_stack(double cbm[][4][4],double sc[][4][5], double L[], double bc[][3], double r[], double *load,
double A[][KK], double B[])
{
 //subroutine for stacking the components of the linear system Ax=B

      short i, j, ic;
      double ro,ri,r2, ro2, r3, ro3, r4, ro4, b1t2, b2t2;
      int a;

      for(i = 0; i<KK ; i++)                //initialize [A],{B}
      {
            B[i] = 0.0;
            for(j = 0; j<KK ; j++)
            A[i][j] = 0.0;
      }

 //next two loops stack elements associated with the integrated B.C.
      b1t2 = 0;
      b2t2 = 0;


      for(i = 0; i<K ;i++)
      {
            L[i] = sqrt(fabs(cbm[i][1][1] / cbm[i][2][2]));
            a=0;
```

```
      //set up terms for switching for the degenerate cases
            if(L[i]==1.0)
                  a = 1;
            if(L[i]==2.0)
                  a=2;

            //define radii terms for ease of programming and debugging
            ri = r[i];
            ro = r[i+1];
            r2 = pow(r[i],2);
            ro2 = pow(r[i+1],2);
            ro3 = pow(r[i+1],3);
            r3 = pow(r[i],3);
            ro4 = pow(r[i+1],4);
            r4 = pow(r[i],4);

            switch(a)
            {
            case 1:
                  {
                        A[0][0] += 0.5*(ro2-r2)*(cbm[i][0][0]);
                        A[0][0] += 0.5*((log(ro)-0.5)*ro2-(log(ri)-0.5)*r2)*bc[i][0]*cbm[i][0][1];
                        A[0][0] += 0.5*((log(ro)+0.5)*ro2-(log(ri)+0.5)*r2)*bc[i][0]*cbm[i][0][2];

                        A[0][1] += (cbm[i][0][3]+bc[i][1]*(cbm[i][0][1]+2*cbm[i][0][2]))*(ro3-r3)/3.0;

                        A[1][0] += (cbm[i][0][3]*(ro3-r3))/3.0;
                        A[1][0] += (((1.0/3.0)-log(ri))*cbm[i][1][3]+((-2.0/3.0)-
log(ri))*cbm[i][2][3])*bc[i][0]*r3/3.0;
                        A[1][0] += ((log(ro)-
(1.0/3.0))*cbm[i][1][3]+((2.0/3.0)+log(ro))*cbm[i][2][3])*bc[i][0]*ro3/3.0;
                        A[1][1] += (cbm[i][3][3]+bc[i][1]*(cbm[i][1][3]+2*cbm[i][2][3]))*(ro4-r4)/4.0;


                        b1t2 += 0.5*((log(ro)-0.5)*ro2-(log(ri)-0.5)*r2)*cbm[i][0][1]*bc[i][2];
                        b1t2 += 0.5*((log(ro)+0.5)*ro2-(log(ri)+0.5)*r2)*cbm[i][0][2]*bc[i][2];
                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;
```

```
                        b2t2 += ((3.0*log(ro)-
1.0)*cbm[i][1][3]+(3.0*log(ro)+2.0)*cbm[i][2][3])*ro3*bc[i][2]/9.0;
                        b2t2 -= ((3.0*log(ri)-
1.0)*cbm[i][1][3]+(3.0*log(ri)+2.0)*cbm[i][2][3])*r3*bc[i][2]/9.0;
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            case 2:
                  {
                        A[0][0] += (cbm[i][0][0]+bc[i][0]*(cbm[i][0][1]+cbm[i][0][2]))*(ro2-r2)/2.0;
                        A[0][1] += (((4.0*log(ro)-7.0/3.0)*ro3-(4.0*log(ri)-
7.0/3.0)*r3)*cbm[i][0][1]*bc[i][1])/3.0;
                        A[0][1] += (((4.0*log(ro)-1.0/3.0)*ro3-(4.0*log(ri)-
1.0/3.0)*r3)*2*cbm[i][0][2]*bc[i][1])/3.0;
                        A[0][1] += (ro3-r3)*cbm[i][0][3]/3.0;

                        A[1][0] += (cbm[i][0][3]+bc[i][0]*(cbm[i][1][3]+cbm[i][2][3]))*(ro3-r3)/3.0;
                        A[1][1] += (ro4-r4)*cbm[i][3][3]/4.0+((log(ro)-
0.5)*cbm[i][1][3]+2*log(ro)*cbm[i][2][3])*bc[i][1]*ro4;
                        A[1][1] -= ((0.5-log(ri))*cbm[i][1][3]-2*log(ri)*cbm[i][2][3])*bc[i][1]*r4;


                        b1t2 += (cbm[i][0][1]+cbm[i][0][2])*(ro2-r2)*(bc[i][2])/2.0;
                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;

                        b2t2 += (cbm[i][1][3]+cbm[i][2][3])*(ro3-r3)*bc[i][2]/3.0;
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            default:
                  {//A[0][i] = Fx Conditions  A[1][i] = Tx Conditions
                        A[0][0] += (sc[i][0][0]*( ro2 - r2 )) / 2.0;                              //e terms
                        A[1][0] += (sc[i][3][0]*( ro3 - r3 )) / 3.0;
                        A[0][1] += (sc[i][0][1]*( ro3 - r3 )) / 3.0;                              //g terms
                        A[1][1] += (sc[i][3][1]*( ro4 - r4 )) / 4.0;

                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;                                 //thermal
terms
```

```
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            }
      }

      ic = 0;
      for(i = 2; i<(KK-1) ; i += 2)
      {
            ic++;


      //set up terms for switching for the degenerate cases
            a=0;
            if(L[ic-1]==1.0)
                  a = 1;
            if(L[ic-1]==2.0)
                  a = 2;

            //define radii terms for ease of programming and debugging

            ri = r[ic-1];
            ro = r[ic];
            r2 = pow(r[ic-1],2);
            ro2 = pow(r[ic],2);
            ro3 = pow(r[ic],3);
            r3 = pow(r[ic-1],3);
            ro4 = pow(r[ic],4);
            r4 = pow(r[ic-1],4);

            switch(a)
            {
            case 1:
                  {
                        A[0][i] = (cbm[ic-1][0][1]+cbm[ic-1][0][2])*(r2-ro2)/2.0;
                        A[0][i+1] = (log(ro)-log(ri))*(cbm[ic-1][0][1]-cbm[ic-1][0][2]);//check 2*cbm[ic-
1][0][2]
                        A[1][i] = (ro3-r3)*(cbm[ic-1][1][3]+cbm[ic-1][2][3])/3.0;
```

```
                        A[1][i+1] = (ro-ri)*(cbm[ic-1][1][3]-cbm[ic-1][2][3]);

                  }
                  break;
            case 2:
                  {
                        A[0][i] = (cbm[ic-1][0][1]+2*cbm[ic-1][0][2])*(ro3-r3)/3.0;
                        A[0][i+1] = (2.0*cbm[i][0][2]-cbm[i][0][1])*((1.0/ro) - (1.0/ri));
                        A[1][i] = (cbm[ic-1][1][3]+2*cbm[ic-1][2][3])*(ro4-r4)/4;
                        A[1][i+1] = (cbm[ic-1][1][3]-2*cbm[ic-1][2][3])*(log(ro)-log(ri));
                  }
                  break;
            default:
                  {
                        A[0][i] = sc[ic-1][0][2]*(pow(ro, 1+L[ic-1]) - pow(ri, 1+L[ic-1]) ) / (1+L[ic-1]);
                        A[0][i+1] = sc[ic-1][0][3]*(pow(ro, 1-L[ic-1]) - pow(ri, 1-L[ic-1]) ) / (1-L[ic-
1]);
                        A[1][i] = sc[ic-1][3][2]*(pow(ro, 2+L[ic-1]) - pow(ri, 2+L[ic-1]) ) / (2+L[ic-1]);
                        A[1][i+1] = sc[ic-1][3][3]*(pow(ro, 2-L[ic-1]) - pow(ri, 2-L[ic-1]) ) / (2-L[ic-
1]);
                  }
                  break;
            }
      }
      //Applied Loads
      B[0] = (load[2] / (2 * Pi)) - b1t2;
      B[1] = (load[3] / (2 * Pi)) - b2t2;

 //next loop stacks the elements associated with the interface conditions w[i](r) = w[i+1](r) and sigma-r

      ic = 0;
      for(i = 2; i<(KK-3) ; i += 2)
      {

            //set up terms for switching for the degenerate cases
            a=0;
            if(L[ic]==1.0)
                  a = 1;
```

```
            if(L[ic]==2.0)
                  a=2;

            ic++;

            switch(a)
            {

            case 1:
                  {
                        //The sigma r conditions
                        A[i][0] = (cbm[ic-1][0][2]+bc[ic-1][0]*(cbm[ic-1][1][2]*log(r[ic])+cbm[ic-
1][2][2]*(log(r[ic])+1)));
                        A[i][0] -
=(cbm[ic][0][2]+bc[ic][0]*(cbm[ic][1][2]*log(r[ic])+cbm[ic][2][2]*(log(r[ic])+1)));
                        A[i][1] = ((cbm[ic-1][1][2]+2*cbm[ic-1][2][2])*bc[ic-1][1]+cbm[ic-1][2][3])*r[ic];
                        A[i][1] -=((cbm[ic][1][2]+2*cbm[ic][2][2])*bc[ic][1]+cbm[ic][2][3])*r[ic];

                        A[i][i] =(cbm[ic-1][1][2]+cbm[ic-1][2][2]) ;
                        A[i][i+1] = (cbm[ic-1][1][2]-cbm[ic-1][2][2])*pow(r[ic],-2);//check2*cbm[ic-
1][2][2]
                        A[i][i+2] = -(cbm[ic][1][2]+cbm[ic][2][2]);
                        A[i][i+3] = -(cbm[ic][1][2]-cbm[ic][2][2])*pow(r[ic],-2);//check2*cbm[ic-1][2][2]

                        //The w(r) conditions
                        A[i+1][0] =(bc[ic-1][0]-bc[ic][0])*r[ic]*log(r[ic]);
                        A[i+1][1] =(bc[ic-1][1]-bc[ic][1])*r[ic]*r[ic];


                        A[i+1][i] = r[ic];
                        A[i+1][i+1] = 1.0/r[ic];
                        A[i+1][i+2] = -r[ic];
                        A[i+1][i+3] = -1.0/r[ic];

                        //The thermal contributions
                        B[i] = (cbm[ic][1][2]*log(r[ic])+cbm[ic][2][2]*(log(r[ic])+1))*bc[ic][2];
                        B[i] -= (cbm[ic-1][1][2]*log(r[ic])+cbm[ic-1][2][2]*(log(r[ic])+1))*bc[ic-1][2];
                        B[i] += sc[ic][2][4] - sc[ic-1][2][4];
```

```
                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic]*log(r[ic]);
                  }
                  break;

            case 2:
                  {
                        //The sigma r conditions
                        A[i][0] = (cbm[ic-1][0][2]+bc[ic-1][0]*(cbm[ic-1][1][2]+cbm[ic-1][2][2]));
                        A[i][0] -=(cbm[ic][0][2]+bc[ic][0]*(cbm[ic][1][2]+cbm[ic][2][2]));
                        A[i][1] = (cbm[ic-1][2][3]+bc[ic-1][1]*(cbm[ic-1][1][2]*(4.0*log(r[ic])-
1)+2.0*cbm[ic-1][2][2]*(4.0*log(r[ic])+1.0)))*r[ic];
                        A[i][1] -=(cbm[ic][2][3]+bc[ic][1]*(cbm[ic][1][2]*(4.0*log(r[ic])-
1)+2.0*cbm[ic][2][2]*(4.0*log(r[ic])+1.0)))*r[ic];

                        A[i][i] =(cbm[ic-1][1][2]+2*cbm[ic-1][2][2])*r[ic] ;
                        A[i][i+1] = (cbm[ic-1][1][2]-2*cbm[ic-1][2][2])*pow(r[ic],-3);
                        A[i][i+2] = -(cbm[ic][1][2]+2*cbm[ic][2][2])*r[ic];
                        A[i][i+3] = -(cbm[ic][1][2]-2*cbm[ic][2][2])*pow(r[ic],-3);

                        //The w(r) conditions
                        A[i+1][0] =(bc[ic-1][0]-bc[ic][0])*r[ic]*log(r[ic]);
                        A[i+1][1] =(bc[ic-1][1]-bc[ic][1])*r[ic]*r[ic];


                        A[i+1][i] = r[ic];
                        A[i+1][i+1] = 1.0/r[ic];
                        A[i+1][i+2] = -r[ic];
                        A[i+1][i+3] = -1.0/r[ic];

                        //The thermal contributions
                        B[i] = (cbm[ic][1][2] + cbm[ic][2][2])*bc[ic][2]-(cbm[ic-1][1][2] + cbm[ic-
1][2][2])*bc[ic-1][2];
                        B[i] += sc[ic][2][4] - sc[ic-1][2][4];

                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic];
                  }
                  break;
```

```
            default:
                  {
                        A[i][0] = sc[ic-1][2][0] - sc[ic][2][0];                    //eo terms
                        A[i][1] = ( sc[ic-1][2][1] - sc[ic][2][1] ) * r[ic];  //go terms

                        A[i + 1][0] = ( bc[ic-1][0] - bc[ic][0] ) * r[ic];
                        A[i + 1][1] = ( bc[ic-1][1] - bc[ic][1] ) * r[ic]*r[ic];

                        A[i][i] = sc[ic-1][2][2] * pow(r[ic], (L[ic-1] - 1) );
                        A[i][i+1] = sc[ic-1][2][3] * pow(r[ic], (-L[ic-1] - 1) );
                        A[i][i+2] = -sc[ic][2][2] * pow(r[ic], (L[ic] - 1) );
                        A[i][i+3] = -sc[ic][2][3] * pow(r[ic], (-L[ic] - 1) );

                        A[i+1][i] = pow(r[ic], L[ic-1] );
                        A[i+1][i+1] = pow(r[ic], -L[ic-1] );
                        A[i+1][i+2] = -pow(r[ic], L[ic] );
                        A[i+1][i+3] = -pow(r[ic], -L[ic] );

                        B[i] = sc[ic][2][4] - sc[ic-1][2][4];
                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic];
                  }
                  break;
            }

      }

//next lines stack the matrix elements associated with the pressure B.C.

      //The inner surface pressure conditions
            a=0;
            //set up terms for switching for the degenerate cases
            if(L[0]==1.0)
                  a = 1;
            if(L[0]==2.0)
                  a=2;
            switch(a)
            {
            case 1:
```

```
                  {
                        A[KK-2][0] =
(cbm[0][0][2]+bc[0][0]*(cbm[0][1][2]*log(r[0])+cbm[0][2][2]*(log(r[0])+1)));
                        A[KK-2][1] = ((cbm[0][1][2]+2*cbm[0][2][2])*bc[0][1]+cbm[0][2][3])*r[0];
                        A[KK-2][2] = (cbm[0][1][2]+cbm[0][2][2]);
                        A[KK-2][3] = (cbm[0][1][2]-cbm[0][2][2])*pow(r[0],-2);

                        B[KK-2] = -load[1] - (cbm[0][1][2]*log(r[0])+cbm[0][2][2]*(log(r[0])+1))*bc[0][2] -
sc[0][2][4];
                  }
                  break;
            case 2:
                  {
                        A[KK-2][0] = (cbm[0][0][2]+bc[0][0]*(cbm[0][1][2]+cbm[0][2][2]));
                        A[KK-2][1] = (cbm[0][1][2]*(4*log(r[0])-
1)+2*cbm[0][2][2]*(4*log(r[0])+1))*bc[0][1]*r[0];
                        A[KK-2][2] = (cbm[0][1][2]+2*cbm[0][2][2])*r[0];
                        A[KK-2][3] = (cbm[0][1][2]-2*cbm[0][2][2])*pow(r[0],-3);

                        B[KK-2] = -load[1] - (cbm[0][1][2]+cbm[0][2][2])*bc[0][2] - sc[0][2][4];
                  }
                  break;
            default:
                  {
                        A[KK-2][0] = sc[0][2][0];
                        A[KK-2][1] = sc[0][2][1] * r[0];
                        A[KK-2][2] = sc[0][2][2] * pow(r[0], (L[0] - 1) );
                        A[KK-2][3] = sc[0][2][3] * pow(r[0], (-L[0] - 1) );

                        B[KK-2] = -load[1] - sc[0][2][4];
                  }
                  break;
            }
      //The outer surface pressure conditions
                  a=0;
            //set up terms for switching for the degenerate cases
            if(L[K-1]==1.0)
                  a = 1;
```

```
            if(L[K-1]==2.0)
                  a=2;
            switch(a)
            {
            case 1:
                  {
                        A[KK-2][0] = (cbm[K-1][0][2]+bc[K-1][0]*(cbm[K-1][1][2]*log(r[K])+cbm[K-
1][2][2]*(log(r[K])+1)));
                        A[KK-2][1] = ((cbm[K-1][1][2]+2*cbm[K-1][2][2])*bc[K-1][1]+cbm[K-1][2][3])*r[K];
                        A[KK-2][2] = (cbm[K-1][1][2]+cbm[K-1][2][2]);
                        A[KK-2][3] = (cbm[K-1][1][2]-cbm[K-1][2][2])*pow(r[K],-2);

                        B[KK-2] = -load[0] - (cbm[K-1][1][2]*log(r[K])+cbm[K-1][2][2]*(log(r[K])+1))*bc[K-
1][2]- sc[K-1][2][4];
                  }
                  break;
            case 2:
                  {
                        A[KK-2][0] = (cbm[K-1][0][2]+bc[K-1][0]*(cbm[K-1][1][2]+cbm[K-1][2][2]));
                        A[KK-2][1] = (cbm[K-1][1][2]*(4*log(r[K])-1)+2*cbm[K-
1][2][2]*(4*log(r[K])+1))*bc[K-1][1]*r[K];
                        A[KK-2][2] = (cbm[K-1][1][2]+2*cbm[K-1][2][2])*r[K];
                        A[KK-2][3] = (cbm[K-1][1][2]-2*cbm[K-1][2][2])*pow(r[K],-3);

                        B[KK-2] = -load[0] - (cbm[K-1][1][2]+cbm[K-1][2][2])*bc[K-1][2]- sc[K-1][2][4];
                  }
                  break;
            default:
                  {
                        A[KK-1][0] = sc[K-1][2][0];
                        A[KK-1][1] = sc[K-1][2][1] * r[K];
                        A[KK-1][KK-2] = sc[K-1][2][2] * pow(r[K], ( L[K-1] - 1) );
                        A[KK-1][KK-1] = sc[K-1][2][3] * pow(r[K], ( -L[K-1] - 1) );

                        B[KK-1] = -load[0] - sc[K-1][2][4];
                  }
                  break;
            }
```

```
}
void output(double x[], double r[], double L[], double bc[][3], double Calc_Strain[], char strainposition)
{
      double ex=0.0, eo=0.0, gxo=0.0, w=0.0, Rc;
      int a, i;


      for (int ab=0;ab<3;ab++)
      {
            Calc_Strain[ab]=0.0;
      }

      //Inner or Outer Surface Strains?
      if(strainposition =='i'||strainposition=='I')
      {
            i=0;
            Rc=r[0];
      }
      else  //Outer Surface is the Default condition
      {
            Rc=r[K];
            i=K-1;
      }

      a=0;
      if(L[i]==1.0)
            a = 1;
      if(L[i]==2.0)
            a=2;
      switch (a)
      {
            case 1:           //Isotropic or Transversely Isotropic
                  {
//                      w = bc[i][0]*Rc*log(Rc)*x[0] + bc[i][1]*x[1]*Rc*Rc + x[2*(i)+2]*Rc + x[2*(i)+3]/Rc
+ bc[i][2]*Rc*log(Rc);
                        ex = x[0];        // Ex: Epsilon x
```

```
                        eo = bc[i][0]*log(Rc)*x[0] + bc[i][1]*x[1]*Rc + x[2*(i)+2] + x[2*(i)+3]/(Rc*Rc) +
bc[i][2]*log(Rc);
//                      eo = w/Rc;        // Eo: Epsilon theta
                        gxo = x[1]*Rc;    // Gxo: Gamma x-theta
                  }
                  break;
            case 2:           //bc[i][1] = 0
                  {
//                      w = bc[i][0]*Rc*x[0] + bc[i][1]*x[1]*pow(Rc,2)*(4*log(Rc)-1) + x[2*(i)+2]*pow(Rc,
L[i]) + x[2*(i)+3]*pow(Rc, -L[i]) + bc[i][2]*Rc;
                        ex = x[0];        // Ex: Epsilon x
                        eo = bc[i][0]*x[0] + bc[i][1]*x[1]*Rc*(4*log(Rc)-1) + x[2*(i)+2]*pow(Rc, L[i]-1) +
x[2*(i)+3]*pow(Rc, -L[i]-1) + bc[i][2];
//                      eo = w/Rc;        // Eo: Epsilon theta
                        gxo = x[1]*Rc;    // Gxo: Gamma x-theta
                  }
                  break;
            default:    //General Case
                  {
//                      w = bc[i][0]*Rc*x[0] + bc[i][1]*x[1]*Rc*Rc + x[2*(i)+2]*pow(Rc, L[i]) +
x[2*(i)+3]*pow(Rc, -L[i]) + bc[i][2]*Rc;
                        ex = x[0];        // Ex: Epsilon x
                        eo = bc[i][0]*x[0] + bc[i][1]*x[1]*Rc + x[2*(i)+2]*pow(Rc, L[i]-1) +
x[2*(i)+3]*pow(Rc, -L[i]-1) + bc[i][2];
//                      eo = w/Rc;        // Eo: Epsilon theta
                        gxo = x[1]*Rc;    // Gxo: Gamma x-theta
                  }
                  break;
      }

                                                                        //Epsilon X
            Calc_Strain[0] = ex;
                                                                        //Epsilon Y
            Calc_Strain[1] = eo;
                                                                        //Epsilon XY
            Calc_Strain[2] = gxo;
```

```
}
```

10.4 Elastic_Solution.h

```
//Elastic_Solution.h

#include "Inversion.h"
#include "matrix.h"
#include <iostream.h>
#include <fstream.h>
/*****************************/
/*routines internal to CCM*/
/*****************************/
void Elastic_Solution(double E[][10], int matnum, int mat[], double r[], double theta[], double *load,
double Calc_Strain[], char strainposition);
void Cbar(double (*C)[7], double theta[], double E[][10], int matnum, int mat[], double cbm[][4][4], double
et[][4]);
void stress(double cbm[][4][4], double et[][4], double L[], double bc[][3], double sc[][4][5]);
void mat_stack(double cbm[][4][4], double sc[][4][5], double L[], double bc[][3], double r[], double *load,
double A[][KK], double B[]);
void output(double x[], double r[], double L[], double bc[][3], double Calc_Strain[], char strainposition);
void CMatrix(double E[][10], double (*C)[7], int matval);   //c matrix subroutine
```

10.5 Data_Input.cpp

```
//Data_Input.cpp

#include "Data_Input.h"
```

```
ifstream inputfile, loadfile,expstrain;         //Data file containing input data (material properties,
geometry, loads)
ofstream tracking;                        //Data files for data output and intermediate values

int Input(double E[][10], int P[], int mat[], double radii[], double theta[], double Loads[][4], double
Exp_Strain[][3], char position[])
{
      inputfile.open("Testinput.dat");    // Input.dat contains initial guess E's, and geometry
      loadfile.open("TestLOADS.dat");           // loads.dat contains all loading conditions (must alter
numloads value in Inversion.h)
      expstrain.open("Teststrain.dat");// expstrain.dat contains all the measured strain values (ex, ey,
gxy) for each load condition (numload # of sets)
      tracking.open("tracking.dat");      // Opening Data files for output and tracking of values


      double temp;
      int i,j,matval, matnum;

//    First input value must be the number of material layers

      inputfile>>matnum;

//    Read in the Active Parameters (start at 0 to 6)
      for(i=0;i<NAP;i++)
      {

      inputfile>>P[i];        // Active Parameter list

      }

      for(matval=0;matval<matnum;matval++)
      {
            for(i=0; i<10; i++)
                  inputfile >> E[matval][i];
      }

//Reads in all surface radii
      for(j=0; j<K+1; j++)
```

```
            inputfile >> radii[j];


//Reads in each ply angle (degrees) and converts to radians
      for(i=0; i<K; i++)
      {
            inputfile >> temp;
            theta[i] = temp * (Pi/180.0);
            inputfile >> mat[i];
      }

//Reads in all load and experimental strain values
      for (i=0; i<numloads; i++)                    // !!!!LOAD CONDITIONS!!!!                !!!STRAIN
CONDITIONS!!!
      {
            expstrain>>position[i];                 // position[i] = i or o for inner or outer surface
strains
            for(j=0; j<4; j++)                            // Loads[0] = External Pressure
      exp_strain[][0] = e-x
                  loadfile >> Loads[i][j];          // Loads[1] = Internal Pressure
      exp_strain[][1] = e-y
            for(int k=0;k<3;k++)                    // Loads[2] = Axial Load
      exp_strain[][2] = gamma-xy
                  expstrain>> Exp_Strain[i][k]; // Loads[3] = Torsional Load
      }


//Writes all data and stiffness values to the tracking file
      Output_data(E,P,matnum, mat,radii,theta,Loads, Exp_Strain, position);
      inputfile.close();
      loadfile.close();                                         //Close the data files
      expstrain.close();
      tracking.close();

      cout << endl;
      return matnum;
}
//-------------------------------------------------------------------------------------
```

```
void Output_data(double E[][10], int P[], int matnum, int mat[], double radii[], double theta[], double
Loads[][4], double Exp_Strain[][3], char position[])
{

      tracking << "Tube properties entered:" << endl;

//Prints Elastic Moduli (E1,E2,E3,G12,nu12,nu13,nu23,a1,a2,a3)
      for(int i=0;i<matnum;i++)
      {
            tracking << "\nMaterial #"<<i+1;
            tracking << "\nE1 =\t"<<E[i][0];
            tracking << "\nE2 =\t"<<E[i][1];
            tracking << "\nE3 =\t"<<E[i][2];
            tracking << "\nG12 =\t"<<E[i][3];
            tracking << "\nnu12 =\t"<<E[i][4];
            tracking << "\nnu13 =\t"<<E[i][5];
            tracking << "\nnu23 =\t"<<E[i][6];
            tracking << "\nalpha-1 =\t"<<E[i][7];
            tracking << "\nalpha-2 =\t"<<E[i][8];
            tracking << "\nalpha-3 =\t"<<E[i][9]<<endl<<endl;
      }
// Print the Active Value list
      for(i=0;i<NAP;i++)
      {
            tracking << "\nActive Value #\t"<<i+1;
            switch(P[i])
            {
            case 0: tracking << "\tE1";
                  break;
            case 1: tracking << "\tE2";
                  break;
            case 2: tracking << "\tE3";
                  break;
            case 3: tracking << "\tG12";
                  break;
            case 4: tracking << "\tnu12";
                  break;
            case 5: tracking << "\tnu13";
```

```
                  break;
            case 6: tracking << "\tnu23";
                  break;
            }
      }

      tracking <<"\n\nTube geometry entered:\n";

      for(i=0; i<K; i++)
      {
            tracking << "Ri and Ro of ply #" << i+1<< "\t"<<radii[i]<<"\t\t"<<radii[i+1]<<"\t\tAngle"<<
"\t"<< theta[i]*(180/Pi)<<"\tMaterial #\t"<<mat[i]+1<<endl;
      }
      for(int ii=0;ii<numloads;ii++)
      {
// Printing out the Load conditions to the tracking file
            tracking << "\nLoad Set #"<< ii+1<<endl;
            tracking << "\nLoading Conditions and Measured Strains:" << endl;
            tracking << "Axial Load = \t" << Loads[ii][2] << endl;
            tracking << "Torque = \t" << Loads[ii][3] << endl;
            tracking << "Internal Pressure = \t" << Loads[ii][1] << endl;
            tracking << "External Pressure = \t" << Loads[ii][0] << endl;

//Printing out the Measured Strain values for each applied load condition

            tracking << "\nSurface = ";
            if (position[ii]=='i'||position[ii]=='I')
                  tracking << "\tInner Surface Strains";
            else if (position[ii]=='o'||position[ii]=='O')
                  tracking << "\tOuter Surface Strains";
            else
                  tracking << "\tINVALID Response!";
            tracking << "\nAxial Strain (ex) =\t" << Exp_Strain[ii][0] << endl;
            tracking << "Hoop Strain (ey) =\t" << Exp_Strain[ii][1] << endl;
            tracking << "Shear Strain (gxy) =\t" << Exp_Strain[ii][2] << endl;
      }

}
```

10.6 Data_Input.h

```
// Data Input.h


#include <iostream.h>
#include <iomanip.h>
#include "Inversion.h"
#include <fstream.h>


int Input(double E[][10], int P[], int mat[], double radii[], double theta[], double Loads[][4], double
Exp_Strain[][3], char position[]);
void Output_data(double E[][10], int P[], int matnum, int mat[], double radii[], double theta[], double
Loads[][4], double Exp_Strain[][3], char position[]);
```

10.7 Matrix.cpp

```
#include "Matrix.h"
#include <math.h>
#include <stdlib.h>
#include <iostream.h>
//Matrix.cpp


void SysScale(double *A,double *B,int n)
```

The following subroutines have been omitted due to copyright considerations. The appropriate subroutines can be found in Numerical Recipes (sections on LU Decomposition and Backsubstitution).

```
int LUDecomposition (double *A,int n,int *indx,int d)
void LUBackSub (double *A,int n,int *indx,double *B)
long Gauss(double *A,double *B,double *X,int n)
```

10.8 Matrix.h

```
//Matrix.h

#include <math.h>
#include "Inversion.h"


void SysScale(double *A,double *B,int n);
int LUDecomposition (double *A,int n,int *indx,int d);
void LUBackSub (double *A,int n,int *indx,double *B);
long Gauss(double *A,double *B,double *X,int n);
```

10.9 Jacobian.cpp

```
#include <fstream.h>
#include "jacobian.h"
#include "Elastic_Solution.h"

//Jacobian.cpp

void Jacobian(double E[][10], int P[], int matnum, int mat[], double r[],double th[],double
load[][4],double Exp_Strain[][3], char position[], double jaco[][NAP])
{
      int i,j;
      double h=0.00001;
```

```
      double Etemp[5][10];
      double PHIminus[ND], PHIplus[ND];



      for(i=0;i<matnum;i++)
      {
            for(j=0; j<10; j++)
                  Etemp[i][j] = E[i][j];
      }

      for(i=0; i<NAP; i++)
      {
            Etemp[0][P[i]] = E[0][P[i]]*(1-h);
            PhiSub(Etemp, matnum, mat, r, th, load, Exp_Strain, position, PHIminus);

            Etemp[0][P[i]] = E[0][P[i]]*(1+h);

            PhiSub(Etemp, matnum, mat, r, th, load, Exp_Strain, position, PHIplus);


            Etemp[0][P[i]] = E[0][P[i]];

            for(j=0; j<ND; j++)
                  jaco[j][i] = (PHIplus[j] - PHIminus[j])/(2 * h * E[0][P[i]]);
   }

}

void PhiSub(double E[][10], int matnum, int mat[], double r[],double th[],double load[][4],double
Exp_Strain[][3], char position[], double PHI[])
{
      double Calc_Strain[3];


      for(int i=0 ; i<numloads ; i++)
      {
            Elastic_Solution(E, matnum, mat, r, th, &(load[i][0]), Calc_Strain, position[i]);
```

```
            PHI[i*3] = Exp_Strain[i][0] - Calc_Strain[0];            // Ex
            PHI[i*3+1] = Exp_Strain[i][1] - Calc_Strain[1];          // Ey
            PHI[i*3+2] = Exp_Strain[i][2] - Calc_Strain[2];             // Gxy
      }
}
```

10.10 Jacobian.h

```
//Jacobian.h

#include "Inversion.h"  //for the parameters NAP,ND,etcÉ

void Jacobian(double E[][10],int P[], int matnum, int mat[], double r[],double th[],double
eload[][4],double Exp_Strain[][3], char position[], double jaco[][NAP]);
void PhiSub(double E[][10], int matnum, int mat[], double r[],double th[],double eload[][4],double
Exp_Strain[][3], char position[], double PHI[ND]);
```

10.11 Input Files

10.11.1 A typical load program

Po	Pi	Fx	Tx
0	400	0	0
0	0	9900	0
0	0	0	3700

10.11.2 A typical strain file

Surface	ex	eq	gxq
o	-2.692E-05	0.000128074	1.26742E-05
o	0.000460103	-0.000128363	-1.03551E-05
o	-1.76911E-06	-1.32018E-06	0.000419356

10.11.3 A typical input file

Data file	Description
1	Number of different materials
0	Active Parameters
1	
2	
3	
4	
5	
6	
1.03633e+008	Guess Values for material properties (E1, E2, E3, etc – Not Cij)
6.56901e+007	
1.35229e+009	
1.22377e+007	
0.0220688	
0.276952	
0.20983	
0	CTE’s
0	
0	
0.991833333	Interfacial Radii
0.997901042	
1.00396875	
1.010036458	

1.016104167
1.022171875
1.028239583
1.034307292
1.040375
1.046442708
1.052510417
1.058578125
1.064645833
1.070713542
1.07678125
1.082848958
1.088916667
1.094984375
1.101052083
1.107119792
1.1131875
1.119255208
1.125322917
1.131390625
1.137458333

0	0	Ply Orientation and Material Type
90	0	
0	0	
90	0	
0	0	
90	0	
0	0	
90	0	
0	0	
90	0	
0	0	
90	0	
90	0	
0	0	
90	0	
0	0	
90	0	

```
0     0
90    0
0     0
90    0
0     0
90    0
0     0
```

11 Appendix E. Nelder-Mead Simplex Method

Here is the source code for the Nelder Mead Simplex method for minimization of the error function for the composite tube model

- Inversion.cpp
- Inversion.h
- Elastic_Solution.cpp
- Elastic_Solution.h
- Data_Input.cpp
- Data_Input.h
- Matrix.cpp
- Matrix.h
- Jacobian.cpp
- Jacobian.h

11.1 Inversion.cpp

```
// Inversion Program -> Inversion.cpp

#include <stdlib.h>
#include <iostream.h>
#include <iomanip.h>

#include "Data_Input.h"
#include "Elastic_Solution.h"
#include "Inversion.h"
#include "jacobian.h"
#include <time.h>
```

```
void main()
{
      int count, i,j,ndim=7,finished=0;
      int matnum, mat[K];
      double E[5][10], C[5][7], Etemp[5][10],C_Strain[numloads][3],Calculated_Strain[3];
      double radii[K+1], theta[K], Loads[numloads][4], Exp_Strain[numloads][3];
      double SSEmin=1.0,  duration;
      double P[8][7], tolerance, Y[8], (*pP)[7],(*pE)[10], (*pC)[7];
      char  position[numloads];
      ofstream  CijFile;
      clock_t start, finish;
      ifstream InputP;

      InputP.open("PMatrix.dat");
      CijFile.open("Cij.dat");

      tolerance=1e-8;
      start = clock();  //Starts the timer for time of calculation

// STEP 1.  Read in all data

      matnum = Input(E, C, mat, radii, theta, Loads, Exp_Strain, position);
      //Reads in the values for Cij and E - in case of multiple layers - changes C[0][i] values
      //the rest are constant throughout all calculations

// Read in the 8 sets of start values

      for(i=0;i<8;i++)
      {
            for(j=0;j<7;j++)
                  InputP>>P[i][j];        //The eight initial start values (Ei values)
      }
//Calculate the SSE values for each of the start values
      for(i=0;i<matnum;i++)
      {
            for(j=0;j<10;j++)
```

```
                Etemp[i][j]=E[i][j];
        }


//check input values to make sure it works

        CijFile<<"\n\nVertex\t"<<"SSE\t"<<"C11\t"<<"C12\t"<<"C13\t"<<"C22\t"<<"C23\t"<<"C33\t"<<"C66"<<endl;
        for (i=0;i<8;i++)
        {
                for(j=0;j<7;j++)
                        Etemp[0][j]=P[i][j];
                E_to_C(Etemp,C,matnum);
                Y[i]=Calc_SSE(Etemp, C, matnum, mat, radii, theta, Loads, Exp_Strain, position);

                cout <<Y[i]<<endl;
                CijFile<<i<<"\t"<<Y[i];
                for(j=0;j<7;j++)
                        CijFile<<"\t"<<C[i][j];
                CijFile<<endl;

        }

        CijFile<<"Vertex\t"<<"SSE\t"<<"E1\t"<<"E2\t"<<"E3\t"<<"G12\t"<<"nu12\t"<<"nu13\t"<<"nu23"<<endl;
        for (i=0;i<8;i++)
        {


                CijFile<<i<<"\t"<<Y[i];
                for(j=0;j<7;j++)
                        CijFile<<"\t"<<P[i][j];
                CijFile<<endl;
        }

//Run Nelder-Mead Simplex Minimization method


        pP=P;
```

```
      finished=amoeba(pP, Y, ndim, tolerance, &count, E, C,matnum,mat,radii, theta, Loads, Exp_Strain,
position);


      if(finished!=0)
      {

      }
//Output final values

            cout<< "\nNumber of iterations = "<< count+1<< endl;
            CijFile << "\nIterations = \t"<< count+1<< endl;

      for(i=0;i<8;i++)
      {
            for(j=0;j<7;j++)
                  Etemp[0][j]=P[i][j];

            CijFile << "\nFinal Cij Values:"<< "\t\t\tFinal E Values:"<< endl;;

            E_to_C(Etemp,C,matnum);

            CijFile << "\nC11"<< "\t"<< C[i][0]<<"\t\t"<<"E1\t"<< P[i][0];
            CijFile << "\nC12"<< "\t"<< C[i][1]<<"\t\t"<<"E2\t"<< P[i][1];
            CijFile << "\nC13"<< "\t"<< C[i][2]<<"\t\t"<<"E3\t"<< P[i][2];
            CijFile << "\nC22"<< "\t"<< C[i][3]<<"\t\t"<<"G12\t"<< P[i][3];
            CijFile << "\nC23"<< "\t"<< C[i][4]<<"\t\t"<<"nu12\t"<< P[i][4];
            CijFile << "\nC33"<< "\t"<< C[i][5]<<"\t\t"<<"nu13\t"<< P[i][5];
            CijFile << "\nC66"<< "\t"<< C[i][6]<<"\t\t"<<"nu23\t"<< P[i][6];
            cout<< "\nFinal SSE Value = " << Y[i] << endl;
            CijFile<< "\nFinal SSE Value = " << Y[i] << endl;
      }



      CijFile<<"\n\nVertex\t"<<"SSE\t"<<"C11\t"<<"C12\t"<<"C13\t"<<"C22\t"<<"C23\t"<<"C33\t"<<"C66"<<endl;
      for (i=0;i<8;i++)
      {
```

```
            cout <<Y[i]<<endl;
            CijFile<<i<<"\t"<<Y[i];
            for(j=0;j<7;j++)
                  CijFile<<"\t"<<C[i][j];
            CijFile<<endl;
      }
      CijFile<<"Vertex\t"<<"SSE\t"<<"E1\t"<<"E2\t"<<"E3\t"<<"G12\t"<<"nu12\t"<<"nu13\t"<<"nu23"<<endl;
      for (i=0;i<8;i++)
      {
            CijFile<<i<<"\t"<<Y[i];
            for(j=0;j<7;j++)
                  CijFile<<"\t"<<P[i][j];
            CijFile<<endl;
      }


//Calculate the strain response for the solution values and output in table for easy plotting

      for(i=0;i<7;i++)
            E[0][i]=P[0][i];

      E_to_C(E,C,matnum);
      CijFile<<"\n\nCalculated Strains for the Final Values";
      CijFile<<"\nLoads\t\t\t\t\t"<<"Measured Strains\t\t\t\t"<<"Calculated Strains\n";
      CijFile<<"Po\t"<<"Pi\t"<<"Fx\t"<<"Tx\t\t"<<"Ax\t"<<"Hoop\t"<<"Shear\t\t"<<"Ax\t"<<"Hoop\t"<<"Shear\t"
<<"\tSurface\n";

      for( i=0 ; i<numloads ; i++)
      {

            Elastic_Solution(C, E, matnum, mat, radii, theta, &(Loads[i][0]), Calculated_Strain,
position[i]);

            for(int j=0;j<3;j++)
                  C_Strain[i][j]=Calculated_Strain[j];
            for(j=0;j<4;j++)
                  CijFile<<Loads[i][j]<<"\t";
            CijFile<<"\t";
```

```
            for(j=0;j<3;j++)
                  CijFile<<Exp_Strain[i][j]<<"\t";
            CijFile<<"\t";
            for(j=0;j<3;j++)
                  CijFile<<C_Strain[i][j]<<"\t";
            CijFile<<"\t"<<position[i]<<endl;

      }

// Calculates time used to find solution

      finish = clock();
      duration = (double)(finish - start)/CLOCKS_PER_SEC;
      if(duration>60.0)
      {
            duration=duration/60.0;
            cout << "\nCalculation Time:  "<<duration<< " minutes."<< endl;
            CijFile << "\nCalculation Time:  "<<duration<< " minutes."<< endl;

      }
      else
      {
            cout << "\nCalculation Time:  "<<duration<< " seconds."<< endl;
            CijFile << "\nCalculation Time:  "<<duration<< " seconds."<< endl;

      }
      switch (finished)
      {

            case 0:
            {
                  cout<<"\nLocal Minimum Found\n";
                  break;
            }
            case 2:
            {
                  cout<<"\nMaximum Iterations\n";
                  return;
```

```
            }
      }

      CijFile.close();
}

//-------------------------------------------------------------------------------------------------
/*Calc_Increment calculates the error value PHI and SSE as well as the incremental step value for Cij
*/
double Calc_SSE(double E[][10], double C[][7], int matnum, int mat[], double radii[], double theta[],
double Loads[][4], double Exp_Strain[][3], char position[])
{
      int i;
      double SSE, PHI[ND];


      SSE = 0.0;


      for(i=0;i<ND;i++)
            PHI[i] = 0.0;


      PhiSub(E, C, matnum, mat, radii,theta,Loads, Exp_Strain, position, PHI);

      for(i=0;i<ND;i++)
            SSE += pow(PHI[i],2);   //Calculate the Sum of Square Errors

      return SSE;
}


//-------------------------------------------------------------------------------------------------

/*************************************************************************************/
//This function converts the Cij Values back to E1, E2, E3, G12, nu12, nu13, nu23

void C_to_E(double *c, double *Evalues)
```

```
{
	double S[4][4], col[4], CijMatrix[4][4];
	int i, j, indx[4], d=1;

	//Need to build the Cij matrix
	for(i=0;i<4;i++)
	{
		col[i]=0.0;
		for(j=0;j<4;j++)
		{
			CijMatrix[i][j]=0.0;
			S[i][j] = 0.0;
		}
	}

	CijMatrix[0][0] = c[0]; //C11
	CijMatrix[0][1] = c[1]; //C12
	CijMatrix[1][0] = c[1]; //C21
	CijMatrix[0][2] = c[2]; //C13
	CijMatrix[2][0] = c[2]; //C31
	CijMatrix[1][1] = c[3]; //C22
	CijMatrix[1][2] = c[4]; //C23
	CijMatrix[2][1] = c[4]; //C32
	CijMatrix[2][2] = c[5]; //C33
	CijMatrix[3][3] = c[6]; //C66


//Invert the Cij Matrix to get the Sij Matrix

	LUDecomposition(&CijMatrix[0][0],4,indx,d);
	for(j=0;j<4;j++)
	{
		for(i=0;i<4;i++)
			col[i] = 0.0;
		col[j] = 1.0;
		LUBackSub(&(CijMatrix[0][0]),4,indx,&col[0]);
		for(i=0;i<4;i++)
			S[i][j]=col[i];
```

```
	}

//Calculate E values from the Sij Values

	Evalues[0] = 1.0/S[0][0];
	Evalues[1] = 1.0/S[1][1];
	Evalues[2] = 1.0/S[2][2];
	Evalues[3] = 1.0/S[3][3];
	Evalues[4] = -S[0][1]*Evalues[0];
	Evalues[5] = -S[0][2]*Evalues[0];
	Evalues[6] = -S[1][2]*Evalues[1];
}

void E_to_C(double (*e)[10], double (*c)[7], int matnum) //c matrix subroutine
{
	double v;

	for(int i=0;i<matnum;i++)
	{
		v = (1 - e[i][4] *(e[i][4] * e[i][1] / e[i][0] + 2 * e[i][6] * e[i][5] * e[i][2] / e[i][0]) -
e[i][5] * e[i][5] * e[i][2] / e[i][0] - e[i][6] * e[i][6] * e[i][2] / e[i][1]);

		c[i][0] = (1 - e[i][6] * e[i][6] * e[i][2] / e[i][1]) * e[i][0] / v;	// c11
		c[i][1] = (e[i][4] + e[i][5] * e[i][6] * e[i][2] / e[i][1]) * e[i][1] / v;	// c12
		c[i][2] = (e[i][5] + e[i][4] * e[i][6]) * e[i][2] / v;			// c13
		c[i][3] = (1 - e[i][5] * e[i][5] * e[i][2] / e[i][0]) * e[i][1] / v;	// c22
		c[i][4] = (e[i][6] + e[i][4] * e[i][5] * e[i][1] / e[i][0]) * e[i][2] / v;	// c23
		c[i][5] = (1 - e[i][4] * e[i][4] * e[i][1] / e[i][0]) * e[i][2] / v;	// c33
		c[i][6] = e[i][3];						// c66

		v=0.0;
	}
}
```

The following subroutines have been removed due to copyright considerations. The appropriate subroutines can be adapted from those in Numerical Recipes (Nelder-Mead simplex routines).

```
void GET_PSUM (double (*p)[7], double *psum)

void SWAP(double *a, double *b)

int amoeba (double (*p)[7], double y[], int ndim, double ftol, int *nfunk,double E[][10], double C[][7],
int matnum, int mat[], double radii[], double theta[], double Loads[][4], double Exp_Strain[][3], char
position[])


double amotry(double (*p)[7], double y[], double psum[], int ndim, int ihi, double fac, double E[][10],
double C[][7], int matnum, int mat[], double radii[], double theta[], double Loads[][4], double
Exp_Strain[][3], char position[])
```

11.2 Inversion.h

```
//Inversion.h

// Header file for the Inverse Solution
// Modify the needed information for each different material run
#include <math.h>
#include <fstream.h>

#define K 20                    //Number of plies in the composite lay-up
#define KK 2*K+2        //Number of equations for Elastic Solution
#define dT 0                    //Temperature change for thermal strains and stresses
#define Pi acos(-1)             //Define Pi
#define numloads 10             //Number of load conditions applied (1 Fx + 1 Tx + 1 Pi = 3 !!!REQUIRED!!!)
#define NAP 7                   //Number of active parameters (number of parameters being optimized - 7 Cij
values)
#define ND  3*numloads  //Number of data points (number of strain points given - 3 strains per load
applied)
#define Acceptable 1e-030     //Convergence Criteria - SSE value that is sufficiently small
```

```
#define MaxIteration 100      //Number of iterations before quitting

#define TINY 1.0e-12
#define NMAX 20000


double Calc_SSE(double E[][10], double C[][7],int matnum, int mat[], double radii[], double theta[], double
Loads[][4], double Exp_Strain[][3], char position[]);
void C_to_E(double *c, double *Evalues);
void E_to_C(double (*e)[10], double (*c)[7], int matnum);

int amoeba (double (*p)[7], double y[], int ndim, double ftol, int *nfunk, double E[][10], double C[][7],
int matnum, int mat[], double radii[], double theta[], double Loads[][4], double Exp_Strain[][3], char
position[]);
double amotry(double (*p)[7], double y[], double psum[], int ndim, int ihii, double fac, double E[][10],
double C[][7], int matnum, int mat[], double radii[], double theta[], double Loads[][4], double
Exp_Strain[][3], char position[]);
void GET_PSUM (double (*p)[7], double *psum);
void SWAP(double *a, double *b) ;
```

11.3 Elastic_Solution.cpp

```
//Elastic_Solution.cpp
#include "Elastic_Solution.h"
#include <iostream.h>
#include <fstream.h>

void Elastic_Solution(double C[][7], double E[][10], int matnum, int mat[], double r[], double th[], double
*load, double Calc_Strain[], char strainposition)
{
      int i=0,indx[KK];
      long theErr=0;
      double sc[K][4][5], bc[K][3], L[K], cbm[K][4][4], et[K][4];
```

```
      double A[KK][KK], B[KK];


      Cbar(C, th, E, matnum, mat, cbm, et);           //Transformations to create the cbar matrix

      stress(cbm, et, L, bc, sc);                     //Relations between strain, displacement and stress

      mat_stack(cbm, sc, L, bc, r, load, A, B); //Creates matrix to solve for constants Eo,Go, A1's and
A2's

      SysScale(&(A[0][0]),B,KK);                      //Scaling to increase numerical accuracy

      for (int jj=0;jj<KK;jj++)
            indx[jj] = 0;

      LUDecomposition(&(A[0][0]),KK,indx,i);    //performs PA=LU for LUBackSub

      LUBackSub(&(A[0][0]),KK,indx,B);          //Solves system PAx=PB (returns x in B)


      output(B, r, L, bc, Calc_Strain, strainposition);//Returns calculated strains

}

//*********************************************************
//***********Internal Subroutines*****************************


void Cbar(double C[][7], double theta[], double E[][10], int matnum, int mat[], double cbm[][4][4], double
et[][4])
{
 //c-bar matrix function

      int i,j,k,p;
      double m=0.0,n=0.0;

      for(i=0;i<K;i++)
            for(j=0;j<4;j++)
```

```
                for(k=0;k<4;k++)
                      cbm[i][j][k]=0.0;

      for(i=0;i<K;i++)
      {
            m = cos(theta[i]);
            n = sin(theta[i]);

            p = mat[i];  //allows for the different material layers

            cbm[i][0][0] = C[p][0]*pow(m,4) + (2*pow(m*n,2))*(C[p][1] + 2*C[p][6]) + C[p][3]*pow(n,4);
            cbm[i][0][1] = pow(m*n,2)*(C[p][0] + C[p][3] - 4*C[p][6]) + C[p][1]*(pow(m,4) + pow(n,4));
            cbm[i][0][2] = C[p][2]*m*m + C[p][4]*n*n;
            cbm[i][0][3] = m*n*(C[p][0]*m*m - C[p][3]*n*n - (C[p][1] + 2*C[p][6])*(m*m - n*n));
            cbm[i][1][1] = C[p][0]*pow(n,4) + (2*pow(m*n,2))*(C[p][1] + 2*C[p][6]) + C[p][3]*pow(m,4);
            cbm[i][1][2] = C[p][2]*n*n + C[p][4]*m*m;
            cbm[i][1][3] = m*n*(C[p][0]*n*n - C[p][3]*m*m + (C[p][1] + 2*C[p][6])*(m*m - n*n));
            cbm[i][2][2] = C[p][5];
            cbm[i][2][3] = m*n*(C[p][2] - C[p][4]);
            cbm[i][3][3] = (C[p][0] + C[p][3] - 2*C[p][1])*pow(m*n,2) + C[p][6]*pow((m*m - n*n),2);

            for(j = 1; j<4 ; j++)
                  for(k = 0; k<j ; k++)
                        cbm[i][j][k] = cbm[i][k][j];  //stacks symmetric terms

            et[i][0] = (E[p][7]*m*m + E[p][8]*n*n)*dT;   //X       thermal strain
            et[i][1] = (E[p][7]*n*n + E[p][8]*m*m)*dT;   //THETA   thermal strain
            et[i][2] = E[p][9]*dT;                      //R       thermal strain
            et[i][3] = 2*m*n*(E[p][7] - E[p][8])*dT;     //X-THETA thermal strain
      }
}

void stress(double cbm[][4][4],double et[][4],double L[],double bc[][3],double sc[][4][5])
{
 //stress coefficients

      short i,j;
      double zz=0;
```

```
int a;

for(i=0 ; i<K ; i++)
{
      a=0;
      L[i] = sqrt(fabs(cbm[i][1][1] / cbm[i][2][2]));
//    cout <<"\nL["<<i<<"]\t"<<L[i];
      if(L[i]==1.0)
            a = 1;
      if(L[i]==2.0)
            a=2;

switch(a)
      {
      case 1:  //Isotropic and Transversely Isotropic Condition cbm[i][1][1] = cbm[i][2][2]
            {
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2])/(2*cbm[i][2][2]);
//Gamma
                  bc[i][1] = (cbm[i][1][3] - 2*cbm[i][2][3])/(4*cbm[i][2][2] - cbm[i][1][1]);
//Omega
                  bc[i][2] = 1/(2*cbm[i][2][2]);
            //Psi

                  zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                  zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                  bc[i][2] *= zz;

                  for(j = 0; j<4 ; j++)
                  {
                        sc[i][j][4] = - et[i][0]*cbm[i][0][j] - et[i][1]*cbm[i][1][j];
                        sc[i][j][4] -=  et[i][2]*cbm[i][2][j] + et[i][3]*cbm[i][j][3];
                        //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                        //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                        //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                        //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                        //                              :last indx=4-> constant
                  }
```

```
            }
            break;
      case 2:  //Causes the b[i][1] term to blowup in default equations
            {
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2])/(3*cbm[i][2][2]);        //Gamma
                  bc[i][1] = (cbm[i][1][3] - 2*cbm[i][2][3])/(16*cbm[i][2][2]);     //Omega
                  bc[i][2] = 1/(3*cbm[i][2][2]);
//Psi

                  zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                  zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                  bc[i][2] *= zz;

                  for(j = 0; j<4 ; j++)
                  {

                        sc[i][j][4] = - et[i][0]*cbm[i][0][j] - et[i][1]*cbm[i][1][j];
                        sc[i][j][4] -=  et[i][2]*cbm[i][2][j] + et[i][3]*cbm[i][j][3];
                        //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                        //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                        //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                        //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                        //                              :last indx=4-> constant
                  }
            }
            break;
      default:  //The General Solution for the Composite Cylinder model
            {
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2]) / (cbm[i][2][2] - cbm[i][1][1]);
//Gamma
                  bc[i][1] = (cbm[i][1][3] - 2 * cbm[i][2][3]) / (4 * cbm[i][2][2] - cbm[i][1][1]);
//Omega
                  bc[i][2] = 1 / (cbm[i][2][2] - cbm[i][1][1]);
            //Psi

                  zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                  zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];
```

```
                        bc[i][2] *= zz;

                        for(j = 0; j<4 ; j++)
                        {
                              sc[i][j][0] = cbm[i][0][j] + bc[i][0]*(cbm[i][1][j] + cbm[i][2][j]);
                              sc[i][j][1] = cbm[i][j][3] + bc[i][1]*(cbm[i][1][j] + 2*cbm[i][2][j]);
                              sc[i][j][2] = cbm[i][1][j] + L[i] * cbm[i][2][j];
                              sc[i][j][3] = cbm[i][1][j] - L[i] * cbm[i][2][j];
                              sc[i][j][4] = bc[i][2]*(cbm[i][1][j] + cbm[i][2][j])- et[i][0]*cbm[i][0][j];
                              sc[i][j][4] += -et[i][1]*cbm[i][1][j] - et[i][2]*cbm[i][2][j] -
et[i][3]*cbm[i][j][3];
                              //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                              //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                              //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                              //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                              //                              :last indx=4-> constant

                        }
                  }
            }
      }
}


void mat_stack(double cbm[][4][4],double sc[][4][5], double L[], double bc[][3], double r[], double *load,
double A[][KK], double B[])
{
 //subroutine for stacking the components of the linear system Ax=B

      short i, j, ic;
      double ro,ri,r2, ro2, r3, ro3, r4, ro4, b1t2, b2t2;
      int a;

      for(i = 0; i<KK ; i++)                //initialize [A],{B}
      {
            B[i] = 0.0;
            for(j = 0; j<KK ; j++)
```

```
            A[i][j] = 0.0;
      }

//next two loops stack elements associated with the integrated B.C.
      b1t2 = 0;
      b2t2 = 0;

      for(i = 0; i<K ;i++)
      {
            L[i] = sqrt(fabs(cbm[i][1][1] / cbm[i][2][2]));
            a=0;
      //set up terms for switching for the degenerate cases
            if(L[i]==1.0)
                  a = 1;
            if(L[i]==2.0)
                  a=2;

            //define radii terms for ease of programming and debugging
            ri = r[i];
            ro = r[i+1];
            r2 = pow(r[i],2);
            ro2 = pow(r[i+1],2);
            ro3 = pow(r[i+1],3);
            r3 = pow(r[i],3);
            ro4 = pow(r[i+1],4);
            r4 = pow(r[i],4);

            switch(a)
            {
            case 1:
                  {
                        A[0][0] += 0.5*(ro2-r2)*(cbm[i][0][0]);
                        A[0][0] += 0.5*((log(ro)-0.5)*ro2-(log(ri)-0.5)*r2)*bc[i][0]*cbm[i][0][1];
                        A[0][0] += 0.5*((log(ro)+0.5)*ro2-(log(ri)+0.5)*r2)*bc[i][0]*cbm[i][0][2];

                        A[0][1] += (cbm[i][0][3]+bc[i][1]*(cbm[i][0][1]+2*cbm[i][0][2]))*(ro3-r3)/3.0;

                        A[1][0] += (cbm[i][0][3]*(ro3-r3))/3.0;
```

```
                        A[1][0] += (((1.0/3.0)-log(ri))*cbm[i][1][3]+((-2.0/3.0)-
log(ri))*cbm[i][2][3])*bc[i][0]*r3/3.0;
                        A[1][0] += ((log(ro)-
(1.0/3.0))*cbm[i][1][3]+((2.0/3.0)+log(ro))*cbm[i][2][3])*bc[i][0]*ro3/3.0;
                        A[1][1] += (cbm[i][3][3]+bc[i][1]*(cbm[i][1][3]+2*cbm[i][2][3]))*(ro4-r4)/4.0;


                        b1t2 += 0.5*((log(ro)-0.5)*ro2-(log(ri)-0.5)*r2)*cbm[i][0][1]*bc[i][2];
                        b1t2 += 0.5*((log(ro)+0.5)*ro2-(log(ri)+0.5)*r2)*cbm[i][0][2]*bc[i][2];
                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;

                        b2t2 += ((3.0*log(ro)-
1.0)*cbm[i][1][3]+(3.0*log(ro)+2.0)*cbm[i][2][3])*ro3*bc[i][2]/9.0;
                        b2t2 -= ((3.0*log(ri)-
1.0)*cbm[i][1][3]+(3.0*log(ri)+2.0)*cbm[i][2][3])*r3*bc[i][2]/9.0;
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            case 2:
                  {
                        A[0][0] += (cbm[i][0][0]+bc[i][0]*(cbm[i][0][1]+cbm[i][0][2]))*(ro2-r2)/2.0;
                        A[0][1] += (((4.0*log(ro)-7.0/3.0)*ro3-(4.0*log(ri)-
7.0/3.0)*r3)*cbm[i][0][1]*bc[i][1])/3.0;
                        A[0][1] += (((4.0*log(ro)-1.0/3.0)*ro3-(4.0*log(ri)-
1.0/3.0)*r3)*2*cbm[i][0][2]*bc[i][1])/3.0;
                        A[0][1] += (ro3-r3)*cbm[i][0][3]/3.0;

                        A[1][0] += (cbm[i][0][3]+bc[i][0]*(cbm[i][1][3]+cbm[i][2][3]))*(ro3-r3)/3.0;
                        A[1][1] += (ro4-r4)*cbm[i][3][3]/4.0+((log(ro)-
0.5)*cbm[i][1][3]+2*log(ro)*cbm[i][2][3])*bc[i][1]*ro4;
                        A[1][1] -= ((0.5-log(ri))*cbm[i][1][3]-2*log(ri)*cbm[i][2][3])*bc[i][1]*r4;


                        b1t2 += (cbm[i][0][1]+cbm[i][0][2])*(ro2-r2)*(bc[i][2])/2.0;
                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;

                        b2t2 += (cbm[i][1][3]+cbm[i][2][3])*(ro3-r3)*bc[i][2]/3.0;
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
```

```
                }
                break;
          default:
                {//A[0][i] = Fx Conditions  A[1][i] = Tx Conditions
                      A[0][0] += (sc[i][0][0]*( ro2 - r2 )) / 2.0;                              //e terms
                      A[1][0] += (sc[i][3][0]*( ro3 - r3 )) / 3.0;
                      A[0][1] += (sc[i][0][1]*( ro3 - r3 )) / 3.0;                              //g terms
                      A[1][1] += (sc[i][3][1]*( ro4 - r4 )) / 4.0;

                      b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;                                 //thermal
terms
                      b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                }
                break;
          }
     }

     ic = 0;
     for(i = 2; i<(KK-1) ; i += 2)
     {
          ic++;


     //set up terms for switching for the degenerate cases
          a=0;
          if(L[ic-1]==1.0)
                a = 1;
          if(L[ic-1]==2.0)
                a = 2;

          //define radii terms for ease of programming and debugging

          ri = r[ic-1];
          ro = r[ic];
          r2 = pow(r[ic-1],2);
          ro2 = pow(r[ic],2);
          ro3 = pow(r[ic],3);
          r3 = pow(r[ic-1],3);
```

```
            ro4 = pow(r[ic],4);
            r4 = pow(r[ic-1],4);

            switch(a)
            {
            case 1:
                  {
                        A[0][i] = (cbm[ic-1][0][1]+cbm[ic-1][0][2])*(r2-ro2)/2.0;
                        A[0][i+1] = (log(ro)-log(ri))*(cbm[ic-1][0][1]-cbm[ic-1][0][2]);//check 2*cbm[ic-
1][0][2]
                        A[1][i] = (ro3-r3)*(cbm[ic-1][1][3]+cbm[ic-1][2][3])/3.0;
                        A[1][i+1] = (ro-ri)*(cbm[ic-1][1][3]-cbm[ic-1][2][3]);

                  }
                  break;
            case 2:
                  {
                        A[0][i] = (cbm[ic-1][0][1]+2*cbm[ic-1][0][2])*(ro3-r3)/3.0;
                        A[0][i+1] = (2.0*cbm[i][0][2]-cbm[i][0][1])*((1.0/ro) - (1.0/ri));
                        A[1][i] = (cbm[ic-1][1][3]+2*cbm[ic-1][2][3])*(ro4-r4)/4;
                        A[1][i+1] = (cbm[ic-1][1][3]-2*cbm[ic-1][2][3])*(log(ro)-log(ri));
                  }
                  break;
            default:
                  {
                        A[0][i] = sc[ic-1][0][2]*(pow(ro, 1+L[ic-1]) - pow(ri, 1+L[ic-1]) ) / (1+L[ic-1]);
                        A[0][i+1] = sc[ic-1][0][3]*(pow(ro, 1-L[ic-1]) - pow(ri, 1-L[ic-1]) ) / (1-L[ic-
1]);
                        A[1][i] = sc[ic-1][3][2]*(pow(ro, 2+L[ic-1]) - pow(ri, 2+L[ic-1]) ) / (2+L[ic-1]);
                        A[1][i+1] = sc[ic-1][3][3]*(pow(ro, 2-L[ic-1]) - pow(ri, 2-L[ic-1]) ) / (2-L[ic-
1]);
                  }
                  break;
            }
      }
      //Applied Loads
      B[0] = (load[2] / (2 * Pi)) - b1t2;
      B[1] = (load[3] / (2 * Pi)) - b2t2;
```

```
 //next loop stacks the elements associated with the interface conditions w[i](r) = w[i+1](r) and sigma-r

      ic = 0;
      for(i = 2; i<(KK-3) ; i += 2)
      {

            //set up terms for switching for the degenerate cases
            a=0;
            if(L[ic]==1.0)
                  a = 1;
            if(L[ic]==2.0)
                  a=2;


            ic++;

            switch(a)
            {

            case 1:
                  {
                        //The sigma r conditions
                        A[i][0] = (cbm[ic-1][0][2]+bc[ic-1][0]*(cbm[ic-1][1][2]*log(r[ic])+cbm[ic-
1][2][2]*(log(r[ic])+1)));
                        A[i][0] -
=(cbm[ic][0][2]+bc[ic][0]*(cbm[ic][1][2]*log(r[ic])+cbm[ic][2][2]*(log(r[ic])+1)));
                        A[i][1] = ((cbm[ic-1][1][2]+2*cbm[ic-1][2][2])*bc[ic-1][1]+cbm[ic-1][2][3])*r[ic];
                        A[i][1] -=((cbm[ic][1][2]+2*cbm[ic][2][2])*bc[ic][1]+cbm[ic][2][3])*r[ic];

                        A[i][i] =(cbm[ic-1][1][2]+cbm[ic-1][2][2]) ;
                        A[i][i+1] = (cbm[ic-1][1][2]-cbm[ic-1][2][2])*pow(r[ic],-2);//check2*cbm[ic-
1][2][2]
                        A[i][i+2] = -(cbm[ic][1][2]+cbm[ic][2][2]);
                        A[i][i+3] = -(cbm[ic][1][2]-cbm[ic][2][2])*pow(r[ic],-2);//check2*cbm[ic-1][2][2]

                        //The w(r) conditions
```

```
                        A[i+1][0] =(bc[ic-1][0]-bc[ic][0])*r[ic]*log(r[ic]);
                        A[i+1][1] =(bc[ic-1][1]-bc[ic][1])*r[ic]*r[ic];


                        A[i+1][i] = r[ic];
                        A[i+1][i+1] = 1.0/r[ic];
                        A[i+1][i+2] = -r[ic];
                        A[i+1][i+3] = -1.0/r[ic];

                        //The thermal contributions
                        B[i] = (cbm[ic][1][2]*log(r[ic])+cbm[ic][2][2]*(log(r[ic])+1))*bc[ic][2];
                        B[i] -= (cbm[ic-1][1][2]*log(r[ic])+cbm[ic-1][2][2]*(log(r[ic])+1))*bc[ic-1][2];
                        B[i] += sc[ic][2][4] - sc[ic-1][2][4];
                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic]*log(r[ic]);
                  }
                  break;

            case 2:
                  {
                        //The sigma r conditions
                        A[i][0] = (cbm[ic-1][0][2]+bc[ic-1][0]*(cbm[ic-1][1][2]+cbm[ic-1][2][2]));
                        A[i][0] -=(cbm[ic][0][2]+bc[ic][0]*(cbm[ic][1][2]+cbm[ic][2][2]));
                        A[i][1] = (cbm[ic-1][2][3]+bc[ic-1][1]*(cbm[ic-1][1][2]*(4.0*log(r[ic])-
1)+2.0*cbm[ic-1][2][2]*(4.0*log(r[ic])+1.0)))*r[ic];
                        A[i][1] -=(cbm[ic][2][3]+bc[ic][1]*(cbm[ic][1][2]*(4.0*log(r[ic])-
1)+2.0*cbm[ic][2][2]*(4.0*log(r[ic])+1.0)))*r[ic];

                        A[i][i] =(cbm[ic-1][1][2]+2*cbm[ic-1][2][2])*r[ic] ;
                        A[i][i+1] = (cbm[ic-1][1][2]-2*cbm[ic-1][2][2])*pow(r[ic],-3);
                        A[i][i+2] = -(cbm[ic][1][2]+2*cbm[ic][2][2])*r[ic];
                        A[i][i+3] = -(cbm[ic][1][2]-2*cbm[ic][2][2])*pow(r[ic],-3);

                        //The w(r) conditions
                        A[i+1][0] =(bc[ic-1][0]-bc[ic][0])*r[ic]*log(r[ic]);
                        A[i+1][1] =(bc[ic-1][1]-bc[ic][1])*r[ic]*r[ic];


                        A[i+1][i] = r[ic];
```

```
                        A[i+1][i+1] = 1.0/r[ic];
                        A[i+1][i+2] = -r[ic];
                        A[i+1][i+3] = -1.0/r[ic];

                        //The thermal contributions
                        B[i] = (cbm[ic][1][2] + cbm[ic][2][2])*bc[ic][2]-(cbm[ic-1][1][2] + cbm[ic-
1][2][2])*bc[ic-1][2];
                        B[i] += sc[ic][2][4] - sc[ic-1][2][4];

                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic];

                  }
                  break;

            default:
                  {

                        A[i][0] = sc[ic-1][2][0] - sc[ic][2][0];                    //eo terms
                        A[i][1] = ( sc[ic-1][2][1] - sc[ic][2][1] ) * r[ic];  //go terms

                        A[i + 1][0] = ( bc[ic-1][0] - bc[ic][0] ) * r[ic];
                        A[i + 1][1] = ( bc[ic-1][1] - bc[ic][1] ) * r[ic]*r[ic];

                        A[i][i] = sc[ic-1][2][2] * pow(r[ic], (L[ic-1] - 1) );
                        A[i][i+1] = sc[ic-1][2][3] * pow(r[ic], (-L[ic-1] - 1) );
                        A[i][i+2] = -sc[ic][2][2] * pow(r[ic], (L[ic] - 1) );
                        A[i][i+3] = -sc[ic][2][3] * pow(r[ic], (-L[ic] - 1) );

                        A[i+1][i] = pow(r[ic], L[ic-1] );
                        A[i+1][i+1] = pow(r[ic], -L[ic-1] );
                        A[i+1][i+2] = -pow(r[ic], L[ic] );
                        A[i+1][i+3] = -pow(r[ic], -L[ic] );

                        B[i] = sc[ic][2][4] - sc[ic-1][2][4];
                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic];

                  }
                  break;
```

```
            }
      }

 //next lines stack the matrix elements associated with the pressure B.C.

      //The inner surface pressure conditions
            a=0;
            //set up terms for switching for the degenerate cases
            if(L[0]==1.0)
                  a = 1;
            if(L[0]==2.0)
                  a=2;
            switch(a)
            {
            case 1:
                  {
                        A[KK-2][0] =
(cbm[0][0][2]+bc[0][0]*(cbm[0][1][2]*log(r[0])+cbm[0][2][2]*(log(r[0])+1)));
                        A[KK-2][1] = ((cbm[0][1][2]+2*cbm[0][2][2])*bc[0][1]+cbm[0][2][3])*r[0];
                        A[KK-2][2] = (cbm[0][1][2]+cbm[0][2][2]);
                        A[KK-2][3] = (cbm[0][1][2]-cbm[0][2][2])*pow(r[0],-2);

                        B[KK-2] = -load[1] - (cbm[0][1][2]*log(r[0])+cbm[0][2][2]*(log(r[0])+1))*bc[0][2] -
sc[0][2][4];
                  }
                  break;
            case 2:
                  {
                        A[KK-2][0] = (cbm[0][0][2]+bc[0][0]*(cbm[0][1][2]+cbm[0][2][2]));
                        A[KK-2][1] = (cbm[0][1][2]*(4*log(r[0])-
1)+2*cbm[0][2][2]*(4*log(r[0])+1))*bc[0][1]*r[0];
                        A[KK-2][2] = (cbm[0][1][2]+2*cbm[0][2][2])*r[0];
                        A[KK-2][3] = (cbm[0][1][2]-2*cbm[0][2][2])*pow(r[0],-3);

                        B[KK-2] = -load[1] - (cbm[0][1][2]+cbm[0][2][2])*bc[0][2] - sc[0][2][4];
                  }
                  break;
            default:
```

```
                {
                        A[KK-2][0] = sc[0][2][0];
                        A[KK-2][1] = sc[0][2][1] * r[0];
                        A[KK-2][2] = sc[0][2][2] * pow(r[0], (L[0] - 1) );
                        A[KK-2][3] = sc[0][2][3] * pow(r[0], (-L[0] - 1) );

                        B[KK-2] = -load[1] - sc[0][2][4];
                }
                break;
        }
    //The outer surface pressure conditions
                a=0;
        //set up terms for switching for the degenerate cases
        if(L[K-1]==1.0)
                a = 1;
        if(L[K-1]==2.0)
                a=2;
        switch(a)
        {
        case 1:
                {
                        A[KK-2][0] = (cbm[K-1][0][2]+bc[K-1][0]*(cbm[K-1][1][2]*log(r[K])+cbm[K-
1][2][2]*(log(r[K])+1)));
                        A[KK-2][1] = ((cbm[K-1][1][2]+2*cbm[K-1][2][2])*bc[K-1][1]+cbm[K-1][2][3])*r[K];
                        A[KK-2][2] = (cbm[K-1][1][2]+cbm[K-1][2][2]);
                        A[KK-2][3] = (cbm[K-1][1][2]-cbm[K-1][2][2])*pow(r[K],-2);

                        B[KK-2] = -load[0] - (cbm[K-1][1][2]*log(r[K])+cbm[K-1][2][2]*(log(r[K])+1))*bc[K-
1][2]- sc[K-1][2][4];
                }
                break;
        case 2:
                {
                        A[KK-2][0] = (cbm[K-1][0][2]+bc[K-1][0]*(cbm[K-1][1][2]+cbm[K-1][2][2]));
                        A[KK-2][1] = (cbm[K-1][1][2]*(4*log(r[K])-1)+2*cbm[K-
1][2][2]*(4*log(r[K])+1))*bc[K-1][1]*r[K];
                        A[KK-2][2] = (cbm[K-1][1][2]+2*cbm[K-1][2][2])*r[K];
                        A[KK-2][3] = (cbm[K-1][1][2]-2*cbm[K-1][2][2])*pow(r[K],-3);
```

```
                        B[KK-2] = -load[0] - (cbm[K-1][1][2]+cbm[K-1][2][2])*bc[K-1][2]- sc[K-1][2][4];
                  }
                  break;
            default:
                  {
                        A[KK-1][0] = sc[K-1][2][0];
                        A[KK-1][1] = sc[K-1][2][1] * r[K];
                        A[KK-1][KK-2] = sc[K-1][2][2] * pow(r[K], ( L[K-1] - 1) );
                        A[KK-1][KK-1] = sc[K-1][2][3] * pow(r[K], ( -L[K-1] - 1) );

                        B[KK-1] = -load[0] - sc[K-1][2][4];
                  }
                  break;
            }
}
void output(double x[], double r[], double L[], double bc[][3], double Calc_Strain[], char strainposition)
{
      double ex=0.0, eo=0.0, gxo=0.0, w=0.0, Rc;
      int a, i;


      for (int ab=0;ab<3;ab++)
      {
            Calc_Strain[ab]=0.0;

      }

      //Inner or Outer Surface Strains?
      if(strainposition =='i'||strainposition=='I')
      {
            i=0;
            Rc=r[0];

      }
      else  //Outer Surface is the Default condition
      {
            Rc=r[K];
```

```
            i=K-1;

      }

      a=0;
      if(L[i]==1.0)
            a = 1;
      if(L[i]==2.0)
            a=2;
      switch (a)
      {
            case 1:           //Isotropic or Transversely Isotropic
                  {
                        w = bc[i][0]*Rc*log(Rc)*x[0] + bc[i][1]*x[1]*Rc*Rc + x[2*(i)+2]*Rc + x[2*(i)+3]/Rc
+ bc[i][2]*log(Rc);
                        ex = x[0];        // Ex: Epsilon x
                        eo = w/Rc;        // Eo: Epsilon theta
                        gxo = x[1]*Rc;    // Gxo: Gamma x-theta
                  }
                  break;
            case 2:           //bc[i][1] = 0
                  {
                        w = bc[i][0]*Rc*x[0] + bc[i][1]*x[1]*pow(Rc,2)*(4*log(Rc)-1) + x[2*(i)+2]*pow(Rc,
L[i]) + x[2*(i)+3]*pow(Rc, -L[i]) + bc[i][2]*Rc;
                        ex = x[0];        // Ex: Epsilon x
                        eo = w/Rc;        // Eo: Epsilon theta
                        gxo = x[1]*Rc;    // Gxo: Gamma x-theta
                  }
                  break;
            default:    //General Case
                  {
                        w = bc[i][0]*Rc*x[0] + bc[i][1]*x[1]*Rc*Rc + x[2*(i)+2]*pow(Rc, L[i]) +
x[2*(i)+3]*pow(Rc, -L[i]) + bc[i][2]*Rc;
                        ex = x[0];        // Ex: Epsilon x
                        eo = w/Rc;        // Eo: Epsilon theta
                        gxo = x[1]*Rc;    // Gxo: Gamma x-theta
                  }
                  break;
```

```
      }

            Calc_Strain[0] = ex;                                          //Epsilon X

            Calc_Strain[1] = eo;                                          //Epsilon Y

            Calc_Strain[2] = gxo;                                         //Epsilon XY
}
```

11.4 Elastic_Solution.h

```
 //Elastic_Solution.h

#include "Inversion.h"
#include "matrix.h"
#include <iostream.h>
#include <fstream.h>
/*****************************/
/*routines internal to CCM*/
/*****************************/
void Elastic_Solution(double C[][7], double E[][10], int matnum, int mat[], double r[], double theta[],
double *load, double Calc_Strain[], char strainposition);
void Cbar(double C[][7], double theta[], double E[][10], int matnum, int mat[], double cbm[][4][4], double
et[][4]);
void stress(double cbm[][4][4], double et[][4], double L[], double bc[][3], double sc[][4][5]);
void mat_stack(double cbm[][4][4], double sc[][4][5], double L[], double bc[][3], double r[], double *load,
double A[][KK], double B[]);
void output(double x[], double r[], double L[], double bc[][3], double Calc_Strain[], char strainposition);
```

11.5 Data_Input.cpp

```
//Data_Input.cpp

#include "Data_Input.h"

ifstream inputfile, loadfile,expstrain;         //Data file containing input data (material properties,
geometry, loads)
ofstream tracking, outputfile;                        //Data files for data output and intermediate values

int Input(double E[][10], double C[][7], int mat[], double radii[], double theta[], double Loads[][4],
double Exp_Strain[][3], char position[])
{
      inputfile.open("Testinput.dat");    // Input.dat contains initial guess E's, and geometry
      loadfile.open("TestLOADS.dat");           // loads.dat contains all loading conditions (must alter
numloads value in Inversion.h)
      expstrain.open("Teststrain.dat");// expstrain.dat contains all the measured strain values (ex, ey,
gxy) for each load condition (numload # of sets)
      tracking.open("tracking.dat");      // Opening Data files for output and tracking of values


      double temp;
      char datatype;
      int i,j,matval, matnum;

//    First input value must be the number of material layers

      inputfile>>matnum;

      for(matval=0;matval<matnum;matval++)
      {
            inputfile>>datatype;

//If the first value in the input file is 'C' then reads in Cij values
            if((datatype == 'C')||(datatype=='c'))
            {
```

```
                for(i=0;i<7;i++)
                      inputfile>>C[matval][i];
                for(i=7;i<10;i++)
                      inputfile>>E[matval][i];                        //reads in the thermal coefficients
                C_to_E(&(C[matval][0]),&(E[matval][0]));  //calculates E's
          }

//If 'E' - Reads in Elastic Moduli (E1,E2,E3,G12,nu12,nu13,nu23,a1,a2,a3) IN ORDER!!!
          if((datatype == 'E')||(datatype=='e'))
          {
                for(i=0; i<10; i++)
                      inputfile >> E[matval][i];
                CMatrix(E,C,matnum);          //Calculates the Cij values
          }
    }

//Reads in all surface radii
    for(j=0; j<K+1; j++)
          inputfile >> radii[j];


//Reads in each ply angle (degrees) and converts to radians
    for(i=0; i<K; i++)
    {
          inputfile >> temp;
          theta[i] = temp * (Pi/180.0);
          inputfile >> mat[i];
    }

//Reads in all load and experimental strain values
    for (i=0; i<numloads; i++)                    // !!!!LOAD CONDITIONS!!!!               !!!STRAIN
CONDITIONS!!!
    {
          expstrain>>position[i];                 // position[i] = i or o for inner or outer surface
strains
          for(j=0; j<4; j++)                            // Loads[0] = External Pressure
    exp_strain[][0] = e-x
```

```
                  loadfile >> Loads[i][j];            // Loads[1] = Internal Pressure
      exp_strain[][1] = e-y
            for(int k=0;k<3;k++)                      // Loads[2] = Axial Load
      exp_strain[][2] = gamma-xy
                  expstrain>> Exp_Strain[i][k]; // Loads[3] = Torsional Load
      }


//Writes all data and stiffness values to the tracking file
      Output_data(E,matnum, mat,radii,theta,Loads, Exp_Strain, position, C);
      inputfile.close();
      loadfile.close();                                          //Close the data files
      expstrain.close();
      tracking.close();

      cout << endl;
      return matnum;
}
//-------------------------------------------------------------------------------------------
void Output_data(double E[][10],int matnum, int mat[], double radii[], double theta[], double Loads[][4],
double Exp_Strain[][3], char position[], double C[][7])
{

      tracking << "Tube properties entered:" << endl;

//Prints Elastic Moduli (E1,E2,E3,G12,nu12,nu13,nu23,a1,a2,a3)
      for(int i=0;i<matnum;i++)
      {
            tracking << "\nMaterial #"<<i+1;
            tracking << "\nE1 =\t"<<E[i][0];
            tracking << "\nE2 =\t"<<E[i][1];
            tracking << "\nE3 =\t"<<E[i][2];
            tracking << "\nG12 =\t"<<E[i][3];
            tracking << "\nnu12 =\t"<<E[i][4];
            tracking << "\nnu13 =\t"<<E[i][5];
            tracking << "\nnu23 =\t"<<E[i][6];
            tracking << "\nalpha-1 =\t"<<E[i][7];
            tracking << "\nalpha-2 =\t"<<E[i][8];
```

```
            tracking << "\nalpha-3 =\t"<<E[i][9]<<endl<<endl;
      }

      tracking <<"\n\nTube geometry entered:\n";

      for(i=0; i<K; i++)
      {
            tracking << "Ri and Ro of ply #" << i+1<< "\t"<<radii[i]<<"\t\t"<<radii[i+1]<<"\t\tAngle"<<
"\t"<< theta[i]*(180/Pi)<<"\tMaterial #\t"<<mat[i]+1<<endl;
      }
      for(int ii=0;ii<numloads;ii++)
      {
// Printing out the Load conditions to the tracking file
            tracking << "\nLoad Set #"<< ii+1<<endl;
            tracking << "\nLoading Conditions and Measured Strains:" << endl;
            tracking << "Axial Load = \t" << Loads[ii][2] << endl;
            tracking << "Torque = \t" << Loads[ii][3] << endl;
            tracking << "Internal Pressure = \t" << Loads[ii][1] << endl;
            tracking << "External Pressure = \t" << Loads[ii][0] << endl;

//Printing out the Measured Strain values for each applied load condition

            tracking << "\nSurface = ";
            if (position[ii]=='i'||position[ii]=='I')
                  tracking << "\tInner Surface Strains";
            else if (position[ii]=='o'||position[ii]=='O')
                  tracking << "\tOuter Surface Strains";
            else
                  tracking << "\tINVALID Response!";
            tracking << "\nAxial Strain (ex) =\t" << Exp_Strain[ii][0] << endl;
            tracking << "Hoop Strain (ey) =\t" << Exp_Strain[ii][1] << endl;
            tracking << "Shear Strain (gxy) =\t" << Exp_Strain[ii][2] << endl;
      }
      tracking << "\n\nInitial Cij Values:" << endl;
      for(i=0;i<matnum;i++)
      {
            tracking << "\nMaterial #"<<i+1;
            tracking << "\nC11 = \t"<<C[i][0];
```

```
            tracking << "\nC12 = \t"<<C[i][1];
            tracking << "\nC13 = \t"<<C[i][2];
            tracking << "\nC22 = \t"<<C[i][3];
            tracking << "\nC23 = \t"<<C[i][4];
            tracking << "\nC33 = \t"<<C[i][5];
            tracking << "\nC66 = \t"<<C[i][6]<<endl<<endl;

      tracking << endl;
      }
}

// Calculates the C Matrix Values from the input data
//---------------------------------------------------------------------------
void CMatrix(double E[][10], double C[][7], int matnum) //c matrix subroutine
{
      double v;

      for(int i=0;i<matnum;i++)
      {
            v = (1 - E[i][4] *(E[i][4] * E[i][1] / E[i][0] + 2 * E[i][6] * E[i][5] * E[i][2] / E[i][0]) -
E[i][5] * E[i][5] * E[i][2] / E[i][0] - E[i][6] * E[i][6] * E[i][2] / E[i][1]);

            C[i][0] = (1 - E[i][6] * E[i][6] * E[i][2] / E[i][1]) * E[i][0] / v;        // c11
            C[i][1] = (E[i][4] + E[i][5] * E[i][6] * E[i][2] / E[i][1]) * E[i][1] / v;     // c12
            C[i][2] = (E[i][5] + E[i][4] * E[i][6]) * E[i][2] / v;                   // c13
            C[i][3] = (1 - E[i][5] * E[i][5] * E[i][2] / E[i][0]) * E[i][1] / v;        // c22
            C[i][4] = (E[i][6] + E[i][4] * E[i][5] * E[i][1] / E[i][0]) * E[i][2] / v;     // c23
            C[i][5] = (1 - E[i][4] * E[i][4] * E[i][1] / E[i][0]) * E[i][2] / v;        // c33
            C[i][6] = E[i][3];                                                  // c66

            v=0.0;
      }
}
```

11.6 Data_Input.h

```
// Data Input.h


#include <iostream.h>
#include <iomanip.h>
#include "Inversion.h"
#include <fstream.h>



int Input(double E[][10], double C[][7], int mat[], double radii[], double theta[], double Loads[][4],
double Exp_Strain[][3], char position[]);
void Output_data(double E[][10], int matnum, int mat[], double radii[], double theta[], double Loads[][4],
double Exp_Strain[][3], char position[], double C[][7]);
void CMatrix(double E[][10], double C[][7], int matval);    //c matrix subroutine
```

11.7 Matrix.cpp

```
#include "Matrix.h"
#include <math.h>
#include <stdlib.h>
#include <iostream.h>
//Matrix.cpp
```

The following subroutines have been omitted due to copyright considerations. The appropriate subroutines can be found in Numerical Recipes (sections on LU Decomposition and Backsubstitution).

```
void SysScale(double *A,double *B,int n)
int LUDecomposition (double *A,int n,int *indx,int d)
void LUBackSub (double *A,int n,int *indx,double *B)
long Gauss(double *A,double *B,double *X,int n)
```

11.8 Matrix.h

```
//Matrix.h

#include <math.h>
#include "Inversion.h"


void SysScale(double *A,double *B,int n);
int LUDecomposition (double *A,int n,int *indx,int d);
void LUBackSub (double *A,int n,int *indx,double *B);
long Gauss(double *A,double *B,double *X,int n);
```

11.9 Jacobian.cpp

```
#include <fstream.h>
#include "jacobian.h"
#include "Elastic_Solution.h"

//Jacobian.cpp

void PhiSub(double E[][10],double C[][7], int matnum, int mat[], double r[],double th[],double
load[][4],double Exp_Strain[][3], char position[], double PHI[])
```

```
{


      double Calc_Strain[3];


      for(int i=0 ; i<numloads ; i++)
      {

            Elastic_Solution(C, E, matnum, mat, r, th, &(load[i][0]), Calc_Strain, position[i]);


            PHI[i*3] = Exp_Strain[i][0] - Calc_Strain[0];            // Ex
            PHI[i*3+1] = Exp_Strain[i][1] - Calc_Strain[1];          // Ey
            PHI[i*3+2] = Exp_Strain[i][2] - Calc_Strain[2];             // Gxy

      }

}
```

11.10 Jacobian.h

```
//Jacobian.h

#include "Inversion.h"  //for the parameters NAP,ND,etcÉ

void Jacobian(double E[][10],double C[][7],int matnum, int mat[], double r[],double th[],double
eload[][4],double Exp_Strain[][3], char position[], double jaco[][NAP]);
void PhiSub(double E[][10],double C[][7], int matnum, int mat[], double r[],double th[],double
eload[][4],double Exp_Strain[][3], char position[], double PHI[ND]);
```

11.11 Pmatrix.dat

```
5.09E+06
```

6.07E+05
5.15E+05
1.21E+05
1.24E-01
2.89E-01
2.66E-01

4.05E+06
5.44E+05
4.32E+05
1.20E+05
9.94E-02
3.44E-01
2.78E-01

4.70E+06
4.50E+05
5.15E+05
1.07E+05
1.06E-01
2.42E-01
3.46E-01

6.19E+06
3.75E+05
5.29E+05
1.24E+05
1.03E-01
2.29E-01
3.55E-01

4.80E+06
5.03E+05
3.97E+05
9.54E+04
8.56E-02
2.49E-01
2.47E-01

4.39E+06
3.83E+05
4.95E+05
8.66E+04
8.55E-02
2.77E-01
2.54E-01

5.60E+06
5.91E+05
5.25E+05
1.08E+05
1.20E-01
2.99E-01
2.97E-01

5.39E+06
5.41E+05
5.97E+05
9.14E+04
1.24E-01
2.51E-01
3.38E-01

12 Appendix E. Forward Solution

Here is the source code for the Forward Solution containing the degenerate solutions

- Forward_Solution.cpp
- Forward_Solution.h
- Elastic_Solution.cpp
- Elastic_Solution.h
- Data_Input.cpp
- Data_Input.h
- Matrix.cpp
- Matrix.h

12.1 Forward_Solution.cpp

```
// First attempt at writing a Forward Solution Program

#include <iostream.h>
#include <iomanip.h>
#include "Data_Input.h"
#include "Forward_Solution.h"
#include "Elastic_Solution.h"
#include <fstream.h>
#include <math.h>

void Profile(double cc[][7], double ee[][10], double rad[K+1], double th[K], int mat[matnum], double
BB[KK], ofstream responsexy, ofstream response12);

void main()
{
      int i,mat[K], theErr=0;
```

```
      double E[matnum][10], C[matnum][7],B[KK];
      double radii[K+1], theta[K], Loads[numloads][4], Ep[6], EpX[6];
      ofstream strainfile;
      ofstream responsexy, response12;

      responsexy.open("responsexy.dat");
      response12.open("response12.dat");
      strainfile.open("strain.dat");


      Input(E, radii, theta,mat, Loads);        //Reads the input file for all data values
      CMatrix(E,C);                                   //Calculates the Stiffness Matrix Values

            Output_data(E,radii,theta,mat,Loads,C);   //Writes all data and stiffness values to the screen
and tracking file

            strainfile << "Loading Condition\n";

            for(i=0;i<numloads;i++)
            {
                  theErr=Elastic_Solution(C,E,radii,theta,mat, (&Loads[i][0]),Ep, EpX, B);

                  if (theErr==1)
                  {
                        cout<<"Singular Matrix";
                        strainfile<<"\nSingular Matrix\n";
                  }
                  strainfile << "\nExternal Pressure\t"<<Loads[i][0];
                  strainfile << "\t\tAxial Strain"<<"\tHoop Strain"<<"\tShear Strain";
                  strainfile << "\nInternal Pressure\t"<<Loads[i][1]<<"\tInner Surface
Strain\t"<<EpX[0]<<"\t"<< EpX[1]<<"\t"<<EpX[2];
                  strainfile << "\nAxial Force\t"<<Loads[i][2]<<"\tOuter Surface Strain\t"<<
EpX[3]<<"\t"<<EpX[4]<<"\t"<<EpX[5];

                  strainfile << "\nTorque\t"<<Loads[i][3];
                  strainfile << endl;

                  responsexy<<"Load Condition =\t"<<i;
```

```
                  responsexy<<"\tPo =\t"<<Loads[i][0]<<"\tPi =\t"<<Loads[i][1]<<"\tAxial Force
=\t"<<Loads[i][2]<<"\tTorque =\t"<<Loads[i][3];

                  response12<<"Load Condition =\t"<<i;
                  response12<<"\tPo =\t"<<Loads[i][0]<<"\tPi =\t"<<Loads[i][1]<<"\tAxial Force
=\t"<<Loads[i][2]<<"\tTorque = \t"<<Loads[i][3];

                  Profile(C,E,radii,theta,mat,B,responsexy, response12);

            }
      strainfile.close();

      //Calculate the stress/strain response through the tube wall thickness


      responsexy.close();
      response12.close();

}

void Profile(double cc[][7], double ee[][10], double rad[K+1], double th[K],int mat[], double BB[KK],
ofstream responsexy, ofstream response12)
{

      int i=0,j=0,a;
      double axstrain, axstress,hoopstrain, hoopstress, shearstrain,shearstress, radstrain, radstress,r;
      double sc[K][4][5],bc[K][3], L[K], cbm[K][4][4], et[K][4];
      double e1, e2, e3, g12,s1,s2,s3,s12,m,n;


      Cbar(cc, th, mat, ee, cbm, et);

      stress(cbm, et, L, bc, sc);

      responsexy<<"\nPly number"<<"\tRadius"<<"\tAx Strain"<<"\tHoop Strain"<<"\tShear Strain"<<"\tRadial
Strain"<<"\tAx Stress"<<"\tHoop Stress"<<"\tShear Stress"<<"\tRadial Stress"<<endl;
      response12<<"\nPly number"<<"\tRadius"<<"\t1 Strain"<<"\t2 Strain"<<"\t12 Strain"<<"\t3 Strain"<<"\t1
Stress"<<"\t2 Stress"<<"\t12 Stress"<<"\t3 Stress"<<endl;
```

```
      for (i=0;i<K;i++)
      {
            a=0;
            if(L[i]==1.0)
                  a=1;
            if(L[i]==2.0)
                  a=2;

            for(j=0;j<5;j++)
            {
                  //increments steps through each layer
                  r= rad[i]+j*((rad[i+1]-rad[i])/4);
//************check bc[i][2] definitions - does it need to be mult by r???******************
                  switch(a)
                  {
                  case 1:
                        {
                        //Strain Response x-y coordinates
                              radstrain = (BB[2*i+2])-(BB[2*i+3]*pow(r,-
2))+bc[i][0]*BB[0]*(log(r)+1)+2*bc[i][1]*BB[1]*r+bc[i][2]*(log(r)+1);
                              hoopstrain = (BB[2*i+2])+(BB[2*i+3]*pow(r,-
2))+bc[i][0]*BB[0]*log(r)+bc[i][1]*BB[1]*r+bc[i][2]*log(r);
                              axstrain = BB[0];
                              shearstrain = BB[1]*r;
                        }
                        break;
                  case 2:
                        {
                        //Strain Response x-y coordinates
                              radstrain = (2*BB[2*i+2]*r)-(2*BB[2*i+3]*pow(r,-
3))+bc[i][0]*BB[0]+bc[i][1]*BB[1]*2*r*(4*log(r)+1)+bc[i][2];
                              hoopstrain = (BB[2*i+2]*r)+(BB[2*i+3]*pow(r,-
3))+bc[i][0]*BB[0]+bc[i][1]*BB[1]*r*(4*log(r)-1)+bc[i][2];
                              axstrain = BB[0];
                              shearstrain = BB[1]*r;

                        }
```

```
                        break;
                  default:
                        {
                        //Strain Response x-y coordinates
                              radstrain = (L[i]*BB[2*i+2]*pow(r, L[i]-1))-(L[i]*BB[2*i+3]*pow(r,-L[i]-
1))+bc[i][0]*BB[0]+2*bc[i][1]*BB[1]*r+bc[i][2];
                              hoopstrain = BB[2*i+2]*pow(r,L[i]-1)+BB[2*i+3]*pow(r,-L[i]-
1)+bc[i][0]*BB[0]+bc[i][1]*BB[1]*r+bc[i][2];
                              axstrain = BB[0];
                              shearstrain = BB[1]*r;
                        }
                        break;
                  }

            //Stress Response x-y coordinates
                  axstress = cbm[i][0][0]*(axstrain-et[i][0])+cbm[i][0][1]*(hoopstrain-
et[i][1])+cbm[i][0][2]*(radstrain-et[i][2])+cbm[i][0][3]*(shearstrain-et[i][3]);
                  hoopstress =cbm[i][0][1]*(axstrain-et[i][0])+cbm[i][1][1]*(hoopstrain-
et[i][1])+cbm[i][1][2]*(radstrain-et[i][2])+cbm[i][1][3]*(shearstrain-et[i][3]);
                  radstress = cbm[i][0][2]*(axstrain-et[i][0])+cbm[i][1][2]*(hoopstrain-
et[i][1])+cbm[i][2][2]*(radstrain-et[i][2])+cbm[i][2][3]*(shearstrain-et[i][3]);
                  shearstress =  cbm[i][0][3]*(axstrain-et[i][0])+cbm[i][1][3]*(hoopstrain-
et[i][1])+cbm[i][2][3]*(radstrain-et[i][2])+cbm[i][3][3]*(shearstrain-et[i][3]);

            //Rotate to the material coordinates 1-2
                  m=cos(th[i]);
                  n=sin(th[i]);

                  e1 = axstrain*m*m+hoopstrain*n*n+shearstrain*m*n;
                  e2 = axstrain*n*n+hoopstrain*m*m-shearstrain*m*n;
                  e3 = radstrain;
                  g12 = 2*m*n*(hoopstrain-axstrain)+shearstrain*(m*m-n*n);

                  s1 = axstress*m*m+hoopstress*n*n+shearstress*m*n;
                  s2 = axstress*n*n+hoopstress*m*m-shearstress*m*n;
                  s3 = radstress;
                  s12 = 2*m*n*(hoopstress-axstress)+shearstress*(m*m-n*n);
```

```
            //Output the data to a file

      responsexy<<i+1<<"\t"<<r<<"\t"<<axstrain<<"\t"<<hoopstrain<<"\t"<<shearstrain<<"\t"<<radstrain<<"\t"<
<axstress<<"\t"<<hoopstress<<"\t"<<shearstress<<"\t"<<radstress<<endl;

      response12<<i+1<<"\t"<<r<<"\t"<<e1<<"\t"<<e2<<"\t"<<g12<<"\t"<<e3<<"\t"<<s1<<"\t"<<s2<<"\t"<<s12<<"\t
"<<s3<<endl;


            }
      }
      responsexy <<endl<<endl;
      response12 <<endl<<endl;
}
```

12.2 Forward_Solution.h

```
// Header file for the Forward solution
// Modify the needed information for each different material run
#include <math.h>

#define K 20                  //Number of plies in the composite lay-up
#define KK 2*K+2        //Number of equations for Elastic Solution
#define h 0.0001        //Size of difference used in approximate derivative
#define dT 0                  //Temperature change for thermal strains and stresses
#define Pi acos(-1)
#define numloads 50           //Number of load conditions
#define matnum 1        //Number of different material components (for hybrid composites)
```

12.3 Elastic_Solution.cpp

```
#include "Elastic_Solution.h"
```

```
//Forward Solution -- Elastic_Solution.cpp

int Elastic_Solution(double C[][7], double E[][10], double r[], double th[], int mat[],double *load, double
Ep[], double EpX[], double B[])
{
      int i=0,j,jj,k,indx[KK],flag1=0, flag2=0, num, theErr=0;
      double sc[K][4][5], bc[K][3], L[K], cbm[K][4][4], et[K][4];
      double Amatrix[KK][KK];

      for(j=0;j<KK;j++)
            for(k=0;k<KK;k++)
                  Amatrix[j][k]=0.0;

      num=KK;
      Cbar(C, th, mat,E, cbm, et);                          //Transformations to create the cbar matrix

      stress(cbm, et, L, bc, sc);                                 //Relations between strain, displacement
and stress


      mat_stack(cbm,sc, L, bc, r, load, &(Amatrix[0]), B);        //Creates matrix to solve for constants
Eo,Go, A1's and A2's


      for(j=0;j<KK;j++)                                     //Checks A for isotropic, single layer
condition of
      {                                                                     //either pure tension or pure
shear
            if(Amatrix[j][0]==0.0)
                  flag1+=1;
            if(Amatrix[j][1]==0.0)
                  flag2+=1;
      }

      if(flag1==KK-1)                                             //Isotropic condition of pure shear - no
axial strain
      {
            for(j=0;j<KK;j++)
```

```
            {
                  for(k=0;k<KK-1;k++)
                  {
                        Amatrix[j][k]= Amatrix[j][k+1];
                        B[k]=B[k+1];
                  }
            }
            num=KK-1;
      }

            if(flag2==KK-1)                                    //Isotropic condition of pure axial
tension - no shear strain
      {
            for(j=0;j<KK;j++)
            {
                  for(k=1;k<KK-1;k++)
                  {
                        Amatrix[j][k]= Amatrix[j][k+1];
                        B[k]=B[k+1];
                  }
            }
            num=KK-1;
      }


      SysScale(&(Amatrix[0][0]),B,num);                  //Scaling to increase numerical accuracy


      for (jj=0;jj<num;jj++)
            indx[jj] = 0;

      theErr=LUDecomposition(&(Amatrix[0][0]),num,indx,i);  //performs PA=LU for LUBackSub
      if (theErr!=0)    return(theErr);

      LUBackSub(&(Amatrix[0][0]),num,indx,B);                  //Solves system PAx=PB (returns x in B)


      if (flag1==KK-1)
```

```
	{
		for(j=KK-1;j>0;j--)
			B[j]=B[j-1];
		B[0]=0;
	}
	if (flag2==KK-1)
	{
		for(j=KK-1;j>1;j--)
			B[j]=B[j-1];
		B[1]=0;
	}

	output(B, th, r, L, bc, Ep, load, EpX);		//Returns calculated strains
	return(theErr);
}

//*************************************************************
//***********Internal Subroutines*****************************


void Cbar(double C[][7], double theta[], int mat[], double E[][10], double cbm[][4][4], double et[][4])
{
 //c-bar matrix function

	short i,j,k;
	int p;
	double m=0.0,n=0.0;

	for(i=0;i<K;i++)
	{
		m = cos(theta[i]);
		n = sin(theta[i]);

		p = mat[i];

		cbm[i][0][0] = C[p][0]*pow(m,4) + (2*pow(m*n,2))*(C[p][1] + 2*C[p][6]) + C[p][3]*pow(n,4);
		cbm[i][0][1] = pow(m*n,2)*(C[p][0] + C[p][3] - 4*C[p][6]) + C[p][1]*(pow(m,4) + pow(n,4));
		cbm[i][0][2] = C[p][2]*m*m + C[p][4]*n*n;
```

```
            cbm[i][0][3] = m*n*(C[p][0]*m*m - C[p][3]*n*n - (C[p][1] + 2*C[p][6])*(m*m - n*n));
            cbm[i][1][1] = C[p][0]*pow(n,4) + (2*pow(m*n,2))*(C[p][1] + 2*C[p][6]) + C[p][3]*pow(m,4);
            cbm[i][1][2] = C[p][2]*n*n + C[p][4]*m*m;
            cbm[i][1][3] = m*n*(C[p][0]*n*n - C[p][3]*m*m + (C[p][1] + 2*C[p][6])*(m*m - n*n));
            cbm[i][2][2] = C[p][5];
            cbm[i][2][3] = m*n*(C[p][2] - C[p][4]);
            cbm[i][3][3] = (C[p][0] + C[p][3] - 2*C[p][1])*pow(m*n,2) + C[p][6]*pow((m*m - n*n),2);

            for(j = 1; j<4 ; j++)
                  for(k = 0; k<j ; k++)
                        cbm[i][j][k] = cbm[i][k][j];  //stacks symmetric terms

            et[i][0] = (E[p][7]*m*m + E[p][8]*n*n)*dT;   //X       thermal strain
            et[i][1] = (E[p][7]*n*n + E[p][8]*m*m)*dT;   //THETA   thermal strain
            et[i][2] = E[p][9]*dT;                      //R       thermal strain
            et[i][3] = 2*m*n*(E[p][7] - E[p][8])*dT;     //X-THETA thermal strain

      }
}


void stress(double cbm[][4][4],double et[][4],double L[],double bc[][3],double sc[][4][5])
{
 //stress coefficients with the added degenerate cases of L=1,2

      short i,j;
      double zz=0;
      int a;

      for(i=0 ; i<K ; i++)
      {
            a=0;
            L[i] = sqrt(fabs(cbm[i][1][1] / cbm[i][2][2]));
            if(L[i]==1.0)
                  a = 1;
            if(L[i]==2.0)
                  a=2;
```

```
      switch(a)
      {
      case 1:  //Isotropic and Transversely Isotropic Condition cbm[i][1][1] = cbm[i][2][2]
            {
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2])/(2*cbm[i][2][2]);
//Gamma
                  bc[i][1] = (cbm[i][1][3] - 2*cbm[i][2][3])/(4*cbm[i][2][2] - cbm[i][1][1]);
//Omega
                  bc[i][2] = 1/(2*cbm[i][2][2]);
            //Psi

                  zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                  zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                  bc[i][2] *= zz;

                  for(j = 0; j<4 ; j++)
                  {
                        sc[i][j][4] = - et[i][0]*cbm[i][0][j] - et[i][1]*cbm[i][1][j];
                        sc[i][j][4] -=  et[i][2]*cbm[i][2][j] + et[i][3]*cbm[i][j][3];
                        //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                        //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                        //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                        //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                        //                              :last indx=4-> constant
                  }
            }
            break;
      case 2:  //Causes the b[i][1] term to blowup in default equations
            {
                  bc[i][0] = (cbm[i][0][1] - cbm[i][0][2])/(3*cbm[i][2][2]);        //Gamma
                  bc[i][1] = (cbm[i][1][3] - 2*cbm[i][2][3])/(16*cbm[i][2][2]);     //Omega
                  bc[i][2] = 1/(3*cbm[i][2][2]);
//Psi

                  zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                  zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];
```

```
                        bc[i][2] *= zz;

                        for(j = 0; j<4 ; j++)
                        {
                              sc[i][j][4] = - et[i][0]*cbm[i][0][j] - et[i][1]*cbm[i][1][j];
                              sc[i][j][4] -=  et[i][2]*cbm[i][2][j] + et[i][3]*cbm[i][j][3];
                              //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                              //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                              //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                              //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                              //                              :last indx=4-> constant
                        }
                  }
                  break;
            default:  //The General Solution for the Composite Cylinder model
                  {
                        bc[i][0] = (cbm[i][0][1] - cbm[i][0][2]) / (cbm[i][2][2] - cbm[i][1][1]);
      //Gamma
                        bc[i][1] = (cbm[i][1][3] - 2 * cbm[i][2][3]) / (4 * cbm[i][2][2] - cbm[i][1][1]);
      //Omega
                        bc[i][2] = 1 / (cbm[i][2][2] - cbm[i][1][1]);
                  //Psi

                        zz=(cbm[i][0][2]-cbm[i][0][1])*et[i][0]+(cbm[i][1][2]-cbm[i][1][1])*et[i][1];
                        zz += (cbm[i][2][2]-cbm[i][1][2])*et[i][2]+(cbm[i][2][3]-cbm[i][1][3])*et[i][3];

                        bc[i][2] *= zz;

                        for(j = 0; j<4 ; j++)
                        {
                              sc[i][j][0] = cbm[i][0][j] + bc[i][0]*(cbm[i][1][j] + cbm[i][2][j]);
                              sc[i][j][1] = cbm[i][j][3] + bc[i][1]*(cbm[i][1][j] + 2*cbm[i][2][j]);
                              sc[i][j][2] = cbm[i][1][j] + L[i] * cbm[i][2][j];
                              sc[i][j][3] = cbm[i][1][j] - L[i] * cbm[i][2][j];
                              sc[i][j][4] = bc[i][2]*(cbm[i][1][j] + cbm[i][2][j])- et[i][0]*cbm[i][0][j];
                              sc[i][j][4] += -et[i][1]*cbm[i][1][j] - et[i][2]*cbm[i][2][j] -
et[i][3]*cbm[i][j][3];
```

```
                              //j=0: SIGMAx      coefficients :last indx=0-> coeff of Eo
                              //j=1: SIGMAtheta  coefficients :last indx=1-> coeff of (Go R)
                              //j=2: SIGMAr      coefficients :last indx=2-> coeff of A1 R^(L-1)
                              //j=3: TAUx-theta  coefficients :last indx=3-> coeff of A2 R^(-L-1)
                              //                              :last indx=4-> constant

                        }
                  }
            }
      }
}


void mat_stack(double cbm[][4][4],double sc[][4][5], double L[], double bc[][3], double r[], double *load,
double (*A)[KK], double B[])
{
 //subroutine for stacking the components of the linear system Ax=B

      short i, j, ic;
      double ro,ri,r2, ro2, r3, ro3, r4, ro4, b1t2, b2t2;
      int a;

      for(i = 0; i<KK ; i++)                //initialize [A],{B}
      {
            B[i] = 0.0;
            for(j = 0; j<KK ; j++)
            A[i][j] = 0.0;
      }

 //next two loops stack elements associated with the integrated B.C.
      b1t2 = 0;
      b2t2 = 0;

      for(i = 0; i<K ;i++)
      {
            L[i] = sqrt(fabs(cbm[i][1][1] / cbm[i][2][2]));
            a=0;
      //set up terms for switching for the degenerate cases
```

```
            if(L[i]==1.0)
                  a = 1;
            if(L[i]==2.0)
                  a=2;

            //define radii terms for ease of programming and debugging
            ri = r[i];
            ro = r[i+1];
            r2 = pow(r[i],2);
            ro2 = pow(r[i+1],2);
            ro3 = pow(r[i+1],3);
            r3 = pow(r[i],3);
            ro4 = pow(r[i+1],4);
            r4 = pow(r[i],4);

            switch(a)
            {
            case 1:
                  {
                        A[0][0] += 0.5*(ro2-r2)*(cbm[i][0][0]);
                        A[0][0] += 0.5*((log(ro)-0.5)*ro2-(log(ri)-0.5)*r2)*bc[i][0]*cbm[i][0][1];
                        A[0][0] += 0.5*((log(ro)+0.5)*ro2-(log(ri)+0.5)*r2)*bc[i][0]*cbm[i][0][2];

                        A[0][1] += (cbm[i][0][3]+bc[i][1]*(cbm[i][0][1]+2*cbm[i][0][2]))*(ro3-r3)/3.0;

                        A[1][0] += (cbm[i][0][3]*(ro3-r3))/3.0;
                        A[1][0] += (((1.0/3.0)-log(ri))*cbm[i][1][3]+((-2.0/3.0)-
log(ri))*cbm[i][2][3])*bc[i][0]*r3/3.0;
                        A[1][0] += ((log(ro)-
(1.0/3.0))*cbm[i][1][3]+((2.0/3.0)+log(ro))*cbm[i][2][3])*bc[i][0]*ro3/3.0;
                        A[1][1] += (cbm[i][3][3]+bc[i][1]*(cbm[i][1][3]+2*cbm[i][2][3]))*(ro4-r4)/4.0;


                        b1t2 += 0.5*((log(ro)-0.5)*ro2-(log(ri)-0.5)*r2)*cbm[i][0][1]*bc[i][2];
                        b1t2 += 0.5*((log(ro)+0.5)*ro2-(log(ri)+0.5)*r2)*cbm[i][0][2]*bc[i][2];
                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;
```

```
                        b2t2 += ((3.0*log(ro)-
1.0)*cbm[i][1][3]+(3.0*log(ro)+2.0)*cbm[i][2][3])*ro3*bc[i][2]/9.0;
                        b2t2 -= ((3.0*log(ri)-
1.0)*cbm[i][1][3]+(3.0*log(ri)+2.0)*cbm[i][2][3])*r3*bc[i][2]/9.0;
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            case 2:
                  {
                        A[0][0] += (cbm[i][0][0]+bc[i][0]*(cbm[i][0][1]+cbm[i][0][2]))*(ro2-r2)/2.0;
                        A[0][1] += (((4.0*log(ro)-7.0/3.0)*ro3-(4.0*log(ri)-
7.0/3.0)*r3)*cbm[i][0][1]*bc[i][1])/3.0;
                        A[0][1] += (((4.0*log(ro)-1.0/3.0)*ro3-(4.0*log(ri)-
1.0/3.0)*r3)*2*cbm[i][0][2]*bc[i][1])/3.0;
                        A[0][1] += (ro3-r3)*cbm[i][0][3]/3.0;

                        A[1][0] += (cbm[i][0][3]+bc[i][0]*(cbm[i][1][3]+cbm[i][2][3]))*(ro3-r3)/3.0;
                        A[1][1] += (ro4-r4)*cbm[i][3][3]/4.0+((log(ro)-
0.5)*cbm[i][1][3]+2*log(ro)*cbm[i][2][3])*bc[i][1]*ro4;
                        A[1][1] -= ((0.5-log(ri))*cbm[i][1][3]-2*log(ri)*cbm[i][2][3])*bc[i][1]*r4;


                        b1t2 += (cbm[i][0][1]+cbm[i][0][2])*(ro2-r2)*(bc[i][2])/2.0;
                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;

                        b2t2 += (cbm[i][1][3]+cbm[i][2][3])*(ro3-r3)*bc[i][2]/3.0;
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            default:
                  {//A[0][i] = Fx Conditions  A[1][i] = Tx Conditions
                        A[0][0] += (sc[i][0][0]*( ro2 - r2 )) / 2.0;                              //e terms
                        A[1][0] += (sc[i][3][0]*( ro3 - r3 )) / 3.0;
                        A[0][1] += (sc[i][0][1]*( ro3 - r3 )) / 3.0;                              //g terms
                        A[1][1] += (sc[i][3][1]*( ro4 - r4 )) / 4.0;

                        b1t2 += (sc[i][0][4]*( ro2 - r2 )) / 2.0;                                 //thermal
terms
```

```
                        b2t2 += (sc[i][3][4]*( ro3 - r3 )) / 3.0;
                  }
                  break;
            }
      }

      ic = 0;
      for(i = 2; i<(KK-1) ; i += 2)
      {
            ic++;


      //set up terms for switching for the degenerate cases
            a=0;
            if(L[ic-1]==1.0)
                  a = 1;
            if(L[ic-1]==2.0)
                  a = 2;

            //define radii terms for ease of programming and debugging

            ri = r[ic-1];
            ro = r[ic];
            r2 = pow(r[ic-1],2);
            ro2 = pow(r[ic],2);
            ro3 = pow(r[ic],3);
            r3 = pow(r[ic-1],3);
            ro4 = pow(r[ic],4);
            r4 = pow(r[ic-1],4);

            switch(a)
            {
            case 1:
                  {
                        A[0][i] = (cbm[ic-1][0][1]+cbm[ic-1][0][2])*(r2-ro2)/2.0;
                        A[0][i+1] = (log(ro)-log(ri))*(cbm[ic-1][0][1]-cbm[ic-1][0][2]);//check 2*cbm[ic-
1][0][2]
                        A[1][i] = (ro3-r3)*(cbm[ic-1][1][3]+cbm[ic-1][2][3])/3.0;
```

```
                        A[1][i+1] = (ro-ri)*(cbm[ic-1][1][3]-cbm[ic-1][2][3]);
                  }
                  break;
            case 2:
                  {
                        A[0][i] = (cbm[ic-1][0][1]+2*cbm[ic-1][0][2])*(ro3-r3)/3.0;
                        A[0][i+1] = (2.0*cbm[i][0][2]-cbm[i][0][1])*((1.0/ro) - (1.0/ri));
                        A[1][i] = (cbm[ic-1][1][3]+2*cbm[ic-1][2][3])*(ro4-r4)/4;
                        A[1][i+1] = (cbm[ic-1][1][3]-2*cbm[ic-1][2][3])*(log(ro)-log(ri));
                  }
                  break;
            default:
                  {
                        A[0][i] = sc[ic-1][0][2]*(pow(ro, 1+L[ic-1]) - pow(ri, 1+L[ic-1]) ) / (1+L[ic-1]);
                        A[0][i+1] = sc[ic-1][0][3]*(pow(ro, 1-L[ic-1]) - pow(ri, 1-L[ic-1]) ) / (1-L[ic-
1]);
                        A[1][i] = sc[ic-1][3][2]*(pow(ro, 2+L[ic-1]) - pow(ri, 2+L[ic-1]) ) / (2+L[ic-1]);
                        A[1][i+1] = sc[ic-1][3][3]*(pow(ro, 2-L[ic-1]) - pow(ri, 2-L[ic-1]) ) / (2-L[ic-
1]);
                  }
                  break;
            }
      }
      //Applied Loads
      B[0] = (load[2] / (2 * Pi)) - b1t2;
      B[1] = (load[3] / (2 * Pi)) - b2t2;


 //next loop stacks the elements associated with the interface conditions w[i](r) = w[i+1](r) and sigma-r

      ic = 0;
      for(i = 2; i<(KK-3) ; i += 2)
      {

            //set up terms for switching for the degenerate cases
            a=0;
            if(L[ic]==1.0)
                  a = 1;
```

```
            if(L[ic]==2.0)
                  a=2;
            ic++;

            switch(a)
            {

            case 1:
                  {
                        //The sigma r conditions
                        A[i][0] = (cbm[ic-1][0][2]+bc[ic-1][0]*(cbm[ic-1][1][2]*log(r[ic])+cbm[ic-
1][2][2]*(log(r[ic])+1)));
                        A[i][0] -
=(cbm[ic][0][2]+bc[ic][0]*(cbm[ic][1][2]*log(r[ic])+cbm[ic][2][2]*(log(r[ic])+1)));
                        A[i][1] = ((cbm[ic-1][1][2]+2*cbm[ic-1][2][2])*bc[ic-1][1]+cbm[ic-1][2][3])*r[ic];
                        A[i][1] -=((cbm[ic][1][2]+2*cbm[ic][2][2])*bc[ic][1]+cbm[ic][2][3])*r[ic];

                        A[i][i] =(cbm[ic-1][1][2]+cbm[ic-1][2][2]) ;
                        A[i][i+1] = (cbm[ic-1][1][2]-cbm[ic-1][2][2])*pow(r[ic],-2);//check2*cbm[ic-
1][2][2]
                        A[i][i+2] = -(cbm[ic][1][2]+cbm[ic][2][2]);
                        A[i][i+3] = -(cbm[ic][1][2]-cbm[ic][2][2])*pow(r[ic],-2);//check2*cbm[ic-1][2][2]

                        //The w(r) conditions
                        A[i+1][0] =(bc[ic-1][0]-bc[ic][0])*r[ic]*log(r[ic]);
                        A[i+1][1] =(bc[ic-1][1]-bc[ic][1])*r[ic]*r[ic];


                        A[i+1][i] = r[ic];
                        A[i+1][i+1] = 1.0/r[ic];
                        A[i+1][i+2] = -r[ic];
                        A[i+1][i+3] = -1.0/r[ic];

                        //The thermal contributions
                        B[i] = (cbm[ic][1][2]*log(r[ic])+cbm[ic][2][2]*(log(r[ic])+1))*bc[ic][2];
                        B[i] -= (cbm[ic-1][1][2]*log(r[ic])+cbm[ic-1][2][2]*(log(r[ic])+1))*bc[ic-1][2];
                        B[i] += sc[ic][2][4] - sc[ic-1][2][4];
                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic]*log(r[ic]);
```

```
                }
                break;

        case 2:
                {
                        //The sigma r conditions
                        A[i][0] = (cbm[ic-1][0][2]+bc[ic-1][0]*(cbm[ic-1][1][2]+cbm[ic-1][2][2]));
                        A[i][0] -=(cbm[ic][0][2]+bc[ic][0]*(cbm[ic][1][2]+cbm[ic][2][2]));
                        A[i][1] = (cbm[ic-1][2][3]+bc[ic-1][1]*(cbm[ic-1][1][2]*(4.0*log(r[ic])-
1)+2.0*cbm[ic-1][2][2]*(4.0*log(r[ic])+1.0)))*r[ic];
                        A[i][1] -=(cbm[ic][2][3]+bc[ic][1]*(cbm[ic][1][2]*(4.0*log(r[ic])-
1)+2.0*cbm[ic][2][2]*(4.0*log(r[ic])+1.0)))*r[ic];

                        A[i][i] =(cbm[ic-1][1][2]+2*cbm[ic-1][2][2])*r[ic] ;
                        A[i][i+1] = (cbm[ic-1][1][2]-2*cbm[ic-1][2][2])*pow(r[ic],-3);
                        A[i][i+2] = -(cbm[ic][1][2]+2*cbm[ic][2][2])*r[ic];
                        A[i][i+3] = -(cbm[ic][1][2]-2*cbm[ic][2][2])*pow(r[ic],-3);

                        //The w(r) conditions
                        A[i+1][0] =(bc[ic-1][0]-bc[ic][0])*r[ic]*log(r[ic]);
                        A[i+1][1] =(bc[ic-1][1]-bc[ic][1])*r[ic]*r[ic];


                        A[i+1][i] = r[ic];
                        A[i+1][i+1] = 1.0/r[ic];
                        A[i+1][i+2] = -r[ic];
                        A[i+1][i+3] = -1.0/r[ic];

                        //The thermal contributions
                        B[i] = (cbm[ic][1][2] + cbm[ic][2][2])*bc[ic][2]-(cbm[ic-1][1][2] + cbm[ic-
1][2][2])*bc[ic-1][2];
                        B[i] += sc[ic][2][4] - sc[ic-1][2][4];

                        B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic];
                }
                break;

        default:
```

```
                {
                      A[i][0] = sc[ic-1][2][0] - sc[ic][2][0];                    //eo terms
                      A[i][1] = ( sc[ic-1][2][1] - sc[ic][2][1] ) * r[ic];  //go terms

                      A[i + 1][0] = ( bc[ic-1][0] - bc[ic][0] ) * r[ic];
                      A[i + 1][1] = ( bc[ic-1][1] - bc[ic][1] ) * r[ic]*r[ic];

                      A[i][i] = sc[ic-1][2][2] * pow(r[ic], (L[ic-1] - 1) );
                      A[i][i+1] = sc[ic-1][2][3] * pow(r[ic], (-L[ic-1] - 1) );
                      A[i][i+2] = -sc[ic][2][2] * pow(r[ic], (L[ic] - 1) );
                      A[i][i+3] = -sc[ic][2][3] * pow(r[ic], (-L[ic] - 1) );

                      A[i+1][i] = pow(r[ic], L[ic-1] );
                      A[i+1][i+1] = pow(r[ic], -L[ic-1] );
                      A[i+1][i+2] = -pow(r[ic], L[ic] );
                      A[i+1][i+3] = -pow(r[ic], -L[ic] );

                      B[i] = sc[ic][2][4] - sc[ic-1][2][4];
                      B[i+1] = ( bc[ic][2] - bc[ic-1][2] ) * r[ic];
                }
                break;
          }
     }

//next lines stack the matrix elements associated with the pressure B.C.

     //The inner surface pressure conditions
     a=0;
     //set up terms for switching for the degenerate cases
     if(L[0]==1.0)
           a = 1;
     if(L[0]==2.0)
           a=2;
     switch(a)
     {
     case 1:
           {
                A[KK-2][0] = (cbm[0][0][2]+bc[0][0]*(cbm[0][1][2]*log(r[0])+cbm[0][2][2]*(log(r[0])+1)));
```

```
                A[KK-2][1] = ((cbm[0][1][2]+2*cbm[0][2][2])*bc[0][1]+cbm[0][2][3])*r[0];
                A[KK-2][2] = (cbm[0][1][2]+cbm[0][2][2]);
                A[KK-2][3] = (cbm[0][1][2]-cbm[0][2][2])*pow(r[0],-2);

                B[KK-2] = -load[1] - (cbm[0][1][2]*log(r[0])+cbm[0][2][2]*(log(r[0])+1))*bc[0][2] -
sc[0][2][4];
            }
            break;
      case 2:
            {
                A[KK-2][0] = (cbm[0][0][2]+bc[0][0]*(cbm[0][1][2]+cbm[0][2][2]));
                A[KK-2][1] = (cbm[0][1][2]*(4*log(r[0])-1)+2*cbm[0][2][2]*(4*log(r[0])+1))*bc[0][1]*r[0];
                A[KK-2][2] = (cbm[0][1][2]+2*cbm[0][2][2])*r[0];
                A[KK-2][3] = (cbm[0][1][2]-2*cbm[0][2][2])*pow(r[0],-3);

                B[KK-2] = -load[1] - (cbm[0][1][2]+cbm[0][2][2])*bc[0][2] - sc[0][2][4];
            }
            break;
      default:
            {
                A[KK-2][0] = sc[0][2][0];
                A[KK-2][1] = sc[0][2][1] * r[0];
                A[KK-2][2] = sc[0][2][2] * pow(r[0], (L[0] - 1) );
                A[KK-2][3] = sc[0][2][3] * pow(r[0], (-L[0] - 1) );

                B[KK-2] = -load[1] - sc[0][2][4];
            }
            break;
      }
      //The outer surface pressure conditions
      a=0;
      //set up terms for switching for the degenerate cases
      if(L[K-1]==1.0)
            a = 1;
      if(L[K-1]==2.0)
            a=2;
      switch(a)
      {
```

```
      case 1:
            {
                  A[KK-2][0] = (cbm[K-1][0][2]+bc[K-1][0]*(cbm[K-1][1][2]*log(r[K])+cbm[K-
1][2][2]*(log(r[K])+1)));
                  A[KK-2][1] = ((cbm[K-1][1][2]+2*cbm[K-1][2][2])*bc[K-1][1]+cbm[K-1][2][3])*r[K];
                  A[KK-2][2] = (cbm[K-1][1][2]+cbm[K-1][2][2]);
                  A[KK-2][3] = (cbm[K-1][1][2]-cbm[K-1][2][2])*pow(r[K],-2);

                  B[KK-2] = -load[0] - (cbm[K-1][1][2]*log(r[K])+cbm[K-1][2][2]*(log(r[K])+1))*bc[K-1][2]-
sc[K-1][2][4];
            }
            break;
      case 2:
            {
                  A[KK-2][0] = (cbm[K-1][0][2]+bc[K-1][0]*(cbm[K-1][1][2]+cbm[K-1][2][2]));
                  A[KK-2][1] = (cbm[K-1][1][2]*(4*log(r[K])-1)+2*cbm[K-1][2][2]*(4*log(r[K])+1))*bc[K-
1][1]*r[K];
                  A[KK-2][2] = (cbm[K-1][1][2]+2*cbm[K-1][2][2])*r[K];
                  A[KK-2][3] = (cbm[K-1][1][2]-2*cbm[K-1][2][2])*pow(r[K],-3);

                  B[KK-2] = -load[0] - (cbm[K-1][1][2]+cbm[K-1][2][2])*bc[K-1][2]- sc[K-1][2][4];
            }
            break;
      default:
            {
                  A[KK-1][0] = sc[K-1][2][0];
                  A[KK-1][1] = sc[K-1][2][1] * r[K];
                  A[KK-1][KK-2] = sc[K-1][2][2] * pow(r[K], ( L[K-1] - 1) );
                  A[KK-1][KK-1] = sc[K-1][2][3] * pow(r[K], ( -L[K-1] - 1) );

                  B[KK-1] = -load[0] - sc[K-1][2][4];
            }
            break;
      }
}
void output(double BB[], double th[], double rad[], double L[], double bc[][3], double Ep[], double *load,
double EpX[])
{
```

```
	int surface[2]={0,K-1},i,j,a;
	double radstrain, hoopstrain, axstrain, shearstrain, r;

	for (int ab=0;ab<6;ab++)
	{
		EpX[ab]=0.0;
		Ep[ab]=0.0;
	}

	for (j=0;j<2;j++)
	{
		i=surface[j];
		a=0;
		if(L[i]==1.0)
			a=1;
		if(L[i]==2.0)
			a=2;

		if(j==1)
			r=rad[i+1];
		else
			r=rad[i];

//************check bc[i][2] definitions - does it need to be mult by r???******************
		switch(a)
		{
		case 1:
			{
			//Strain Response x-y coordinates
				radstrain = (BB[2*i+2])-(BB[2*i+3]*pow(r,-
2))+bc[i][0]*BB[0]*(log(r)+1)+2*bc[i][1]*BB[1]*r+bc[i][2]*(log(r)+1);
				hoopstrain = (BB[2*i+2])+(BB[2*i+3]*pow(r,-
2))+bc[i][0]*BB[0]*log(r)+bc[i][1]*BB[1]*r+bc[i][2]*log(r);
				axstrain = BB[0];
				shearstrain = BB[1]*r;
			}
			break;
		case 2:
```

```
                {
                //Strain Response x-y coordinates
                      radstrain = (2*BB[2*i+2]*r)-(2*BB[2*i+3]*pow(r,-
3))+bc[i][0]*BB[0]+bc[i][1]*BB[1]*2*r*(4*log(r)+1)+bc[i][2];
                      hoopstrain = (BB[2*i+2]*r)+(BB[2*i+3]*pow(r,-
3))+bc[i][0]*BB[0]+bc[i][1]*BB[1]*r*(4*log(r)-1)+bc[i][2];
                      axstrain = BB[0];
                      shearstrain = BB[1]*r;
                }
                break;
          default:
                {
                //Strain Response x-y coordinates
                      radstrain = (L[i]*BB[2*i+2]*pow(r, L[i]-1))-(L[i]*BB[2*i+3]*pow(r,-L[i]-
1))+bc[i][0]*BB[0]+2*bc[i][1]*BB[1]*r+bc[i][2];
                      hoopstrain = BB[2*i+2]*pow(r,L[i]-1)+BB[2*i+3]*pow(r,-L[i]-
1)+bc[i][0]*BB[0]+bc[i][1]*BB[1]*r+bc[i][2];
                      axstrain = BB[0];
                      shearstrain = BB[1]*r;
                }
                break;
          }
          EpX[3*j]=axstrain;
          EpX[3*j+1]=hoopstrain;
          EpX[3*j+2]=shearstrain;
     }
}
```

12.4 Elastic_Solution.h

```
//Elastic_Solution.h

#include "Forward_Solution.h"
#include "matrix.h"
```

```
#include <iostream.h>
#include <fstream.h>
/*****************************/
/*routines internal to CCM*/
/*****************************/
int Elastic_Solution(double C[][7], double E[][10], double r[], double theta[], int mat[], double *load,
double Ep[], double EpX[], double B[]);
void Cbar(double C[][7], double theta[], int mat[], double E[][10], double cbm[][4][4], double et[][4]);
void stress(double cbm[][4][4], double et[][4], double L[], double bc[][3], double sc[][4][5]);
void mat_stack(double cbm[][4][4],double sc[][4][5], double L[], double bc[][3], double r[], double *load,
double (*A)[KK], double B[]);
void output(double BB[], double theta[], double rad[], double L[], double bc[][3], double Ep[], double
*load,double EpX[]);
```

12.5 Data_Input.cpp

```
// File for input and output of information to data files

#include "Data_Input.h"

ifstream inputfile, loadfile;             //Data file containing input data (material properties, geometry,
loads)
ofstream tracking, outputfile;                  //Data files for data output and intermediate values
char *input, *load;

void Input(double E[][10], double radii[], double theta[], int mat[], double Loads[numloads][4])
{
//    Change for input from User

      input = "Testinput.dat";
      load = "TestLOADS.dat";

      inputfile.open(input);
      loadfile.open(load);
```

```
      tracking.open("tracking.dat");            // Opening Data files for input, output and tracking of
values
      outputfile.open("output.dat");

      int i,j;
      double temp;

      for(i=0; i<matnum;i++)
      {
            for( j=0; j<10; j++)                //Reads in Elastic Moduli
(E1,E2,E3,G12,nu12,nu13,nu23,a1,a2,a3) IN ORDER!!!
                  inputfile >> E[i][j];
      }

      for(j=0; j<K+1; j++)                      //Reads in all surface radii
            inputfile >> radii[j];

      for(i=0; i<K; i++)                              //Reads in each ply angle (degrees) and converts to
radians
      {
            inputfile >> temp;
            theta[i] = temp * (Pi/180);
            inputfile >> mat[i];                //Reads in the material number information (0,1,2...)
      }

      for(i=0;i<numloads;i++)
      {                                                       // !!!!LOAD CONDITIONS!!!!
            for(j=0; j<4; j++)                          // Loads[0] = External Pressure
                  loadfile >> Loads[i][j];      // Loads[1] = Internal Pressure
      }                                                       // Loads[2] = Axial Load
                                                            // Loads[3] = Torsional Load
}

void Output_data(double E[][10],double radii[], double theta[], int mat[], double Loads[numloads][4],
double C[][7])
{
      tracking << "Tube properties entered:" << endl;
      for(int i=0;i<matnum;i++)
```

```
	{
		tracking << "\n\nMaterial #"<<i+1;
		for(int j=0;j<10;j++)
		{
			tracking << "\nE["<<j<<"] =\t"<<E[i][j];
		}
		tracking<<endl<<endl;
	}
	tracking <<"\nTube geometry entered:\n";

	for(i=0; i<K; i++)
	{
		tracking << "Ri and Ro of ply #" << i+1<< "\t"<<radii[i]<<"\t\t"<<radii[i+1]<<"\t\tAngle"<<
"\t"<< theta[i]*(180/Pi)<<"\tMaterial #\t"<<mat[i]<<endl;
	}

	for(i=0;i<numloads;i++)
	{
		tracking << "\nLoading Conditions #"<<i+1<<":" << endl;
		tracking << "Axial Load = \t" << Loads[i][2] << endl;
		tracking << "Torque = \t" << Loads[i][3] << endl;
		tracking << "Internal Pressure = \t" << Loads[i][1] << endl;
		tracking << "External Pressure = \t" << Loads[i][0] << endl<<endl;
	}
	for (i=0;i<matnum;i++)
	{
		tracking<<"\nCij Values for Material #"<<mat[i]<<endl;
		for (int j=0;j<7;j++)
			tracking << "\nC["<<j<<"] = "<<C[i][j];

		tracking<< endl<<endl;
	}
	tracking << endl;
	tracking.close();
}

// Calcualtes the C Matrix Values from the input data
```

```
void CMatrix(double E[][10], double C[][7]) //c matrix subroutine
{
      double v;

      for(int i=0;i<matnum;i++)
      {
            v = (1 - E[i][4] *(E[i][4] * E[i][1] / E[i][0] + 2 * E[i][6] * E[i][5] * E[i][2] / E[i][0]) -
E[i][5] * E[i][5] * E[i][2] / E[i][0] - E[i][6] * E[i][6] * E[i][2] / E[i][1]);

            C[i][0] = (1 - E[i][6] * E[i][6] * E[i][2] / E[i][1]) * E[i][0] / v;        // c11
            C[i][1] = (E[i][4] + E[i][5] * E[i][6] * E[i][2] / E[i][1]) * E[i][1] / v;     // c12
            C[i][2] = (E[i][5] + E[i][4] * E[i][6]) * E[i][2] / v;                    // c13
            C[i][3] = (1 - E[i][5] * E[i][5] * E[i][2] / E[i][0]) * E[i][1] / v;        // c22
            C[i][4] = (E[i][6] + E[i][4] * E[i][5] * E[i][1] / E[i][0]) * E[i][2] / v;     // c23
            C[i][5] = (1 - E[i][4] * E[i][4] * E[i][1] / E[i][0]) * E[i][2] / v;        // c33
            C[i][6] = E[i][3];                                                  // c66
      }
}
```

12.6 Data_Input.h

```
// Data Input.h


#include <iostream.h>
#include <iomanip.h>
#include "Forward_Solution.h"
#include <fstream.h>


void Input(double E[][10], double radii[], double theta[],int mat[], double Loads[numloads][4]);
void Output_data(double E[][10], double radii[], double theta[],int mat[], double Loads[numloads][4],
double C[][7]);
void CMatrix(double E[][10], double C[][7]);    //c matrix subroutine
```

12.7 Matrix.cpp

```
#include "Matrix.h"
#include <math.h>
#include <stdlib.h>
#include <iostream.h>
//Matrix.cpp
```

The following subroutines have been omitted due to copyright considerations. The appropriate subroutines can be found in Numerical Recipes (sections on LU Decomposition and Backsubstitution).

```
void SysScale(double *A,double *B,int n)
int LUDecomposition (double *A,int n,int *indx,int d)
void LUBackSub (double *A,int n,int *indx,double *B)
long Gauss(double *A,double *B,double *X,int n)
```

12.8 Matrix.h

```
//Matrix.h

#include <math.h>



void SysScale(double *A,double *B,int n);
int LUDecomposition (double *A,int n,int *indx,int d);
void LUBackSub (double *A,int n,int *indx,double *B);
long Gauss(double *A,double *B,double *X,int n)
```

13 Vita

Robert Hansbrough Carter was born on May 22, 1971 to Hans and Judy Carter in Richmond, Virginia. He graduated from Lloyd C. Bird High School in Chesterfield, Virginia in 1989, and began his studies at Virginia Tech that fall. Majoring in Materials Science and Engineering, he received his Bachelor's degree in December 1993, and began working towards his Master's degree in January of 1994. After successfully completing his Master's in 1996, he began his PhD work in 1997. He completed this work for his PhD in July, 2001.