High Speed, High Precision Processing of Dry Pre-Pregs for Composite Structures

Submitted in fulfilment of the requirements for the degree of Doctor of Engineering in Mechanical Engineering in the Faculty of Engineering and the Built Environment at the Durban University of Technology

Avinash Ramsaroop

AUGUST 2019

Supervisor: Prof. K. Kanny Date: 15 August 2019

ABSTRACT

In this study, a computational code was developed that was used to optimise the fibre orientation angles, layer thicknesses, number of fibre layers and the weight in fibre reinforced composites. In addition, an interface was created between the computational code and the robotic apparatus that performed the fibre placement.

Fibre reinforced composites are extremely versatile materials and may be tailor designed to suit various applications. However, the design techniques commonly associated with composite structures make them inadequate for industries with high production rates. Conventional design techniques have the disadvantage of numerous tedious and laborious matrix calculations. Also, there is uncertainty with assigning values to the input parameters for the equations used in the design process. In addition, conventional design techniques result in constant stiffness structures, that is, structures with the same fibre layup throughout. These disadvantages result in increased manufacturing costs as more material and labour, than necessary, are used.

This study presents a solution in the form of computational codes developed in Matlab. The codes are used to perform all the necessary matrix calculations easily and swiftly. Further, the uncertainty experienced with the input parameters, in conventional design techniques, is removed. The code is used to optimise the fibre orientation angles and layer thicknesses in a composite structure, as well as the number of fibre layers and the weight. In addition, it is able to create variable stiffness structures, that is, structures where the fibre layup varies throughout. The use of the code in the design process would decrease the design costs, as the design time is reduced, and decrease the material costs, as only the required amount of material is used. The developed codes were validated using examples from texts, finite element modelling and experimental methods.

The development of the computational codes created a problem with regards to the fibre layup process. Any process that employed manual placement of the fibres became inadequate as they proved to be extremely labour intensive and would result in

increased labour costs. Further, with manual placement of fibres, precise fibre orientation cannot be guaranteed. Therefore it was decided to use an automated system, such as Robotic Fibre Placement (RFP), to perform the fibre layup. Such a system was designed and built in-house in a previous study. In the current study, a Matlab code was developed as an interface between the developed computational code and the fibre placement system.

Other codes as well as graphical interfaces were developed in order to improve the interaction between the user and the codes.

DECLARATION

This thesis is being submitted to the Durban University of Technology for the degree of Doctor of Engineering in Mechanical Engineering. I declare that this work is my own, and any other work included that is not mine has been referenced accordingly. Further, this thesis has not been submitted before for any degree or examination to any other university or institution. Any publication has been in the form of conference and journal papers, which are listed in a later section.

_____ _____

Avinash Ramsaroop Date

Student No: 20429589

_____ _____

Prof. K. Kanny Date

Supervisor

ACKNOWLEDGEMENTS

The compiling of this thesis was a combination of long hours of research and perseverance, which would not have been possible without the assistance and support of key figures in both academic and personal capacities.

First, I would like to thank my supervisor, Professor Krishnan Kanny, for all his help during the course of my study. He was always there to offer advice, encouragement, guidance and inspiration.

I am grateful to Sherry Lore, from ATA Engineering in San Diego, California, USA, for her assistance with the licensing and operation of the IMAT software package used in this study. I would like to thank Advanced Dynamics Corporation for providing information on Robotic Fibre Placement in the form of papers and images.

My research would have proved extremely difficult had it not been for the financial support of KENTRON, through the HYSTOU 4 program, and funding from the National Research Foundation (NRF).

I want to express my gratitude to my parents, Paul and Indira Ramsaroop, for their love and support over the years, especially during my studies.

And finally, I want to extend my appreciation and gratitude to Karina Govender for the inspiration, love and support that she gave me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CONFERENCE AND JOURNAL PAPERS

Conference Papers:

1. Ramsaroop, A., Kanny, K. and Govender, P., "Flexible End-Effector for Fibre Placement", Second Robotics and Mechatronic Symposium, Bains Lodge, Bloemfontein, Free State, South Africa, November 2008.

2. Ramsaroop, A. and Kanny, K., "A Matlab Program to Assist in the Design Phase of Composite Structures and Components", 25th International Conference on CAD/CAM, CSIR International Convention Centre, Pretoria, South Africa, July 2010.

3. Ramsaroop, A. and Kanny, K., "Development of a Program to Aid the Design of Composite Structures", The Seventh International Conference on Engineering Computational Technology, Universidad Politécnica de Valencia, Valencia, Spain, September 2010.

4. Ramsaroop, A. and Kanny, K., "Design of Variable Stiffness Composites", Durban University of Technology Research Day, Durban University of Technology, Durban, South Africa, November 2011.

5. Ramsaroop, A. and Kanny, K., "Fibre Angle and Layer Thickness Optimisation of Fibre Reinforced Composites via Matlab", First International Conference on Composites, Biocomposites and Nanocomposites (ICCBN 2013), Durban University of Technology, Durban, South Africa, December 2013.

6. Ramsaroop, A. and Kanny, K., "Design Optimisation of Fibre Reinforced Composite Structures Using Matlab", Second International Conference on Composites, Biocomposites and Nanocomposites (ICCBN 2015), Durban University of Technology, Durban, South Africa, October 2015.

7. Ramsaroop, A. and Kanny, K., “The Design of Three Dimensional Fibre Reinforced Composite Structures Using Patran and Matlab”, Third International Conference on Composites, Biocomposites and Nanocomposites (ICCBN 2018), Nelson Mandela Bay Stadium, Port Elizabeth, South Africa, November 2018.

Journal Papers:

1. Ramsaroop, A. and Kanny, K., “Using Matlab to Design and Analyse Composite Laminates", Engineering, Vol. 2, 2010, pp. 904 – 916

1. INTRODUCTION

1.1 Overview

In this chapter the research problem and sub-problems are presented. First a brief history of the discovery and use of metals and composite materials is given. This is proceeded by the problems experienced with the design and fabrication of composite materials, and in particular, fibre reinforced structures. The shortcomings of the current or conventional design methods are discussed with emphasis on optimisation. Thereafter, the technique developed to overcome some of the design pitfalls is suggested. The chapter concludes with the discussion of an appropriate fibre layup technique that compliments the new design approach.

1.2 Metals and composites

Metals and metal alloys have been used for various applications throughout the centuries. According to Figure 1.1, gold was discovered around 10 000 BC, followed by copper in 5000 BC, which sparked off the Bronze and Iron Ages [1-2]. The fabrication of metal structures and components was initially difficult, but as processing techniques improved so too did the fabrication of these structures and components.

Figure 1.1 also shows that composites were used through the ages although in a very primitive form. In 1862 Alexander Parkes demonstrated the first man-made plastic at the Great International Exhibition in London [3]. The material, called Parkesine, was obtained from cellulose and could be heated and moulded into any shape. More plastics were developed and it was found that they were easier to mould into complex shapes than metal, and was just as durable. Plastic components were limited to low-end applications due to their relatively low strength properties; however, the development of composite materials changed the use and applications of plastics.

Composite materials have superior strength-to-weight and stiffness-to-weight ratios when compared to metals and metal alloys [4-6]. They also exhibit excellent corrosion and wear resistant properties, and have outstanding durability. Due to these and other characteristics, the use of composites has grown rapidly [1,5-7], and they are replacing

metal components in many applications such as sporting equipment, automotive body panels and chassis, and structural components in the aerospace industry.

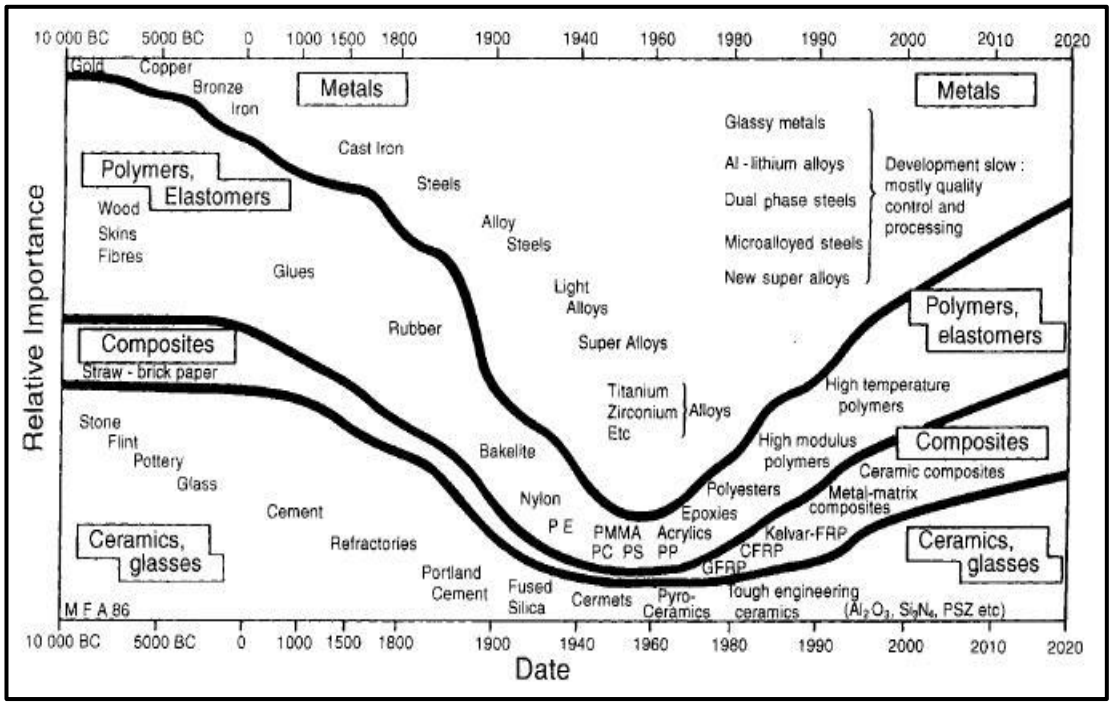


Figure 1.1: Approximate discovery dates of various materials through the ages [1]

Metals and their alloys are isotropic materials in that they exhibit the same mechanical properties in all three principle directions. On the other hand, with composite materials and, in particular, fibre reinforced composites, these properties may vary in each direction. This means that the mechanical properties may be superior in a specific direction when compared to the other two. Therefore fibre reinforced composite materials may be tailor designed to satisfy specific application requirements. For example, assume that a structure is loaded in one direction only as in the case of a towing bar between two vehicles. The bar requires excellent tensile strength characteristics in the longitudinal direction, and very little strength properties in the other two principle directions. Therefore the bar may be fabricated from a fibre reinforced composite material that is tailored to satisfy this loading condition.

Fibre reinforced composites have the ability to be moulded into complex-shaped parts [3]. Structures that were previously assembled from several smaller metal components

may now be fabricated into a large composite part. This saves on fabrication time as fewer components need to be manufactured, and assembly time is also reduced due to fewer parts. An example of this is the fully composite bus shell fabricated by North American Bus Industries [8]. They produced the shell as one large component instead of several smaller parts, which decreased fabrication and assembly costs and time. This made it possible to produce more bus shells in the same time when compared to using metal components.

It may be gathered from the above statements that fibre reinforced composites are extremely versatile materials, and may be used in the design and fabrication of various structures. However, in the fast-paced world that we live in, product demand is high and manufacturers strive to keep processing costs low [9-10]. Current design and fabrication techniques for composite materials have the disadvantage of being slow and laborious processes. These methods are adequate for once-off components or small batch productions where speed is not essential. However, in industries where high production rates are vital, as in the automotive sector, these low volume low speed techniques are unacceptable. Further, current methods are extremely labour intensive and this results in higher production costs. Hence, there is a need for an innovative design method as well as a dynamic fabrication technique that will enable high speed high precision processing of composite structures and that will decrease production costs. The current drawbacks as well as the proposed solution regarding the design and fabrication of fibre reinforced composite structures are discussed in turn.

1.3 Shortcomings of the design processes of fibre reinforced composites

Hooke's Law is commonly used to relate the stresses and strains experienced by a material. The governing equation for isotropic materials is straightforward and simple to use. However, in the case of fibre reinforced composites, the Hooke's Law equations are more complex. These equations are discussed in Chapter 3 and involve numerous matrix computations which prove to be quite tedious and laborious [11]. Manual calculations may vary from a few hours to several days or weeks depending on the intricacy of the design. This influences manufacturing costs as the design time forms a part of these costs. A longer design time will mean higher costs.

In order to demonstrate the tediousness and laboriousness of a typical composite component, consider the design of a passenger aircraft wing. There are several factors that need to be taken into account during the design process, such as weight, lift and drag forces, wind loading, thermal loading, fluid dynamics (fuel), and geometry (aspect ratio, taper ratio, thickness ratio, cross-section, shape, etc.). The next step is the selection of a material that is able to accommodate these factors. If the selected material is isotropic, such as aluminium, then the design process is completed and manufacturing may begin. However, if an orthotropic material, such as a composite, and in particular, a fibre reinforced composite, is selected, then additional calculations are required as the stress-strain relationships differ from that of an isotropic material.

An isotropic material's properties are the same in all three principal directions and therefore the stress-strain relationship is one-dimensional. This implies that regardless of orientation of the material, the moduli (tensile, flexural, shear) are constant. Conversely a fibre reinforced composite is orthotropic and exhibits different properties in the principal directions. In this type of material the applied loads are predominantly carried by the fibres, along each fibre's longitudinal axis. This means that orientation of the fibres play an important role in defining the properties of a fibre reinforced composite. Therefore further, more complex calculations need to be performed to determine the stress-strain relationships in this type of material.

The equations used in these computations are based on Hooke's Law for two-dimensional laminates. They require six input parameters or constraints, namely, the material constants (stiffness and shear moduli, and Poisson's ratios), material limits (ultimate tensile, compressive and shear stresses), applied loading conditions, number of fibre layers, fibre orientation angle of each layer, and thickness of each layer. The first three constraints are known at the onset of a design, and, for simplicity, will be referred to as material-loading parameters. The number of fibre layers, fibre orientation angle of each layer, and thickness of each layer, or the fibre layup parameters, are undefined at the start of the design process as there are no initial values for the fibre orientation angles and layer thicknesses. This leads to guesswork or

uncertainty in assigning values to these parameters, which becomes more problematic later on. However, experienced designers are able to make an educated assumption.

The Hooke's Law equations for two-dimensional laminates consists of various matrices and entails numerous matrix computations. The first of these matrices is the stiffness matrix where the entries are dependent on the material-loading parameters of the laminate. This matrix remains constant during the design process unless there is more than one type of reinforcement such as in a hybrid fibre reinforced composite. In this case, there would be a stiffness matrix for each fibre type, and this adds to the complexity and tediousness of the calculations.

The stiffness matrix caters for fibres orientated along the 0° direction. However, fibre reinforced composites have varying fibre orientation angles and therefore the stiffness matrix has to be modified or transformed to accommodate for the various fibre orientation angles. Therefore there has to be a transformed stiffness matrix for each fibre orientation angle, which adds to the volume of the calculations. The transformed stiffness matrices are then used to laboriously determine three other matrices. These are used in conjunction with the applied loading to calculate the mid-plane strains and curvatures in the laminate as a whole. The global and local stresses and strains may then be determined at any point in the structure. This proves to be a tedious exercise as the matrices vary for each point.

At this stage there is one step remaining in the design process and that is failure analysis via a suitable failure theory. Various failure theories or criterion are available and the choice of failure theory is dependent on the designer. If failure does occur, the design process has to be repeated, with adjusted fibre layup parameters, until there is no failure. This means that all the calculations have to be repeated, which is a laborious and tedious procedure, and further compounds the design process by prolonging a solution. As mentioned earlier, a longer design process increases manufacturing costs. Additionally, there is uncertainty on which of the fibre layup parameters to change and by what quantity. This in turn makes the design process become a trial and error exercise.

On the other hand, if no failure occurs, the structure could possibly be overdesigned (this does not include instances where overdesigning was desired such as in the case of implementing safety factors). This is also undesirable as it means that more fibres than necessary will be used and consequently more labour incurred in placing these extra fibres. This increases production costs and hence the cost of the part as a whole. To overcome this a redesign may be beneficial, however, this would mean repeating the Hooke's Law computations which in turn will increase design and manufacturing costs.

It may be surmised that design methods that employ conventional processes as above are tedious and laborious in nature. Due to the complexity of these manual calculations, designers commonly use a finite set of fibre orientation angles in their designs [12]. An example of such a set may be 0°, 30°, 45°, 60°, 90°, and their negative counterparts. Choosing from a finite set of angles may reduce the time taken and the tediousness of the manual calculations, but the chosen angles may not be the optimum ones for the structure. This may result in the composite structure being overdesigned and, as a result, the material and labour costs would increase.

Alternatively, designers may employ computational software to accomplish the numerous calculations involved instead of using manual methods. There are various computational packages available that assist with the complex matrix algebra such as Promal, The Laminator, eFunda and Composite Star. Although these packages may drastically reduce the tediousness as well as the computational time, they are still based on conventional design methods. The assigning of values to the fibre layup parameters is still an uncertainty, and, if failure occurs, the entire design process will still need to be repeated in a trial and error manner. A finite set of fibre orientation angles may be used but, as stated earlier, this may result in an overdesigned structure. It may be concluded that the use of currently available computational software in the design process does not result in optimised fibre reinforced structures.

A further disadvantage of conventional design methods is the creation of constant stiffness structures. Typically a fibre reinforced structure is designed according to the

region of highest stress concentration. This region is determined and the fibre layup parameters that overcome the highest stress is determined. The entire structure is then laid up according to this design and, since fibre orientation is directly related to stiffness, the stiffness of the structure is constant throughout. However, some regions in the structure may require less reinforcement (and stiffness) but, due to the constant stiffness design, results in more fibres being used than required. Consequently, the material and labour costs increases. The above statements demonstrate that conventional design techniques are inadequate to meet the demands of high production rate industries. Clearly a new design approach is needed that surpasses these disadvantages.

1.4 New design approach for fibre reinforced composites

In this study a design approach is put forward that eliminates the disadvantages discussed above, that is, the tedious and laborious matrix calculations, uncertain input parameters, non-optimised fibre layup parameters, overdesigned structures, constant stiffness structures, and high material and labour costs. This new approach involves the development of computational codes in Matlab to assist in the design process.

The computational code only requires the three known constraints as inputs, that is, the three material-loading parameters. Therefore the uncertainty associated with assigning values to the three unknown inputs, or the fibre layup parameters, is eliminated. The tediousness and laboriousness experienced when performing the matrix calculations is also removed as these are now executed automatically. In addition, the design time is reduced which will decrease the overall manufacturing costs. The new design approach optimises the fibre layup parameters according to the applied loading, and thus minimises the possibility of producing overdesigned structures. This in turn reduces the material and labour costs as only the precise amount of fibres are used.

Another advantage of the new design approach is the creation of variable stiffness structures instead of constant stiffness structures as was the case when using conventional design techniques. A variable stiffness structure has varying fibre layup

parameters throughout, and, as stiffness is directly related to these parameters, it has varying stiffness as well. Variable stiffness structures are both multi-strength, as its strength (stiffness) varies from point to point, and multi-directional, because there is more than one fibre orientation angle on a particular layer. These structures have been shown to have significant improvements in mechanical properties when compared to constant stiffness structures.

To illustrate the constant and variable stiffness concepts, and to distinguish between the fibre layup of the conventional methods and the new design approach, consider the square flat plate shown in Figure 1.2. It was loaded by a force P as shown, and was fixed at the holes. The plate, with its loading conditions, was input into a finite element package to determine the stress distribution induced by the force P.

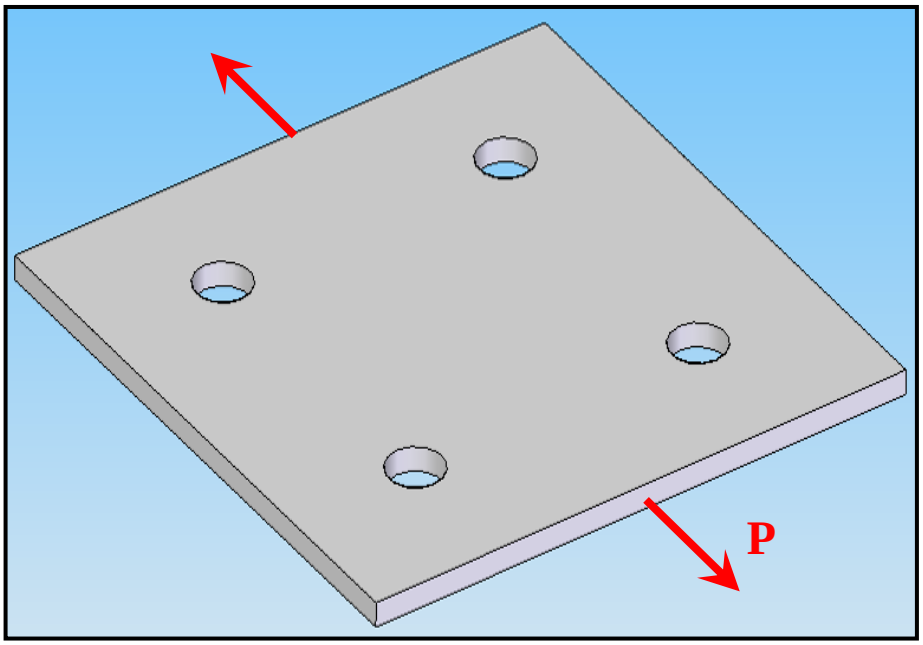


Figure 1.2: Flat plate with four holes loaded by force P

This stress distribution is shown in Figure 1.3 and it may be seen that there was a high stress concentration around the four holes while the rest of the plate was virtually unstressed. Intuitively, one would place extra reinforcement (fibres) only around the holes to increase the load carrying capability in these regions. However, with conventional methods, the fibre layup designed for the high stress concentration areas

(around the holes) would be used throughout the structure as illustrated in Figure 1.4(a). As a result the regions with very low stresses (white area in Figure 1.3) would have unnecessary fibre reinforcement and this in turn increases the material and labour costs of the part.

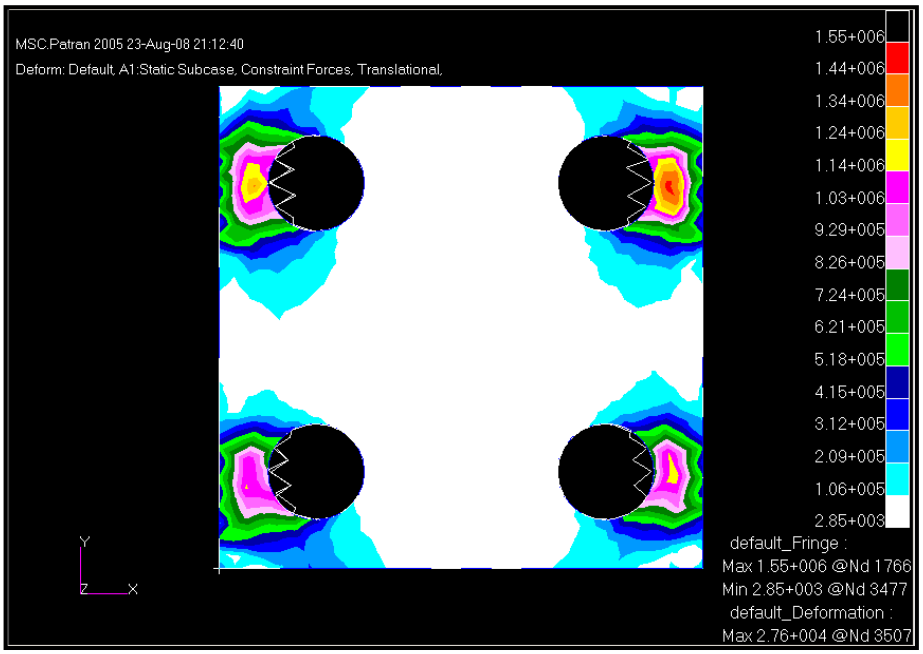


Figure 1.3: Stress distribution of the plate shown in Figure 1.2

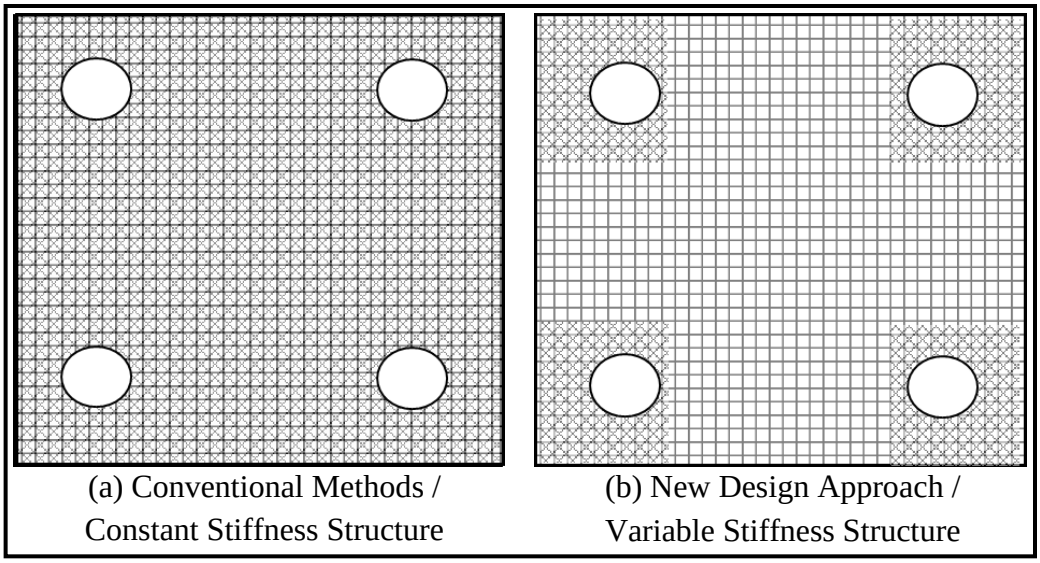


Figure 1.4: Fibre layup of the plate shown in Figure 1.2 using (a) conventional methods, and (b) the new design approach

On the other hand, using the developed Matlab code in the design process would result in a variable stiffness structure with a more intuitive fibre layup, as illustrated in Figure 1.4(b). Here only the regions of high stress concentrations (around the holes) have extra reinforcement. Therefore, a cost saving would be realised when compared to the conventional layup case (Figure 1.4(a)) as there were no unnecessary fibres used. Consequently, there would be a weight reduction as less fibres were used as well as a decrease in labour costs. These savings reduce the overall cost of the part.

The development of the Matlab design code is the first step in achieving high speed high precision processing of composite structures. The next step would be to pair the new design approach with the correct fibre placement method.

1.5 Choosing the correct fibre placement method

In composite material processing the factor that takes the most time is fibre orientation and placement. Shortening this time will yield higher production rates. As discussed above, designers usually use a finite set of fibre orientation angles to simplify calculations. A further reason for the finite set is to reduce the time for the fibre layup process, which is usually performed manually. A design with an array of fibre orientation angles that are not from a finite set will make the fibre layup process extremely labour intensive. As each angle is not predetermined, it will have to be measured out during fibre placement thus increasing the fibre layup time. In the case of a standardised set of angles, the fibre orientations are predetermined and this makes fibre placement easier as templates may be used. The result is a faster fibre layup time and less effort being required to achieve the desired fibre orientations as well as lower labour costs compared to the case of using non-standardised fibre orientation angles.

However, the developed design code does not use a predetermined set of fibre orientation angles and this will make the fibre layup process extremely labour intensive as templates cannot be used. Further, manually placing fibres will make precise fibre orientation difficult [12] as well as possibly shifting or disorientating the placed fibres in the underlying layers. This implies that any manual fibre placement techniques are now inadequate.

The logical solution would be to use an automated fibre placement system to perform the fibre layup. This will keep the labour cost and material scrap rate low, while increasing repeatability and production rates. Further, control over precise fibre orientation and placement is more attainable with an automated fibre placement system than with a manual one. Robotic Fibre Placement (RFP) is the technique that fulfils these requirements. This process uses a robotic arm together with a special or purpose-made end-effector to place dry or wet fibres in the desired orientation.

A robotic end-effector was designed and constructed in another project. In the current study, another Matlab code was developed to interface between the design code (discussed above) and the programming code for the robotic arm and end-effector. This new code converts the results from the design code to input code files for the robotic arm and end-effector. These files may then be loaded into the robotic arm controller to execute the fibre layup process.

The developed design code together with the RFP process enables the high speed high precision processing of dry pre-pregs for composite structures. Other codes were developed to enable better interaction with the user and these are discussed in the subsequent chapters.

1.6 Summary

The design and fibre layup processes commonly associated with fibre reinforced composite structures make them inadequate for industries with high production rates. Conventional design techniques have the disadvantage of several tedious and laborious matrix calculations, the use of finite sets of fibre orientation angles, overdesigned structures, non-optimised fibre layup parameters, uncertain input parameters for Hooke's Law, constant stiffness structures, and, high material and labour costs.

The solution to these problems, and the primary objective of this research work, was a new design approach that used a computational code to optimise the design process, and create multi-strength, multi-directional structures. The developed code performed all the necessary matrix calculations easily and swiftly as well as removed the

uncertainty experienced with conventional design methods. It eliminated the need for finite sets of fibre orientation angles and subsequently reduced the design process costs as the design time was reduced. Further, the code optimised the fibre orientation angles, layer thicknesses and weight of a composite structure, and was able to create variable stiffness (multi-strength, multi-directional) structures. Consequently, the material and labour costs were reduced as well as installation costs.

The development of the new design approach posed a new problem with regards to the fibre layup process. Any manual fibre placement method became inadequate as they proved to be extremely labour intensive and resulted in increased labour costs. Further, with manual placement of fibres, precise fibre orientation cannot be guaranteed. Therefore it was decided to use Robotic Fibre Placement (RFP), which is an automated system, to perform the fibre layup. A fibre placement end-effector was designed and constructed in a previous study. The secondary objective of the current research was to develop an interface between the design code and this end-effector.

The following chapter discusses the research associated with this study. Studies put forth by other researchers regarding the optimisation of constant stiffness and variable stiffness structures are examined. Thereafter the manual and automated fibre layup processes are investigated, with emphasis on the (RFP) process. The resources utilised to achieve the objectives of this study, namely, the Matlab environment, Hooke's Law equations, micromechanics equations, finite element modelling and experimental procedures are described in Chapter 3. Chapters 4 to 7 discusses the development of the various Matlab codes, while Chapter 8 concludes this study.

2. LITERATURE REVIEW

2.1 Overview

The discussion in this chapter begins with the description of composite materials, its constituents and some applications of fibre reinforced composites. It was emphasised in the opening chapter that fibre orientation was an important aspect in the design process, and that fibre optimisation was vital to avoid failure and overdesign. Therefore studies conducted by other researchers into optimisation of the fibre orientation angles and layer thicknesses are explored in this chapter. A further discussion in Chapter 1 centred on variable stiffness structures. In this chapter, work done in this area is examined to show the improvements that variable stiffness structures have over constant stiffness structures. The chapter is concluded with the discussion on the various manual and automated fibre layup processes that are available, and the studies conducted on the Robotic Fibre Placement (RFP) process.

2.2 Description of composite materials

A composite material is produced by the combination of two or more materials that have different properties [13-14]. This could mean that a composite material is similar to an alloy; however, this is not the case. In an alloy, the different constituents dissolve or blend into each other to form the final material, and these constituents are indistinguishable from each other in the final product. In a composite material, the different constituents do not dissolve or blend into each other, and can be easily distinguished from one another in the final product [14-15]. The different constituents act together giving the overall composite material enhanced properties that are superior to those of the separate constituent parts.

As mentioned in the previous chapter, metals and their alloys are isotropic materials. Composites, on the other hand, are anisotropic or have different properties in each direction. This infers that composites can be tailor designed to satisfy specific application requirements. They also have the ability to be easily formed into complex-shaped components. This allows for structures to be fabricated as one large part instead of several smaller pieces.

A composite material consists of two basic elements, namely, the matrix and the reinforcement [13-15]. The reinforcement is usually the constituent that provides the composite material with its strength. It can exist in particulate or fibre form, or a combination of these (hybrid structure) [16]. The matrix is the constituent that surrounds and binds the reinforcement together [14-15]. Further, it serves as protection against damage and aids in the transfer of the applied loads to the reinforcement.

The most commonly used composite material for modern applications is fibre reinforced composites [17-18]. These composites are usually a laminated structure containing layers of unidirectional, random or woven fibre reinforcement embedded in a polymer matrix. Figure 2.1 shows a cross-section of a fibre reinforced composite structure where the circles are the fibres and the surrounding portion is the matrix.

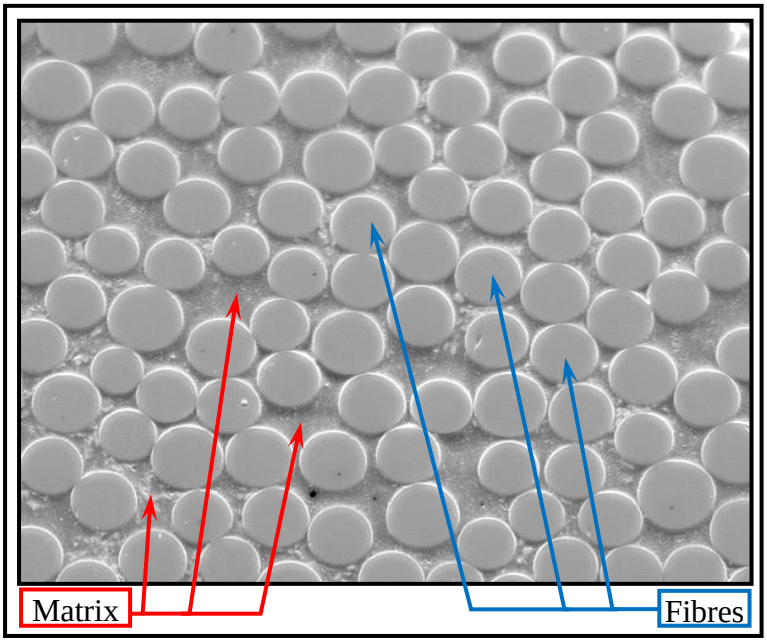


Figure 2.1: Electron Micrograph showing the cross-section of a fibre reinforced composite [19]

2.3 Constituents of fibre reinforced composites

The properties of a fibre reinforced composite depend on the choice of fibre reinforcement and matrix materials [20]. Some types of matrix materials include metals, ceramics and carbon [14-15,21]. These materials are used in highly specialised

applications. Ceramics are used for high temperature purposes while carbon is utilised in applications where there is high friction and wear. The most commonly used matrix is a polymer resin which is either thermosetting or thermoplastic [14-15].

A thermosetting polymer is in liquid form when being prepared, and hardens and becomes rigid upon curing [14,22]. The curing process is irreversible and therefore these materials cannot be reshaped after they are set. These plastics have good corrosion and wear resistant properties. The most common thermosetting matrix systems are polyester and epoxy resins [15]. A thermoplastic or thermosoftening polymer is rigid at low temperatures but softens or becomes glassy when it is heated [14,23]. Their use is less than that of thermosetting polymers but they have better fracture toughness properties, longer shelf life, and better capacity for recycling [14].

The other significant component in fibre reinforced composites is the fibre itself. This may be divided into two categories, namely, natural and synthetic. Natural fibres are obtained from nature, and incorporate both plant and animal fibres [24], however, plant fibres are the ones usually used in composite materials. Natural fibres are subjected to a pre-processing technique to allow them to be utilised in fibre reinforced composites [25]. Figure 2.2 shows the main classifications of plant fibres. Common natural fibres used in composite materials include hemp, kenaf, sisal and jute, however, banana, coconut and palm fibres are becoming increasingly popular.

Synthetic fibres are man-made entities and are usually produced via extrusion. Nylon, Olefin, Acrylic and Polyester are examples of common synthetic fibres [26]. Specialty synthetic fibres include Spandex, Vinalon, Aramid, Carbon, Glass, etc. The most commonly used synthetic fibres in composite materials are carbon, aramid (Kevlar) and glass. These fibres are shown in Figure 2.3 in their woven mat form.

Carbon fibres are expensive but provide the best combination of high strength, high stiffness (high modulus), low thermal expansion and low density [28-29]. They do, however, have low elongation which reduces their flexibility. These fibres have many

applications in the aerospace and automotive industries [30]. Other applications include sailboats, musical instruments, and sporting equipment.

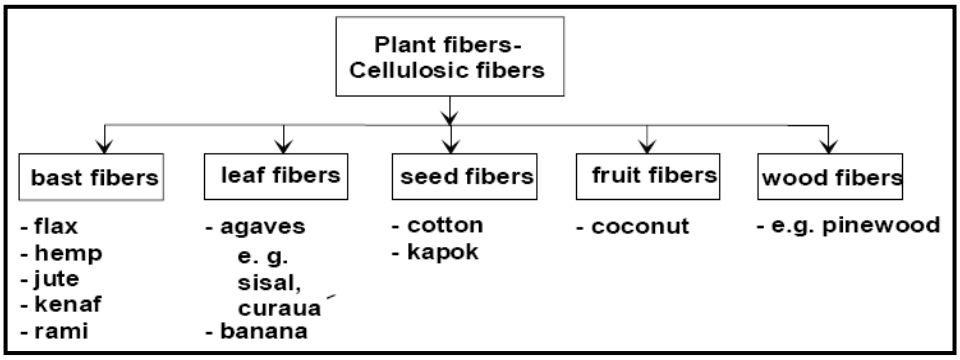


Figure 2.2: Classification of plant fibres [25]

Figure 2.3: From left to right: glass, carbon, aramid woven fibre mats [27]

Kevlar is a type of aramid fibre and is used for high-performance composite applications where light weight, high strength and stiffness, damage resistance, and resistance to fatigue and impact are important [31-33]. The major industrial applications for aramid fibres include flame-resistant clothing and helmets, body armour, boat hull material, sporting equipment, and loudspeaker diaphragms [32-33].

Glass fibres are the most widely used fibre reinforcement and the lowest in cost [34-35]. They have a lower strength and modulus but a higher density and are less brittle when compared to carbon and aramid fibres. There are many different types of glass fibres that are used in composite materials [36-37]. E-glass is the most commonly used and has a relatively low cost. It has good electrical insulating, water resistant, tensile and compressive strength, and stiffness properties. However, it exhibits poor impact resistance. S-glass has an extra high strength-to-weight ratio and is more expensive than E-glass. It is used primarily for military and aerospace applications. C-glass and A-glass have excellent chemical resistance, and D-glass has superior electrical properties and is used in high performance electronic applications.

2.4 Types of fibre reinforced composites

There are generally two types of fibre reinforced composites, namely, naturally occurring and man-made. Wood is an example of a naturally occurring fibre reinforced composite structure [13-14]. It has long fibres of cellulose (reinforcement) that are held together by a weaker substance called lignin (matrix). Cotton and linen also contain cellulose, but the binding strength of lignin is what makes a piece of timber stronger than a bundle of cotton fibres.

The earliest man-made fibre reinforced composite was mud/straw bricks [13-14] used around 10 000 BC [1]. Dried mud could make a strong wall where all the applied forces were compressive; however, the mud broke easily in applications where tensile forces were present. A piece of straw, on the other hand, has great strength when it is stretched but almost no strength when it is crumpled up. Therefore pieces of straw (reinforcement) were embedded in blocks of mud (matrix) to produce the mud/straw bricks. These bricks made excellent building materials as they had both good compressive strength and good tensile strength.

Concrete is an example of a particulate filled composite material [14-15] where the particulate filler is small stones or gravel, which is bound together by cement that acts as the matrix. The resulting material has excellent compressive strength but very poor tensile properties. Therefore steel rods are used as a type of fibre reinforcement to

improve the tensile strength of the concrete. The rods have excellent tensile properties with low compressive capabilities. However, the combination of the steel rods in the concrete results in a structure that has both good compressive and tensile properties. The material produced is reinforced concrete and is an example of a hybrid composite as it has more than one type of reinforcement, namely, fibre and particulate.

2.5 Applications of fibre reinforced composites

The aerospace industry has been using fibre reinforced composites for the past 40-50 years [38]. Figure 2.4 shows the increasing use of composite materials at Airbus over the decades, as well as the components fabricated from these materials. At first composites were used only in secondary structures, but as knowledge and development of these materials improved, their utilisation in primary structures increased. Airbus uses fibre reinforced composite materials in the tail box of their A310 [40] and in the fuselage of the A320 [41]. The A380 has its central wing box fabricated from carbon fibre reinforced composites [42] while other fibre reinforced composites are used extensively in the wings, fuselage sections, tail surfaces, and doors. Over the decades Airbus has increased the weight of composite structures in their aircraft from less than 5% in the A300 to more than 20% in the A380 to around 53% in the A350. Composite structures have also been extensively used by other aircraft manufacturers such as Boeing in their Boeing 787 Dreamliner [43-44]. Although high precision is required in the aerospace industry, it does not encompass high speed processing.

Composite materials are also used in the automobile industry. Natural fibre reinforced composites are used in the interior of vehicles in door panels, cabin linings, etc. Figure 2.5 shows the components fabricated by Mercedes Benz from natural fibre composites for their E-class model [45]. A 20% weight reduction and an increase in mechanical properties were achieved with the use of natural fibre composite components when compared to their metal counterparts [25]. The components manufactured from these composite materials do not form any part of the critical structure of the vehicle. Therefore, although high speed processing may be possible, high precision is not required. This is further emphasised by the use of random fibre mats.

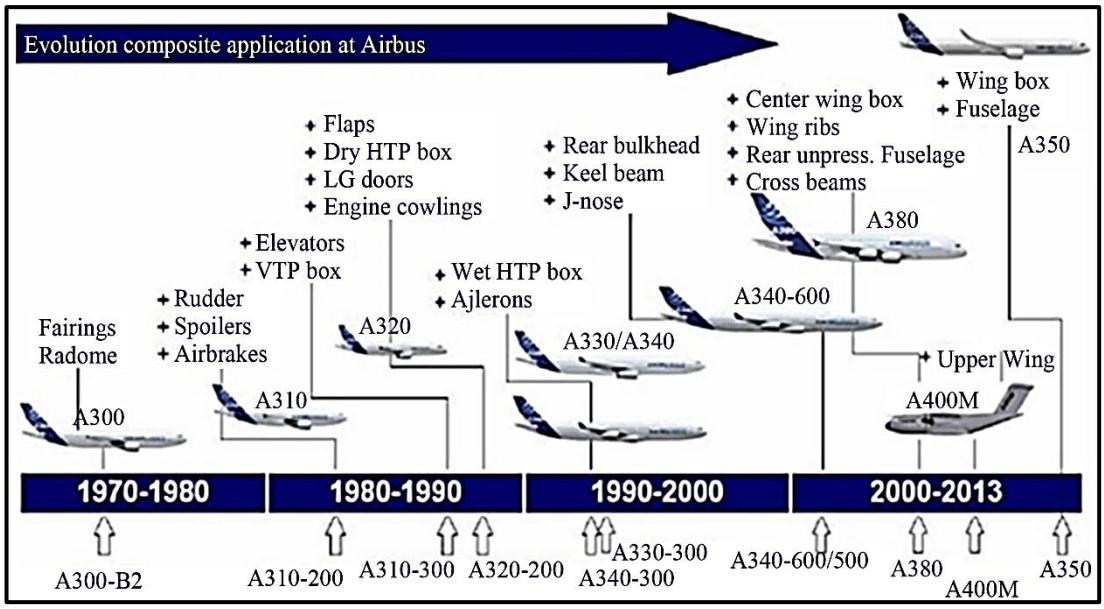


Figure 2.4: Use of composite structures at Airbus [39]

Figure 2.5: Natural fibre reinforced composite components in the Mercedes Benz E-Class series [45]

The automotive industry also uses synthetic fibre reinforced composites for components such as bumpers, spoilers and body panels. The Chevrolet Corvette was the first mass produced vehicle to use fibre reinforced composites for its body panels [46]. This is a further example of high speed low precision processing as random fibre mats are used. Formula 1 race cars have their chassis and body panels constructed from carbon fibre reinforced composites [47]. Great precision is required for these components as they are load bearing structures. However, speed is not an important factor because these are low volume custom-built products.

Applications in the construction industry include non-structural gratings, claddings, and full structural systems such as framing for industrial supports, buildings, long span roof structures, tanks, bridge components, and complete bridge systems [45-49]. Fibre reinforced composite pedestrian bridges have been constructed and installed in Delft, Netherlands [50] and over the Manzaneres River in Madrid, Spain [51]. Both bridges are 44m in length and are shown in Figures 2.6 and 2.7, respectively. Various other fibre reinforced pedestrian bridges have been built in Europe [52], Canada [53] and USA [54]. In the construction industry certain applications may require some degree of precision, however, in these applications processing speed is not a crucial factor as the products are either once-off or low volume. Applications such as roof structures, tiles, etc. require medium to high production rates but low precision.

Fibre reinforced composites are also used to strengthen existing bridge structures by wrapping these materials around bridge columns and beams [55]. Fibrwrap Construction Inc. of Los Angeles, California, USA used this wrapping technique with glass and aramid reinforced composites for a seismic retrofit of the Arroyo Seco Bridge located in Pasadena, California, USA [56]. The composite jackets provided strength comparable to that of full steel jackets. This type of application does not require high speed processing as it is a once-off process.

Other applications include swimming pools [57], satellite components, covers, flexible and rigid ducting, golf club shafts and other sporting equipment, yacht and boat hulls, and many more. They are also used in vibration damping applications and dentistry.

Figure 2.6: Glass fibre reinforced pedestrian bridge in Delft, Netherlands [50]

Figure 2.7: Carbon fibre reinforced pedestrian bridge in Madrid, Spain [51]

In the above mentioned applications, as well as many others, there are varying degrees of production precision and speed that is required. Some have high precision and low speed while others have high speed and low precision. There are very few, if any, that have high precision and high speed production rates. This is mainly due to the elevated material and labour costs involved in running such manufacturing facilities. Further, the design cost is also considered to be part of the manufacturing costs. The longer the design time, the higher the overall manufacturing costs.

Therefore the establishment of high speed, high precision processing begins with an enhanced design process that reduces the design time and optimises the fibre layup parameters, in particular, the fibre orientation angle of each layer and the thickness of each layer. Optimisation of these parameters will reduce material usage and thereby decrease material and labour costs, and hence the overall manufacturing costs. Further, Fares et al. [58] stated that using the fibre orientation angle for optimal design methods was more effective that using the layer thicknesses, however, using both the fibre orientation angle and layer thickness for optimisation was the most efficient. Research conducted into the optimisation of these parameters is presented below.

2.6 Optimisation of the fibre orientation angle and layer thickness

The fibre orientation angle directly influences the strength and stiffness properties of a composite structure. A study by Bakir and Hashem [59] showed that the tensile and impact properties of a fibre reinforced composite varied with varying fibre orientation angle. Hence optimising the fibre orientation angle will give the best strength and stiffness properties. Kim et al [60] conducted a study where they considered the fibre orientation angles as design variables and subjected fibre reinforced composites to in-plane loadings to determine an efficient approach for strength optimisation. Kim et al [61-63] developed a software package using C++ for the optimisation of the fibre orientation angle. This package allowed the user to input data easily and graphically displayed the outputs. Park et al [64] developed genetic algorithms to optimise the design of symmetric laminates subjected to various loadings and boundary conditions. Other studies [65-69] have optimised the fibre orientation angle for their specific applications.

However, according to Akbulut and Sonmez [12], designers traditionally use a finite set of fibre orientation angles in their designs. Therefore, in an optimisation process, the fibre orientation angle is converted to the nearest discrete value in the set. This, however, does not result in the optimum fibre orientation angle being chosen. Hence the need for a more robust fibre orientation angle optimisation process.

The layer thickness affects the stiffness of the composite and has a direct influence on the weight. A correct optimisation process will result in a low-weight structure with low manufacturing costs. There are numerous optimisation processes that have been put forward by researchers. In these studies, some of which are discussed below, researchers have developed genetic algorithms and computational methods to achieve their goals.

Krikanov [70] developed a graphical method to find the optimum layer thickness in composite pressure vessels, although, in this work, the fibre orientation angles were fixed. However, other researchers have varied both fibre orientation angle and layer thickness in their optimisation work. Walker and Smith [71] performed a study where they combined genetic algorithms and finite element methods to minimise the mass and deflection of fibre reinforced structures. Both the fibre orientation angle and layer thickness were varied, however, these values were selected from a discrete set.

Paluch et al [72] used finite element analysis and a genetic algorithm to optimise composite structures. In their work both the layer thickness and the fibre orientation angle were varied. They concluded that using this method leads to significant improvement in weight saving.

Bruyneel [73] proposed a general procedure for the optimal design of fibre reinforced composites. In his study he varied both the layer thickness and fibre orientation angle. It was numerically shown that the proposed procedure efficiently designed composite structures for strength and stiffness criteria. However, he concluded that the solution was not the global optimum.

Omkar et al [74-76] presented a generic model for the design optimisation of composite structures in order to reduce the overall weight. The design variables included the number of layers, thickness of each layer, and the fibre orientation angles. They reported that their model performed satisfactorily and resulted in composite designs with superior properties and significant weight savings. However, they limited their research to simple beam structures and each layer thickness was kept constant, that is, the thickness did not vary across the layer.

Barakat and Abu-Farsakh [5] conducted research where they varied both the fibre orientation angle and layer thickness in order to optimise boron/epoxy and carbon\epoxy composite laminates with regards to strength under in-plane loading. They concluded that the minimum thickness and weight for the laminates were obtained at the fibre orientation angle that yielded the maximum stresses.

Akbulut et al [77] performed a study on the optimisation of composite laminates by minimising the laminate thickness. These laminates were subjected to both in-plane and out-of-plane loadings. They developed an algorithm to perform the minimisation and the results were reliable and consistent. It was stated that the design process for composite materials should not be based on intuition or experience. This is because optimal design can be counter intuitive with certain loading conditions.

Farshi et al [78] presented an algorithm for the minimum thickness design of composite laminates in order to satisfy a fundamental frequency constraint. The algorithm designed symmetric laminates and there were no limitations in the number of layers or fibre orientation angles. They compared their findings with other published results and determined that their method was practical and efficient.

In another study, Farshi et al [79] proposed a method for the weight optimisation of multi-layered symmetric fibre reinforced composite plates. In their method the plate thickness was built up via sequential addition of new layers. The optimisation procedure was performed in two stages. In the first stage the fibre orientation angles were the design variables, while the thicknesses were varied in the second stage. They

reported that the proposed method was efficient. The current study follows the same principle in that the fibre orientation angles are optimised first and thereafter the layer thicknesses.

The above studies demonstrate the importance and benefits of fibre orientation angle and thickness optimisation processes, especially with regard to weight reduction. However, these research works were based on conventional design techniques that result in constant stiffness structures. Much research has been conducted on these types of structures. Ghiasi et al [80] conducted a review on constant stiffness design methods where they compared numerous techniques put forth by several researchers (139 in total). They discovered some methods were faster than others in determining a solution but more expensive, while others were moderately fast with minimal costs. Some processes were designed for specific laminates, loading conditions and/or situations. They concluded that hybrid techniques (combination of two or more methods) gave the best results as these benefitted from all the advantages of their constituent methods.

It is evident from the above discussion that constant stiffness design methods have been extensively researched. One of the major contributors to this type of design style is conventional fibre layup techniques, which mostly consist of manual fibre layup. However, with new advancements in the fibre layup process, interest into research of design methods involving variable stiffness structures have been sparked [81-82]. The current study is based on variable stiffness design methods.

2.7 The variable stiffness design concept

Ghiasi et al [83] conducted a review of the various methods of implementing the variable stiffness design approach that were available. They stated that this concept bore high design costs due to the large number of design variables required, which lead to a need for higher computational resources. They formulated a rank or order of methods that was used for the variable stiffness design approach. However, choosing the appropriate method proved difficult and resulted in more complexity of the design problems.

Setoodeh et al [84-85] compared variable stiffness and constant stiffness laminate design. They found that the laminate stiffness was significantly increased when using the variable stiffness design approach. More research conducted by Setoodeh et al [86] focused on the design of variable stiffness composite panels in an attempt to maximise the buckling load. They followed an approach by Gürdal et al [87] where variable stiffness laminates were designed using continuous curvilinear fibre paths. If the fibre paths were curvilinear rather than straight, the fibre orientation varied and thus the stiffness properties differed from one point to another. Setoodeh et al [86] compared the buckling loads of variable and constant stiffness plates and found that there was significant improvement in properties in the variable stiffness case. However, they experienced complexity in maintaining uniform fibre paths. Further the influence of extensional stiffness and flexural stiffness distribution could not be investigated individually.

Gürdal et al [88] conducted a study into the buckling response of variable stiffness laminates. They researched two scenarios, namely, varying the stiffness parallel to the loading and varying the stiffness perpendicular to the loading. The latter case was found to provide a higher degree of improvement as the applied loads were redistributed more effectively.

Research conducted by Lopes et al [89] entailed the analyses of buckling, progressive damage and failure of composite panels with constant and variable stiffness, and with and without central cut-outs. They discovered that the variable stiffness laminates where the fibre tows were allowed to overlap produced the highest improvements in retardation of damage initiation as well as increasing the structural strength. It was reported that the benefit was higher than 55%.

Lopes et al [90] performed another study where the buckling and first-ply failure of variable stiffness laminates versus constant stiffness laminates loaded in compression were examined. They found that the quality and structural improvements of variable stiffness over constant stiffness laminates were possible. Their results showed that the buckling and first-ply failure loads were increased.

The studies discussed above dealt with the optimisation of the buckling response due to compressive axial loading. There has been more work done on buckling analysis with this type of loading [91-99]. Other researchers performed optimisation studies on buckling of variable stiffness composite cylinders subjected to bending loads. Blom et al [100] reported up to 17% increase in buckling load when compared to conventional laminated cylinders. However, the laminates studied had a constant thickness. A study by Rouhi et al [101] revealed that the radius did not have a significant effect on improving the buckling load of variable stiffness cylinders. Their results showed that an increase of up to 38.5% in buckling load was obtainable for low length-to-radius ratios. Other studies [102-103] also reported improvements in buckling loads.

There has been optimisation research conducted on the frequency response and vibration analysis of variable stiffness composite structures. Abdalla et al [104] performed a study on the maximisation of the natural frequency of composite panels. They postulated that variable stiffness laminates could increase the fundamental frequency of composite structures. Rectangular laminates were designed and the numerical results showed that there was a significant increase in the optimal fundamental frequency of the variable stiffness panels when compared to constant stiffness panels.

Akhavan et al [105] used finite element analysis and third order shear deformation theory in an investigation into the mode shapes and natural frequencies of vibration of fibre reinforced composite laminates. It was found that the variable stiffness design approach could significantly change the mode shapes. Further, the natural frequencies may significantly be increased or decreased. Blom et al. [106] also researched the maximum fundamental frequency of composites using the variable stiffness concept. Other vibration analysis studies [107-109] have been conducted.

It may be surmised from the above work that variable stiffness structures are superior to constant stiffness structures. There were significant improvements reported with regards to the mechanical properties, especially with respect to buckling and vibration analysis. However, it was suggested by Lopes et al. [90] that even more significant

improvements in mechanical properties could be achieved if the laminates were optimised for strength rather than buckling.

A study by Khani et al. [110] investigated the design of variable stiffness laminates for maximum strength. They designed a square plate with a central hole under tensile loading. The design variables were the lamination parameters (fibre orientation angles and number of layers). The numerical results showed that the variable stiffness laminate strength-optimal design was approximately three times better than the quasi-isotropic stiffness-optimal design. The total laminate thickness was fixed.

The current study presents a technique that designs and optimises variable stiffness composites for maximum strength and minimum weight. It allows for the unrestricted optimisation (values are not from a finite set) of the fibre orientation angle and layer thickness, and consequently, the number of fibre layers. The technique is applicable to both two-dimensional and three-dimensional structures.

With the advent of the variable stiffness design method, a new problem arises with regards to the fibre layup process. It is important to correctly align the fibres according to the designed orientation angles in order to impart the desired mechanical properties to the component. Further, the correct process should reduce the fabrication time and reduce material and labour costs. In light of this, the current fibre layup methods that are used in constant stiffness structures need to be evaluated to determine whether they are adequate for fabrication of variable stiffness structures. Manual and automated layup processes were examined.

2.8 Fibre layup processes for fibre reinforced composites

The popular conventional methods for the fabrication of fibre reinforced composites include hand lay-up, manual tape-laying, Vacuum Assisted Resin Infusion Moulding (VARIM), Resin Transfer Moulding (RTM), Vacuum Assisted Resin Transfer Moulding (VARTM), and filament winding. Each technique has its advantages and disadvantages with respect to the fibre layup process.

Manual hand lay-up, shown in Figure 2.8, and manual tape-laying are the most common fabrication methods used [11,111]. In these methods fibres are laid out manually and resin is painted over them. More fibres and resin are added until the desired specifications are achieved. These two methods are fairly quick to perform and are not very labour intensive. However, they are inefficient as resin rich areas may be formed. Further, repeatability is very low, that is, a fabricated part cannot be easily reproduced with the same properties as the previous one. The material scrap rate is high which means that there is more material being used than required and therefore there is unnecessary wastage. The production rates using these techniques are satisfactory but this is overpowered by lower and inconsistent material properties. The precise control of fibre orientation is possible but this is extremely difficult and increases labour costs.

Figure 2.8: Manual hand lay-up of fibres [112]

The VARIM, RTM and VARTM techniques, schematically shown in Figures 2.9, 2.10 and 2.11, respectively, prove to be more efficient than the above methods. There is a more even distribution of resin, lower material scrap rate and more precise control of fibre orientation [111,115]. In these methods all the fibres are laid out and resin is then either infused or injected through the fibres. The VARIM, RTM and VARTM

techniques have good repeatability but they prove to be rather labour intensive when it comes to the fibre layup. Further, production rates are low as the setup steps are quite time consuming.

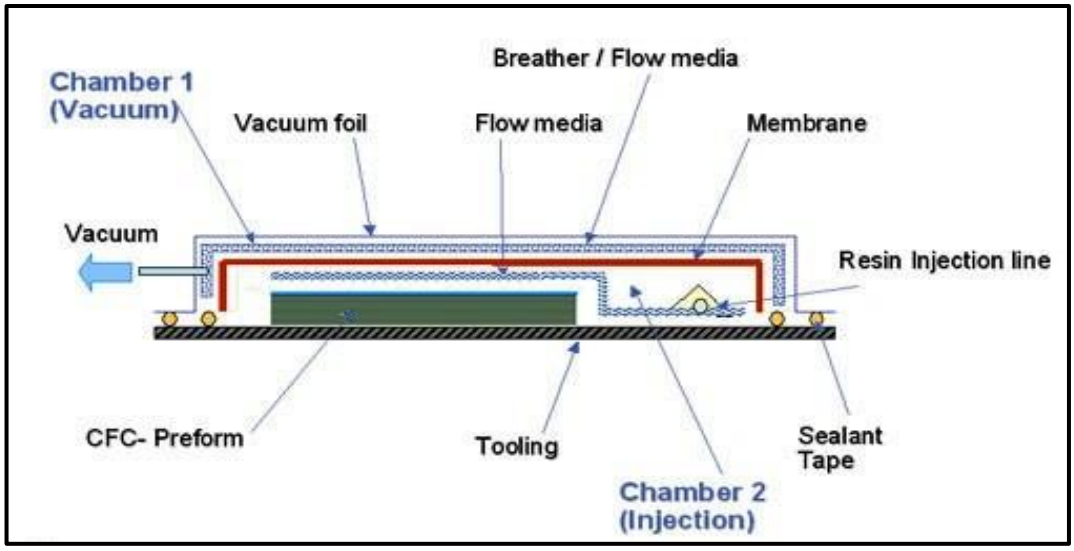


Figure 2.9: Schematic of VARIM technique [113]

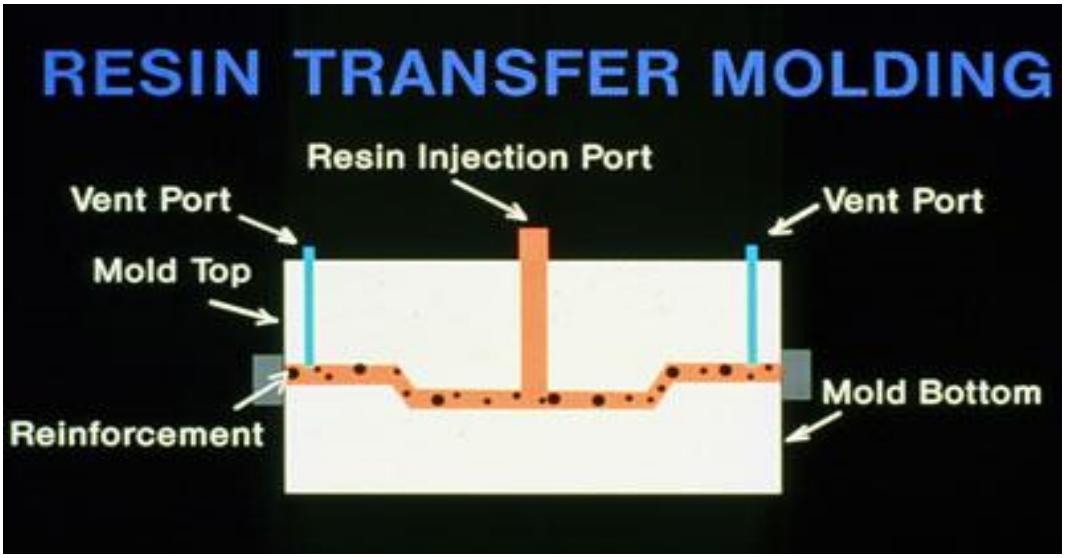


Figure 2.10: Schematic of RTM technique [112]

From the fabrication techniques discussed thus far, it is evident that a manual fibre layup process is inadequate for components designed using the variable stiffness method. These types of components require a fibre layup process that has excellent

repeatability, low material scrap rate and low labour costs. Further, it must be capable of high production rates and precise control over fibre orientation and placement.

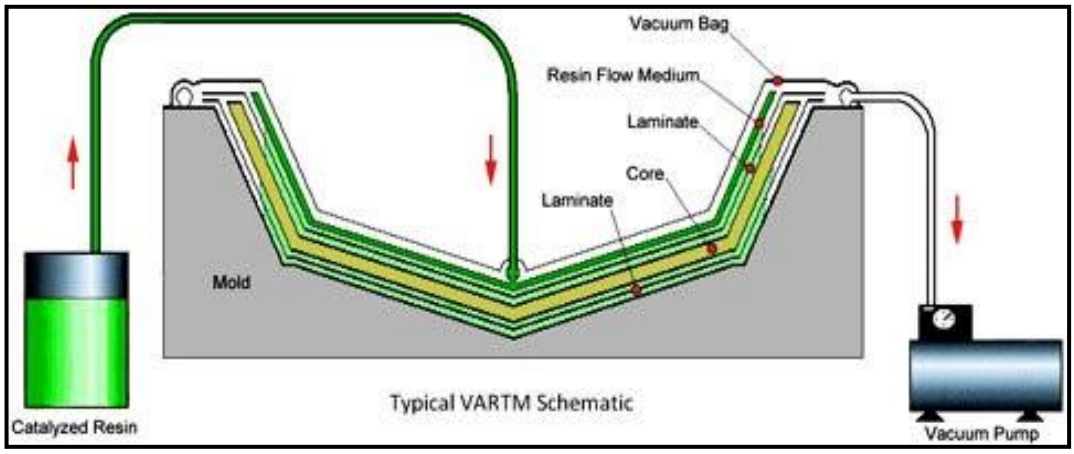


Figure 2.11: Schematic of VARTM technique [114]

The logical solution would be to turn to automated techniques. A study conducted by Bullock et al. [116] found that fibre orientation and placement in an automated fibre placement process was approximately seven times faster when compared to hand layup methods. Also, the fibres were more accurately positioned and were closer to the design specifications. Three popular fabrication methods with automated fibre placement systems were examined, beginning with filament winding.

Filament winding, shown in Figure 2.12, is an automated technique where wetted fibre is applied to a rotating mandrel. This allows for faster fibre placement and a fair control of fibre orientation. However, the precise control of true 0° fibres (fibres along the axis of the part) has proven difficult [119]. Further, it has a disadvantage in that only continuous fibres can be used, that is, fibres cannot be cut and restarted. Further, concave regions cannot be fabricated [111].

Automated Tape Placement (ATP), shown in Figure 2.13, is another automated technique employed in the fabrication of composite structures. It uses a specially designed placement head to lay resin infused fibre tapes. An advantage of this process is that fibres can be cut and restarted, thus enabling a higher control of fibre direction.

This technique is limited to fabrication of open section parts that are flat or have gentle contours. Any shape that is more complex than this will result in fibre buckling.

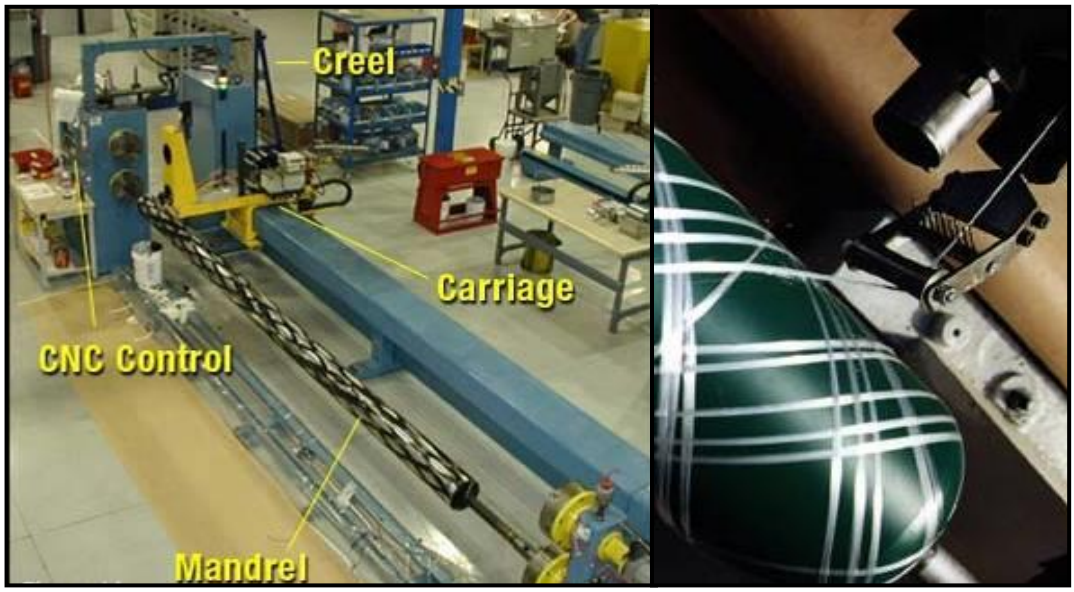


Figure 2.12: Filament winding technique [117-118]

Figure 2.13: Automated Tape Placement (ATP) [120]

The third fabrication method to be examined is Robotic Fibre Placement (RFP). It is simply a process where a robotic arm, with a purpose-designed manipulator or end-effector, is used to place fibres in a mould. The arm serves to position the end-effector at the required point of fibre placement, and the desired orientation and final placement of the fibres is performed by the end-effector. An example of a RFP system is shown in Figure 2.14.

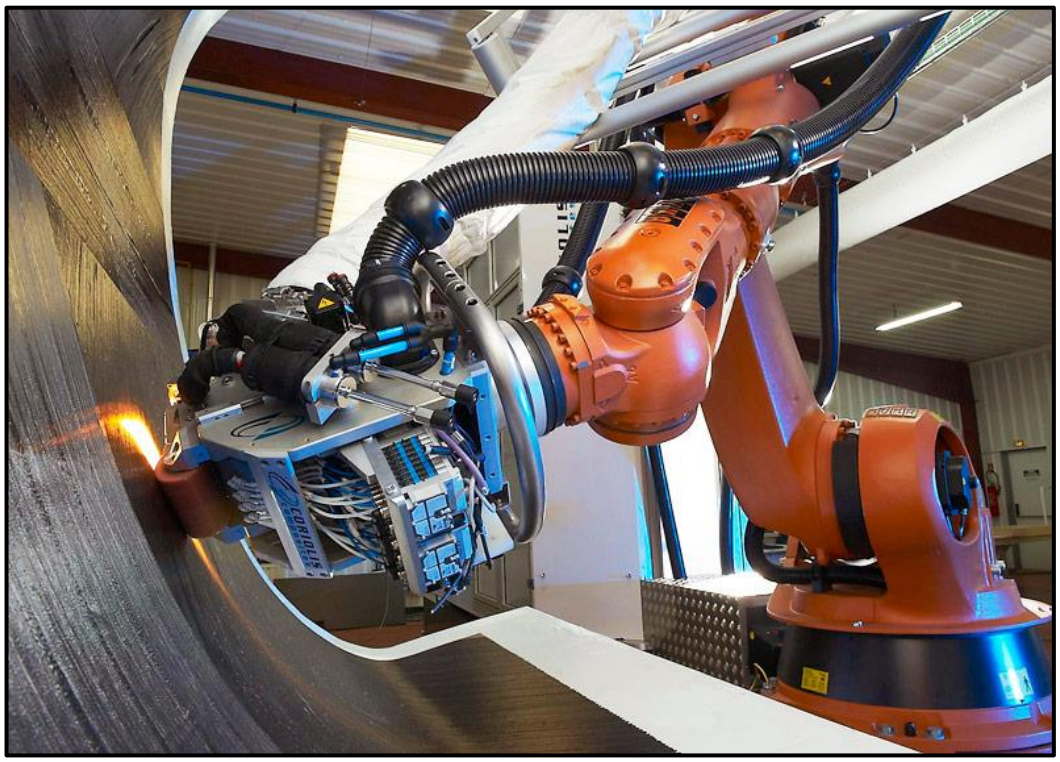


Figure 2.14: Robotic Fibre Placement (RFP) system [121]

RFP inherits all the advantages from both the filament winding and ATP processes. It is able to cut and restart fibre tows, and, debulk and consolidate the material in-situ [51-52]. This means that the fibres may continuously be cut and brought together into a single mass without the need for re-feeding the material into the RFP system and with no human intervention. Further, this fibre placement system can layup either continuous or chopped fibres, or a combination of these. RFP may be utilised for both thermosetting [51-52] and thermoplastic [65] manufacturing processes. Additionally, this system also has increased flexibility with little to no mould shape restraints. It can place fibres on open or closed surfaces as well as on mould shapes that consist of both

open and closed sections. However, the most crucial advantage of RFP is the ability to accurately control fibre orientation and placement.

A study on the fabrication of open surfaces using robotic fibre placement was conducted by Shirinzadeh et al [111]. Three different methods of fibre placement were tested and it was found that, in two of the cases, tow wrinkling occurred and gaps between fibres were formed. The third method provided them with a successfully manufactured composite panel. However, difficulties in orientating continuous fibres in multiple directions were experienced.

There has been much research into robotic fibre placement of thermoplastic or thermosetting pre-pregs (pre-impregnated fibres). A study involving robotic fibre placement of thermoplastic composites was conducted by Wakeman et al [122]. They used a robotic arm to locally reinforce a glass mat thermoplastic by an over-moulding process. The results indicated that the tensile properties of a composite panel may be increased by this process. Further, they suggested that manufacturing at high speeds using this process was possible with no significant losses in strength and rigidity.

Automated Dynamics Corporation (ADC), located in New York, USA, is a composites manufacturing company that implemented the RFP process. They have made great advancements to the technology [119,123-127]. ADC designed an end-effector for fibre reinforced thermoplastic tape-laying, and this is schematically shown in Figure 2.15. They have used this technology to fabricate helicopter drive shafts, tail booms and floor panels (Figure 2.16) for Bell Helicopter [119]. Work was also done on fibre placement of thermosetting pre-pregs. A new end-effector was designed for this and the schematic is shown in Figure 2.17. This end-effector was used to fabricate the V-22 Main Rotor Grip Element for Bell Helicopter, which is shown in Figure 2.18.

The NASA Langley Research Centre, located in Virginia, USA, conducted research [130] into fibre reinforced thermoplastic placement. Due to the high melt viscosities of high performance polymers, they developed an impregnation technique, shown in Figure 2.19, which spread finely ground polymer powder onto un-sized carbon fibre

tows. ADC manufactured the RFP machine with an end-effector that heated the tow as it was placed thus melting the thermoplastic powder in-situ. A schematic of this end-effector is shown in Figure 2.20. This method is very effective in the laying of fibre reinforced thermoplastic tows.

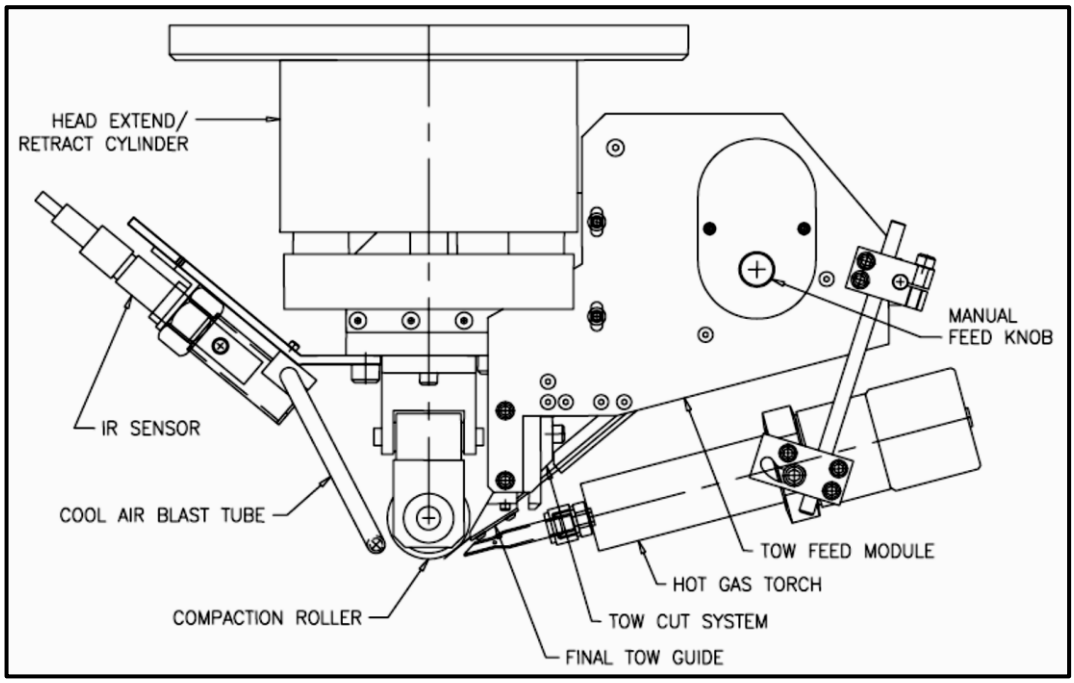
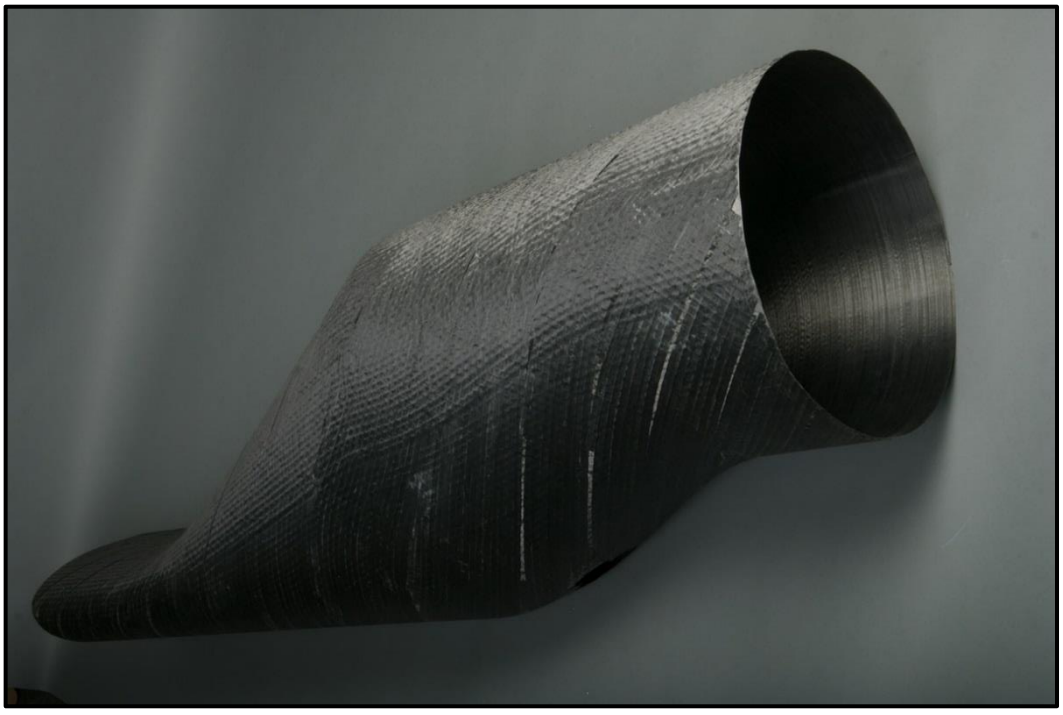


Figure 2.15: Thermoplastic fibre placement end-effector by ADC [128]

The RFP research conducted by National Composites Centre (NCC) in Ohio, USA, entailed the use of a chopper head, binding powder and vacuum assisted fibre placement [131]. A schematic of this fibre placement technique is shown in Figure 2.21. The fibres were placed on a moulded screen which was subjected to vacuum pressure on the underside. The vacuum held the fibres in place and the fibre length could be varied on the fly. Each layer of fibres was sprayed with a powder binder. Upon completion of the fibre layout, a second screen was used to compact the fibres while hot air was blown through the screens to cure the binder. The pre-form was then cooled and removed from the mould. This method resulted in the fabrication of dry pre-forms, that is, the fibres were not pre-impregnated with resin prior to the layup process. The pre-form may now be infused or injected with resin by some other fabrication technique. Figure 2.22 shows a dry pre-form of a helmet fabricated using

this technique. NCC claims that the chopper head can lay 90 % of the fibres within ±10° of the desired direction. This means that the fibre placements were not completely accurate, however, the end result was satisfactory and labour costs were decreased.

(a)



(b)

(c)

Figure 2.16: Fabricated components by ADC for Bell Helicopter: (a) drive shaft, (b) tail boom, and (c) floor panel [129]

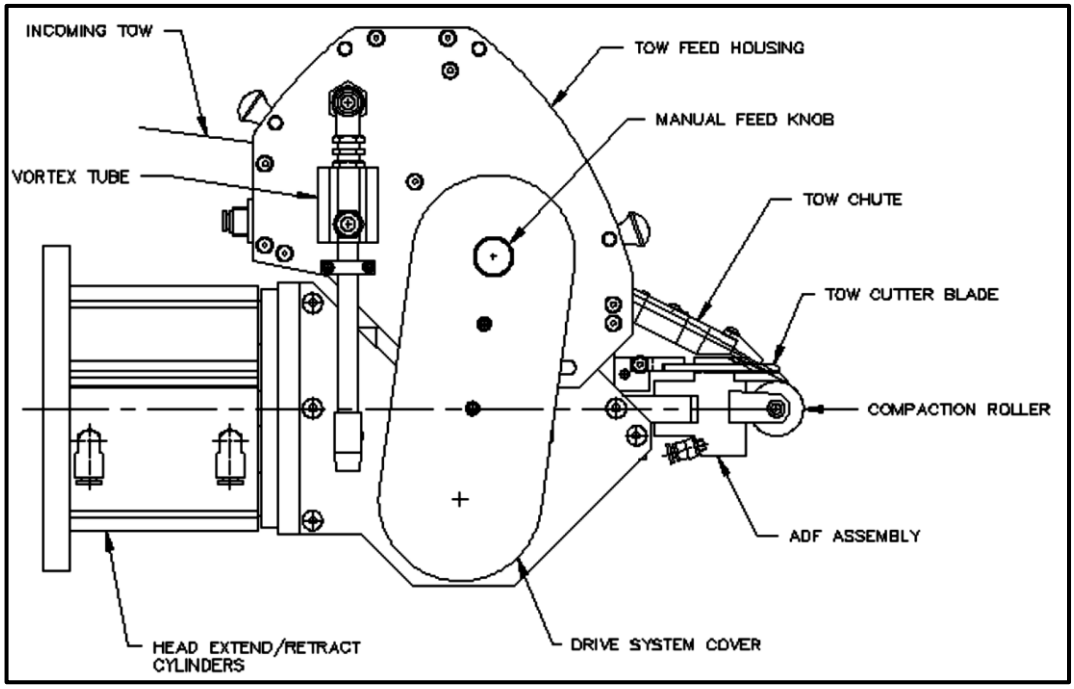


Figure 2.17: Thermosetting fibre placement end-effector by ADC [123]

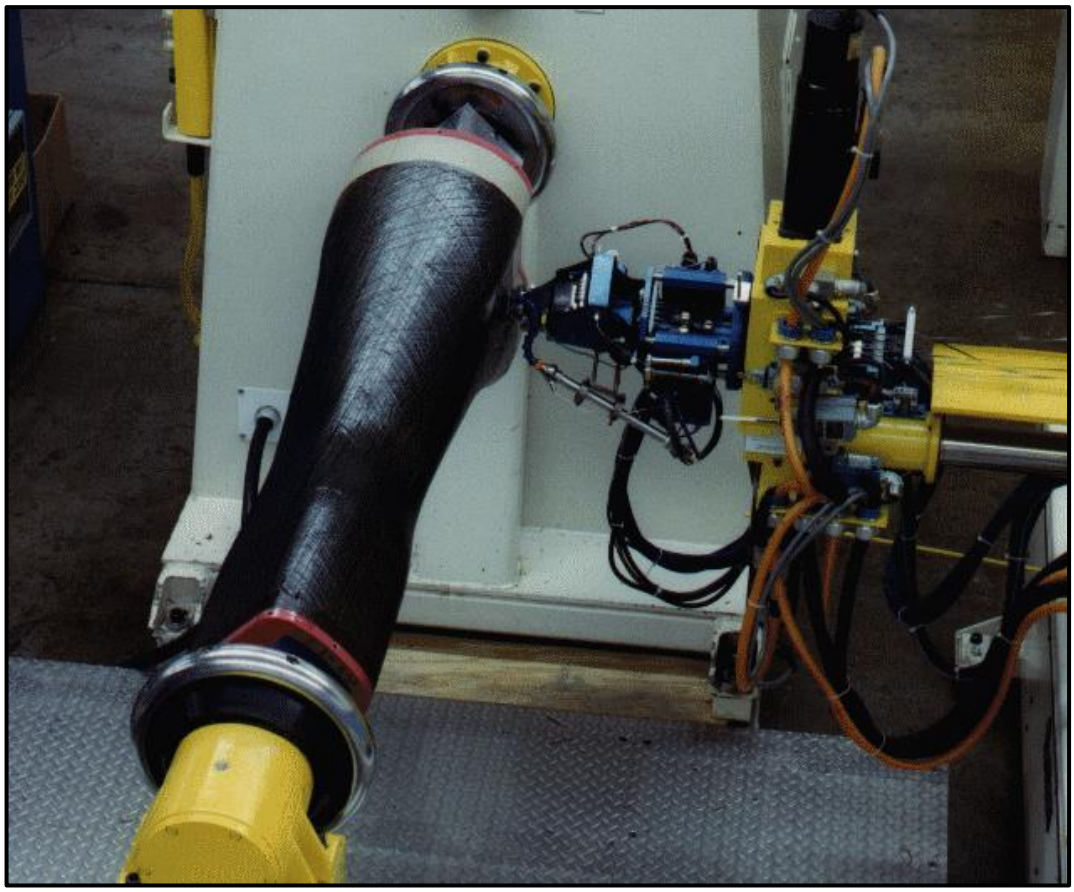


Figure 2.18: V-22 Main Rotor Grip fabricated by ADC for Bell Helicopter [129]

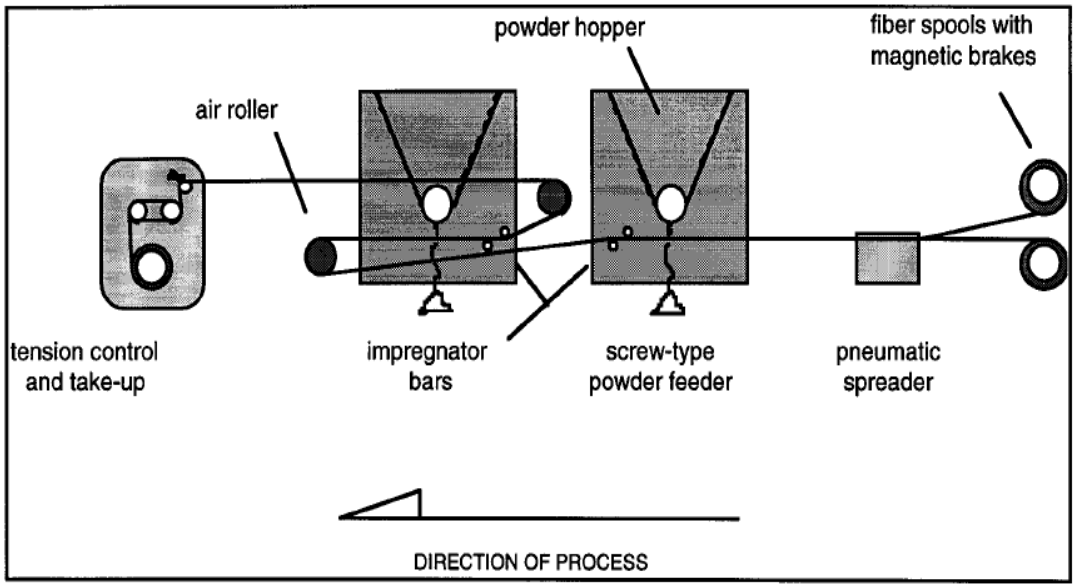


Figure 2.19: Schematic of NASA powder coating line [130]

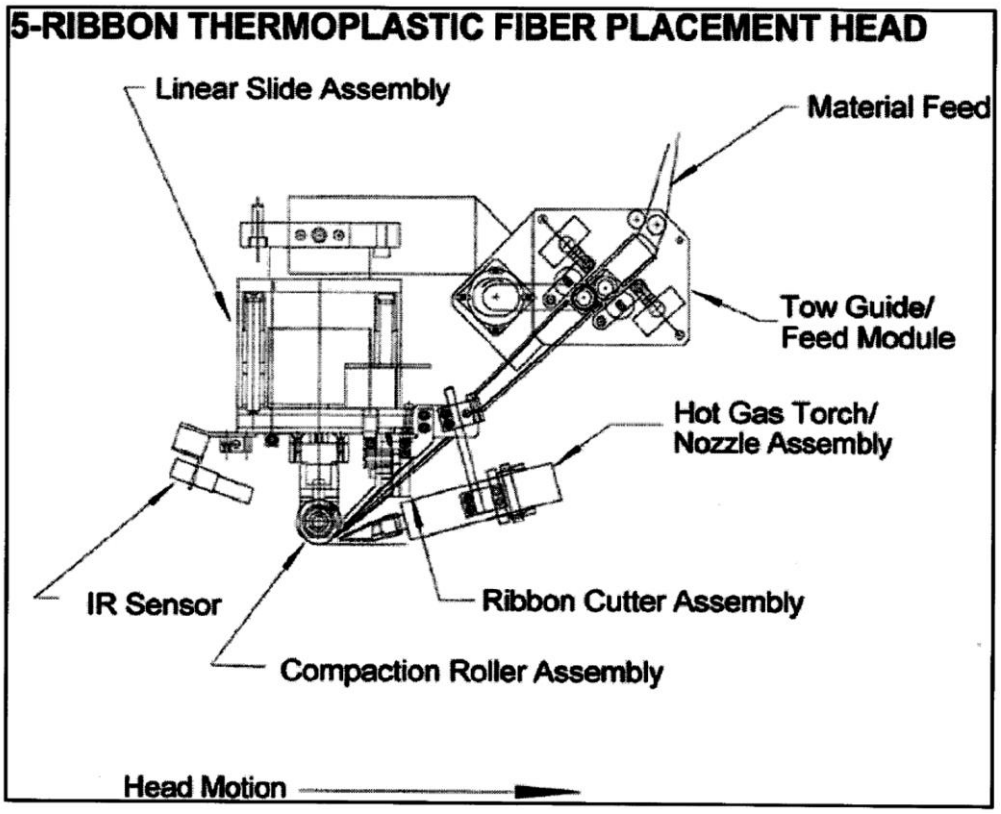


Figure 2.20: Schematic of the end-effector used by NASA [130]

From the above examination of fabrication techniques with automated fibre placement systems, it is evident that the RFP process is ideal for the variable stiffness design method. It is able to accurately orientate and place fibres, as well as reducing labour costs as fibre placement is automated. Studies have shown that it is capable of high production rates, and that it has excellent repeatability and a low material scrap rate.

In light of this, a RFP system, similar to that by NCC [131], was designed and built at the Durban University of Technology in another study. The end-effector laid dry fibres in a mould subjected to vacuum pressure. The vacuum kept the fibres in place during placement and a liquid binder was used to hold the fibres together once the vacuum was removed. The dry pre-preg was then able to be infused or injected with a thermosetting or thermoplastic matrix via any fabrication method such as VARIM, RTM or VARTM. In the current study an interface between the variable stiffness design method and the previously built RFP system was developed to allow for the fabrication of dry pre-pregs.

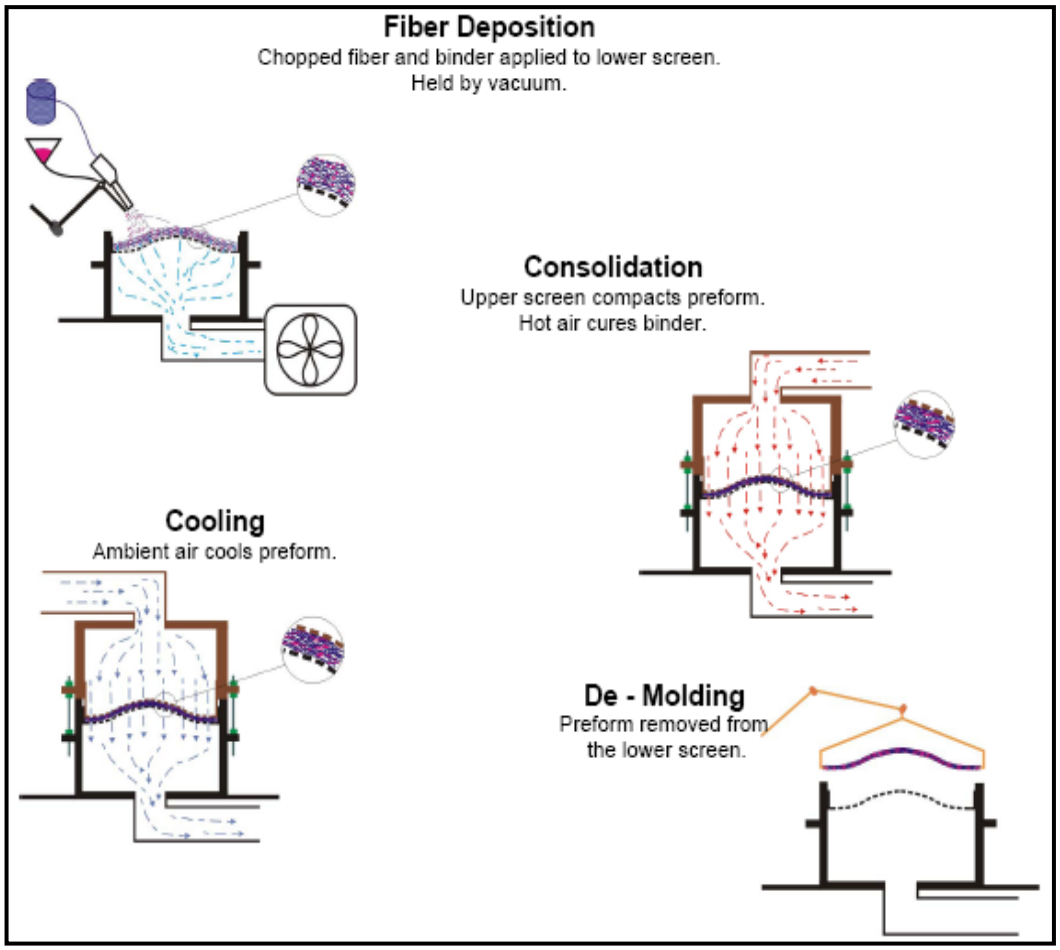


Figure 2.21: Fibre placement technique implemented by NCC [131]

Figure 2.22: Helmet pre-forms fabricated by NCC [131]

2.9 Summary

The previous chapter mentioned that the design of fibre reinforced composites required the use of Hooke's Law. Two sets of inputs were needed, namely, the material-loading properties (material constants, material limits and loading conditions) and the fibre layup properties. The former remains fixed during the design process while the latter may be varied and has a direct influence on the strength, stiffness and weight of the composite. Therefore the optimisation of the fibre layup parameters leads to the optimisation of the design process.

The presented literature disclosed that by optimising the fibre layup parameters, the mechanical properties were improved as well as achieving a reduction in weight. This naturally leads to a decrease in material costs. It was further shown that variable stiffness composites are superior to constant stiffness ones. There has been extensive research conducted with regards to the buckling and vibration analysis of variable stiffness composites. However, it was suggested that more improvement in properties could be obtained if composites were optimised for strength rather than buckling [90].

This set the precedent for the primary objective of this research work which was to develop a design technique that is able to optimise the fibre layup parameters of variable stiffness composites for maximum strength and minimum weight. The technique does not use finite sets of values, and is relevant for two-dimensional and three-dimensional structures. Further, the technique was based on the method proposed by Farshi et al. [79] where the fibre orientation angle and layer thickness were optimised in turn.

With the development of the variable stiffness design method, a suitable fibre layup process needed to be found. Processes with manual fibre placement proved to be inadequate as precise control over fibre orientation and placement was extremely difficult or non-existent. The answer to this problem was automated fibre placement systems, in particular, Robotic Fibre Placement (RFP). This technique has many advantages over other automated systems and has excellent repeatability, low material

scrap rate and low labour costs. It is also capable of high production rates and precise control over fibre orientation and placement.

In another study at the Durban University of Technology, a RFP system with a fibre placement end-effector was designed and built. The secondary objective of the current research work was to develop an interface that converted the resulting fibre layup parameters, from the variable stiffness design method, to robotic code in order to allow for the fabrication of dry fibre pre-pregs.

The following chapter discusses the hypotheses of the current study, and the research objectives and goals required to fulfil these hypotheses. The resources that were used to achieve these objectives, as well as the research outline, are also discussed.

3. RESEARCH METHODOLOGY AND RESEARCH DESIGN

3.1 Overview

The hypotheses and objectives of the study, as well as the tools required to fulfil the objectives, are discussed in this chapter. These tools include Hooke's Law equations for composite materials, the Composite Micromechanics equations, the programming environment, and the methods used to validate the developed codes. The various codes that were developed are also briefly discussed.

3.2 Hypotheses

- If a computational code can be developed that is capable of optimising the design process of a fibre reinforced composite, then it is conceivable to use this developed code to create multi-strength, multi-directional (variable stiffness) structures.
- If the above design optimisation code is coupled with an appropriate fibre layup technique, then the high speed, high precision processing of dry pre-pregs is possible.

3.3 Research objectives

The objectives required to fulfil the hypotheses set out above are listed below, and includes the goals necessary to achieve each objective.

A. Develop a computational code that optimises the design process, that is, makes the process quick, simple and accurate. Further, this code must enable the creation of multi-strength, multi-directional structures.
   - A.1. Produce the necessary computational functions to perform calculations easily and swiftly.
   - A.2. Develop a code that optimises the fibre orientation angle only. The layer thickness will be kept constant.
   - A.3. Expand on the code from A.2 above to include optimisation of the layer thickness.
   - A.4. Use the code developed in A.3 above to design multi-strength, multi-directional (variable stiffness) structures. The structure would be divided into

two-dimensional elements via a finite element package. Each element can then be treated as a separate laminate and the fibre layup can be determined for each element.

A.5. Validate the developed codes by comparing results obtained in the computational package with examples from texts, finite element analyses, and experimentation.

B. Create a software interface between the code developed in 1 above, and the fibre placement end-effector designed and built in the previous study.

B.1. Examine the design and operation of each component of the end-effector.

B.2. Develop a software interface to enable the transition from the design phase to the manufacturing phase.

C. Develop other codes / interfaces that enable the user to interact with the computational environment more easily.

C.1. Create Graphical User Interfaces (GUIs) in order to allow visual interaction between the user and the developed codes.

C.2. Create a database from which the user can choose the composite material, along with its properties, to be used in the design process.

C.3. Develop a code that allows the user to view/add/edit/delete materials from the above database.

The thought processes and mechanisms that were involved in achieving the above objectives are detailed in the sections that follow. However, prior to fulfilling these objectives, there were two preliminary investigations that needed to be completed, namely, the design process and the programming environment.

3.4 Preliminary investigations

Before the optimisation of the fibre layup parameters could commence, it was necessary to investigate the aspects involved in a design process. The design of any structure requires the understanding of the relationship between the stress and strain characteristics of the structure. This relationship dictates the behaviour of the structure

when subjected to external forces. Stress is the expression of these forces per unit cross-sectional area. When a stress is applied to a structure, it causes a change in shape or deformation. The comparison of this deformation to the original shape is referred to as strain.

The stress-strain relationship of any structure is best described using equations. Therefore, in order to understand the relationship between the stress and strain characteristics in laminated fibre reinforced composites, it was crucial to examine the equations that are normally used in the design process of such materials. It was theorised that the optimisation of these equations, or the way these equations were used, would optimise the design process itself. Hence the equations were essential in the development of the computational functions that were required to achieve the research objectives. These equations are presented and discussed in the next two sections.

The second preliminary investigation was to determine an appropriate computational coding environment/software. The software had to ensure easy coding as well as being able to handle the numerous matrix computations associated with the equations used in the design process. The coding environment that was decided upon is discussed in a following section.

3.4.1 The stress-strain equations for fibre reinforced composite laminates

The stresses and strains at any point in a body that is linearly elastic and undergoes small deformations are related by Hooke's Law [132]. The stress-strain behaviour in an isotropic material is shown in equation (3.1) where σ is the normal stress, ε is the strain and E is Young's Modulus.

$$\sigma = E\,\varepsilon \tag{3.1}$$

A fibre reinforced composite material is not isotropic and hence has different properties in the different principal directions. As a result the stress-strain relationship

is not as straightforward as in equation (3.1). The stress-strain equations for fibre reinforced composite materials may be derived from the general stress-strain relationships for three-dimensional bodies [4,132-134].

However, before this is done, a co-ordinate system needs to be established. Consider the fibre reinforced composite plate shown in Figure 3.1. The rod-like structures represent the fibres and the rectangular block represents the matrix. The direction along the fibre length is designated 1. Designation 2 is given to the direction transverse to the fibre length but in the plane of the plate. The direction transverse to both the fibre length and the plane of the plate is designated 3.

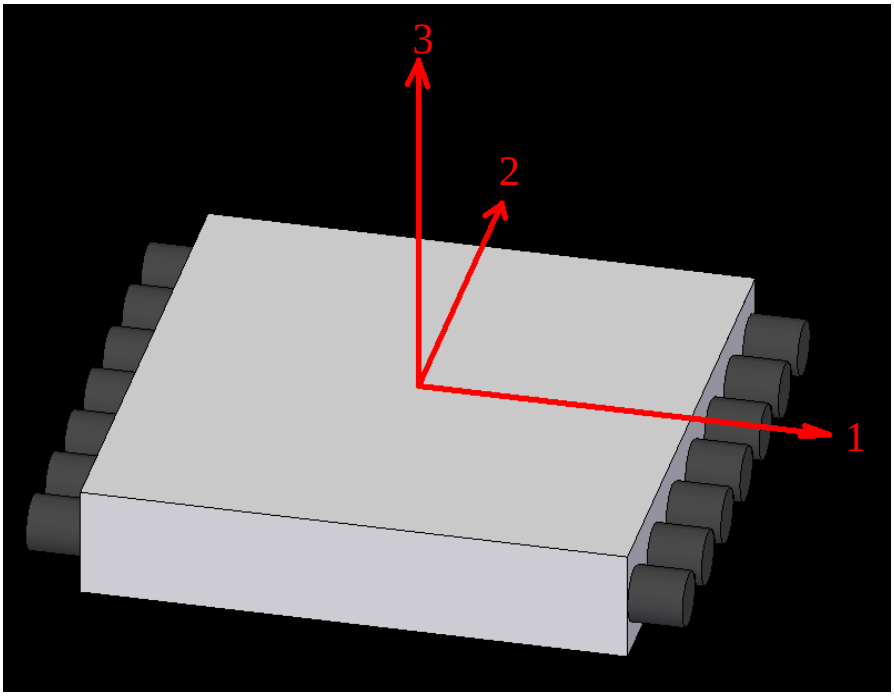


Figure 3.1: Fibre reinforced composite plate showing material co-ordinate system

Based on the co-ordinate system in Figure 3.1, the stress-strain relationship for a non-linearly elastic three-dimensional material may be defined [132-133]. Equation (3.2) describes this relationship and was derived from Hooke's Law [132].

The [*C*] matrix in equation (3.2) has 36 independent constants. However, if the material is orthotropic, that is, it has three mutually perpendicular planes of symmetry,

the [*C*] matrix may be reduced [132-133]. Hence equation (3.2) now becomes equation (3.3) where the [*C*] matrix has only nine independent constants [132]. Under certain conditions equation (3.3) is further reduced. These are investigated next.

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \tau_{23} \\ \tau_{31} \\ \tau_{12} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \gamma_{23} \\ \gamma_{31} \\ \gamma_{12} \end{bmatrix} \tag{3.2}$$

where: $\sigma_{1,2,3}$ are the normal stresses in directions 1, 2, 3, respectively

$\tau_{23,31,12}$ are the shear stresses in the 2-3, 3-1, 1-2 planes, respectively

$\varepsilon_{1,2,3}$ are the normal strains in directions 1, 2, 3, respectively

$\gamma_{23,31,12}$ are the shear strains in the 2-3, 3-1, 1-2 planes, respectively

[*C*] is a matrix of constants and is called the stiffness matrix

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \tau_{23} \\ \tau_{31} \\ \tau_{12} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ C_{12} & C_{22} & C_{23} & 0 & 0 & 0 \\ C_{13} & C_{23} & C_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & C_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \gamma_{23} \\ \gamma_{31} \\ \gamma_{12} \end{bmatrix} \tag{3.3}$$

3.4.1.1 Stress-strain relationships for a single fibre layer

If plane stress conditions are assumed, as is in the case of continuous unidirectional fibre composites [111-112], then equation (3.3) may be reduced to two-dimensions. The result is equations (3.4) and (3.5) [132], where [*Q*] is the reduced stiffness matrix, [*S*] is the compliance matrix, and $[S] = [Q]^{-1}$.

The elements of the [*Q*] and [*S*] matrices in equations (3.4) and (3.5) are dependent on the material constants. These include the stiffness moduli, shear moduli and Poisson's

ratios. Equations (3.6) and (3.7) are used to determine the elements of the [*Q*] and [*S*] matrices, respectively [132].

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \tau_{12} \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} & 0 \\ Q_{12} & Q_{22} & 0 \\ 0 & 0 & Q_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \gamma_{12} \end{bmatrix} \tag{3.4}$$

$$\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \gamma_{12} \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} & 0 \\ S_{12} & S_{22} & 0 \\ 0 & 0 & S_{66} \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \tau_{12} \end{bmatrix} \tag{3.5}$$

$$Q_{11} = \frac{E_1}{1-\nu_{12}\nu_{21}} \quad , \quad Q_{12} = \frac{\nu_{12}E_2}{1-\nu_{12}\nu_{21}} \quad , \quad Q_{22} = \frac{E_2}{1-\nu_{12}\nu_{21}} \quad , \quad Q_{66} = G_{12} \tag{3.6}$$

$$S_{11} = \frac{1}{E_1} \quad , \quad S_{12} = -\frac{\nu_{12}}{E_1} \quad , \quad S_{22} = \frac{1}{E_2} \quad , \quad S_{66} = \frac{1}{G_{12}} \tag{3.7}$$

where: $E_{1,2}$ are Young's moduli in directions 1 and 2, respectively

G_{12} is the shear modulus in the 1-2 plane

$\nu_{12,21}$ are Poisson's ratios in the 1-2 and 2-1 planes, respectively

The use of equations (3.4) to (3.7) would result in the stresses and strains being found on a particular layer. However these results are not comparable to other layers as there is no point of reference. The co-ordinate system used (Figure 3.1) is relative to the fibre axis on a particular layer, but not the composite laminate as a whole. It does not take into account the fibre orientation angle on a particular layer in relation to its surroundings and to the other layers.

Hence, a new system of co-ordinates would be needed that relates the co-ordinate system on each layer to the laminate as a whole. In Figure 3.2 the 1-2 system (referred to as the local co-ordinates) and the x-y system (referred to as the global co-ordinates)

are related to each other by the angle θ. This angle is the angle between the fibre's longitudinal axis (direction 1) and the laminate's principle axis (direction x).

As a result of the new co-ordinate system, the stresses and strains in equations (3.4) and (3.5) would now have to be related to the x-y co-ordinate system. This relationship is shown in equation (3.8) [132].

$$\begin{bmatrix}\sigma_1\\ \sigma_2\\ \tau_{12}\end{bmatrix} = [T]\begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix} \qquad \textit{and} \qquad \begin{bmatrix}\varepsilon_1\\ \varepsilon_2\\ \gamma_{12}\end{bmatrix} = [R][T][R]^{-1}\begin{bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{bmatrix} \tag{3.8}$$

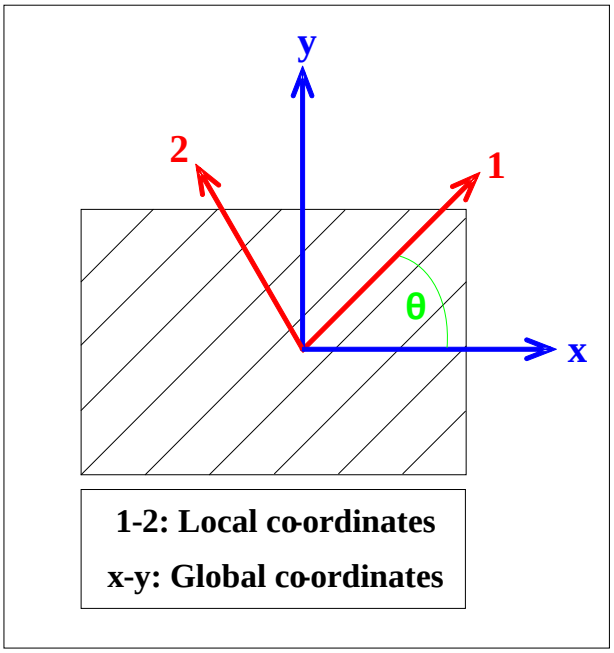


Figure 3.2: Global co-ordinate system in relation to local co-ordinate system

In equation (3.8), [T] is the transformation matrix and [R] is the Reuter matrix. These are defined in equation (3.9) [132], where $c = \cos\theta$ and $s = \sin\theta$.

$$T = \begin{bmatrix} c^2 & s^2 & 2sc \\ s^2 & c^2 & -2sc \\ -sc & sc & c^2 - s^2 \end{bmatrix} \qquad \textit{and} \qquad R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{3.9}$$

Substituting equation (3.8) into equations (3.4) and (3.5) results in equations (3.10) and (3.11) [132], where $[\overline{Q}]$ is the transformed reduced stiffness matrix, $[\overline{S}]$ is the transformed reduced compliance matrix, and $[\overline{S}] = [\overline{Q}]^{-1}$. The $[\overline{Q}]$ matrix is determined by equation (3.12) [132].

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} \overline{Q}_{11} & \overline{Q}_{12} & \overline{Q}_{16} \\ \overline{Q}_{12} & \overline{Q}_{22} & \overline{Q}_{26} \\ \overline{Q}_{16} & \overline{Q}_{26} & \overline{Q}_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} \tag{3.10}$$

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \overline{S}_{11} & \overline{S}_{12} & \overline{S}_{16} \\ \overline{S}_{12} & \overline{S}_{22} & \overline{S}_{26} \\ \overline{S}_{16} & \overline{S}_{26} & \overline{S}_{66} \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} \tag{3.11}$$

$$[\overline{Q}] = [T]^{-1}[Q][R][T][R]^{-1} \tag{3.12}$$

It is important to note that in the above equations, it is not necessary to determine the [*S*] and $[\overline{S}]$ matrices as they are the inverse of the [*Q*] and $[\overline{Q}]$ matrices, respectively. This would reduce the number of calculations performed.

From the above, it may be established that there are three steps required to determine the stresses and strains for a single layer composite in the global co-ordinate system. First, the [*Q*] matrix must be determined using equation (3.6) and the material constants mentioned earlier (E_1, E_2, G_{12}, ν_{12} and ν_{21}). Next, the $[\overline{Q}]$ matrix is calculated by using the fibre orientation angle (θ), equations (3.9) and (3.12), and the [*Q*] matrix computed in the previous step. Lastly, the global stresses and strains are found using equation (3.10) or equation (3.11).

The discussion thus far focused on the stress-strain equations for a single layer composite. However, fibre reinforced composites usually consist of several layers, and therefore a new set of relationships were developed.

3.4.1.2 Stress-strain relationships for multiple fibre layers

Equations (3.13) and (3.14) show the stress-strain relationships for multiple fibre layers [132].

$$\begin{bmatrix} N_x \\ N_y \\ N_{xy} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{16} \\ A_{12} & A_{22} & A_{26} \\ A_{16} & A_{26} & A_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{bmatrix} + \begin{bmatrix} B_{11} & B_{12} & B_{16} \\ B_{12} & B_{22} & B_{26} \\ B_{16} & B_{26} & B_{66} \end{bmatrix} \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix} \tag{3.13}$$

$$\begin{bmatrix} M_x \\ M_y \\ M_{xy} \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} & B_{16} \\ B_{12} & B_{22} & B_{26} \\ B_{16} & B_{26} & B_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{bmatrix} + \begin{bmatrix} D_{11} & D_{12} & D_{16} \\ D_{12} & D_{22} & D_{26} \\ D_{16} & D_{26} & D_{66} \end{bmatrix} \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix} \tag{3.14}$$

where: N_x, N_y are the normal forces, and N_{xy} is the shear force

M_x, M_y are the bending moments, and M_{xy} is the twisting moment

ε^0 are the mid-plane strains

κ is the mid-plane curvatures

The terms ε^0 and κ are related to the global co-ordinates (x-y system) as shown in equation (3.15), where z is the distance from the mid-plane of the laminate [132].

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{bmatrix} + z \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix} \tag{3.15}$$

The [A], [B], and [D] matrices in equations (3.13) and (3.14) are known as the extensional, coupling, and bending stiffness matrices, respectively. The elements of these matrices may be determined by the following three equations [132]:

$$A_{ij} = \sum_{k=1}^{n} \left[\overline{(Q_{ij})}\right]_k \left(h_k - h_{k-1}\right) \qquad i = 1,2,3 \quad j = 1,2,3 \tag{3.16}$$

$$B_{ij} = \frac{1}{2}\sum_{k=1}^{n}\left[\left(\overline{Q_{ij}}\right)\right]_k \left(h_k^2 - h_{k-1}^2\right) \qquad i=1,2,3 \quad j=1,2,3 \tag{3.17}$$

$$D_{ij} = \frac{1}{3}\sum_{k=1}^{n}\left[\left(\overline{Q_{ij}}\right)\right]_k \left(h_k^3 - h_{k-1}^3\right) \qquad i=1,2,3 \quad j=1,2,3 \tag{3.18}$$

where: n is the number of layers

$\left[\left(\overline{Q_{ij}}\right)\right]_k$ is the i-th, j-th element of the $[\overline{Q}]$ matrix of the k-th layer

h_k is the distance of the bottom of the k-th layer from the mid-plane of the laminate

Figure 3.3 illustrates the positions of h_k from the mid-plane, and equation (3.19) may be used to determine the values of h_k [132]. Equations (3.13) and (3.14) may be written in a more condensed as equation (3.20) [132].

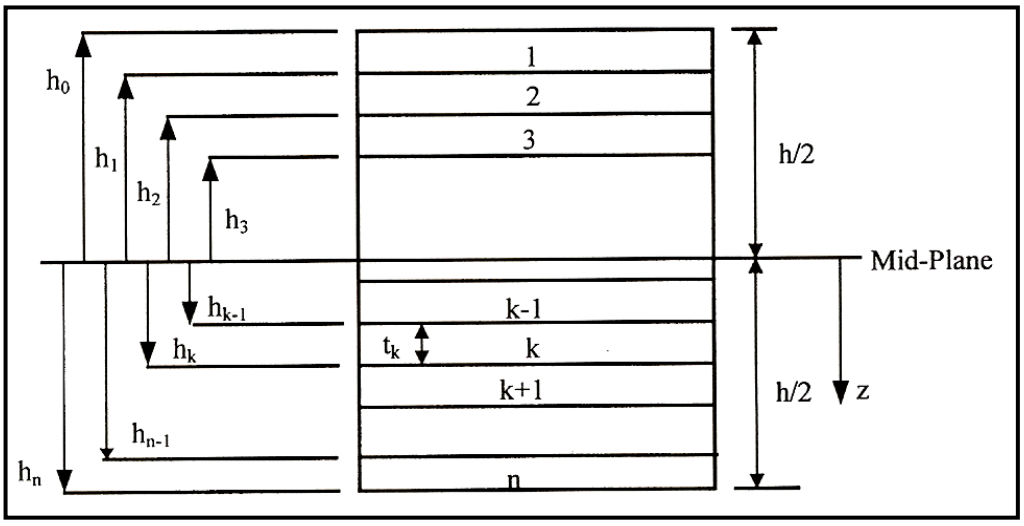


Figure 3.3: Locations of layers in a composite structure [132]

$$h = \sum_{k=1}^{n} t_k \quad ; \qquad h_0 = -\frac{h}{2} \quad ; \qquad h_k = h_{k-1} + t_k \tag{3.19}$$

The tools required to determine the local and global stresses and strains have been discussed. Equations (3.4) to (3.12) may be used to resolve the stresses and strains on

each layer, while equations (3.15) to (3.20) relate the stresses and strains throughout the laminate. In the formulation of these equations certain assumptions were made [132], namely,

1. Each layer is orthotropic, homogeneous and elastic.
2. There is no slip between the layer interfaces.
3. A line that is straight and perpendicular to the mid-plane remains straight and perpendicular to the mid-plane during deformation.
4. The length of a straight line in the z-direction remains constant.
5. The laminate follows plane stress conditions, that is, it is thin and is loaded only in its plane.
6. Displacements are small and continuous throughout the laminate.

$$\begin{bmatrix} N_x \\ N_y \\ N_{xy} \\ M_x \\ M_y \\ M_{xy} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{16} & B_{11} & B_{12} & B_{16} \\ A_{12} & A_{22} & A_{26} & B_{12} & B_{22} & B_{26} \\ A_{16} & A_{26} & A_{66} & B_{16} & B_{26} & B_{66} \\ B_{11} & B_{12} & B_{16} & D_{11} & D_{12} & D_{16} \\ B_{12} & B_{22} & B_{26} & D_{12} & D_{22} & D_{26} \\ B_{16} & B_{26} & B_{66} & D_{16} & D_{26} & D_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \\ \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix} \tag{3.20}$$

It is significant to point out that any computational code that is developed based on the above equations would have the same assumptions as well. Further, according to Kaw [132], the above equations are valid not only for unidirectional composites, but also for composites with fibres that are woven perpendicular to each other, or contain short fibres that are aligned perpendicular to each other or in one direction.

Once the laminate has been designed, it is important to determine whether any failure occurs. The following section discusses the failure theory used in this study.

3.4.1.3 The Tsai-Wu failure theory

There are various failure theories for composite laminates but the Tsai-Wu failure criterion was chosen for the purpose of this research. This failure theory was selected

as it closely correlates with experimental data [132], and it has been used by other researchers [73-77,89-90,100,110] in their studies. The Tsai-Wu failure criterion is given by equation (3.21) [132].

$$H_1\sigma_1 + H_2\sigma_2 + H_6\tau_{12} + H_{11}\sigma_1^2 + H_{22}\sigma_2^2 + H_{66}\tau_{12}^2 + 2H_{12}\sigma_1\sigma_2 < \quad 1 \qquad (3.21)$$

where: $\sigma_{1,2}$ are the local normal stresses in directions 1 and 2 on a particular layer

τ_{12} is the local shear stress in the 1-2 plane on a particular layer

$H_{1,2,6,11,22,66,12}$ are the Tsai-Wu parameters

The parameters for the Tsai-Wu failure criterion are given by equation (3.22) [132].

$$H_1 = \frac{1}{\left(\sigma_1^T\right)_{ult}} - \frac{1}{\left(\sigma_1^C\right)_{ult}}; \qquad H_2 = \frac{1}{\left(\sigma_2^T\right)_{ult}} - \frac{1}{\left(\sigma_2^C\right)_{ult}}; \qquad H_6 = 0$$

$$H_{11} = \frac{1}{\left(\sigma_1^T\right)_{ult}\left(\sigma_1^C\right)_{ult}}; \qquad H_{22} = \frac{1}{\left(\sigma_2^T\right)_{ult}\left(\sigma_2^C\right)_{ult}}; \qquad H_{66} = \frac{1}{\left(\tau_{12}\right)_{ult}^2} \qquad (3.22)$$

$$H_{12} = -\frac{1}{2}\sqrt{\frac{1}{\left(\sigma_1^T\right)_{ult}\left(\sigma_1^C\right)_{ult}\left(\sigma_2^T\right)_{ult}\left(\sigma_2^C\right)_{ult}}}$$

where: $(\sigma_{1,2}{}^T)_{ult}$ are the ultimate tensile stresses in directions 1 and 2

$(\sigma_{1,2}{}^C)_{ult}$ are the ultimate compressive stresses in directions 1 and 2

$(\tau_{12})_{ult}$ is the ultimate shear stress in the 1-2 plane

In order to better facilitate the use of this failure theory, each stress component of equation (3.21) was multiplied by a variable [132]. This variable is referred to as the strength ratio (*SR*). Its purpose is to directly determine by what ratio the local stresses must be increased or decreased to avoid failure. *SR* also directly relates to the applied loading. Combining *SR* in equation (3.21) results in equation (3.23) [132].

$$(H_1\sigma_1 + H_2\sigma_2 + H_6\tau_{12})SR + (H_{11}\sigma_1^2 + H_{22}\sigma_2^2 + H_{66}\tau_{12}^2 + 2H_{12}\sigma_1\sigma_2)SR^2 < \quad 1 \qquad (3.23)$$

The criterion for *SR* is that it must be greater than zero. If *SR* is less than 1, then failure occurs because it means that the loading needs to decrease to avoid failure. A *SR* value of 1 implies that the composite laminate is perfectly suited for the applied loading conditions. If the value of *SR* is greater than 1, it implies that the laminate is capable of carrying more than the applied loading. For example, a *SR* value of 1.5 means that the loading may be increased by 50% without inducing failure.

3.4.2 Composite Micromechanics equations

In the previous section, the equations governing the stress-strain behaviour of fibre reinforced composite laminates were discussed. These equations ultimately depend on the material constants (E_1, E_2, G_{12}, ν_{12} and ν_{21}) and the material limits (($\sigma_1^T)_{ult}$, ($\sigma_2^T)_{ult}$, ($\sigma_1^C)_{ult}$, ($\sigma_2^C)_{ult}$ and $(\tau_{12})_{ult}$). These parameters may be determined by experimentation, however this process is quite costly and time consuming [132]. Hence equations were developed to determine the material constants and limits analytically.

However, before examining the equations for the material properties mentioned above, it is essential to understand what factors contribute to the strength of a fibre reinforced composite. The fibres are the load bearing elements in a fibre reinforced composite. This suggests that the more fibres there are per unit area, the higher the load bearing capability. However, other properties may decrease. A balance needs to be established depending on the application. Therefore the material constants and limits will vary depending on the fibre-matrix ratio. This ratio is expressed as a fraction and is the amount of fibre or matrix present in the composite. The fibre-matrix ratio in terms of volume is determined by equation (3.24) [132].

$$V_f = \frac{v_f}{v_c} \quad , \quad V_m = \frac{v_m}{v_c} \quad , \quad V_f + V_m = 1 \tag{3.24}$$

where: $v_{c,\,f,\,m}$ is the volume of the composite, fibre and matrix, respectively
$V_{f,\,m}$ is the volume fraction of the fibre and matrix, respectively

The fibre-matrix ratio in terms of weight is determined by equation (3.25) [132].

$$W_f = \frac{w_f}{w_c} \quad , \quad W_m = \frac{w_m}{w_c} \quad , \quad W_f + W_m = 1 \tag{3.25}$$

where: $w_{c,\,f,\,m}$ is the weight of the composite, fibre and matrix, respectively
$W_{f,\,m}$ is the weight fraction of the fibre and matrix, respectively

The density of a fibre reinforced laminate in terms of volume fractions and weight fractions are given by equations (3.26) and (3.27), respectively [132].

$$\rho_c = \rho_f V_f + \rho_m V_m \tag{3.26}$$

$$\frac{1}{\rho_c} = \frac{W_f}{\rho_f} + \frac{W_m}{\rho_m} \tag{3.27}$$

where: $\rho_{c,\,f,\,m}$ is the density of the composite, fibre and matrix, respectively

Now that the fibre-matrix ratios (volume and weight fractions) have been established, the equations needed for the evaluation of the material constants (E_1, E_2, G_{12}, ν_{12} and ν_{21}) may be examined.

3.4.2.1 Evaluation of the material constants

There are several models available, each with their own fundamental equations, which may be used to determine the material constants. These include the Strength of Materials model and Elasticity model. Unlike the other models, the Elasticity model accounts for Hooke's law in three dimensions [132]. Therefore this model and its fundamental equations were chosen for this research work. However, the Elasticity model cannot be used to evaluate the transverse Young's modulus (E_2), and therefore the equation for this material constant was obtained from the Strength of Materials model. The formulae for the longitudinal Young's modulus (E_1), the transverse Young's modulus (E_2), the shear modulus in the 1-2 plane (G_{12}), Poisson's ratio in the

1-2 plane (ν_{12}), and Poisson's ratio in the 2-1 plane (ν_{21}) are given by equations (3.28), (3.29), (3.30), (3.31) and (3.32), respectively [132].

$$E_1 = E_f V_f + E_m V_m + \frac{4\left(\nu_m - \nu_f\right)^2 V_f V_m}{\frac{V_f}{K_m} + \frac{V_m}{K_f} + \frac{1}{G_f}} \tag{3.28}$$

$$\frac{1}{E_2} = \frac{V_f}{E_f} + \frac{V_m}{E_m} \tag{3.29}$$

$$G_{12} = G_f \frac{G_f V_f + G_m\left(1 + V_m\right)}{G_m V_f + G_f\left(1 + V_m\right)} \tag{3.30}$$

$$\nu_{12} = \nu_f V_f + \nu_m V_m + \frac{\left(\nu_m - \nu_f\right)\left(\frac{1}{K_f} - \frac{1}{K_m}\right) V_f V_m}{\frac{V_f}{K_m} + \frac{V_m}{K_f} + \frac{1}{G_f}} \tag{3.31}$$

$$\nu_{21} = \nu_{12} \frac{E_2}{E_1} \tag{3.32}$$

where: $E_{f,\,m}$ is Young's modulus of the fibre and matrix, respectively

$\nu_{f,\,m}$ is Poisson's ratio of the fibre and matrix, respectively

$G_{f,\,m}$ is the shear modulus of the fibre and matrix, respectively

$K_{f,\,m}$ is the bulk modulus of the fibre and matrix, respectively

Equations (3.28) and (3.31) make use of a quantity called the bulk modulus. This is basically used for determining volume changes in a body that is subjected to hydrostatic stresses [132]. Equation (3.33) gives the formula for the bulk modulus of the fibre and matrix [132].

$$K_f = \frac{E_f}{3(1-2\nu_f)} \quad , \quad K_m = \frac{E_m}{3(1-2\nu_m)} \tag{3.33}$$

The material constants are not the only properties of concern. All materials have a limit as to how much loading it may sustain. Therefore, in addition to the material constants, the limits also have to be calculated. The equations for the material limit properties are discussed next.

3.4.2.2 Evaluation of the material limits

This section examines the equations that are used for determining the material limits ($(\sigma_1^T)_{ult}$, $(\sigma_2^T)_{ult}$, $(\sigma_1^C)_{ult}$, $(\sigma_2^C)_{ult}$ and $(\tau_{12})_{ult}$). The formula for the ultimate longitudinal tensile strength ($(\sigma_1^T)_{ult}$) is given by equation (3.34), where $(\sigma_f)_{ult}$ is the ultimate tensile strength of the fibre [132].

$$\left(\sigma_1^T\right)_{ult} = \left(\sigma_f\right)_{ult}\left(V_f + \frac{E_m}{E_f}V_m\right) \tag{3.34}$$

The ultimate longitudinal compressive strength ($(\sigma_1^C)_{ult}$) is given by equation (3.35) [132].

$$\left(\sigma_1^C\right)_{ult} = \min\left[\left(\sigma_1^C\right)_{ult1} \quad , \quad \left(\sigma_1^C\right)_{ult2} \quad , \quad \left(\sigma_1^C\right)_{ult3}\right] \tag{3.35}$$

In equation (3.35), $(\sigma_1^C)_{ult}$ is the minimum value of three quantities. The first of these is obtained by equation (3.36), where $(\varepsilon_2^T)_{ult}$ is the ultimate transverse tensile strain [132]. The value for $(\varepsilon_2^T)_{ult}$ may be found using equation (3.37) [132].

$$\left(\sigma_1^C\right)_{ult1} = \frac{E_1\left(\varepsilon_2^T\right)_{ult}}{\nu_{12}} \tag{3.36}$$

$$\left(\varepsilon_2^T\right)_{ult} = \min\left[\left(\varepsilon_2^T\right)_{ult1} \quad , \quad \left(\varepsilon_2^T\right)_{ult2}\right] \tag{3.37}$$

In equation (3.37), $(\varepsilon_2^T)_{ult}$ is minimum of two values. These are determined by equations (3.38) and (3.39), where $(\sigma_m^T)_{ult}$ is the ultimate tensile strength of the matrix [132].

$$\left(\varepsilon_2^T\right)_{ult1} = \frac{\left(\sigma_m^T\right)_{ult}}{E_m}\left(1 - V_f^{1/3}\right) \tag{3.38}$$

$$\left(\varepsilon_2^T\right)_{ult2} = \frac{\left(\sigma_m^T\right)_{ult}}{E_m}\left[\frac{d}{s}\left(\frac{E_m}{E_f} - 1\right) + 1\right] \tag{3.39}$$

The *d/s* value in equation (3.39) is obtained using equation (3.40), where *d* is the diameter of the fibres and *s* is the centre-to-centre spacing between fibres [132].

$$\frac{d}{s} = \sqrt{\frac{4V_f}{\pi}} \tag{3.40}$$

The second quantity that is needed to evaluate $(\sigma_1^C)_{ult}$ in equation (3.35) is acquired from equation (3.41), where S_1^C is the extensional mode buckling stress and S_2^C is the shear mode buckling stress [132]. The values for S_1^C and S_2^C are found using equations (3.42) and (3.43), respectively [132].

$$\left(\sigma_1^C\right)_{ult2} = \min\left[S_1^C, S_2^C\right] \tag{3.41}$$

$$S_1^C = 2\left[V_f + V_m\frac{E_m}{E_f}\right]\sqrt{\frac{V_f E_m E_f}{3V_m}} \tag{3.42}$$

$$S_2^C = \frac{G_m}{V_m} \tag{3.43}$$

The last quantity needed for equation (3.35) is determined from equation (3.44), where $(\tau_f)_{ult}$ and $(\tau_m)_{ult}$ are the ultimate shear strength of the fibre and matrix, respectively [132].

$$\left(\sigma_1^C\right)_{ult3} = 2\left[\left(\tau_f\right)_{ult} V_f + \left(\tau_m\right)_{ult} V_m\right] \tag{3.44}$$

In short, the values obtained from equations (3.36), (3.41) and (3.44) are input into equation (3.35), and the minimum of these is the ultimate longitudinal compressive strength ($(\sigma_1{}^C)_{ult}$).

The next material limits to be determined are the ultimate transverse tensile ($(\sigma_2{}^T)_{ult}$) and compressive ($(\sigma_2{}^C)_{ult}$) strengths. These are found using equations (3.45) and (3.46), respectively, where $(\sigma_m{}^C)_{ult}$ is the ultimate compressive strength of the matrix [132].

$$\left(\sigma_2^T\right)_{ult} = E_2 \frac{\left(\sigma_m^T\right)_{ult}}{E_m}\left[\frac{d}{s}\left(\frac{E_m}{E_f} - 1\right) + 1\right] \tag{3.45}$$

$$\left(\sigma_2^C\right)_{ult} = E_2 \frac{\left(\sigma_m^C\right)_{ult}}{E_m}\left[\frac{d}{s}\left(\frac{E_m}{E_f} - 1\right) + 1\right] \tag{3.46}$$

The last of the material limits to be calculated is the ultimate shear strength ($(\tau_{12})_{ult}$) and this is determined by equation (3.47), where $(\tau_m)_{ult}$ is the ultimate shear strength of the matrix [132].

$$\left(\tau_{12}\right)_{ult} = G_{12} \frac{\left(\tau_m\right)_{ult}}{G_m}\left[\frac{d}{s}\left(\frac{G_m}{G_f} - 1\right) + 1\right] \tag{3.47}$$

In addition to the above material properties, there are other properties of interest. These include the coefficients of thermal and moisture expansion. The equations for these are examined.

3.4.2.3 Coefficients of thermal and moisture expansion

The coefficient of thermal expansion is defined as the change in linear dimension of a body per unit length per unit change in temperature [132]. This coefficient in directions 1 and 2 are given by equations (3.48) and (3.49), respectively, where α_f and α_m are the coefficients of thermal expansion for the fibre and matrix, respectively [132].

$$\alpha_1 = \frac{\alpha_f E_f V_f + \alpha_m E_m V_m}{E_1} \tag{3.48}$$

$$\alpha_2 = (1+\nu_f)\alpha_f V_f + (1+\nu_m)\alpha_m V_m - \alpha_1 \nu_{12} \tag{3.49}$$

The coefficient of moisture expansion is the change in linear dimension of a body per unit length per unit change in weight of moisture content per unit weight of the body [132]. Equations (3.50) and (3.51) are used to determine this coefficient in directions 1 and 2, respectively [132]. In these equations β_m is the coefficient of moisture expansion for the matrix.

$$\beta_1 = \frac{E_m}{E_1}\frac{\rho_c}{\rho_m}\beta_m \tag{3.50}$$

$$\beta_2 = (1+\nu_m)\frac{\rho_c}{\rho_m}\beta_m - \beta_1 \nu_{12} \tag{3.51}$$

All the necessary equations required for the first preliminary investigation have been examined. The next section discusses the second of the preliminary investigations which dealt with the selection of an appropriate computational coding environment.

3.4.3 Computational coding environment

In Chapter 1 the tediousness of manual calculations and the laboriousness of conventional design techniques were discussed. The design process was described as a trial and error exercise which was a repetitive procedure. It was suggested that computational coding be used to assist in the variable stiffness design process. This

would remove the tediousness and laboriousness from repetitive tasks as it would lead to the generation of commands that automates performing specific functions and/or solving given problems [135]. The developed code needed to alleviate the tedious and laborious matrix calculations, and perform all computations simply and effortlessly to avoid the need for higher computational resources. It had to allow for the design of three-dimensional structures and optimise the fibre layup parameters.

As mentioned in Chapter 1, there were software packages available that performed the complex matrix algebra easily as well as reducing the computational time. However, these packages were based on constant stiffness design techniques and would therefore not be appropriate for this (variable stiffness) study. A solution would have been to modify these packages, but this was not permitted due to propriety and copyright infringements. Hence, the proposed codes would have to be developed from the ground up using an appropriate computational software.

There are numerous programming languages and environments that may have been used to develop the various codes in this study, such as C, C++, C#, Visual Basic, etc. However, most of them, if not all, do not have built-in matrix mathematical functions, and therefore these would have to be programmed in. This would have increased the time for the study as well as digressing from the main objectives.

On the other hand, Matlab, a numerical computing package developed by MathWorks [136-137], possesses many built-in functions including those for matrix and vector algebra. It may also be interfaced with programs written in other languages, such as C, C++, C# and FORTRAN [136-139]. Further, Matlab has a development environment for Graphical User Interfaces (GUIs). All these features made Matlab the logical choice for the computational coding environment.

The above completed the preliminary investigations and the focus moved to the fulfilment of the research objectives. The next section outlines how each objective was achieved as well as the layout for the remaining chapters.

3.5 Research outline

At the beginning of this chapter, three objectives were stated. In order to fulfil each objective, goals were set out. This section discusses how each goal was achieved.

3.5.1 Fulfilment of Research Objective 1

The first goal for the first objective was to develop computational functions that performed the necessary calculations easily and swiftly. As the Hooke's Law equations describe the stress-strain relationships in a fibre reinforced composite laminate, the functions would be based on these equations. A code was developed centred on conventional design techniques and was called Constant Stiffness Method (CSM). Six inputs were required, namely, the three material-loading parameters (material constants, material limits and loading conditions) and the three fibre layup parameters (number of fibre layers, fibre orientation angle of each layer, and thickness of each layer). The outputs from this developed code were the global and local stresses and strains as well as the failure analysis. The computational functions created for this code were necessary for future coding. The development of the CSM code is examined in detail in the next chapter.

For the second goal, a code called Fibre-Angle-Opt (FAO) was developed. This code required only the three material-loading parameters as inputs. The code determined the various stresses and strains, and performed angle optimisation and failure analysis during the design process while keeping the layer thicknesses constant. Outputs from this code were the number of fibre layers and the fibre orientation angle of each layer. The development of this code is discussed in Chapter 5.2.

The FAO code was modified to incorporate a thickness optimisation function in order to achieve the third goal. This modified code was called Angle-Thick-Opt (ATO). The ATO code examined adjacent fibre layers and combined them into a single layer if certain criteria were met. Inputs for this code were the same as the FAO code, and the outputs were the complete fibre layup parameters, that is, the number of fibre layers, fibre orientation angle of each layer, and thickness of each layer. As a consequence of the thickness optimisation, the structure also underwent a reduction in weight when

compared to conventional design methods. This implied that the ATO code was also capable of weight optimisation. This code is examined in Chapter 5.3.

The above codes were developed for two-dimensional laminates, however, structures are three-dimensional entities and therefore a new design code was required to cater for this. The ATO code was thus extended to allow for the design of three-dimensional objects. This extended code completed the fourth goal and was called Variable Stiffness Method (VSM). In this method a structure was divided into simpler parts, each with its own loading conditions. The VSM code could then regard each part as a separate entity, and use the ATO code to determine each entity's fibre layup parameters. The result was a structure that had various fibre layup parameters throughout, and, as stiffness is directly related to the fibre layup parameters, it had varying stiffness as well. Therefore the structure was multi-strength, since its strength (stiffness) varied from point to point, as well as being multi-directional, because there was more than one fibre orientation angle on a particular layer. The operation of the VSM code is discussed in Chapter 5.4.

The final goal for objective 1 was the validation of all the developed computational codes. Three methods of validation have been used in this study. The first was comparison against examples from texts, while the second was the use of finite element modelling. The last validation method was experimentation.

In the development of the CSM code, validation was in the form of comparison with worked examples from Kaw [132] and other texts [4,133]. Since the final outcome was known, the comparison indicated the correctness or accuracy of the various developed computational functions. The results from this code was used to build a finite element model for validation of future developed codes.

Finite Element Modelling (FEM) or Finite Element Analysis (FEA) is a numerical technique used to find approximate solutions to partial differential equations [140]. It comprises of a numeric solver, which performs the necessary computations, and a pre/post processor, which is used for setting the input parameters (loadings, boundary

conditions, etc.) and viewing the output results (stress distribution, deflections, etc.). In engineering, FEM is used to simulate real world applications, for example, structural integrity of aircraft fuselages, dam walls, etc.; fluid flow through pipes, over surfaces, etc.; heat and radiation distribution in structures, fluids, etc. [140-142].

In a FEM package, a structure is divided into smaller and simpler parts called finite elements. This enables calculations to be performed on smaller segments rather than the structure as a whole, and this may simplify and shorten computations. Algebraic and differential equations are applied to each finite element and the results are superimposed to give the overall solution.

In this study, Patran was used for the pre/post processor. Geometry may be imported from various CAD packages or constructed in Patran itself [143]. Further, it allows for the easy creation of meshes on surfaces and solids via automated or manual methods or a combination of both. It has a built-in laminate modeller that allows the user to create a composite material with the required number of layers, fibre orientation angles and layer thicknesses. Additionally, multiple FEA solvers are supported. The solver used in this study was MSC Nastran. Engineers use it to analyse static and dynamic responses in both linear and nonlinear systems [144]. This solver may also be used in virtual prototyping as well as to determine product life and passenger comfort.

Each of the above developed codes, namely, FAO, ATO and VSM, were validated using the finite element model developed with the CSM code. In each case, the laminate/structure was designed with the relevant code and the results, namely, the fibre layup parameters, along with the material properties, were input into Patran. The stresses obtained with Matlab were compared to those obtained with the finite element package. An exact matching of values would imply an accurate design process, thus validating the relevant code.

While finite element modelling gives an accurate description or analysis of the behaviour of a structure, physical testing provides more reassurance and unmistakable validation. Therefore it was decided to fabricate and test a structure that was designed

using the VSM code. If the structure performed according to the design specifications, then the code would be further validated.

3.5.2 Fulfilment of Research Objective 2

This objective required two goals to be achieved. The first was to examine the fibre placement end-effector that was built in a previous study. The design and operation of each of the components, as well as their interaction and control, are discussed in the first part of Chapter 6.

The second part of Chapter 6 deals with the second goal which was the development of a software interface between the VSM code and the controller of the fibre placement end-effector. The code for this interface was developed in Matlab and was called VSM-TO-RFP. It converts the results from the VSM code to input code files for the end-effector's controller. These files may then be loaded into the controller to execute the fibre layup process.

3.5.3 Fulfilment of Research Objective 3

The third research objective consisted of three goals. The first was the development of Graphical User Interfaces (GUIs) to enable better interaction between the user and the developed codes. The standard Matlab environment has a Command Line Interface (CLI), that is, it accepts and executes commands one line at a time. Script files may be used for the command input but outputs are usually given in an unformatted form. The CLI also poses a problem to first-time users as commands are not obvious and the environment is unintuitive and unfriendly [145]. Further, user interaction is limited.

Therefore it is usually common practice to use a Graphical User Interface (GUI) in a programming environment. A GUI allows users to interact with a program or device through graphical icons and other visual indicators [146]. GUIs provide a more "friendly" outlook to the underlying code, and outputs may be shown in a variety of forms that are more distinguishable and recognisable. In light of this, GUIs were created in Matlab for the codes developed in this study, namely, the CSM, VSM and VSM-TO-RFP codes.

The second of the three goals was the creation of a database from which the user could choose composite materials for the design process. This composite database was to be available in the CSM and VSM GUIs as a drop-down list. The database was created from the Matlab command line via a script file. However, adding or editing entries from this database proved to be difficult from the command line.

In light of the above, a code and GUI were developed in Matlab for the management of the composite database (third goal). The user would be able to view, edit, add and delete materials/properties from the database. Additionally, databases containing the material properties for various fibres and matrices were also created. These two databases could be used to create composite materials via the micromechanics equations.

A final GUI was created that controlled the execution of the CSM, VSM, VSM-TO-RFP and database management codes and GUIs.

3.6 Summary

This chapter presented the research hypotheses and the objectives required to fulfil them. There were three objectives, each with its own goals. However, prior to achieving these goals, two preliminary investigations needed to be conducted.

The first of these investigations dealt with the examination of the equations that govern the stress-strain behaviour in fibre reinforced composites. These included Hooke's Law for two-dimensional laminates, and the Composite Micromechanics equations. In addition, the Tsai-Wu failure theory was chosen for this study, as it closely correlates with experimental results.

The second preliminary investigation dealt with the selection of an appropriate computational coding environment. Matlab was chosen due to its built-in functions for matrix algebra and its development environment for GUIs. It also has the ability to be interfaced with other programming languages.

Once the preliminary investigations were completed, the focus turned to fulfilling the research objectives and their goals. The first objective consisted of five goals, of which four required the development of computational codes. The CSM code, which was based on conventional design techniques, was developed to achieve the first goal. The second goal was fulfilled with the development of the FAO code, which performed angle optimisation while keeping the layer thicknesses constant. For the third goal, a thickness optimisation function was incorporated with the FAO code to create the ATO code. This code optimised both the fibre orientation angle and layer thickness, and consequently the weight of the laminate. The code developed for the fourth goal (VSM code) was capable of designing the fibre layup parameters for multi-strength, multi-directional structures. The last goal of the first objective was the validation of the developed codes. The validation methods included comparison against examples from texts, the use of finite element modelling, and experimental results.

There were two goals for the second objective. For the first one, the design, operation and control of the fibre placement end-effector, built in a previous study, was examined. The second goal entailed the development of a software interface (VSM-TO-RFP code) between the VSM code and the controller of the fibre placement end-effector.

The third objective contained three goals. The first was the development of GUIs for the CSM, VSM and VSM-TO-RFP codes. The second goal was the creation of a database of composite materials for use with the CSM and VSM GUIs. The code and GUI that allowed the editing, addition and deletion of materials from this database was the basis of the third goal. An additional GUI was created that controlled the execution of all developed codes and GUIs.

The following chapter discusses the constant stiffness design method and the development of the CSM code. The validation of this code was comparison against examples from literature.

4. THE CONSTANT STIFFNESS METHOD DESIGN APPROACH

4.1 Overview

In this chapter the conventional or constant stiffness design approach is presented. The discussion begins with the design procedure using the equations shown in Chapter 3. This leads to the development of the Constant Stiffness Method (CSM) code by means of the Matlab environment. The purpose of this code was to generate some of the functions needed for future coding as well as to create a finite element model for the validation of future codes.

4.2 Textbook design approach

The procedure for the conventional design of fibre reinforced composites was available from various texts. The method commonly used requires six inputs [4,132-133], namely:

- the material constants (E_1, E_2, G_{12}, v_{12} and v_{21})
- the material limits ($(\sigma_1^T)_{ult}$, $(\sigma_1^C)_{ult}$, $(\sigma_2^T)_{ult}$, $(\sigma_2^C)_{ult}$, $(\tau_{12})_{ult}$)
- applied loading conditions
- the number of layers (n)
- the fibre orientation angle of each layer (θ)
- the thickness of each layer (t)

The procedure for a typical conventional or constant stiffness design method, along with the relevant equations from Chapter 3, is as follows:

1. Calculate the [Q] matrix for the fibre reinforced laminate using equation (3.6).
2. Compute the [T] matrix for each layer using equation (3.9).
3. The [$\overline{Q}$] matrix for each layer may now be determined via equation (3.12).
4. Using equation (3.19), find the location of the top and bottom surface of each layer, h_k, where k = 1 to n.
5. Calculate the [A], [B], and [D] matrices by using equations (3.16), (3.17), and (3.18), respectively.

6. Substitute these matrices together with the applied loading conditions into equation (3.20), and solve the six simultaneous equations to determine the mid-plane strains (ε^0) and curvatures (κ).
7. The global strains at any point in the laminate may be found by equation (3.15).
8. The global stresses that correspond to the strains in step 7 may be computed using equation (3.10).
9. Using equation (3.8), calculate the local stresses and strains that correspond to the global stresses and strains found in steps 7 and 8.
10. Determine the parameters for the Tsai-Wu failure criterion via equation (3.22).
11. Using the parameters from step 10 and the local stresses found in step 9, determine the strength ratio (*SR*) values via equation (3.23).

Once the *SR* values have been calculated, the designer may then compare them to the criteria mentioned in Chapter 3 to determine whether the structure would fail or not. If failure occurs, one or more of the inputs has to be adjusted, and the above procedure would have to be repeated until there is no failure. The implication here is that there may be many iterations before the failure criterion was passed. Consequently, manually performing all the calculations required in the above procedure takes a considerable amount of time. An example of a manually calculated design is presented in Appendix A. This lead to a need for a program code that was able to perform the necessary calculations easily and efficiently. The developed code was named the Constant Stiffness Method (CSM).

4.3 The Constant Stiffness Method (CSM) code

The CSM code follows the steps shown in Figure 4.1 which was based on the above procedure. First the code prompts the user for the six input parameters, namely, the material constants, material limits, loading conditions, number of fibre layers, fibre orientation angle of each layer, and thickness of each layer. Next the [Q], [T], and [$\overline{Q}$] matrices as well as the h_k values are determined. All data concerning each layer (E_1, E_2, G_{12}, ν_{12}, θ, t, [Q], [T], [$\overline{Q}$] and h_k) are stored in an array. The array is used to calculate the [A], [B], and [D] matrices. These matrices together with the applied

loading conditions are used to compute the mid-plane strains (ε^0) and curvatures (κ). Thereafter the global and local stresses and strains are calculated. The local stresses are compared to the material limits, via the Tsai-Wu failure criterion, to determine whether failure occurs in the composite laminate.

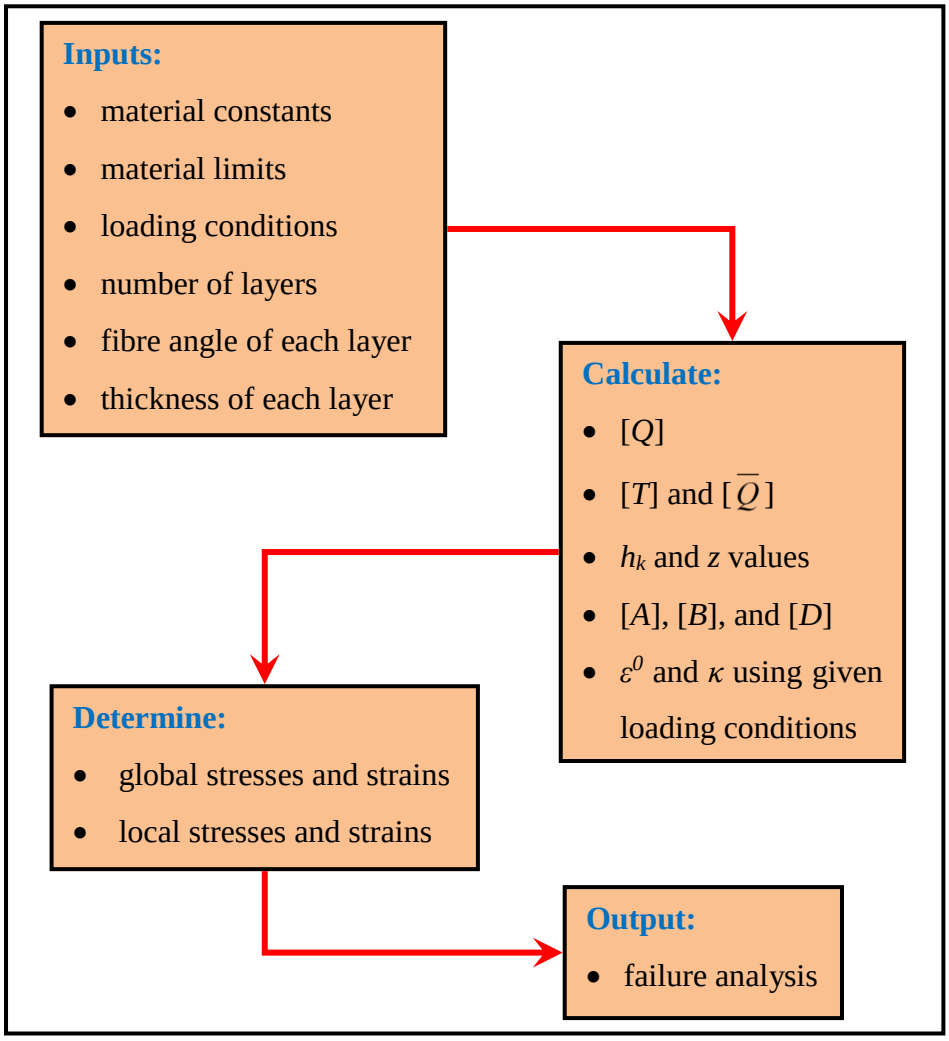


Figure 4.1: Flowchart illustrating the Constant Stiffness Method (CSM) code

The computation of the various matrices and other relevant data was each carried out in its own separate function. This means there was one function to determine the [Q] matrix, another to calculate the [T] matrix, and so on. The purpose of the separate functions was to enable easier coding in future. A script file was written that controlled the use of each function. The Matlab code represented by the above flowchart is shown and examined line by line in Appendix B.

4.4 Testing of the developed code

The CSM code was tested against manually calculated examples in the available texts [132-133], and the results were exactly comparable to the manual computations. In an example (Example 4.3) by Kaw [132], the stresses and strains in a graphite/epoxy composite laminate were examined. This laminate consisted of three layers, with fibre orientation angles of 0°, 30° and -45°, and each layer had a thickness of 5mm. The material constants, material limits and loading conditions for this example are given in Table 4.1. The calculated global strains, global stresses, local strains and local stresses are shown in Tables 4.2 to 4.5, respectively, for the top, middle and bottom of each layer.

Table 4.1: Material constants, material limits and loading conditions for graphite/epoxy composite example in Kaw [132]

Material Constants				Material Limits					Loading	
E_1 (GPa)	E_2 (GPa)	G_{12} (GPa)	ν_{12}	$(\sigma_1^T)_{ult}$ (MPa)	$(\sigma_1^C)_{ult}$ (MPa)	$(\sigma_2^T)_{ult}$ (MPa)	$(\sigma_2^C)_{ult}$ (MPa)	$(\tau_{12})_{ult}$ (MPa)	Forces (N)	Moments (Nm)
181	10.3	7.17	0.28	1500	1500	40	246	68	1000	0
									1000	0
									0	0

Table 4.2: Global strains from Example 4.3 in Kaw [132]

Global Strains (m/m) in Example 4.3				
Ply #	**Position**	ε_x	ε_y	γ_{xy}
1 (0°)	Top	8.944 (10^{-8})	5.955 (10^{-6})	–3.836 (10^{-6})
	Middle	1.637 (10^{-7})	5.134 (10^{-6})	–2.811 (10^{-6})
	Bottom	2.380 (10^{-7})	4.313 (10^{-6})	–1.785 (10^{-6})
2 (30°)	Top	2.380 (10^{-7})	4.313 (10^{-6})	–1.785 (10^{-6})
	Middle	3.123 (10^{-7})	3.492 (10^{-6})	–7.598 (10^{-7})
	Bottom	3.866 (10^{-7})	2.670 (10^{-6})	2.655 (10^{-7})
3 (–45°)	Top	3.866 (10^{-7})	2.670 (10^{-6})	2.655 (10^{-7})
	Middle	4.609 (10^{-7})	1.849 (10^{-6})	1.291 (10^{-6})
	Bottom	5.352 (10^{-7})	1.028 (10^{-6})	2.316 (10^{-6})

The values in Table 4.1 and the fibre orientation angles of 0°, 30° and -45° were used as inputs in the CSM code. The global and local stresses and strains were computed

and are shown in Figures 4.2 to 4.5. The stress and strain values in each of these figures were exactly comparable to the corresponding values in Tables 4.2 to 4.5. The conclusion from this was that the CSM code accurately determined the stresses and strains in a unidirectional fibre reinforced composite laminate.

Table 4.3: Global stresses from Example 4.3 in Kaw [132]

Global Stresses (Pa) in Example 4.3				
Ply #	Position	σ_x	σ_y	τ_{xy}
1 (0°)	Top	3.351 (10^4)	6.188 (10^4)	–2.750 (10^4)
	Middle	4.464 (10^4)	5.359 (10^4)	–2.015 (10^4)
	Bottom	5.577 (10^4)	4.531 (10^4)	–1.280 (10^4)
2 (30°)	Top	6.930 (10^4)	7.391 (10^4)	3.381 (10^4)
	Middle	1.063 (10^5)	7.747 (10^4)	5.903 (10^4)
	Bottom	1.434 (10^5)	8.102 (10^4)	8.426 (10^4)
3 (–45°)	Top	1.235 (10^5)	1.563 (10^5)	–1.187 (10^5)
	Middle	4.903 (10^4)	6.894 (10^4)	–3.888 (10^4)
	Bottom	–2.547 (10^4)	–1.840 (10^4)	4.091 (10^4)

Table 4.4: Local strains from Example 4.3 in Kaw [132]

Local Strains (m/m) in Example 4.3				
Ply #	**Position**	ε_1	ε_2	γ_{12}
1 (0°)	Top	8.944 (10^{-8})	5.955 (10^{-6})	–3.836 (10^{-6})
	Middle	1.637 (10^{-7})	5.134 (10^{-6})	–2.811 (10^{-6})
	Bottom	2.380 (10^{-7})	4.313 (10^{-6})	–1.785 (10^{-6})
2 (30°)	Top	4.837 (10^{-7})	4.067 (10^{-6})	2.636 (10^{-6})
	Middle	7.781 (10^{-7})	3.026 (10^{-6})	2.374 (10^{-6})
	Bottom	1.073 (10^{-6})	1.985 (10^{-6})	2.111 (10^{-6})
3 (–45°)	Top	1.396 (10^{-6})	1.661 (10^{-6})	–2.284 (10^{-6})
	Middle	5.096 (10^{-7})	1.800 (10^{-6})	–1.388 (10^{-6})
	Bottom	–3.766 (10^{-7})	1.940 (10^{-6})	–4.928 (10^{-7})

The CSM code also performed the failure analysis via the Tsai-Wu failure theory. The resulting *SR* values for the top, middle and bottom of each layer are shown in Figure 4.6. The rows represent the layers and columns 1 to 3 represent the top, middle and bottom of each layer, respectively. The *SR* values in Figure 4.6 indicated that the composite laminate would have first failed on layer 1 at approximately 61% of the

applied load (*SR* = 0.6147). It was evident from this that the input fibre orientation angles or the number of layers, or both of these were inadequate and needed adjustment for the laminate to carry the applied loading. Further verification of the computational results was obtained via a finite element analysis.

Table 4.5: Local stresses from Example 4.3 in Kaw [132]

Local Stresses (Pa) in Example 4.3

Ply #	Position	σ_1	σ_2	τ_{12}
1 (0°)	Top	3.351 (10^4)	6.188 (10^4)	–2.750 (10^4)
	Middle	4.464 (10^4)	5.359 (10^4)	–2.015 (10^4)
	Bottom	5.577 (10^4)	4.531 (10^4)	–1.280 (10^4)
2 (30°)	Top	9.973 (10^4)	4.348 (10^4)	1.890 (10^4)
	Middle	1.502 (10^5)	3.356 (10^4)	1.702 (10^4)
	Bottom	2.007 (10^5)	2.364 (10^4)	1.513 (10^4)
3 (–45°)	Top	2.586 (10^5)	2.123 (10^4)	–1.638 (10^4)
	Middle	9.786 (10^4)	2.010 (10^4)	–9.954 (10^3)
	Bottom	–6.285 (10^4)	1.898 (10^4)	–3.533 (10^3)

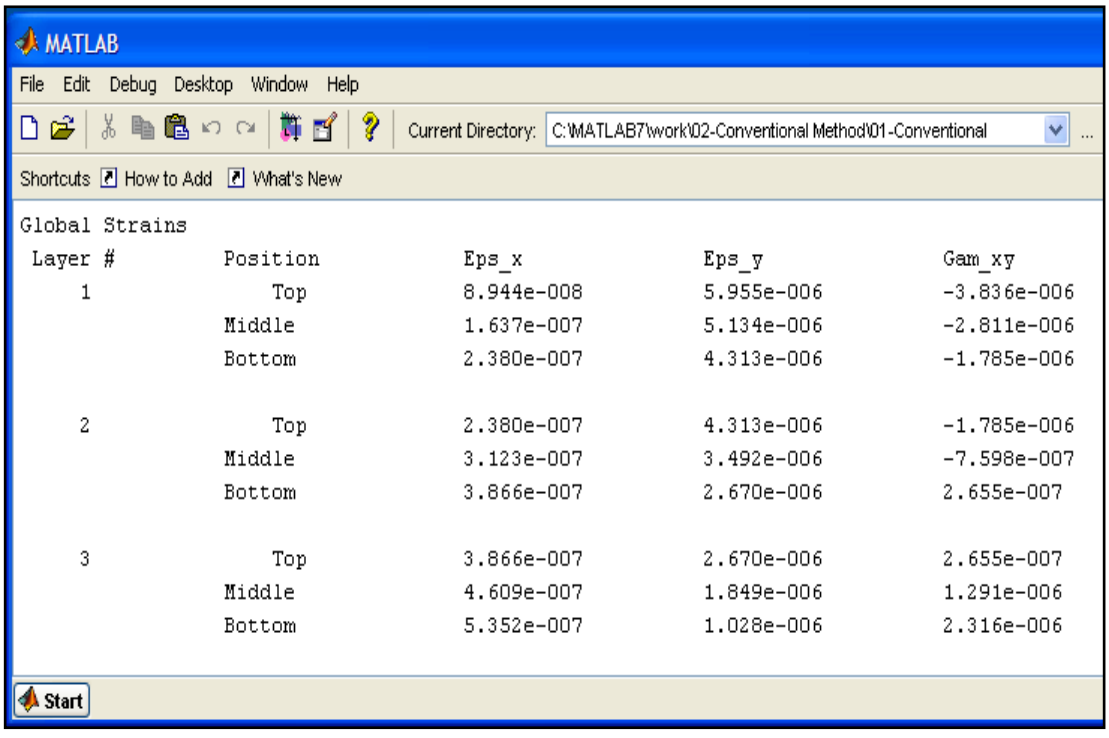
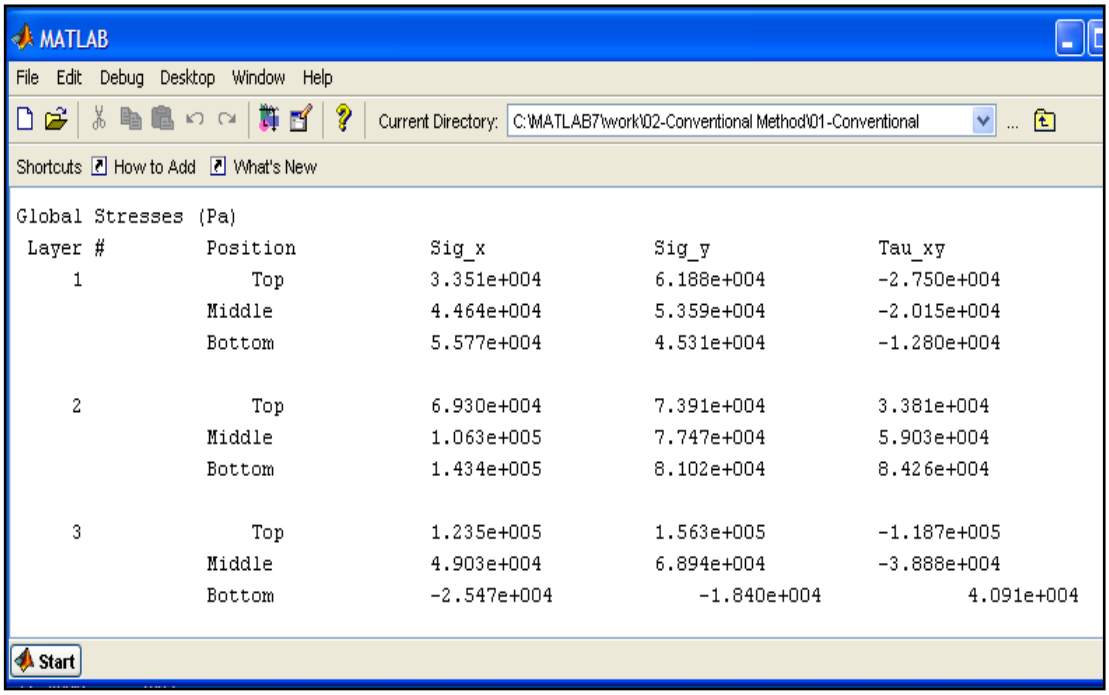


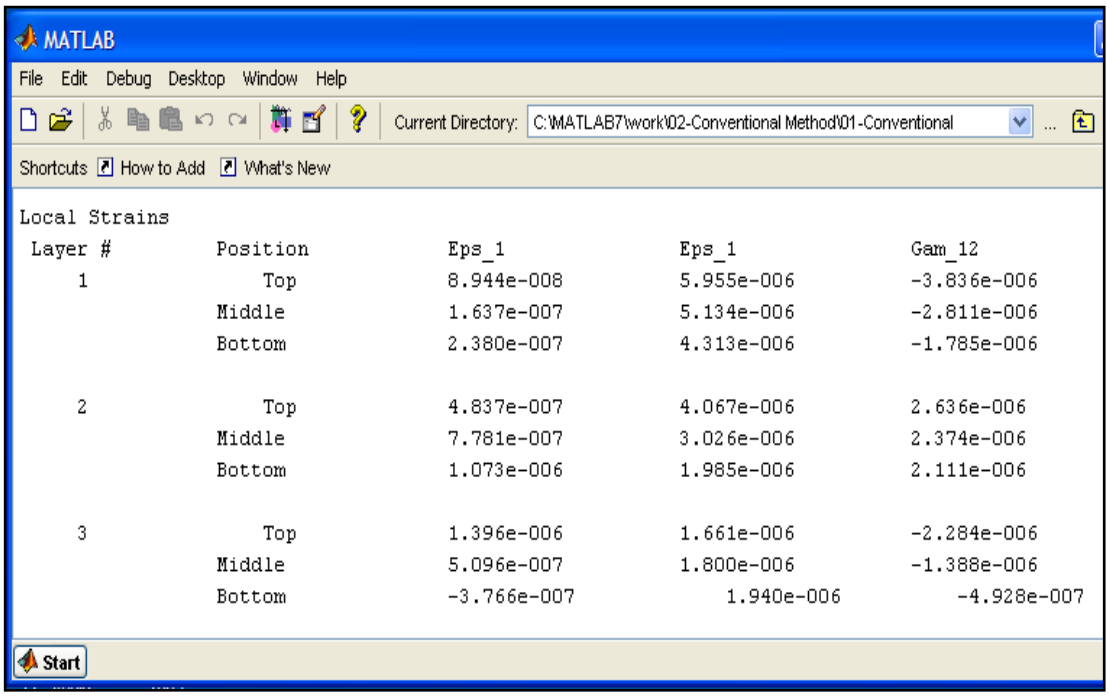
Figure 4.2: Global strains from Constant Stiffness Method (CSM) code with Table 4.1 as inputs



Global Stresses (Pa)

Layer #	Position	Sig_x	Sig_y	Tau_xy
1	Top	3.351e+004	6.188e+004	-2.750e+004
	Middle	4.464e+004	5.359e+004	-2.015e+004
	Bottom	5.577e+004	4.531e+004	-1.280e+004
2	Top	6.930e+004	7.391e+004	3.381e+004
	Middle	1.063e+005	7.747e+004	5.903e+004
	Bottom	1.434e+005	8.102e+004	8.426e+004
3	Top	1.235e+005	1.563e+005	-1.187e+005
	Middle	4.903e+004	6.894e+004	-3.888e+004
	Bottom	-2.547e+004	-1.840e+004	4.091e+004

Figure 4.3: Global stresses from Constant Stiffness Method (CSM) code with Table 4.1 as inputs



Local Strains

Layer #	Position	Eps_1	Eps_1	Gam_12
1	Top	8.944e-008	5.955e-006	-3.836e-006
	Middle	1.637e-007	5.134e-006	-2.811e-006
	Bottom	2.380e-007	4.313e-006	-1.785e-006
2	Top	4.837e-007	4.067e-006	2.636e-006
	Middle	7.781e-007	3.026e-006	2.374e-006
	Bottom	1.073e-006	1.985e-006	2.111e-006
3	Top	1.396e-006	1.661e-006	-2.284e-006
	Middle	5.096e-007	1.800e-006	-1.388e-006
	Bottom	-3.766e-007	1.940e-006	-4.928e-007

Figure 4.4: Local strains from Constant Stiffness Method (CSM) code with Table 4.1 as inputs

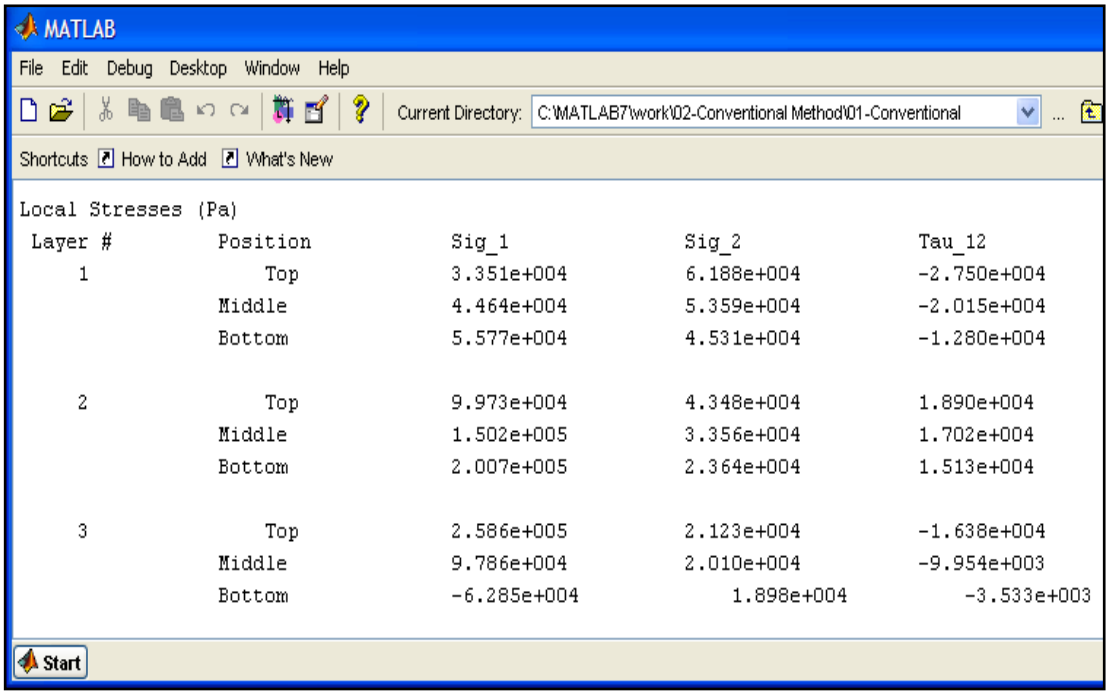


Figure 4.5: Local stresses from Constant Stiffness Method (CSM) code with Table 4.1 as inputs

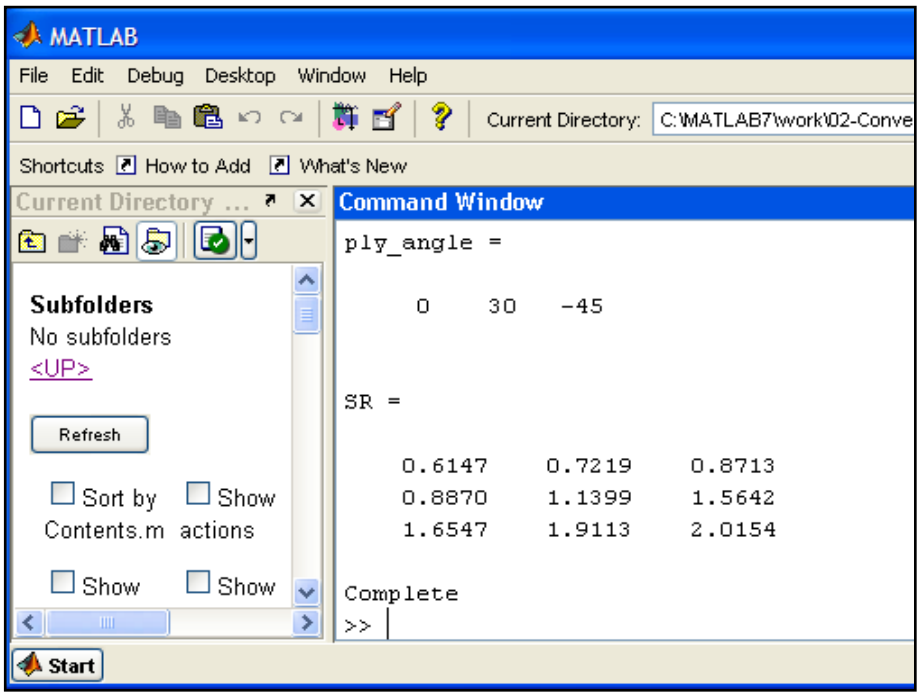


Figure 4.6: *SR* ratios from Constant Stiffness Method (CSM) code with Table 4.1 as inputs

4.5 Finite element analysis

A finite element model representing a laminated composite plate was set up in Patran. This plate consisted of the material constants, material limits and loading conditions shown in Table 4.1. The fibre orientation angles used were 0°, 30°, and -45° and each layer was 5mm thick. The analysis was run and the resulting fringe plots of the stress components of each layer were plotted. These plots are shown in Figure 4.7.

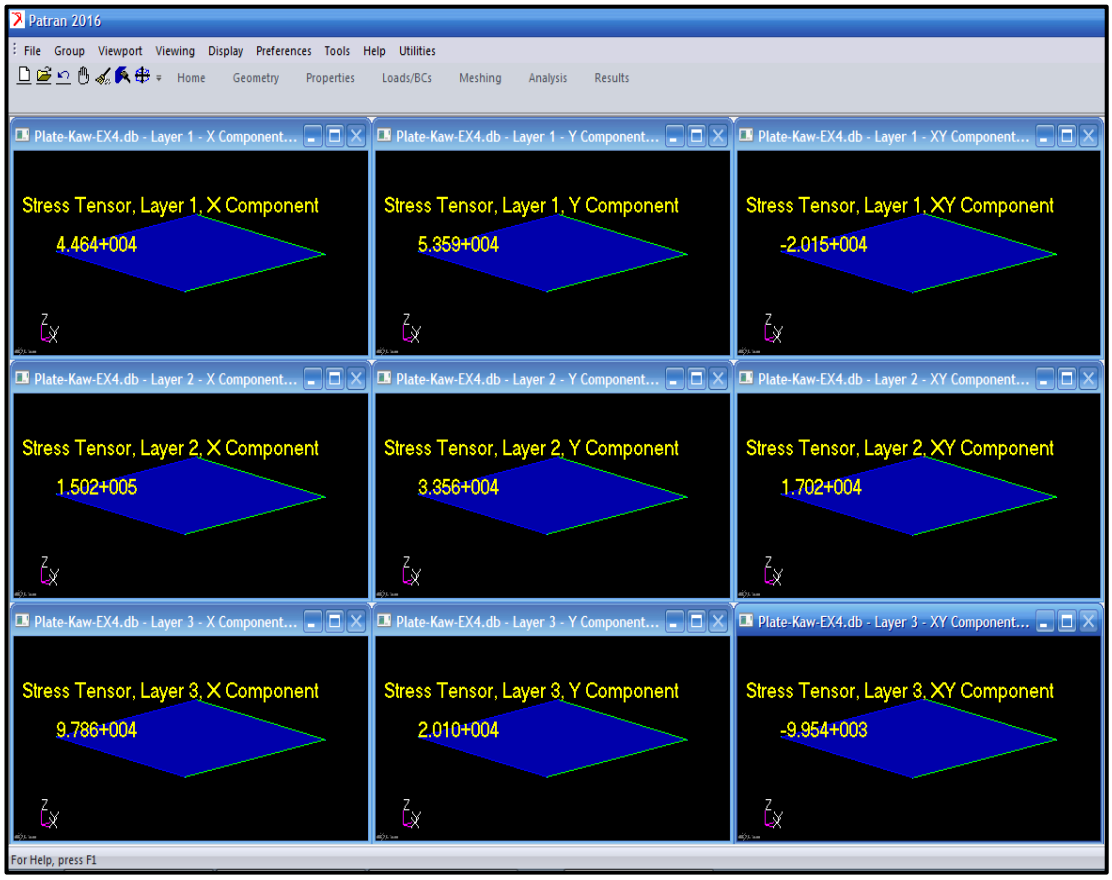


Figure 4.7: Fringe plots of stress components from Patran for 0°, 30°, and -45° laminate with Table 4.1 as inputs

The stress plots from the finite element package were compared to the local stresses experienced by the laminate analysed with the CSM code. It may be seen that the stress values in Figure 4.7 were exactly equal to those incurred at the middle of each layer in either Table 4.5 or Figure 4.5. This implied that the finite element model was correctly established, and that it may be used for future codes to validate the local mid-layer stresses incurred in unidirectional fibre reinforced laminates.

4.6 Graphical User Interface (GUI) for the CSM code

As mentioned in Chapter 3, the outputs in Matlab are in an unformatted form, and it is common practice to use a Graphical User Interface (GUI). Hence a GUI for the CSM code was developed in the Matlab environment and is shown in Figure 4.8. It consists of two panels, namely, the Input Panel and the Output Panel. These are discussed in further detail.

Figure 4.8: Graphical User Interface for Constant Stiffness Method (CSM) code

The Input Panel is shown in Figure 4.9, and it contains two tables, two sets of drop-down lists, and nine buttons. The button labelled H displays the Help file, which is shown in Figure 4.10. It gives a brief description of the GUI, and outlines the operational procedures that must be followed to correctly use it. The "EXIT" button, labelled as X, closes this GUI.

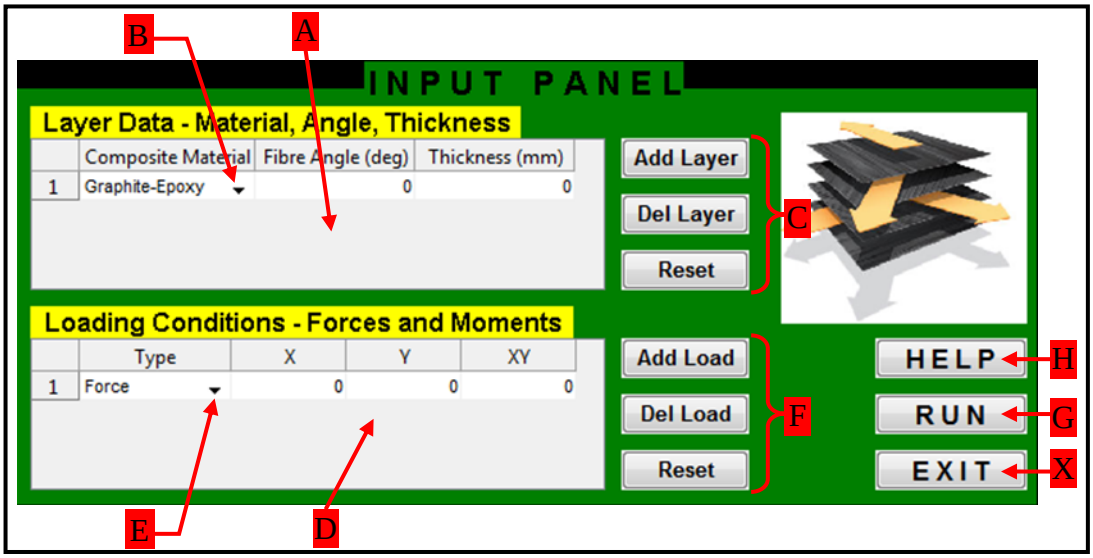


Figure 4.9: Input Panel of the CSM code's Graphical User Interface

The first table, labelled A, is where the composite materials, fibre orientation angles and layer thicknesses are entered. Each row on the table represents a fibre layer. The composite material to be used is selected from a database of materials via the drop-down list labelled B. A different material may be selected for each layer. The angle and thickness values are entered manually. The set of buttons labelled C is used for the addition of a layer ("Add Layer" button), deletion of a layer ("Del Layer" button), or clears the table of all values ("Reset" button).

The second table, labelled D in Figure 4.9, is the input interface for the loading conditions. The type of loading, either force or moment, is selectable from the drop-down list labelled E. For each loading condition, the x-, y-, and xy-components need to be entered manually. The set of buttons labelled F adds a load ("Add Load"), removes a load ("Del Load"), or clears the table of all values ("Reset").

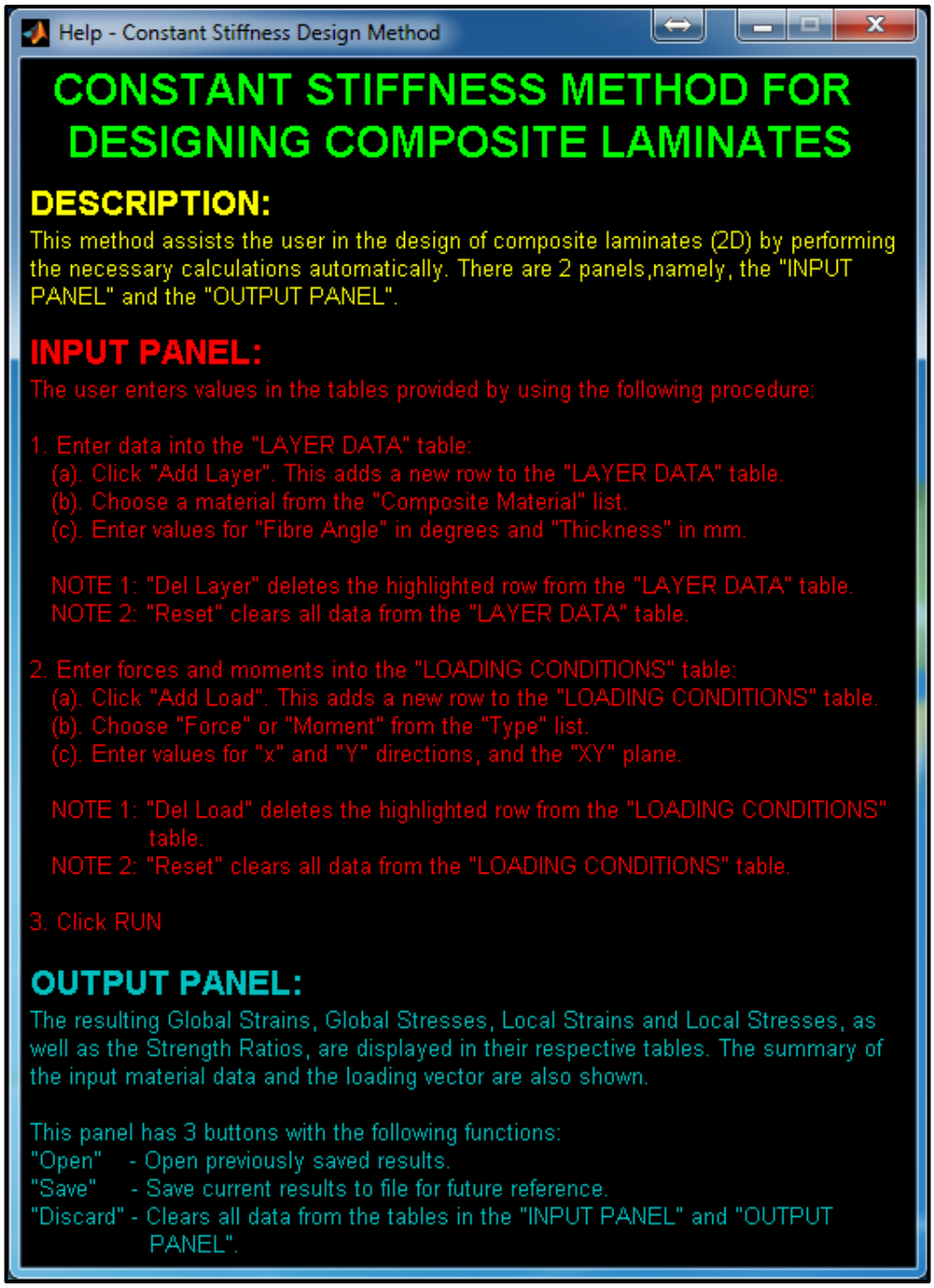


Figure 4.10: Help function for the CSM code

Once the two tables are populated with the desired input values, the calculation of the global and local stresses and strains, and the failure analysis may be initiated via the "RUN" button, labelled G. Upon initiation, the code first verifies that the tables are not empty and that the table entries are valid numerical values. If an error is

encountered, a message is displayed to prompt the user to rectify the situation. After all errors are fixed, the material constants and material limits for the selected materials are imported from the materials database, which is discussed later in Chapter 7. The various computations are performed via the CSM code, and the results are displayed in the Output Panel which is shown in Figure 4.11.

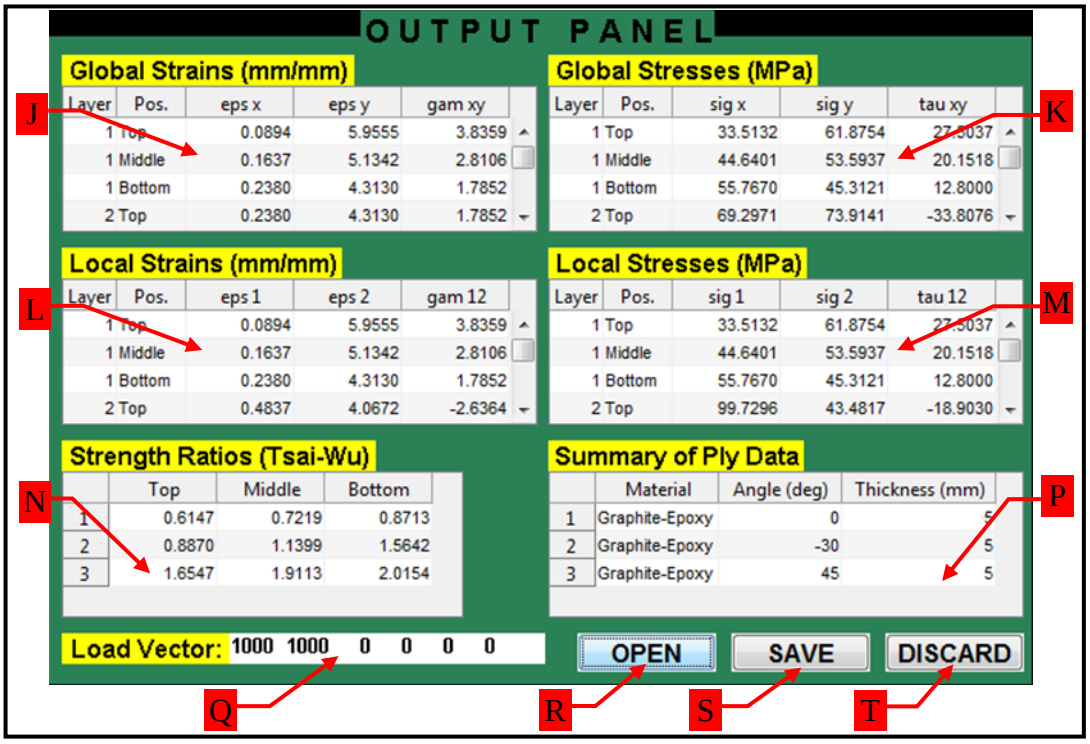


Figure 4.11: Output Panel of the CSM code's Graphical User Interface

The Output Panel consists of six tables, one vector, and three buttons. After all the calculations have been performed, the global strains, global stresses, local strains and local stresses are displayed in the tables labelled J, K, L and M, respectively. Each table is appropriately titled and displays the units of it values. The table labelled N displays the strength ratios that were determined using the Tsai-Wu failure criterion. A summary of the input materials, fibre orientation angles and layer thicknesses is shown in the table labelled P. This data is important when viewing saved files as it displays the inputs that were used. The loading vector, denoted by label Q, is of similar importance and is also displayed. The "OPEN" button, labelled as R, is used to view previously saved results, and the "SAVE" button, labelled as S, saves the current

results. The “DISCARD” button (labelled T) clears all the data from the tables in both panels. The Matlab code for the above GUI is examined line-by-line in Appendix B.

4.7 Summary

The procedure commonly used in conventional or constant stiffness design methods was outlined. It was established that many iterations of the procedure would be necessary for a design to pass the Tsai-Wu failure criterion. This in turn would lead to numerous and laborious manual calculations. Therefore a computational code was developed in Matlab that would perform all relevant computations easily and efficiently. The code eliminated the laboriousness of the calculations and resulted in quicker computational times. This fulfilled the first goal of the first research objective.

The code was named Constant Stiffness Method (CSM), and was based on the procedure outlined at the beginning of the chapter. It required six inputs, namely, the material constants, material limits, loading conditions, number of layers, fibre orientation angle of each layer, and thickness of each layer. The code was divided into functions where each function performed a specific calculation. This improved efficiency and enabled easier coding for future codes. The outputs of the code included the global and local stresses and strains, the strength ratios, and the Tsai-Wu failure analysis.

The CSM code was tested against a manually calculated example and the results were exactly comparable. It was concluded that the program code was able to correctly design unidirectional fibre reinforced composite laminates. Finite element modelling was used for further verification, and, here as well, the results were exactly the same as that obtained with the CSM code. The finite element model would be used to validate the codes developed later. A Graphical User Interface (GUI) was developed in Matlab to enhance user interaction and allow for easier input of values and viewing of output results.

Although the CSM code performed the calculations easily and efficiently and reduced the design process time, it did not remove all of the disadvantages associated with

conventional design methods as discussed in Chapter 1. These included uncertainty in which of the fibre layup parameters to change when failure occurs, non-optimised fibre orientation angles and layer thicknesses, and overdesigned structures. Further, the code was not able to design multi-strength, multi-directional structures.

Clearly the CSM code did not meet the requirements set out for the primary objective of this study. The next chapter discusses the development of other codes that achieve this objective. These codes include the Fibre-Angle-Opt code, which optimises just the fibre orientation angles, the Angle-Thick-Opt code, which combines the Fibre-Angle-Opt code with a thickness optimisation function, and the Variable Stiffness Method code, which is used for the design of multi-strength, multi-directional structures.

5. THE VARIABLE STIFFNESS METHOD DESIGN APPROACH

5.1 Overview

The primary objective of this study, which was to develop a new design approach that optimised the fibre layup parameters (number of fibre layers, fibre orientation angle of each layer and thickness of each layer) of fibre reinforced composite structures, is addressed in this chapter. Further, the design approach should have the capability of creating multi-strength and multi-directional (variable stiffness) structures. Computational codes developed in Matlab were used to achieve this objective.

Three codes were developed for the optimisation process, namely, Fibre-Angle-Opt (FAO), Angle-Thick-Opt (ATO) and Variable Stiffness Method (VSM). The FAO code kept the layer thicknesses fixed and optimised the fibre orientation angles only. A thickness optimisation function was then integrated with this code to create the ATO code, which optimised both the fibre orientation angles and layer thicknesses. The operation of the ATO code was enhanced to form the VSM code, which was capable of creating multi-strength and multi-directional fibre reinforced composite structures.

Unlike the CSM code, the above codes only required the material-loading parameters (material constants, material limits and loading conditions) as inputs. In this study, the fibre orientation angle and layer thickness were the design variables that were directly optimised, while the optimisation of the number of layers was a consequence of these optimisations. A further result of the above optimisation processes was the weight optimisation of the composite structure. The operation of each of the above codes are examined in turn.

5.2 The Fibre-Angle-Opt (FAO) code

The FAO code was based on Hooke's Law and made use of the computational functions developed with the CSM code. The FAO code iteratively varied the fibre orientation angles while keeping the layer thicknesses constant in order to design a composite structure that was able to sustain the applied loading conditions. The fibre orientation angles and number of layers were the outputs of this code.

The operation of the FAO code may best be described by the flowchart in Figure 5.1. First the material constants and material limits would have to be entered (Block 1). Using this data, the [*Q*] matrix (Block 2) and the Tsai-Wu parameters (Block 3) may be immediately calculated as these do not vary with the fibre orientation angle but rather with the material properties. Next step involves the input of the applied loading conditions (Block 4) and the evaluation of the resultant forces and moments (Block 5).

The following block (Block 6) is where the relevant matrices and the various stresses and strains are computed. This block is principally the CSM code, and therefore the fibre layup parameters also need to be known. However, these were not part of the input variables and hence the code makes an initial assumption. The assumption is that the laminate has one layer with a fibre orientation angle of 0° and a specified fixed thickness. This assumption would be made for each new layer added to the laminate.

The above assumption is used in Block 6 to compute the h_k and z values as well as the [*T*], $[\overline{Q}]$, [*A*], [*B*] and [*D*] matrices. Further, the mid-plane strains (ε^0) and curvatures (κ), and, the global and local stresses and strains are also evaluated. In Block 7, the Tsai-Wu failure theory is applied and values for the strength ratio (*SR*) are obtained. The *SR* values are then analysed (Block 8) to verify if failure occurred. As discussed in Chapter 3, the ideal value for *SR* would be 1, however, obtaining this value proved to be difficult in that many iterations of the code was required. This did not effectively reduce the design time. Therefore it was decided to allow *SR* to fall within a range of values. The lower limit of this range has to be 1, as any value below this would mean failure, and the upper limit may be set to any value above 1. In this study, the upper limit was set to 1.2 to ensure convergence of values as well as to avoid overdesigning.

If, in Block 8, the *SR* values for each layer are between 1 and 1.2, then the design of the laminate would be complete and the outputs are returned (Block 13). On the other hand, if the *SR* value for any of the layers is not in this range, then the fibre orientation angle, for that particular layer, is varied (Block 9) in order to obtain a better *SR* value, that is, a value closer to the range between 1 and 1.2. Since the fibre orientation angles

change, the matrix calculations are repeated (Block 10) to attain new *SR* values. In Block 11, the new and old values are compared, and the fibre orientation angle with the better *SR* value is retained.

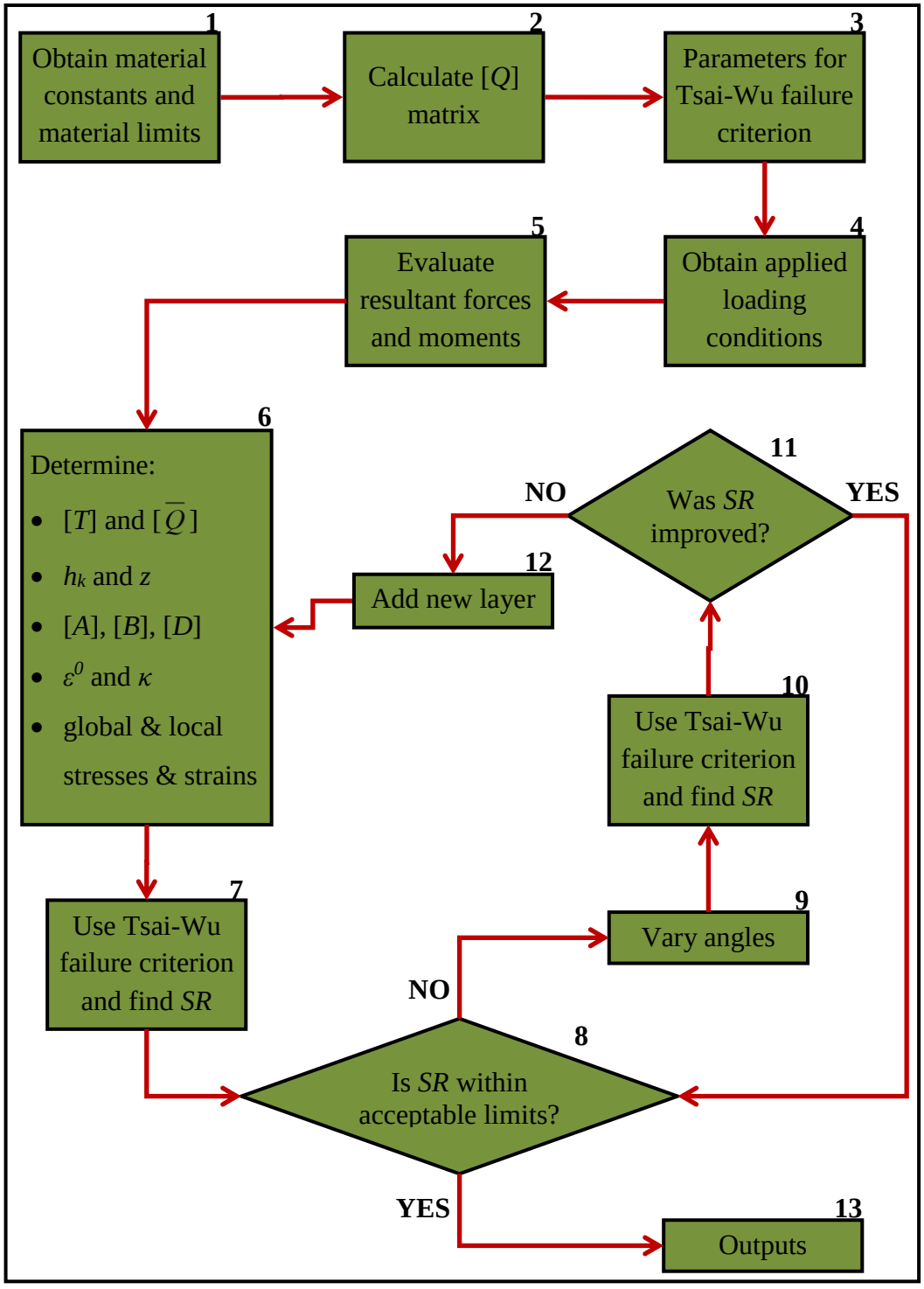


Figure 5.1: Flowchart describing the Fibre-Angle-Opt (FAO) code

For example, assume there was a layer with a fibre orientation angle of 30° and a *SR* value of 0.8. This value would not be within acceptable limits and therefore the angle would be changed in an attempt to obtain a better *SR* value. Assume that at 40° the *SR* value was found to be 0.9, and since this would be better than the previous value, the angle of the 30° layer would be changed to 40°. However, if at 40° the *SR* value was 0.7, then the 30° angle would be retained.

At this stage (Block 11), it is determined whether any *SR* values were improved, and, if this is the case, then the process in Block 8 is repeated. If the *SR* values are within the specified range, then the outputs are returned (Block 13), otherwise the fibre orientation angles are varied (Block 9). The procedure in Blocks 8 to 11 is repeated until the *SR* values for each layer of the laminate are either within the specified range or cannot be improved upon anymore. If no more improvement can be made and the values are not within the specified range, a new layer is added via Block 12. Since the number of layers have changed, the various matrices, stresses, strains, and *SR* values have to be recalculated using Block 6.

The above procedure (Blocks 6 to 12) is the angle optimisation process. It is repeated until the design criterion is achieved, that is, until the *SR* values for each layer of the laminate is between 1 and 1.2. Once this is achieved, the outputs are returned (Block 13), which, at this stage, is the number of layers and the fibre orientation angle of each layer. The FAO code is shown, with a line-by-line explanation, in Appendix C.

The FAO code was tested with different sets of loading conditions and material properties. One of these was the graphite/epoxy laminate used to validate the CSM code. The material constants, material limits and loading conditions for this laminate were given in Table 4.1. These values, and a fixed layer thickness of 5mm, were used as inputs in the FAO code. The outputs from the code are shown in Figure 5.2. It may be seen that, although the number of layers required was the same as in the case of the CSM code, the fibre orientation angles are now different. With the FAO code, the required fibre orientation angles are 49°, -79° and 49° instead of the previous 0°, 30°

and -45°. Further, all the *SR* values in Figure 5.2 are between 1 and 1.2 indicating that there was no failure or overdesign.

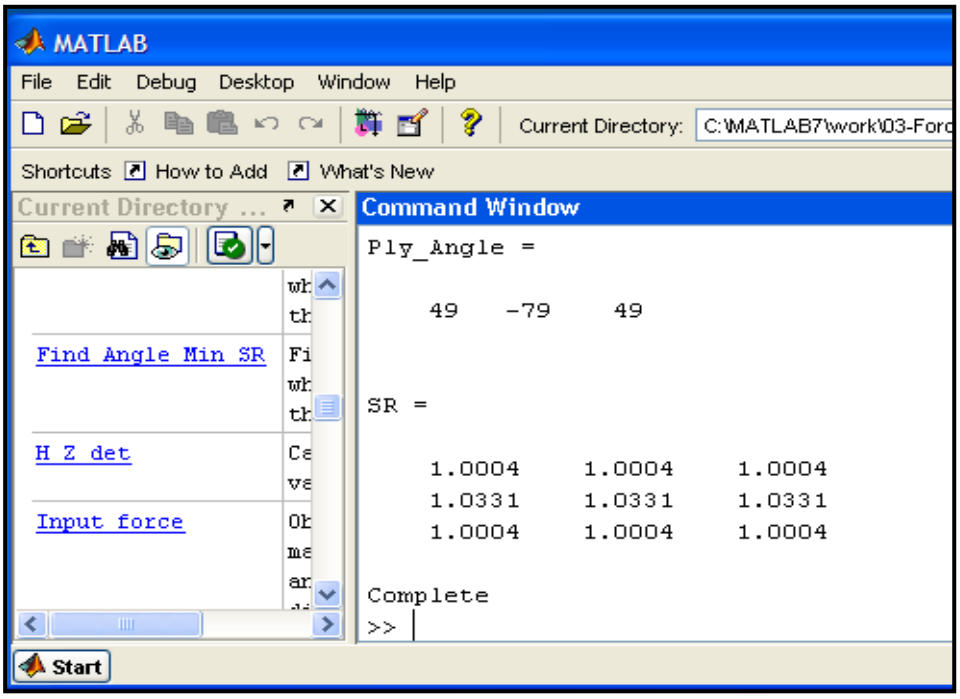


Figure 5.2: Results from the FAO code with Table 4.1 and fixed layer thicknesses of 5mm as inputs

The above results needed to be validated and the CSM code was used for this since its operation was previously corroborated. The new fibre orientation angles (49°, -79° and 49°), along with the material-loading parameters in Table 4.1, were used as inputs in the CSM code. The resulting *SR* values were equal to those in Figure 5.2. Further, the laminate designed with the CSM code was only able to sustain approximately 61% of the applied loading before failure occurred. This was evident from Figure 4.6 where the lowest *SR* value was 0.6147. In contrast, the laminate designed with the FAO code was capable of bearing the full applied loading conditions without failure (*SR* value of 1 in Figure 5.2).

Further evidence of the increase in load carrying capability may be obtained by the comparison of the local stresses of the laminates designed by the FAO code (Figure 5.3) and the CSM code (Figure 4.5). In Figure 4.5, the value at the top of the first layer

was 33.5 MPa, while, for the same position in Figure 5.3, the value is 75.7 MPa. This clearly demonstrates that the load carrying capability has increased. Similar trends may be seen in other areas where the laminate had failed with the CSM code.

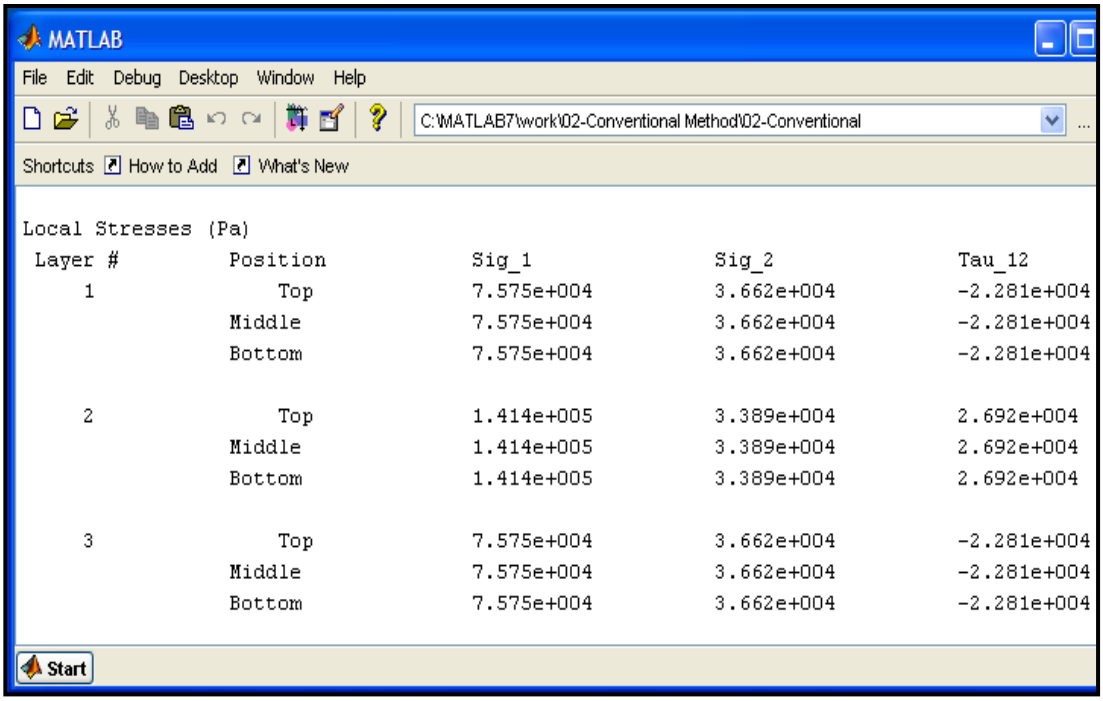


Figure 5.3: Local stresses for the laminate with fibre orientation angles of 49°, -79°, and 49°, and fixed layer thicknesses of 5mm

Additional validation of the FAO code was obtained via finite element analysis with Patran. The finite element model that was used was identical to the one employed in the validation of the CSM code, with the exception of the fibre orientation angles. In the new model the fibre orientation angles were 49°, -79°, and 49°. The fringe plots of the X, Y and XY components of the stress tensor for each layer are shown in Figure 5.4. These results are exactly comparable with those in Figure 5.3 for the middle of each layer, and this validates the FAO code as the design values are equal to the simulated values.

The above proves that the FAO code designs accurately. Now it must be shown that it designs optimally. Therefore, it was decided to investigate whether an increase in the loading conditions for the above laminate would result in failure. The applied loads

were increased by 10%, while the material constants and material limits were left unchanged. The new material-loading parameters are shown in Table 5.1. These values, together with the fibre orientation angles of 49°, -79° and 49°, and fixed layer thicknesses of 5mm, were used as inputs in the CSM code. The resulting *SR* values, shown in Figure 5.5, indicated that failure would occur as all the *SR* values were less than 1. Hence it may be deduced that the FAO code optimally designed a composite laminate for the material-loading parameters in Table 4.1, with fixed layer thicknesses of 5mm, as an increase in loading conditions resulted in failure.

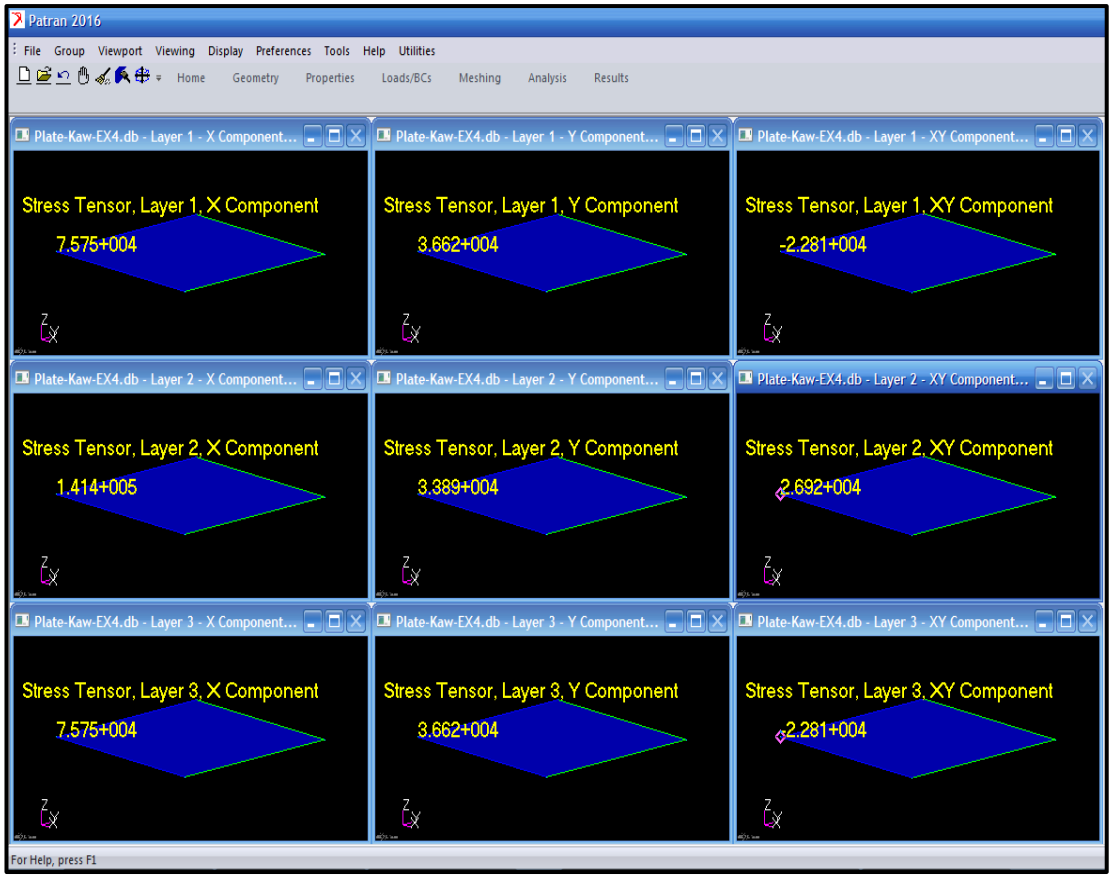


Figure 5.4: Patran fringe plots of stress components for the laminate with fibre orientation angles of 49°, -79°, and 49°, and fixed layer thicknesses of 5mm

The FAO code was able to optimally design a composite laminate for the given material-loading parameters with fixed layer thicknesses. However, this was not a desirable solution because a fixed thickness may lead to overdesign and invariably increase the fabrication costs as more fibres than necessary would be used. Therefore

the solution would be to develop a code that was capable of optimising both the fibre orientation angles and the layer thicknesses. This code was called Angle-Thick-Opt (ATO).

Table 5.1: Material constants, material limits and loading conditions for 49°, -79° and 49° laminate with 10% increase in applied loads

Material Properties				Material Limits					Loading	
E_1 (GPa)	E_2 (GPa)	G_{12} (GPa)	ν_{12}	$(\sigma_1^T)_{ult}$ (MPa)	$(\sigma_1^C)_{ult}$ (MPa)	$(\sigma_2^T)_{ult}$ (MPa)	$(\sigma_2^C)_{ult}$ (MPa)	$(\tau_{12})_{ult}$ (MPa)	Forces (N)	Moments (Nm)
181	10.3	7.17	0.28	1500	1500	40	246	68	1100	0
									1100	0
									0	0

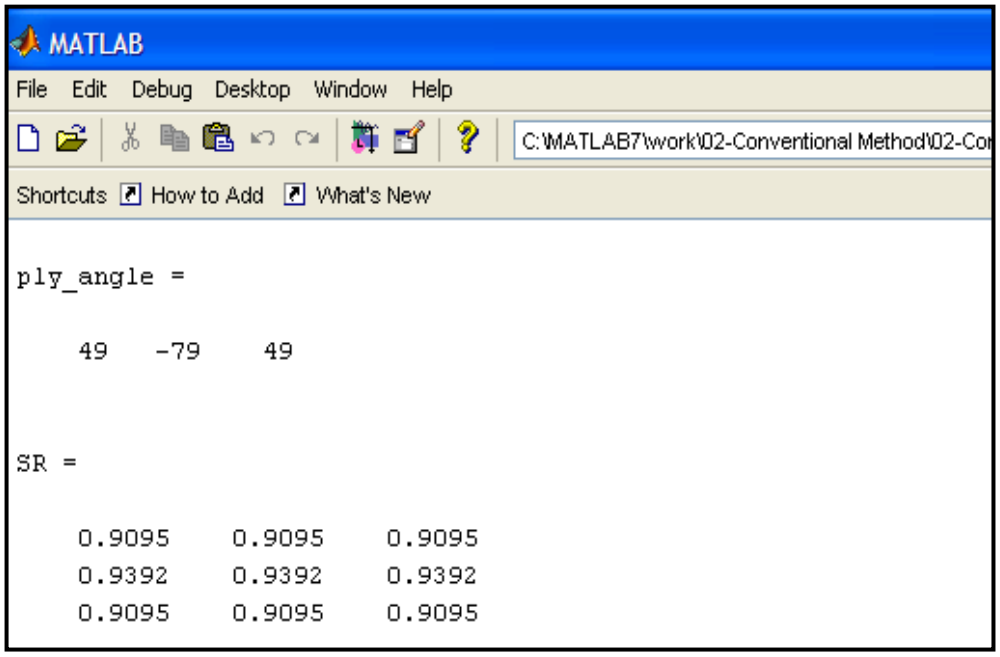


Figure 5.5: *SR* ratios for 49°, -79° and 49° laminate with 10% increase in loading

5.3 The Angle-Thick-Opt (ATO) code

The ATO code is essentially the FAO code with an add-on function for the thickness optimisation. This code performs the optimisation of the fibre orientation angles and periodically optimises the thickness of each of the fibre layers. The inputs include the material-loading parameters, and the outputs are the complete fibre layup parameters, that is, the number of fibre layers, fibre orientation angle of each layer, and the

thickness of each layer. The operation of the ATO code is best illustrated by the flowchart in Figure 5.6. This flowchart is identical to the one in Figure 5.1 with the exception of the added thickness optimisation function. The procedure describing the operation of the ATO code is as follows:

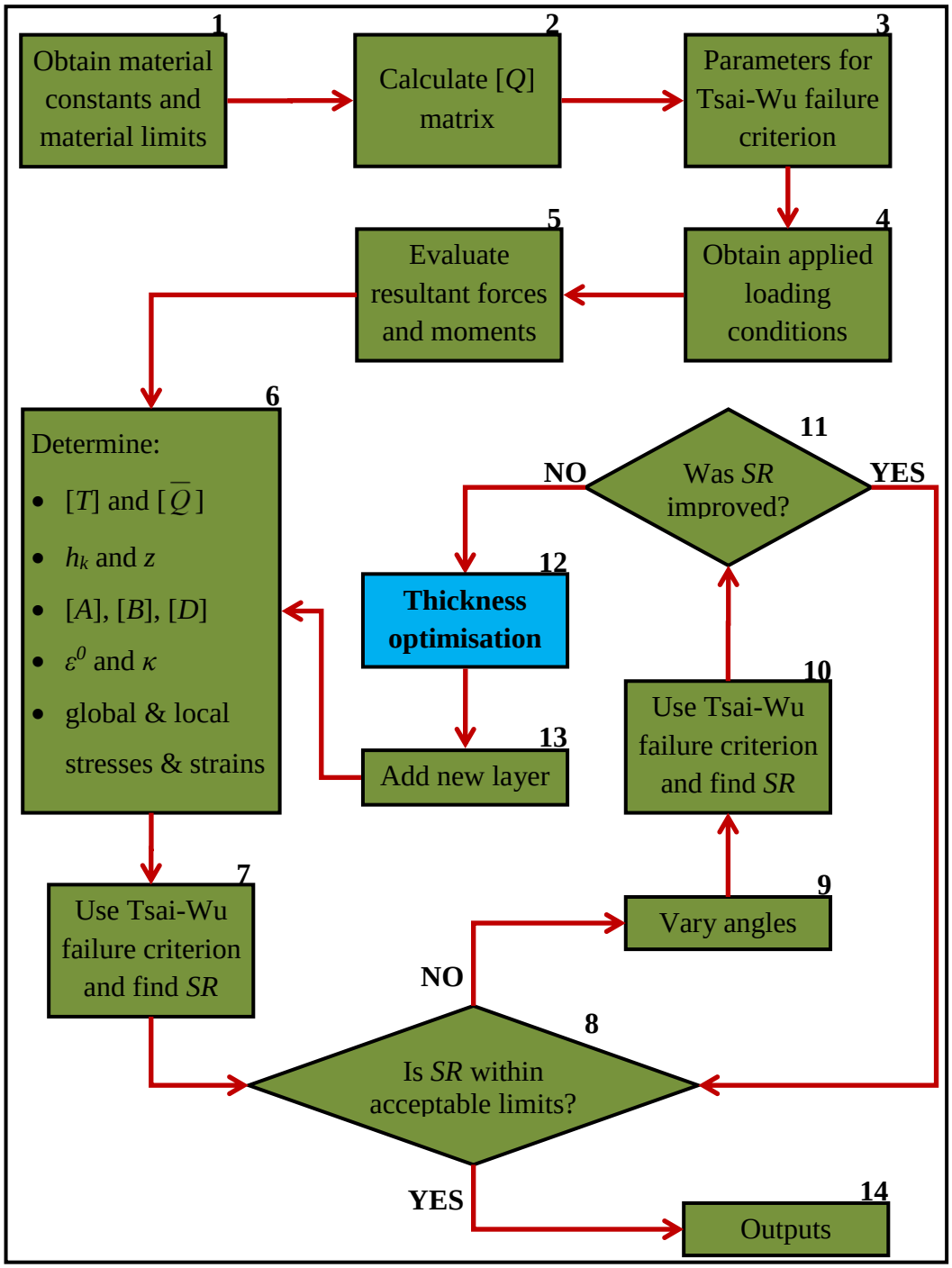


Figure 5.6: Flowchart describing the operation of the Angle-Thick-Opt (ATO) code

Block 1: The material properties and material limits are entered.

Block 2: The stiffness matrix [*Q*] is determined.

Block 3: Parameters for the Tsai-Wu failure criterion are calculated.

Block 4: Applied loading conditions are entered.

Block 5: The resultant forces and moments are determined.

Block 6: The relevant matrices and the various stresses and strains are computed.

Block 7: *SR* values for each layer are determined via the Tsai-Wu failure theory.

Block 8: The *SR* values are tested to determine whether they are within the specified range (between 1 and 1.2). If this condition is met, then the outputs are given (Block 14), otherwise the fibre orientation angles are varied (Block 9).

Block 9: Fibre orientation angles are varied to achieve *SR* values closer to the specified range.

Block 10: The matrix calculations are redone and new *SR* values are attained.

Block 11: Old and new *SR* values are compared to determine if there was an improvement. If this is the case, the new SR values are tested to determine whether they are within the specified range (Block 8). Otherwise the thickness optimisation function is executed (Block 12).

Block 12: The thickness optimisation function is executed and, upon completion, the angle optimisation is repeated, if required.

Block 13: A new layer is added. The relevant calculations are preformed (Block 6).

Block 14: The fibre layup parameters are outputted.

It is important to recall that with the FAO code, each new layer that was added to the laminate was initially assumed to have a fibre orientation angle of 0° with the specified fixed thickness. A similar assumption is used for the ATO code. Each layer that is added on has a fibre orientation angle of 0°, with a thickness that is equal to the thickness of one fibre layer of the composite material used in the design. This one-layer thickness value forms part of the material-loading parameters.

In the above procedure, Block 12 executes the thickness optimisation function. This function tests the current fibre layup parameters against certain criteria. Thickness

optimisation is performed, if necessary, and then Block 13 in the above procedure is processed. Figure 5.7 contains a flowchart describing the operation of the thickness optimisation function, where the dashed rectangular region represents Block 12 in Figure 5.6. The inputs of this block are the fibre layup parameters from Block 11.

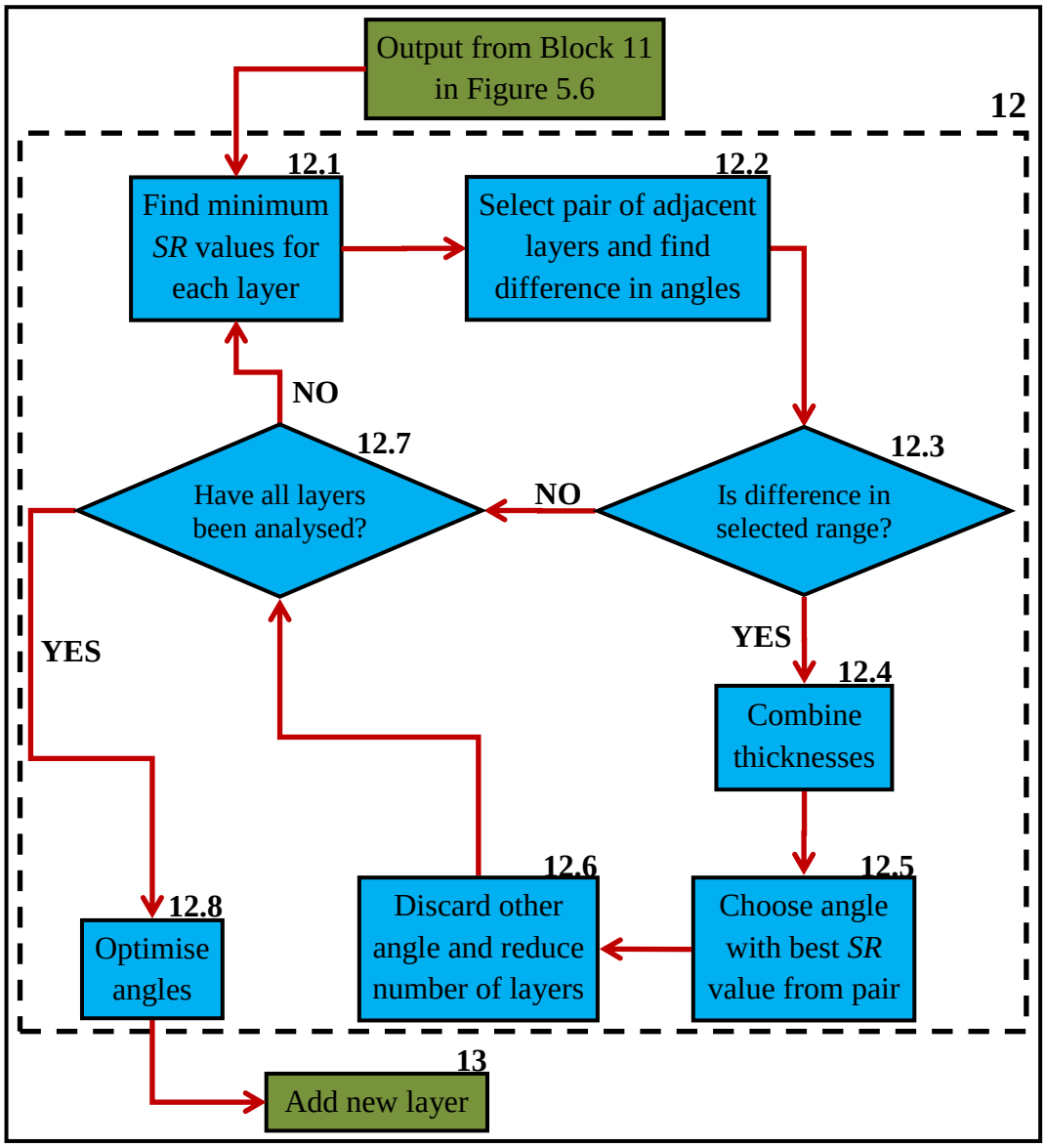


Figure 5.7: Flowchart describing the operation of the thickness optimisation function

The first step of the procedure in Figure 5.7 is to determine the minimum *SR* value of each layer (Block 12.1) and store it for later use. Next (Block 12.2), a pair of adjacent layers is chosen and the difference of their angles is calculated. In Block 12.3 it is

determined whether the absolute value of this difference is within a specified range. For this study, a range of 0° to 10° was used, however this may be amended according to the designer's specifications. If the difference is within the specified range, then the thicknesses of the layers under examination are combined/added (Block 12.4). The *SR* values stored earlier are used to determine which layer has the better value (Block 12.5), and the fibre orientation angle of the better layer is retained. (It should be noted that the chosen fibre orientation angle may not be the optimum, however, the angle optimisation process is repeated later on.) In Block 12.6, the other angle along with the layer it represents is discarded, and, as a result the number of layers decreases.

In order to better explain the above, consider a pair of adjacent layers where the first layer had a fibre orientation angle of 52° and a *SR* value of 1.09, and the second layer had a fibre orientation angle of 59° and a *SR* value of 1.12. In Block 12.2, the difference of the angles (7°) would be calculated, and, since this is in the range specified in Block 12.3, the thicknesses of the two layers would be added together (Block 12.4). The *SR* values for both layers would then be examined by Block 12.5, and the layer with the better *SR* value would be retained. In this case, the 59° layer would be retained since its *SR* value was higher than that of the 52° layer. The 52° layer would be removed from the laminate in Block 12.6, and hence decrease the number of fibre layers.

The next step in the flowchart (Block 12.7) is to verify whether all the layer pairs have been examined. If there are any remaining pairs, the procedure in Blocks 12.1 to 12.7 are repeated until all the layer pairs have been analysed. When this is completed, Block 12.8 is executed and the angle optimisation is redone with the new layer thicknesses. Once the thickness optimisation function has accomplished its task, a new fibre layer is added on (Block 13), if necessary. For every new layer that is added on, the thickness optimisation function is executed at the appropriate time. The line-by-line examination of the ATO code is shown in Appendix C.

The ATO code was tested using the previous graphite/epoxy laminate with the material-loading parameters given in Table 4.1. The one-layer thickness value was set to 1mm. The outputs from this design procedure are shown in Figure 5.8. Here four

fibre layers were needed with angles of -53°, 55°, 4° and -89°, and thicknesses of 1mm, 2mm, 2mm and 1mm, respectively. These results, and those obtained with the FAO code, were tabulated (Table 5.2) in order to provide a better perspective of their differences.

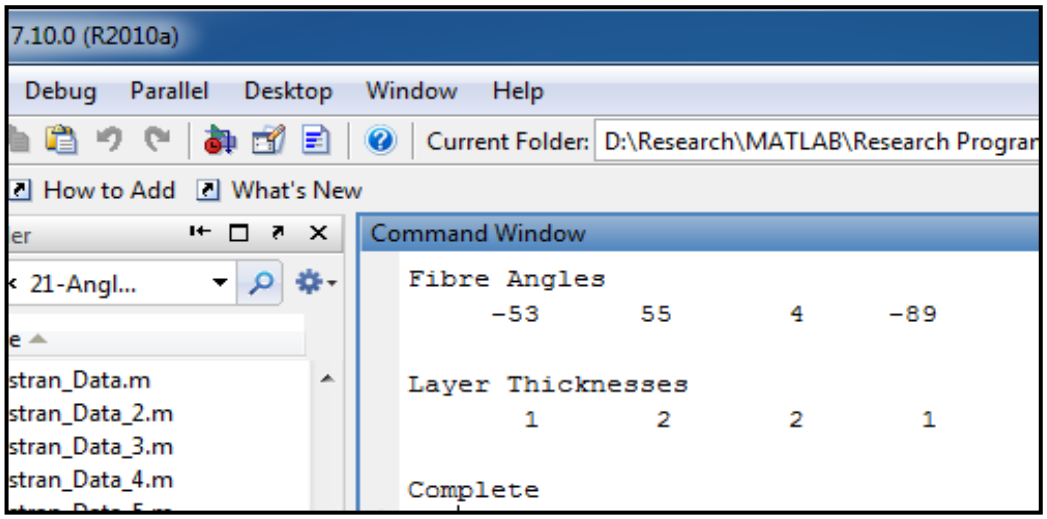


Figure 5.8: Results of Angle-Thick-Opt (ATO) code

Table 5.2: Comparison of outputs from FAO code, with fixed thickness, and ATO code, with variable thickness

	Case A: FAO code (fixed thickness)	**Case B: ATO code (variable thickness)**
Number of layers	3	4
Fibre orientation angles (°)	49 -79 49	-53 55 4 -89
Layer thickness (mm)	5 5 5	1 2 2 1
Total thickness (mm)	15	6

In Table 5.2 it may be seen that for the same loading conditions, Case B (ATO code) required more fibre layers than Case A (FAO code). However, in Case B the total laminate thickness was reduced to 6mm, which was less than half the thickness required for Case A. This implied that less material would be used in Case B, which translates to a saving in material costs and consequently manufacturing costs. It may be argued that, due to more layers being required in Case B, more fibre orientations would have to be performed during the fibre placement process, and this may lead to

an increase in labour costs. However, this would not be the case as fibre placement would be accomplished autonomously via the Robotic Fibre Placement (RFP) method.

In addition to the thickness optimisation, there was also a reduction in the laminate mass. The volumes of the laminates in Cases A and B were calculated using the overall thicknesses, shown in Table 5.2, and assuming each laminate to be 300 X 300mm. These values are shown in Table 5.3 and, together with the density of graphite/epoxy (1620kg/m^3), were used to determine the mass of the laminate in each case. The table shows that the mass for the laminate in Case B decreased by 1.3 kg or 60% when compared to Case A. This implies that the ATO code is capable of weight optimisation as well.

Table 5.3: Comparison of laminates masses for Cases A and B

	Dimension (mm)	**Volume (m^3)**	**Mass (kg)**
Case A	300 X 300 X 15	0.00135	2.187
Case B	300 X 300 X 6	0.00054	0.875

The results from Case B were validated using the Patran finite element environment. The model employed for the analysis was identical to the one used above for the validation of the FAO code with the exception of the fibre layup parameters. Values for the number of layers, fibre orientation angles and layer thicknesses were obtained from Figure 5.8, and the material properties and loading conditions were acquired from Table 4.1.

The fringe plots of the stress components from the finite element analysis are shown in Figure 5.9. The local stresses experienced by the laminate that were calculated in Matlab are shown in Figure 5.10. The stress values from Figure 5.9 were compared to the stresses incurred at the middle of each layer in Figure 5.10, and it was found that they were exactly equal. This validated that the ATO code was able to optimise the fibre orientation angles and layer thicknesses, and consequently the number of fibre layers and weight, in a fibre reinforced composite laminate.

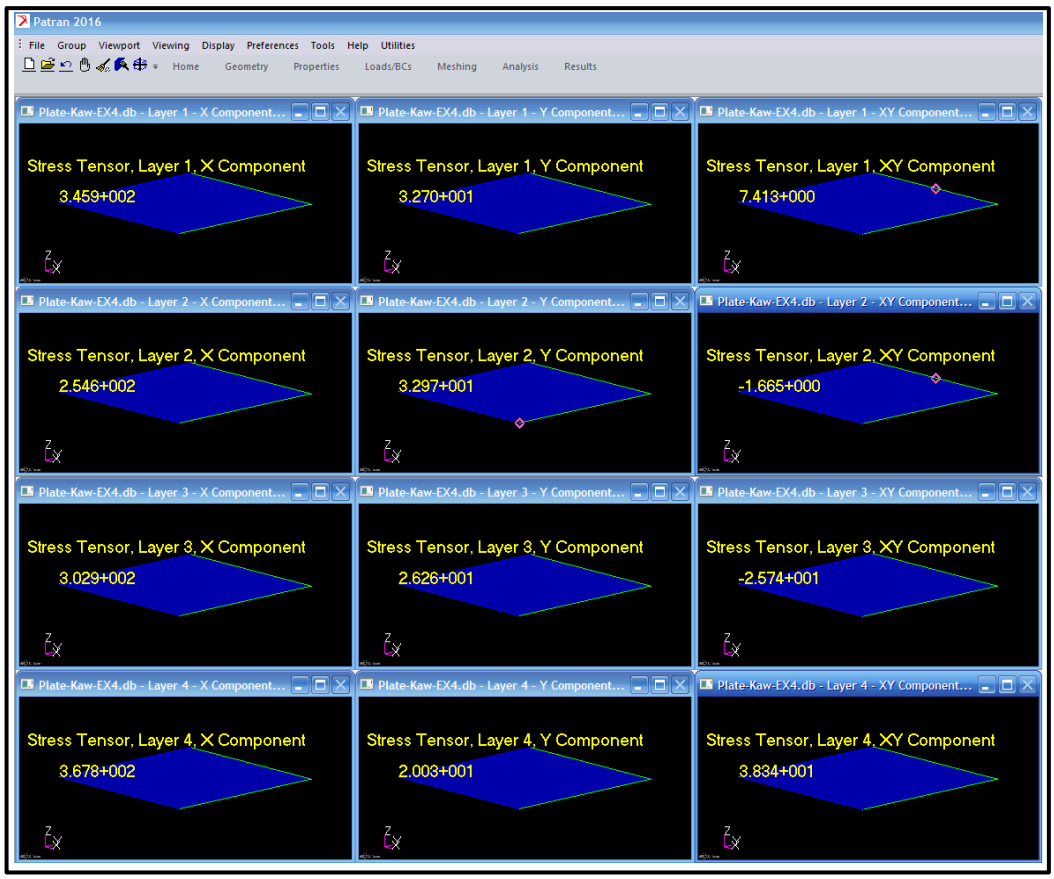
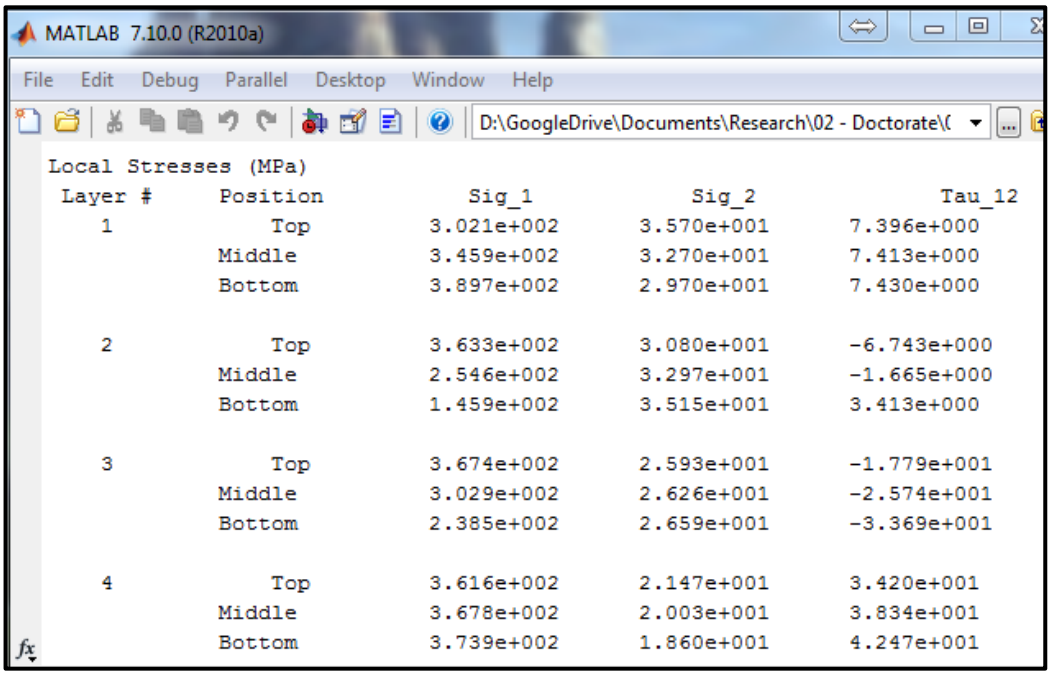


Figure 5.9: Patran fringe plots showing stress components for the laminate in Case B



Local Stresses (MPa)

Layer #	Position	Sig_1	Sig_2	Tau_12
1	Top	3.021e+002	3.570e+001	7.396e+000
	Middle	3.459e+002	3.270e+001	7.413e+000
	Bottom	3.897e+002	2.970e+001	7.430e+000
2	Top	3.633e+002	3.080e+001	-6.743e+000
	Middle	2.546e+002	3.297e+001	-1.665e+000
	Bottom	1.459e+002	3.515e+001	3.413e+000
3	Top	3.674e+002	2.593e+001	-1.779e+001
	Middle	3.029e+002	2.626e+001	-2.574e+001
	Bottom	2.385e+002	2.659e+001	-3.369e+001
4	Top	3.616e+002	2.147e+001	3.420e+001
	Middle	3.678e+002	2.003e+001	3.834e+001
	Bottom	3.739e+002	1.860e+001	4.247e+001

Figure 5.10: Local stresses of laminate in Case B when analysed in Matlab

Up to this point, the developed codes were tested with graphite/epoxy composite laminates. It was decided to determine whether a different composite material would yield similar results. The material chosen was glass/epoxy as it is commonly used for many applications. The material-loading parameters for this material are shown in Table 5.4.

Table 5.4: Material-loading parameters for the glass/epoxy laminate

Material Properties				Material Limits					Loading	
E_1 (GPa)	E_2 (GPa)	G_{12} (GPa)	ν_{12}	$(\sigma_1^T)_{ult}$ (MPa)	$(\sigma_1^C)_{ult}$ (MPa)	$(\sigma_2^T)_{ult}$ (MPa)	$(\sigma_2^C)_{ult}$ (MPa)	$(\tau_{12})_{ult}$ (MPa)	Forces (N)	Moments (Nm)
38.6	8.27	4.14	0.26	1062	610	31	118	72	1000	0
									1000	0
									0	0

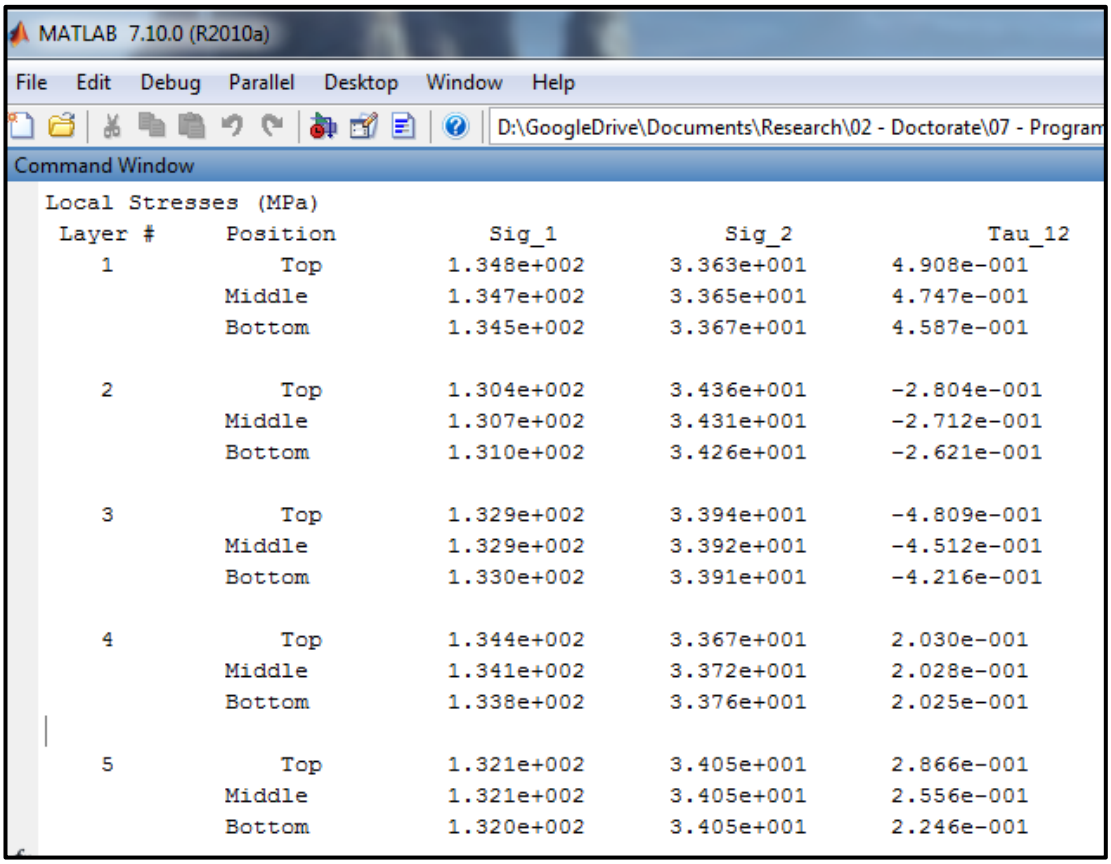
Both the FAO and ATO codes were used to design the glass/epoxy laminate with the material-loading parameters from Table 5.4. The results from each of these codes are shown in Table 5.5. From the table it may be seen that, with the FAO code, six glass fibre layers were required, with each layer having a fixed thickness of 5mm, and hence giving an overall laminate thickness of 30mm. If the laminate was assumed to be 300 X 300mm, and using the density for glass/epoxy (1790 kg/m^3), the calculated mass would be 4.833kg. In the case of the ATO code, there were ten layers required with a total laminate thickness of 12mm. The mass, for the same laminate size as the FAO design, was calculated to be 1.933kg, which was a 60% reduction over the FAO design. These results are similar to the graphite/epoxy laminate and serves to further emphasise the ability of the ATO code to optimise the fibre orientation angles and layer thicknesses, and consequently the number of layers and weight, of fibre reinforced laminates of various composite materials.

The fibre orientation angles and layer thicknesses from Table 5.5 for the ATO code, along with the material-loading parameters from Table 5.4, were used in a finite element model, similar to the one employed for the graphite/epoxy laminate, to determine the stresses incurred. The local stresses obtained from Matlab (Figure 5.11) were compared to the fringe plots of the stress components from the finite element

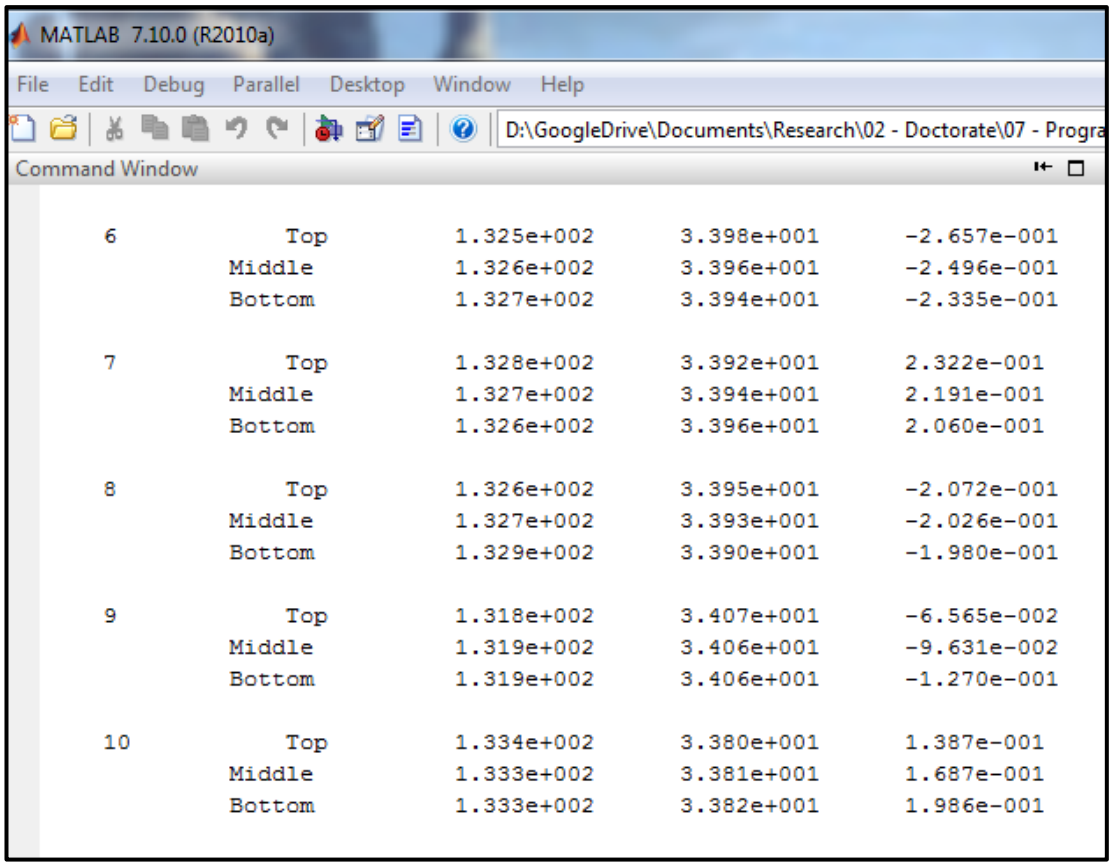
analysis (Figure 5.12). It was found that the stresses at the middle of each layer in Figure 5.11 exactly matched the values in the fringe plots in Figure 5.12. This serves to further reinforce the validation of the ATO code performed earlier.

Table 5.5: Results from FAO and ATO codes for glass/epoxy laminate

	FAO code	**ATO code**
Number of layers	6	10
Fibre orientation angles (°)	38 43 16 17 45 36	69 -10 -40 84 46 -21 72 -10 31 -61
Layer thickness (mm)	5 5 5 5 5 5	1 2 1 2 1 1 1 1 1 1
Total thickness (mm)	30	12
Mass (kg)	4.833	1.933



(a)



6	Top	1.325e+002	3.398e+001	-2.657e-001
	Middle	1.326e+002	3.396e+001	-2.496e-001
	Bottom	1.327e+002	3.394e+001	-2.335e-001
7	Top	1.328e+002	3.392e+001	2.322e-001
	Middle	1.327e+002	3.394e+001	2.191e-001
	Bottom	1.326e+002	3.396e+001	2.060e-001
8	Top	1.326e+002	3.395e+001	-2.072e-001
	Middle	1.327e+002	3.393e+001	-2.026e-001
	Bottom	1.329e+002	3.390e+001	-1.980e-001
9	Top	1.318e+002	3.407e+001	-6.565e-002
	Middle	1.319e+002	3.406e+001	-9.631e-002
	Bottom	1.319e+002	3.406e+001	-1.270e-001
10	Top	1.334e+002	3.380e+001	1.387e-001
	Middle	1.333e+002	3.381e+001	1.687e-001
	Bottom	1.333e+002	3.382e+001	1.986e-001

(b)

Figure 5.11: Local stress values from Matlab of the glass/epoxy laminate for layers (a) 1 to 5, and (b) 6 to 10

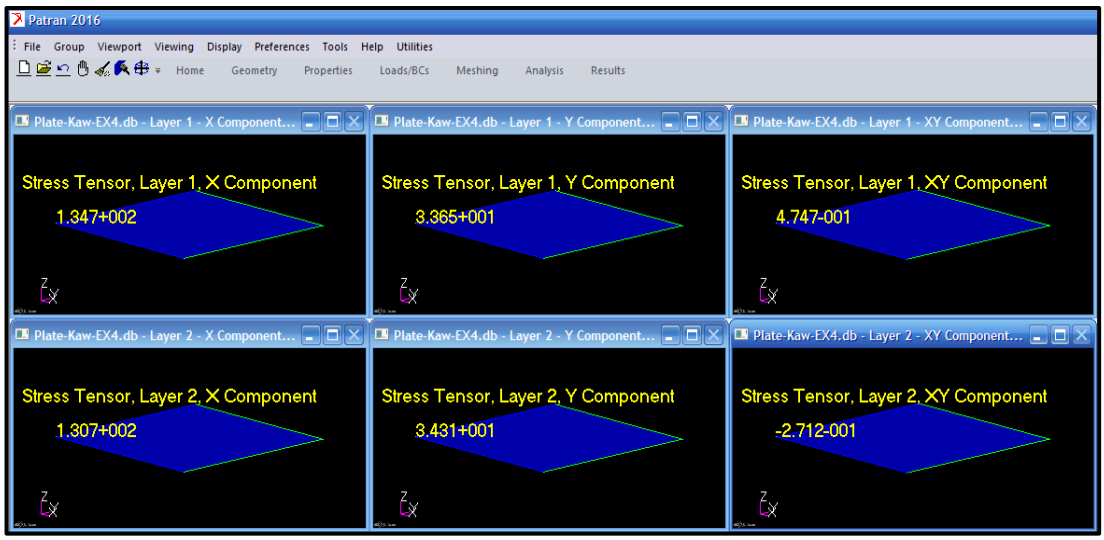


Figure 5.12: Fringe plots showing stress components of the glass/epoxy laminate

The ATO code was able to optimise the fibre layup parameters as well as the weight of a composite laminate. However, the design still addressed constant stiffness structures. Further, the code was based on Hooke's Law and therefore only valid for two-dimensional entities. The main objective of this research was to develop a process that is able to design three-dimensional, variable stiffness structures, and therefore the ATO code was expanded to allow for this. A new code was developed and called Variable Stiffness Method (VSM).

5.4 The Variable Stiffness Method (VSM) code

As mentioned in Chapter 1, a variable stiffness structure is one where the stiffness varies throughout the structure. As stiffness is directly related to the fibre layup parameters, the simplest method of creating a variable stiffness structure would be to vary the fibre orientation angles across a particular layer, or have additional fibre layers in certain regions of the structure. Variable stiffness structures are therefore multi-strength (varying stiffness) and multi-directional (varying fibre orientation angles) entities.

The Variable Stiffness Method (VSM) code was developed in this study in order to design variable stiffness structures with varying fibre orientation angles as well as additional fibre layers in specific regions of the structure. Initially, a finite element model of the structure, with its material-loading parameters, needs to be set up and analysed in Patran. This is necessary in order to determine the force distribution through the structure. Matlab codes would then be employed to establish the fibre layup parameters throughout the structure based on the force distribution. The operation of the VSM code is detailed below, however, it is prudent to first understand the basic concepts of finite elements.

A finite element package is able to divide any structure into smaller sections via meshing, as illustrated in Figure 5.13. A mesh comprises of mesh lines which form shapes such as quadrilaterals, triangles, hexagons, etc. These shapes are referred to as elements, and the intersections of the mesh lines are called nodes. The VSM code utilises these concepts in its operation.

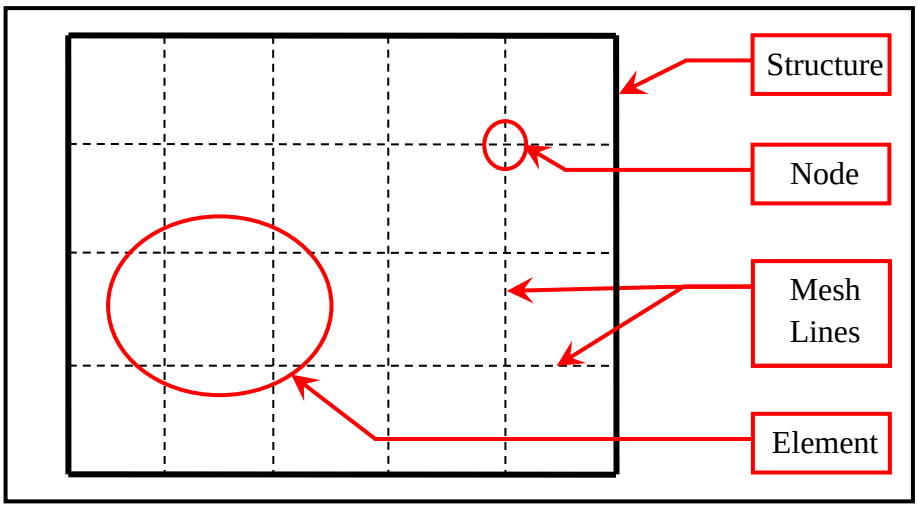


Figure 5.13: Illustrated concepts of meshing, mesh lines, elements and nodes

Consider a structure that had been meshed in a finite element package. Each element that was created may be analysed as a separate entity (or laminate), provided that the loading conditions for that element were known. In finite element packages such as Patran, loading applied to the structure is distributed through the nodes, and therefore the loading conditions for each element may be easily determined. Hence, if each element was treated as a separate laminate, the ATO code may then be used to design the fibre layup for each element/laminate. To better understand this concept, consider an open fibre reinforced box. It consists of five sides and each side may be considered as a two-dimensional entity, each with its corresponding loading conditions. Therefore, each side may be designed as a separate fibre reinforced laminate, and then combined to form the three-dimensional composite structure. The design approach used for variable stiffness structures was based on this concept and the procedure that was followed is illustrated in Figure 5.14.

The first step in the procedure (Block A) requires the construction of the part, or structure being designed, as a shell. This may be accomplished via a Computer Aided Design (CAD) package and then imported into the finite element package, or directly through the finite element package itself. The next step (Block B) divides the surfaces of the structure into elements by means of meshing. The recommended mesh shape is quadrilaterals and the mesh size should be varied. In areas with little or no change in

shape, the mesh size should be coarse, while in regions with abrupt changes in section, such as bends, corners, holes, etc., the mesh size should be made finer.

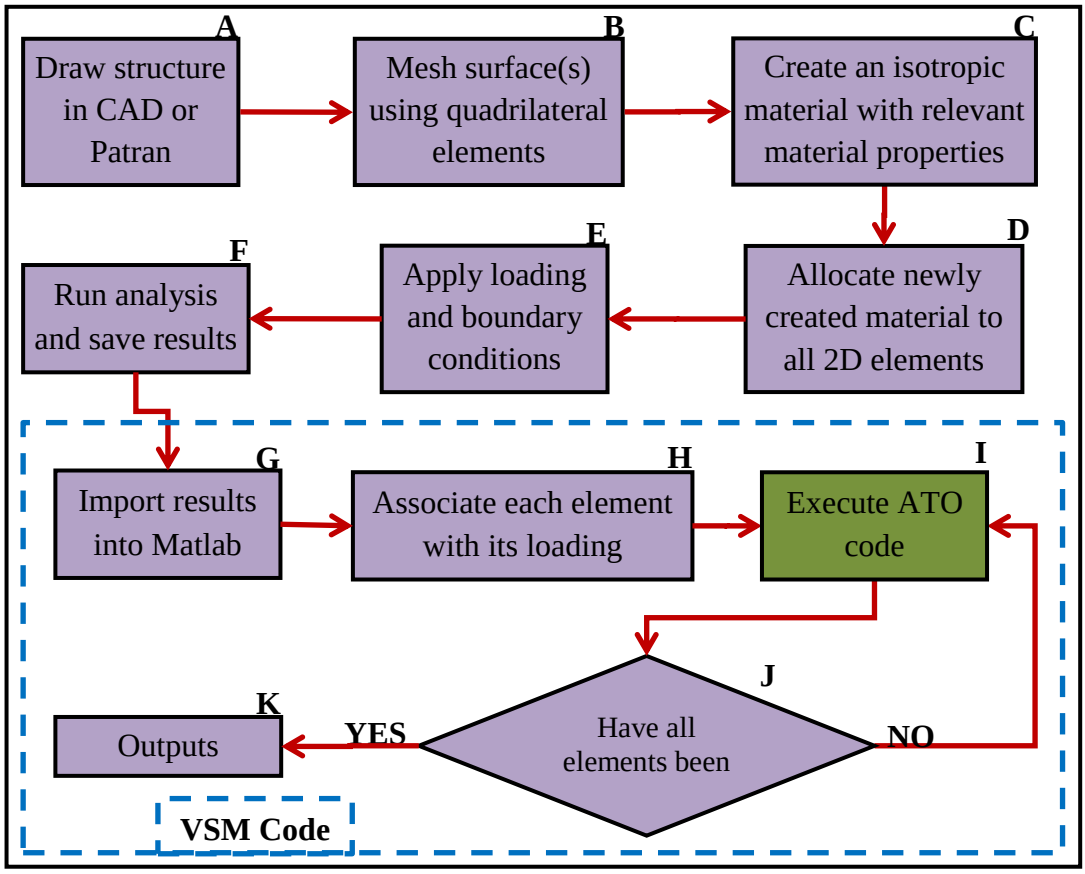


Figure 5.14: Flowchart illustrating the procedure followed in the design of a variable stiffness composite structure

In Block C, each element of the mesh is defined as an isotropic material, for example steel, with the appropriate material properties. The key here is to determine the load distribution through the structure. If a laminated composite was used instead of an isotropic material, the load distribution would be affected by the fibre orientation angle or the orthotropic nature of the composite. Since an isotropic material has the same properties in all directions, it would not influence the load distribution. The next step (Block D) is to assign the isotropic material to all the mesh elements. The boundary and loading conditions are then applied to the structure (Block E). The analysis of the finite element model is carried out by Block F, and the results, which includes the load distribution through the structure, are saved in binary format as an OP2 file.

The next steps in the procedure, Blocks G to K in Figure 5.14 (dashed rectangular area), constitutes the VSM code. In Block G, the results (OP2 file) are imported into Matlab using a software package called IMAT, which was obtained from ATA Engineering in San Diego, California, USA. This package is used to convert the OP2 file from binary into a format that is recognised by Matlab. The imported results include the number of elements, element nodes (nodes contained in each element), nodal forces and moments, geometry of the structure, and other entities. In Block H, the element nodes, and, the nodal forces and moments are used to associate each element with its loading conditions. Each element may now be considered as a separate laminate, each with its own material-loading parameters. The ATO code is then executed to determine the fibre layup parameters of each element (Blocks I and J). Once all the elements have been analysed, the outputs are given by Block K, which include the fibre layup parameters as well as the strength ratios, global and local stresses and strains, and the loading vector for each and every element in the finite element mesh. The examination of the VSM code is performed in Appendix C.

The VSM code was tested with the glass/epoxy properties from the previous section. A finite element model was set up in Patran and is shown in Figure 5.15. The model consisted of a flat surface that was meshed into four quadrilateral elements. An isotropic material was created and assigned to each of these elements. The loading conditions shown in Table 5.4 were applied to the model. The analysis was executed and the results were saved as an OP2 file.

The finite element results were imported into Matlab using IMAT and the VSM code. Each element was associated with its loading conditions. The ATO code was executed for each element in order to determine its optimum fibre layup parameters. These results are shown in Figure 5.16. It may be seen from the figure that the fibre layup parameters were exactly the same for each element. Further, these parameters were exactly comparable to those obtained for the ATO case above, (Table 5.5). Therefore the local stresses experienced by each element would be the same and equal to those shown in Figure 5.11. This implied that the VSM code was able to design a three-

dimensional fibre reinforced composite structure by determining the fibre layup parameters for each of the two-dimensional elements in that structure.

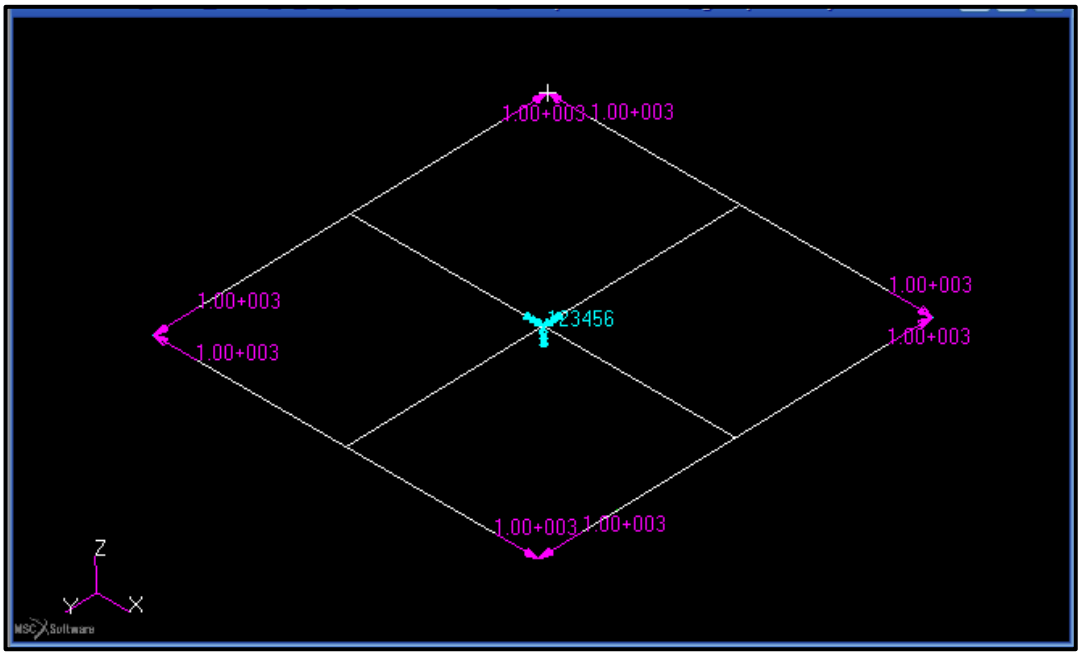


Figure 5.15: Finite element model showing elements, and applied boundary and loading conditions for testing of VSM code

Up to this point the developed codes were validated using examples from literature and finite element analyses. While these proved to be adequate, experimental validation would carry more weight and remove any doubt that the VSM code could be used to design three-dimensional, multi-directional, multi-strength structures. The next section discusses an experimental study of the VSM code.

5.5 Experimental study of the VSM code

There are three parts or phases to this study, which include design, fabrication and testing. The structure chosen for this experiment was a C-channel as it is quite straightforward to fabricate but also has the complexity of being a structure in more than one plane. The C-channel was fabricated from a unidirectional carbon fibre and epoxy composite. Table 5.6 shows the material properties and material limits for this composite. The length of the channel was 200mm and contained cross-sectional dimensions as shown in Figure 5.17.

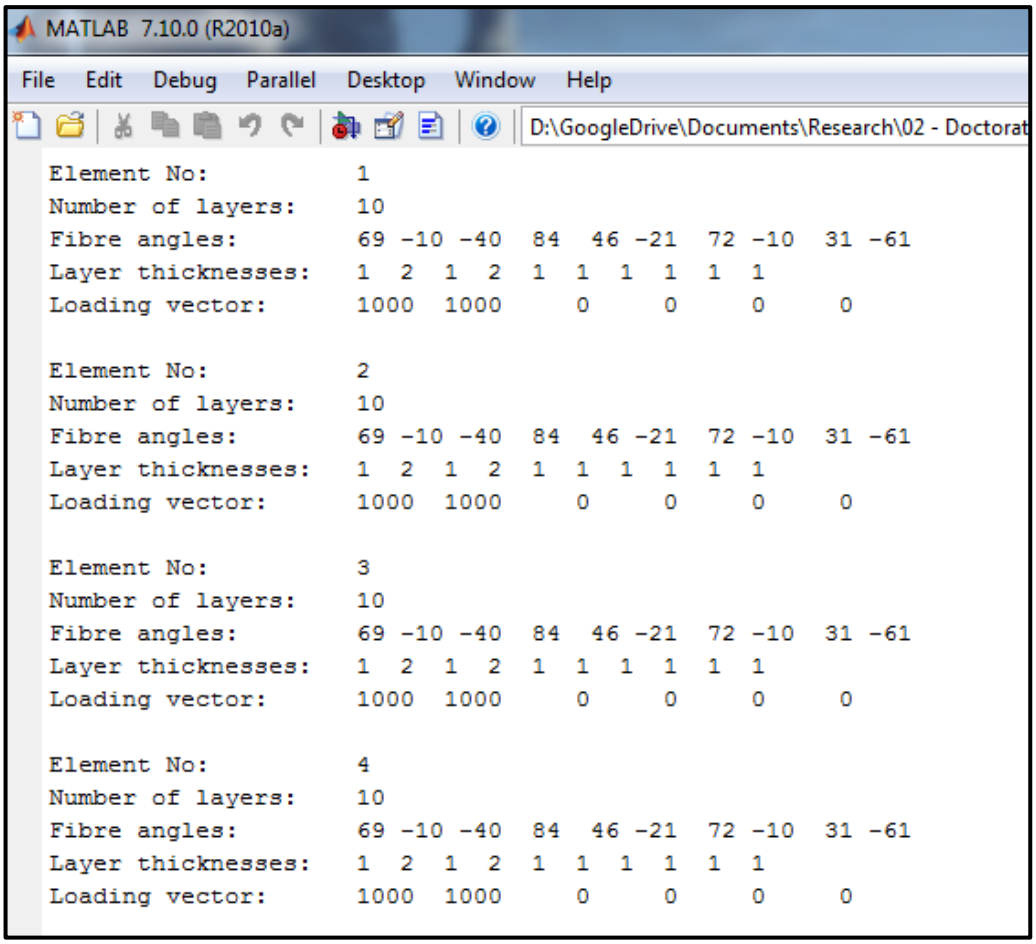


Figure 5.16: Matlab results of each element for the variable stiffness design of the glass/epoxy laminate

Table 5.6: Material properties and material limits for unidirectional carbon fibre/epoxy composite

Material Properties				Material Limits				
E_1 (GPa)	E_2 (GPa)	G_{12} (GPa)	ν_{12}	$(\sigma_1^T)_{ult}$ (MPa)	$(\sigma_1^C)_{ult}$ (MPa)	$(\sigma_2^T)_{ult}$ (MPa)	$(\sigma_2^C)_{ult}$ (MPa)	$(\tau_{12})_{ult}$ (MPa)
110	6.5	5	0.3	2000	790	29.5	117	83.9

For the design phase, the procedure laid out in Figure 5.14 was followed in order to determine the fibre layup parameters for the C-channel. The structure was modelled, with the length mentioned above and the cross-sectional dimensions shown in Figure 5.17, in the Patran environment. The surface was meshed using quadrilateral elements, and in total 40 elements were formed. An isotropic material was created and these

properties were assigned to all two-dimensional elements. Thereafter the loading and boundary conditions were applied, and the analysis was run. The C-channel was constrained at the centre and subjected to a tensile load of 5kN along the length of the channel (x direction). The finite element model with these applied conditions is shown in Figure 5.18.

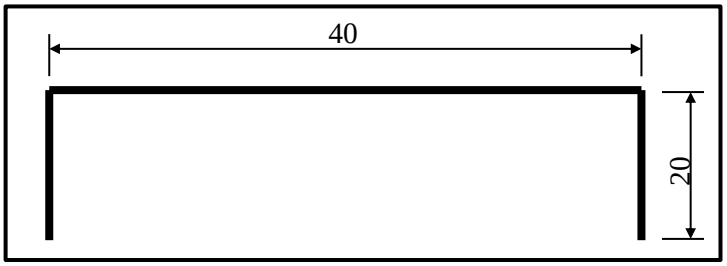


Figure 5.17: Cross-sectional dimensions of the fabricated C-channel

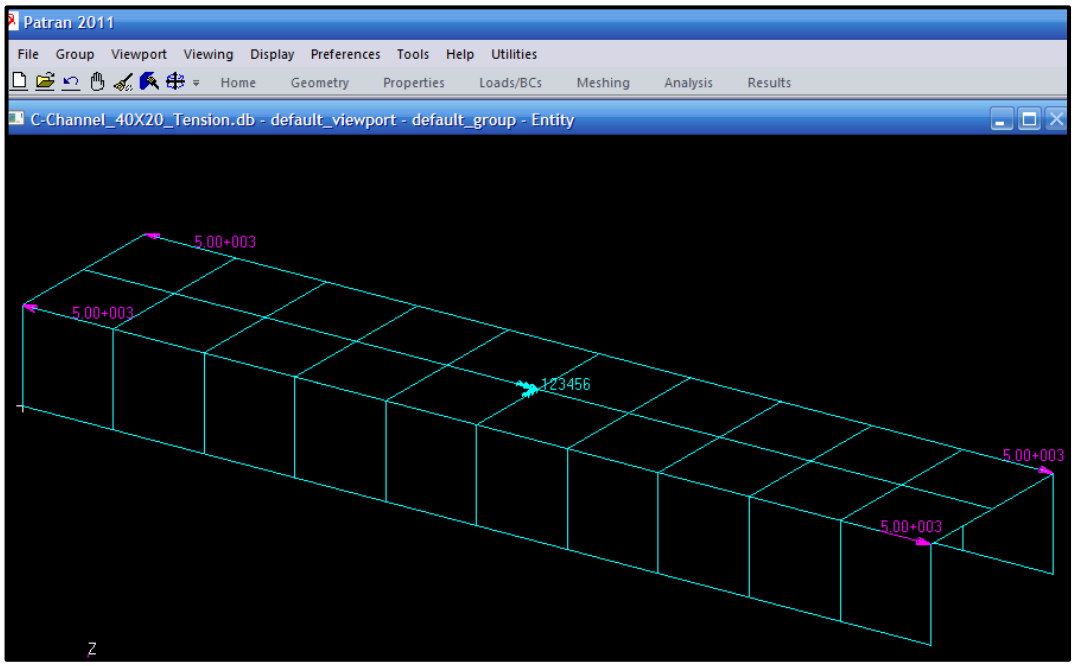


Figure 5.18: Finite element model of C-channel showing the applied loading and boundary conditions

The VSM code was then used to determine the fibre layup parameters for the three-dimensional structure. First, the results from Patran were imported into Matlab. Next, each element was associated with its loading conditions. The ATO code was used to

determine the fibre layup parameters for each element from the finite element model. The results from the VSM code, which is, the fibre layup parameters for each element, is too cumbersome to be shown here and is therefore shown in Appendix D. It may be seen from the results that all of the elements contain one layer with varying thicknesses. There are six different angles throughout the structure, namely 0°, -2°, 2°, -5°, 5°, and 90°. Figure 5.19 shows an illustrated topographical view of the C-channel with colour coding to represent the fibre layup parameters throughout the structure. The layup parameters for each colour code is given in Table 5.7.

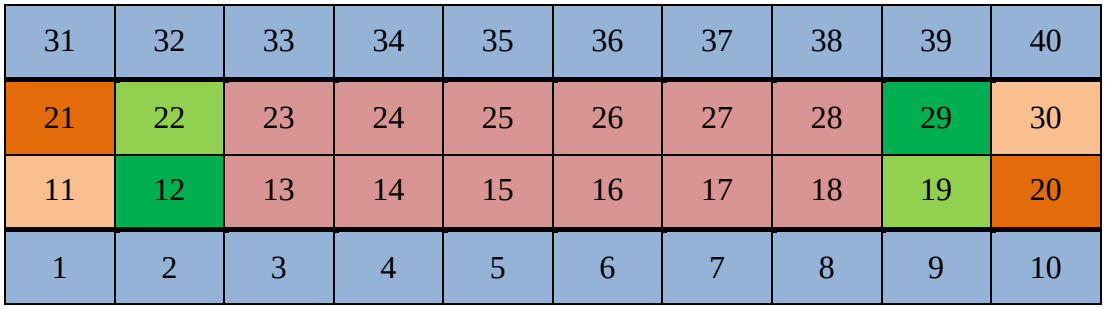


Figure 5.19: Colour coded topographical view of the fibre layup results from the VSM code for each element in the finite element model for the C-channel

Table 5.7: Fibre layup parameters for colour coding in Figure 5.19

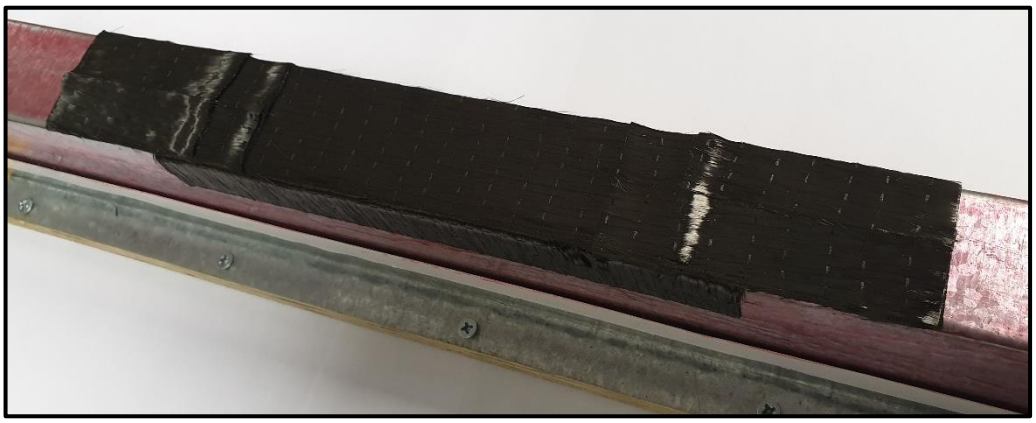
Colour Code	Element Numbers	Number of fibre layers	Fibre orientation angle of each layer (°)	Thickness of each layer (mm)
	1-10, 31-40	1	90	0.70
	11, 30	1	-5	3.85
	12, 29	1	-2	3.15
	13-18, 23-28	1	0	2.10
	19, 22	1	2	3.15
	20, 21	1	5	3.85

The next step in the experimental study was the fabrication of the C-channel according to the fibre layup parameters in Table 5.7 and the layout in Figure 5.19. Unidirectional carbon fibre was cut and laid on a mould as per the designed fibre orientation angles and layer thicknesses. Epoxy resin was then infused into the fibres via the Vacuum Assisted Resin Infusion Moulding (VARIM) method. Figure 5.20 shows the various

aspects of the fabrication process, namely, the mould, the fibres laid up on the mould, the C-channel prepared for vacuum infusion, and the final product. The tabs shown in Figure 5.20(d) were incorporated into the C-channel to provide for the gripping of the channel in the test fixture.

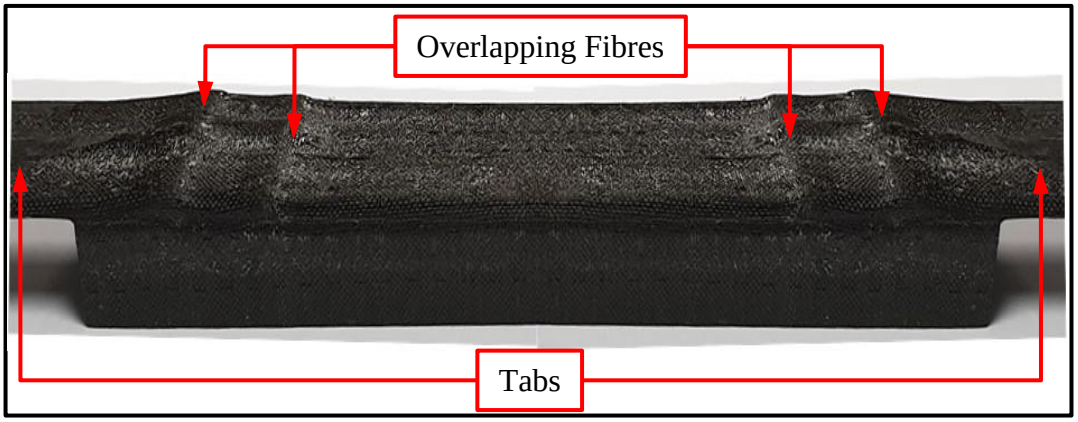
It is important to note that if the fibres were laid on the mould as per the layout shown in Figure 5.19, the result would be areas with discontinuous fibres. This is not desirable in a composite structure as it decreases the overall mechanical properties. However, it was reported by Lopes et al. [89] that the strength characteristics in a discontinuous fibre reinforced composite may be enhanced if the fibres were overlapped. Figure 5.20(d) shows areas with overlapping fibres between regions or elements with different fibre orientation angles.

(a)



(b)

(c)



(d)

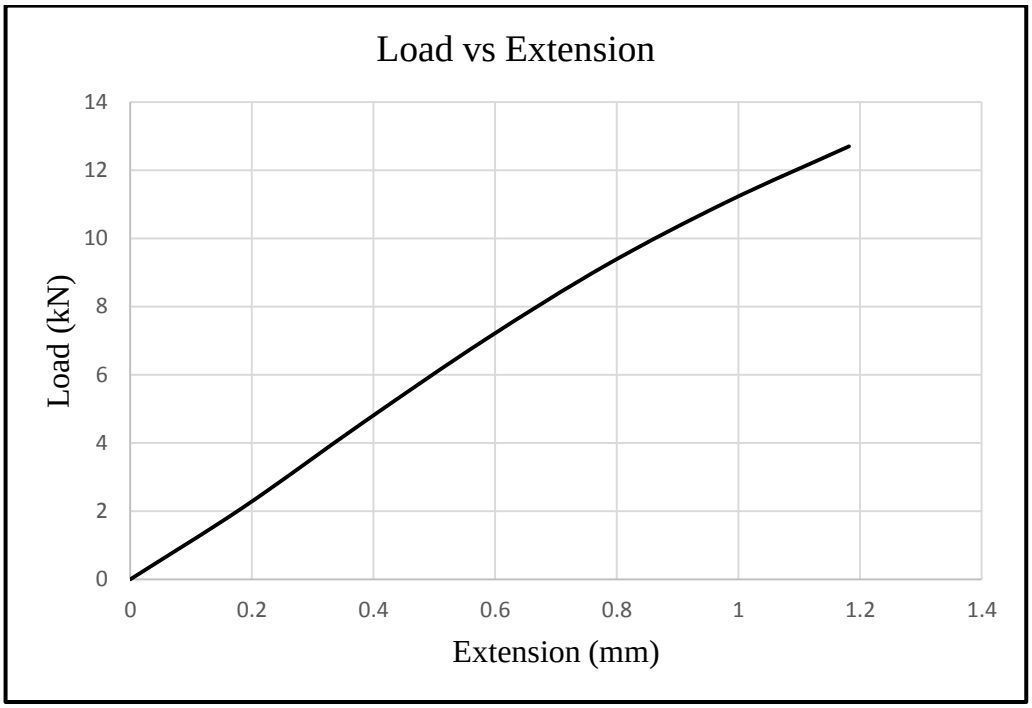
Figure 5.20: Various aspects in the fabrication of the C-channel, namely, (a) the mould, (b) the fibres laid up on the mould, (c) the C-channel prepared for vacuum infusion, and (d) the final product

The final phase in the experimental study was the testing of the fabricated channel. The test apparatus used was a MTS Criterion with a 2-column load frame configuration and a 30kN load cell. The channel was tested at a crosshead speed of 2 mm/min until failure occurred. Figure 5.21(a) shows the fractured channel after testing. The point of fracture was on the 0° layers, in particular at the location of elements 13 and 18 or 23 and 28. Failure at this point was expected as these elements obtained the lowest *SR* value (1.0493) as may be seen in Appendix D.

The failure load was 12.7kN which may be seen on the Load versus Extension graph shown in Figure 5.21(b). This value was 2.5 times greater than the design load (5kN). The reason for this may be attributed to the areas of overlapping fibres shown in Figure

5.20(d). The CSM code was used to determine the load carrying capability of the overlapping fibres in the region between the 0° and 2° fibre orientation angles. Figure 5.22 shows the results from the CSM code with the critical values highlighted. It may be seen that, at a load of 9.4kN, the minimum SR value is 1.0036. Therefore this region was capable of bearing a load of 9.4kN. In addition to this, the C-channel was modelled using a fibre volume fraction of 59%, however, the actual fibre volume fraction after fabrication was 63%. These factors contributed to the higher failure load.

(a)



(b)

Figure 5.21: (a) Fractured C-channel after testing, and (b) Load vs Extension graph of C-channel

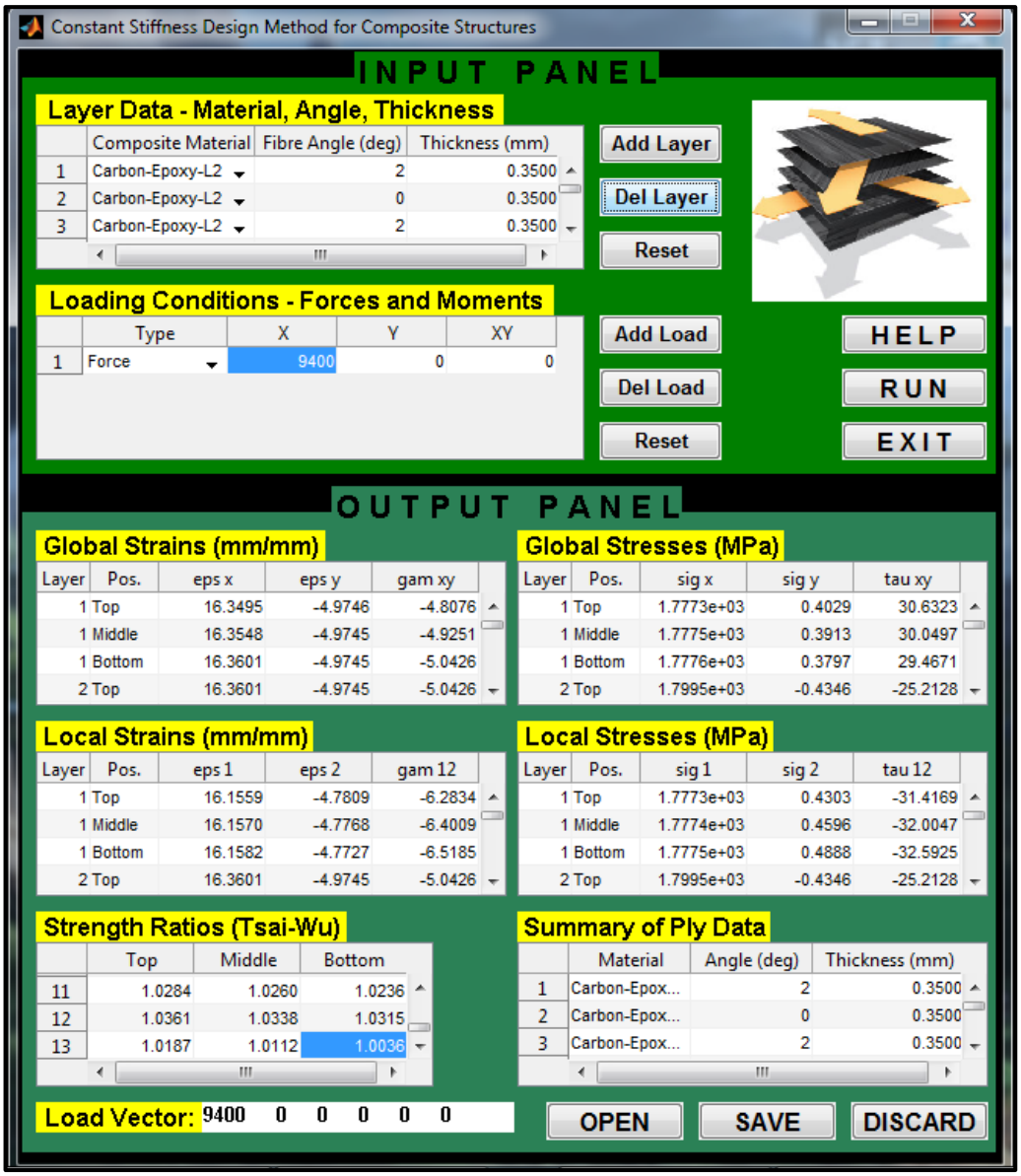


Figure 5.22: Results of the CSM code of the region between the 0° and 2° fibre orientation angles in the C-channel

5.6 Limitations of the developed codes

The codes developed thus far have limitations with regards to the conditions in which they may be used. Due to the codes being based on Hooke's Law, they will share the same limitations. In the formulation of these equations certain assumptions were made [132], namely:

1. Each layer is orthotropic, homogeneous and elastic.
2. There is no slip between the layer interfaces.

3. A line that is straight and perpendicular to the mid-plane remains straight and perpendicular to the mid-plane during deformation.
4. The length of a straight line in the z-direction remains constant.
5. The laminate follows plane stress conditions, that is, it is thin and is loaded only in its plane.
6. Displacements are small and continuous throughout the laminate.

It is significant to point out that any computational code that is developed based on the above equations would have the same assumptions as well. Further, according to Kaw [132], the above equations are valid not only for unidirectional composites, but also for composites with fibres that are woven perpendicular to each other, or contain short fibres that are aligned perpendicular to each other or in one direction.

Since the VSM design technique makes use of discontinuous fibres, it is important that fibres in adjacent regions of differing fibre orientation angles are overlapped [89].

5.7 Graphical User Interface (GUI) for the VSM code

A GUI was developed for the VSM code and is shown in Figure 5.23. It consists of three panels, namely, Input Panel, Elemental Results and Additional Results.

The input panel, shown in Figure 5.24, consists of a drop-down menu and four push buttons. As with the GUI for the CSM code, there is a help function available. This is shown in Figure 5.25 and is run via the pushbutton labelled H. The help function gives a brief description of the GUI and outlines the operational procedures that must be followed to use it correctly. The "EXIT" button, labelled X, closes the GUI.

The drop-down menu labelled A is used to select the composite material for the design process. The loading and boundary conditions, as well as the FEM data of the structure, is imported from the OP2 file via the pushbutton labelled B. The name of the file is displayed in the textbox labelled C. The "RUN" button, which is labelled as D, is used to verify that the inputs are valid, and then executes the VSM code. The outputs of the code are displayed in the Elemental Results panel.

Figure 5.23: Graphical User Interface for Variable Stiffness Method (VSM) code

The Elemental Results panel, shown in Figure 5.26, comprises of one table and four push buttons. The table labelled E is used to display the number of layers, fibre orientation angles, layer thicknesses and the force vector for each element. Previously

saved results are loaded via the "OPEN" button, labelled F, and the "SAVE" button, labelled G, is used to save the current results to a file. Additional results may be viewed by clicking on an entry in the table, and these are displayed on the Additional Results panel. The button labelled J is used to clear the data from the tables on the Additional Results panel, and the data from all tables and textboxes in the GUI are cleared via the button labelled K.

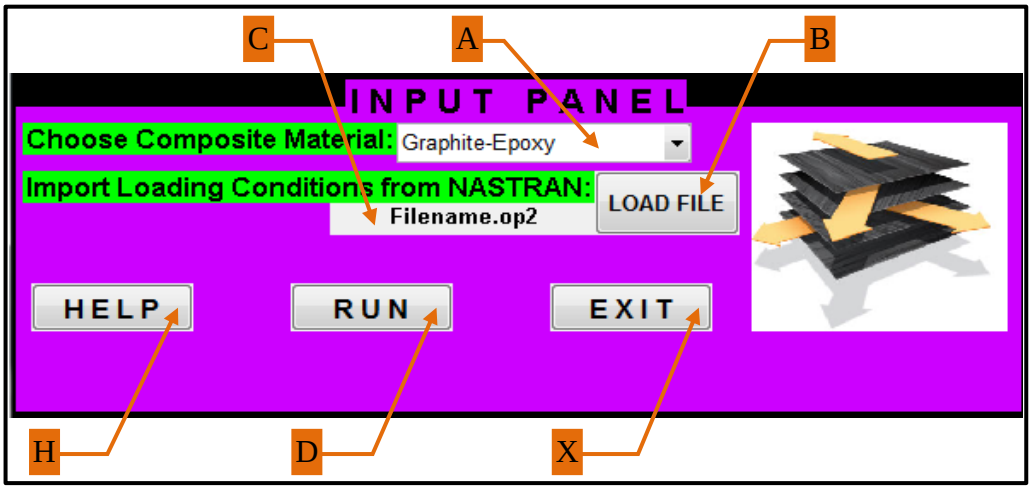


Figure 5.24: Input Panel for the VSM code's Graphical User Interface

In the Additional Results panel, which is shown in Figure 5.27, the space labelled by L displays the element under consideration. The global strains, global stresses, local strains and local stresses, for a particular element, are displayed in the tables labelled M, N, P and Q, respectively. Each table has the appropriate SI unit shown above it. The strength ratios for the element are displayed using the table labelled R. A summary of the materials, fibre orientation angles and thicknesses for this particular element are shown in the table labelled S.

The Matlab code for the above GUI is examined line-by-line in Appendix C.

5.8 Summary

The primary research objective was addressed in this chapter. Three computational codes were developed in order to optimise the fibre layup parameters of a fibre reinforced composite. The inputs of the codes were the material-loading parameters,

and the outputs included the number of fibre layers, fibre orientation angle of each layer and thickness of each layer. The operation of each code was discussed in detail with the help of flowcharts.

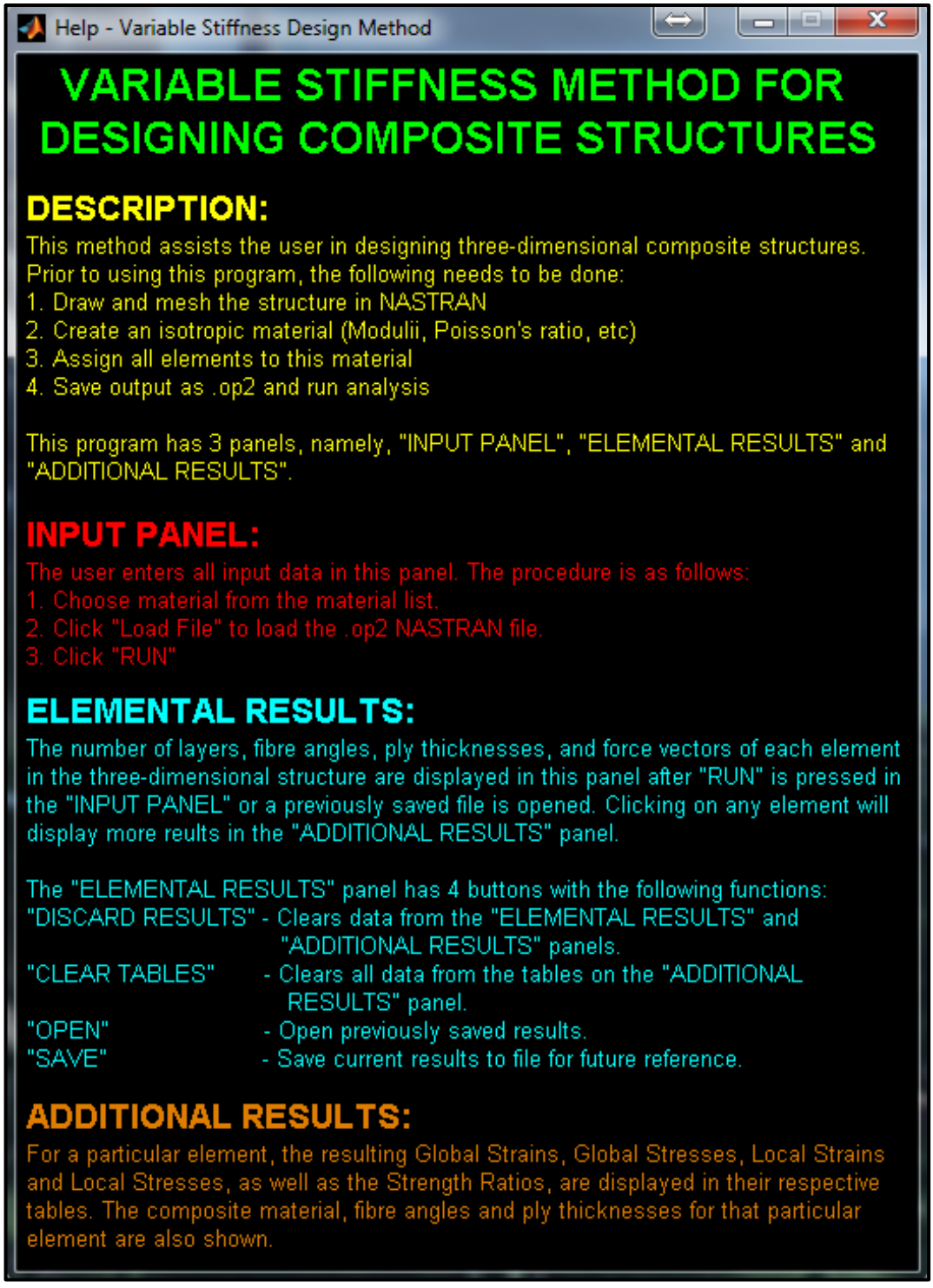


Figure 5.25: Help function for the VSM code

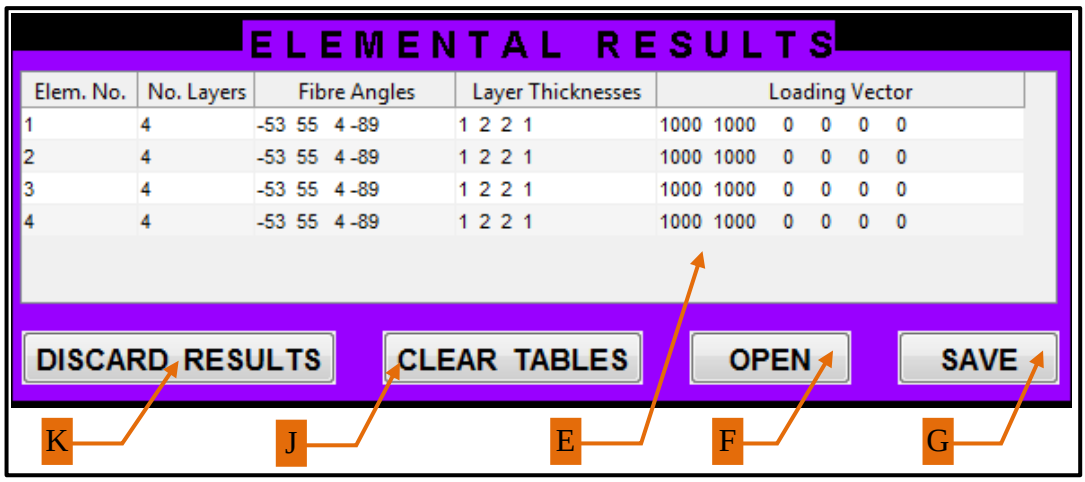


Figure 5.26: Elemental Results Panel for the VSM code's Graphical User Interface

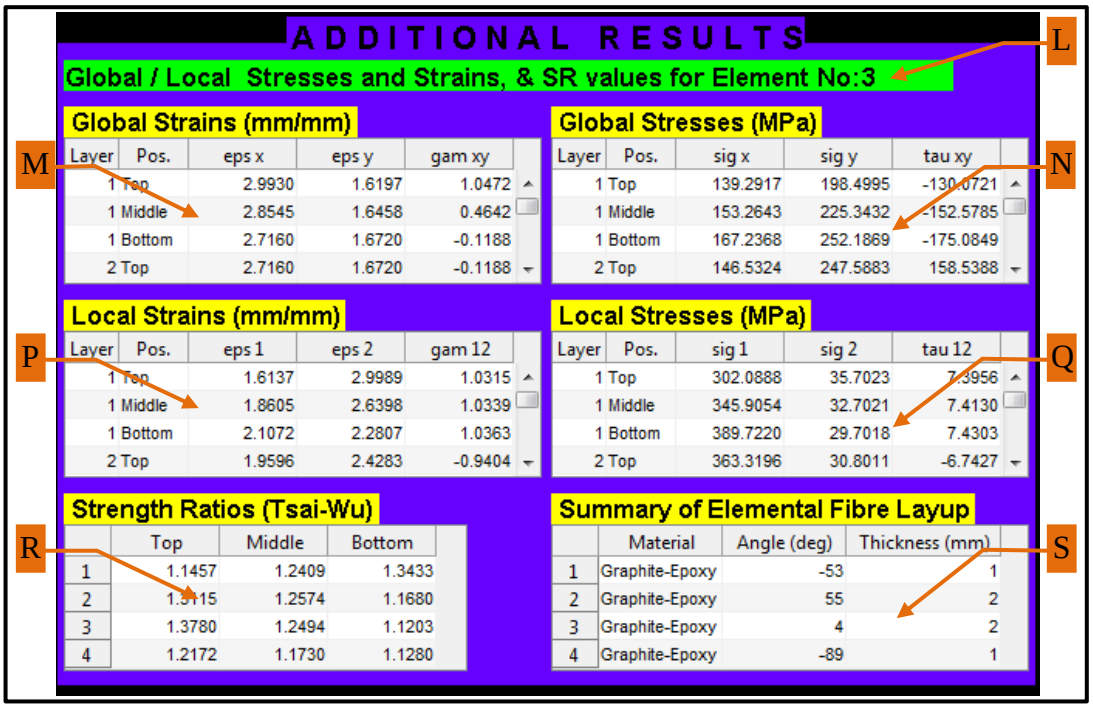


Figure 5.27: Additional Results Panel for the VSM code's Graphical User Interface

The first code was called Fibre-Angle-Opt (FAO) and was based on the Hooke's Law equations in Chapter 3. This code was used to optimise only the fibre orientation angles while keeping the layer thicknesses fixed. The developed code was validated using the CSM code as well as the Patran finite element environment.

The Angle-Thick-Opt (ATO) code was the next code that was created. This code combined the FAO code with a thickness optimisation function. The code optimised both the fibre orientation angle and layer thickness, and, consequently, the number of fibre layers and the weight of the composite laminate. Patran was used to validate the code.

The last code developed for the optimisation of the fibre layup parameters was called Variable Stiffness Method (VSM). In order to use this code, the structure being designed had to be divided into finite elements via Patran. Each element was then considered as a separate laminate with its own loading conditions. The VSM code was used to associate each element with its loading conditions, and then evaluate the fibre layup parameters using the ATO code. The VSM code was validated using finite element analysis and experimental results.

A Graphical User Interface was developed to aid the user in entering the design parameters and to view the outputs in an appropriate format.

The next research objective was to develop an interface between the design and fabrication phases. This is discussed in the next chapter.

6. TRANSITION FROM DESIGN PHASE TO FABRICATION PROCESS

6.1 Overview

The second research objective in Chapter 3, which was the development of an interface code between the VSM code and the fibre placement apparatus, is discussed in this chapter. The different operations, including the control mechanisms and components of the fibre placement unit, are examined. This is followed by the explanation of the robotic arm coding, and the development of the interface code in Matlab. The chapter concludes with the development of the Graphical User Interface (GUI).

6.2 Tools for the fabrication (fibre placement) process

The two previous chapters dealt with the techniques that were employed in the design of laminated fibre reinforced composite structures, and the development of the Matlab codes that assisted with the design process. The focus now shifts to the fabrication of these structures, in particular, the fibre placement/layup process. As mentioned previously, this was to be achieved via Robotic Fibre Placement (RFP). One of the key components of the RFP process is the robotic arm, and for this study, the Mitsubishi RV-2AJ was chosen as it was readily available. This arm is shown in Figure 6.1, and its specifications are given in Appendix E.

Moving from the design phase to the fibre layup process required two steps. The first was the design, control and operation of the robotic end-effector that was used for the orientation and placement of the dry fibres as per the design constraints. This aspect was researched in a previous study [147] and is briefly discussed below. The second step was the program interface between the RFP end-effector and the Variable Stiffness Method (VSM) code. This interface is examined later in more detail.

6.3 Design, control and operation of the robotic end-effector

An end-effector is basically the attachment at the end of a robotic arm. Although end-effectors may be purchased off-the-shelf, they are often tailor designed according to the specific tasks they need to perform. An end-effector was designed in-house to fulfil the following tasks:

1. Pulling of fibre tows
2. Cutting of fibres to specified lengths
3. Orientation of fibres as per the design requirements
4. Placement of fibres on a mould

Figure 6.1: The Mitsubishi RV-2AJ robotic arm chosen for this study [148]

The original design was to have an end-effector incorporating the above functions mounted onto the robotic arm. However, that proved to be cumbersome and infringed on the weight limitations of the arm. Therefore it was decided to design two separate units, with one being situated off the arm and the other being the end-effector. The first unit was used for the pulling and cutting of the fibres, while the end-effector was used to orientate and place the fibres in the mould. The components of both units were

fabricated at a local Further Education and Training (FET) college and were assembled in-house.

6.3.1 The fibre pulling-and-cutting unit

The operation of this unit was to (i) pull the fibre tows to the required length, (ii) cut the fibres, and (iii) hold them in position for collection by the end-effector. The fibre pulling-and-cutting unit is shown in Figure 6.2 with a single fibre tow loaded. The two rollers, labelled Roller A and Roller B, guide the fibre tows into the unit. Roller A is fixed and is fabricated from nylon to ensure the smooth flow of the fibres into the unit. Roller B is free-moving and consists of a metal bar that is coated with rubber. This roller resists the pulling of the fibres thereby aiding in keeping the fibres in tension.

Below Roller B is the Collimator, and its function is to prevent entanglement of the fibre tows. The tows are pulled into the unit via the two pulling rollers, which are made from the same material as Roller B. They are driven by the stepper motor, which is coupled to one of the rollers via a shaft coupler.

The cutting of the fibres is performed by the cutting mechanism, and the fibres are then held in place, for collection by the end-effector, on the Fibre Collection Plate. These two components may be seen in more detail in Figure 6.3. This figure shows the blade that is used for the cutting of the fibres as well as a clamp that grips the fibres prior to cutting. The clamp prevents any movement or crinkling of the fibres during the cutting process. Both the blade and the clamp are pneumatically controlled and their actuators or cylinders are shown in Figure 6.3. The cut fibres are held in place via a perforated plate that is subjected to vacuum pressure. Figure 6.4 shows the vacuum unit located at the rear of the fibre pulling-and-cutting unit directly behind the perforated plate.

The operation of the pulling-and-cutting unit may be summarised in six steps:

1. Activate the stepper motor in order to pull fibres via the pulling rollers.
2. Run the motor until the required fibre length is obtained.
3. Hold the fibres by activating the clamp cylinder.
4. Turn on the vacuum unit to provide vacuum pressure to the perforated plate.

5. Cut fibres by activating the blade cylinder.
6. Wait for collection of cut fibres by the end-effector.

The fibre pulling-and-cutting unit performs the first two of the four tasks listed at the beginning of this section. The second two tasks, the orientation and placement of fibres, are performed by the end-effector in conjunction with the robotic arm.

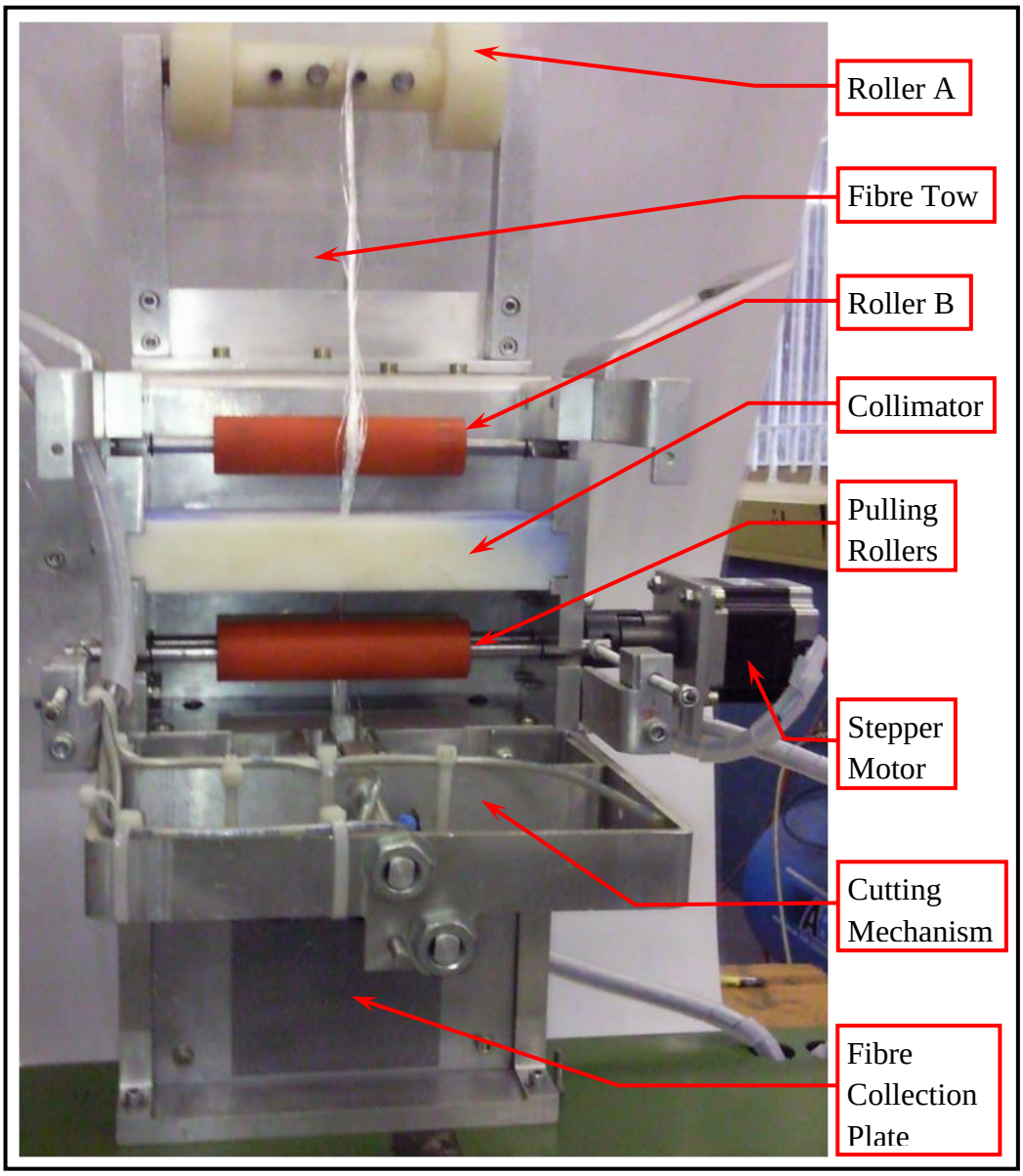


Figure 6.2: Front view of the fibre pulling-and-cutting unit

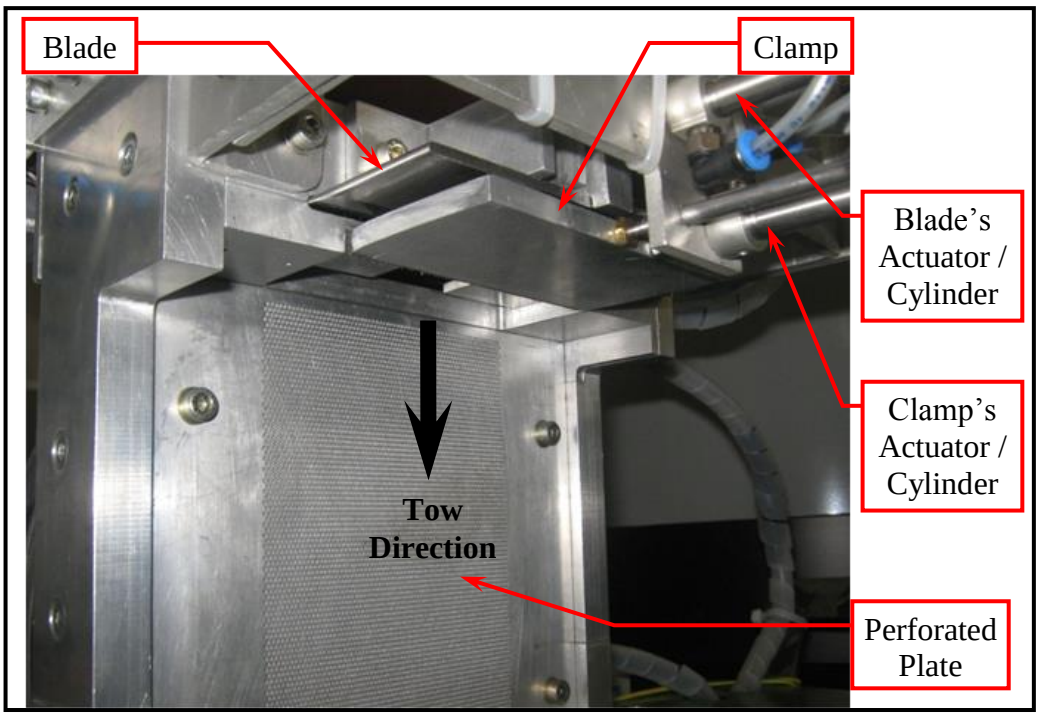


Figure 6.3: Cutting mechanism and Fibre Collection Plate of pulling-and-cutting unit

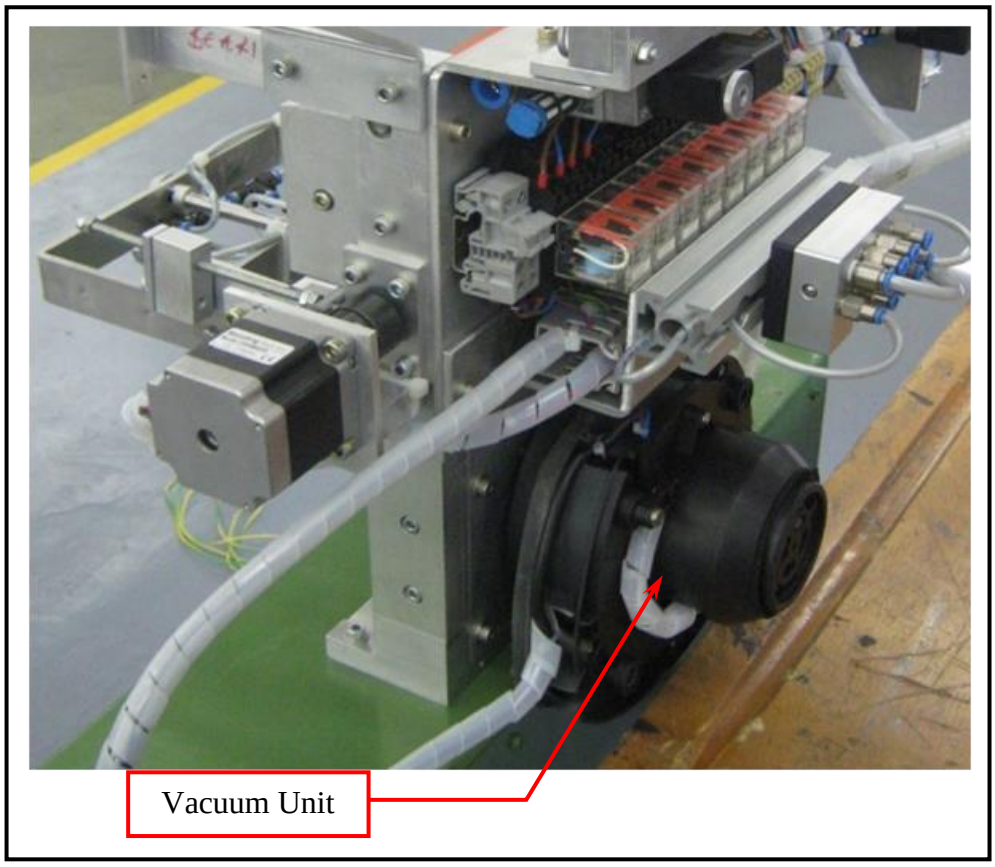


Figure 6.4: Vacuum unit at rear of pulling-and-cutting unit

6.3.2 The fibre orientation and placement end-effector

Once the fibres are cut, they are to be collected by the end-effector for placement in a mould. Figure 6.5 shows the assembled end-effector unit. The Attachment Clamp is used to attach the end-effector to the robotic arm. The collection-placement tool collects the cut fibres from the pulling-and-cutting unit and holds them in place prior to placing them in the mould. This tool uses vacuum pressure, supplied by the vacuum port, to hold the fibres in place. The collection of the cut fibres requires the extension and retraction of the collection-placement tool. This movement is made possible by the guiding rods and the pneumatic cylinder. The cylinder has two proximity sensors attached which are used to determine whether the cylinder is fully extended or fully retracted. The front face of the collection-placement tool is a perforated plate, as shown in Figure 6.6. It was found that Veroboard could be used as the plate because it eliminates the need for machining. Further, the size and distribution of the holes on the board are adequate for the application.

The operation of the fibre placement end-effector may be summarised as follows:

1. Move end-effector to collection point via the robotic arm.
2. Activate pneumatic cylinder on the end-effector to extend collection-placement tool to collect cut fibres from the pulling-and-cutting unit.
3. Simultaneously switch off vacuum on pulling-and-cutting unit and switch on vacuum for collection-placement tool.
4. Activate pneumatic cylinder on the end-effector to retract collection-placement tool to original position.
5. Move end-effector to placement position via the robotic arm.
6. Orientate and place fibres in mould.

Both the fibre pulling-and-cutting unit and the fibre placement end-effector use vacuum pressure to hold the fibres in place. In order to ensure that the placed fibres are kept in their correct orientation and position, the mould is also subjected to vacuum pressure.

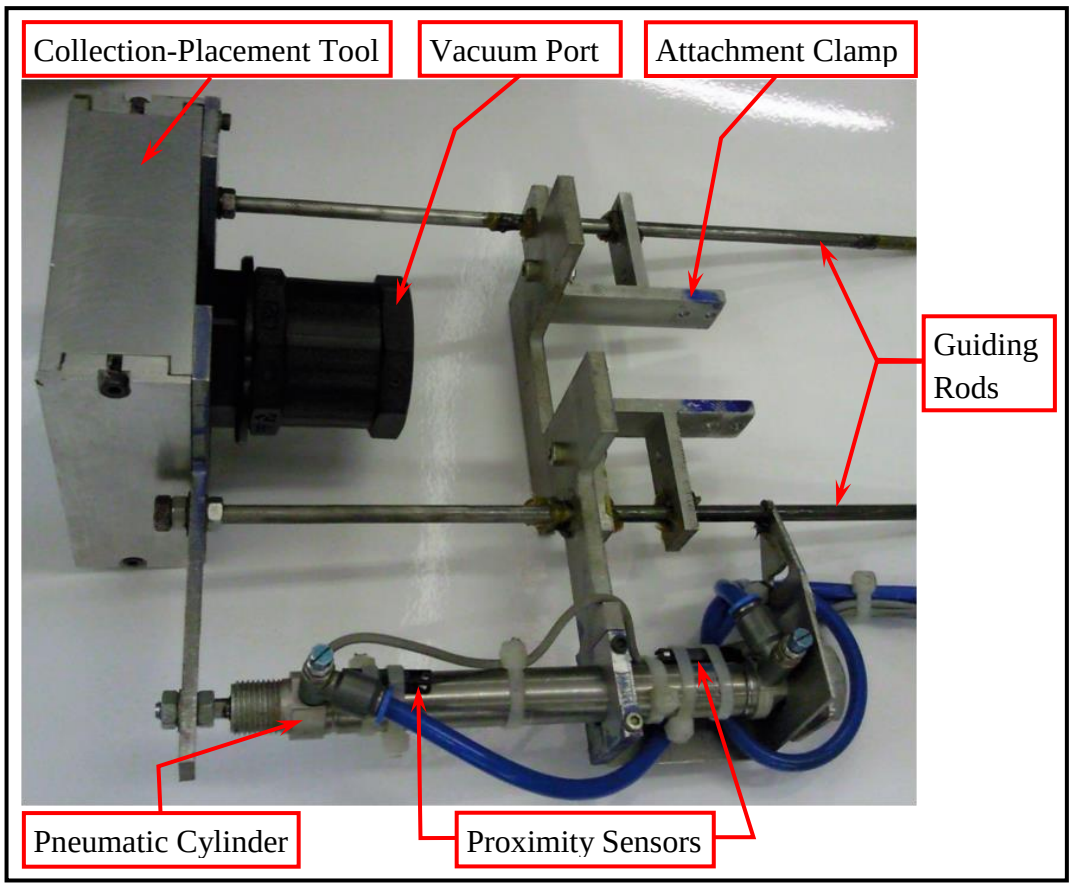


Figure 6.5: Top view of the fibre placement end-effector

Figure 6.6: Front view of collection-placement tool

Thus far the design and operation of the fibre pulling-and-cutting unit and the fibre placement end-effector have been shown and explained. However their interaction with each other and with the robotic arm requires discussion. The next section examines the control interface and operation between these components/systems.

6.3.3 Control interface and operation

The fibre pulling-and-cutting unit consists of various components that perform specific tasks. These components operate either electrically or pneumatically, and their interaction with each other has to be controlled. This is achieved by a S7-200 Siemens Micro PLC with CPU model 224 DC/DC/DC, which is shown in Figure 6.7. The figure also shows additional components that are required to assist with the control interface. These include power supplies, pneumatic and vacuum switches, relays, etc. A high speed counter (not shown) is also used. Control of the pulling and cutting operations are discussed in turn.

The pulling operation involves the pulling rollers and stepper motor, which are shown in Figure 6.2. The motor activates when a pulse is received from the motor controller, shown in Figure 6.7, which in turn is controlled by the programmable logic controller (PLC). Each pulse sent by the controller rotates the motor by 0.9°. In Appendix F, the correlation between the required fibre length and number of pulses is presented. Table F.1 shows the conversion of fibre lengths, from 25 to 150mm, to the number of pulses. This table may also be used in reverse to convert from number of pulses to fibre length.

On the PLC, the number of pulses required for each fibre length is assigned to a separate input. This means that, for the fibre length of 25mm, the number of pulses required is assigned to the first input on the PLC. The number of pulses required for the fibre length of 26mm is assigned to the second input, and so on. A high speed counter (HSC) counts the number of pulses sent to the motor and stops the motor when the count reaches the predetermined value. Once the fibre is pulled to the required length, the cutting operation commences.

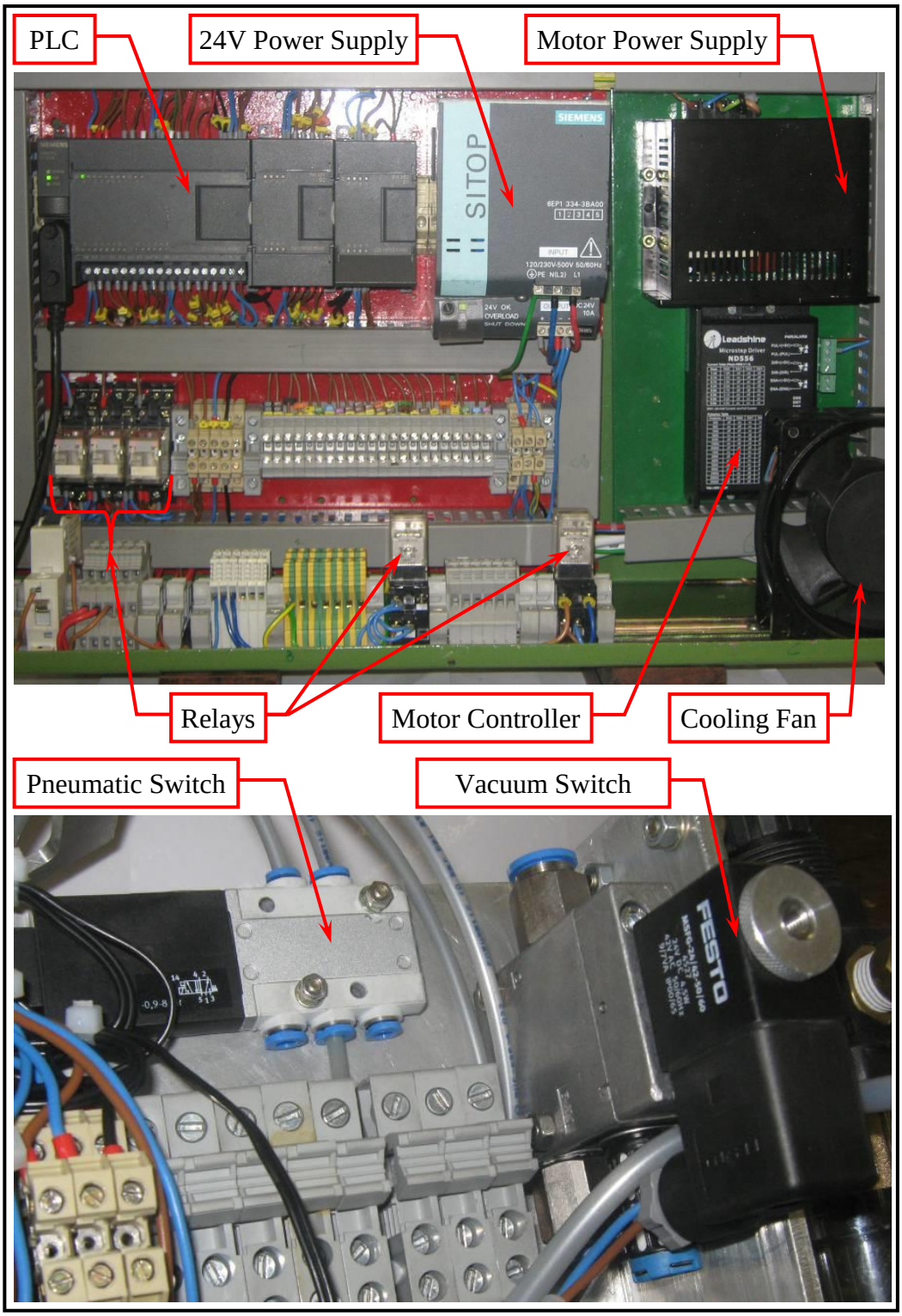


Figure 6.7: Control interface of the fibre pulling-and-cutting unit

The blade, clamp and perforated plate shown in Figure 6.3, and the vacuum unit shown in Figure 6.4, are used in the cutting process. The pneumatic cylinders attached to the

blade and clamp performs the linear motion of these devices. The operation of these cylinders are managed by the PLC via the pneumatic switch shown in Figure 6.7. The vacuum unit that provides vacuum pressure to the perforated plate is controlled by the PLC via the vacuum switch (Figure 6.7).

The components of the end-effector, shown in Figure 6.5, are also controlled by the PLC. The operation of the pneumatic cylinder is governed by means of a pneumatic control valve. As mentioned earlier, this cylinder has two proximity sensors, attached at either end, and these are connected to the PLC. The sensors signals the PLC when the cylinder is fully extended or fully retracted. The PLC also controls when vacuum pressure is supplied to the end-effector as well as to the mould.

The movement of the robotic arm is handled by its own controller, namely, CR1-571. This controller works in conjunction with the PLC to pull, cut, orientate and place fibres. Figure 6.8 shows a flowchart that illustrates this relationship. The blocks in blue represent the PLC commands, while the blocks in orange are associated with commands from the robot controller.

In the first step of the flowchart (Block 1), the robot controller sends a signal to the PLC to start the vacuum unit connected to the mould. The robot controller then sends an instruction to the robotic arm to move to a point where it waits to collect the cut fibres from the pulling-and-cutting unit (Block 2). A signal is sent from the robot controller to the PLC (Block 3) containing the required number of pulses for the specified fibre length.

The PLC activates the stepper motor (Figure 6.3), via its controller, and pulls the fibres into the pulling-and-cutting unit (Block 4). The HSC counts the number of pulses, and stops the motor once the count reaches the predetermined value corresponding to the required fibre length (Block 5). The motor has an electronic brake which is activated when the motor is stopped (Block 6).

In Block 7, the PLC sends a signal to the pneumatic switch (Figure 6.7) to activate the clamp cylinder (Figure 6.3) in order to hold the pulled fibres against the perforated plate (Figure 6.3). The clamp is left in the activated position until the cut fibres are collected. The vacuum unit shown in Figure 6.4 is switched on (Block 8) to keep the fibres in place once they have been cut. In Block 9, the PLC sends a signal to the pneumatic switch (Figure 6.7) to advance the blade cylinder (Figure 6.3) to cut the fibres, and then to retract the blade.

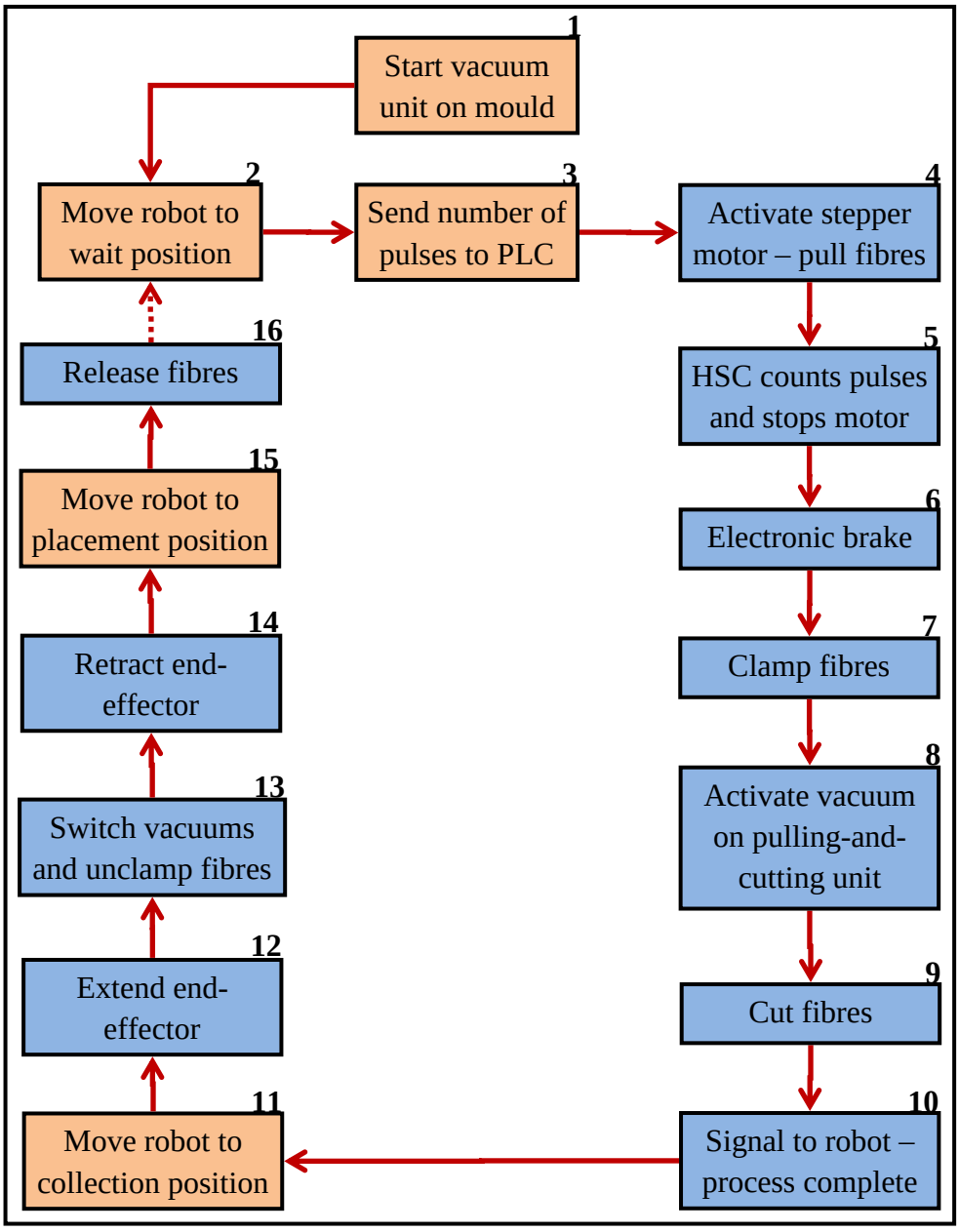


Figure 6.8: Flowchart showing the robot controller and PLC relationship

After cutting, the PLC sends a signal to the robot controller to move the robotic arm to the collection position (Blocks 10 and 11). The pneumatic cylinder mounted on the end-effector (Figure 6.5) receives a signal from the PLC to extend to meet the surface of the perforated plate on the pulling-and-cutting unit (Block 12). A proximity sensor on the cylinder signals the PLC when this movement has been completed.

At this point (Block 13), the PLC is used to simultaneously deactivate the vacuum on the pulling-and-cutting unit and activate the vacuum connected to the end-effector. The clamp holding the cut fibres (Figure 6.3) is then released. In Block 14, the PLC sends an instruction to the pneumatic cylinder on the end-effector to retract. The second proximity sensor on the cylinder sends a signal to the PLC when this step is completed. The robot controller then moves the robotic arm to the designated placement position (Block 15) and orientates the fibres as per the design requirements. The PLC sends a signal to the vacuum switch to deactivate the vacuum on the end-effector which releases the fibres in the mould (Block 16).

In Figure 6.8, Blocks 2 to 16 represents one fibre placement, and therefore this process would need to be repeated for subsequent placements. The vacuum pressure on the mould is kept active until all fibre placements are completed. The fibres are then sprayed with a binder or adhesive to hold the placements and orientations when the mould vacuum is switched off.

This section examined the design, control and operation of the RFP system (pulling-and-cutting unit and fibre placement end-effector) that was developed in the previous study [147]. The next section discusses the development of the software interface between this RFP system and the VSM code.

6.4 Matlab code for Robotic Fibre Placement

In the previous chapter, the VSM code was developed and used to design a fibre reinforced composite structure according to a particular set of loading conditions. The previous section dealt with the design, fabrication and control of the robotic end-effector used for the laying up of fibres in the RFP process. These two systems, that

is, the VSM code and the RFP process, were separate entities. However, in order to move from the design process to the manufacturing phase, the individual systems needed to be linked or related by another component. This necessitated the need for a software interface.

The movements of the robotic arm used in this study may be simulated by means of a software package called COSIMIR Robotics. In this package, the environment in which the robot operates may be modelled in detail. Further, the instructional code that controls the robot's movements and the execution of its other functions may also be developed in this package. In this study the modelling of the working environment was not essential, however, the development of the instructional code was crucial. There are two key components of the instructional code, namely, a position list and a command list.

The position list is simply a record of all the points the robot needs to move to. Each point is defined by rectangular coordinates (X, Y, Z), and, roll, pitch and yaw angles (A, B, C). A position list is created by direct input via the keyboard or by capturing the on-screen location of the robotic arm. The movement of the robot to the points on the position list is done via the command list.

The command list contains the directives the robot must follow. These include the sequence of positions the robot must move to, as well as the appropriate time to conduct these movements. Commands instructing the robot on additional functions that need to be carried out and the correct instance for their execution are also included. The directives in the command list were developed using MELFA Basic IV, which was the recommended programming language for the Mitsubishi RV-2AJ robotic arm.

Once the position and command lists are created, they must be compiled in the COSIMIR package. The generated code may then be transferred to the robot controller which would command the robotic arm to perform all the specified tasks. In this case the tasks involved pulling, cutting, orientating and placing of fibres, or, in other words, the fibre placement process.

As mentioned earlier, the task at hand was to develop a software interface between the design process (VSM code) and the fibre layup method (RFP process). The flowchart in Figure 6.9 illustrates this concept. Matlab was used to develop this interface and the program code was designated VSM-TO-RFP. The basic function of this code was to generate the position and command lists using the outputs of the VSM code. These lists could then be imported into COSIMIR, tested, compiled, and then sent to the robot controller which would then command the robot, with the above designed end-effector, to execute the fibre layup process.

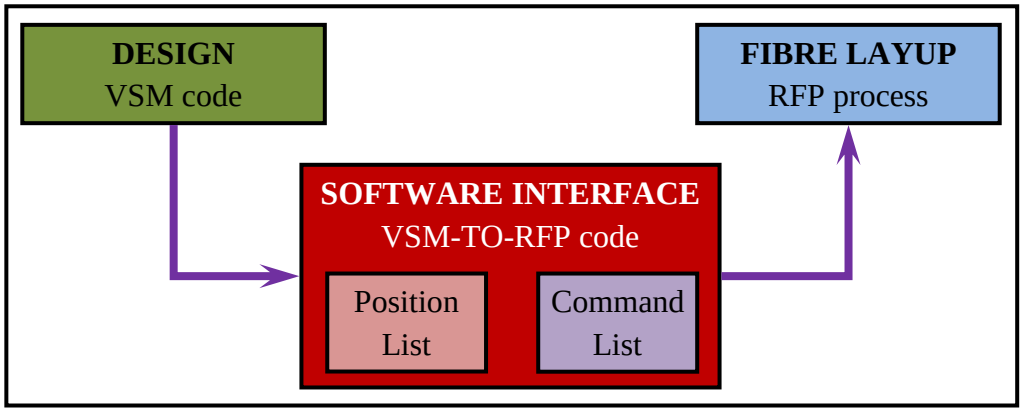


Figure 6.9: Flowchart illustrating link between the design and fibre layup phases

There are two parts to the VSM-TO-RFP code. The first part generates the position list and may be represented by the flowchart shown in Figure 6.10. In Block 1, the data from the finite element model (FEM) is obtained from the saved results file of the VSM design process. This data includes the node positions, list of nodes for each element, etc. Each element from the FEM may exist in one or more planes. Therefore the plane of each element as well as the horizontal and vertical lengths of the element are determined in Block 2.

In the next step (Block 3) the fibre length for each layer of every element in the structure is calculated. The fibre length is determined trigonometrically using the horizontal or vertical length (depending on the plane) of the element and the fibre orientation angle of the layer in question. If the calculated fibre length is a decimal value, it is rounded up to the next whole number. This is done in order to easily obtain

the number of pulses required from Table F.1 in Appendix F. Further, if the fibre length is greater than the constraints of the fibre pulling-and-cutting unit and collection-placement tool, then it is adjusted accordingly.

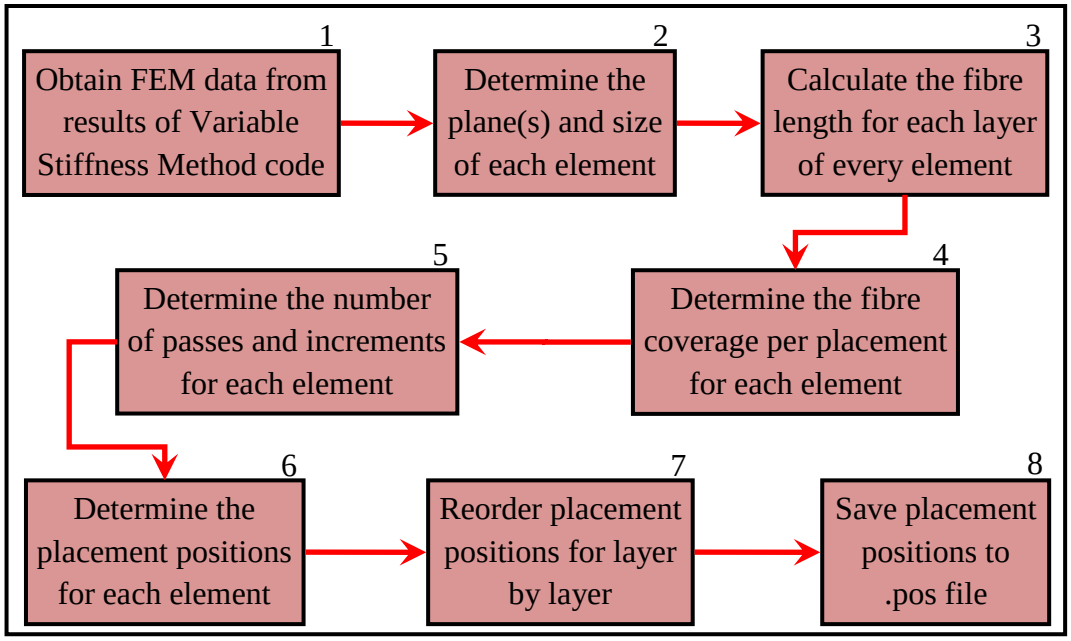


Figure 6.10: Flowchart illustrating generation of the position list

In Block 4, the fibre coverage per placement for each layer of every element is determined. Fibre coverage is the area occupied by the collected fibres when they are placed in the mould. The width of this area or horizontal coverage is determined by the spread of the fibre tows fed into the fibre pulling-and-cutting unit and the cosine of the fibre orientation angle, while the length or vertical coverage is computed from the fibre length calculated in the previous step and the sine of the fibre orientation angle.

Once the fibre coverage for each element is known, the number of horizontal and vertical placements or passes required, for the fibre collection-placement tool to completely cover each element with the essential amount of fibres, is determined (Block 5). The horizontal number of passes for each layer of an element is found by dividing the horizontal length of each element by the horizontal fibre coverage of each

layer in that element. These values are rounded to the next whole number as fractional placements are not possible. The number of vertical passes is similarly found.

In Block 5, a further function is performed and that is to determine the horizontal and vertical increments for each layer of every element. An increment, in this context, is the movement required by the robotic arm to go from one fibre placement position to the next, for each layer in every element. The horizontal and vertical increments are calculated by dividing the horizontal and vertical lengths by the horizontal and vertical passes, respectively.

All the necessary data required to compile the position list has been determined. In Block 6, for each element, the horizontal and vertical increments for each layer (obtained via Block 5) is added to the position of the first node of that element. The result is the fibre placement position (X, Y, Z, A, B, C) for each increment of each layer in every element. These positions are then rearranged in Block 7 to facilitate the placement of the fibres layer-by-layer rather than element-by-element. In other words, the positions are arranged so that the fibres for the first layer of all the elements are placed, followed by the second layer, and so forth. This will ensure that the fibres at the element boundaries overlap evenly and do not form stepped regions.

In the last block in Figure 6.10 (Block 8), the fibre placement positions are saved in a file with a .pos extension. It should be noted that three extra positions are required in the position list. The first of these corresponds to the robot's waiting position (Block 2 in Figure 6.8), while the second is the collection point for the cut fibres (Block 11 in Figure 6.8). The third position is an intermediary point between the wait and collection positions, which ensures that the robotic arm does not collide with the mould or the pulling-and-cutting unit. The Matlab code that generates the position list is shown in Appendix G. This code was used to generate the position list for the graphite/epoxy laminate from the previous chapter. Figure 6.11 shows a portion of this list.

The second part of the VSM-TO-RFP code creates the command list. This list contains the MELFA Basic IV instructional code for the robotic arm and the fibre placement

end-effector. The procedure that is followed in generating the command list is illustrated by the flowchart shown in Figure 6.12.

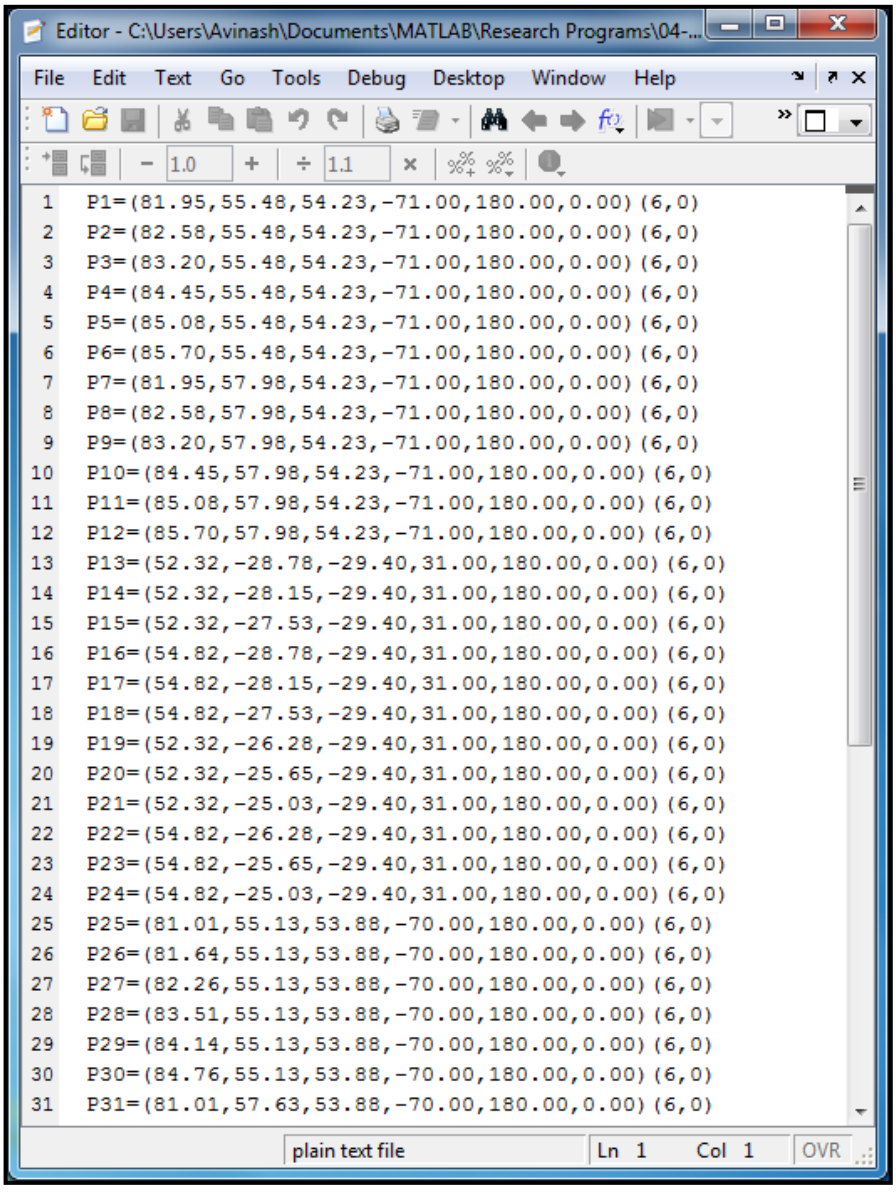


Figure 6.11: Portion of position list for the graphite/epoxy laminate from Chapter 5

In the first step (Block A), the instructional code to start the vacuum pump connected to the mould is generated. Next (Block B), the fibre length for the current fibre placement is converted to the corresponding number of pulses. As discussed earlier, each fibre length is associated with a separate output of the robot controller, which in turn is linked to a unique input of the PLC.

In Block C, the MELFA Basic IV instructional code for Blocks 2 to 16 in Figure 6.8 is generated for a particular fibre length. As mentioned before, this (Blocks 2 to 16) is for a single fibre placement, therefore Block D creates a loop between Blocks B and C to ensure that the instructional code for all fibre placements are created.

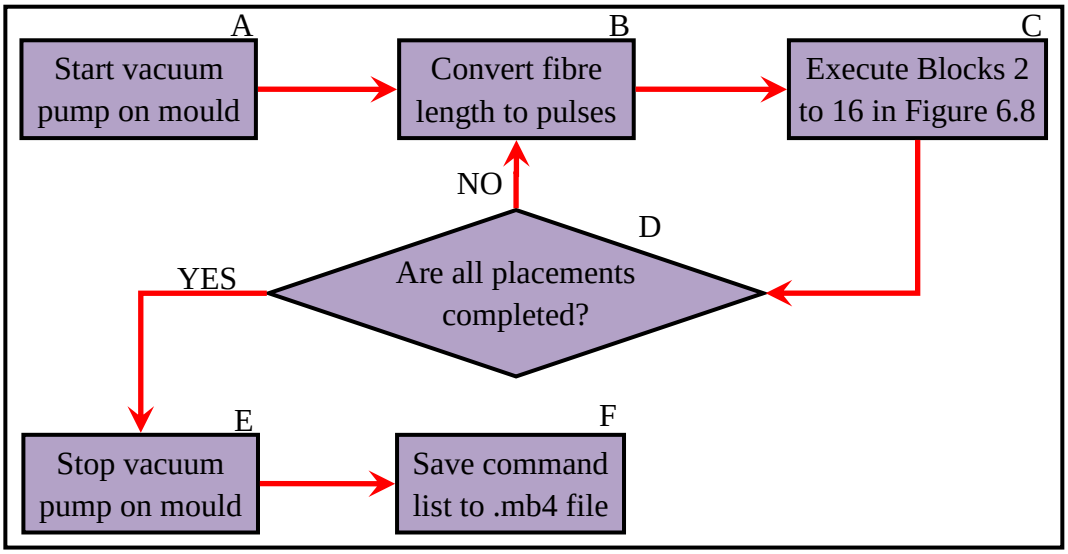


Figure 6.12: Flowchart illustrating the generation of the command list

Once this has been completed, the code to switch off the vacuum pump on the mould is created (Block E), and the generated command list is saved to a file with an .mb4 extension (Block F). The Matlab code for producing the command list is shown in Appendix G. This code was used to create the command list for the graphite/epoxy laminate from the previous chapter. Figure 6.13 shows a portion of the generated command list.

The position and command lists may now be transferred to the robot controller via the professional version of the COSIMIR package to commence the RFP process. The university possesses the educational version, and therefore it was not possible to upload these lists to the controller. However, similar lists were input directly to the controller via the teach pendant and the result was successful fibre placements [147].

As mentioned previously, working in the Matlab environment may prove to be cumbersome. Therefore a GUI was developed for the VSM-TO-RFP code to enable easier input of parameters as well as providing better presentation of results.

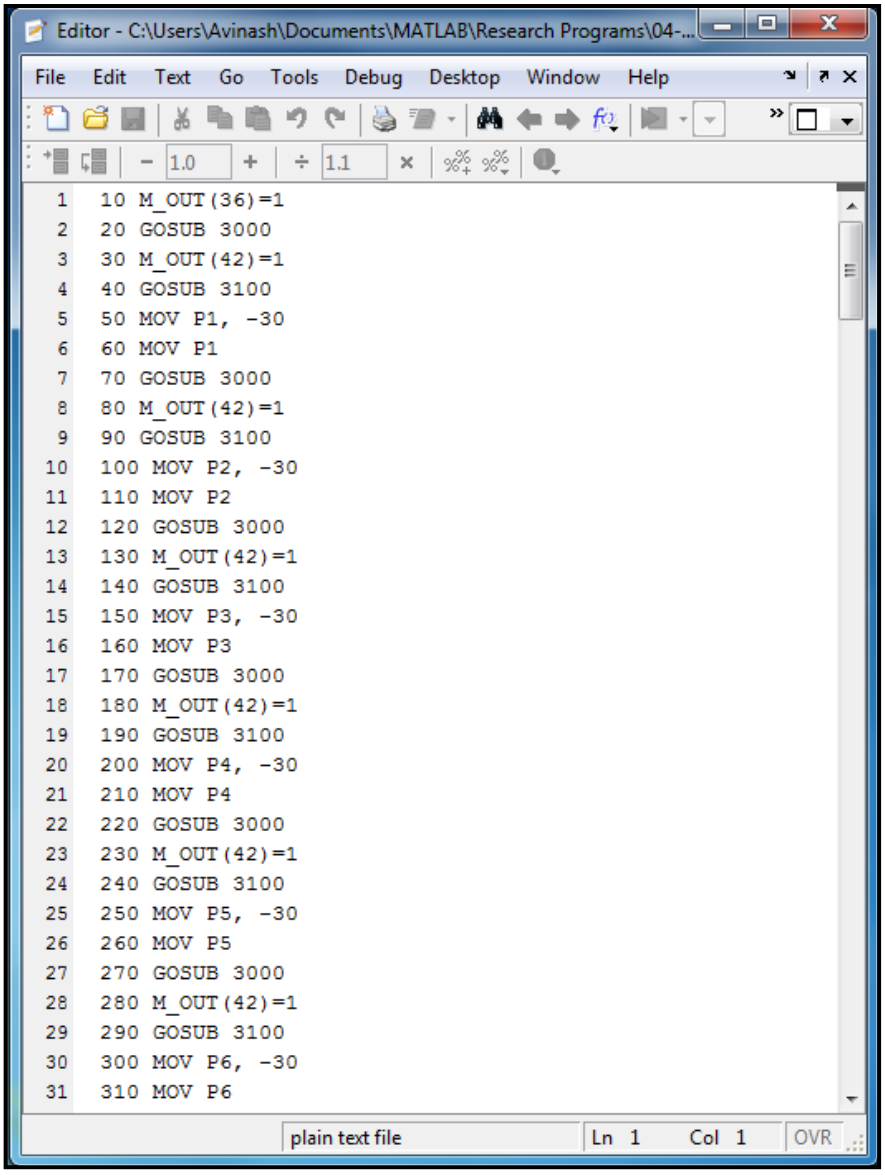


Figure 6.13: Portion of command list for the graphite/epoxy laminate from Chapter 5

6.5 Graphical User Interface (GUI) for the VSM-TO-RFP code

The GUI for the VSM-TO-RFP code is shown in Figure 6.14. It consists of the Import Panel, Input Panel, and Robot Code. Each of these are discussed in turn.

The layout of the Import Panel is shown in Figure 6.15. The panel contains five pushbuttons, one table and a textbox. The “HELP” button, labelled H, initiates the help GUI which is shown in Figure 6.16. This facility gives an on-screen description of the GUI and explains the various input parameters as well as the operation of the pushbuttons. The GUI is dismissed via the “EXIT” button labelled as X.

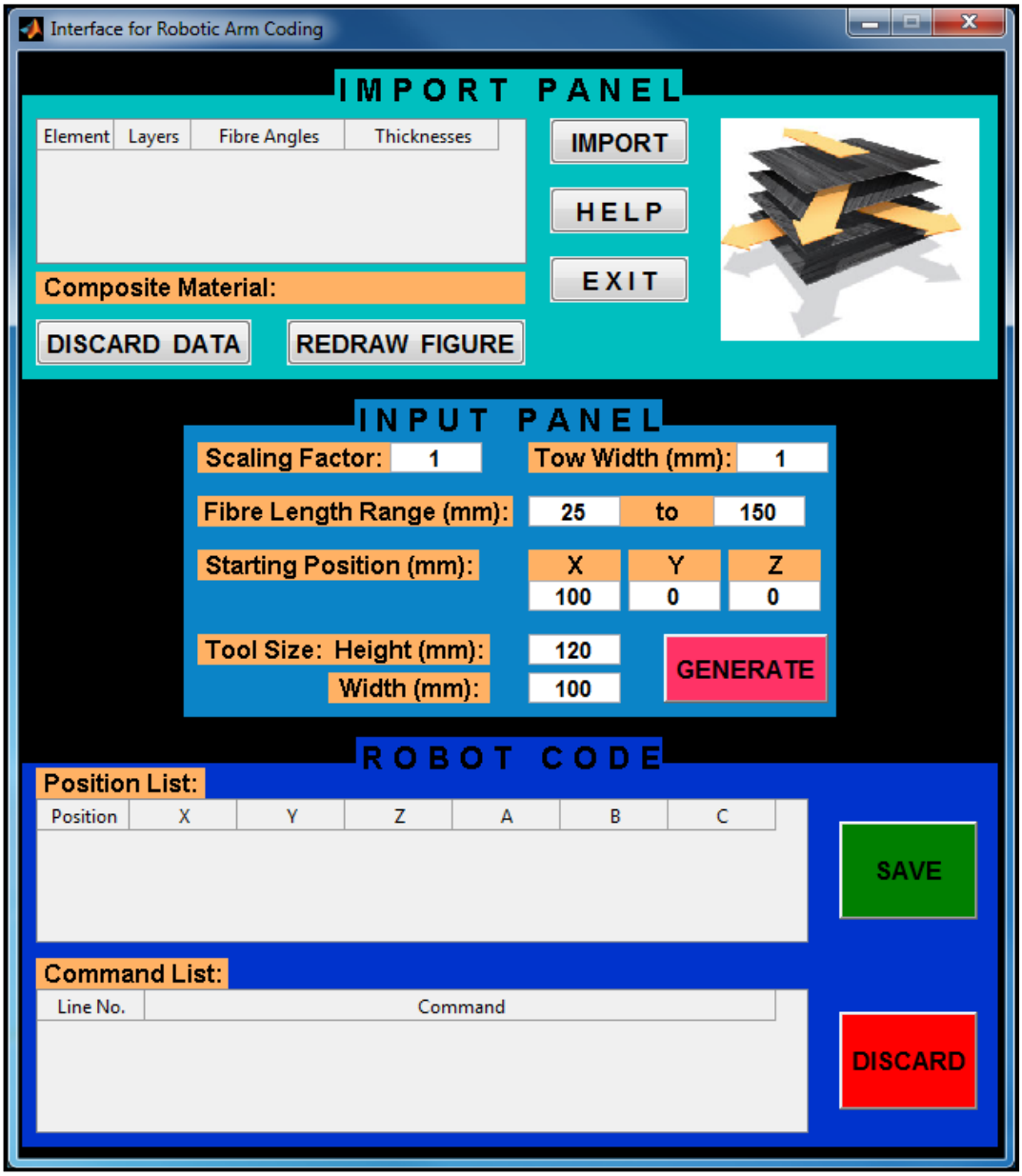


Figure 6.14: Graphical User Interface (GUI) for the VSM-TO-RFP code

The “IMPORT” button, labelled A, is used to import results from the VSM design process. Some of these results, namely, the element number, number of layers in each element, fibre orientation angle of each layer, and the thickness of each layer are presented in the table labelled B. The textbox, labelled C, displays the composite material used in the design process. In addition to the presentation of the imported results, the code also uses the FEM data to generate a plot of the design structure. This plot is prepared in a new window and is displayed automatically once the imported results are loaded. An example of such a plot is shown in Figure 6.17.

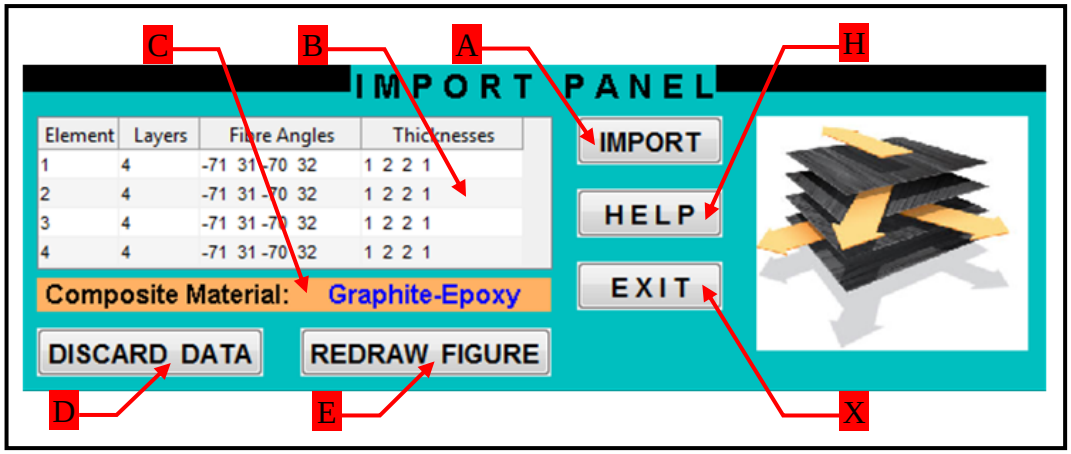


Figure 6.15: Import Panel of the VSM-TO-RFP GUI

The “DISCARD DATA” button, labelled D, clears all data from the tables on the Import Panel and Robot Code panel. It also closes the FEM plot window. The “REDRAW FIGURE” button, labelled E, produces a new plot of the finite element structure if the plot window was previously closed. This action is performed only if there is data available in the table labelled B. Once the results have been imported, it may be converted to MELFA Basic IV format via the Input Panel which is shown in Figure 6.18.

The Input Panel consists of various textboxes, for the entering of input parameters, and one pushbutton. The first parameter, labelled F, is the “Scaling Factor”. This input allows for the resizing of the FEM structure as finite element analyses are generally performed on scale models. The next input parameter is the “Tow Width”, labelled G,

which is the total width of the fibre tows that are loaded into the fibre pulling-and-cutting unit. The minimum and maximum allowable fibre lengths are entered in the textboxes labelled J. The minimum value is the limitation of the fibre pulling-and-cutting unit, and the maximum value is the limitation of the collection-placement tool.

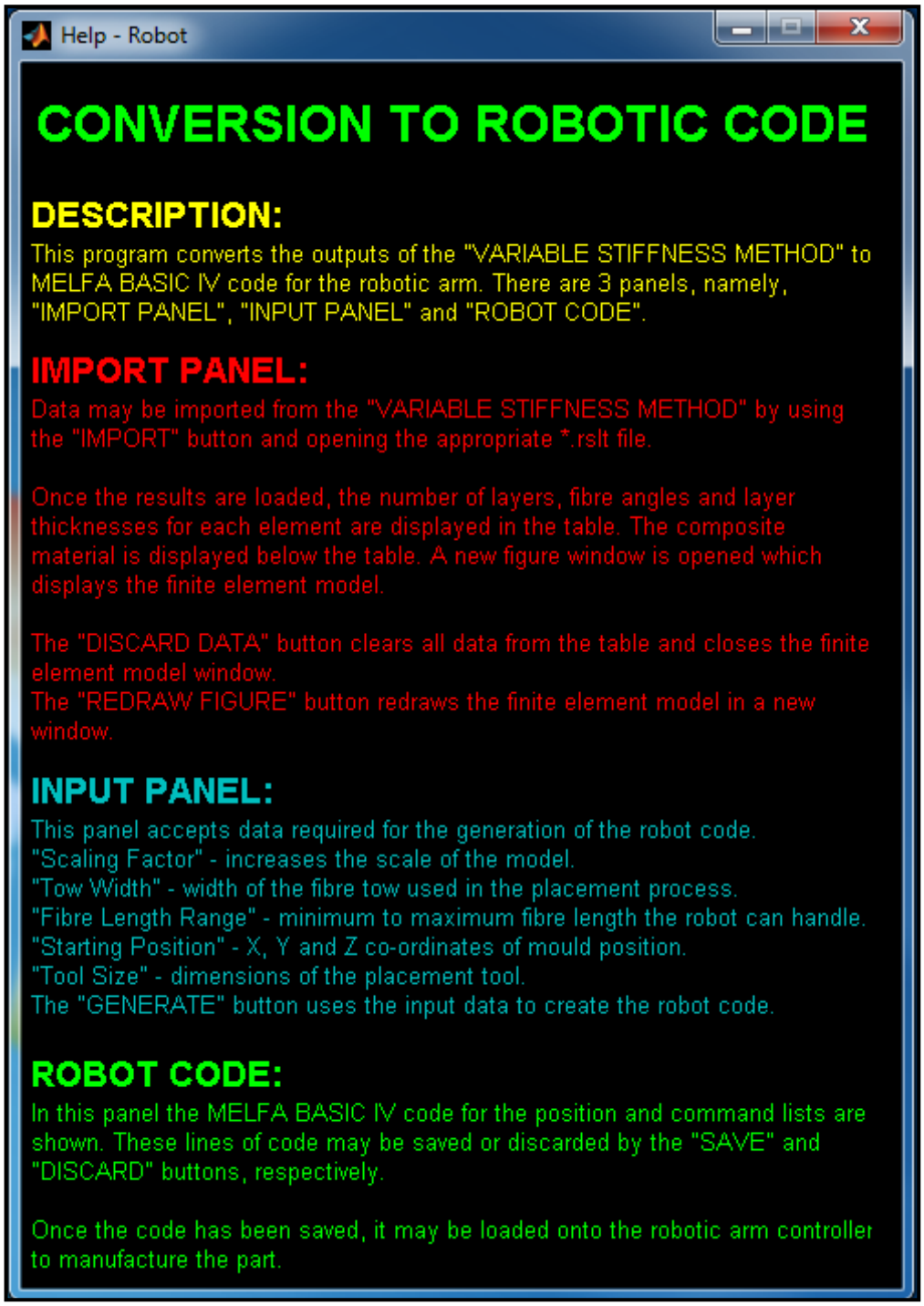


Figure 6.16: Help facility for the VSM-TO-RFP GUI

The robot controller requires the position of the mould, or in other words, the position where fibre placement must begin. The X, Y, Z co-ordinates of this position are entered into the spaces labelled by K. The last of the input parameters is the dimensions of the collection-placement tool which are entered in the textboxes labelled L. The pushbutton labelled M initiates the process of generating the position and command lists. When the button is activated, the GUI code confirms that values have been entered into the input textboxes, and checks that these entries are valid. Any discrepancies produces error messages that prompt the user to rectify the fault. If no errors are encountered, the code that produces the position and command lists is executed. These lists are presented in the tables on the Robot Code panel which is shown in Figure 6.19.

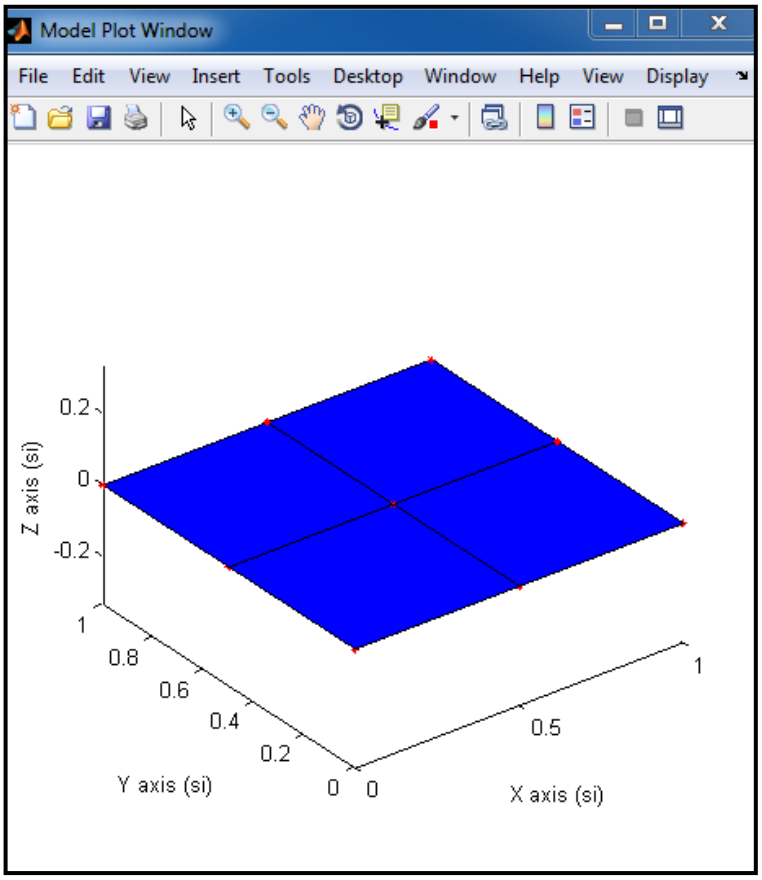


Figure 6.17: Finite element plot window of the VSM-TO-RFP GUI

The Robot Code panel is the last panel of the VSM-TO-RFP GUI. It consists of two tables and two pushbuttons. The tables labelled N and P display the generated position and command lists, respectively. The "SAVE" button, labelled Q, stores the position and command lists in files with their respective file extensions. The button labelled R clears all the data from the Position List and Command List tables. The line-by-line analysis of the Matlab code for the above GUI is performed in Appendix G.

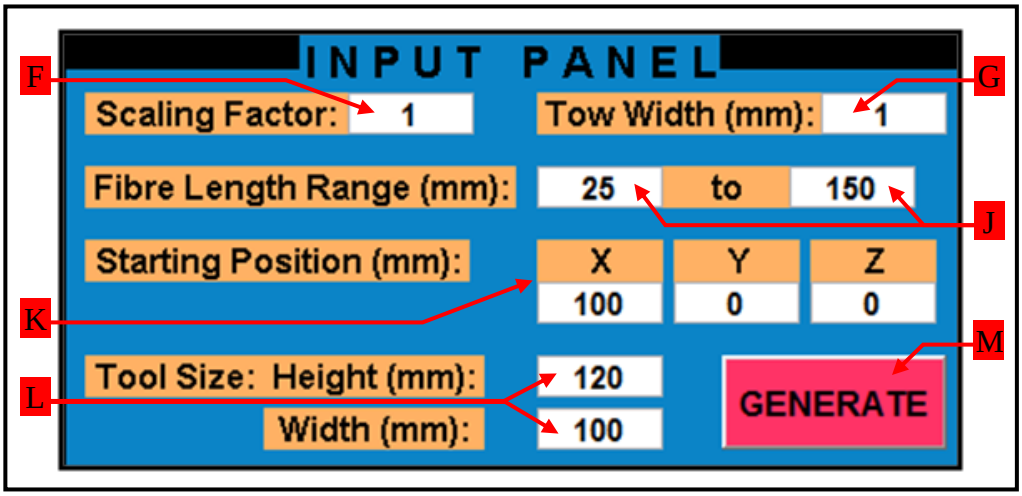


Figure 6.18: Input Panel of the VSM-TO-RFP GUI

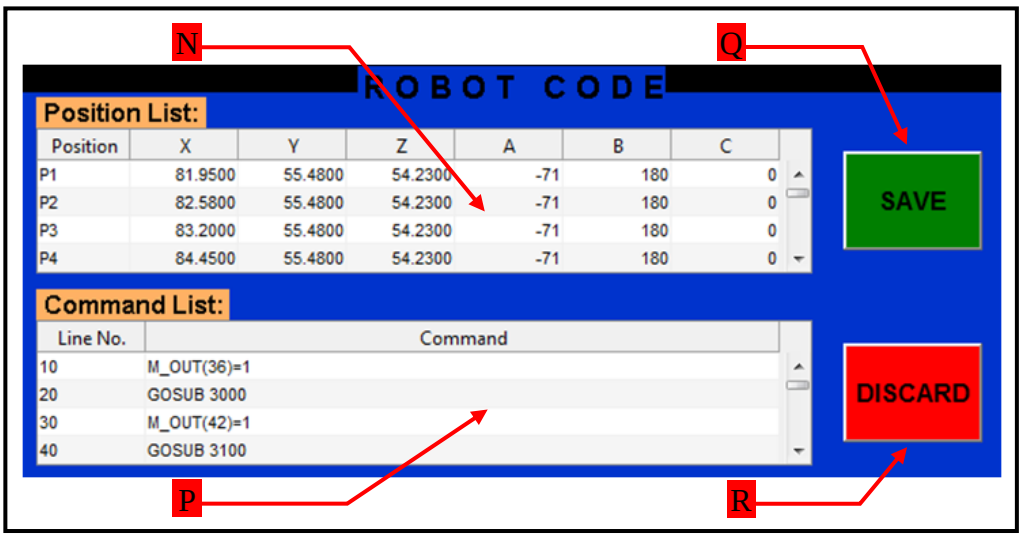


Figure 6.19: Robot Code Panel of the VSM-TO-RFP GUI

6.6 Summary

The focus of this chapter was to fulfil the second research objective stated in Chapter 3. This was to create a software interface between the VSM code, created in the previous chapter, and the RFP system, developed in the previous study.

In this study, the Mitsubishi RV-2AJ robotic arm was employed for the RFP process. The robotic arm required a specialised set of tools to fulfil its role in the fibre placement process. The design, fabrication and control of the purpose-made tools were done in a separate study. Two units were designed and manufactured to accomplish the required tasks, namely, pulling, cutting, orientation and placement of fibres.

The first of these, the fibre pulling-and-cutting unit, was situated off the robotic arm and was used for the pulling and cutting of fibres to the design specifications. It consisted of rubber coated rollers connected to a stepper motor that pulled the fibres into the unit. The fibres were then cut to a specified length via a pneumatically operated blade. The cut fibres were held in place against a perforated plate via vacuum pressure while awaiting collection by the collection-placement end-effector.

The second unit was the robotic end-effector which was simply a vacuum chamber with a perforated face. Its function was to collect the cut fibres from the pulling-and-cutting unit, and release them at the correct placement point in the mould. The orientation and transportation of the cut fibres from the pulling-and-cutting unit to the placement position on the mould was carried out by the RV-2AJ robotic arm.

The functions of the pulling-and-cutting unit were controlled via a PLC while the robotic arm had its own controller. The interaction between the PLC and robot controller was discussed. The mould needed to be kept under constant vacuum pressure to ensure that the placed fibres maintained their orientation and position. This was also controlled by the PLC.

An interface code that was developed in Matlab was used to link the design process (VSM code) and the fibre layup method (RFP process). The code, called VSM-TO-

RFP, generated a position list and a command list. The position list contained the points of all the fibre placements the robotic arm needed to achieve. Each point was defined by rectangular coordinates (X, Y, Z), and twist, pitch and roll angles (A, B, C).

The command list consisted of the instructions the robotic arm must execute. These include the movement of the arm to the points in the position list, as well as commands for the pulling, cutting, collection and placement of the fibres. The command list was generated with MELFA Basic IV instructional code as this was recommended for the Mitsubishi RV-2AJ robotic arm.

A Graphical User Interface (GUI) was developed to enable easier use of the VSM-TO-RFP code as well as allowing for better presentation of results. The Matlab code used for the generation of the position and command lists, as well as for the GUI, is shown and examined in Appendix G.

7. DATABASE OF MATERIALS AND MAIN GRAPHICAL INTERFACE

7.1 Overview

The Matlab codes for the CSM and VSM design processes made use of a database of materials in their respective GUI. This chapter discusses the management of this database as well as other databases that were used. Further, the creation of a main GUI that controls all the previously developed graphical interfaces is also examined.

7.2 Database of materials

There were three databases that were utilised, namely, composite, fibre and matrix. The CSM and VSM codes made use of the composite database only. The other two databases (fibre and matrix) were used for the addition of materials to the composite database via micromechanical techniques. The management of the three databases may be best described directly from the GUI. Figure 7.1 shows the initial GUI for the database management. The GUI consists of an Operations panel with four buttons (labelled A, B, C and D), a "HELP" button (labelled H) and an "EXIT" button (labelled X). Unlike the previous GUIs, this one has two help interfaces. The first, shown in Figure 7.2, describes the purpose of the GUI and the function of each panel, while the second, shown in Figure 7.3, explains the procedures to be followed for each operation. The "EXIT" button closes the interface.

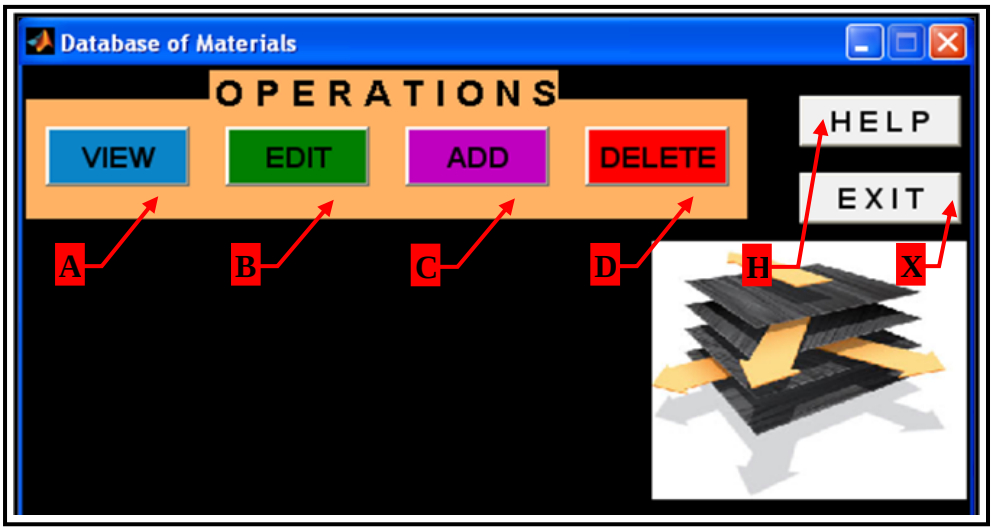


Figure 7.1: Initial Graphical User Interface for the database management

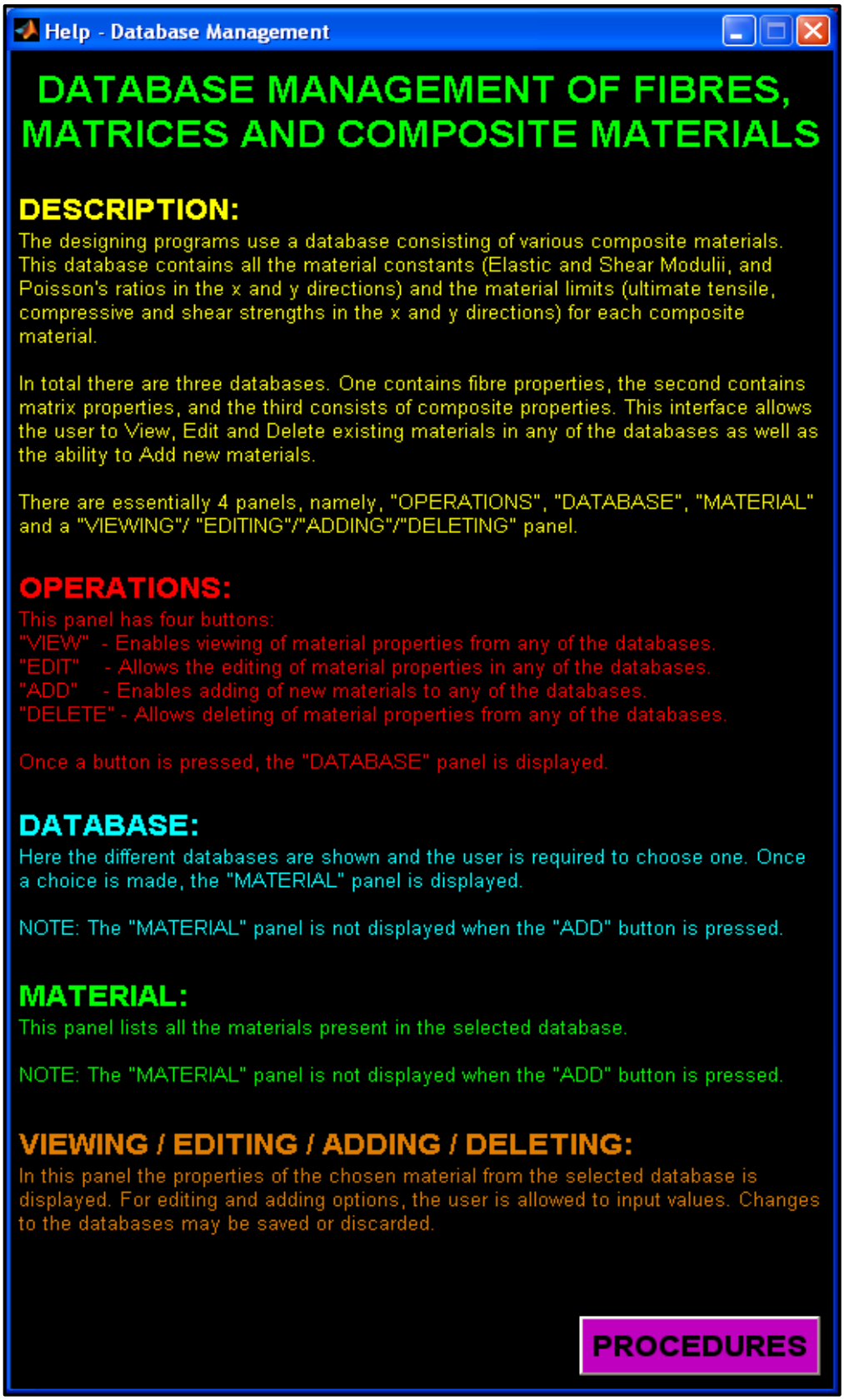


Figure 7.2: First help interface for the Database of Materials GUI

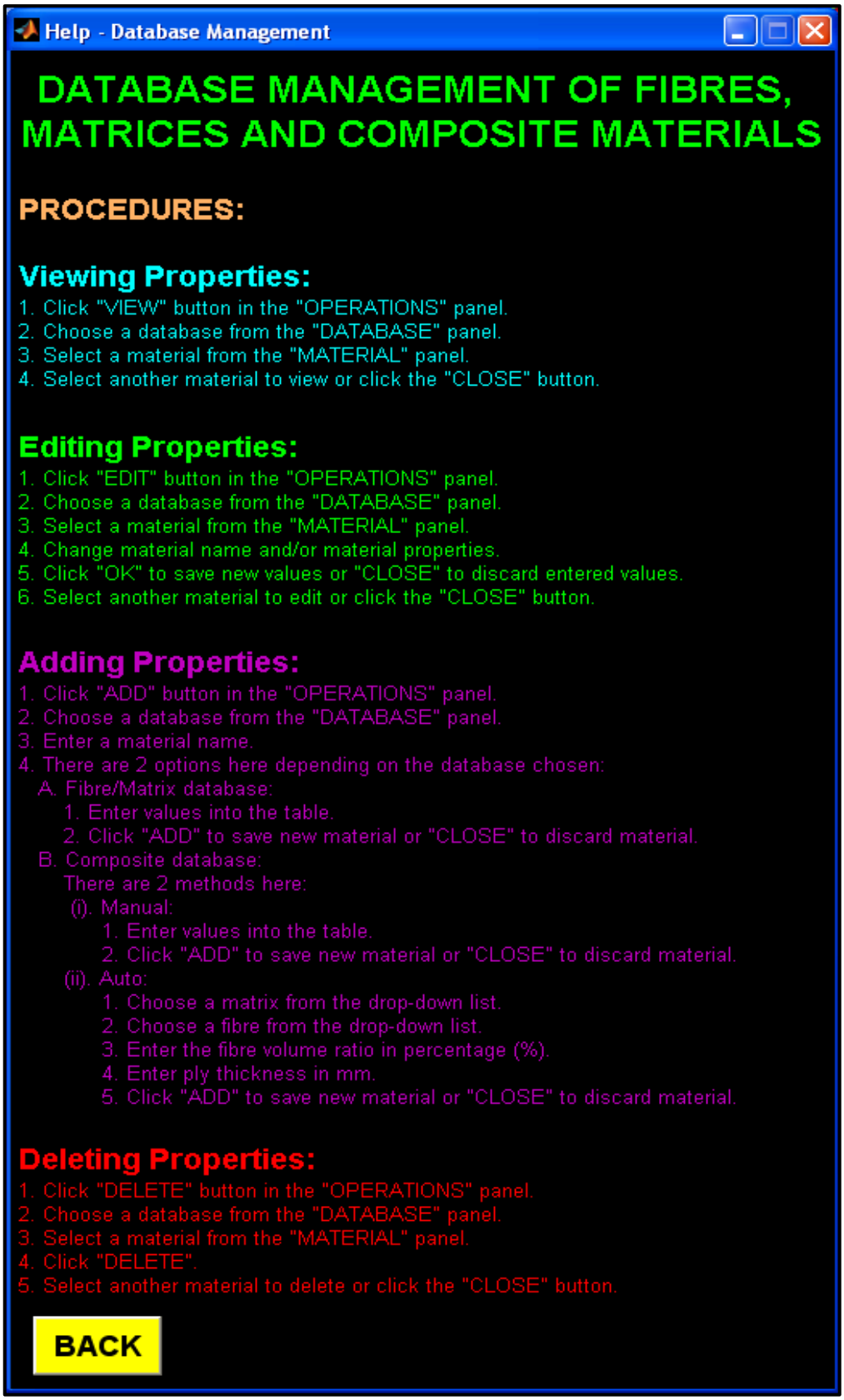


Figure 7.3: Second help interface for the Database of Materials GUI

The buttons on the Operations panel in Figure 7.1 enable the user to view (label A), edit (label B), add (label C) or delete (label D) materials from the three databases. Other panels and layouts are initiated depending on the operation selected. Choosing the "VIEW", "EDIT" or "DELETE" button activates the Database panel, shown in Figure 7.4 and labelled E. This panel is used to select the working database, and, upon a selection, activates the Material panel (labelled F). The user may now select a material to view, edit or delete. In the case where the "ADD" button is chosen, only the Database panel is activated.

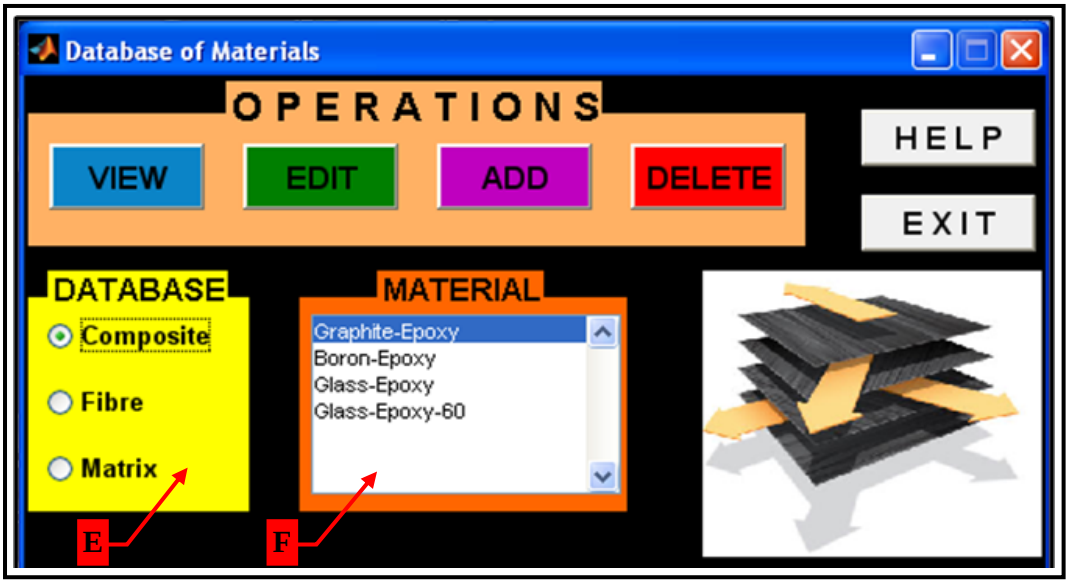


Figure 7.4: Database of Materials GUI showing the database and material selection panels

After the database and material selections have been made, or, in the case of the "ADD" button, just the database selection, the output panel is activated. The layout of this panel differs depending on the selected operation. The various layouts, namely, Viewing Properties, Editing Properties, Adding Properties, and Deleting Properties, are discussed in turn.

The first layout, the Viewing Properties panel, is for viewing the selected material's properties. This layout is shown in Figure 7.5 together with the Operations, Database and Material panels discussed above. The textbox labelled G displays the database

chosen in the panel labelled E. The name of the material selected in the panel labelled F is shown in the textbox labelled J. Properties for the selected material are given in the table labelled K. The table may be scrolled to the right or left to view all the properties, and the cells may also be resized using the mouse. The button labelled Z is used to close the panel and return the interface to its initial appearance as in Figure 7.1.

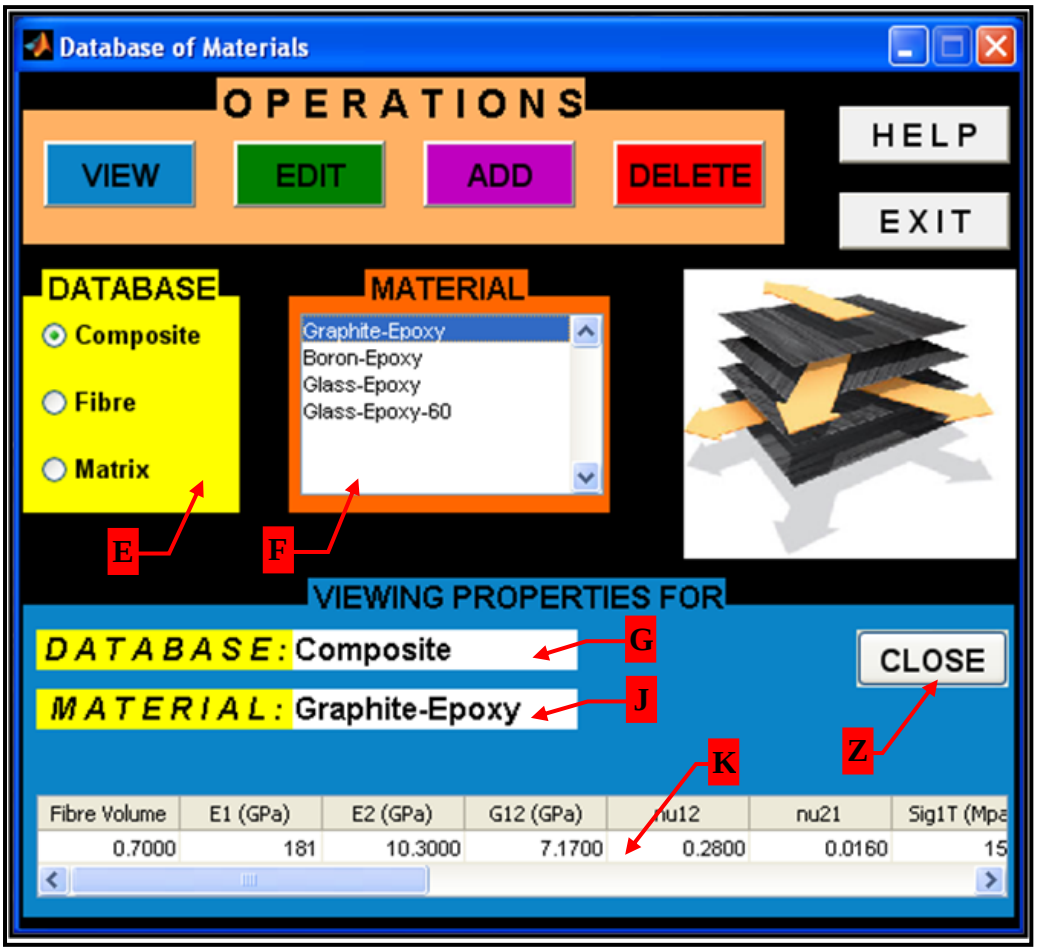


Figure 7.5: Database of Materials GUI showing the Viewing Properties layout

The next layout, shown in Figure 7.6, is used for the editing of a material's properties. As before, the textbox labelled G shows the database that was selected in the panel labelled E, and the textbox labelled L displays the name of the material chosen in the panel labelled F. The properties of the selected material are presented in the table labelled M. However, unlike before, the material name and the entries of the table may

be edited/altered. If there are non-numeric entries in the table or the textbox labelled L is left blank, an error message is displayed that prompts the user to correct the error. After all editing has been completed, the new data is saved using the "OK" button labelled N. The panel is closed and returned to the original interface (Figure 7.1) via the button labelled Z.

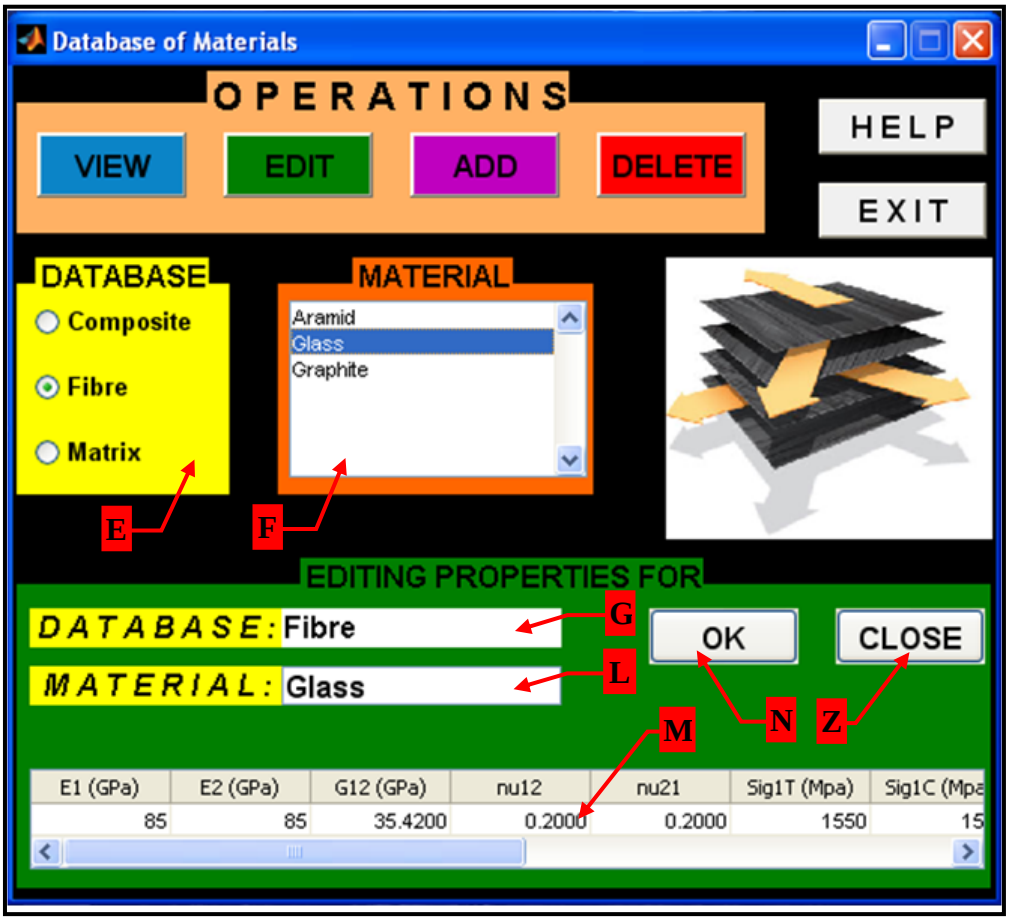


Figure 7.6: Database of Materials GUI showing the Editing Properties layout

The layout for the Deleting Properties panel is shown in Figure 7.7 and this is used to delete a material from any selected database. The name of the selected database is displayed in the textbox labelled G, the name of the chosen material is shown in the textbox labelled P, and the properties of the selected material are presented in the table labelled Q. Pressing the "DELETE" button, labelled R, removes the selected material from the chosen database. The "CLOSE" button, labelled Z, returns the GUI to its original state (Figure 7.1).

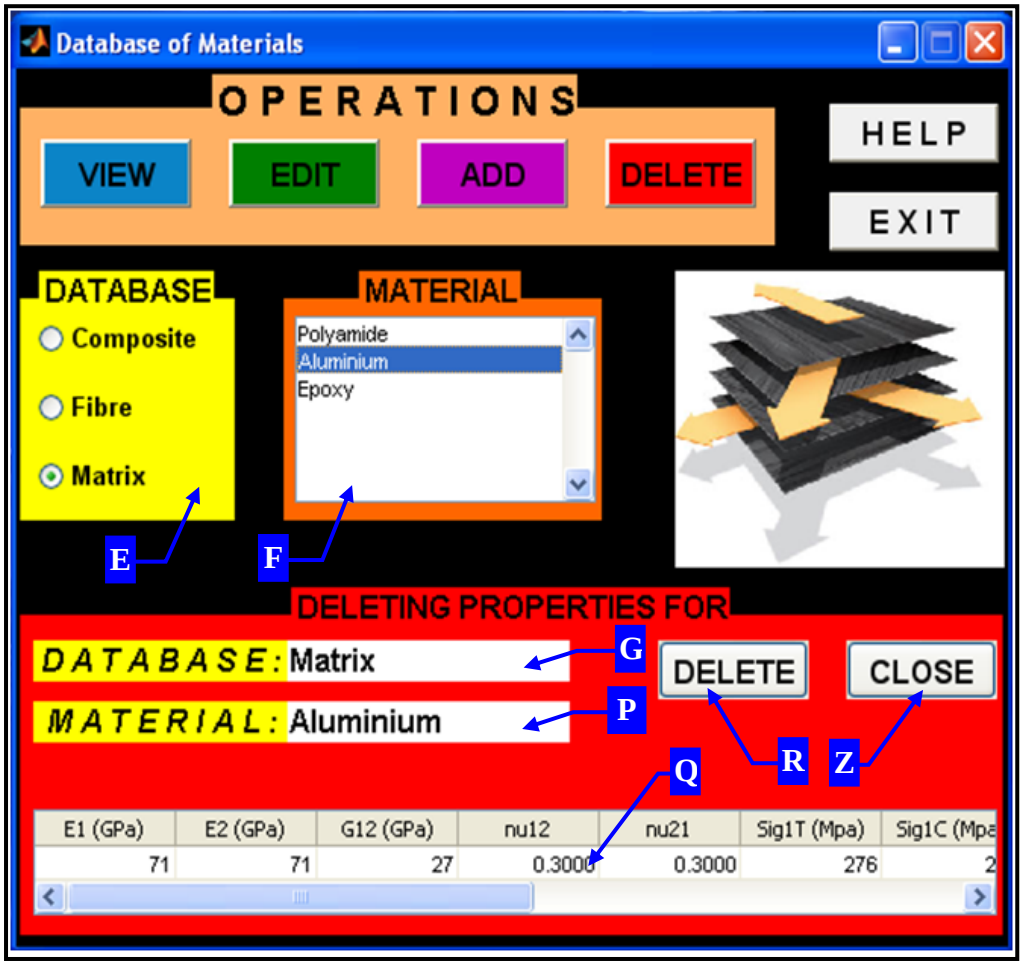


Figure 7.7: Database of Materials GUI showing the Deleting Properties layout

The last layout, the Adding Properties panel, is used to add materials to any of the three databases by manual input or micromechanical methods. The operation of this panel differs from the previous ones. As mentioned earlier, clicking the "ADD" button, labelled C in Figure 7.1, activates the "Database" panel labelled E. Once a database has been chosen, the Material panel is bypassed and the Adding Properties panel is shown. The layout of this panel depends on the choice of database.

If the Fibre or Matrix database is selected then the GUI appears as shown in Figure 7.8. The textbox labelled G displays the chosen database, which in this case would be either the Fibre or Matrix database. The name for the new material is entered into the textbox labelled S, and the material properties are manually entered into the table labelled T. All the entered information is appended to the relevant database via the

"ADD" button labelled U. An error message is displayed if the textbox labelled S is left blank or if there are non-numeric entries in the table labelled T. As before, clicking the button labelled Z closes the panel and returns the GUI to its initial state (Figure 7.1).

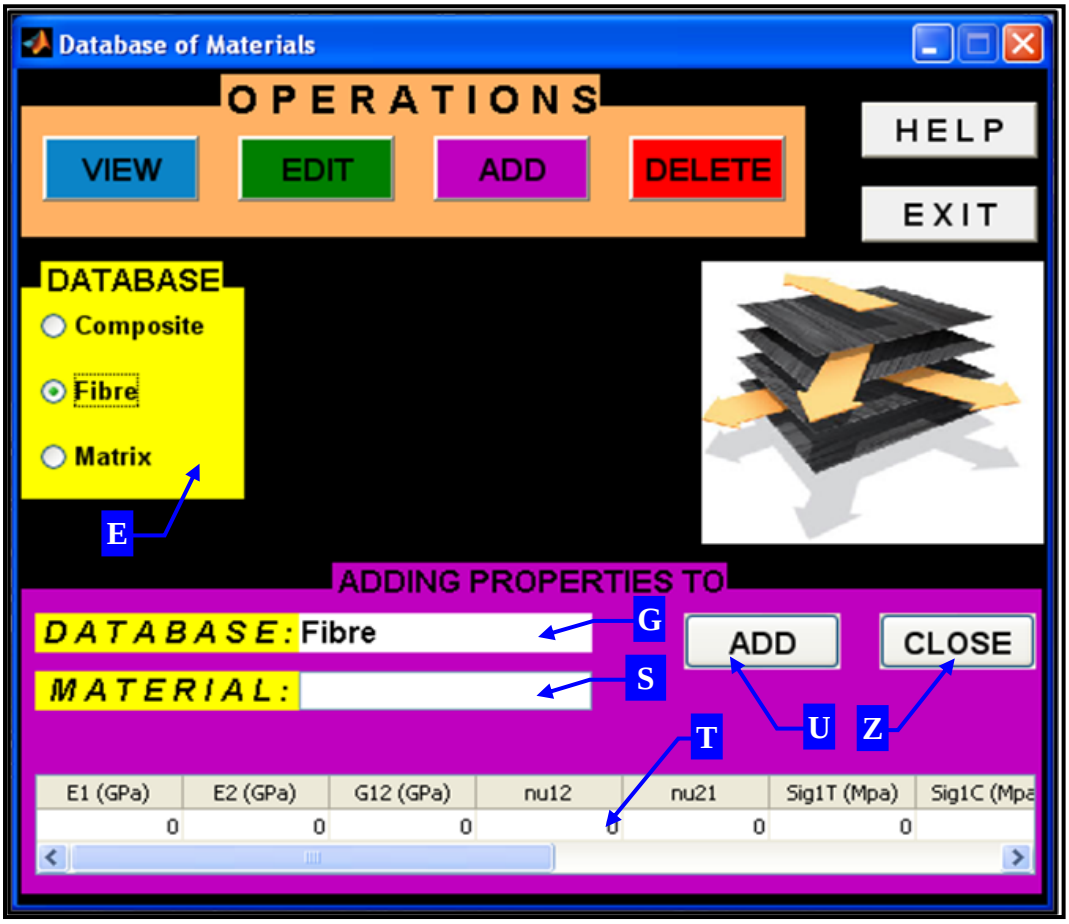


Figure 7.8: Database of Materials GUI showing the Adding Properties layout for the Fibre/Matrix database selection

If the Composite database is chosen, the resulting Adding Properties layout is shown in Figure 7.9. Here, as well, the selected database is displayed in the textbox labelled G and the name of the new material is entered into the textbox labelled S. There are two new buttons available, namely, "Manual" (labelled V) and "Auto" (labelled W). Clicking the "Manual" button changes the panel's layout as shown in Figure 7.10, which is similar to the Fibre/Matrix case. Material properties are manually entered into the table labelled Y, and the "ADD" button (labelled U) is used to append the data to

the database. Error messages are displayed if the textbox labelled S is blank or if there are non-numeric entries in the table labelled Y.

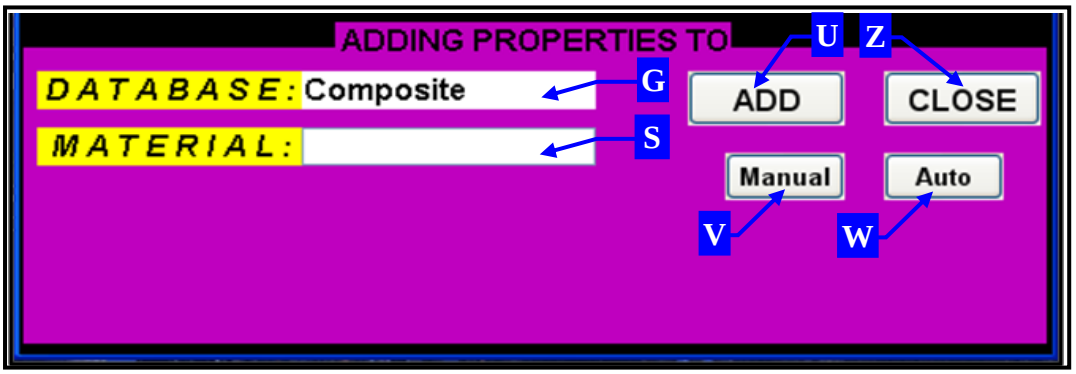


Figure 7.9: Database of Materials GUI showing the Adding Properties layout for the Composite database selection

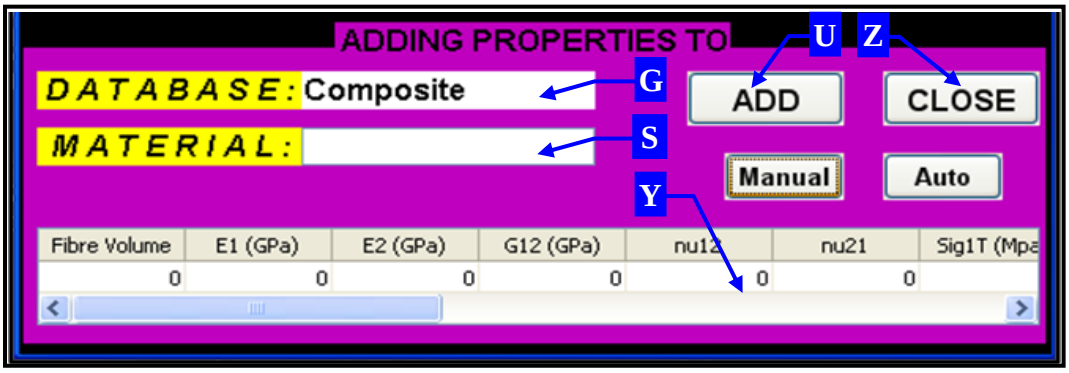


Figure 7.10: Database of Materials GUI showing the Adding Properties layout for the Composite database selection for the "Manual" option

If the "Auto" button, labelled W in Figure 7.9, is clicked, two drop-down lists and two textboxes are activated. These are used to create a composite material via micromechanical methods. This layout of the Adding Properties panel is shown in Figure 7.11. As before, the textbox labelled G displays the selected database, and the name of the new material is entered into the textbox labelled S. The matrix material for the proposed composite is chosen from the drop-down list labelled AA, while the fibre reinforcement is selected from the drop-down list labelled BB. The fibre volume ratio and the thickness of one layer is entered into the textboxes labelled CC and DD, respectively. Error messages are output for non-numeric entries.

The Matlab code for the micromechanical analysis function is executed when the "ADD" button (labelled U) is clicked. This function uses the entered values, and equations 3.28 to 3.33, to determine the material constants for the new composite. The material limits are evaluated via equations 3.34 to 3.47, and the coefficients of thermal and moisture expansion are calculated using equations 3.48 to 3.51. The new material is then appended to the Composite database.

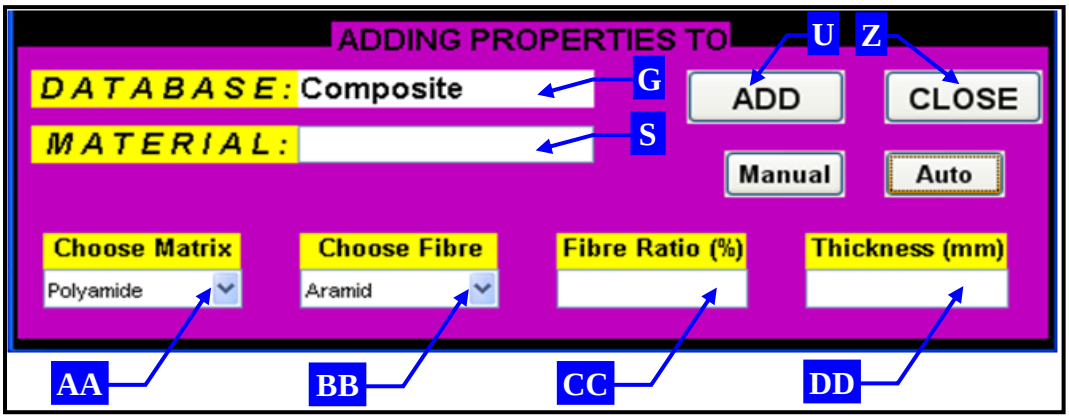


Figure 7.11: Database of Materials GUI showing the Adding Properties layout for the Composite database selection for the "Auto" option

The Matlab code for the above GUI, as well as for the micromechanical analysis function, is examined line-by-line in Appendix H. A further GUI, which controlled all the Matlab codes and GUIs developed thus far, was created in order to minimise the processor and memory resources used by the developed codes. The operation of the new GUI is discussed next.

7.3 Main graphical interface

All the developed Matlab codes and their GUIs were combined into a single package called the Composite Design Assistant. The main interface of this package is shown in Figure 7.12 and consists of six buttons. The buttons labelled 1, 2, 3 and 4 are used to execute the GUI for the CSM, VSM, Database Management, and VSM-TO-RFP codes, respectively. The help function, shown in Figure 7.13, is accessed via the button labelled H, and describes the function of each of the other GUIs. The "EXIT" button labelled X is used to close this interface.

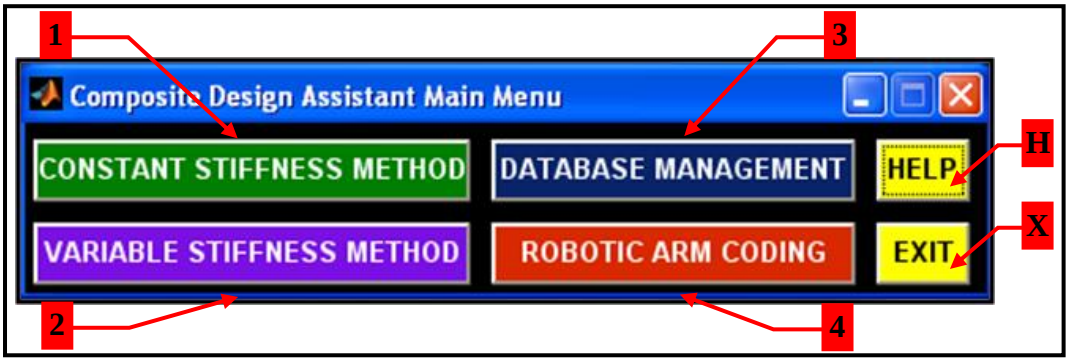


Figure 7.12: Main GUI to control all other Matlab codes and GUIs

The coding for this GUI does not allow simultaneous execution of the various GUIs, that is, only one aspect of the package may be implemented at any given time. For example, if the GUI for the VSM code is executed, then the other GUIs (CSM, Database Management, and VSM-TO-RFP) cannot be run. If the execution of another GUI is attempted, an error message is displayed. The Matlab code for this GUI is shown in Appendix I.

7.4 Summary

The CSM and VSM GUIs each consisted of a drop-down list from where a composite material was chosen for the design process. The materials contained in this drop-down list emanated from a database containing material properties for various fibre reinforced composites. A method was required to allow for the adding and editing of the materials in this database. Therefore a GUI was developed for the management of this database as well as two other databases that contained properties for fibre and matrix materials.

The developed GUI was used for the viewing, editing, addition and deletion of materials from any of the three databases. These operations were discussed in detail. There were help interfaces included that described the purpose and function of the GUI as well as the procedures to be followed in its operation.

Another GUI, the Composite Design Assistant, was also developed. This GUI controlled the execution of all the Matlab codes and GUIs that were developed.

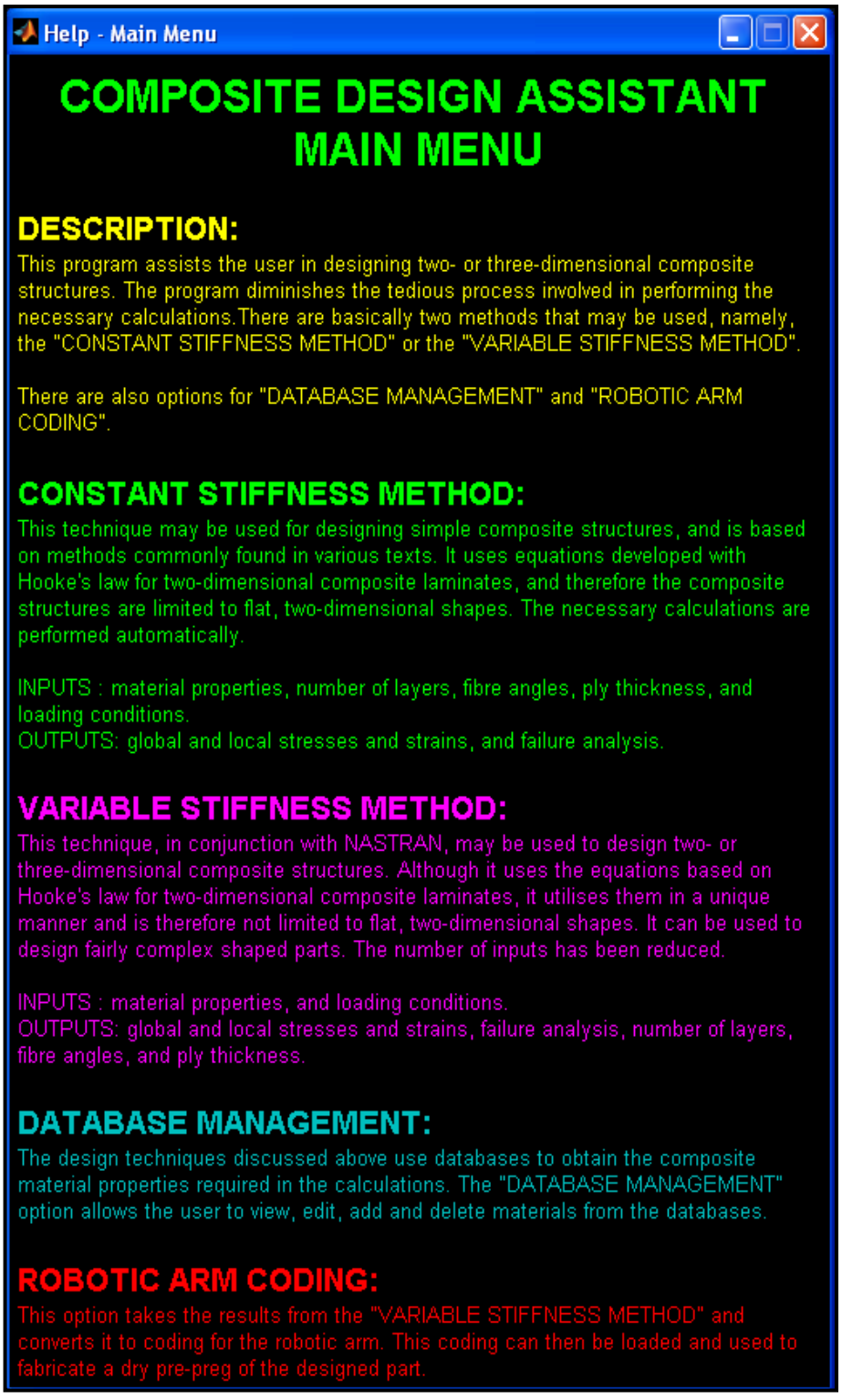


Figure 7.13: Help function for main GUI

8. CONCLUSION

Composite materials, in particular fibre reinforced composites, have superior strength-to-weight and stiffness-to-weight ratios when compared to metals and their alloys. As a result, composites are replacing metals in many applications. In addition, these materials are extremely versatile and may be tailored to satisfy specific design requirements. The fibre orientation can be precisely controlled in order sustain the applied loading more efficiently and produce a structure with superior properties. However, conventional design methods do not allow for this.

Conventional design techniques have the disadvantage of several tedious and laborious matrix calculations, as well as uncertain input parameters for these calculations. In order to overcome this, designers usually make use of a finite set of fibre orientation angles. However, this would result in non-optimised fibre layup parameters and overdesigned structures. Further, the entire structure would be designed for the area of highest stress concentration. This means that although some areas require less reinforcement, the fibre layup would be the same throughout. The result is a constant stiffness structure, which incurs high material and labour costs.

The solution to the above problems was to develop a design technique that performed the necessary calculations easily and automatically, and in a shorter period of time. This technique should not use finite sets of fibre orientation angles, and must optimise the fibre layup properties for maximum strength and minimum weight. Further, it should be able to design two-dimensional and three-dimensional structures for variable stiffness, that is, structures that are multi-strength (varying stiffness) and multi-directional (varying fibre layup). It was decided to use computational software/coding to develop this design method.

The first research objective in this study was to develop a computational code that optimised the fibre layup parameters and created multi-strength, multi-directional structures. This objective contained five goals. The first of these was to develop the computational functions required to perform the relevant calculations effortlessly and

quickly. A code called Constant Stiffness Method (CSM), which was based on the procedure used in conventional design techniques, was developed and discussed in Chapter 4. The code was divided into functions where each function performed a specific calculation. This improved efficiency and enabled easier coding for future codes. The CSM code was validated using a manually calculated example and a finite element model. This model was used to validate the codes developed later.

The CSM code was not capable of optimising the fibre orientation angles or layer thicknesses. Therefore another code, called Fibre-Angle-Opt (FAO), was created and examined in Chapter 5. This code was based on Hooke's Law, and was used to optimise only the fibre orientation angles while keeping the layer thicknesses constant. The CSM code and the finite element model, created above, was used to validate the FAO code. The development of this code satisfied the second goal of the first research objective.

The third goal involved the creation of a computational code that optimised both the fibre orientation angles and layer thicknesses in a fibre reinforced composite laminate. This code, which was called Angle-Thick-Opt (ATO), combined the FAO code with a thickness optimisation function. This function was run periodically during the angle optimisation process, and was used to combine the thicknesses of adjacent layers, if certain criteria were met. The ATO code also optimised the number of fibre layers and the weight of the laminate. The code was validated using the Patran environment.

Since the above codes were based on Hooke's Law, they could only be used on two-dimensional entities. However, structures are three-dimensional and therefore the ATO code was expanded to cater for this. A code, called Variable Stiffness Method (VSM), was developed and used to design multi-strength, multi-directional structures (fourth goal of first objective). For this code, the structure was first divided into finite elements. Next, the VSM code was used to associate each element with its loading conditions, and then determine the fibre layup parameters, for each element, using the ATO code. The code was validated using finite element analysis and experimental results.

The development of the above code created a new problem with regards to the fibre layup process. The manual fibre placement techniques were inadequate because they increased labour costs and fibre orientation could not be accurately controlled. The solution was to use an automated fibre placement method. For this study, the Robotic Fibre Placement (RFP) process was chosen. The RFP process uses a robotic arm, with a purpose-made end-effector, to lay fibres in a mould. It is able to perform the fibre layup process rapidly and precisely control the orientation and placement of fibres.

A RFP system was designed and built in a previous study. The secondary objective of this study was to create an interface between the design code developed above and the RFP system from the previous study. This was discussed in Chapter 6. The first of two goals for the second objective was to examine the design and operation of the components of the RFP system. This system comprises of a fibre pulling-and-cutting unit which is situated off the robotic arm and used for the pulling and cutting of fibres according to the design specifications, a fibre collection-placement tool which is used to collect the cut fibres and place them in a mould, and a Mitsubishi RV-2AJ robotic arm which transports and orientates the cut fibres.

The second goal of the second objective was to create a software interface between the VSM code and the developed RFP process. A Matlab code called VSM-TO-RFP was developed and used to generate the position and command lists required for fibre placement. The position list contains the coordinates and orientations of the fibre placement points and the command list consists of the instructions the RFP system must follow, which includes the pulling, cutting, collection and placement of the fibres.

The last objective of this study contained three goals which included the development of Graphical User Interfaces (GUIs), a database of materials to be used in the design process, and a system that manages this database. GUIs were developed for the CSM, VSM, VSM-TO-RFP and database management codes in order to enable better/easier interaction between the user and the various codes. Each of the developed GUIs also contains a help facility that describes the purpose of the GUI and the procedures to be

followed in its operation. A main GUI was created and used to control the execution of all the developed GUIs and codes.

A database of composite materials was created and made available in the CSM and VSM GUIs as drop-down lists. The user is able to choose a material from these lists to be used in the design process. A code was developed that enables the user to maintain (view/edit/add/delete) entries in this database as well as two others containing properties for fibre and matrix materials. These two databases may be used to create composite materials via micromechanical methods.

The fulfilment of the three research objectives of this study, achieves the hypotheses set out in Chapter 3. A design process has been developed, using a computational code, which optimises the fibre orientation angle and thickness of each layer, and consequently the number of layers and weight, in a fibre reinforced composite. This code may be used to design multi-strength and multi-directional, or, in other words, variable stiffness structures. Additionally, a software interface between the design optimisation code and a fibre layup process was created. This enables the high speed, high precision processing of dry pre-pregs for composite structures.

Recommendations for future work include the testing of the code in the design of more complex shaped structures, such as structural frames, cargo containers, turbine blades, pressure vessels, wing structures, etc. Further, the code should output multiple sets of optimal fibre layup parameters for each element. The designer should then have the option to choose the fibre layup parameter set they prefer.

In addition, the GUI/software must be adapted to give the user more options/control over the design parameters. These include allowing the user to define the upper limit for the *SR* values, whether to run the thickness optimisation function, and to define the range used in the thickness optimisation function. The code should also incorporate safety factors in order to comply with design standards.

REFERENCES

[1] Ashby, M. F., "The evolution of engineering materials", Department of Engineering, Cambridge University, England, http://smart-materials.blogspot.com/2011/06/evolution-of-engineering-materials.html

[2] Bellchamber, J., "Metal 101 – A History of Alloys", Metalcraft Magazine, 2005, http://www.bellchamber.net/Publication/History%20of%20Alloy.htm

[3] Bellis, M., "The History of Plastics: Timeline of Plastics", About.com, http://inventors.about.com/od/pstartinventions/a/plastics.htm

[4] Christensen, R. M., "Mechanics of Composite Materials," Dover Publications, New York, 2005.

[5] Barakat, S. A. and Abu-Farsakh, G. A., "The Use of an Energy-Based Criterion to Determine Optimum Configurations of Fibrous Composites," Composite Science and Technology, Vol. 59, No. 12, 1999, pp. 1891–1899.

[6] Park, C. H., Lee, W. I., Han, W. S. and Vautrin, A., "Weight Minimization of Composite Laminated Plates with Multiple Constraints," Composite Science and Technology, Vol. 63, No. 7, 2003, pp. 1015–1026.

[7] Almeida, F.S. and Awruch, A.M., "Design optimization of composite laminated structures using genetic algorithms and finite element analysis," Composite Structures, Vol. 88, 2009, pp. 443–454.

[8] About.com, "NABI Unveils Bus with Composite Body", About.com, http://composite.about.com/library/PR/1999/blnabi1.htm

[9] Kaufmann, M., Zenkert, D., and Mattei, C., "Cost optimization of composite aircraft structures including variable laminate qualities", Composites Science and Technology, Vol. 68, 2008, pp. 2748 – 2754

[10] Park, C.H., Saouab, A., Bréard, J., Han, W.S., Vautrin, A., and Lee, W.I., "An integrated optimisation for the weight, the structural performance and the cost of composite structures", Composites Science and Technology, Vol. 69, 2009, pp. 1101 – 1107

[11] Mazumdar, S.K., "Composite Manufacturing: Materials, Product, and Process Engineering", CRC Press LLC, 2002

[12] Akbulut, M., and Sonmez, F.O., "Optimum design of composite laminates for minimum thickness", Computers and Structures, Vol. 86, No. 21–22, 2008, pp. 1974 – 1982

[13] Wikipedia, "Composite Material", Wikipedia, the free encyclopaedia, December 2006, http://en.wikipedia.org/wiki/Composite_material

[14] Cooperative Research Centre for Advanced Composite Structures Ltd, "Putting it together – the science and technology of composite materials", NOVA Science in the News, November 2000, http://www.science.org.au/nova/059/059key.htm

[15] About.com, "What's a Composite?", About.com, http://composite.about.com/od/aboutcompositesplastics/l/aa060297.htm

[16] Course Notes: Mechanics of Composite Materials, University of Kwa-Zulu Natal, 2001

[17] Wikipedia, "Fibre-reinforced plastic", Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Fibre-reinforced_plastic

[18] Chung. Natalie Y.L., "Fibre Reinforced Polymer (FRP) Composites", http://gnatchung.tripod.com/FRP/

[19] Strassler, H.E., "Fiber-Reinforcing Materials for Dental Resins", Inside Dentistry, 2008, Vol. 4, Issue 5

[20] Fiberglass Grating Manufacturers Council, "Why specify FRP?", American Composites Manufacturers Association, http://www.acmanet.org/fgmc/why_specify.htm

[21] Kainer, K.U., "Metal Matrix Composites: Custom-made Materials for Automotive and Aerospace Engineering", John Wiley & Sons, 2006

[22] Wikipedia, "Thermosetting Polymer", Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Thermoset

[23] Wikipedia, "Thermoplastic", Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Thermoplastic

[24] Wikipedia, "Natural Fiber", Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Natural_fibres

[25] Schuh, T.G., "Renewable Materials for Automotive Applications", Daimler-Chrysler AG, Stuttgart, http://www.ienica.net/fibresseminar/schuh.pdf

[26] Wikipedia, “Synthetic Fiber”, Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Synthetic_fiber

[27] Julius, “Carbon Audi-TT”, SlotForum, http://www.slotforum.com/resources/CarbonTT/index.htm

[28] Wikipedia, “Carbon Fiber”, Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Carbon_fiber

[29] Hegde, R.R., Dahiya, A., and Kamath, M.G., “Carbon Fibers”, http://www.engr.utk.edu/mse/Textiles/CARBON%20FIBERS.htm

[30] Wikipedia, “Carbon-fiber-reinforced polymer”, Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Carbon-fiber-reinforced_polymer

[31] Kopeliovich, D., “Kevlar (aramid) fiber reinforced polymers”, SubsTech, Substances and Technologies, http://www.substech.com/dokuwiki/doku.php?id=kevlar_aramid_fiber_reinforced_polymers

[32] Wikipedia, “Aramid”, Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Aramid

[33] AZoM.com, “Aramid Fibre (Kevlar / Twaron)”, AZoM.com, The A to Z of Materials, http://www.azom.com/article.aspx?ArticleID=1384

[34] Wikipedia, “Fiberglass”, Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Glass-reinforced_plastic

[35] Gupta, V.B., and Kothari, V.K., “Manufactured Fibre Technology”, London: Chapman and Hall, 1997, 544-546, ISBN 0-412-54030-4

[36] About.com, “Glass Fiber Reinforcements”, About.com, http://composite.about.com/library/glossary/g/bldef-g2440.htm

[37] Fibre Reinforced Plastic, “Types of Glass Fibre”, The Fibre Reinforced Plastic & Composite Technology Resource Centre, http://www.fibre-reinforced-plastic.com/2010/04/types-of-glass-fibre.html

[38] Quilter, A., “Composites in Aerospace Applications”, ESDU International, http://france.ihs.com/NR/rdonlyres/AEF9A38E-56C3-4264-980C-D8D6980A4C84/0/444.pdf

[39] Blumenberg, B., “Airbus A350: Composites on Trial Part I”, Environmental Graffiti, http://www.environmentalgraffiti.com/sciencetech/airbus-composites-trial-part-one/13300

[40] Chunna, L.X.T.W.X., “Composite materials in the application of Airbus aircraft”, Aviation News, http://www.auairs.com/html/94953_Composite-materials-in-the-application-of-Airbus-aircraft.html

[41] Aerospace Technology, “Airbus A320 Single Aisle Medium Range Airliner, Europe”, http://www.aerospace-technology.com/projects/a320/

[42] Wikipedia, “Airbus A380”, Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Airbus_A380

[43] Aero Magazine, “Boeing 787: From the ground up”, The Boeing Company, http://www.boeing.com/commercial/aeromagazine/articles/qtr_4_06/article_04_2.html

[44] Wikipedia, “Boeing 787 Dreamliner”, Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/Boeing_787_Dreamliner

[45] FlexForm Technologies, “Where are natural fiber composites used in automobiles?”, FlexForm Technologies, http://www.naturalfibersforautomotive.com/?cat=9

[46] Montgomery, R., “Corvette’s Unique Body”, America’s Sports Car, http://www.moldedfiberglass.com/library/news/news_MFGCorp_CorvetteCelebration.pdf

[47] Thick Engineering, “Composite materials in automotive engineering”, Think Engineering, Engineering Blog and Technologies, http://www.thinkengineering.net/104/composite-materials-in-automotive-engineering/automotive-engineering/

[48] Nanni, A., "International Research on Advanced Composites in Construction (IRACC-96)", Final Report to US National Science Foundation, August 1996

[49] Benjamin, M., and Tang, P.E., “FRP Composites Technology Brings Advantages to the American Bridge Building Industry”, 2nd International Workshop on Structural Composites for Infrastructure Applications, Cairo, Egypt, December 2003, http://www.fhwa.dot.gov/bridge/frp/egypt.pdf

[50] Lightweight Structures B.V., “Composite pedestrian bridge in Delft”, Lightweight Structures B.V., http://www.lightweight-structures.com/composite-pedestrian-bridge-in-delft/index.html

[51] Reinforced Plastics.com, “Madrid sees installation of carbon composite pedestrian bridge”, Reinforced Plastics.com, April 2011, http://www.reinforcedplastics.com/view/17065/madrid-sees-installation-of-carbon-composite-pedestrian-bridge-/

[52] Summerscales, J., “Fibre-reinforced polymer composite bridges in Europe”, Advanced Composites Manufacturing Centre, The University of Plymouth, http://www.tech.plym.ac.uk/sme/composites/bridges.htm

[53] El-Salakawy, E., Kassem, C., and Benmokrane, B., “Field Application of FRP Composite Bars as Reinforcement for Bridge Decks”, 4th Structural Specialty Conference of the Canadian Society for Civil Engineering, Montreal, Canada, June 2002, http://www.pultrall.com/pdf/PaperWottonBridge.pdf

[54] Black, S., “How Are Composite Bridges Performing?”, Composites Technology, December 2003, http://www.compositesworld.com/articles/how-are-composite-bridges-performing

[55] Griffiths, J.R., “Plastic Highway Bridges”, CSA, November 2000, http://www.csa.com/discoveryguides/bridge/overview.php

[56] “Composites seismic retrofit of the Arroyo Seco bridge completed successfully”, Advanced Materials & Composites News, May 2000, Vol. 22, No. 9, pp 1–3

[57] BRE and Trend 2000 Ltd, “Composite Swimming Pools”, Polymer Composites as Construction Materials, http://projects.bre.co.uk/composites/pdf/AP19_Swimming%20Pools.pdf

[58] Fares, M.E., Youssif, Y.G., and Alamir, A.E., “Optimal design and control of composite laminated plates with various boundary conditions using various plate theories”, Composite Structures, 2002, Vol. 56, pp. 1 – 12

[59] Bakir, B. and Hashem, H., “Effect of fibre orientation for fibre glass reinforced composite material on mechanical properties”, International Journal of Mining, Metallurgy & Mechanical Engineering, 2013, Vol. 1, No. 5, pp. 341 – 345

[60] Kim, C.W., Hwang, W., Park, H.C. and Han, K.S., “Stacking sequence optimization of laminated plates”, Composite Structures, 1997, Vol. 39, No. 3-4, pp. 283 – 288

[61] Kim, J.S., Kim, C.G., Hong, C.S., and Hahn, H.T., “Development of concurrent engineering system for design of composite structures”, Composite Structures, 2000, Vol. 50, pp 297 – 309

[62] Kim, J.S., Kim, N.P., and Han, S.H., “Optimal stiffness design of composite laminates for a train carbody by an expert system and enumeration method”, Composite Structures, 2005, Vol. 68, pp. 147 – 156

[63] Kim, J.S., “Development of a user-friendly expert system for composite laminate design”, Composite Structures, 2007, Vol. 79, pp. 76 – 83

[64] Park, J.H., Hwang, J.H., Lee, C.S. and Hwang, W., “Stacking sequence design of composite laminates for maximum strength using genetic algorithms”, Composite Structures, 2001, Vol. 52, pp. 217 – 231

[65] Oh, J.H., Kim, Y.G. and Lee, D.G., “Optimum bolted joints for hybrid composite materials”, Composite Structures, 1997, Vol. 38, No. 1-4, pp. 329 – 341

[66] Adali, S., Richter, A. and Verijenko, V.E., “Optimization of shear-deformable laminated plates under buckling and strength criteria”, Composite Structures, 1997, Vol. 39, No. 3-4, pp. 167 – 178

[67] Keller, D., “Optimization of ply angles in laminated composite structures by a hybrid, asynchronous, parallel evolutionary algorithm”, Composite Structures, 2010, Vol. 92, pp. 2781 – 2790

[68] Khandan, R., Noroozi, S., Sewell, P., Vinney, J. and Koohgilani, M., “Optimum design of fibre orientation in composite laminate plates for out-plane stresses”, Advances in Materials Science and Engineering, 2012, Hindawi Publishing Corporation

[69] Lee, D.S., Morillo, C., Bugeda, G., Oller, S. and Onate, E., “Multilayered composite structure design optimisation using distributed/parallel multi-objective evolutionary algorithms”, Composite Structures, 2012, Vol. 94, pp. 1087 – 1096

[70] Krikanov, A.A., “Composite pressure vessels with higher stiffness”, Composite Structures, 2000, Vol. 48, pp. 119 – 127

[71] Walker, M. and Smith, R.E., “A technique for the multiobjective optimisation of laminated composite structures using genetic algorithms and finite element analysis”, Composite Structures, 2003, Vol. 62, pp. 123 – 128

[72] Paluch, B., Grédiac, M., and Faye, A., “Combining a finite element programme and a genetic algorithm to optimize composite structures with variable thickness”, Composite Structures, 2008, Vol. 83, pp. 284 – 294

[73] Bruyneel, M., “A general and effective approach for the optimal design of fiber reinforced composite structures”, Composites Science and Technology, 2006, Vol. 66, pp. 1303 – 1314

[74] Omkar, S.N., Khandelwal, R., Yathindra, S., Naik, G.N., and Gopalakrishnan, S., “Artificial immune system for multi-objective design optimization of composite structures”, Engineering Applications of Artificial Intelligence, 2008, Vol. 21, pp. 1416 – 1429

[75] Omkar, S.N., Khandelwal, R., Ananth, T.V.S., Naik, G.N., and Gopalakrishnan, S., “Quantum behaved Particle Swarm Optimization (QPSO) for multi-objective design optimization of composite structures”, Expert Systems with Applications, 2009, Vol. 36, pp. 11312 – 11322

[76] Omkar, S.N., Senthilnath, J., Khandelwal, R., Naik, G.N., and Gopalakrishnan, S., “Artificial Bee Colony (ABC) for multi-objective design optimization of composite structures”, Applied Soft Computing, 2011, Vol. 11, pp. 489 – 499

[77] Akbulut, M., and Sonmez, F.O., “Design optimization of laminated composites using a new variant of simulated annealing”, Computers and Structures, 2011, Vol. 89, pp. 1712 – 1724

[78] Farshi, B., and Rabiei, R., “Optimum design of composite laminates for frequency constraints”, Composite Structures, 2007, Vol. 81, pp. 587 – 597

[79] Farshi, B., and Herasati, S., “Optimum weight design of fiber composite plates in flexure based on a two level strategy”, Composite Structures, 2006, Vol. 73, pp. 495 – 504

[80] Ghiasi, H., Pasini, D., and Lessard, L., "Optimum stacking sequence design of composite materials Part I: Constant stiffness design", Composite Structures, 2009, Vol. 90, pp.1 – 11

[81] Evans, D.O., Vaniglia, M.M., and Hopkins, P.C., "Fiber placement process study", 34th International SAMPE Symposium, May 1989, pp. 1822 – 1833

[82] Tatting, B.F., and Gürdal, Z., "Design and manufacture of elastically tailored tow placed plates", Technical Report CR-211919, NASA, August 2002

[83] Ghiasi, H., Fayazbakhsh, K., Pasini, D., and Lessard, L., "Optimum stacking sequence design of composite materials Part II: Variable stiffness design", Composite Structures, 2010, Vol. 93, pp.1 – 13

[84] Setoodeh, S., Abdalla, M.M., and Gürdal, Z., "Design of variable–stiffness laminates using lamination parameters", Composites: Part B, 2006, Vol. 37, pp. 301 – 309

[85] Setoodeh, S., Gürdal, A., and Watson, L.T., "Design of variable-stiffness composite layers using cellular automata", Computer Methods in Applied Mechanics and Engineering, 2006, Vol. 195, pp. 836 – 851

[86] Setoodeh, S., Abdalla, M.M., IJsselmuiden, S.T., and Gürdal, Z., "Design of variable-stiffness composite panels for maximum buckling load", Composite Structures, 2009, Vol. 87, pp. 109 – 117

[87] Gürdal, Z., and Olmedo, R., "In-plane response of laminates with spatially varying fibre orientations: Variable stiffness concept", American Institute of Aeronautics and Astronautics Journal, 1993, Vol. 31, No. 4, pp.751 – 758

[88] Gürdal, Z., Tatting, B.F., and Wu, C.K., "Variable stiffness composite panels: Effects of stiffness variation on the in-plane and buckling response", Composites: Part A, 2008, Vol. 39, pp. 911 – 922

[89] Lopes, C.S., Camanho, P.P., Gürdal, Z., and Tatting, B.F., "Progressive failure analysis of tow-placed, variable-stiffness composite panels", International Journal of Solids and Structures, 2007, Vol. 44, pp. 8493 – 8516

[90] Lopes, C.S., Gürdal, Z., and Camanho, P.P., "Variable-stiffness composite panels: Buckling and first-ply failure improvements over straight-fibre laminates", Computers and Structures, 2008, Vol. 86, pp. 897 – 907

[91] Lopes, C.S., Gürdal, Z., and Camanho, P.P., "Tailoring for strength of composite steered-fibre panels with cutouts", Composites: Part A, 2010, Vol. 41, pp. 1760 – 1767

[92] Wu, Z., Weaver, P.M., Raju, G. and Kim, B.C., "Buckling analysis and optimisation of variable angle tow composite plates", Thin-Walled Structures, 2012, Vol. 60, pp. 163 – 172

[93] Wu, Z., Weaver, P.M. and Raju, G., "Postbuckling optimisation of variable angle tow composite plates", Composite Structures, 2013, Vol. 103, pp. 34 – 42

[94] White, S.C., Raju, G. and Weaver, P.M., "Initial post-buckling of variable-stiffness curved panels", Journal of the Mechanics and Physics of Solids, 2014, Vol. 71, pp. 132 – 155

[95] Marouene, A., Boukhili, R., Chen, J. and Yousefpour A., "Buckling behavior of variable-stiffness composite laminates manufactured by the tow-drop method", Composite Structures, 2016, Vol. 139, pp. 243 – 253

[96] Zucco, G., Groh, R.M.J., Madeo, A. and Weaver, P.M., "Mixed shell element for static and buckling analysis of variable angle tow composite plates", Composite Structures, 2016, Vol. 152, pp. 324 – 338

[97] Coburn, B.H. and Weaver, P.M., "Buckling analysis, design and optimisation of variable-stiffness sandwich panels", International Journal of Solids and Structures, 2016, Vol. 96, pp. 217 – 228

[98] Irisarri, F.-X., Peeters, D.M.J. and Abdalla, M.M., "Optimisation of ply drop order in variable stiffness laminates", Composite Structures, 2016, Vol. 152, pp. 791 – 799

[99] Peeters, D.M.J., Irisarri, F.-X. and Abdalla, M.M., "Optimizing the ply dropping order in variable stiffness, variable thickness laminates using stacking sequence tables", ECCM17, Jun 2016, Munich, Germany

[100] Blom, A.W., Stickler, P.B., and Gürdal, Z., "Optimization of a composite cylinder under bending by tailoring stiffness properties in circumferential direction", Composites: Part B, 2010, Vol. 41, pp. 157 – 165

[101] Rouhi, M., Ghayoor, H., Hoa, S.V. and Hojjati, M., "Effect of structural parameters on design of variable-stiffness composite cylinders made by fiber steering", Composite Structures, 2014, Vol. 118, pp. 472 – 481

[102] Rouhi, M., Ghayoor, H., Hoa, S.V. and Hojjati, M., "Multi-objective design optimization of variable stiffness composite cylinders", Composites: Part B, 2015, Vol. 69, pp. 249 – 255

[103] Peeters, D.M.J., Hesse, S. and Abdalla, M.M., "Stacking sequence optimisation of variable stiffness laminates with manufacturing constraints", Composite Structures, 2015, Vol. 125, pp. 596 – 604

[104] Abdalla, M.M., Setoodeh, S., and Gürdal, Z., "Design of variable stiffness composite panels for maximum fundamental frequency using lamination parameters", Composite Structures, 2007, Vol. 81, pp. 283 – 291

[105] Akhavan, H., and Ribeiro, P., "Natural modes of vibration of variable stiffness composite laminates with curvilinear fibers", Composite Structures, 2011, Vol. 93, pp. 3040 – 3047

[106] Blom, A.W., Setoodeh, S., Hol, J.M.A.M., and Gürdal, Z., "Design of variable-stiffness conical shells for maximum fundamental eigenfrequency", Computers and Structures, 2008, Vol. 86, pp. 870 – 878

[107] Akbarzadeh, A.H., Nik, M.A. and Pasini, D., "Vibration responses and suppression of variable stiffness laminates with optimally steered fibers and magnetostrictive layers", Composites: Part B, 2016, Vol. 91, pp. 315 – 326

[108] Vescovini, R. and Dozio, L., "A variable-kinematic model for variable stiffness plates: Vibration and buckling analysis", Composite Structures, 2016, Vol. 142, pp. 15 – 26

[109] Tan, P. and Nie, G.J., "Free and forced vibration of variable stiffness composite annular thin plates with elastically restrained edges", Composite Structures, 2016, Vol. 149, pp. 398 – 407

[110] Khani, A., IJsselmuiden, S.T., Abdalla, M.M., and Gürdal, Z., "Design of variable stiffness panels for maximum strength using lamination parameters", Composites: Part B, 2011, Vol. 42, pp. 546 – 552

[111] Shirinzadeh, B., Alici, G., Foong, C.W., and Cassidy, G., "Fabrication process of open surfaces by robotic fibre placement", Robotics and Computer-Integrated Manufacturing, 2004, Vol. 20, pp 17–28

[112] Precisioneering DKG, "Glossary of Laminating Methods", http://www.precisioneering.com/glossary_laminating_methods.htm

[113] Summerscales, J., "Composites Design and Manufacture", Advanced Composites Manufacturing Centre, The University of Plymouth, http://www.tech.plym.ac.uk/sme/MATS324/MATS324C7%20infusion.htm

[114] Energetx Composites, "VARTM", Energetx Composites, LLC, http://www.energetxcomposites.com/vartm.html

[115] Shirinzadeh, B., Cassidy, G., Oetomo, D., Alici, G., and Ang, M.H. Jr., "Trajectory generation for open-contoured structures in robotic fibre placement", Robotics and Computer-Integrated Manufacturing, 2007, Vol. 23, pp 380–394

[116] Bullock, F., Kowalski, S., and Young, R., "Automated prepreg tow-placement for composite structures", 35th International SAMPE Symposium, 1990, Vol. 35, pp. 734 – 743

[117] Addax, "Advanced Composites – Manufacturing: Filament Winding Process", Addax.com, http://www.addax.com/technology/filament_winding.html

[118] Etamax Engineering, "Filament winding", Etamax Engineering, http://www.etamax.com.au/filament_winding.html

[119] Mondo, J.A., Pasanen, M.J., Langone, R.J., and Martin, J.P., "Advances in automated fibre placement of aircraft structures", Automated Dynamics Corporation, Schenectady, New York, USA

[120] Black, S., "Getting To Know "Black Aluminum": An introduction to CFRP", Modern Machine Shop, 2008, http://www.mmsonline.com/articles/getting-to-know-black-aluminum

[121] Coriolis Composites, "The use of robots for fiber placement", Coriolis Composites, http://www.coriolis-composites.com/technologie.php

[122] Wakeman, M.D., Hagstrand, P.O., Bonjour, F., Bourban, P.E., and Manson, J.A.E., "Robotic tow placement for local reinforcement of glass mat

thermoplastics (GMTs)", Composites Part A: Applied Science and Manufacturing, 2002, 33 pp 1199 – 1208

[123] Favaloro, M., and Hauber, D., "Process and design considerations for the automated fibre placement process", Automated Dynamics Corporation, Schenectady, New York, USA

[124] Martin, J.P., Langone, R.J., Pasanen, M.J., and Mondo, J.A., "Cost-effective, automated equipment for advanced composite structure development and production", Automated Dynamics Corporation, Schenectady, New York, USA

[125] Pasanen, M.J., Martin, J.P., Langone, R.J., and Mondo, J.A., "Advanced composite fibre placement: Process to application", Automated Dynamics Corporation, Schenectady, New York, USA

[126] Grant, C., and Martin, J., "Automated processing technology for composites: Current status and vision for the future", Automated Dynamics Corporation, Schenectady, New York, USA

[127] Langone, R.J., Pasanen, M., Mondo, J.A., and Martin, J., "Continued development of automated, in-situ processing for thermoplastic composite structures and components", Automated Dynamics Corporation, Schenectady, New York, USA

[128] Mondo, J.A., Hauber, D., Langone, R., and Quinn, L., "High Speed Processing of Thermoplastic Composites for Oilfield Pipe and Tubular Applications", Automated Dynamics Corporation, Schenectady, New York, USA

[129] Aerospace gallery, Automated Dynamics Corporation, Schenectady, New York, USA, http://www.automateddynamics.com/wp-content/uploads/2013/11

[130] Johnston, N.J., Towell, T.W., Marchello, J.M., and Grenoble, R.W., "Automated fabrication of high performance composites: An overview of research at the Langley Research Centre", NASA Langley Research Centre, Hampton, VA, USA

[131] Brandt, M.R., and Reeve, S.R., "Directed Fibre pre-form case studies", Composites 2001 Convention and Trade Show, Composites Fabricators Association, October 3 – 6, 2001

[132] Kaw, A.K., "Mechanics of composite materials", CRC Press LLC, Boca Raton, Florida, U.S.A., 1997

[133] Herakovich, C.T., “Mechanics of fibrous composites”, John Wiley & Sons Inc., New York, U.S.A., 1998

[134] Hyer, M.W., “Stress analysis of fiber-reinforced composite materials”, WCB/McGraw-Hill, U.S.A., 1998

[135] Wikipedia, “Computer programming”, Wikipedia, the free encyclopaedia, https://en.wikipedia.org/wiki/Computer_programming

[136] Wikipedia, “MATLAB”, Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/MATLAB

[137] Dukkipati, R.V., “MATLAB for Mechanical Engineers”, New Age International (P) Limited, New Delhi, 2008

[138] Gilat, A., “MATLAB: An Introduction with Applications”, John Wiley & Sons Inc., USA, 2008

[139] Hahn, B.D., and Valentine, D.T., “Essential MATLAB for Engineers and Scientists”, Butterworth-Heinemann, Italy, 2007

[140] Wikipedia, “Finite element method”, Wikipedia, the free encyclopaedia, https://en.wikipedia.org/wiki/Finite_element_method

[141] Center for Aerospace Structures, “FEM Modeling: Introduction”, Center for Aerospace Structures, University of Colorado, Boulder, http://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/IFEM.Ch06.d/IFEM.Ch06.Slides.d/IFEM.Ch06.Slides.pdf

[142] Lin, L, “Introduction to Finite Element Modeling”, Department of Mechanical Engineering, University of California at Berkeley, http://www.me.berkeley.edu/~lwlin/me128/FEMNotes.pdf

[143] MSC Software Corporation, “Patran”, MSC Software Corporation, 2015, http://www.mscsoftware.com/product/patran

[144] MSC Software Corporation, “MSC Nastran”, MSC Software Corporation, 2015, http://www.mscsoftware.com/product/msc-nastran

[145] Wikipedia, “Command-line interface”, Wikipedia, the free encyclopaedia, https://en.wikipedia.org/wiki/Command-line_interface

[146] Wikipedia, “Graphical user interface”, Wikipedia, the free encyclopaedia, https://en.wikipedia.org/wiki/Graphical_user_interface

[147] Kazadi, S., “A Specialised End-Effector for the Manufacture of Complex Shaped Dry Pre-Pregs”, Masters Dissertation, Durban University of Technology, 2012

[148] Fatih University, “Laboratories – Control Systems Laboratory”, Fatih University – Electrical and Electronics Engineering, http://dis.fatih.edu.tr/store/images/el17_W8VtF4ck.jpg

[149] Mitsubishi Electric, “MELFA Industrial Robots”, Mitsubishi Electric – Factory Automation, http://www.mitsubishi-automation.com

APPENDIX A: DESIGN OF A FIBRE REINFORCED COMPOSITE LAMINATE VIA MANUAL CALCULATIONS

This appendix demonstrates the manual calculations involved in designing a fibre reinforced composite laminate. The design procedure employed here was laid out in Chapter 4. The calculations below are for the example in Chapter 4 from Kaw [132]. The laminate material was graphite/epoxy and it consisted of three layers with fibre orientation angles of 0°, 30° and -45°, and layer thicknesses of 5mm each. The material constants, material limits and loading conditions were given in Table 4.1 and is shown here for convenience in Table A.1.

Table A.1: Material properties and loading conditions for manually calculated example

Material Constants				Material Limits					Loading	
E_1 (GPa)	E_2 (GPa)	G_{12} (GPa)	ν_{12}	$(\sigma_1^T)_{ult}$ (MPa)	$(\sigma_1^C)_{ult}$ (MPa)	$(\sigma_2^T)_{ult}$ (MPa)	$(\sigma_2^C)_{ult}$ (MPa)	$(\tau_{12})_{ult}$ (MPa)	Forces	Moments
181	10.3	7.17	0.28	1500	1500	40	246	68	1000	0
									1000	0
									0	0

The first step in the design procedure outlined in Chapter 4 is to determine the reduced stiffness matrix [Q] for the graphite/epoxy material using equation (3.6).

$$Q_{11} = \frac{E_1}{1-\nu_{12}\nu_{21}} = \frac{181}{1-(0.28)(0.016)} = 181.8\,GPa$$

$$Q_{12} = \frac{\nu_{12}E_2}{1-\nu_{12}\nu_{21}} = \frac{(0.28)(10.3)}{1-(0.28)(0.016)} = 2.897\,GPa$$

$$Q_{22} = \frac{E_2}{1-\nu_{12}\nu_{21}} = \frac{10.3}{1-(0.28)(0.016)} = 10.35\,GPa$$

$$Q_{66} = G_{12} = 7.17\,GPa$$

Hence [Q] is:

$$[Q] = \begin{bmatrix} Q_{11} & Q_{12} & 0 \\ Q_{12} & Q_{22} & 0 \\ 0 & 0 & Q_{66} \end{bmatrix} = \begin{bmatrix} 181.8 & 2.897 & 0 \\ 2.897 & 10.35 & 0 \\ 0 & 0 & 7.17 \end{bmatrix} (10^9)\, Pa$$

The next step is to determine the transformation matrix [T] for each orientation angle of the fibre layup via equation (3.9).

$$[T]_0 = \begin{bmatrix} c^2 & s^2 & 2sc \\ s^2 & c^2 & -2sc \\ -sc & sc & c^2 - s^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$[T]_{30} = \begin{bmatrix} c^2 & s^2 & 2sc \\ s^2 & c^2 & -2sc \\ -sc & sc & c^2 - s^2 \end{bmatrix} = \begin{bmatrix} 0.750 & 0.250 & 0.866 \\ 0.250 & 0.750 & -0.866 \\ -0.433 & 0.433 & 0.500 \end{bmatrix}$$

$$[T]_{-45} = \begin{bmatrix} c^2 & s^2 & 2sc \\ s^2 & c^2 & -2sc \\ -sc & sc & c^2 - s^2 \end{bmatrix} = \begin{bmatrix} 0.500 & 0.500 & -1 \\ 0.500 & 0.500 & 1 \\ 0.500 & -0.500 & 0 \end{bmatrix}$$

Step three in the procedure is to calculate the $[\overline{Q}]$ matrix for each layer using equation (3.12).

$$[\overline{Q}]_0 = [T]^{-1}[Q][R][T][R]^{-1}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 181.8 & 2.897 & 0 \\ 2.897 & 10.35 & 0 \\ 0 & 0 & 7.17 \end{bmatrix} (10^9) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} 181.8 & 2.897 & 0 \\ 2.897 & 10.35 & 0 \\ 0 & 0 & 7.17 \end{bmatrix} (10^9)\, Pa$$

$$\left[\overline{Q}\right]_{30} = [T]^{-1}[Q][R][T][R]^{-1}$$

$$= \begin{bmatrix} 0.750 & 0.250 & 0.866 \\ 0.250 & 0.750 & -0.866 \\ -0.433 & 0.433 & 0.500 \end{bmatrix}^{-1} \begin{bmatrix} 181.8 & 2.897 & 0 \\ 2.897 & 10.35 & 0 \\ 0 & 0 & 7.17 \end{bmatrix} \left(10^9\right) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \cdots$$

$$\begin{bmatrix} 0.750 & 0.250 & 0.866 \\ 0.250 & 0.750 & -0.866 \\ -0.433 & 0.433 & 0.500 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} 109.40 & 32.46 & 54.19 \\ 32.46 & 23.65 & 20.05 \\ 54.19 & 20.05 & 36.74 \end{bmatrix} \left(10^9\right) Pa$$

$$\left[\overline{Q}\right]_{-45} = [T]^{-1}[Q][R][T][R]^{-1}$$

$$= \begin{bmatrix} 0.500 & 0.500 & -1 \\ 0.500 & 0.500 & 1 \\ 0.500 & -0.500 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 181.8 & 2.897 & 0 \\ 2.897 & 10.35 & 0 \\ 0 & 0 & 7.17 \end{bmatrix} \left(10^9\right) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \cdots$$

$$\begin{bmatrix} 0.500 & 0.500 & -1 \\ 0.500 & 0.500 & 1 \\ 0.500 & -0.500 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} 56.66 & 42.32 & -42.87 \\ 42.32 & 56.66 & -42.87 \\ -42.87 & -42.87 & 46.59 \end{bmatrix} \left(10^9\right) Pa$$

Next (Step 4), the location of the top and bottom surfaces of each layer (h_k) has to be determined using equation (3.19).

$$h = \sum_{k=1}^{n} t_k = 0.005 + 0.005 + 0.005 = 0.015\, m$$

$$h_0 = -\frac{h}{2} = -\frac{0.015}{2} = -0.0075\, m$$

$$h_1 = h_0 + t_1 = -0.0075 + 0.005 = -0.0025\ m$$
$$h_2 = h_1 + t_2 = -0.0025 + 0.005 = 0.0025\ m$$
$$h_3 = h_2 + t_3 = 0.0025 + 0.005 = 0.0075\ m$$

The following step (Step 5) requires that the [A], [B], and [D] matrices be calculated using equations (3.16), (3.17), and (3.18), respectively.

$$A_{ij} = \sum_{k=1}^{3} \left[\left(\overline{Q_{ij}}\right)\right]_k \left(h_k - h_{k-1}\right)$$

$$[A] = \left[\overline{Q}\right]_1 \left(h_1 - h_0\right) + \left[\overline{Q}\right]_2 \left(h_2 - h_1\right) + \left[\overline{Q}\right]_3 \left(h_3 - h_2\right)$$

$$= \begin{bmatrix} 181.8 & 2.897 & 0 \\ 2.897 & 10.35 & 0 \\ 0 & 0 & 7.17 \end{bmatrix} \left(10^9\right)\left(\left(-0.0025\right) - \left(-0.0075\right)\right) + \cdots$$

$$\begin{bmatrix} 109.40 & 32.46 & 54.19 \\ 32.46 & 23.65 & 20.05 \\ 54.19 & 20.05 & 36.74 \end{bmatrix} \left(10^9\right)\left(\left(0.0025\right) - \left(-0.0025\right)\right) + \cdots$$

$$\begin{bmatrix} 56.66 & 42.32 & -42.87 \\ 42.32 & 56.66 & -42.87 \\ -42.87 & -42.87 & 46.59 \end{bmatrix} \left(10^9\right)\left(\left(0.0075\right) - \left(0.0025\right)\right)$$

$$= \begin{bmatrix} 1.739\left(10^9\right) & 3.884\left(10^8\right) & 5.663\left(10^7\right) \\ 3.884\left(10^8\right) & 4.533\left(10^8\right) & -1.141\left(10^8\right) \\ 5.663\left(10^7\right) & -1.141\left(10^8\right) & 4.525\left(10^8\right) \end{bmatrix} Pa.m$$

$$B_{ij} = \frac{1}{2}\sum_{k=1}^{3} \left[\left(\overline{Q_{ij}}\right)\right]_k \left(h_k^2 - h_{k-1}^2\right)$$

$$[B] = \frac{1}{2}\left[\overline{Q}\right]_1 \left(h_1^2 - h_0^2\right) + \frac{1}{2}\left[\overline{Q}\right]_2 \left(h_2^2 - h_1^2\right) + \frac{1}{2}\left[\overline{Q}\right]_3 \left(h_3^2 - h_2^2\right)$$

$$= \frac{1}{2}\begin{bmatrix} 181.8 & 2.897 & 0 \\ 2.897 & 10.35 & 0 \\ 0 & 0 & 7.17 \end{bmatrix}(10^9)\left((-0.0025)^2-(-0.007.)^2\right) + \cdots$$

$$\frac{1}{2}\begin{bmatrix} 109.40 & 32.46 & 54.19 \\ 32.46 & 23.65 & 20.05 \\ 54.19 & 20.05 & 36.74 \end{bmatrix}(10^9)\left((0.0025)^2-(-0.0025)^2\right) + \cdots$$

$$\frac{1}{2}\begin{bmatrix} 56.66 & 42.32 & -42.87 \\ 42.32 & 56.66 & -42.87 \\ -42.87 & -42.87 & 46.59 \end{bmatrix}(10^9)\left((0.0075)^2-(0.0025)^2\right)$$

$$= \begin{bmatrix} -3.129(10^6) & 9.855(10^5) & -1.072(10^6) \\ 9.855(10^5) & 1.158(10^6) & -1.072(10^6) \\ -1.072(10^6) & -1.072(10^6) & 9.855(10^5) \end{bmatrix} Pa.m^2$$

$$D_{ij} = \frac{1}{3}\sum_{k=1}^{3}\left[\left(\overline{Q_{ij}}\right)\right]_k \left(h_k^3 - h_{k-1}^3\right)$$

$$[D] = \frac{1}{3}\left[\overline{Q}\right]_1\left(h_1^3-h_0^3\right) + \frac{1}{3}\left[\overline{Q}\right]_2\left(h_2^3-h_1^3\right) + \frac{1}{3}\left[\overline{Q}\right]_3\left(h_3^3-h_2^3\right)$$

$$= \frac{1}{3}\begin{bmatrix} 181.8 & 2.897 & 0 \\ 2.897 & 10.35 & 0 \\ 0 & 0 & 7.17 \end{bmatrix}(10^9)\left((-0.0025)^3-(-0.0075)^3\right) + \cdots$$

$$\frac{1}{3}\begin{bmatrix} 109.40 & 32.46 & 54.19 \\ 32.46 & 23.65 & 20.05 \\ 54.19 & 20.05 & 36.74 \end{bmatrix}(10^9)\left((0.0025)^3-(-0.0025)^3\right) + \cdots$$

$$\frac{1}{3}\begin{bmatrix} 56.66 & 42.32 & -42.87 \\ 42.32 & 56.66 & -42.87 \\ -42.87 & -42.87 & 46.59 \end{bmatrix}(10^9)\left((0.0075)^3-(0.0025)^3\right)$$

$$= \begin{bmatrix} 3.343(10^4) & 6.461(10^3) & -5.240(10^3) \\ 6.461(10^3) & 9.320(10^3) & -5.596(10^3) \\ -5.240(10^3) & -5.596(10^3) & 7.663(10^3) \end{bmatrix} Pa.m^3$$

In Step 6, the calculated [A], [B] and [D] matrices, and the applied loading conditions must be substituted into equation (3.20) to determine the mid-plane strains (ε^0) and curvatures (κ).

$$\begin{bmatrix} N_x \\ N_y \\ N_{xy} \\ M_x \\ M_y \\ M_{xy} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{16} & B_{11} & B_{12} & B_{16} \\ A_{12} & A_{22} & A_{26} & B_{12} & B_{22} & B_{26} \\ A_{16} & A_{26} & A_{66} & B_{16} & B_{26} & B_{66} \\ B_{11} & B_{12} & B_{16} & D_{11} & D_{12} & D_{16} \\ B_{12} & B_{22} & B_{26} & D_{12} & D_{22} & D_{26} \\ B_{16} & B_{26} & B_{66} & D_{16} & D_{26} & D_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \\ \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix}$$

$$\begin{bmatrix} 1000 \\ 1000 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1.739(10^9) & 3.884(10^8) & 5.663(10^7) & -3.129(10^6) & 9.855(10^5) & -1.072(10^6) \\ 3.884(10^8) & 4.533(10^8) & -1.141(10^8) & 9.855(10^5) & 1.158(10^6) & -1.072(10^6) \\ 5.663(10^7) & -1.141(10^8) & 4.525(10^8) & -1.072(10^6) & -1.072(10^6) & 9.855(10^5) \\ -3.129(10^6) & 9.855(10^5) & -1.072(10^6) & 3.343(10^4) & 6.461(10^3) & -5.240(10^3) \\ 9.855(10^5) & 1.158(10^6) & -1.072(10^6) & 6.461(10^3) & 9.320(10^3) & -5.596(10^3) \\ -1.072(10^6) & -1.072(10^6) & 9.855(10^5) & -5.240(10^3) & -5.596(10^3) & 7.663(10^3) \end{bmatrix} \begin{bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \\ \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix}$$

$$\begin{bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \\ \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix} = \begin{bmatrix} 3.123(10^{-7}) \\ 3.492(10^{-6}) \\ -7.598(10^{-7}) \\ 2.971(10^{-5}) \\ -3.285(10^{-4}) \\ 4.101(10^{-4}) \end{bmatrix} \frac{m/m}{1/m}$$

The next three steps (Steps 7 to 9) involve the calculation of the global and local stresses and strains. However, the point in the laminate where this is required needs to

be decided upon. The distance from this point to the mid-plane is represented by the variable z in equation (3.15). In Kaw [132] it was decided to find the stresses and strains at the top, middle and bottom of each layer. Therefore the z values for the top, middle and bottom of each layer are shown below.

$$
\begin{aligned}
z_{1,Top} &= h_0 = -0.0075\ m \\
z_{1,Mid} &= h_0 + t_1/2 = -0.0050\ m \\
z_{1,Bot} &= h_1 = -0.0025\ m
\end{aligned}
$$

$$
\begin{aligned}
z_{2,Top} &= h_1 = -0.0025\ m \\
z_{2,Mid} &= h_1 + t_2/2 = 0\ m \\
z_{2,Bot} &= h_2 = 0.0025\ m
\end{aligned}
$$

$$
\begin{aligned}
z_{3,Top} &= h_2 = 0.0025\ m \\
z_{3,Mid} &= h_2 + t_3/2 = 0.0050\ m \\
z_{3,Bot} &= h_3 = 0.0075\ m
\end{aligned}
$$

Now that the required z values are computed, Step 7 may be executed to determine the global strains via equation (3.15). Below is the calculation for the global strains at the top of the 30° layer where $z = z_{2,\,Top}$.

$$
\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix}_{30^\circ,Top} = \begin{bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{bmatrix} + z_{2,Top} \begin{bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{bmatrix}
$$

$$
= \begin{bmatrix} 3.123(10^{-7}) \\ 3.492(10^{-6}) \\ -7.598(10^{-7}) \end{bmatrix} + (-0.0025) \begin{bmatrix} 2.971(10^{-5}) \\ -3.285(10^{-4}) \\ 4.101(10^{-4}) \end{bmatrix}
$$

$$
= \begin{bmatrix} 2.380(10^{-7}) \\ 4.313(10^{-6}) \\ -1.785(10^{-6}) \end{bmatrix} m/m
$$

The global stresses corresponding to the above calculated global strains are found using equation (3.10) in Step 8.

$$\begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix}_{30°,Top} = \begin{bmatrix}\overline{Q}_{11} & \overline{Q}_{12} & \overline{Q}_{16}\\ \overline{Q}_{12} & \overline{Q}_{22} & \overline{Q}_{26}\\ \overline{Q}_{16} & \overline{Q}_{26} & \overline{Q}_{66}\end{bmatrix}\begin{bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{bmatrix}$$

$$= \begin{bmatrix}109.40 & 32.46 & 54.19\\ 32.46 & 23.65 & 20.05\\ 54.19 & 20.05 & 36.74\end{bmatrix}\left(10^9\right)\begin{bmatrix}2.380\left(10^{-7}\right)\\ 4.313\left(10^{-6}\right)\\ -1.785\left(10^{-6}\right)\end{bmatrix}$$

$$= \begin{bmatrix}6.930\left(10^4\right)\\ 7.391\left(10^4\right)\\ 3.381\left(10^4\right)\end{bmatrix} Pa$$

The local strains and stresses (Step 9) at the top of the 30° layer may be determined via equation (3.8).

$$\begin{bmatrix}\varepsilon_1\\ \varepsilon_2\\ \gamma_{12}\end{bmatrix}_{30°,Top} = [R][T]_{30}[R]^{-1}\begin{bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{bmatrix}_{30°,Top}$$

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 2\end{bmatrix}\begin{bmatrix}0.750 & 0.250 & 0.866\\ 0.250 & 0.750 & -0.866\\ -0.433 & 0.433 & 0.500\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 2\end{bmatrix}^{-1}\begin{bmatrix}2.380\left(10^{-7}\right)\\ 4.313\left(10^{-6}\right)\\ -1.785\left(10^{-6}\right)\end{bmatrix}$$

$$= \begin{bmatrix}4.837\left(10^{-7}\right)\\ 4.067\left(10^{-6}\right)\\ 2.636\left(10^{-6}\right)\end{bmatrix} m/m$$

$$\begin{bmatrix}\sigma_1\\ \sigma_2\\ \tau_{12}\end{bmatrix}_{30°,Top} = [T]_{30}\begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix}_{30°,Top}$$

$$= \begin{bmatrix} 0.750 & 0.250 & 0.866 \\ 0.250 & 0.750 & -0.866 \\ -0.433 & 0.433 & 0.500 \end{bmatrix}\begin{bmatrix} 6.930(10^4) \\ 7.391(10^4) \\ 3.381(10^4) \end{bmatrix}$$

$$= \begin{bmatrix} 9.973(10^4) \\ 4.348(10^4) \\ 1.890(10^4) \end{bmatrix} Pa$$

The global and local stresses and strains experienced at the points represented by the other *z* values, calculated above, may be determined by repeating Steps 7 to 9. The stresses and strains calculated above as well as for the other *z* values are shown in Tables 4.2 to 4.5. These computations prove to be cumbersome and time consuming especially when performed manually. However, the calculations are not yet completed.

The local stresses in conjunction with the Tsai-Wu failure theory may be used to determine the *SR* values at the top, middle and bottom of each layer. However, first the parameters for the failure criterion (Step 10) have to be determined using equation (3.22). The ultimate stress values needed for equation (3.22) is given in Table A.1.

$$H_1 = \frac{1}{\left(\sigma_1^T\right)_{ult}} - \frac{1}{\left(\sigma_1^C\right)_{ult}} = \frac{1}{1500} - \frac{1}{1500} = 0\ MPa^{-1}$$

$$H_2 = \frac{1}{\left(\sigma_2^T\right)_{ult}} - \frac{1}{\left(\sigma_2^C\right)_{ult}} = \frac{1}{40} - \frac{1}{246} = 2.0935\left(10^{-2}\right) MPa^{-1}$$

$$H_6 = 0\ MPa^{-1}$$

$$H_{11} = \frac{1}{\left(\sigma_1^T\right)_{ult}\left(\sigma_1^C\right)_{ult}} = \frac{1}{(1500)(1500)} = 4.4444\left(10^{-7}\right) MPa^{-2}$$

$$H_{22} = \frac{1}{\left(\sigma_2^T\right)_{ult}\left(\sigma_2^C\right)_{ult}} = \frac{1}{(40)(246)} = 1.0163\left(10^{-4}\right) MPa^{-2}$$

$$H_{66} = \frac{1}{\left(\tau_{12}\right)_{ult}^2} = \frac{1}{(68)^2} = 2.1626\left(10^{-4}\right) MPa^{-2}$$

$$H_{12} = -\frac{1}{2}\sqrt{\frac{1}{\left(\sigma_1^T\right)_{ult}\left(\sigma_1^C\right)_{ult}\left(\sigma_2^T\right)_{ult}\left(\sigma_2^C\right)_{ult}}} = -\frac{1}{2}\sqrt{\frac{1}{(1500)(1500)(40)(246)}}$$
$$= -3.3603\left(10^{-6}\right) MPa^{-2}$$

Using the parameters calculated above and equation (3.23), the Tsai-Wu failure theory gives *SR* at the top of the 30° layer as (Step 11):

$$(H_1\sigma_1 + H_2\sigma_2 + H_6\tau_{12})SR + (H_{11}\sigma_1^2 + H_{22}\sigma_2^2 + H_{66}\tau_{12}^2 + 2H_{12}\sigma_1\sigma_2)SR^2 < 1$$

$$\left([0][9.973(10^1)] + [2.0935(10^{-2})][4.348(10^1)] + [0][1.890(10^1)]\right) SR + \cdots$$

$$\left(\begin{array}{l} [4.4444(10^{-7})][9.973(10^1)^2] + [1.0163(10^{-4})][4.348(10^1)^2] + \cdots \\ [2.1626(10^{-4})][1.890(10^1)^2] + 2[-3.3603(10^{-6})][9.973(10^1)][4.348(10^1)] \end{array}\right) SR^2 < 1$$

$$[9.1025(10^{-1})]SR + [2.4466(10^{-1})]SR^2 < 1$$

$$[2.4466(10^{-1})]SR^2 + [9.1025(10^{-1})]SR - 1 = 0$$

$$SR = 0.8870 \quad or \quad SR = -4.6076$$

Since *SR* cannot be negative, only the first value is applicable. Step 11 is repeated to determine the *SR* values at the top, middle and bottom for each layer in the laminate. These values are shown in Table A.2 and are exactly the same as those in Figure 4.6.

Table A.2: *SR* values at top, middle and bottom of each layer

Layer No.	Top	Middle	Bottom
1	0.6147	0.7219	0.8713
2	0.8870	1.1399	1.5642
3	1.6547	1.9113	2.0154

Table A.2 shows the *SR* values for the first layer are less than one. This implies that failure would occur and that the designed laminate would be unable to bear the applied loading. The first layer would fail at approximately 61% of the applied load, followed closely by the next two layers as they would now not be able to bear the full applied load.

Clearly, to overcome failure, the input parameters (number of layers, fibre orientation angles, and layer thicknesses) need to be altered. The question arises of which one(s) to change and by what quantity. If one or more of the parameters are altered, then all the above calculations need to be performed from scratch. Thereafter, if failure still occurs, the process needs to be repeated until there is no failure. This could take an enormous amount of time especially due to the uncertainty in altering the input parameters. It may be concluded that manual calculations prove to be extremely time consuming.

APPENDIX B: MATLAB CODE FOR CONSTANT STIFFNESS METHOD

This appendix is a line-by-line examination of the Constant Stiffness Method (CSM) code discussed in Chapter 4. The code consists of a script file, and several function files. The script file is used to manage the input parameters and control the outputs to the screen. The function files contain the code for the computation of the various matrices in the Hooke's Law equations presented in Chapter 3. The code for the Graphical User Interface (GUI) is discussed last. It should be noted that although each line of code presented below is numbered consecutively, each function is a separate file. Further, the code contains comments and this is depicted by "%" at the beginning of the line.

Main script file: Constant_Stiffness_Method

```
%Material data for Constant Stiffness Method
1    N = 4;
2    Mat_Data = zeros(N,17);
3    for a = 1:N
4        Mat_Data(a,:) = [0.7, 181,10.3, 7.17, 0.28, 0.016,...
5                         1500, 1500, 40, 246, 68,...
6                         0.02, 22.50, 0.00, 0.60, 1.62, 1.00];
7    end
8    Ply_Angle = [-71 31 -70 32];
9    Ply_Thick = [1 2 2 1];
10   NM = [1000;1000;0;0;0;0];

11   [GL_LL_S_S, SR_T_W] = Conventional_Method(N, Mat_Data,
     Ply_Angle, Ply_Thick, NM);

%View global and local stresses and strains
12   desn = input('Do you wish to view the global and local stresses
     and strains [Y/N]: ','s');
13   if desn=='Y' || desn=='y'
14       View_Stress_Strain_Menu(N, GL_LL_S_S);
15   end
16   disp(SR_T_W)
17   disp('Complete')
```

In line 1 of the above script file, the number of layers (*N*), present in the composite laminate, is stipulated. A matrix, called *Mat_Data*, which contains *N* rows and 17 columns, is created in line 2. The purpose of this matrix is to store the material data

for each layer of the laminate. This consists of the fibre volume fraction, the five material constants (longitudinal and transverse elastic moduli, shear modulus, major and minor Poisson's ratios), the five material limits (ultimate longitudinal tensile and compressive strengths, ultimate transverse tensile and compressive strengths, ultimate in-plane shear strength), the four coefficients of expansion (longitudinal and transverse coefficients of thermal expansion, longitudinal and transverse coefficients of moisture expansion), specific gravity and the single-layer thickness. The values for the *Mat_Data* matrix are assigned via the coded loop in lines 3 to 7. The data for the fibre orientation angles, layer thicknesses and loading conditions of the laminate are assigned to arrays in lines 8, 9 and 10, respectively.

In line 11 the function called **Conventional_Method**, which is used to control the execution of the functions that determine the global and local stresses and strains, and the strength ratios, is run. The output variables of this function are on the left hand side of the equal sign and the inputs are on the right hand side. The discussion of the code for this function follows. In lines 12 to 15, the user is prompted to choose whether to display the global and local stresses and strains. If a 'y' or 'Y' response is entered, the **View_Stress_Strain_Menu** function in line 14 is executed. The purpose and operation of this function is discussed later. The calculated strength ratios are displayed via the code in line 16, and the code in line 17 displays a message signalling the end of the run.

Function: Conventional_Method

```
18  function [GL_LL_S_S, SR_T_W] = Conventional_Method(N, Mat_Data,
    Ply_Angle, Ply_Thick, NM)

%Material data extraction
19  Mat_Consts = cell (N,1);
20  Mat_Limits = cell (N,1);
21  for a = 1:N
22      Mat_Consts{a} = Mat_Data(a,2:6);
23      Mat_Limits{a} = Mat_Data(a,7:11);
24  end

%Calculates Q,T,Qbar
25  All_Matrix = cell(N,3);
26  for p = 1:N
```

```
27        Q = Q_matrix(Mat_Consts{p,1});
28        [T,Qbar] = TQbar_matrix(Q,Ply_Angle(p));
29        All_Matrix(p,:) = {Q, T, Qbar};
30    end

%Calculate h & z values
31    [h,z] = H_Z_det(N,Ply_Thick);

%Determine A,B,D matrices
32    ABD = ABD_matrix(N,h,All_Matrix(:,3));

%Input forces and calculates Ep0-K vector
33    Ep0_K = inv(ABD)*NM;

%Calculate global and local stresses and strains on each layer
34    [GL_LL_S_S] =
      Stress_Strain(N,z,Ep0_K,All_Matrix(:,2),All_Matrix(:,3));

%Tsai-Wu Failure Theory
35    SR_T_W = zeros(N,3);
36    for u = 1:N
37        for v = 1:3
38            H_Val = Tsai_Wu_Par(Mat_Limits{u});
39            SR_T_W(u,v) = Tsai_Wu(H_Val, GL_LL_S_S{4}{u,v});
40        end
41    end
```

The above function follows the procedure detailed in Chapter 4 and is represented by the flowchart in Figure 4.1. It is used to control the interaction of the various functions with each other. The code in line 18 is a function assignment, and, as mentioned earlier, the outputs are on the left of the equal sign while the right hand side contains the input variables. The inputs include the number of layers (*N*), the material data for each layer (*Mat_Data*), the fibre orientation angle of each layer (*Ply_Angle*), the thickness of each layer (*Ply_Thick*), and the loading conditions (*NM*). There are two outputs, namely, the global and local stresses and strains (*GL_LL_S_S*), and the strength ratios from the Tsai-Wu failure criterion (*SR_T_W*).

The matrix *Mat_Data* contains more data than is required for the matrix calculations that follow. The code in lines 19 to 24 is used to extract the relevant information, that is, the material constants and material limits, and stores these properties using the *Mat_Consts* and *Mat_Limits* arrays.

The next part of the code performs all the necessary matrix algebra. In lines 25 to 30, the [Q], [T] and [$\overline{Q}$] matrices are determined via the **Q_matrix** and **TQbar_matrix** functions. These matrices are stored using the array *All_Matrix* for each layer in the laminate. The *h* and *z* values are determined by the **H_Z_det** function in line 31. All data required to compute the [*A*], [*B*] and [*D*] matrices is now available. These are calculated in line 32 via the **ABD_matrix** function. The mid-plane strains and curvatures are found using the code in line 33.

The first output of the **Conventional_Method** function (*GL_LL_S_S*), which is the global and local stresses and strains, is calculated in line 34 using the **Stress_Strain** function. Thereafter, in lines 35 to 41, the **Tsai_Wu_Par** and **Tsai_Wu** functions are used to determine the second output (*SR_T_W*) which is the strength ratios.

Function: Q_matrix

```
42   function Q = Q_matrix(inp)
%Calculates stiffness matrix (Q)
43   den = 1-inp(4)*inp(5);
44   Q11 = inp(1)/den;
45   Q22 = inp(2)/den;
46   Q12 = inp(4)*inp(2)/den;
47   Q66 = inp(3);
48   Q = [Q11 Q12 0; Q12 Q22 0; 0 0 Q66];
```

The **Q_matrix** function is based on equation (3.6), and is executed in line 27 above (**Conventional_Method** function). It is used to determine the [*Q*] matrix for each layer in the laminate. The input (*inp*) is the material constants for a particular layer, and the [*Q*] matrix is the output. The code in lines 43 to 47 determine the elements of the matrix, while the output is given by line 48.

Function: TQbar_matrix

```
49   function [T,Qbar] = TQbar_matrix(q, angle)
%Calculations T and Qbar matrices
50   c = cosd(angle);
51   s = sind(angle);
52   T = [c^2 s^2 2*s*c; s^2 c^2 -2*s*c; -s*c s*c c^2-s^2];
53   R = [1 0 0; 0 1 0; 0 0 2];
54   Qbar = inv(T)*q*R*T*inv(R);
```

This function is used to determine the [T] and [$\overline{Q}$] matrices for a particular layer. The inputs (line 49) include the [Q] matrix (q) and fibre orientation angle for the layer (*angle*). In lines 50 and 51, the cosine and sine of the fibre orientation angle are calculated. These are used to evaluate the [T] matrix (line 52), which is based on equation (3.9). The [Q] matrix (which was input (q)), the [T] matrix (calculated in line 52), and the Reuter matrix (defined in line 53) are used to compute the output [$\overline{Q}$] matrix in line 54.

Function: H_Z_det

```
55  function [H,Z] = H_Z_det(N,thick)
%Determines h and z values

%Calculate h values
56  hT = sum(thick);
57  H = zeros(N+1);
58  Z = zeros(N,3);
59  H(1) = -hT/2;
60  for w = 2:N+1
61      H(w) = H(w-1) + thick(w-1);
62  end

%Calculate z values
63  for u = 1:N
64      Z(u,1) = H(u);
65      Z(u,2) = H(u) + thick(u)/2;
66      Z(u,3) = H(u+1);
67  end
```

The **H_Z_det** function is used to find the h values required to determine the [A], [B] and [D] matrices, and the z values needed for the global strains. The inputs (line 55) include the number of layers (N) and the thickness of each layer (*thick*). The first part of the code (lines 56 to 62) is based on equation (3.19) and is used to determine the h values, which are the positions of the top and bottom of each layer from the mid-plane. The z values, which are the distances from the mid-plane to the points where the global strains will be determined, are found using lines 63 to 67. In this particular case, the code is written such that the calculated z values correspond to the top, middle and bottom position of each layer of the laminate.

Function: ABD_matrix

```
68  function ABD = ABD_matrix(N,h,Qbar)
%Determine A,B,D matrices
69  A = zeros(3,3);
70  B = zeros(3,3);
71  D = zeros(3,3);
72  for i = 1:3
73      for j = 1:3
74          sumA = 0;
75          sumB = 0;
76          sumD = 0;
77          for k = 1:N
78              sumA = sumA + Qbar{k,1}(i,j)*(h(k+1)-h(k));
79              sumB = sumB + Qbar{k,1}(i,j)*(h(k+1)^2-h(k)^2);
80              sumD = sumD + Qbar{k,1}(i,j)*(h(k+1)^3-h(k)^3);
81          end
82          A(i,j) = sumA;
83          B(i,j) = sumB/2;
84          D(i,j) = sumD/3;
85      end
86  end
87  ABD = [A B;B D];
```

The [*A*], [*B*] and [*D*] matrices are found via the above function. The input parameters (line 68) are the number of layers (*N*), distances of the top and bottom of each layer from the mid-plane (*h*), and the $[\overline{Q}]$ matrix for all the layers (*Qbar*). The code in lines 69 to 71 is used to initialise the arrays that would be used to store the calculated values for the [*A*], [*B*] and [*D*] matrices. These arrays are populated with values using the nested loops from line 72 to 86. The code in lines 78, 79 and 80 correspond to equations (3.16), (3.17) and (3.18), respectively. The output array (*ABD*) is given by line 87, and is in the form shown in equation 3.20.

As this point in the **Conventional_Method** function (line 33), the mid-plane strains and curvatures are determined. Thereafter the stresses and strains in the laminate are calculated via the **Stress_Strain** function.

Function: Stress_Strain

```
88  function gl_ll_strain_stress = Stress_Strain(N, Z, ep0_k, T,
    Qbar)
%Calculates global and local stresses and strains
89  R = [1 0 0;0 1 0;0 0 2];
```

```
90  gl_strain = cell(N,3);
91  gl_stress = cell(N,3);
92  ll_strain = cell(N,3);
93  ll_stress = cell(N,3);

94  for u = 1:N
95      for v = 1:3
96          ep_xy = [ep0_k(1);ep0_k(2);ep0_k(3)] +...
                    Z(u,v)*[ep0_k(4);ep0_k(5);ep0_k(6)];
97          sig_xy = Qbar{u,1}*ep_xy;
98          ep_12 = R*T{u,1}*inv(R)*ep_xy;
99          sig_12 = T{u,1}*sig_xy;
100         gl_strain(u,v) = {ep_xy};
101         gl_stress(u,v) = {sig_xy};
102         ll_strain(u,v) = {ep_12};
103         ll_stress(u,v) = {sig_12};
104     end
105 end
106 gl_ll_strain_stress = {gl_strain gl_stress ll_strain
    ll_stress};
```

The global and local stresses and strains are calculated using this function. The inputs (line 88) include the number of layers (*N*), distances from the mid-plane (*Z*), mid-plane strains and curvatures (*ep0_k*), the transformation matrices for all the layers (*T*), and the $[\overline{Q}]$ matrices for all the layers (*Qbar*). The Reuter matrix is defined in line 89, and lines 90 to 93 is used to initialise arrays containing *N* rows and three columns, which would be used to store the computed values. Each row in these arrays represents a layer of the laminate, and each column corresponds to the top, middle and bottom of each layer.

The nested loops from line 94 to 105 are used to calculate the various stresses and strains, and store them in the appropriate array. The code in line 96 represents equation (3.15) and is used to determine the global strains. The global stresses are calculated in line 97, which is equivalent to equation (3.10). In lines 98 and 99, the local strains and stresses are respectively computed. The calculated stresses and strains are stored in the correct arrays via lines 100 to 103. The output of the **Stress_Strain** function, given in line 106, is an array consisting of the global and local stress and strain arrays.

Function: Tsai_Wu_Par

```
107 function H_Val = Tsai_Wu_Par(limits)
%Modified Tsai-Wu Failure Theory Parameters
108 H_1 = (1/limits(1)) - (1/limits(2));
109 H_11 = 1/(limits(1)*limits(2));
110 H_2 = (1/limits(3)) - (1/limits(4));
111 H_22 = 1/(limits(3)*limits(4));
112 H_6 = 0;
113 H_66 = 1/limits(5)^2;
114 H_12 = -0.5*sqrt(1/(limits(1)*limits(2)*limits(3)*limits(4)));
115 H_Val = [H_1 H_11 H_2 H_22 H_6 H_66 H_12];
```

The **Tsai_Wu_Par** function is based on equation (3.22) and is used to determine the parameters for the Tsai-Wu failure criterion. The input is the material limits (*limits*) and the output (*H_Val*) is an array containing the calculated parameters.

Function: Tsai_Wu

```
116 function sr = Tsai_Wu(h_v, stress)
%Tsai-Wu Failure Theory with Mises-Hencky criterion
117 a = h_v(2)*stress(1)^2 + h_v(4)*stress(2)^2 +
    h_v(6)*stress(3)^2 + 2*h_v(7)*stress(1)*stress(2);
118 b = h_v(1)*stress(1) + h_v(3)*stress(2) + h_v(5)*stress(3);
119 c = -1;
120 %W1 = (-b+sqrt(b^2-4*a*c))/2/a;
121 %W2 = (-b-sqrt(b^2-4*a*c))/2/a;
122 sr = (-b+sqrt(b^2-4*a*c))/(2*a);
```

In this function the strength ratios, from the Tsai-Wu failure criterion in equation (3.23), are computed. The inputs of the function include the Tsai-Wu parameters (*h_v*) and the local stresses (*stress*). The output is the strength ratio (*sr*). Equation (3.23) is a quadratic equation and may therefore be solved using the quadratic formula. The code in lines 117 to 119 is used to determine the *a*, *b* and *c* coefficients required for the formula. The two roots may be found using the commented-out lines (lines 120 and 121). However, as stipulated in Chapter 3, the strength ratio cannot be negative and thus the negative root (line 121) is not applicable. The code in line 122 represents the positive root and is given as the output of the function.

As mentioned earlier, the **Constant_Stiffness_Method** script file provides the user with the option of viewing the calculated global and local stresses and strains (line 12).

The displaying of these properties requires the execution of the function **View_Stress_Strain_Menu** in line 14.

Function: View_Stress_Strain_Menu

```
123 function View_Stress_Strain_Menu(N, gl_ll_s_s)
%View global and local stresses and strains

124 ch = 0;
125 while ch ~= 5
126     disp(' ')
127     disp('Global and Local Stress and Strain Menu')
128     disp(' ')
129     disp('1. Global Strains')
130     disp('2. Global Stresses')
131     disp('3. Local Strains')
132     disp('4. Local Stresses')
133     disp('5. Quit')
134     disp(' ')
135     ch = input('Enter selection: ');
136     disp(' ')
137     switch (ch)
138         case {1}
139             Par = {'Eps_x','Eps_y','Gam_xy'};
140             disp('Global Strains')
141         case {2}
142             Par = {'Sig_x','Sig_y','Tau_xy'};
143             disp('Global Stresses (Pa)')
144         case {3}
145             Par = {'Eps_1','Eps_1','Gam_12'};
146             disp('Local Strains')
147         case {4}
148             Par = {'Sig_1','Sig_2','Tau_12'};
149             disp('Local Stresses (Pa)')
150         case {5}
151             return;
152         otherwise
153             disp('Invalid input')
154     end
155     View_Stress_Strain (N, Par, gl_ll_s_s{ch});
156 end
```

The inputs of the above function (line 123) are the number of layers (*N*), and the array containing the global and local stresses and strains (*gl_ll_s_s*). The code in lines 126 to 134 is used to display a menu of the available stresses and strains. The user is prompted (line 135) to input an appropriate choice (*ch*). Numeric integers from 1 to 5

are the only acceptable inputs. In lines 138 to 149, the variable *Par* is assigned different string sets depending on the value of *ch*. The function is closed when a value of 5 is entered (lines 150 to 151). In lines 152 to 153, an error message is displayed if any values or characters, other than 1, 2, 3, 4 or 5, are entered. A function called **View_Stress_Strain**, which is examined below, is executed in line 155.

Function: View_Stress_Strain

```
157 function View_Stress_Strain(N, par, St_Sr)
%Displays global or local stresses or strains
158 pos = {'Top ','Middle ','Bottom '};
159 fprintf(' Layer # \t\t %s','Position')
160 fprintf('\t\t\t ')
161 fprintf(par{(1)})
162 fprintf('\t\t\t\t ')
163 fprintf(par{(2)})
164 fprintf('\t\t\t\t ')
165 fprintf(par{(3)})
166 fprintf('\n')

167 for u = 1:N
168     fprintf('\t %.0f',u)
169     for v = 1:3
170         fprintf('\t\t\t\t %s',pos{(v)})
171         fprintf('\t\t\t %1.3e',St_Sr{u,v})
172         fprintf('\n')
173     end
174     fprintf('\n')
175 end
```

The purpose of this function is to arrange the stress or strain values in a table form as depicted in Figures 4.2 to 4.5. The inputs include the number of layers (*N*), the column headings for the numerical values (*Par*), and the stress or strain values passed from the previous function (*St_Sr*). The headings for each row and column are displayed via lines 159 to 166. The layout of the values is set by the code in lines 167 to 175.

Graphical User Interface (GUI) code

As mentioned in Chapter 4, the GUI for the CSM code consists of two panels (Input Panel and Output Panel) and contains various tables and buttons. The GUI requires two files, namely, a figure file and a code file. The figure file contains the graphical

layout and is shown in Figure B.1. The positions of the various pushbuttons, radio buttons, toggle switches, textboxes, and so forth are stored in this file. In addition, this file also contains the settings for the display colours of the GUI, the size and colour of the fonts, and the text for the textboxes.

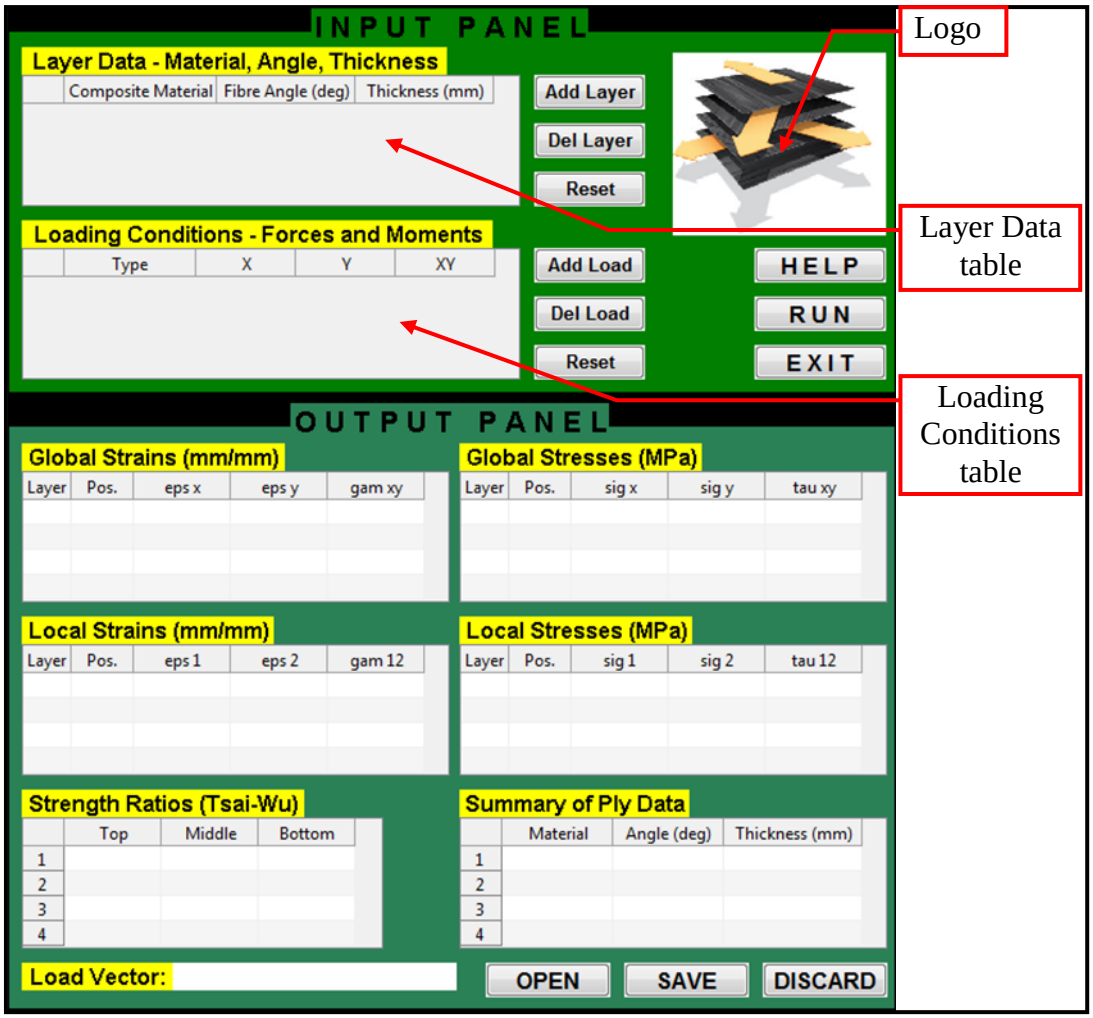


Figure B.1: Graphical layout of GUI for Constant Stiffness Method code

The functions essential for the operation of the GUI are included in the code file. This file replaces the script file (**Constant_Stiffness_Method**). It accepts and monitors the inputs entered by the user. Further, the **View_Stress_Strain_Menu** function is no longer needed as outputs are automatically displayed. The code file is shown below.

```
function varargout = GUI_Conventional(varargin)
% GUI_CONVENTIONAL M-file for GUI_Conventional.fig
% GUI_CONVENTIONAL, by itself, creates a new GUI_CONVENTIONAL or
raises the existing singleton*.
% H = GUI_CONVENTIONAL returns the handle to a new GUI_CONVENTIONAL
or the handle to the existing singleton*.
% GUI_CONVENTIONAL('CALLBACK',hObject,eventData,handles,...) calls
the local function named CALLBACK in GUI_CONVENTIONAL.M with the
given input arguments.
% GUI_CONVENTIONAL('Property','Value',...) creates a new
GUI_CONVENTIONAL or raises the existing singleton*.  Starting from
the left, property value pairs are applied to the GUI before
GUI_Conventional_OpeningFcn gets called.  An unrecognized property
name or invalid value makes property application stop.  All inputs
are passed to GUI_Conventional_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help
GUI_Conventional
% Last Modified by GUIDE v2.5 10-Jan-2011 12:36:45

% Begin initialization code - DO NOT EDIT
   gui_Singleton = 1;
   gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @GUI_Conventional_OpeningFcn, ...
    'gui_OutputFcn',  @GUI_Conventional_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
   if nargin && ischar(varargin{1})
        gui_State.gui_Callback = str2func(varargin{1});
   end
   if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
   else
       gui_mainfcn(gui_State, varargin{:});
   end
% End initialization code - DO NOT EDIT

% --- Executes just before GUI_Conventional is made visible.
   function GUI_Conventional_OpeningFcn(hObject, eventdata,
    handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_Conventional (see
VARARGIN)

% Choose default command line output for GUI_Conventional
   handles.output = hObject;
   M_M = find(strcmp(varargin, 'MainMenu'));
   handles.MainMenuVar = varargin{M_M+1};
   handles.HelpVal = 0;
```

```
% Update handles structure
17   guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
18   function Pict_Axes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Pict_Axes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
% Hint: place code in OpeningFcn to populate Pict_Axes
19   axes(hObject)
20   imshow('Layers.jpg')

% --- Outputs from this function are returned to the command line.
21   function varargout = GUI_Conventional_OutputFcn(hObject,
     eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
22   varargout{1} = handles.output;


%---------P R O G R A M M I N G   S T A R T S   H E R E----------%
23   Pict_Axes_CreateFcn(handles.Pic, eventdata, handles);
24   set(handles.Layer_Table,'Data',[]);
25   set(handles.Load_Table,'Data',[]);

26   MatN = load('Database-Comp.mat','Name');
27   for t = 1:length(MatN.Name)
28       Col1(t) = MatN.Name(t);
29   end
30   ColForm = {Col1,'numeric','numeric'};
31   set(handles.Layer_Table,'ColumnFormat',ColForm);
32   uiwait(handles.Figure_Conventional)


%----------I N P U T   L A Y E R   D A T A   T A B L E----------%
% --- Executes when selected cell(s) is changed in Layer_Table.
33   function Layer_Table_CellSelectionCallback(hObject, eventdata,
     handles)
% hObject    handle to Layer_Table (see GCBO)
% eventdata  structure with the following fields (see UITABLE)
% Indices: row and column indices of the cell(s) currently selected
% handles    structure with handles and user data (see GUIDATA)
34   handles.LaySel = eventdata.Indices;
35   guidata(hObject,handles);

% --- Executes on button press in Add_Layer_Button.
36   function Add_Layer_Button_Callback(hObject, eventdata, handles)
% hObject    handle to Add_Layer_Button (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
37   Data = get(handles.Layer_Table,'Data');
38   k = size(Data,1)+1;
39   Data{k,1} = 'Graphite-Epoxy';
40   Data{k,2} = 0;
41   Data{k,3} = 0;
42   set(handles.Layer_Table,'Data',Data);

% --- Executes on button press in Del_Layer_Button.
43   function Del_Layer_Button_Callback(hObject, eventdata, handles)
% hObject    handle to Del_Layer_Button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
44   if isfield(handles,'LaySel')
45       Row = handles.LaySel(1);
46       Data = get(handles.Layer_Table,'Data');
47       Data(Row,:) = '';
48       set(handles.Layer_Table,'Data',Data);
49   end

% --- Executes on button press in Reset_Layer_Button.
50   function Reset_Layer_Button_Callback(hObject, eventdata,
     handles)
% hObject    handle to Reset_Layer_Button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
51   set(handles.Layer_Table,'Data',[]);


%---------I N P U T   L O A D I N G   D A T A   T A B L E----------%
% --- Executes when selected cell(s) is changed in Load_Table.
52   function Load_Table_CellSelectionCallback(hObject, eventdata,
     handles)
% hObject    handle to Load_Table (see GCBO)
% eventdata  structure with the following fields (see UITABLE)
% Indices: row and column indices of the cell(s) currently selected
% handles    structure with handles and user data (see GUIDATA)
53   handles.LoadSel = eventdata.Indices;
54   guidata(hObject,handles);

% --- Executes on button press in Add_Load_Button.
55   function Add_Load_Button_Callback(hObject, eventdata, handles)
% hObject    handle to Add_Load_Button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
56   Data = get(handles.Load_Table,'Data');
57   k = size(Data,1)+1;
58   Data{k,1} = 'Force';
59   Data{k,2} = 0;
60   Data{k,3} = 0;
61   Data{k,4} = 0;
62   set(handles.Load_Table,'Data',Data);
```

```
% --- Executes on button press in Del_Load_Button.
63  function Del_Load_Button_Callback(hObject, eventdata, handles)
% hObject    handle to Del_Load_Button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
64  if isfield(handles,'LoadSel')
65      Row = handles.LoadSel(1);
66      Data = get(handles.Load_Table,'Data');
67      Data(Row,:) = '';
68      set(handles.Load_Table,'Data',Data);
69  end

% --- Executes on button press in Reset_Load_Button.
70  function Reset_Load_Button_Callback(hObject, eventdata,
    handles)
% hObject    handle to Reset_Load_Button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
71  set(handles.Load_Table,'Data',[]);


%----I N P U T   P A N E L   E X E C U T A B L E   B U T T O N S---%
% --- Executes on button press in Conv_Input_Panel_Help.
72  function Conv_Input_Panel_Help_Callback(hObject, eventdata,
    handles)
% hObject    handle to Conv_Input_Panel_Help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
73  handles.HelpFig = Help_Conv;
74  handles.HelpVal = 1;
75  guidata(hObject,handles);

% --- Executes on button press in Conv_Input_Panel_Run.
76  function Conv_Input_Panel_Run_Callback(hObject, eventdata,
    handles)
% hObject    handle to Conv_Input_Panel_Run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Gets material,fibre angle and thickness data from Layer Data Table
77  LayerGo = CheckData(hObject, eventdata, handles, 'Layer');
78  if LayerGo == 1
79      CompMat = load('Database-Comp.mat');
80      Layer_Data = get(handles.Layer_Table,'Data');
81      N = size(Layer_Data,1);
82      Mat_Data = zeros(N,17);
83      Ply_Angle = zeros(1,N);
84      Ply_Thick = zeros(1,N);
85      for y = 1:N
86          for u = 1:length(CompMat.Name)
87              if strcmp(Layer_Data{y,1},CompMat.Name{u})
88                  Mat_Data(y,:) = CompMat.Data(u,:);
89              end
```

```
90          end
91          Ply_Angle(y) = Layer_Data{y,2};
92          Ply_Thick(y) = Layer_Data{y,3};
93      end
94  end

%Gets loading conditions data from Loading Data Table
95  LoadGo = 0;
96  if LayerGo == 1
97      LoadGo = CheckData(hObject, eventdata, handles, 'Load');
98  end
99  if LoadGo == 1
100     Load_Data = get(handles.Load_Table,'Data');
101     L = size(Load_Data,1);
102     F_X = 0;
103     F_Y = 0;
104     F_XY = 0;
105     M_X = 0;
106     M_Y = 0;
107     M_XY = 0;
108     for y = 1:L
109         switch Load_Data{y,1}
110             case ('Force')
111                 F_X = F_X + Load_Data{y,2};
112                 F_Y = F_Y + Load_Data{y,3};
113                 F_XY = F_XY + Load_Data{y,4};
114             case ('Moment')
115                 M_X = M_X + Load_Data{y,2};
116                 M_Y = M_Y + Load_Data{y,3};
117                 M_XY = M_XY + Load_Data{y,4};
118         end
119     end
120     NM = [F_X;F_Y;F_XY;M_X;M_Y;M_XY];
121 end

%Finds global and local stresses and strains, and SR matrix
122 if (LayerGo == 1) && (LoadGo == 1)
123 HH = helpdlg('Calculating results. This may take a few
    minutes.','Please Wait...');
124 [GL_LL_S_S, SR_T_W] = Conventional_Method(N, Mat_Data,
    Ply_Angle, Ply_Thick, NM);
125     for t = 1:400000000; end;
126     close(HH);
%Populate output tables with results
127     for r = 1:4
128         TableData = View_Stress_Strain(N, GL_LL_S_S{r});
129         switch r
130             case (1)
131                 set(handles.G_Strain_Table,'Data',TableData);
132             case (2)
133                 set(handles.G_Stress_Table,'Data',TableData);
```

```
134             case (3)
135                 set(handles.L_Strain_Table,'Data',TableData);
136             case (4)
137                 set(handles.L_Stress_Table,'Data',TableData);
138         end
139     end
140     set(handles.SR_Table,'Data',SR_T_W);
141     PlyData = get(handles.Layer_Table,'Data');
142     set(handles.Summary_Table,'Data',PlyData);
143     Force_Vector = num2str([F_X F_Y F_XY M_X M_Y M_XY]);
144     set(handles.Force_Vector_Text,'String',Force_Vector);
145 end

%--Checks for data in input tables; checks for non-numeric values--%
146 function RunGo = CheckData(hObject, eventdata, handles, TABLE)
147 if strcmp(TABLE,'Layer')
148     Data = get(handles.Layer_Table,'Data');
149     Col = 2;
150     Name = {'Fibre Angle','Thickness'};
151 else
152     Data = get(handles.Load_Table,'Data');
153     Col = 3;
154     Name = {'X','Y','XY'};
155 end
156 if size(Data,1) == 0
157     err = horzcat('There is no data entered in the ',TABLE,'
        table');
158     errordlg(err,'ERROR','modal');
159     RunGo = 0;
160     return;
161 end
162 N = size(Data,1);
163 for x = 1:Col
164     for y = 1:N
165         if isnan(Data{y,x+1})
166             err = {horzcat('The value in the ',Name{x},'
                column for layer ',num2str(y),' is not
                numeric');...
                'Please correct before continuing...'};
167             errordlg(err,'ERROR','modal');
168             RunGo = 0;
169             return;
170          end
171     end
172 end

173 if strcmp(TABLE,'Layer')
174     for y = 1:N
175         if Data{y,3} == 0
176             err = {horzcat('Layer ',num2str(y),' has zero
                thickness');...
```

```
                'Please correct before continuing...'};
177             errordlg(err,'ERROR','modal');
178             RunGo = 0;
179             return;
180         end
181     end
182 end
183 RunGo = 1;

% --- Executes on button press in Conv_Input_Panel_Close.
184 function Conv_Input_Panel_Close_Callback(hObject, eventdata,
    handles)
% hObject    handle to Conv_Input_Panel_Close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
185 Figure_Conventional_CloseRequestFcn(handles.Figure_Conventional
    , eventdata, handles);


%---O U T P U T   P A N E L   E X E C U T A B L E   B U T T O N S--%
% --- Executes on button press in Conv_Output_Panel_Open.
186 function Conv_Output_Panel_Open_Callback(hObject, eventdata,
    handles)
% hObject    handle to Conv_Output_Panel_Open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
187 [FileName,PathName] = uigetfile('*.conv','Select the results
    file');
188 if (FileName ~= 0)
189 OpenFile = [PathName,FileName];
190 Table_Values = load(OpenFile,'-mat');
191 set(handles.G_Strain_Table,'Data',Table_Values.G_Strain);
192 set(handles.G_Stress_Table,'Data',Table_Values.G_Stress);
193 set(handles.L_Strain_Table,'Data',Table_Values.L_Strain);
194 set(handles.L_Stress_Table,'Data',Table_Values.L_Stress);
195 set(handles.SR_Table,'Data',Table_Values.SR);
196 set(handles.Summary_Table,'Data',Table_Values.Ply_Data);
197 set(handles.Force_Vector_Text,'String',Table_Values.Force);
198 end

% --- Executes on button press in Conv_Output_Panel_Save.
199 function Conv_Output_Panel_Save_Callback(hObject, eventdata,
    handles)
% hObject    handle to Conv_Output_Panel_Save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
200 G_Strain = get(handles.G_Strain_Table,'Data');
201 G_Stress = get(handles.G_Stress_Table,'Data');
202 L_Strain = get(handles.L_Strain_Table,'Data');
203 L_Stress = get(handles.L_Stress_Table,'Data');
204 SR = get(handles.SR_Table,'Data');
205 Ply_Data = get(handles.Summary_Table,'Data');
```

```
206 Force = get(handles.Force_Vector_Text,'String');
207 [FileName,PathName] = uiputfile('*.conv','Save Results As');
208 if (FileName ~= 0)
209     SaveFile = [PathName,FileName];
210     save(SaveFile,'G_Strain','G_Stress','L_Strain','L_Stress',
      'SR','Ply_Data','Force');
211 end

% --- Executes on button press in Conv_Output_Panel_Discard.
212 function Conv_Output_Panel_Discard_Callback(hObject, eventdata,
    handles)
% hObject    handle to Conv_Output_Panel_Discard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
213 set(handles.Layer_Table,'Data',[]);
214 set(handles.Load_Table,'Data',[]);
215 set(handles.G_Strain_Table,'Data',[]);
216 set(handles.G_Stress_Table,'Data',[]);
217 set(handles.L_Strain_Table,'Data',[]);
218 set(handles.L_Stress_Table,'Data',[]);
219 set(handles.SR_Table,'Data',[]);
220 set(handles.Summary_Table,'Data',[]);
221 set(handles.Force_Vector_Text,'String','');


%------------C L O S E   O R   E X I T   P R O G R A M-------------%
% --- Executes when user attempts to close Figure_Conventional.
222 function Figure_Conventional_CloseRequestFcn(hObject,
    eventdata, handles)
% hObject    handle to Figure_Conventional (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: delete(hObject) closes the figure
223 if handles.HelpVal == 1
224     if ishandle(handles.HelpFig)
225         delete(handles.HelpFig);
226     end
227 end
228 mainHandles = guidata(handles.MainMenuVar);
229 set(mainHandles.Check,'String','GO');
230 uiresume(handles.Figure_Conventional)
231 delete(handles.Figure_Conventional);
```

In the above code, line 1 is a function assignment that is used to accept various inputs (*varargin*) and generate multiple outputs (*varargout*). The GUI is initialised via lines 2 to 11. This is where the basic functions of the GUI are prepared for use. Also the input and output variables (if any) are analysed and recognised here.

The function from line 12 to 17 is executed just before the GUI is made visible. It should be noted that all functions related to the GUI layout and elements, such as textboxes, buttons, etc., have three generic inputs, namely, *hObject*, *eventdata* and *handles*. These variables link the code to the figure file, and are used to store the global variables used in the GUI code. The code in lines 13 to 15, together with that in line 32, is used to lock the GUI, or in other words, prevent the execution of other GUIs from the main graphical interface, which is discussed in Chapter 7. This feature is incorporated in all the GUIs developed in this study. In line 16, a variable (*handles.HelpVal*) is created that tracks whether the help function was used, and line 17 is used to store the newly created variables in the GUI database. The function from line 18 to 20 displays the logo indicated in Figure B.1. The outputs of the GUI (if any) is returned to the command line via the function in lines 21 to 22.

The coding for the various features of the GUI, such as pushbuttons, tables, textboxes, etc., begins at line 23 where the function that displays the logo is executed. The Layer Data and Loading Conditions tables, which are labelled in Figure B.1, are cleared of all values by lines 24 and 25. In line 26, the names of the materials from the composite database are loaded, and these are stored in the drop-down list of the Layer Data table via lines 27 to 31.

The next four functions (lines 33 to 51) deal with the Layer Data table and the three pushbuttons associated with it. The first of these functions, in lines 33 to 35, is used to track the row, in the Layer Data table, that is currently being used or selected. The code in lines 37 to 42 contains the function for the "Add Layer" pushbutton. This button is used to add a new row to the Layer Data table. When the button is pressed, the contents of the table is stored using the variable *Data* (line 37). A new row is then added to the table (lines 39 to 41), and the values stored by the variable *Data* are output to the table (line 42).

The "Del Layer" button is used to delete a row from the Layer Data table. In line 45, the row that is highlighted/selected in the table (obtained from the function in lines 33 to 35) is determined. The values in the table are then stored using the variable *Data*

(line 46), the selected values are deleted (line 47), and the amended values are output to the table (line 48). Lines 50 to 51 contain the code for the "Reset" pushbutton, which is used to clear all entries from the Layer Data table.

The three pushbuttons for the Loading Conditions table operate identically to those discussed above for the Layer Data table. The row on the Loading Conditions table that is currently being used or selected is determined by the function in lines 52 to 54. The "Add Load" pushbutton adds a new row to the table via the function in lines 55 to 62. The function from line 63 to 69 is executed when the "Del Load" button is pressed and is used to delete the highlighted row from the Loading Conditions table. The "Reset" pushbutton is used to clear all data from this table.

The Input Panel has three other pushbuttons, namely, "HELP", "RUN" and "EXIT". The function from line 72 to 75 is used to run the help facility. As mentioned in Chapter 4, this facility gives a brief description of the GUI and shows the correct procedure for its operation. If the help facility was activated, it must be closed when the GUI is exited in order to prevent conflict with the other GUIs and to promote good programming practice. Therefore, in line 74, the *handles.HelpVal* variable is assigned a value of '1' to indicate that the help facility is in use. This variable is used when the GUI is being exited to ensure that the help facility is also closed.

The function of the "RUN" button is contained in lines 76 to 145, and may be divided into three parts. The first two parts deal with obtaining the values from the Layer Data and Loading Conditions tables. Here a function called **CheckData**, which is analysed later, is used to validate the entries in each of the tables. The last part determines the global and local stresses and strains as well as the strength ratios, and populates the tables on the Output Panel.

The first part of the "RUN" button function (lines 77 to 94) is used to extract the material properties, fibre orientation angles and layer thicknesses from the Layer Data table. In line 77, the variable *LayerGo* is assigned as the output of the **CheckData** function. If the value of this variable is '1' (line 78), the composite database is loaded

(line 79), and the entries of the Layer Data table are stored using the array *Layer_Data* (line 80). The number of layers in the laminate (*N*) is determined by line 81 and, based on this, the code in lines 82 to 84 are used to create arrays to store the values for the material properties (*Mat_Data*), fibre orientation angles (*Ply_Angle*), and layer thicknesses (*Ply_Thick*). The nested loop in lines 85 to 93 is used to store the entries from *Layer_Data* array to the appropriate variable for each layer of the laminate.

The extraction of the loading conditions from the Loading Conditions table, which is the second part of the "RUN" button function, is achieved via lines 95 to 121. In line 95, the variable *LoadGo* is set to zero. This variable is similar to the variable *LayerGo* discussed above. If the value of the variable *LayerGo* is '1' (line 96), then the variable *LoadGo* is assigned as the output of the **CheckData** function in line 97. The code in lines 100 to 121 is executed if the value of the variable *LoadGo*, in line 99, is '1'. The entries of the Loading Conditions table are stored using the array *Load_Data* (line 100). The number of loading conditions that were entered is determined in line 101. In lines 102 to 107, variables for the forces and moments in the x- and y-directions as well as in the xy-plane are created and set to zero. These variables are used to store the force and moment components from the *Load_Data* array via the coded loop from line 108 to 119. The loading vector (*NM*) that is used in the stress and strain calculations is assigned values in line 120.

As mentioned above, the global and local stresses and strains as well as the strength ratios are determined by the last part of the "RUN" button function (lines 122 to 145). This part of the code is also used to populate the tables on the Output Panel. If the variables *LayerGo* and *LoadGo* both have a value of '1' (line 122), then a message box is displayed (line 123) that states that the calculations are being performed. The **Conventional_Method** function, with the necessary input parameters, is executed in line 124. The code in line 125 is used as a delay to ensure that the user has time to read the message, and the message box is closed using line 126.

Once the computations are completed, the results are displayed in the tables of the Output Panel via lines 127 to 145. The coded loop from line 127 to 139 passes one set

of stress or strain values at a time to the **View_Stress_Strain** function, which rearranges the data for displaying in the tables on the Output Panel. It should be noted that this function is not the same as the one discussed above and will be examined later. The code in lines 131, 133, 135 and 137 is used to populate the Global Strains, Global Stresses, Local Strains and Local Stresses tables, respectively. The strength ratio values are displayed in the Strength Ratios table via line 140. The values from the Layer Data table on the Input Panel are extracted and displayed on the Summary of Ply Data table on the Output Panel using lines 141 to 142. The code in lines 143 to 144 is used to display the loading vector in the Load Vector textbox.

The **CheckData** function was used earlier in lines 77 and 97. The purpose of this function is to validate the entries of the Layer Data and Loading Conditions tables on the Input Panel. In line 146, the variable *RunGo* is assigned as the output of the function, and the value of this variable may be either '1', if the particular table has valid entries, or '0', if there is at least one invalid entry. Further, in line 146, it may be seen that there is an extra input variable called *TABLE*. This variable may assume one of two values depending on the line of code that initialised the function. In line 77 a value of 'Layer' is assigned to the variable, while the value of 'Load' is allotted by the code in line 97.

If the variable *TABLE* has the value of 'Layer' (line 147), then the entries in the Layer Data table is stored using the array *Data* (line 148), and values for the variables *Col* and *Name* are assigned (lines 149 and 150). These values are required later. On the other hand, if the value of 'Load' is assigned to the variable *TABLE* (line 151), the entries from the Loading Conditions table are stored in the array *Data* (line 152), and the variables *Col* and *Name* assume different values than above (lines 153 and 154).

Once the entries have been extracted from either table, the code in lines 156 to 161 is used to test whether the array *Data* contains any values. If the array is empty, an error message is displayed (lines 157 to 158) stating that there are no entries in the table. The variable *RunGo* is set to zero (line 159), and, in line 160, the execution of the code is returned to the line that initiated the function.

Conversely, if the array *Data* contains values, then the code in lines 162 to 172 is run. Here, particular entries in the array *Data* are tested to ensure that they are numeric values. A non-numeric entry results in an error message being displayed (lines 166 to 167) that prompts the user to correct the erroneous value. In line 168, a value of '0' is assigned to the variable *RunGo*, and the execution of the code is redirected to line that initiated the function.

It is important to remember that a layer in a fibre reinforced laminate cannot have zero thickness. Therefore the code in lines 173 to 182 is used to test the thickness values in the Layer Data table. If any layer has zero thickness, an error message is displayed prompting the user to enter a non-zero value. Here again, the execution of the code is returned to the line that initiated the function.

If all the entries in the relevant table pass the tests in the **CheckData** function, or, in other words, all the data entries are valid, then the variable *RunGo* is set to 1 (line 183). This value is assumed by the variables *LayerGo* in line 77 and *LoadGo* in line 97.

The "EXIT" pushbutton is the last button on the Input Panel. The function in lines 184 to 185 is used to redirect the code to the function in line 222, which is discussed later, where the GUI is properly closed.

There are three pushbuttons on the Output Panel, namely, "OPEN", "SAVE" and "DISCARD". The "OPEN" pushbutton executes the function from line 186 to 198. The purpose of this function is to display previously stored results. When this button is pressed, a window is opened (line 187) where the user may select a previously saved results file. If a file is selected (line 188), it is loaded (line 189), and the code in lines 190 to 197 is used to populate the tables on the Output Panel.

The "SAVE" pushbutton is used to store the current results to a file. First the data from the tables on the Output Panel is stored via lines 200 to 206. A window is opened (line 207) where the user chooses the name and destination of the saved file. If a file name has been specified (line 208), the current data is saved to the file via lines 209 to 210.

The function in lines 212 to 221 is for the “DISCARD” pushbutton, which is used to clear the entries from all the tables on both the Input Panel and Output Panel.

The code in lines 222 to 231 is used to properly close the GUI. If the value of the variable *handles.HelpVal* is ‘1’ (line 223), it signifies that the help windows is open, and this window is closed using lines 224 to 226. As discussed earlier, the GUI was locked to prevent execution of other GUIs from the main graphical interface. The code in lines 228 to 230 is used to unlock the GUI, and the GUI is closed via line 231.

It was mentioned earlier that the function **View_Stress_Strain** was used to display the results on the tables in the Output Panel (line 128). This function is shown below.

Function: View_Stress_Strain (Modified)

```
232 function TableData = View_Stress_Strain(N, SS)
%Displays global or local stresses or strains
233 Pos = {'Top','Middle','Bottom'};
234 TableData = cell(3*N,5);
235 t = 0;
236 for p = 1:N
237     for q = 1:3
238         RowData = SS{p,q}';
239         t = t + 1;
240         TableData{t,1} = p;
241         TableData{t,2} = Pos{q};
242         TableData{t,3} = RowData(1);
243         TableData{t,4} = RowData(2);
244         TableData{t,5} = RowData(3);
245     end
246 end
```

There are two inputs in the above function (line 232), namely, the number of layers (*N*) and a particular set of stress or strain values (*SS*). Parameters that will form part of the output table is assigned to the variable *Pos* in line 233. An array *TableData* is created (line 234) based on the value of *N*, and the variable *t* (line 235) is used as a counter. The nested loop in lines 236 to 246 is used to rearrange the stress/strain values, together with the parameters in the variable *Pos*, for display in the relevant table on the Output Panel.

APPENDIX C: MATLAB CODE FOR VARIABLE STIFFNESS METHOD

In this appendix the codes developed in Chapter 5 for the optimisation of the fibre layup parameters, namely, FAO, ATO and VSM, are examined line-by-line. Each code consists of a script file and numerous function files. As stated in Appendix B, the script file is used to manage the input parameters, execution of the different functions and outputs to the screen. The function files contain the code for the various calculations. Each line of code presented below is numbered consecutively, however, each function is a separate file, unless otherwise stated. There are comments included in the code, and these are depicted by "%" at the beginning of the line. The appendix is concluded with the discussion of the code for the Graphical User Interface (GUI).

Main script file: Fibre-Angle-Opt (FAO)

```
%FAO code: Variable angle but fixed thickness
%Inputs: material properties, material limits, loading conditions
%Outputs: fibre angles, number of layers

1    disp('FIBRE-ANGLE-OPT CODE')
2    disp('INPUT: Material Properties & Loading Conditions')
3    disp('OUTPUT: No. of Layers & Fibre Angles')

%Obtains material properties, allowable stresses
4    Mat_prop = [181 10.3 7.17 0.28 0.016];
5    Input_limits = [1500 1500 40 246 68];
6    Mat_data{1} = Mat_prop;
7    Mat_data{2} = Input_limits;

%Calculation of material dependent variables
     %Calculate Q matrices
8    Q = Q_matrix(Mat_data{1});
     %Calculate Tsai-Wu parameters
9    T_W_Par = Tsai_Wu_Par(Mat_data{2});

%Forces and moments vector
10   NM = [1000;1000;0;0;0;0];

11   Thickness = 5;
12   Lam_Data = {Q,Thickness,NM,T_W_Par};

%Calculation of variables that depend on fibre angle
     %Initial fibre angles
13   Ply_Angle = 0;
     %Calculates T,Qbar,h,z,A,B,D,Ep0-K,global and local stresses
     %and strains, and determines the strength ratio
```

```
14   [SR,~] = S_R_Matrix(Lam_Data,Ply_Angle);

%Finds minimum SR per layer and then min and max of that
15   SR_Layer = min(SR,[],2);
16   SR_Min = min(SR_Layer);
17   SR_Max = max(SR_Layer);

18   while (SR_Min < 1) || (SR_Max > 1.2)
         %Change angles to obtain appropriate SR on each layer
19       Ply_Angle = Optimise_Angles(Lam_Data,Ply_Angle);
20       [SR,~] = S_R_Matrix(Lam_Data,Ply_Angle);

         %Finds minimum SR per layer and then min and max of that
21       SR_Layer = min(SR,[],2);
22       SR_Min = min(SR_Layer);
23       SR_Max = max(SR_Layer);

         %Checks whether min SR is < 1 or max SR is > 1.2
24       if (SR_Min < 1) || (SR_Max > 1.2)
25           index = length(Ply_Angle)+1;
26           Ply_Angle(index) = 0;
27           Lam_Data{2}(index) = Thickness;
28       end
29   end

30   Ply_Angle
31   SR
32   disp('Complete')
```

In lines 1 to 3 of the above script file, the name, inputs and outputs of the code are displayed on the screen. Next (lines 4 to 5) the material properties and material limits are assigned to the variables *Mat_prop* and *Input_limits*, respectively. The contents of these two variables are combined into the array *Mat_data* in lines 6 and 7.

As stated in Chapter 5, the [Q] matrix and the parameters for the Tsai-Wu failure criterion do not depend on the fibre orientation angle. Therefore they may be calculated separately and this is accomplished by the code in lines 8 and 9. The functions in these lines, namely, **Q_matrix** and **Tsai_Wu_Par**, have been analysed in Appendix B, and hence will not be discussed here.

In line 10 the loading vector *NM* is initialised, while the fixed thickness value for each layer is set in line 11. An array *Lam_Data* is used in line 12 to store the data required

to determine the *SR* values, which include the [Q] matrix, the fixed thickness value, the loading vector, and the parameters for the Tsai-Wu failure criterion. The assumption for the initial value of the fibre orientation angle is set in line 13.

The code in line 14 is used to execute the function **S_R_Matrix** which determines the strength ratios at the top, middle, bottom of each layer. Thereafter the minimum *SR* value for each row (layer) is evaluated (line 15), and the minimum and maximum of these values are determined in lines 16 and 17, respectively. This means that the variable *SR_Min* contains the value of the lowest strength ratio for the entire laminate, while the highest strength ratio value is assigned to the variable *SR_Max*.

The coded loop from line 18 to 29 is executed if the value of *SR_Min* is less than 1 or if the value of *SR_Max* is greater than 1.2. In other words, the code in lines 18 to 29 is run if the *SR* value is not in the range specified in Chapter 5. If this is the case, the function **Optimise_Angles** is implemented in line 19. This function is used to obtain the fibre orientation angle that produces the *SR* value closest to the range stipulated in Chapter 5 (between 1 and 1.2). Since the fibre orientation angle has changed, the function **S_R_Matrix** is rerun (line 20) in order to obtain new *SR* values based on the changed angles. New values for the variables *SR_Min* and *SR_Max* are obtained via lines 21 to 23, and, if these values are not in the specified range (line 24), a new layer is added on (line 25) with the initial fibre orientation angle (line 26) and the fixed thickness value (line 27).

The outputs of the FAO code are the number of fibre layers, fibre orientation angle of each layer, and the strength ratios. These are given by the code in lines 30 and 31. In line 32 a message is displayed which indicates that the design process is completed.

Function: S_R_Matrix

```
33   function [SR_T_W,GL_LL_S_S] = S_R_Matrix(Lam_Dat,Angle)
%Determines the strength ratio at top, middle, bottom for each layer
%Initial values
34   N = length(Angle);
35   Ply_Data = cell(N,3);
36   SR_T_W = zeros(N,3);
```

```
%Calculates T,Qbar
37   for p = 1:N
38       [T,Qbar] = TQbar_matrix(Lam_Dat{1}, Angle(p));
39       Ply_Data(p,:) = {Angle(p), T, Qbar};
40   end

%Calculate h and z values
41   [h,z] = H_Z_det(N, Lam_Dat{2});
%Determine A,B,D matrices
42   ABD = ABD_matrix(N, h, Ply_Data(:,3));
%Calculate Ep0-K vector
43   Ep0_K = inv(ABD)*Lam_Dat{3};

%Calculate global and local stresses and strains on each layer
44   GL_LL_S_S = Stress_Strain(N, z, Ep0_K, Ply_Data(:,2),
     Ply_Data(:,3));

%Find strength ratio using Tsai-Wu Failure Theory
45   for u = 1:N
46       for v = 1:3
47           SR_T_W(u,v) = Tsai_Wu(Lam_Dat{4}, GL_LL_S_S{4}{u,v});
48       end
49   end
```

As mentioned earlier, the function **S_R_Matrix** is used to calculate the *SR* values at the top, middle and bottom of each fibre layer. The inputs of the function (line 33) include the laminate data ([Q] matrix, fixed thickness value, loading vector, and parameters for the Tsai-Wu failure criterion) and the fibre orientation angles. The outputs are the strength ratios, and, the global and local stresses and strains.

The above function is essentially the CSM code and almost identical to the function **Conventional_Method** in Appendix B. The number of fibre layers (*N*) is determined in line 34, and, in lines 35 and 36, the arrays *Ply_Data* and *SR_T_W* are initialised. *Ply_Data* is used to store the fibre orientation angles, and the [T] and $[\overline{Q}]$ matrices for each layer, while the strength ratios are stored using *SR_T_W*.

The [T] and $[\overline{Q}]$ matrices, for each layer, are evaluated by the code in lines 37 to 40. In lines 41 to 44, the h_k and *z* values, the [A], [B] and [D] matrices, the mid-plane strains and curvatures, and, the global and local stresses and strains are determined. The *SR* values at the top, middle and bottom of each layer are calculated using the

coded loop in lines 45 to 49. The functions used in lines 37 to 49, namely, **TQbar_matrix**, **H_Z_det**, **ABD_matrix**, **Stress_Strain** and **Tsai_Wu**, are identical to those discussed in Appendix B and will therefore not be discussed here.

Function: Optimise_Angles

```
50  function Ply_Angle = Optimise_Angles(Lam_Dat,Angle)
%Optimises angles to obtain appropriate SR per layer

%Initial values
51  x = 0;
52  y = 0;
53  loop = 0;
%Finds min SR per layer and sorts ascending/descending
54  [SR_S,P] = Sort_SR(Lam_Dat,Angle);
%Corrects angle according to SR for each layer
55  Angle = Best_Angle(Lam_Dat,Angle,SR_S,P);
%Finds NEW min SR per layer and sorts ascending/descending
56  [New_SR_S,New_P] = Sort_SR(Lam_Dat,Angle);

%Compares old and new SR values and changes angles if necessary
57  while SR_S ~= New_SR_S
58      loop = loop + 1;
59      y = y + 1;
60      if New_P(y) ~= P(y)
61          SR_S = New_SR_S;
62          P = New_P;
63          Angle = Best_Angle(Lam_Dat,Angle,SR_S,P);
64          [New_SR_S,New_P] = Sort_SR(Lam_Dat,Angle);
65          y = 0;
66      end
67      if P == New_P
68          x = x + 1;
69          if New_SR_S(x) ~= SR_S(x)
70              SR_S = New_SR_S;
71              P = New_P;
72              Angle = Best_Angle(Lam_Dat,Angle,SR_S,P);
73              [New_SR_S,New_P] = Sort_SR(Lam_Dat,Angle);
74              x = 0;
75              y = 0;
76          end
77      end
78      if loop == 50
79          break
80      end
81  end
82  Ply_Angle = Angle;

83  function [SR_S,P] = Sort_SR(Lam_Dat,Angle)
```

```
%Finds SR; min SR per layer; sorts SR either ascending/descending
84    [SR,~] = S_R_Matrix(Lam_Dat,Angle);
85   SR_Layer_Min = min(SR,[],2);
86   if max(SR_Layer_Min) > 1.2
87       [SR_S,P] = sort(SR_Layer_Min,'descend');
88   else
89       [SR_S,P] = sort(SR_Layer_Min,'ascend');
90   end
```

This function is used to obtain the *SR* value that is closest to the range between 1 and 1.2. The inputs (line 50) include the laminate data (*Lam_Dat*) and the fibre orientation angles (*Angle*), and the output (*Ply_Angle*) is the adjusted fibre orientation angles. The function contains a sub-function called **Sort_SR** which is discussed later.

The variables *x*, *y* and *loop* are initiated in lines 51 to 53 and are used as counters. In line 54, the minimum *SR* value for each layer is calculated and sorted in either ascending or descending order using the sub-function **Sort_SR**. The variable *SR_S* stores the sorted values and the variable *P* contains the position of the values prior to being sorted. Next (line 55) the function **Best_Angle** is executed which varies the fibre orientation angle in order to obtain a better *SR* value. Thereafter (line 56) the sub-function **Sort_SR** is executed to obtain a new *SR* value for each layer based on the angles determined in line 55. These are stored using the variable *New_SR_S* and the variable *New_P* is used to store the position of the values prior to sorting.

In the coded loop from line 57 to 81 the old *SR* values (*SR_S*) are compared to the new *SR* values (*New_SR_S*) to determine which of these are better. First, for a particular layer, the old and new positions of the sorted *SR* values (*P* and *New_P*) are compared (line 60). If they are unequal then it means that the *SR* value was improved and the old values are discarded in favour of the new ones (line 61). The contents of the variable *P* are also overwritten by the variable *New_P* (line 62). The function **Best_Angle** is run to determine if the *SR* values may be further improved (line 63). In line 64, new strength ratios and position values, based on the angles obtained in line 63, are calculated using the sub-function **Sort_SR**. The counter *y* is reset in line 65. This part of the code is executed until the new position values (*New_P*) are equal to the old position values (*P*).

If the variables *New_P* and *P* are equal (line 67), then, for a particular layer, the old and new minimum *SR* values are compared (*SR_S* and *New_SR_S*) in line 69. As before, if these values are unequal, it implies that the *SR* value was improved. Hence, in lines 70 and 71, the old values for the variables *SR_S* and *P* are replaced by the new ones (*New_SR_S* and *New_P*). The function **Best_Angle** and sub-function **Sort_SR** are executed again in lines 72 and 73, respectively. In lines 74 and 75, the counters *x* and *y* are set to zero.

A conditional loop, such as the one in lines 57 to 81, carries the possibility of not fulfilling the set condition. This results in a loop that is infinite. Therefore, in order to avoid this, the code in lines 78 to 80 is used to limit the number of times the conditional loop is run. The output of the function is assigned in line 82.

The sub-function **Sort_SR** is contained in lines 83 to 90. It is used to determine the minimum strength ratios for each layer, sort these values in ascending or descending order, and store the position of the values prior to sorting. The code in line 84 is used to calculate the *SR* values for each layer of the laminate. From these values, the minimum SR value for each layer is found (line 85) and stored in the array *SR_Layer_Min*. If the maximum value in this array is greater than 1.2 (line 86), then the array *SR_Layer_Min* is sorted in descending order (line 87), otherwise it is sorted in ascending order (line 89).

Function: Best_Angle

```
91  function Angg = Best_Angle(Lam_Dat,Angle,SR_S,P)
%Increase/decreases SR depending on value

92  N = length(Angle);
93  for w = 1:N
94      if SR_S(w) > 1.2
95          Angle(P(w)) = Decrease_SR(Lam_Dat,Angle,SR_S(w),P(w));
96      elseif SR_S(w) < 1
97          Angle(P(w)) = Increase_SR(Lam_Dat,Angle,SR_S(w),P(w));
98      end
99  end
100 Angg = Angle;

101 function New_angle = Decrease_SR(Lam_Dat,Angle,min_sr,pos)
```

```
%Decreases SR to be > 1 and < 1.2
102 ang = Angle(pos);
103 for A = -89:90
104     Angle(pos) = A;
        %Determines the strength ratio
105     [SR,~] = S_R_Matrix(Lam_Dat,Angle);
        %Finds smallest strength ratio per layer and sorts
         descending
106     SR_Layer = min(SR,[],2);
107     SR_Lay = sort(SR_Layer,'descend');
        %Compares new and previous strength ratios
108     if (SR_Lay(pos) < min_sr) && (SR_Lay(pos) > 1)
109         min_sr = SR_Lay(pos);
110         ang = A;
111     end
112 end
113 New_angle = ang;

114 function New_angle = Increase_SR(Lam_Dat,Angle,min_sr,pos)
%Increases SR to be > 1 and < 1.2
115 ang = Angle(pos);
116 for A = -89:90
117     Angle(pos) = A;
        %Determines the strength ratio
118     [SR,~] = S_R_Matrix(Lam_Dat,Angle);
        %Finds smallest strength ratio per layer and sorts
         ascending
119     SR_Layer = min(SR,[],2);
120     SR_Lay = sort(SR_Layer,'ascend');
        %Compares new and previous strength ratios
121     if SR_Lay(pos) > min_sr
122         min_sr = SR_Lay(pos);
123         ang = A;
124     end
125 end
126 New_angle = ang;
```

The inputs of this function (line 91) are the laminate data (*Lam_Dat*) (which includes the [Q] matrix, fixed thickness value, loading vector, and parameters for the Tsai-Wu failure criterion), the fibre orientation angles (*Angle*), the sorted minimum *SR* value of each layer (*SR_S*), and the original position of the sorted values (*P*). The **Best_Angle** function contains two sub-functions, namely, **Decrease_SR** and **Increase_SR**.

In line 92 the number of fibre layers is determined. The coded loop from line 93 to 99 is run for each layer in the laminate. If the value of the variable *SR_S* is greater than 1.2 (line 94), then the sub-function **Decrease_SR** is executed (line 95), otherwise the

sub-function **Increase_SR** is run (line 97). The output of the function (*Angg*) is given by line 100.

The sub-function **Decrease_SR** is contained in lines 101 to 113, and is run for each angle (layer) of the laminate in turn. The inputs (line 101) include the laminate data (*Lam_Dat*), the fibre orientation angles (*Angle*), the minimum *SR* value previously calculated for the angle under consideration (*min_sr*), and the position of the minimum *SR* value (*pos*). The output (*New_angle*) is the angle with the best *SR* value.

First, the variable *ang* is assigned the value of the angle under consideration (line 102). Next, in lines 103 to 112, the angle under consideration is varied between -89° and 90°. For each of these angles, the *SR* values are determined (line 105), the minimum *SR* value in each layer is found (line 106), and, these minimum values are sorted in descending order and stored using the variable *SR_Lay* (line 107). Then, in line 108, the *SR* value, at the position *pos* in the variable *SR_Lay*, is tested to determine whether it is less than the previous minimum *SR* value (*min_sr*) and also greater than 1. If this is the case, then the variable *min_sr* takes on the value of the *SR* value at the position *pos* in the variable *SR_Lay* (line 109), and the variable *ang* assumes the value of the angle that produced the new minimum *SR* value for the variable *min_sr* (line 110). The output of the sub-function is set in line 113.

The code for the sub-function **Increase_SR** is in lines 114 to 126, and is similar to that for the sub-function **Decrease_SR**. The inputs and output are the same (line 114), and the angle under consideration is also varied between -89° and 90° (line 116). As before, for each angle, the *SR* values are determined (line 118), and the minimum *SR* value in each layer is found (line 119). However, here the minimum values are sorted in ascending rather than descending order (line 120). These values are stored using the variable *SR_Lay*.

If, in line 121, the *SR* value, at the position *pos* in the variable *SR_Lay*, is greater than the previous minimum *SR* value (*min_sr*), then the variable *min_sr* takes on the value of the *SR* value at the position *pos* in the variable *SR_Lay* (line 122), and the variable

ang assumes the value of the angle that produced the new minimum *SR* value for the variable *min_sr* (line 123). The output of the sub-function is given in line 126.

This completes the analysis of the FAO code. The next developed code discussed in Chapter 5 was the ATO code. Below is the script file and the necessary functions for this code.

Main script file: Angle-Thick-Opt (ATO)

```
%ATO code: Variable angle and thickness
%Inputs: material properties, material limits, loading conditions
%Outputs: fibre angles, layer thicknesses, number of layers

127 disp('ANGLE-THICK-OPT CODE ')
128 disp('INPUT: Material Properties & Loading Conditions')
129 disp('OUTPUT: No. of Layers, Fibre Angles & Layer Thicknesses')

%Obtains material properties, allowable stresses
130 Mat_prop = [38.6 8.27 4.14 0.26 0.056];    %Glass/Epoxy
131 Input_limits = [1062 610 31 118 72];      %Glass/Epoxy
132 Mat_data{1} = Mat_prop;
133 Mat_data{2} = Input_limits;

%Calculation of material dependent variables
    %Calculate Q matrices
134 Q = Q_matrix(Mat_data{1});
    %Calculate Tsai-Wu parameters
135 T_W_Par = Tsai_Wu_Par(Mat_data{2});

%Forces and moments vector
136 NM = [1000;1000;0;0;0;0];

137 Thickness = 1;
138 Lam_Data = {Q,Thickness,NM,T_W_Par};

%Calculation of variables that depend on fibre angle
    %Initial fibre angles
139 Ply_Angle = 0;
    %Calculates T,Qbar,h,z,A,B,D,Ep0-K,global and local stresses
    %and strains, and determines the strength ratio
140 [SR,~] = S_R_Matrix(Lam_Data,Ply_Angle);

%Finds minimum SR per layer and then min and max of that
141 SR_Layer = min(SR,[],2);
142 SR_Min = min(SR_Layer);
143 SR_Max = max(SR_Layer);

144 while (SR_Min < 1) || (SR_Max > 1.2)
```

```
        %Change angles to obtain appropriate SR on each layer
145     Ply_Angle = Optimise_Angles(Lam_Data,Ply_Angle);
146     [Lam_Data,Ply_Angle] =
                    Optimise_Thickness(Lam_Data,Ply_Angle);
147     [SR,~] = S_R_Matrix(Lam_Data,Ply_Angle);
        %Finds minimum SR per layer and then min and max of that
148     SR_Layer = min(SR,[],2);
149     SR_Min = min(SR_Layer);
150     SR_Max = max(SR_Layer);
        %Checks whether min SR is < 1 or max SR is > 1.2
151     if (SR_Min < 1) || (SR_Max > 1.2)
152         index = length(Ply_Angle)+1;
153         Ply_Angle(index) = 0;
154         Lam_Data{2}(index) = Thickness;
155     end
156 end

157 Ply_Angle
158 SR
159 disp('Complete')
```

The above script file is identical to that for the FAO code with the exception of the line that executes the thickness optimisation function. The name, inputs and outputs of the code are displayed on the screen using lines 127 to 129, the material properties and limits are assigned to the array *Mat_Data* via the code in lines 130 to 133, the [Q] matrix and the parameters for the Tsai-Wu failure criterion are determined by lines 134 to 135, and the loading vector is initialised in line 136. For the ATO code, the layer thickness is not fixed but does require an initial (one-layer thickness) value, as discussed in Chapter 5. This parameter is set by the code in line 137. Line 139 is used to assign the initial value for the fibre orientation angle.

The data necessary for determining the *SR* values ([Q] matrix, one-layer thickness value, the loading vector, and parameters for the Tsai-Wu failure criterion) are stored in the array *Lam_Data* using line 138. The strength ratios at the top, middle and bottom of each layer in the laminate are calculated using the function **S_R_Matrix** (line 140). Thereafter the minimum *SR* value for each layer is evaluated (line 141), and the minimum and maximum of these values are found and stored using the variables *SR_Min* and *SR_Max*, respectively (lines 142 to 143).

If the value of *SR_Min* is less than 1 or the value of *SR_Max* is greater than 1.2 (line 144), then the function **Optimise_Angles** is implemented in line 145. Once this has completed, the function **Optimise_Thickness** is run (line 146) in order to optimise the layer thicknesses of the laminate. Since the fibre orientation angles and layer thicknesses have changed, the execution of the function **S_R_Matrix** is repeated (line 147). New *SR* values are calculated based on the changed angles and thicknesses. The code in lines 148 to 150 is used to assign new values to the variables *SR_Min* and *SR_Max*. If these values are not within the specified range (line 151), then a new layer is added on (line 152) with the initial fibre orientation angle (line 153) and the one-layer thickness value (line 154).

The functions **Q_matrix** and **Tsai_Wu_Par** were discussed in Appendix B, and the functions **S_R_Matrix** and **Optimise_Angles** were analysed earlier. The examination of the function **Optimise_Thickness** follows.

Function: Optimise_Thickness

```
160 function [Lam_Dat,Angle] = Optimise_Thickness(Lam_Dat,Angle)
%Combines layers that are within 10 deg of each other

161 NN = length(Angle);
%Find SR matrix and min value per layer
162 [SR,~] = S_R_Matrix(Lam_Dat,Angle);
163 SR_Layer_Min = min(SR,[],2);
164 N = length(Angle);
165 t = 1;

166 while (N > 1) && (t < N)
167     t = t + 1;
168     Diff = Angle(t) - Angle(t-1);
169     if abs(Diff) <= 10
170         Thick = Lam_Dat{2}(t) + Lam_Dat{2}(t-1);
171         if SR_Layer_Min(t) >=  SR_Layer_Min(t-1)
172             Angle(t-1) = '';
173             Lam_Dat{2}(t) = Thick;
174             Lam_Dat{2}(t-1) = '';
175         else %if SR_Layer_Min(t) < SR_Layer_Min(t-1)
176             Angle(t) = '';
177             Lam_Dat{2}(t-1) = Thick;
178             Lam_Dat{2}(t) = '';
179         end
180         [SR,~] = S_R_Matrix(Lam_Dat,Angle);
```

```
181         SR_Layer_Min = min(SR,[],2);
182         N = length(Angle);
183         t = 1;
184     end
185 end

186 if N ~= NN
187     Angle = Optimise_Angles(Lam_Dat,Angle);
188 end
```

As discussed in Chapter 5, the role of this function is to combine the thickness values of adjacent fibre layers if certain criteria are met. The inputs of the function (line 160) are the current laminate data (*Lam_Dat*) and fibre orientation angles (*Angle*), and the outputs include the modified laminate data and fibre orientation angles.

In line 161, the number of fibre layers (*NN*), which represents the original number of layers, is determined. Next (line 162), the *SR* values at the top, middle and bottom of each layer are calculated, and the minimum value for each layer is found (line 163). The variable *N*, in line 164, is assigned the value of the number of fibre layers. This value will change as the layers are added together. A counter (*t*), which keeps count of the number of layers that were optimised, is initialised in line 165.

Thickness optimisation can only be performed if there is more than one layer present. The first part of the conditional statement in line 166 ensures that the optimisation process is run if there is more than one layer present, or in other words, if the value of *N* is greater than 1. The second part of the statement is used as a safeguard so as to prevent runtime errors from occurring by trying to optimise more layers than there really are, that is, more layers than the value of *N*.

The next part of the code (lines 168 to 179) may be best explained by considering two arbitrary adjacent layers, namely, Layer A and Layer B. First the difference in the angles of the two layers is determined (line 168). If this difference is within 10° (line 169) then the thickness value of the two layers are added together and stored using the variable *Thick* (line 170). Next (line 171), it is determined whether the minimum *SR* value for Layer B is greater than or equal to that for Layer A. If this is the case, the

angle representing Layer A is removed from the array *Angle* (line 172), the thickness of Layer B is set to the value of the variable *Thick* (line 173), and the thickness value for Layer A is removed from the array *Lam_Dat* (line 174). On the other hand, if the minimum *SR* value for Layer B is less than that for Layer A (line 175), then the code in line 176 is used to remove the angle representing Layer B from the array *Angle*. Further, the thickness of Layer A is set to the value of the variable *Thick* (line 177), and the thickness value for Layer B is removed from the array *Lam_Dat* (line 178).

The *SR* values for each layer are recalculated using the amended fibre orientation angles and layer thicknesses (line 180). The minimum *SR* value, for each layer, is then found in line 181, and, in line 182, the amended number of layers (*N*) is determined. The counter *t* is reset in line 183. The code in lines 166 to 185 is repeated until all layer pairs have been analysed and no further thickness optimisation is possible.

Once the thickness optimisation process is completed, the current number of fibre layers (*N*) is compared to the original value (*NN*). If they are not equal (line 186) then it means that the fibre orientation angles and layer thicknesses of the laminate have changed. In this case the angle optimisation function **Optimise_Angles** is executed (line 187).

The ATO code optimises the fibre orientation angles and layer thicknesses in a two-dimensional structure. As mentioned in Chapter 5, this did not satisfy the research objectives of the study and hence the VSM code was developed.

Main script file: Variable Stiffness Method (VSM)

```
%Determines the fibre layup parameters for each element in a
structure

189 disp('VARIABLE STIFFNESS DESIGN ASSISTANT')

%Obtains material properties, allowable stresses
190 Mat_Prop = [181 10.3 7.17 0.28 0.016];     %Graphite
191 Mat_Limits = [1500 1500 40 246 68];        %Graphite
192 Mat_Thick = 1;

193 fname = 'plate_graphite-0.op2';
```

```
194 [Results,FEM] =
    Angle_Output_3D_Structure(Mat_Prop,Mat_Limits,Mat_Thick,fname);
195 Name = 'Graphite-Epoxy';
196 Material = zeros(1,17);
197 Material(1) = 45;
198 Material(2:6) = Mat_Prop;
199 Material(7:11) = Mat_Limits;
200 Material(17) = Mat_Thick;

201 save('plate_graph-01','Name','Material','Results','FEM');
202 clc
```

The first line (line 189) in the above script file is used to display the name of the code on the screen. Next the material properties, materials limits and one-layer thickness value are stored (lines 190 to 192) using the variables *Mat_Prop*, *Mat_Limits* and *Mat_Thick*, respectively. For the VSM code, the loading conditions are obtained from the Patran environment via an OP2 file. The name of this file is stipulated in line 193. The code in line 194 executes the function **Angle_Output_3D_Structure** which is used to manage the implementation of the ATO code for each element in the structure.

The material name is assigned to the variable *Name* in line 195, and, in lines 197 to 200, the material parameters (set in lines 190 to 192) are stored using the variable *Material*. The code in line 201 is used to write the contents of the variables *Name* and *Material* as well as the outputs from the function **Angle_Output_3D_Structure** to a file.

Function: Angle_Output_3D_Structure

```
203 function [Results,FEM] =
    Angle_Output_3D_Structure(Mat_Prop,Mat_Limits,Mat_Thick,fname)
%Obtains loading conditions from Patran.
%Determines fibre layup parameters for each element in 3D structure

%Calculate Q matrix
204 Q = Q_matrix(Mat_Prop);
%Calculate Tsai-Wu parameters
205 T_W_Par = Tsai_Wu_Par(Mat_Limits);
%Calculates resultant forces and moments (NM vector matrix)
206 [FEM,NM] = Nastran_Data(fname);

%Determine fibre layup and SR for each layer in each element
207 [row,~] = size(NM);
208 h = waitbar(0,['Completed Element 0 of ',num2str(row)],
```

```
        'Name','Please wait...');

for w = 1:row
    q = 1;
    Skip = 0;
    while q < w
        if NM(w,:) == NM(q,:)
            Results(w,1) = Results(q,1);
            q = w;
            Skip = 1;
        else
            q = q + 1;
        end
    end
    if Skip == 0
        Lam_Data = {Q,Mat_Thick,NM(w,:)',T_W_Par};
        [Ply_Angle,Ply_Thick,SR,GL_LL_S_S] =
            Angle_Output_2D_Element(Lam_Data);
        Results(w,1) = struct('Angle',{Ply_Angle},'Thick',
            {Ply_Thick},'SR',{SR},'GL_LL_S_S',{GL_LL_S_S},
            'NM',{NM(w,:)});
    end
    waitbar(w/row,h,['Completed Element ',num2str(w),' of  '
        ,num2str(row)]);
end
close(h)
```

The above function has four inputs (line 203), namely, the material properties (*Mat_Prop*), material limits (*Mat_Limits*), one-layer thickness value (*Mat_Thick*), and the name of the Patran OP2 file (*fname*). The outputs include the optimised parameters as well as other laminate data for each element (*Results*), and the finite element geometry data from the OP2 file (*FEM*).

The first step is to determine the [Q] matrix and Tsai Wu parameters using the functions **Q_matrix** and **Tsai_Wu_Par**, respectively (lines 204 to 205). These functions are the same as those in Appendix B. Next, in line 206, the function **Nastran_Data** is executed in order to obtain the geometry data of the finite element model (*FEM*) and the loading vector (*NM*) from the OP2 file. The geometry data includes the positions of the nodes, the nodes that constitute each element, and the shape of each element. The number of elements present in the finite element model is equal to the number of rows in the array *NM*.

A message box is displayed, using the code in line 208, which informs the user on the progress of the optimisation process. A coded loop is run from line 209 to 227 for each element in the structure. For each element, the optimised fibre layup parameters as well as the global and local stresses and strains are determined according to that element's loading conditions. The set of loading conditions, for the element under consideration, are compared to those which have already been used in the optimisation process (for the current structure). If a corresponding set is found, then the results of the current element take on the values of the matching element. This reduces the design time and enables the code to be more efficient.

In the first line of the coded loop (line 210), a variable *q*, which is used as a counter, is initiated. Next (line 211), the variable *Skip*, which is used to indicate whether corresponding loading conditions were found, is created. The coded loop, from line 212 to 220, is used to determine whether the current loading conditions are identical to any of those for the previous elements. If this is the case (line 213), then the results for the current element are made equal to that of the corresponding element (line 214), and the variable *Skip* is given a value of '1' (line 216) to indicate that a matching element has been found.

If the value of the variable *Skip* is zero (line 221), then, in line 223, the function **Angle_Output_2D_Element** is executed. This function is used to manage the angle and thickness optimisation processes for the element under consideration. The results for the current element, which include the fibre orientation angles, layer thicknesses, strength ratios, global and local stresses and strains, and the loading vector, are stored using the variable *Results* via line 224. The code in line 226 is used to update the message box that was initiated in line 208, and this message box is closed with the code in line 228.

Function: Nastran_Data

```
229 function [FEM,NM] = Nastran_Data(fname)
%Find resultant forces & moments, & FEM data from NASTRAN op2 file
```

```
%Obtain data from Nastran op2 file
230 [Fdata,~,~] =
    readnas(fname,'blocks',{'GEOM1','GEOM2'},'silent');
231 [Rdata,~,info] =
    readnas(fname,'blocks',{'OEF1X'},'asraw','silent');

%Extract the element forces
232 ind   = 2:2:numel(Rdata.table.OEF1X.record);
233 cc    = cell(numel(ind),1);
234 loads = struct('phys',cc);

%Process the element loads from the OP2
235 for k = 1:numel(ind)
236     eload = reshape(Rdata.table.OEF1X.record{ind(k)},47,[]).';
        %Convert appropriately
237     loads(k).phys  = double(typecast_op2(eload(:,11+(1:36)),
            'single',info.endianswap));
238 end

%Assign other data imported from the op2 file
239 Elements = cell2mat(Fdata.fem.elem.conn);
240 Node_Force = loads.phys;

%Initialise variables
241 [rowE,~] = size(Elements);
242 nm = zeros(rowE,6);

%Calculate resultant forces and moments
243 for p = 1:rowE
244     Fx = (Node_Force(p,2) + Node_Force(p,11) + Node_Force(p,20)
            + Node_Force(p,29))/4;
245     Fy = (Node_Force(p,3) + Node_Force(p,12) + Node_Force(p,21)
            + Node_Force(p,30))/4;
246     Fxy = (Node_Force(p,4) + Node_Force(p,13) +
            Node_Force(p,22) + Node_Force(p,31))/4;
247     Mx = (Node_Force(p,5) + Node_Force(p,14) + Node_Force(p,23)
            + Node_Force(p,32))/4;
248     My = (Node_Force(p,6) + Node_Force(p,15) + Node_Force(p,24)
            + Node_Force(p,33))/4;
249     Mxy = (Node_Force(p,7) + Node_Force(p,16) +
            Node_Force(p,25) + Node_Force(p,34))/4;
250     nm(p,:) = [Fx Fy Fxy Mx My Mxy];
251 end

%Outputs
252 FEM = Fdata.fem;
253 NM = nm;
```

This function is used to obtain the geometry data from the OP2 file as well as to associate each element with its loading conditions. The input of the above function

(line 229) is the name of the Patran OP2 file (*fname*), while the outputs include the geometry data (*FEM*) and the loading vector array (*NM*). The code in lines 230 and 231 uses the function **readnas** to import data from the OP2 file into Matlab. This function is proprietary software from ATA Engineering and therefore cannot be discussed here. The geometry data is imported via line 230 and stored using the array *Fdata*, while, in line 231, the array *Rdata* is used to store the data for the imported loading conditions.

The loading conditions data contains different loading systems that include the applied loads, constraint forces, nodal forces, nodal moments, etc. As mentioned in Chapter 5, the nodal forces and moments are required in order to associate each element with its loading conditions. Therefore, the code in lines 232 to 234 is used to extract these forces and moments from the array *Rdata*, and store them using the array *loads*. This array is then reshaped (lines 235 to 238) in order to arranged the forces and moments into a more useable format. The forces and moments, for each element, are determined in lines 243 to 251. The outputs of the function **Nastran_Data**, namely, the geometry data (*FEM*) and the loading vector (*NM*), are given by lines 252 and 253.

Function: Angle_Output_2D_Element

```
254 function [Ply_Angle,Ply_Thick,SR,GL_LL_S_S] =
    Angle_Output_2D_Element(Lam_Data)
%Determines fibre angles, thicknesses, stresses and strains, SR
values for a particular element of a 3D structure

255 Thick = Lam_Data{2}(1);
256 Ply_Angle = 0;

%Calculates T,Qbar,h,z,A,B,D,Ep0-K,global/local stresses/strains, SR
257 [SR,~] = S_R_Matrix(Lam_Data,Ply_Angle);

%Finds minimum SR per layer and then min and max of that
258 SR_Layer = min(SR,[],2);
259 SR_Min = min(SR_Layer);
260 SR_Max = max(SR_Layer);

261 while (SR_Min < 1) || (SR_Max > 1.2)
        %Change angles to obtain appropriate SR on each layer
262     Ply_Angle = Optimise_Angles(Lam_Data,Ply_Angle);
263     AngOp = 1;
```

```
264     while AngOp == 1
265         [Lam_Data,Ply_Angle,AngOp] = Optimise_Thickness
                (Lam_Data,Ply_Angle);
266     end
267     [SR,~] = S_R_Matrix(Lam_Data,Ply_Angle);
        %Finds minimum SR per layer and then min and max of that
268     SR_Layer = min(SR,[],2);
269     SR_Min = min(SR_Layer);
270     SR_Max = max(SR_Layer);

        %Checks whether min SR is < 1 or max SR is > 1.2
271     if (SR_Min < 1) || (SR_Max > 1.2)
272         index = length(Ply_Angle)+1;
273         Ply_Angle(index) = 0;
274         Lam_Data{2}(index) = Thick;
275     end
276 end
277 Ply_Thick = Lam_Data{2};
278 [~,GL_LL_S_S] = S_R_Matrix(Lam_Data,Ply_Angle);
```

The above function essentially replaces the script file for the ATO code and is run for each element of the structure in turn. This function has one input (line 254), namely, the laminate data (*Lam_Data*), which includes the [Q] matrix, Tsai-Wu parameters, one-layer thickness value, and the loading conditions for a particular layer. The outputs include the optimised fibre orientation angles (*Ply_Angle*), optimised thicknesses (*Ply_Thick*), strength ratios (*SR*), and, the global and local stresses and strains (*GL_LL_S_S*).

The initial values for the layer thickness and fibre orientation angle are set in line 255 and 256, respectively. In line 257, the strength ratios (*SR* values) for the top, middle and bottom of each layer are calculated using the function **S_R_Matrix**. The minimum *SR* value per layer is found (line 258), and then the minimum (*SR_Min*) and maximum (*SR_Max*) of these values are determined (lines 259 and 260).

The conditional loop from line 261 to 276 is run if the value of the variable *SR_Min* is less than 1 or if the value of the variable *SR_Max* is greater than 1.2. The function **Optimise_Angles** is executed (line 262) in order to obtain *SR* values that are closer to the specified range (between 1 and 1.2). In line 263, the variable *AngOp* is assigned a value of '1', which indicates that the angle optimisation process was run. The thickness

optimisation function (**Optimise_Thickness**) is executed in line 265, and, in line 267, the *SR* values, based on the amended fibre orientation angles and layer thicknesses, are determined.

New values are then found for the variables *SR_Min* and *SR_Max* (lines 268 to 270). If the new values for these variables are not in the range between 1 and 1.2 (line 271), then a new layer is added (line 272) with an initial fibre orientation angle of 0° (line 273) and a layer thickness equal to the one-layer thickness value (line 274).

The output arrays *Ply_Thick* and *GL_LL_S_S* are assigned their values in lines 277 and 278, respectively. The functions **S_R_Matrix**, **Optimise_Angles** and **Optimise_Thickness** have been discussed earlier. The last aspect of the VSM code that has to be examined is the Graphical User Interface.

Graphical User Interface (GUI) code

In Appendix B it was stated that a GUI consists of a figure file, which contains the graphical layout, and a code file, which includes the functions and commands for the operation of the GUI. The figure file for the VSM code was shown in Figure 5.23, while the code file is shown below.

```
1    function varargout = GUI_Fibre_Angle_Output(varargin)
% GUI_FIBRE_ANGLE_OUTPUT M-file for GUI_Fibre_Angle_Output.fig
%      GUI_FIBRE_ANGLE_OUTPUT, by itself, creates a new
GUI_FIBRE_ANGLE_OUTPUT or raises the existing singleton*.
%
%      H = GUI_FIBRE_ANGLE_OUTPUT returns the handle to a new
GUI_FIBRE_ANGLE_OUTPUT or the handle to the existing singleton*.
%
GUI_FIBRE_ANGLE_OUTPUT('CALLBACK',hObject,eventData,handles,...)
calls the local function named CALLBACK in GUI_FIBRE_ANGLE_OUTPUT.M
with the given input arguments.
%
%      GUI_FIBRE_ANGLE_OUTPUT('Property','Value',...) creates a new
GUI_FIBRE_ANGLE_OUTPUT or raises the existing singleton*.  Starting
from the left, property value pairs are applied to the GUI before
GUI_Fibre_Angle_Output_OpeningFcn gets called.  An unrecognized
property name or invalid value makes property application stop.  All
inputs are passed to GUI_Fibre_Angle_Output_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
GUI_Fibre_Angle_Output

% Last Modified by GUIDE v2.5 27-Jan-2011 12:43:18
% Begin initialization code - DO NOT EDIT
2    gui_Singleton = 1;
3    gui_State = struct('gui_Name',       mfilename, ...
     'gui_Singleton',  gui_Singleton, ...
     'gui_OpeningFcn', @GUI_Fibre_Angle_Output_OpeningFcn, ...
     'gui_OutputFcn',  @GUI_Fibre_Angle_Output_OutputFcn, ...
     'gui_LayoutFcn',  [] , ...
     'gui_Callback',   []);
4    if nargin && ischar(varargin{1})
5        gui_State.gui_Callback = str2func(varargin{1});
6    end

7    if nargout
8        [varargout{1:nargout}] = gui_mainfcn(gui_State,
             varargin{:});
9    else
10       gui_mainfcn(gui_State, varargin{:});
11   end
% End initialization code - DO NOT EDIT

% --- Executes just before GUI_Fibre_Angle_Output is made visible.
12   function GUI_Fibre_Angle_Output_OpeningFcn(hObject, eventdata,
     handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure_fibre_angle_output
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_Fibre_Angle_Output (see
VARARGIN)

% Choose default command line output for GUI_Fibre_Angle_Output
13   handles.output = hObject;
14   M_M = find(strcmp(varargin, 'MainMenu'));
15   handles.MainMenuVar = varargin{M_M+1};
16   handles.HelpVal = 0;
% Update handles structure
17   guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
18   function Pict_Axes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Pict_Axes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: place code in OpeningFcn to populate Pict_Axes
19   axes(hObject)
```

```
20   imshow('Layers.jpg')

% --- Outputs from this function are returned to the command line.
21   function varargout = GUI_Fibre_Angle_Output_OutputFcn(hObject,
     eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure_fibre_angle_output
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
22   varargout{1} = handles.output;


%---------P R O G R A M M I N G   S T A R T S   H E R E----------%
23   setCompSelectString(handles.Material_Menu,eventdata,handles);
24   set(handles.Results_Table,'Data',[]);
25   Pict_Axes_CreateFcn(handles.Pic, eventdata, handles);
26   uiwait(handles.Figure_Fibre_Angle_Output)


%--------I N P U T   P A N E L   M A T E R I A L   L I S T---------%
% --- Sets content of material drop-down list
27   function setCompSelectString(hObject,eventdata,handles)
28   MatN = load('Database-Comp.mat','Name');
29   set(hObject,'string',MatN.Name);

% --- Executes on selection change in Material_Menu.
30   function Material_Menu_Callback(hObject, eventdata, handles)
% hObject    handle to Material_Menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
Material_Menu contents as cell array contents{get(hObject,'Value')}
returns selected item from Material_Menu

% --- Executes during object creation, after setting all properties.
31   function Material_Menu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Material_Menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
32   if ispc && isequal(get(hObject,'BackgroundColor'),
     get(0,'defaultUicontrolBackgroundColor'))
33       set(hObject,'BackgroundColor','white');
34   end
```

```
%----I N P U T   P A N E L   E X E C U T A B L E   B U T T O N S---%
% --- Executes on button press in Load_File_Button.
function Load_File_Button_Callback(hObject, eventdata, handles)
% hObject    handle to Load_File_Button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[FileName,PathName] = uigetfile('*.op2','Select the op2 code
    file');
if (FileName ~= 0)
    handles.NastranFile = [PathName,FileName];
    set(handles.Filename_Box,'string',FileName);
    guidata(hObject,handles);
end

% --- Executes on button press in Fibre_Angle_Input_Panel_Help.
function Fibre_Angle_Input_Panel_Help_Callback(hObject,
    eventdata, handles)
% hObject    handle to Fibre_Angle_Input_Panel_Help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.HelpFig = Help_FibAng;
handles.HelpVal = 1;
guidata(hObject,handles);

% --- Executes on button press in Fibre_Angle_Input_Panel_Run.
function Fibre_Angle_Input_Panel_Run_Callback(hObject,
    eventdata, handles)
% hObject    handle to Fibre_Angle_Input_Panel_Run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if isfield(handles,'NastranFile')
    if strcmp(handles.NastranFile,'') == 0
        Mat_List = get(handles.Material_Menu,'String');
        Mat_Sel = get(handles.Material_Menu,'Value');
        handles.Mat_Name = Mat_List(Mat_Sel);
        Mat_Data = load('Database-Comp.mat','Data');
        handles.Material = Mat_Data.Data(Mat_Sel,:);
        Mat_Prop = Mat_Data.Data(Mat_Sel,2:6);
        Mat_Limits = Mat_Data.Data(Mat_Sel,7:11);
        Mat_Thick = Mat_Data.Data(Mat_Sel,17);
        Filename = handles.NastranFile;
        [handles.Results,handles.FEM] =
            Angle_Output_3D_Structure(Mat_Prop,Mat_Limits,
            Mat_Thick,Filename);
        guidata(hObject,handles);

        %Populate Results_Table with values
        N_El = size(handles.Results,1);
        Results_Data = cell(N_El,5);
        for v = 1:N_El
            El_Lay = length(handles.Results(v).Angle);
            El_Ang = num2str(handles.Results(v).Angle);
```

```
65              El_Thick = num2str(handles.Results(v).Thick);
66              El_NM = num2str(handles.Results(v).NM);
67              Results_Data(v,:) = {v El_Lay El_Ang El_Thick
                                     El_NM};
68          end
69          set(handles.Results_Table,'Data',Results_Data);
70      else
71          errordlg('No NASTRAN file loaded','ERROR','modal');
72      end
73  else
74      errordlg('No NASTRAN file loaded','ERROR','modal');
75  end

% --- Executes on button press in Fibre_Angle_Input_Panel_Close.
76  function Fibre_Angle_Input_Panel_Close_Callback(hObject,
    eventdata, handles)
% hObject    handle to Fibre_Angle_Input_Panel_Close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
77  Figure_Fibre_Angle_Output_CloseRequestFcn(handles.Figure_Fibre_
    Angle_Output, eventdata, handles)


%-----E L E M E N T A L   R E S U L T S   T A B L E   P A N E L----%
% --- Executes when selected cell(s) is changed in Results_Table.
78  function Results_Table_CellSelectionCallback(hObject,
    eventdata, handles)
% hObject    handle to Results_Table (see GCBO)
% eventdata  structure with the following fields (see UITABLE)
%   Indices: row and column indices of the cell(s) currently
selecteds
% handles    structure with handles and user data (see GUIDATA)
79  if numel(eventdata.Indices) ~= 0
80      EL = eventdata.Indices(1);
81      N = length(handles.Results(EL).Angle);
82      Summary_Table_Data = cell(N,3);
83      set(handles.Element_No,'String',EL);

        %Populate stress/strain tables with values
84      for r = 1:4
85          TableData = View_Stress_Strain(N, handles.Results(EL).
                                        GL_LL_S_S{r});
86          switch r
87              case (1)
88                  set(handles.G_Strain_Table,'Data',TableData);
89              case (2)
90                  set(handles.G_Stress_Table,'Data',TableData);
91              case (3)
92                  set(handles.L_Strain_Table,'Data',TableData);
93              case (4)
94                  set(handles.L_Stress_Table,'Data',TableData);
95          end
```

```
96      end
97      for t = 1:N
98          Summary_Table_Data(t,:) = {handles.Mat_Name{1}
                handles.Results(EL).Angle(t)
                handles.Results(EL).Thick(t)};
99      end
100     set(handles.SR_Table,'Data',handles.Results(EL).SR);
101     set(handles.Summary_Table,'Data',Summary_Table_Data);
102 end


%-E L E M E N T A L   R E S U L T S   E X E C U T A B L E   B U T T O N S-%
% --- Executes on button press in Fibre_Angle_Results_Panel_Discard.
103 function Fibre_Angle_Results_Panel_Discard_Callback(hObject,
    eventdata, handles)
% hObject    handle to Fibre_Angle_Results_Panel_Discard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
104 set(handles.Results_Table,'Data',[]);
105 handles.NastranFile = '';
106 guidata(hObject,handles);
107 set(handles.Filename_Box,'string','Filename.op2');
108 Fibre_Angle_Output_Panel_Clear_Callback(hObject, eventdata,
    handles);

% --- Executes on button press in Fibre_Angle_Results_Panel_Open.
109 function Fibre_Angle_Results_Panel_Open_Callback(hObject,
    eventdata, handles)
% hObject    handle to Fibre_Angle_Results_Panel_Open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
110 [FileName,PathName] = uigetfile('*.rslt','Select the results
    file');
111 if (FileName ~= 0)
112     OpenFile = [PathName,FileName];
113     RR = load(OpenFile,'-mat');
114     handles.Mat_Name = RR.Name;
115     handles.Material = RR.Material;
116     handles.Results = RR.Results;
117     handles.FEM = RR.FEM;
118     guidata(hObject,handles);

        %Populate Results_Table with values
119     N_El = size(handles.Results,1);
120     Results_Data = cell(N_El,5);
121     for v = 1:N_El
122         El_Lay = length(handles.Results(v).Angle);
123         El_Ang = num2str(handles.Results(v).Angle);
124         El_Thick = num2str(handles.Results(v).Thick);
125         El_NM = num2str(handles.Results(v).NM);
126         Results_Data(v,:) = {v El_Lay El_Ang El_Thick El_NM};
127     end
```

```
128     set(handles.Results_Table,'Data',Results_Data);
129 end

% --- Executes on button press in Fibre_Angle_Results_Panel_Save.
130 function Fibre_Angle_Results_Panel_Save_Callback(hObject,
    eventdata, handles)
% hObject    handle to Fibre_Angle_Results_Panel_Save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
131 [FileName,PathName] = uiputfile('*.rslt','Save Results As');
132 if (FileName ~= 0)
133     SaveFile = [PathName,FileName];
134     Name = handles.Mat_Name;
135     Material = handles.Material;
136     Results = handles.Results;
137     FEM = handles.FEM;
138     save(SaveFile,'Name','Material','Results','FEM');
139 end

% --- Executes on button press in Fibre_Angle_Output_Panel_Clear.
140 function Fibre_Angle_Output_Panel_Clear_Callback(hObject,
    eventdata, handles)
% hObject    handle to Fibre_Angle_Results_Panel_Discard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
141 set(handles.Element_No,'String','');
142 set(handles.G_Strain_Table,'Data',[]);
143 set(handles.G_Stress_Table,'Data',[]);
144 set(handles.L_Strain_Table,'Data',[]);
145 set(handles.L_Stress_Table,'Data',[]);
146 set(handles.SR_Table,'Data',[]);
147 set(handles.Summary_Table,'Data',[]);


%------------C L O S E   O R   E X I T   P R O G R A M-------------%
% --- Executes when user attempts to close
Figure_Fibre_Angle_Output.
148 function Figure_Fibre_Angle_Output_CloseRequestFcn(hObject,
    eventdata, handles)
% hObject    handle to Figure_Fibre_Angle_Output (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: delete(hObject) closes the figure
149 if handles.HelpVal == 1
150     if ishandle(handles.HelpFig)
151         delete(handles.HelpFig);
152     end
153 end
154 mainHandles = guidata(handles.MainMenuVar);
155 set(mainHandles.Check,'String','GO');
156 uiresume(handles.Figure_Fibre_Angle_Output)
157 delete(handles.Figure_Fibre_Angle_Output);
```

The GUI code is a function (line 1) that is able to accept multiple inputs (*varargin*) and generate multiple outputs (*varargout*). The basic functions of the GUI are initialised in lines 2 to 11. It is here that any input and output variables are recognised. As mentioned in Appendix B, the functions for the textboxes, pushbuttons, etc. usually have three inputs (*hObject*, *eventdata* and *handles*) that link the code to the figure file.

The function from line 12 to 17 is executed before the GUI is made visible. The code in lines 13 to 15, together with that in line 26, is used to prevent the execution of other GUIs from the main graphical interface. In line 16, the variable *handles.HelpVal* is created and is used to indicate whether the help function was used. The new variables and new values are stored in the GUI database via the code in line 17. The function in lines 18 to 20, along with the code in line 25, is used to display the logo. The outputs of the GUI (if any) are returned to the command line via the function in lines 21 to 22.

The drop-down list in the GUI contains the names of the materials in the composite database. These names are loaded from the database using the code in line 23 as well as the function in lines 27 to 29. The settings of the drop-down list are given by the function in lines 31 to 34. The function in line 30 is used to obtain the selected material.

The Input Panel contains four pushbuttons. The first of these is "LOAD FILE" which is used to select the OP2 file that contains the loading conditions for the current structure. When this button is pressed, the code in line 36 opens a window where the user may select the OP2 file. If a file is selected (line 37), the path and name of the file is stored using the variable *handles.NastranFile* (line 38), and the name of the OP2 file is displayed in the appropriate textbox via line 39.

The next two pushbuttons are "HELP" and "EXIT". The "HELP" button runs the function in lines 42 to 45. The help GUI is launched using line 43, and in line 44, the variable *handles.HelpVal* is assigned a value of '1' to indicate that the help facility is open. Pressing of the button labelled "EXIT" runs the function in lines 76 to 77. Here the code is redirected to the function that closes the GUI (lines 148 to 157), which is discussed later.

The last pushbutton on the Input Panel is "RUN". When this is pressed, the function from line 46 to 75 is executed. First it is determined whether an OP2 file was chosen (lines 47 and 48). If no file is chosen, then an error message is displayed indicating this (lines 71 and 74). On the other hand, if a file is chosen, then the name of the material selected from the drop-down list is obtained (lines 49 and 50), and stored using the variable *handles.Mat_Name* (line 51). In lines 52 and 53, the properties for the selected material are loaded from the composite database. The values for the material constants, material limits, one-layer thickness, and the name of the OP2 file are assigned to the variables *Mat_Prop*, *Mat_Limits*, *Mat_Thick* and *Filename*, respectively (lines 54 to 57). The function **Angle_Output_3D_Structure** is executed in line 58 to determine the optimised fibre orientation angles and layer thicknesses, strength ratios, global and local stresses and strains, loading vector, and the finite element data for each element of the structure. Some of these results (fibre orientation angles, layer thicknesses and loading vector) are displayed in the table on the Elemental Results panel via the code in lines 60 to 69.

As mentioned in Chapter 5, if any of the results/elements in the table on the Elemental Results panel is selected, additional results (global and local stresses and strains, and strength ratios) for the selected element are shown in the tables on the Additional Results panel. The function in lines 78 to 102 is used to populate these tables. If a selection is made (line 79), the position of the selected element is obtained by the code in line 80. Then the coded loop from line 84 to 96 is used to load and display the global strains, global stresses, local strains and local stresses of the selected element, from the array *Results*. Here the function **View_Stress_Strain** is used which was discussed in Appendix B. The entries of the Summary of Elemental Fibre Layup table are determined by the coded loop in lines 97 to 99, and displayed on the Additional Results panel by the code in line 101. The strength ratio values are displayed in the appropriate table using line 100.

The Elemental Results panel has four pushbuttons. The function in lines 103 to 108 is run when the "DISCARD RESULTS" button is used. When this button is pressed, the table on the Elemental Results panel is cleared of all data (line 104), the name of the

OP2 file is removed from the variable *handles.Nastranfile* (line 105), and the textbox on the Input Panel is set to its default value (line 107). Further, in line 108, the function **Fibre_Angle_Output_Panel_Clear_Callback** (lines 140 to 147), which is used to clear the entries from all the tables on the Additional Results panel, is executed. This function is also run when the button labelled "CLEAR TABLES" is pressed.

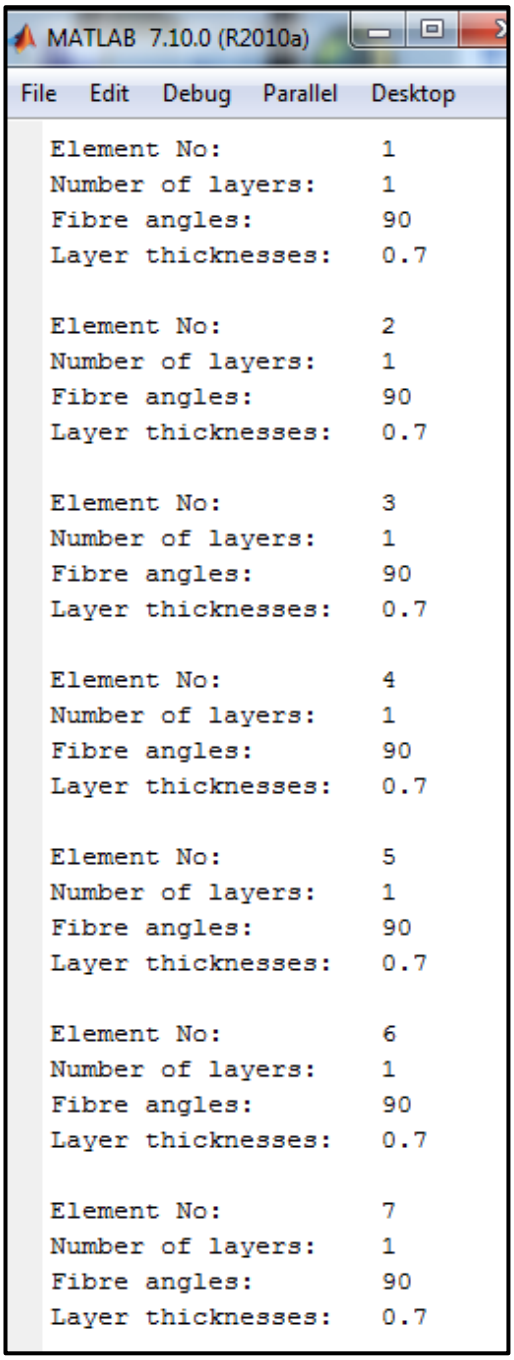
The "OPEN" button is used to load previously saved results. In line 110, a window is displayed where the user may select the results (file) to be loaded. If a file is selected (line 111), then the contents of the file are loaded (lines 112 and 113). The values for the material name, material properties, elemental results and the finite element data are assigned to the variables *handles.Mat_Name*, *handles.Material*, *handles.Results* and *handles.FEM*, respectively (lines 114 to 117). The fibre orientation angle, layer thickness and loading vector results are extracted from the variable *handles.Results* (lines 119 to 127), and displayed in the table on the Elemental Results panel via line 128. As before, if any element in this table is selected, additional results are displayed on the Additional Results panel.

The function from line 130 to 139 is executed when the "SAVE" button is pressed. This function is used to store the current results to a file. A window is opened (line 131) where the user enters the name and path of the file. If a name has been entered (line 132), then the name and path of the file is stored using the variable *SaveFile*. The material name, material properties, elemental results and finite element data are saved to the file, with the name and path stored in the variable *SaveFile*, via line 138.

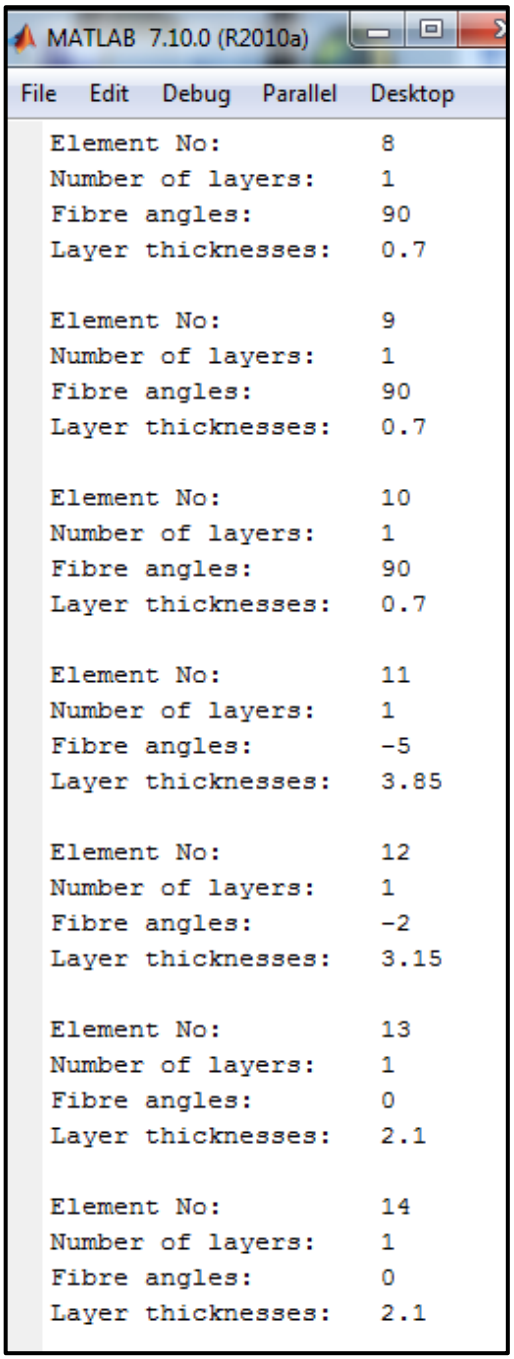
The last function (lines 148 to 157) is used to properly close the current GUI. If the help facility/window is open, that is, if the variable *handles.HelpVal* has a value of '1' (line 149), then it is closed with the code in lines 150 to 152. It was mentioned earlier that the GUI was locked to prevent the execution of other GUIs from the main graphical interface. The code in lines 154 to 156 is used to unlock the GUI. The GUI is closed via line 157.

APPENDIX D: MATLAB RESULTS FOR THE C-CHANNEL

This appendix shows the fibre layup parameters and Strength Ratio (*SR*) values obtained from the VSM code for the C-channel in Chapter 5.

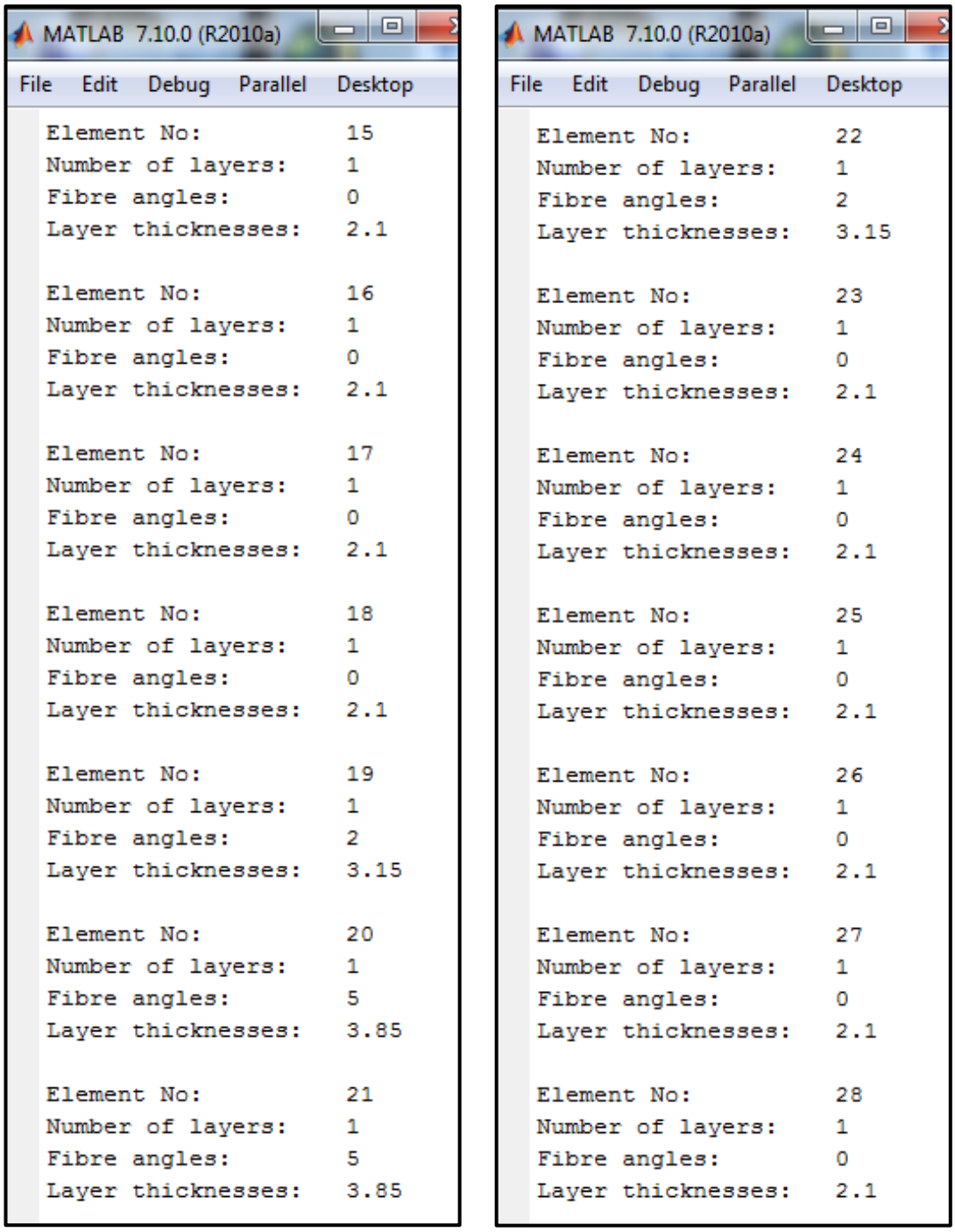


(a)



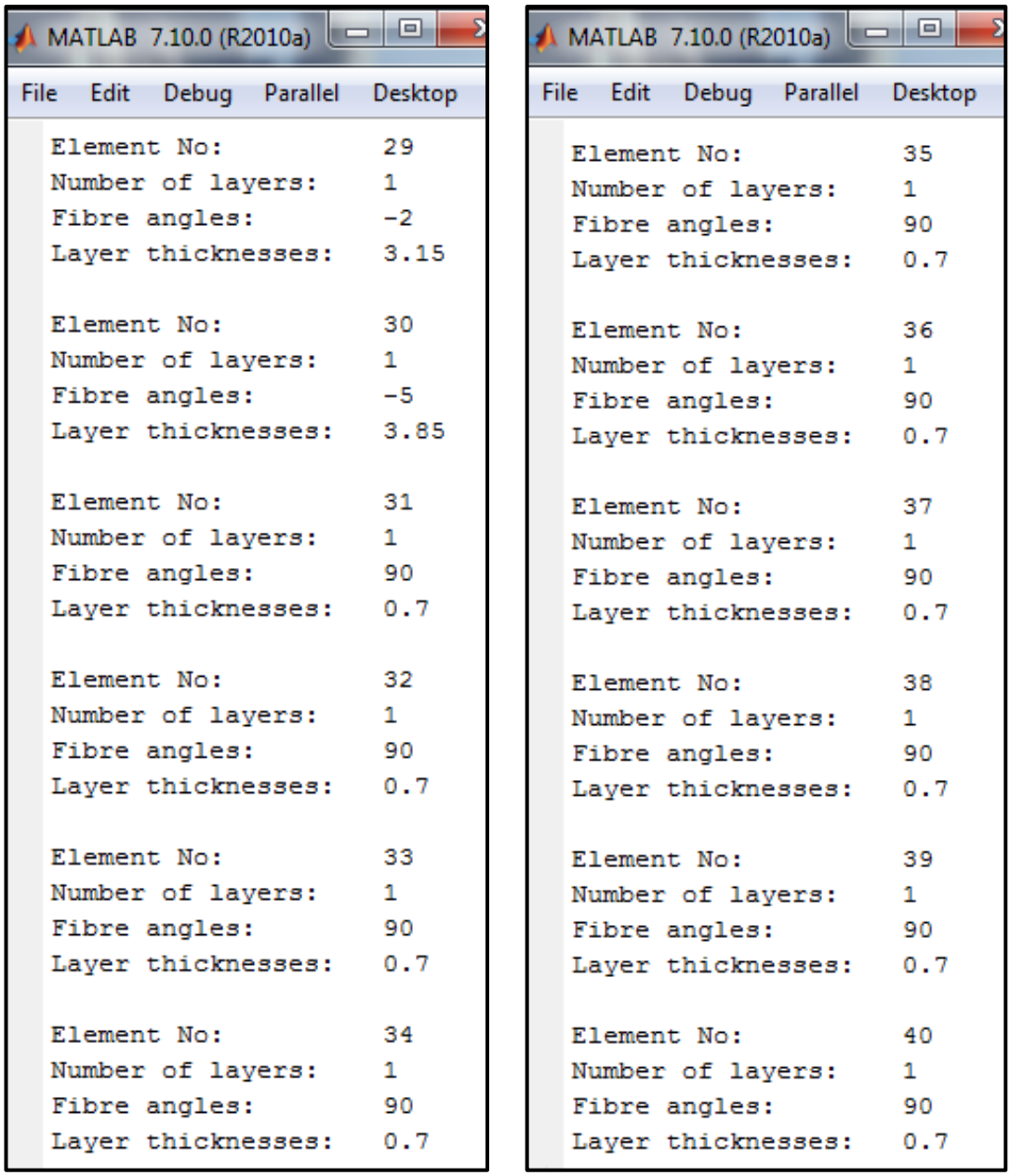
(b)

Figure D.1: C-channel fibre layup parameters for elements (a) 1 to 7, and (b) 8 to 14



(a) (b)

Figure D.2: C-channel fibre layup parameters for elements (a) 15 to 21, and (b) 22 to 28



(a) (b)

Figure D.3: C-channel fibre layup parameters for elements (a) 29 to 34, and (b) 35 to 40

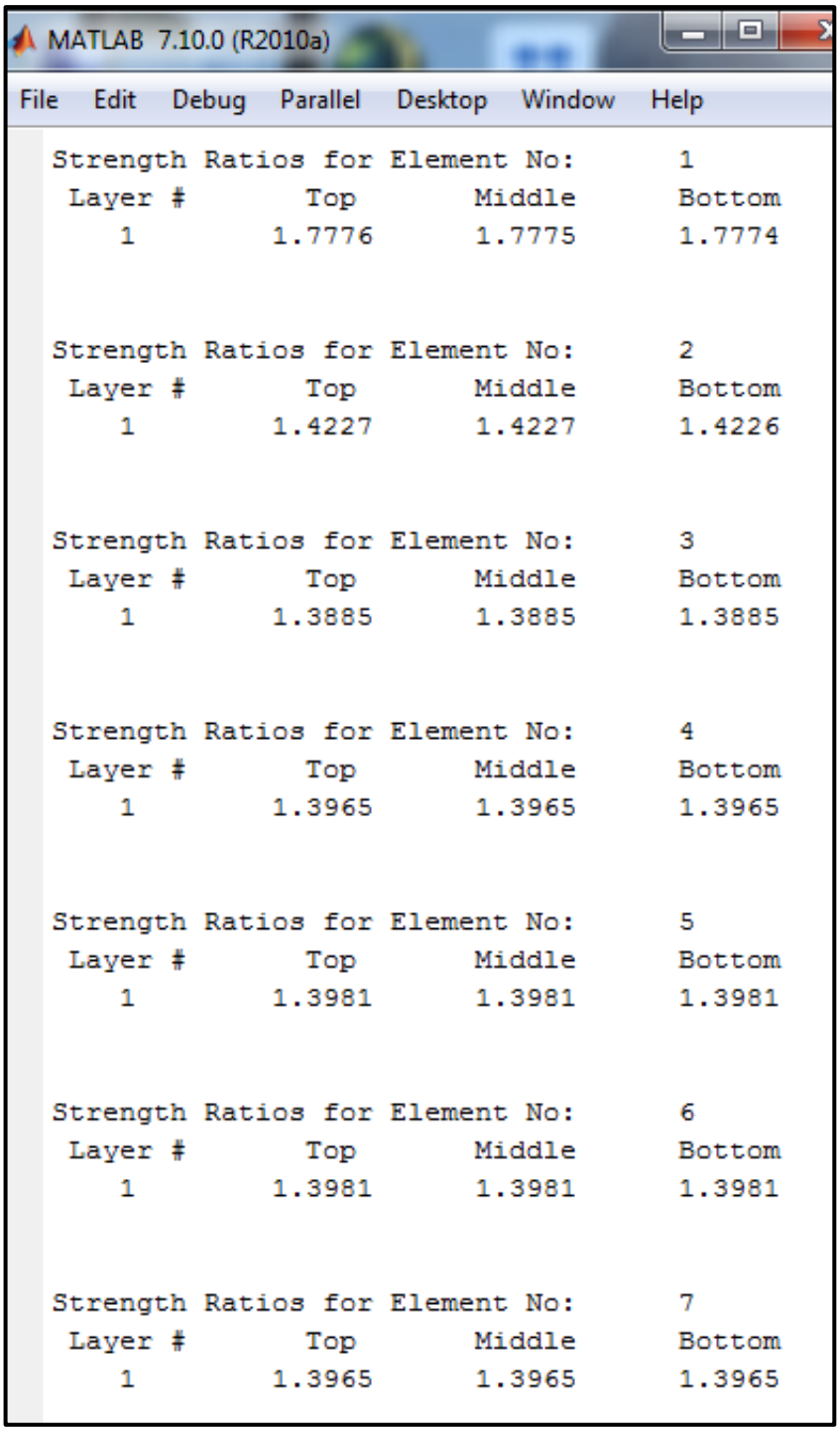


Figure D.4: Strength Ratio (*SR*) values for C-channel for elements 1 to 7

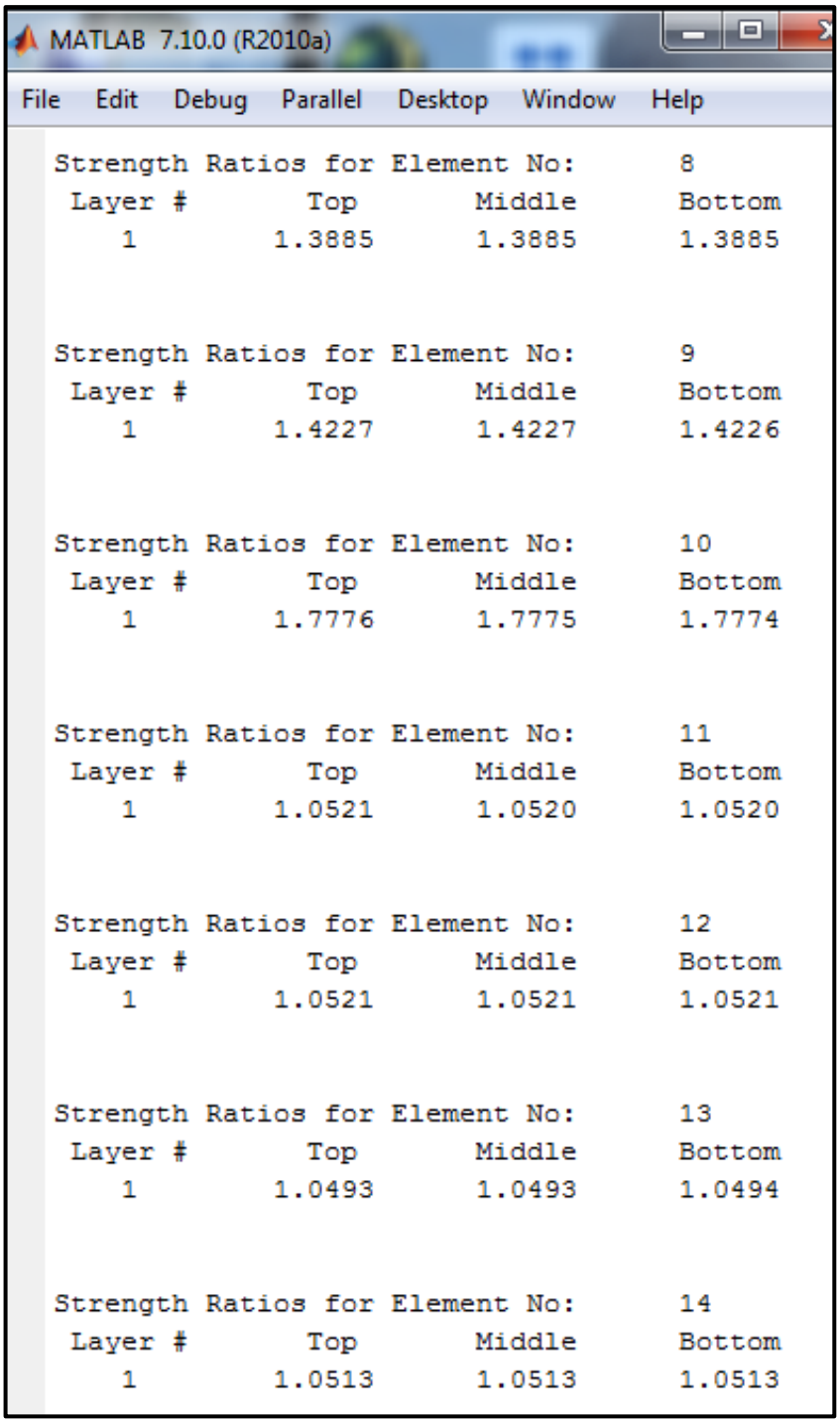


Figure D.5: Strength Ratio (*SR*) values for C-channel for elements 8 to 14

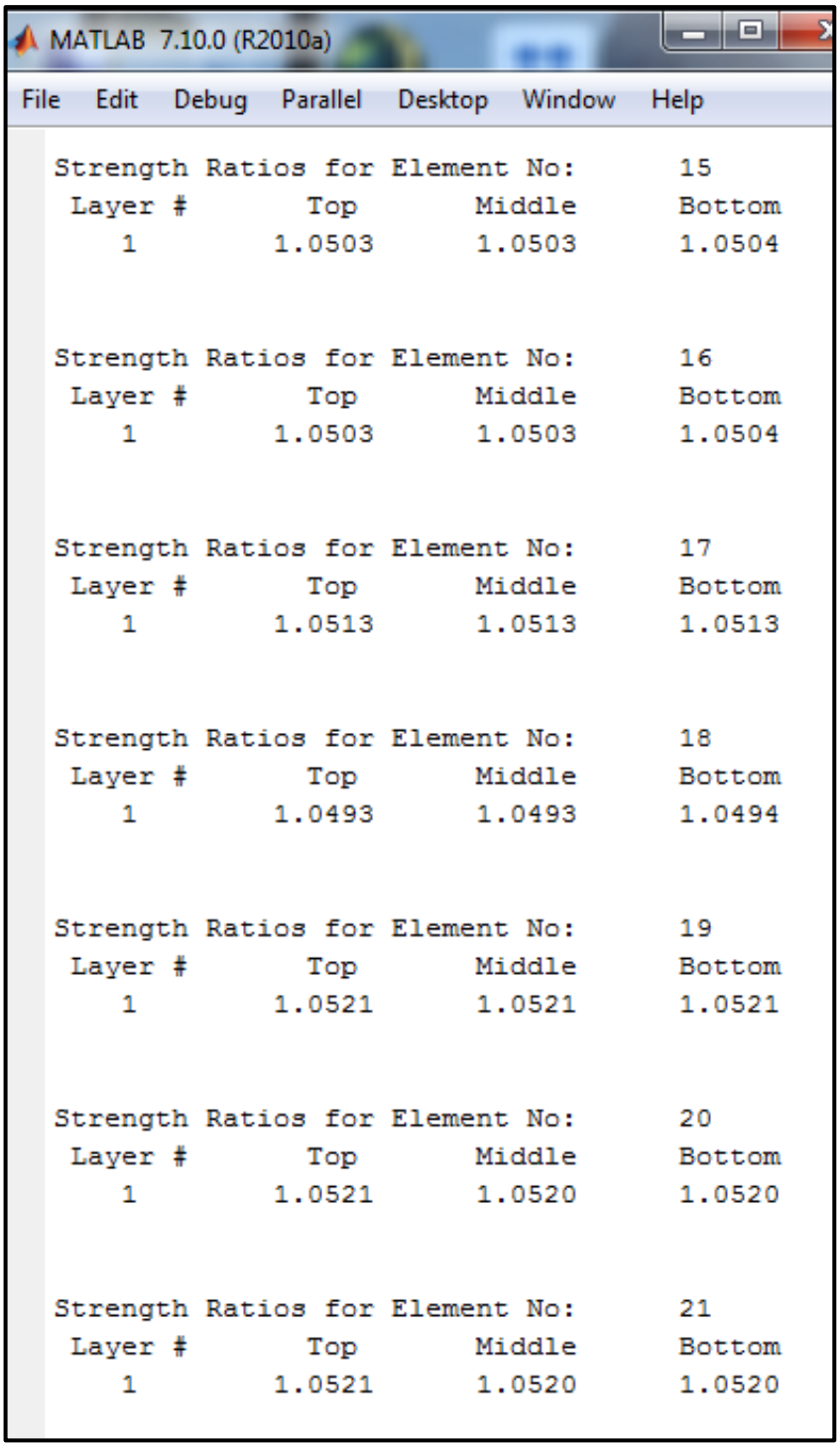


Figure D.6: Strength Ratio (*SR*) values for C-channel for elements 15 to 21

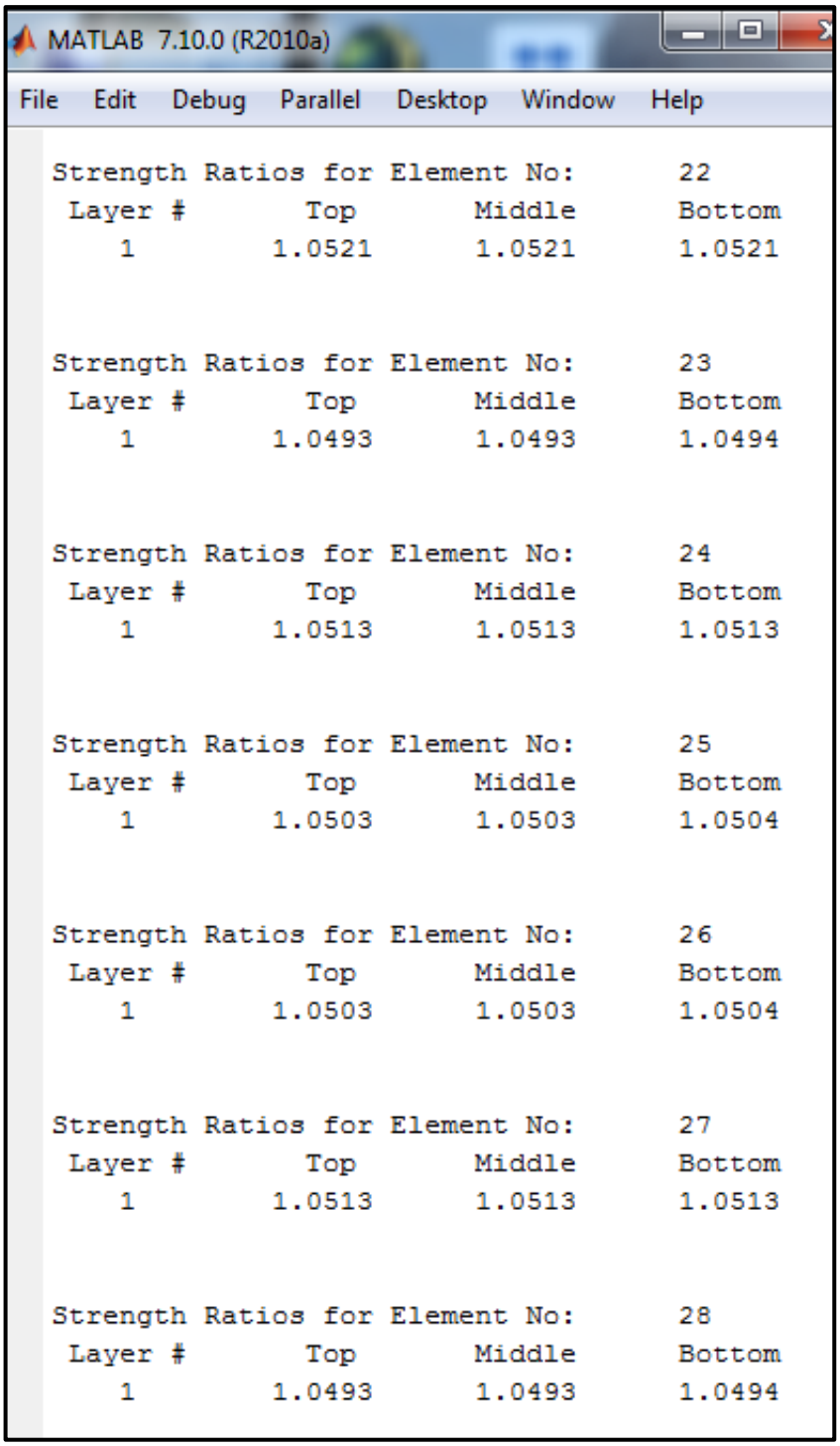


Figure D.7: Strength Ratio (*SR*) values for C-channel for elements 22 to 28

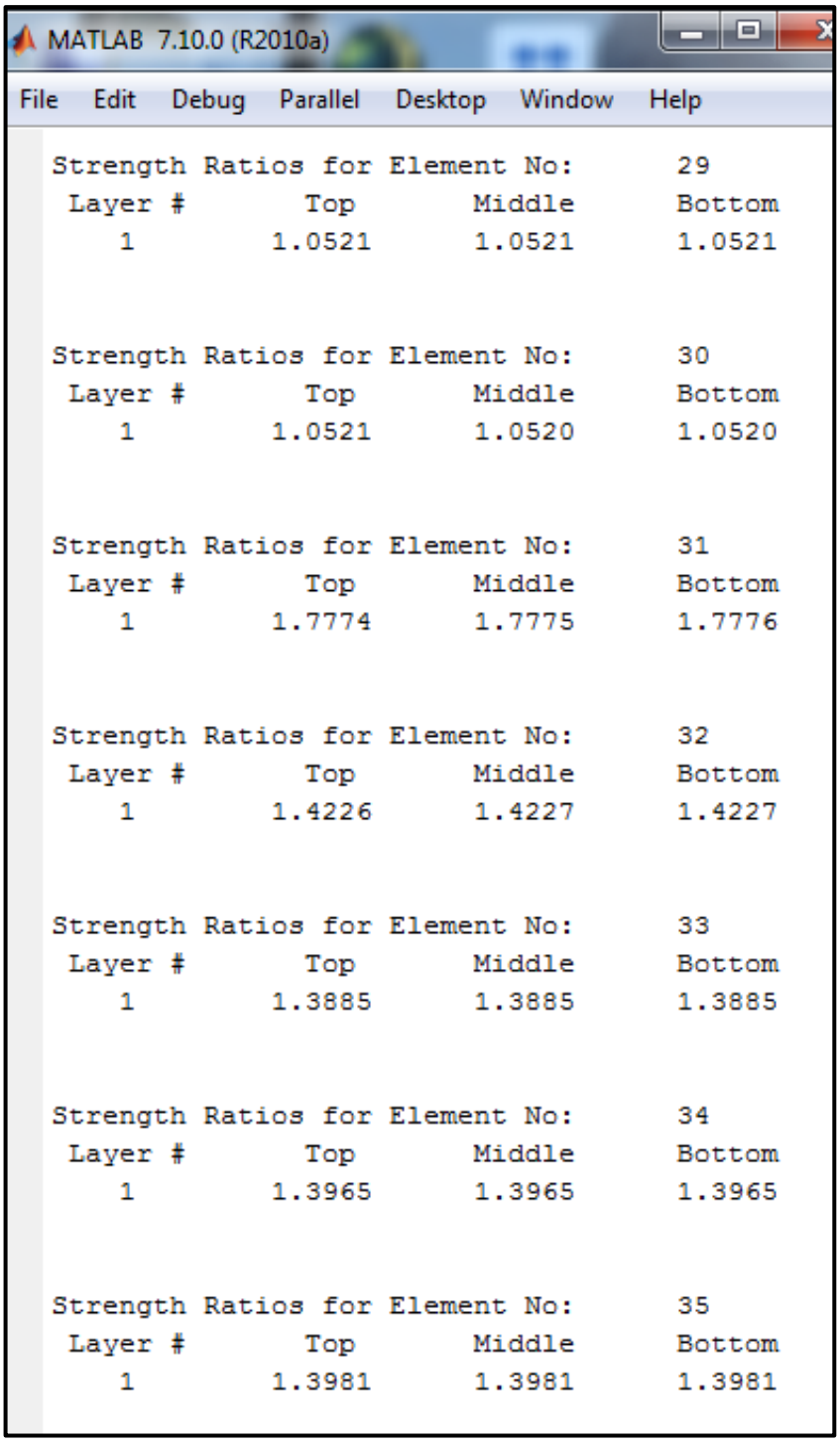


Figure D.8: Strength Ratio (*SR*) values for C-channel for elements 29 to 35

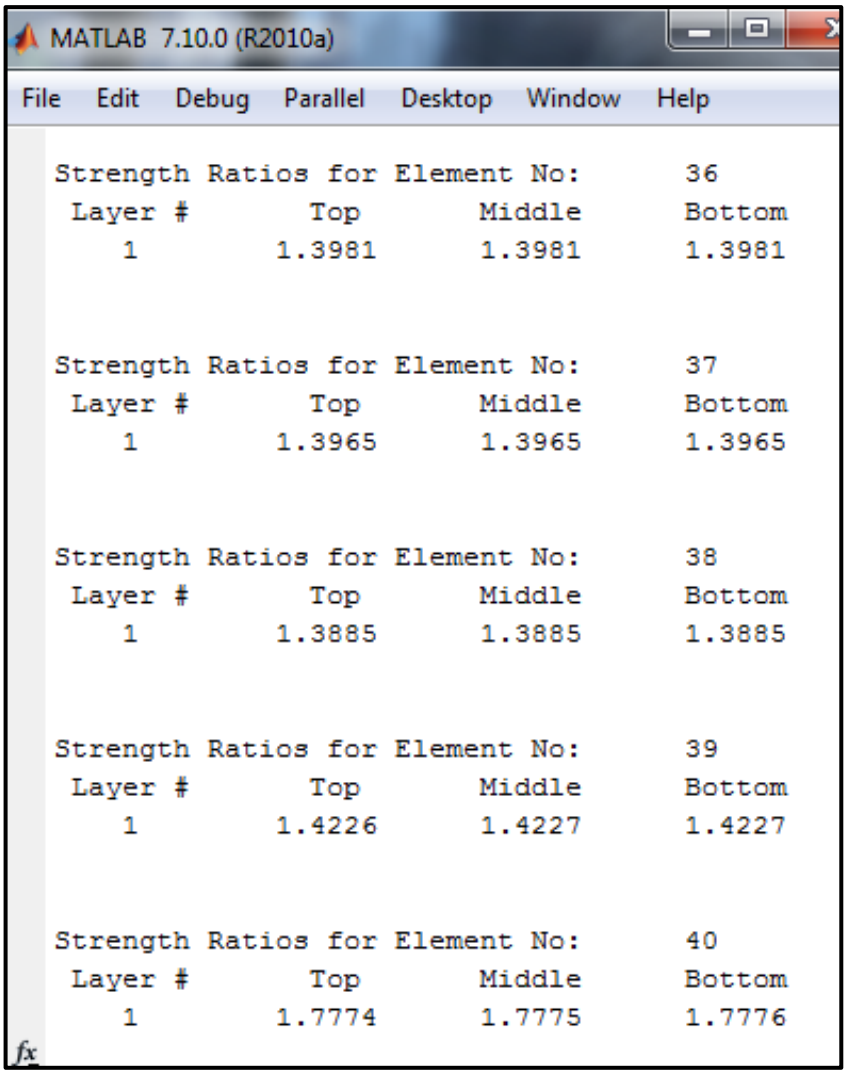


Figure D.9: Strength Ratio (*SR*) values for C-channel for elements 36 to 40

APPENDIX E: MITSUBISHI RV-2AJ SPECIFICATIONS

Table E.1 gives the specifications of the robotic arm used in this study and discussed in Chapter 6. Figure E.1 shows the dimensions and working envelope of the arm.

Table E.1: Specifications of Mitsubishi RV-2AJ robotic arm [149]

Characteristic		Unit	Specification
Degrees of freedom (No. of axes)		-	5
Installation posture		-	Floor or ceiling mounting
Structure		-	Vertical multiple-joint type
Drive system		-	AC servo
Position detection method		-	Absolute encoder
Operating range	Waist (J1)	°	300 (-150 to +150)
	Shoulder (J2)	°	180 (-60 to +120)
	Elbow (J3)	°	230 (-110 to +120)
	Wrist twist (J4)	°	N/A
	Wrist pitch (J5)	°	180 (-90 to +90)
	Wrist roll (J6)	°	400 (-200 to +200)
Maximum speed	Waist (J1)	°/s	180
	Shoulder (J2)	°/s	90
	Elbow (J3)	°/s	135
	Wrist twist (J4)	°/s	N/A
	Wrist pitch (J5)	°/s	180
	Wrist roll (J6)	°/s	210
Maximum composite speed		mm/s	2200
Payload capacity	Rated	kg	1.5
	Maximum	kg	2
Position repeatability		mm	±0.02
Ambient temperature		°C	0 to 40
Weight		kg	17
Tolerable moment	Wrist twist (J4)	Nm	N/A
	Wrist pitch (J5)	Nm	2.16
	Wrist roll (J6)	Nm	1.1
Tolerable inertia	Wrist twist (J4)	kgm^2	N/A
	Wrist pitch (J5)	kgm^2	3.24×10^{-2}
	Wrist roll (J6)	kgm^2	8.43×10^{-3}
Arm reachable radius		mm	410

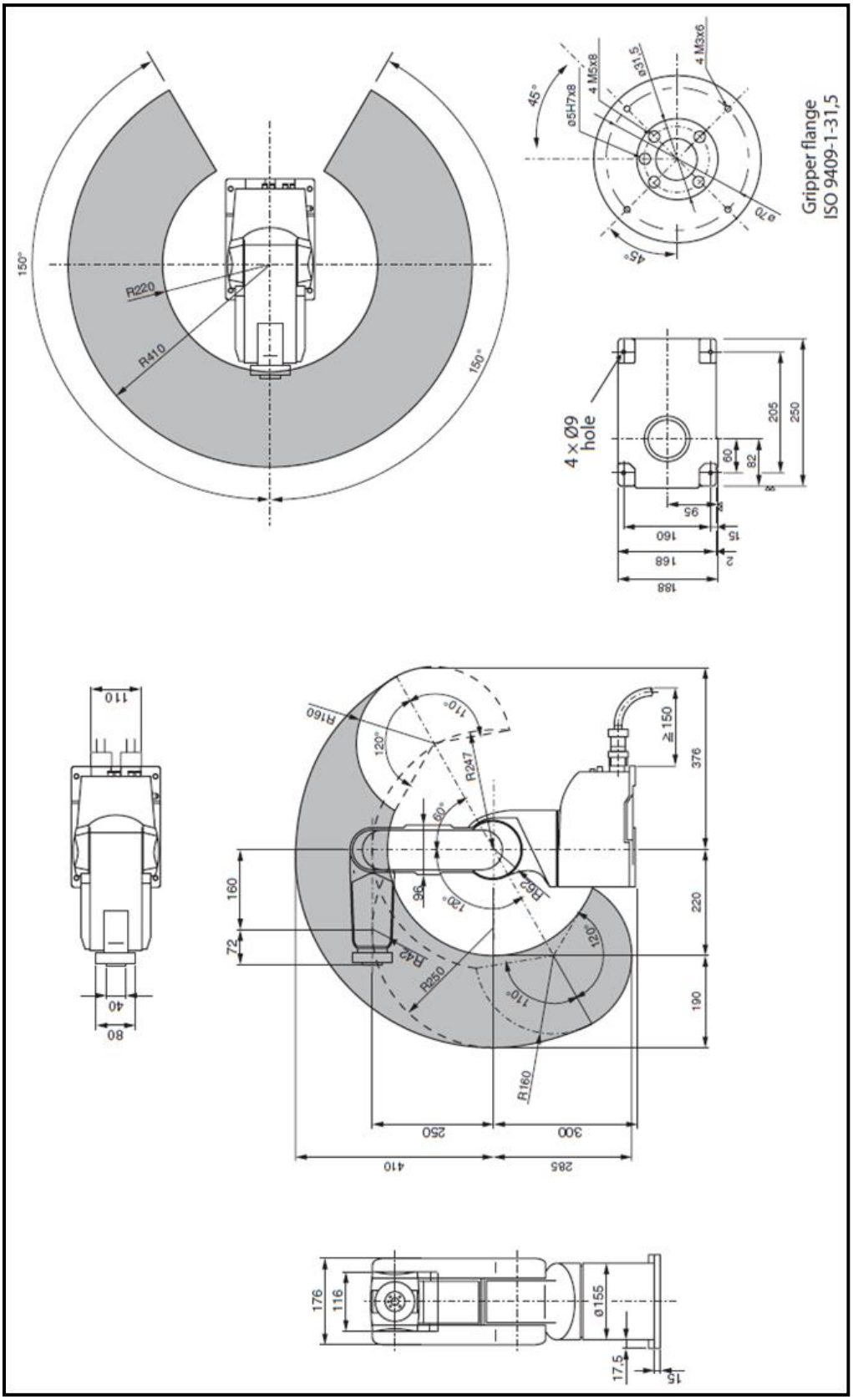


Figure E.1: Dimensions (mm) of the Mitsubishi RV-2AJ robotic arm [149]

APPENDIX F: CONVERSION OF FIBRE LENGTH TO PULSES

In Chapter 6, the stepper motor labelled E in Figure 6.2 requires pulses to rotate. As discussed, the pulses are sent from the motor controller and each pulse rotates the motor by 0.9°. This appendix explains the method used to convert fibre lengths to pulses, and provides a table with conversion values for fibre lengths from 25 to 100mm.

A correlation between the fibre length and number of pulses needs to be established. The fibre is drawn into the pulling-and-cutting unit via the rollers labelled D in Figure 6.2. The length of fibre that passes between the rollers may be determined using the equation for arc length, which is shown below.

$$S = r\,\theta \tag{F.1}$$

where: S is the arc length, r is the radius, θ is the angle in radians

In equation (F.1), the arc length (S) is the fibre length (F_L), and r is the radius of the roller. Therefore, for any given value of F_L, a corresponding angle (θ) may be found. This value of θ is the angle the roller must rotate to achieve the given fibre length (F_L). Since the roller and stepper motor are coupled, the motor should also rotate the same angle as the roller. The value of θ, which is in radians, must be converted to degrees, and the result divided by the number of degrees per pulse. This calculation gives the number of pulses required for the given fibre length F_L. Hence, the correlation between fibre length and number of pulses may be expressed by equation (F.2).

$$NoP = \frac{180\ F_L}{\pi\ r\ DoP} \tag{F.2}$$

where: NoP is the number of pulses, F_L is the fibre length, r is the radius of the roller, and DoP is the degrees per pulse

Equation (F.2) may now be used to convert any fibre length to the number of pulses. Table F.1 shows the equivalent pulses for F_L = 25 to 150mm, r = 15mm, and DoP = 0.9°/pulse.

Table F.1: Fibre lengths converted to number of pulses

Fibre Length (mm)	Pulses (Exact)	Pulses (Rounded up)	New Fibre Length (mm)	Difference (mm)
25	106.103295	107	25.21	0.178
26	110.347427	111	26.15	0.121
27	114.591559	115	27.10	0.063
28	118.835691	119	28.04	0.006
29	123.079823	124	29.22	0.184
30	127.323954	128	30.16	0.126
31	131.568086	132	31.10	0.069
32	135.812218	136	32.04	0.011
33	140.056350	141	33.22	0.189
34	144.300482	145	34.16	0.132
35	148.544614	149	35.11	0.074
36	152.788745	153	36.05	0.017
37	157.032877	158	37.23	0.195
38	161.277009	162	38.17	0.137
39	165.521141	166	39.11	0.080
40	169.765273	170	40.06	0.022
41	174.009404	175	41.23	0.200
42	178.253536	179	42.18	0.143
43	182.497668	183	43.12	0.085
44	186.741800	187	44.06	0.028
45	190.985932	191	45.00	0.206
46	195.230064	196	46.18	0.148
47	199.474195	200	47.12	0.091
48	203.718327	204	48.07	0.033
49	207.962459	208	49.01	0.211
50	212.206591	213	50.19	0.154
51	216.450723	217	51.13	0.096
52	220.694854	221	52.07	0.039
53	224.938986	225	53.01	0.217
54	229.183118	230	54.19	0.159
55	233.427250	234	55.13	0.102
56	237.671382	238	56.08	0.044
57	241.915513	242	57.02	0.222

58	246.159645	247	58.20	0.165
59	250.403777	251	59.14	0.107
60	254.647909	255	60.08	0.050
61	258.892041	259	61.03	0.228
62	263.136173	264	62.20	0.170
63	267.380304	268	63.15	0.113
64	271.624436	272	64.09	0.055
65	275.868568	276	65.03	0.233
66	280.112700	281	66.21	0.176
67	284.356832	285	67.15	0.118
68	288.600963	289	68.09	0.061
69	292.845095	293	69.04	0.003
70	297.089227	298	70.21	0.181
71	301.333359	302	71.16	0.124
72	305.577491	306	72.10	0.066
73	309.821623	310	73.04	0.009
74	314.065754	315	74.22	0.187
75	318.309886	319	75.16	0.129
76	322.554018	323	76.11	0.072
77	326.798150	327	77.05	0.014
78	331.042282	332	78.23	0.192
79	335.286413	336	79.17	0.135
80	339.530545	340	80.11	0.077
81	343.774677	344	81.05	0.020
82	348.018809	349	82.23	0.198
83	352.262941	353	83.17	0.140
84	356.507073	357	84.12	0.083
85	360.751204	361	85.06	0.025
86	364.995336	365	86.00	0.204
87	369.239468	370	87.18	0.146
88	373.483600	374	88.12	0.088
89	377.727732	378	89.06	0.031
90	381.971863	382	90.01	0.209
91	386.215995	387	91.18	0.152
92	390.460127	391	92.13	0.094
93	394.704259	395	93.07	0.036
94	398.948391	399	94.01	0.215
95	403.192522	404	95.19	0.157
96	407.436654	408	96.13	0.100
97	411.680786	412	97.08	0.042
98	415.924918	416	98.02	0.220
99	420.169050	421	99.20	0.163

100	424.413182	425	100.14	0.105
101	428.657313	429	101.08	0.048
102	432.901445	433	102.02	0.226
103	437.145577	438	103.20	0.168
104	441.389709	442	104.14	0.111
105	445.633841	446	105.09	0.053
106	449.877972	450	106.03	0.231
107	454.122104	455	107.21	0.174
108	458.366236	459	108.15	0.116
109	462.610368	463	109.09	0.059
110	466.854500	467	110.03	0.001
111	471.098632	472	111.21	0.179
112	475.342763	476	112.15	0.122
113	479.586895	480	113.10	0.064
114	483.831027	484	114.04	0.007
115	488.075159	489	115.22	0.185
116	492.319291	493	116.16	0.127
117	496.563422	497	117.10	0.070
118	500.807554	501	118.05	0.012
119	505.051686	506	119.22	0.190
120	509.295818	510	120.17	0.133
121	513.539950	514	121.11	0.075
122	517.784082	518	122.05	0.018
123	522.028213	523	123.23	0.196
124	526.272345	527	124.17	0.138
125	530.516477	531	125.11	0.081
126	534.760609	535	126.06	0.023
127	539.004741	540	127.23	0.201
128	543.248872	544	128.18	0.144
129	547.493004	548	129.12	0.086
130	551.737136	552	130.06	0.029
131	555.981268	556	131.00	0.207
132	560.225400	561	132.18	0.149
133	564.469531	565	133.12	0.092
134	568.713663	569	134.07	0.034
135	572.957795	573	135.01	0.212
136	577.201927	578	136.19	0.155
137	581.446059	582	137.13	0.097
138	585.690191	586	138.07	0.040
139	589.934322	590	139.02	0.218
140	594.178454	595	140.19	0.160
141	598.422586	599	141.14	0.103

142	602.666718	603	142.08	0.045
143	606.910850	607	143.02	0.223
144	611.154981	612	144.20	0.166
145	615.399113	616	145.14	0.108
146	619.643245	620	146.08	0.051
147	623.887377	624	147.03	0.229
148	628.131509	629	148.20	0.171
149	632.375641	633	149.15	0.114
150	636.619772	637	150.09	0.056

In Table F.1, the pulses calculated in the second column were rounded up to whole numbers (third column). A pulse value with a decimal fraction means that the motor must rotate for a fraction of a pulse. This is not possible as the motor controller cannot generate pulses in parts. Therefore the pulse values need to be whole numbers.

However, increasing the pulse values increases the fibre lengths. Therefore the pulse values in the third column were converted back to fibre lengths, which are shown in the fourth column. Comparison of the fourth and first columns reveals that the increase in fibre length values is negligible (fifth column), with the largest difference between the two columns being 0.235mm.

The above table may also be used in reverse when converting pulses to fibre lengths.

APPENDIX G: MATLAB CODE FOR RFP SYSTEM

In Chapter 6 the software interface between the RFP process and the VSM code was discussed. It was established that two essential elements, namely, the position list and command list, were required to progress from the design phase to the fabrication stage. This section presents a line-by-line explanation of the Matlab code that generates these lists. First, the main script file, which contains the input parameters, controls the execution of functions, and stores the created lists to files, is examined. Following this is the discussion of the functions and sub-functions involved with the creation of the position and command lists. Each function is examined in turn. This appendix is concluded with the line-by-line analysis of the Graphical User Interface (GUI) for the VSM-TO-RFP code. As mentioned previously, each function is a separate file and the lines beginning with "%" denotes coding comments.

Main script file: VSM-TO-RFP

```
%Generate position and command lists

%Input parameters
1    RR = load('Plate-Graph-3D-Robot.rslt','-mat');
2    Scale = 5;
3    T_Width = 1;
4    FL_Range = [1 100];
5    T_Range = [120 100];
6    ST_Pos = [100 0 0];

%Obtain positions and commands
7    [Fib_Len,Pos_List] = Position_List(RR.FEM,RR.Results,Scale,
     T_Width,FL_Range,T_Range,ST_Pos);
8    Command = Command_List(Fib_Len,FL_Range);

%Save to positions file
9    [N_Pos,~] = size(Pos_List);
10   fid = fopen('Positions2.pos', 'w');
11   for t = 1:N_Pos
12        fprintf(fid, '%s=(%.2f,%.2f,%.2f,%.2f,%.2f,%.2f)(6,0)\n',
          Pos_List{t,:});
13   end
14   fclose(fid);

%Save to commands file
15   [N_Com,~] = size(Command);
16   fid = fopen('Command2.mb4', 'w');
```

```
17  for t = 1:N_Com
18      fprintf(fid, '%s %s\n', Command{t,:});
19  end
20  fclose(fid);

21  disp('Done')
```

In the above code, the saved results from the VSM design process, which includes the fibre layup parameters as well as the finite element data, is loaded in line 1. Lines 2 to 6 contain values for the input parameters. The first of these is the scaling factor (*Scale*) which allows for the resizing of the FEM structure. In line 3, the *T_Width* parameter, which is the total width of the fibre tows loaded into the fibre pulling-and-cutting unit, is assigned. The minimum and maximum allowable fibre lengths are specified in line 4 (*FL_Range*), while line 5 contains the dimensions of the fibre placement tool (*T_Range*). The co-ordinates of the starting position for fibre placement is given in line 6 (*ST_Pos*).

In lines 7 and 8, the functions that create the position and command lists are executed. The examination of these functions, namely, **Position_List** and **Command_List**, and their sub-functions follow this discussion. A file with a .pos extension that stores the position list is created in lines 9 to 14. Similarly, the command list is stored in a file, with an .mb4 extension, via lines 15 to 20. In line 21 a notification is displayed informing the user that the conversion process has been completed.

Function: Position_List

```
22  function [Fib_Len_Lay,Positions] = Position_List(FEM,Results,
    Scale,T_Width,FL_Range,T_Range,ST_Pos)
%Determines the positions required for layer-by-layer fibre layup

23  Elements = cell2mat(FEM.elem.conn);
24  Node_Pos = Scale*FEM.node.coord;
25  [N_El,~] = size(Elements);

%Initialise arrays
26  Elem_Plne = cell(N_El,1);
27  Elem_Ang = zeros(N_El,2);
28  Elem_Size = zeros(N_El,3);
29  Elem_Tilt = zeros(N_El,3);
30  Elem_Lay = zeros(N_El,1);
```

```
Elem_Fib_Len = cell(N_El,1);
Elem_Cov = cell(N_El,1);
Pos_Adj = cell(N_El,1);
Elem_Pass = cell(N_El,1);
Elem_Incr = cell(N_El,1);
Elem_Pos = cell(N_El,1);

%Find plane, angle, size, tilt, number of layers for each element
for p = 1:N_El
    Node1 = Node_Pos(Elements(p,1),:);
    Node2 = Node_Pos(Elements(p,2),:);
    Node3 = Node_Pos(Elements(p,3),:);
    Node4 = Node_Pos(Elements(p,4),:);
    [Elem_Plne{p},Elem_Ang(p,:)] = Elem_Plane_Angle(Node1,
    Node2,Node3,Node4);
    [Elem_Size(p,:),Elem_Tilt(p,:)] = Elem_Size_Tilt(Node1,
    Node2,Node3,Node4,Elem_Plne{p});
    Elem_Lay(p) = length(Results(p).Angle);
end

%Determine fibre length, tow coverage, passes, increments for each
  layer of each element
for q = 1:N_El
    Elem_Fib_Len{q} = Fibre_Length(Elem_Lay(q),
    Results(q).Angle,Elem_Size(q,:),FL_Range);
    [Elem_Cov{q},Pos_Adj{q}] = Elem_Cov_Adj(Elem_Lay(q),
    Results(q).Angle,Elem_Fib_Len{q},T_Width,T_Range);
    [Elem_Pass{q},Elem_Incr{q}] = Elem_Pass_Incr(Elem_Lay(q),
    Elem_Size(q,:),Elem_Cov{q},Elem_Plne{q});
end

%Determine placement positions and orientations element by element
for m = 1:N_El
    Node1 = Node_Pos(Elements(m,1),:);
    Elem_Pos{m} = Elem_Positions(Elem_Lay(m),Node1,ST_Pos,
    Elem_Plne{m},Elem_Ang(m,:),Pos_Adj{m},Elem_Pass{m},
    Elem_Incr{m},Results(m).Angle);
end

%Determine placement positions layer-by-layer
num = 0;
LayE = zeros(N_El,max(Elem_Lay));
for e = 1:N_El
    for f = 1:Elem_Lay(e)
        num = num + (Elem_Pass{e}(f,1) * Elem_Pass{e}(f,2));
        LayE(e,f) = e;
    end
end

NN = 0;
Lay_Pos = zeros(num,6);
```

```
65  Fib_Len_Lay = zeros(num,1);

66  for g = 1:max(Elem_Lay)
67      for h = 1:N_El
68          if LayE(h,g) ~= 0
69              [Row,~] = size(Elem_Pos{h}{g});
70              for n = 1:Row
71                  NN = NN + 1;
72                  Lay_Pos(NN,:) = roundn(Elem_Pos{h}{g}(n,:),-2);
73                  Fib_Len_Lay(NN,1) = Elem_Fib_Len{h}(g);
74              end
75          end
76      end
77  end

%Output layer-by-layer positions as cell array
78  Positions = cell(num+3,7);
79  for z = 1:num
80      Positions(z,1) = {horzcat('P',num2str(z))};
81      Positions(z,2:7) = {Lay_Pos(z,1) Lay_Pos(z,2) Lay_Pos(z,3)
        Lay_Pos(z,4) Lay_Pos(z,5) Lay_Pos(z,6)};
82  end

83  Positions(z+1,1) = {'PREST'};
84  Positions(z+1,2:7) = {240 270 580 0 87.5 0};
85  Positions(z+2,1) = {'PWAIT'};
86  Positions(z+2,2:7) = {350 270 225 0 87.5 0};
87  Positions(z+3,1) = {'PCOLLECT'};
88  Positions(z+3,2:7) = {350 0 225 0 87.5 0};
```

The above function is used to create the position list. It follows the procedure that is represented by the flowchart in Figure 6.10. The function comprises of six inputs and two outputs (line 22). The inputs include the finite element data (*FEM*), fibre layup parameters from the VSM design process (*Results*), scaling factor (*Scale*), tow width (*T_Width*), fibre length range (*FL_Range*), dimensions of the fibre placement tool (*T_Range*), and the starting co-ordinates for fibre placement (*ST_Pos*). The outputs are the fibre length for each layer of each element (*Fib_Len_Lay*) and the position list (*Positions*).

In line 23, the association of each element with its nodes is extracted from the finite element data. Next (line 24) the co-ordinates of each node is scaled via the scaling factor. Line 25 is used to determine the number of elements present in the finite

element model. The various matrices and cell arrays that are used to store the calculated data are created in lines 26 to 36. These entities will be discussed further as they are encountered.

When an element is created in a finite element model, it may exist in one or multiple planes, and, consequently, may be angled to the principal planes (XY, XZ and YZ). Further, each element may differ in size to every other element, and may also be inclined or tilted with respect to the principal axes (X, Y and Z). Therefore, the first step in creating the position list would be to determine the plane, angle, size and tilt of each element in the finite element model. The coded loop in lines 37 to 45 is used to achieve this.

As mentioned in Chapter 6, each element is a quadrilateral, consisting of four nodes. Lines 38 to 41 are used to extract the position of the four nodes constituting each element from the finite element data. This information is used in line 42 by the **Elem_Plane_Angle** function, which is discussed later, to determine the planes in which an element exists as well as the angle of that element to the principal planes. The *Elem_Plne* and *Elem_Ang* arrays are used to store the element's plane and angle data, respectively. The size and tilt of an element is determined by the **Elem_Size_Tilt** function (discussed later) in line 43. The *Elem_Size* and *Elem_Tilt* arrays are used to store the element's size and tilt data, respectively. In line 44, the number of fibre layers present in each element is determined. This information is stored in the *Elem_Lay* array and is important for future functions.

The next steps in Figure 6.10 is to find the fibre length, fibre coverage, number of vertical and horizontal passes, and, the vertical and horizontal increments, for each layer in every element. This is achieved by using the coded loop in lines 46 to 50, which is executed for each element. The fibre lengths for each layer of a particular element are determined by using the **Fibre_Length** function in line 47. The calculated values are stored in the *Elem_Fib_Len* array. In line 48, the fibre coverage for each layer of a particular element is found using the **Elem_Cov_Adj** function. The *Elem_Cov* array is used to store the fibre coverage data, and the *Pos_Adj* array stores

the adjustment positions (discussed later) for the fibre tow on the collection-placement tool. The **Elem_Pass_Incr** function in line 49 is used to determine the number of vertical and horizontal passes required as well as the vertical and horizontal increments, for each layer in every element. The *Elem_Pass* array is used to store the number of passes, while the increments are stored using the *Elem_Incr* array.

At this point all the data necessary to determine the fibre placement positions have been computed. For a particular element, the **Elem_Positions** function in line 53 is used to calculate the placement positions. This data is captured and stored in the *Elem_Pos* array. In order to determine the co-ordinates of each position, a reference point is required. Line 52 is used to assign the first node of the element as its reference point. Lines 52 and 53 are repeated for each element in the finite element model (line 51).

The **Elem_Positions** function determined the fibre placement positions in an element-by-element form, which means that the fibre placements would be executed one element at a time. This is not ideal as stepped regions may be formed at the element boundaries which will decrease the mechanical properties. The preferred fibre layup should be performed layer-by-layer as this will ensure that fibres at the element boundaries overlap evenly thus enhancing the strength characteristics [89]. Therefore the fibre placement positions needed to be rearranged in a layer-by-layer format. This was achieved by lines 55 to 77.

In line 55, the variable *num* is created, which is the total number of entries that would be required in the position list. In the next line, the matrix *LayE* is created, where the number of rows equals the number of elements in the finite element model, and the number of columns equals the highest number of layers in the *Elem_Lay* array. The columns in the *LayE* matrix represent the maximum number of layers a particular element may have. In the nested loop, from line 57 to 62, the value for the variable *num* is calculated (line 59). In line 60, values are assigned to the *LayE* matrix up to the number of layers that exist in an element. For example, if a particular element has three layers, then only the first three columns of the row representing that particular element

in the *LayE* matrix would take on values. The values of remaining columns in that row would remain zero.

The next two lines of the function create a counter *NN* (line 63) and a matrix *Lay_Pos*, which temporarily stores the position list (line 64). In line 65, the function's output variable *Fib_Len_Lay* is initiated. The nested loop in lines 66 to 77 is used to rearrange the placement positions from an element-by-element to a layer-by-layer format. The rearranged positions are stored in the *Lay_Pos* matrix. The fibre lengths are also converted to a layer-by-layer format, and these values are stored in the *Fib_Len_Lay* array. It is important to maintain the association between the placement positions and the fibre lengths as it is a key aspect in the creation of the command list.

In the position list, each position has to have a label, such as P1, P2, and so on, in order to enable easier referencing to the placement position. In lines 79 to 82, a label is combined with each position in the *Lay_Pos* matrix, and this is stored in the function's output array (*Positions*). As mentioned in Chapter 6, three extra positions are required. These positions, namely, PREST, PWAIT and PCOLLECT are amended to the position list via lines 83 to 88. The PWAIT position is the point where the robotic arm waits to collect the cut fibres, while the PCOLLECT position is the point where the cut fibres may be collected from the fibre pulling-and-cutting unit. The PREST position is the intermediary point where the robotic arm moves to in order to avoid collisions with the mould or the pulling-and-cutting unit. This movement is performed before every fibre placement.

As mentioned earlier, the position list is saved to a file via lines 9 to 14. In the above function, several other functions were encountered, namely, **Elem_Plane_Angle**, **Elem_Size_Tilt**, **Fibre_Length**, **Elem_Cov_Adj**, **Elem_Pass_Incr**, and **Elem_Positions**. The discussion on these follows.

Function: Elem_Plane_Angle

```
89  function [Plne,Ang] = Elem_Plane_Angle(Node1,Node2,Node3,Node4)
%Determine the plane(s) and inclination of an element
```

```
90   Plne = '';
91   B_Ang = 0;
92   C_Ang = 0;

%Delta X,Y,Z for diags 1-3 and 2-4
93   Delta13 = Node3 - Node1;
94   Delta24 = Node4 - Node2;
%Gradients of line 1-3 in each plane
95   Grad13 = [Delta13(2)/Delta13(1) Delta13(3)/Delta13(1)
     Delta13(3)/Delta13(2)];
%Gradients of line 2-4 in each plane
96   Grad24 = [Delta24(2)/Delta24(1) Delta24(3)/Delta24(1)
     Delta24(3)/Delta24(2)];

%Correcting for zero div by zero (NaN)
97   for a = 1:3
98       if isnan(Grad13(a))
99           Grad13(a) = 0;
100      end
101      if isnan(Grad24(a))
102          Grad24(a) = 0;
103      end
104  end

%Determine indices of entries with zero or Inf value
105  Gind013 = find(Grad13 == 0 | Grad13 == Inf | Grad13 == -Inf);
106  Gind024 = find(Grad24 == 0 | Grad24 == Inf | Grad24 == -Inf);

%Determine plane(s) of the element
107  if (length(Gind013) == 3) && (length(Gind024) == 3)      % 1
     plane
108      N_Pln = '1';
109      MV = find(Delta13 == 0 & Delta24 == 0);
110      switch (MV)
111          case 1
112              Plne = 'YZ';
113          case 2
114              Plne = 'XZ';
115          case 3
116              Plne = 'XY';
117      end
118  elseif (length(Gind013) == 3) && (length(Gind024) == 2)  % 2
     planes
119      N_Pln = '2-Special';
120      MV = find(Delta24 == 0);
121      switch (MV)
122          case 1
123              Plne = 'XY-XZ';
124          case 2
125              Plne = 'XY-YZ';
126          case 3
```

```
127             Plne = 'XZ-YZ';
128     end
129 elseif (length(Gind013) == 2) && (length(Gind024) == 2)  % 1
    plane
130     N_Pln = '1';
131     MV = find(Delta13 == 0);
132     switch (MV)
133         case 1
134             Plne = 'YZ';
135         case 2
136             Plne = 'XZ';
137         case 3
138             Plne = 'XY';
139     end
140 elseif (length(Gind013) == 2) && (isempty(Gind024))      % 3
    planes
141     N_Pln = '3';
142     Plne = 'ALL';
143 elseif (isempty(Gind013)) && (isempty(Gind024))          % 2 or
    3 planes
144     N_Pln = '2-Normal';
    %Use str line eqn: c = y - mx; c = z - mx; c = z - my;
145     Cxy = Node3(2) - Grad13(1)*Node3(1);
146     Cxz = Node3(3) - Grad13(2)*Node3(1);
147     Cyz = Node3(3) - Grad13(3)*Node3(2);
    %Use Node(2) to find dependent variable
148     Yx = Grad13(1)*Node2(1) + Cxy;
149     Zx = Grad13(2)*Node2(1) + Cxz;
150     Zy = Grad13(3)*Node2(2) + Cyz;
    %Determine if Node(2) satisfies the 3 straight line equations
151     if Yx == Node2(2)
152         Plne = 'XZ-YZ';
153     elseif Zx == Node2(3)
154         Plne = 'XY-YZ';
155     elseif Zy == Node2(3)
156         Plne = 'XY-XZ';
157     else
158         N_Pln = '3';
159         Plne = 'ALL';
160     end
161 end

%Determine inclination of the element
162 switch (N_Pln)
163     case '1'
164         switch (Plne)
165             case 'XY'
166                 B_Ang = 180;
167                 C_Ang = 0;
168             case 'XZ'
169                 B_Ang = 90;
```

```
170                 C_Ang = 0;
171             case 'YZ'
172                 B_Ang = 90;
173                 C_Ang = 90;
174         end
175     case '2-Special'
176         switch (Plne)
177             case 'XY-XZ'
178                 B_Ang = 180-atand(GRAD24(3));
179                 C_Ang = 0;
180             case 'XY-YZ'
181                 B_Ang = 180-atand(GRAD24(2));
182                 C_Ang = 90;
183             case 'XZ-YZ'
184                 B_Ang = 90;
185                 C_Ang = atand(GRAD24(1));
186         end
187     case '2-Normal'
188         switch (Plne)
189             case 'XY-XZ'
190                 B_Ang = 180-atand(GRAD13(3));
191                 C_Ang = 0;
192             case 'XY-YZ'
193                 B_Ang = 180-atand(GRAD13(2));
194                 C_Ang = 90;
195             case 'XZ-YZ'
196                 B_Ang = 90;
197                 C_Ang = atand(GRAD13(1));
198         end
199     case '3'
200         B_Ang = 180-atand(GRAD13(2));
201         C_Ang = atand(GRAD13(1));
202 end

203 Ang = [B_Ang C_Ang];
```

The purpose of the **Elem_Plane_Angle** function is to determine in which plane or planes an element exists, as well as its angle to the principal planes. An element may reside in one plane (XY, XZ or YZ), or two planes (XY-XZ, XY-YZ or XZ-YZ), or all three planes. In order to determine the plane or planes an element resides in, consider an element in the XY-XZ planes as shown in Figure G.1. The projection of this element onto the three planes (XY, XZ and YZ) results in a quadrilateral, as is the case with the XY- and XZ-planes, or a straight line, as in the case of the YZ-plane. It may be deduced from this that if an element lies in two planes then its projection on the third plane will be a straight line.

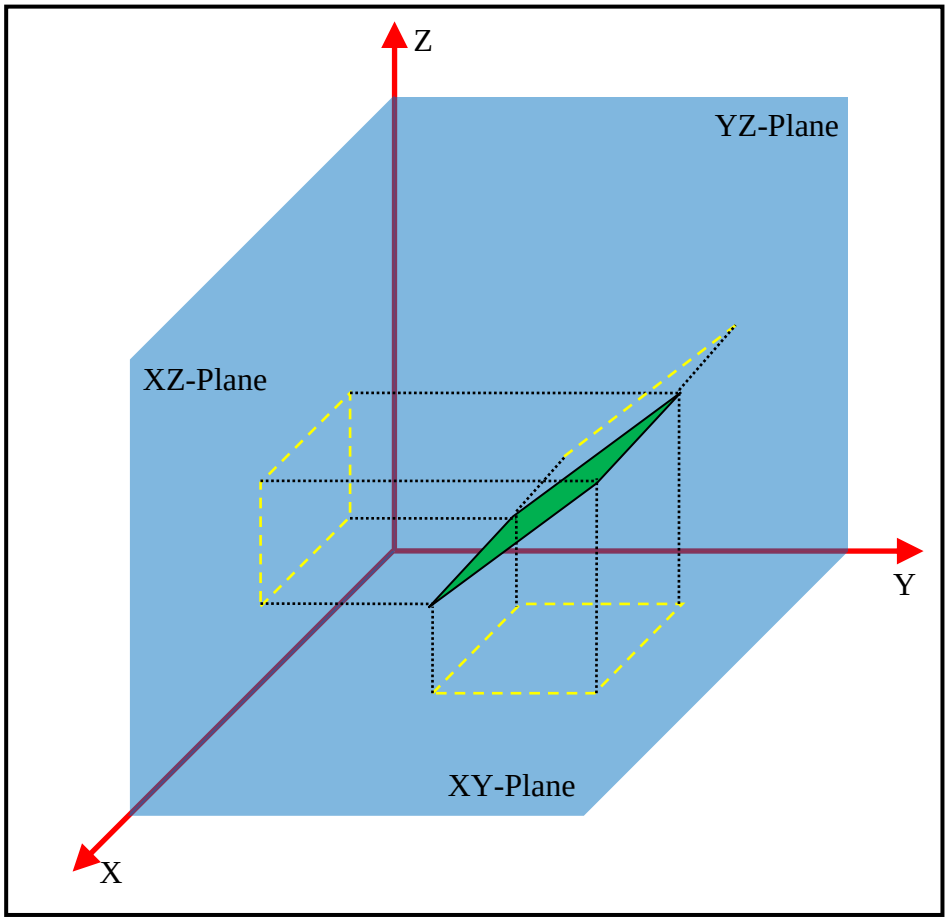


Figure G.1: Element in space projected onto XY, XZ and YZ planes

Likewise, if an element exists in one plane, then its projection onto the remaining two planes would be straight lines. If the element resides in all the planes, there would be no straight line projections onto any of the planes. It may be concluded from this, that the number of straight lines projected by the element may be used to define its plane or planes of existence. Further, the angles of these projected lines would determine the element's gradient relative to the principal planes.

The key in using the above theory lies in the choice of lines to represent the element. Figure G.2 shows an illustration of a quadrilateral element (specified in Chapter 5) with its nodes and edge lines. If the element existed in one plane and the edge lines were parallel to a principal axis, as depicted in the figure, then the projections of these lines onto the other planes would be points. In this case any two adjacent edge lines may be used to represent the element. However, if the same element was inclined in

the plane, then the projection of the edge lines would be lines. This would also be the case if the element existed in two planes and was inclined to both the planes. In these cases there would be ambiguity as to whether the element exists in one or two planes.

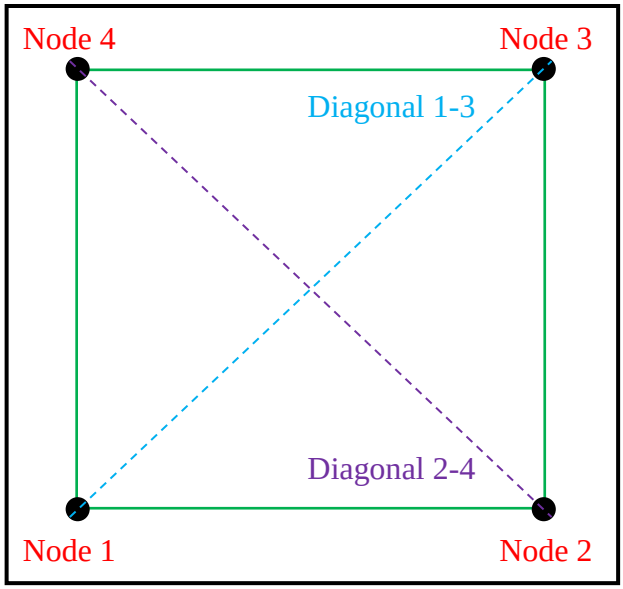


Figure G.2: Positions of nodes and edge lines in an element

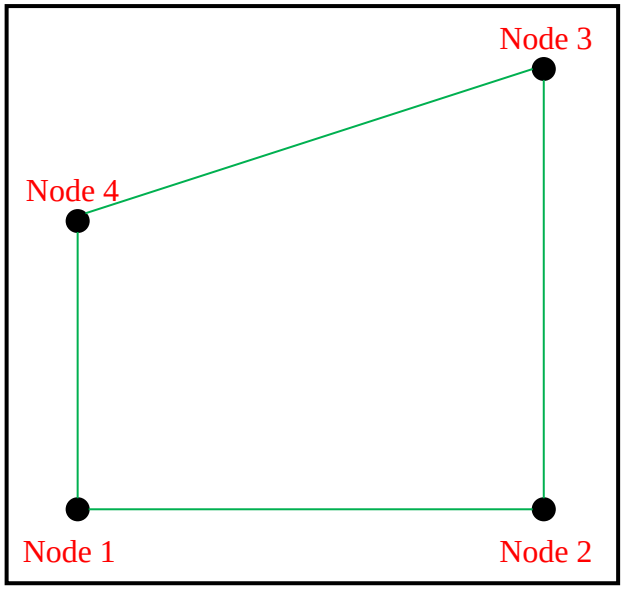


Figure G.3: Illustration of a skew element

Further, although the element is a quadrilateral, it may be skew as shown in Figure G.3. In this case, any two adjacent edge lines would not be the same as any other two.

Also, in an extreme case, two of the edge lines may be in one plane and the other two in a different plane. For these cases, more edge lines would be required to represent the element and this would complicate the coding. These and the above reason make it clear that the edge lines of the element should not be used to represent it.

A better representation of the element would be the main diagonals. As the diagonals cross over the area of the element, they are more sensitive to changes in geometry. Therefore it would be easier to determine the plane or planes in which an element resides. Further, the diagonals always project onto the planes as lines, as they are rarely parallel to any axis. However, there is a possibility of this occurring and this case is depicted in Figure G.4. Here the projection of one of the diagonals is a point as shown. If this happens when the element resides in two planes, it may be erroneously determined that the element exists in one plane. However, at no instance will both the diagonals project as a point on the same plane. Hence, having both main diagonals representing the element eliminates the possibility of the incorrect determination of the planes in which an element exists.

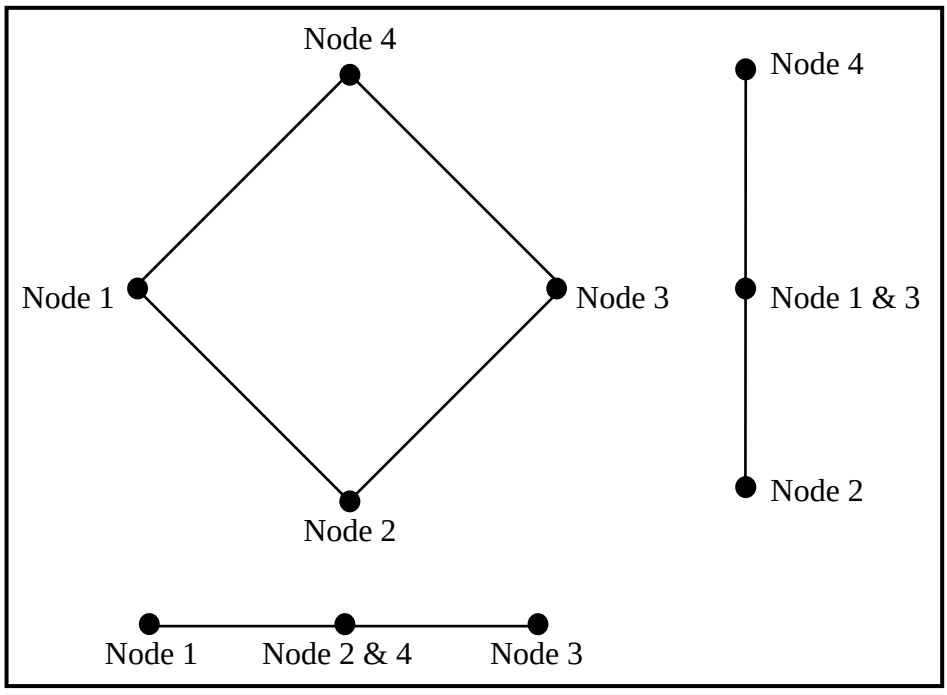


Figure G.4: Special case where diagonals project onto a point

The above concept of using the main diagonals of an element to find the planes in which it exists was incorporated into the **Elem_Plane_Angle** function. The function considers one element at a time. It has four inputs (line 89) which are the co-ordinates of the nodes of the element under consideration. The outputs of the function include the planes of the element (*Plne*) and its angles to the principal planes (*Ang*).

In lines 90 to 92, variables used in the function are initialised. The *B_Ang* and *C_Ang* variables correspond to the B and C angular positions of the robotic arm, respectively. The difference in the X, Y and Z co-ordinates (ΔX, ΔY and ΔZ) of the diagonal formed between Node 1 and Node 3 (Diagonal 1-3) is calculated in Line 93. Similarly, the ΔX, ΔY and ΔZ of the diagonal between nodes 2 and 4 (Diagonal 2-4) is found in line 94. These diagonals are shown in Figure G.2. The Δ values are stored in the arrays *Delta13*, for Diagonal 1-3, and *Delta24*, for Diagonal 2-4. The code in lines 95 and 96 is used to compute the gradients (ΔY/ΔX, ΔZ/ΔX, and ΔZ /ΔY) of Diagonals 1-3 and 2-4 in each plane, respectively. These values are stored using the arrays *Grad13* and *Grad24*, respectively.

If division by zero is encountered in the computations, a value of 'Inf' or '-Inf' (Infinity) is assigned by the Matlab environment. In the special case where zero divided by zero is come upon, a value of 'NaN' (Not a Number) is assigned by Matlab. This case is possible when the diagonals project onto the planes as a point. In lines 97 to 104, the two gradient arrays (*Grad13* and *Grad24*) are examined to determine whether any entry has a value of 'NaN', and if so, equates that particular entry to zero.

In lines 105 and 106 the two gradient arrays are examined and the locations of entries of zero or +/-Inf are found. (To enable easier explanations, these values will be referred to as Z-Inf entries.) The planes in which the element under consideration exists is determined via lines 107 to 161.

If the *Grad13* and *Grad24* arrays both have three Z-Inf entries each (line 107), then the element can only exist in one plane. This corresponds to the special case, depicted by Figure G.4, where the diagonals project as points. Further, the *Delta13* and *Delta24*

arrays both have two zero entries each. The locations of these zero entries in the two arrays are compared to each other in line 109. There can be only one matching position and this is used to determine in which plane the element resides (line 110 to 117). Here the appropriate plane is assigned to the output variable *Plne*. In line 108, a value of '1' is assigned to a variable called *N_Pln* which will be used later to find the angles of the element relative to the principal planes.

If the *Grad13* array has three Z-Inf entries and the *Grad24* array has only two (line 118), then the code from lines 119 to 128 is applied. In this case the element exists in two planes. The *N_Pln* variable is assigned a value of '2-Special' (line 119), and the location of the zero entry in the *Delta24* array is found in line 120. The appropriate planes are assigned to the *Plne* variable via lines 122 to 128.

When the *Grad13* and *Grad24* arrays both have two zero entries each (line 129), the element exists in one plane. The *Delta13* and *Delta24* arrays both have one zero entry each. The code in line 131 is used to find the location of the zero entry in the *Delta13* array, and this determines in which plane the element resides. In lines 132 to 139, the plane of the element is assigned to the *Plne* variable. A value of '1' is assigned to the *N_Pln* variable in line 130.

The element exists in all three planes if the *Grad13* array has two zero entries and the *Grad24* array has none (line 140). The *N_Pln* variable assumes a value of '3' (line 141) and the *Plne* variable is assigned a value of 'ALL' (line 142).

If the *Grad13* and *Grad24* arrays both have no zero entries (line 143), the element may exist in two or three planes. This instance is not as straightforward as the previous cases, however the solution is simple. As mentioned earlier, the diagonals project onto the principal planes as straight lines. The equations for these lines for one of the diagonals must be determined. If the coordinates of either node of the second diagonal satisfies any of the equations, then the element resides in two planes. Otherwise the element exists in all three planes.

In the formulation of the code, it was decided to find the equations of the projected lines of Diagonal 1-3. Since the gradient (*Grad13*) and a point (either Node 1 or 3) are known, the constant or intercept of the straight line equation may be determined. In the function, lines 145 to 147 are used to find the constant for the equation of the straight line in each of the principal planes.

Next, it must be determined whether Nodes 2 or 4 lie on any of the projected lines of Diagonal 1-3. In lines 148 to 150 of the code, the coordinates of Node 2 are used to calculate the value of the dependent variable in the equation of the straight line in each of the principal planes. The calculated values are compared to the appropriate coordinate values of Node 2 (lines 151 to 156), and if they are equal, the element exists in two planes, and the *Plne* variable assumes the relevant value. For example, in line 149 the dependent variable is *z*. The value of this is compared to the value of the Z-coordinate of Node 2 in line 153. If they are equal then the element resides in the XY-YZ plane and this is assigned to the *Plne* variable. If the calculated values do not match the coordinates of Node 2, then the element exists in all planes, and the *Plne* variable is assigned a value of 'ALL' (line 159).

The last part of the code (lines 162 to 203) determines the angle of the element to the three principal planes. The angles depend on the value of *N_Pln*, and are stored using the variables *B_Ang* and *C_Ang*, which correspond to the B and C angular positions of the robotic arm, respectively. These angles are stored in the output array Ang.

Function: Elem_Size_Tilt

```
204 function [Size,Tilt] = Elem_Size_Tilt(Node1,Node2,Node3,Node4,
    Plane)
%Determine the size and tilt of an element
205 Delta12 = Node2 - Node1;
206 Delta23 = Node3 - Node2;
207 Htrue = sqrt(Delta12(1)^2 + Delta12(2)^2 + Delta12(3)^2);

208 switch (Plane)
209     case 'XY'
210         Hdist = Delta12(1);
211         Vdist = Delta23(2);
212         Xang = acosd(Delta12(1)/Htrue);
```

```
213         Yang = 90-Xang;
214         Zang = 90;
215     case 'XZ'
216         Hdist = Delta12(1);
217         Vdist = Delta23(3);
218         Xang = acosd(Delta12(1)/Htrue);
219         Yang = 90;
220         Zang = 90-Xang;
221     case 'YZ'
222         Hdist = Delta12(2);
223         Vdist = Delta23(3);
224         Xang = 90;
225         Yang = acosd(Delta12(2)/Htrue);
226         Zang = 90-Yang;
227     case 'XY-XZ'
228         Hdist = Delta12(1);
229         Vdist = sqrt(Delta23(2)^2 + Delta23(3)^2);
230         Xang = acosd(Delta12(1)/Htrue);
231         Yang = acosd(Delta23(2)/Vdist);
232         Zang = 90-Yang;
233     case 'XY-YZ'
234         Hdist = Delta12(2);
235         Vdist = sqrt(Delta23(1)^2 + Delta23(3)^2);
236         Xang = acosd(Delta23(1)/Vdist);
237         Yang = acosd(Delta12(2)/Htrue);
238         Zang = 90-Xang;
239     case 'XZ-YZ'
240         Hdist = Delta12(3);
241         Vdist = sqrt(Delta23(1)^2 + Delta23(2)^2);
242         Yang = acosd(Delta23(2)/Vdist);
243         Xang = 90-Yang;
244         Zang = acosd(Delta12(3)/Htrue);
245     case 'ALL'
246         Hdist = sqrt(Delta12(1)^2 + Delta12(2)^2);
247         Vdist = sqrt(Delta23(1)^2+Delta23(2)^2+Delta23(3)^2);
248         Xang = acosd(Hdist/Htrue);
249         Yang = 90-Xang;
250         Zang = acosd(Delta23(3)/Vdist);
251 end

%Find longest diagonal
252 Diag13 = Node3 - Node1;
253 Diag24 = Node4 - Node2;
254 L_Diag13 = sqrt(Diag13(1)^2 + Diag13(2)^2 + Diag13(3)^2);
255 L_Diag24 = sqrt(Diag24(1)^2 + Diag24(2)^2 + Diag24(3)^2);
256 M_Diag = max(L_Diag13,L_Diag24);

%Outputs
257 Size = [Hdist Vdist M_Diag];
258 Tilt = [Xang Yang Zang];
```

This function is used to determine the vertical and horizontal lengths of an element, as well as its inclination relative to the three axes (X, Y and Z). There are five inputs (line 204), namely, the coordinates of the four nodes (*Node1*, *Node2*, *Node3* and *Node4*), and the planes the element resides in (*Plane*). The outputs are the size (*Size*), and inclination (*Tilt*) of the element.

In line 205, the difference in the X, Y and Z co-ordinates (ΔX, ΔY and ΔZ) of the line or edge between Node 1 and Node 2 is calculated. Similarly, the ΔX, ΔY and ΔZ values of the edge between nodes 2 and 3 is found in line 206. The true distance of the element's horizontal length is determined in line 207.

The computation of the element's horizontal length (*Hdist*), vertical length (*Vdist*), and the angle with the X-axis (*Xang*), Y-axis (*Yang*) and Z-axis (*Zang*), is performed using lines 208 to 251. The calculation of these parameters depends on which plane the element exists in. In lines 252 to 255, the length of the main diagonals is determined. The larger of the two values is stored using the variable *M_Diag*. This attribute is required for use in other functions. The output arrays of this function are assigned their values in lines 257 and 258. It should be noted that the element's tilt values are not required in this version of the software but is included for future software development.

Function: Fibre_Length

```
259 function F_Len = Fibre_Length(N_Lay,Fib_Ang,E_Siz,FL_Rng)
%Determine fibre length for each layer of an element

260 F_Len = zeros(1,N_Lay);
261 for s = 1:N_Lay
%Finds fibre length
262     if (Fib_Ang(s) ~= 90)
263         Length = E_Siz(1)/cosd(Fib_Ang(s));
264     else
265         Length = E_Siz(2);
266     end
%Corrects fibre length if larger than main diagonal
267     if Length > E_Siz(3)
268         Length = E_Siz(2)/cosd(90-abs(Fib_Ang(s)));
269     end
%Adjusts fibre length for robot constraints
270     if Length > max(FL_Rng)
```

```
271         F_Len(s) = max(FL_Rng);
272     elseif Length < min(FL_Rng)
273         F_Len(s) = min(FL_Rng);
274     else
275         F_Len(s) = Length;
276     end
277 end
```

The purpose of this function is to calculate the fibre length of each layer in a particular element. The input parameters (line 259) include the number of layers in the element (*N_Lay*), the fibre orientation angles of each layer (*Fib_Ang*), the element's size (*E_Siz*), and the fibre length limitations (*FL_Rng*). In line 260, the output array (*F_Len*) is initiated.

A coded loop is initiated in line 261 that is executed for each layer in the element. In lines 262 to 266, the fibre length is determined using the element's horizontal length and the layer's fibre orientation angle. For fibre orientation angles greater than 45°, the fibre length tends to be longer than the element's vertical length. This is not acceptable because, during the fibre layup process, the fibres from this element will encroach upon the next. Further, a longer fibre length means more material usage which would increase costs.

Therefore, the code in lines 267 to 269 is used to correct the calculated fibre length. The calculated value is compared to the length of the longest main diagonal of the element (obtained from the **Elem_Size_Tilt** function). If the fibre length value is larger, the fibre length is recalculated using the element's vertical length and the layer's complementary angle (90° – fibre angle).

The fibre pulling-and-cutting unit and the collection-placement tool have limitations with respect to the length of fibre that may be cut and collected. Line 270 to 279 adjusts the fibre length in accordance with the limitations of the RFP apparatus.

Function: Elem_Cov_Adj

```
278 function [E_Cov,P_Adj] = Elem_Cov_Adj(N_Lay,Fib_Ang,Fib_Len,
    T_Wid,T_Rng)
```

```
%Determine fibre tow coverage for each layer of an element
%Calculate position adjustment to compensate for tow area on tool

279 E_Cov = zeros(N_Lay,2);
280 P_Adj = zeros(N_Lay,2);

%Fibre tow coverage
281 for r = 1:N_Lay
    %Horizontal fibre coverage
282     if (abs(Fib_Ang(r)) >= 0) && (abs(Fib_Ang(r)) <= 45)
283         Hcov = Fib_Len(r) / cosd(Fib_Ang(r));
284     elseif (abs(Fib_Ang(r)) > 45) && (abs(Fib_Ang(r)) <= 90)
285         Hcov = T_Wid / sind(Fib_Ang(r));
286     end
    %Vertical fibre coverage
287     if (abs(Fib_Ang(r)) >= 0) && (abs(Fib_Ang(r)) <= 45)
288         Vcov = T_Wid / cosd(Fib_Ang(r));
289     elseif (abs(Fib_Ang(r)) > 45) && (abs(Fib_Ang(r)) <= 90)
290         Vcov = Fib_Len(r) / sind(Fib_Ang(r));
291     end
292     E_Cov(r,:) = abs([Hcov Vcov]);

%Position adjustment
293     Dist = T_Rng(1)/2 - Fib_Len(r);
294     Hadj = Dist * cosd(Fib_Ang(r));
295     Vadj = Dist * sind(Fib_Ang(r));
296     P_Adj(r,:) = [Hadj Vadj];
297 end
```

The **Elem_Cov_Adj** function determines, for each layer in an element, the area covered by the fibre tow when orientated to the fibre orientation angle. Further, it also creates adjustment parameters to compensate for the position of the fibre tow on the placement tool. The inputs of the above function (line 278) are the number of layers (*N_Lay*), fibre orientation angles of each layer (*Fib_Ang*), fibre lengths (*Fib_Len*), the tow width (*T_Wid*), and the placement tool's dimensions (*T_Rng*). In lines 279 and 280 the output arrays, namely, the fibre coverage (*E_Cov*) and the adjustment parameters (*P_Adj*), are initialised.

A coded loop is executed in line 281 that accounts for all the layers in the element under consideration. The horizontal fibre coverage of the element is determined via lines 282 to 286. If the absolute value of the fibre orientation angle is between 0° and 45° (line 282), then the horizontal fibre coverage is calculated using the fibre length and the cosine of the fibre orientation angle (line 283). However, for a fibre orientation

angle between 45° and 90° (absolute value), the horizontal fibre coverage is found using the tow width and the sine of the fibre orientation angle (line 285). Similarly, the vertical fibre coverage is calculated via lines 287 to 291. The output array *E_Cov* is used to store the absolute values of the horizontal and vertical fibre coverage for each layer of the element (line 292).

As mentioned earlier, this function also determines adjustment parameters, which are position corrections for the fibre collection-placement tool due to the positioning of the fibres on the tool after collection. This may be better explained using Figure G.5. The figure shows an illustration of the fibre collection-placement tool with collected fibres. The black dot in the figure marks the centre of the tool and coincides with the point of attachment of the robotic arm. The movement of the arm to a point in space aligns this dot with the X, Y and Z co-ordinates of that point. When the cut fibres are collected for placement, they are not situated at the black dot but rather at the upper end of the collection-placement tool, as shown in the figure. If these fibres were placed in a mould, they would not be in the correct position. Therefore an adjustment is required to ensure that the fibres are placed at their proper position.

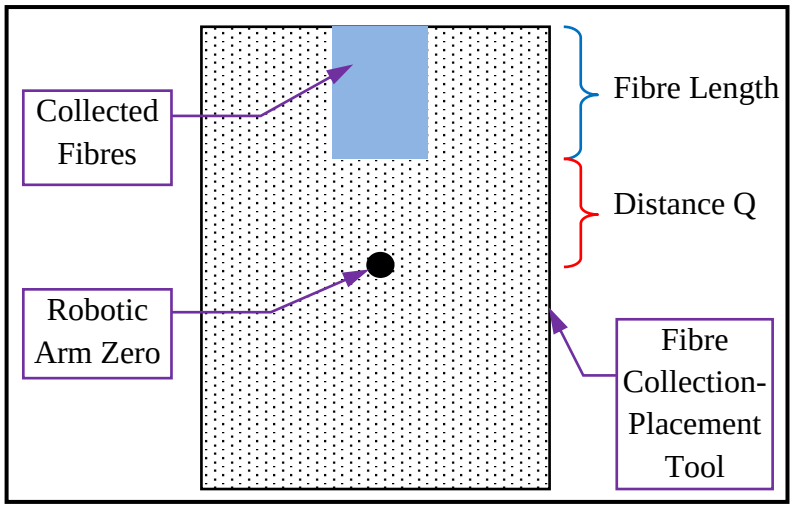


Figure G.5: Position of fibre tow on placement tool

In Figure G.6 the incorrect positioning of the collected fibres is demonstrated. The illustration in Figure G.6(a) is a linear representation of the fibre length and distance

Q, which were shown in Figure G.5. When the fibres are to be placed, the collection-placement tool is orientated to the fibre orientation angle θ, as shown in Figure G.6(b). The tool is moved to the placement position (Figure G.6(c)), and the fibres are placed in the mould. It may clearly be seen that the fibres are placed at the incorrect position. An adjustment in the X, Y and Z co-ordinates of the robotic arm is required. This adjustment is illustrated in Figure G.6(d), where a horizontal and a vertical correction, to place the fibres at the placement position, are shown.

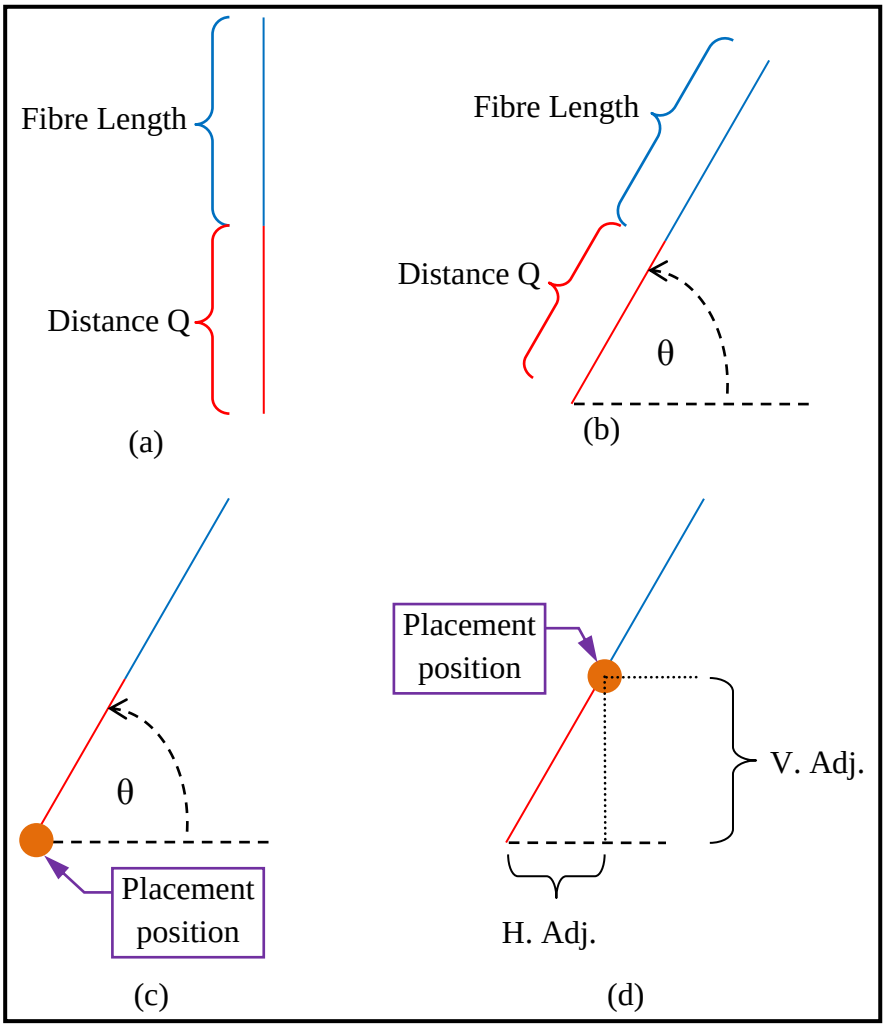


Figure G.6: Linear representations of adjustment positions

In the **Elem_Cov_Adj** function, lines 293 to 295 are used to evaluate the adjustment parameters. The distance Q, shown in Figure G.5, is calculated using line 293. In lines 294 and 295, the horizontal and vertical corrections are determined. The output array

P_Adj is used to store the adjustment parameters for each layer of the element (line 296).

Function: Elem_Pass_Incr

```
298 function [E_Pass,E_Incr] = Elem_Pass_Incr(N_Lay,E_Size,
E_Cov,E_Pln)
%Determine number of passes, increments per layer of each element

299 E_Pass = zeros(N_Lay,2);
300 L_Incr = zeros(N_Lay,2);
301 E_Incr = zeros(N_Lay,3);

302 for t = 1:N_Lay
    %Determine number of horizontal and vertical passes
303     Hpass = E_Size(1)/E_Cov(t,1);
304     Vpass = E_Size(2)/E_Cov(t,2);
305     E_Pass(t,:) = ceil([Hpass Vpass]);
    %Determine horizontal and vertical increments
306     Hinc = E_Size(1)/(E_Pass(t,1)+1);
307     Vinc = E_Size(2)/(E_Pass(t,2)+1);
308     L_Incr(t,:) = [Hinc Vinc];
309 end

310 switch (E_Pln)
311     case 'XY'
312         E_Incr(:,1) = L_Incr(:,1);
313         E_Incr(:,2) = L_Incr(:,2);
314         E_Incr(:,3) = 0;
315     case 'XZ'
316         E_Incr(:,1) = L_Incr(:,1);
317         E_Incr(:,2) = 0;
318         E_Incr(:,3) = L_Incr(:,2);
319     case 'YZ'
320         E_Incr(:,1) = 0;
321         E_Incr(:,2) = L_Incr(:,1);
322         E_Incr(:,3) = L_Incr(:,2);
323     case 'XY-XZ'
324         E_Incr(:,1) = L_Incr(:,1);
325         E_Incr(:,2) = L_Incr(:,2);
326         E_Incr(:,3) = L_Incr(:,2);
327     case 'XY-YZ'
328         E_Incr(:,1) = L_Incr(:,2);
329         E_Incr(:,2) = L_Incr(:,1);
330         E_Incr(:,3) = L_Incr(:,2);
331     case 'XZ-YZ'
332         E_Incr(:,1) = L_Incr(:,2);
333         E_Incr(:,2) = L_Incr(:,2);
334         E_Incr(:,3) = L_Incr(:,1);
```

```
335     case 'ALL'
336         E_Incr(:,1) = L_Incr(:,1);
337         E_Incr(:,2) = L_Incr(:,1);
338         E_Incr(:,3) = L_Incr(:,2);
339 end
```

The inputs of this function (line 298) are the number of layers in an element (*N_Lay*), the size of the element (*E_Size*), the fibre coverage (*E_Cov*), and the planes in which the element resides (*E_Pln*). There are two outputs, namely, the number of passes (*E_Pass*), and the increments (*E_Incr*). The number of passes is based on the fibre coverage, and is the amount of horizontal and vertical placements the tool must perform to completely fill the element with fibres. The increments are the horizontal and vertical distances from one placement position to the next.

The number of passes and the increments for each layer of the element are found using the coded loop in lines 302 to 309. The number of horizontal passes is calculated by dividing the horizontal element length by the horizontal fibre coverage (line 303). Similarly, the code in line 304 computes the number of vertical passes. The output array *E_Pass* is used to store the number of horizontal and vertical passes required for the fibre placements for each layer of the element (line 305).

The horizontal increments, for each layer in the element, is determined by dividing the horizontal element length by the number of horizontal passes (line 306). Likewise, the vertical increments for each layer are evaluated using line 307. The calculated increments are temporarily stored in the *L_Incr* array (line 308).

A fibre placement is performed in a three-dimensional space and therefore consists of X, Y and Z coordinates. Some or all of these coordinates have to be incremented and this depends on which planes the element exists in. The code in lines 310 to 339 stores the X, Y and Z increments, using the *E_Incr* array, for each layer of the element, according to the planes in which the element resides.

Function: Elem_Positions

```
340 function L_Pos = Elem_Positions(N_Lay,Node1,ST_Pos,E_Pl,
```

```
E_Ang,P_Adj,E_Pass,E_Inc,Fib_Ang)
%Determine placement positions and orientations

341 L_Pos = cell(N_Lay,1);
342 for u = 1:N_Lay
343     Num = E_Pass(u,1) * E_Pass(u,2);
344     LL_Pos = zeros(Num,6);
345     Row = 0;
346     for v = 1:E_Pass(u,2)
347         for w = 1:E_Pass(u,1)
348             Row = Row + 1;
349             if (strcmp(E_Pl,'XY')) || (strcmp(E_Pl,'XZ')) ||
                  (strcmp(E_Pl,'XY-XZ'))
350                 Xpos = Node1(1) + w*E_Inc(u,1) - P_Adj(u,1);
351                 Ypos = Node1(2) + v*E_Inc(u,2) - P_Adj(u,2);
352                 Zpos = Node1(3) + v*E_Inc(u,3) - P_Adj(u,2);
353             elseif (strcmp(E_Pl,'YZ')) ||
                  (strcmp(E_Pl,'XY-YZ'))
354                 Xpos = Node1(1) + v*E_Inc(u,1) - P_Adj(u,2);
355                 Ypos = Node1(2) + w*E_Inc(u,2) - P_Adj(u,1);
356                 Zpos = Node1(3) + v*E_Inc(u,3) - P_Adj(u,2);
357             elseif (strcmp(E_Pl,'XZ-YZ'))
358                 Xpos = Node1(1) + v*E_Inc(u,1) - P_Adj(u,2);
359                 Ypos = Node1(2) + v*E_Inc(u,2) - P_Adj(u,2);
360                 Zpos = Node1(3) + w*E_Inc(u,3) - P_Adj(u,1);
361             elseif (strcmp(E_Pl,'ALL'))
362                 Xpos = Node1(1) + w*E_Inc(u,1) - P_Adj(u,1);
363                 Ypos = Node1(2) - w*E_Inc(u,2) + P_Adj(u,1);
364                 Zpos = Node1(3) + v*E_Inc(u,3) - P_Adj(u,2);
365             end
366             Xpos = Xpos + ST_Pos(1);
367             Ypos = Ypos + ST_Pos(2);
368             Zpos = Zpos + ST_Pos(3);
369             Apos = Fib_Ang(u);
370             Bpos = E_Ang(1);
371             Cpos = E_Ang(2);
372             LL_Pos(Row,:) = [Xpos Ypos Zpos Apos Bpos Cpos];
373         end
374     end
375     L_Pos(u) = {LL_Pos};
376 end
```

The **Elem_Positions** function is used to determine the fibre placement coordinates (X, Y and Z) and orientations (A, B and C angles), for each layer of an element. The input parameters (line 340) include the number of fibre layers in the element (*N_Lay*), the X, Y and Z coordinates of the element's first node (*Node1*), the starting position for fibre placement (*ST_Pos*), the planes the element exists in (*E_Pl*), the orientation of

these planes (*E_Ang*), the position corrections (*P_Adj*), the number of horizontal and vertical passes for full fibre coverage (*E_Pass*), the horizontal and vertical increments of these passes (*E_Inc*), and the fibre orientation angle of each layer (*Fib_Ang*). The output array (*L_Pos*) is initialised in line 341.

A coded loop is initiated in line 342 that is executed for every layer of the element under consideration. In line 343, a variable *Num* is assigned the value of the product of the number of horizontal and vertical passes. This value represents the total number of placement positions on that particular layer. An array (*LL_Pos*) is created in line 344 which will be used for the storage of these placement positions. The variable *Row* in line 345 is used as a counter.

A nested loop is initiated in lines 346 and 347. The first loop (line 346) runs for the number of vertical passes in a particular layer, while the second loop (line 347) is executed for the number of horizontal passes of that layer. This means that fibre placements for each layer would be performed horizontally then vertically. In lines 349 to 365, the X coordinate (*Xpos*), Y coordinate (*Ypos*) and Z coordinate (*Zpos*) for the fibre placement positions, of the layer under consideration, are evaluated. These computations are dependent on which planes the element exists in. The calculations include the position of the element's first node (*Node1*), the increment of the horizontal and vertical passes (*E_Inc*), and the position corrections (*P_Adj*).

In lines 366 to 368, the X, Y and Z coordinates of the placement position are modified in order to account for the starting position for fibre placement (*ST_Pos*). In the next three lines of code (lines 369 to 371) values are assigned to the A, B and C angular positions. Position A is the fibre orientation angle of the layer in question, while positions B and C correspond to the previously calculated angles using the **Elem_Plane_Angle** function. The above rectangular and angular positions are stored using the *LL_Pos* array, via line 372, for each horizontal and vertical pass. In line 375, the entries of the *LL_Pos* array are stored, using the *L_Pos* array, for each layer of the element.

It should be noted that the Mitsubishi RV-2AJ robotic arm only has five axes of movement and cannot perform any placements involving the C position. However, the calculation of this position is included to enable the code to be generic, or in other words, the code is able to generate the position list for any robotic manipulator. The functions examined above involve the generation of the position list. The next set of functions are used to produce the command list.

Function: Command_List

```
377 function Command = Command_List(Fib_Len_Lay,FL_Rng)
%Outputs robot commands in MELFA IV format
378 N_Pos = length(Fib_Len_Lay);
379 G_Mark = (roundn(N_Pos*5+5,2)+100)*10;
380 Command = cell(N_Pos*5+21,2);
381 L_No = 10;

%Main Program
382 Command(1,1) = {num2str(L_No)};
383 Command(1,2) = {'M_OUT(36)=1'};
384 for z = 1:N_Pos
385     L_No = L_No + 10;
386     C_No = L_No / 10;
387     Command(C_No,1) = {num2str(L_No)};
388     Command(C_No,2) = {horzcat('GOSUB ',num2str(G_Mark))};
389     L_No = L_No + 10;
390     C_No = L_No / 10;
391     MVal = FibLen_2_MOut(ceil(Fib_Len_Lay(z)),FL_Rng);
392     Command(C_No,1) = {num2str(L_No)};
393     Command(C_No,2) = {horzcat('M_OUT(',num2str(MVal),')=1')};
394     L_No = L_No + 10;
395     C_No = L_No / 10;
396     Command(C_No,1) = {num2str(L_No)};
397     Command(C_No,2) = {horzcat('GOSUB ',num2str(G_Mark+100))};
398     L_No = L_No + 10;
399     C_No = L_No / 10;
400     Command(C_No,1) = {num2str(L_No)};
401     Command(C_No,2) = {horzcat('MOV P',num2str(z),', -30')};
402     L_No = L_No + 10;
403     C_No = L_No / 10;
404     Command(C_No,1) = {num2str(L_No)};
405     Command(C_No,2) = {horzcat('MOV P',num2str(z))};
406 end
407 Command(C_No+1,1) = {num2str(L_No+10)};
408 Command(C_No+1,2) = {horzcat('GOSUB ',num2str(G_Mark))};
409 Command(C_No+2,1) = {num2str(L_No+20)};
410 Command(C_No+2,2) = {'M_OUT(9)=1'};
411 Command(C_No+3,1) = {num2str(L_No+30)};
```

```
412 Command(C_No+3,2) = {'MOV PREST'};
413 Command(C_No+4,1) = {num2str(L_No+40)};
414 Command(C_No+4,2) = {'END'};

%Subprogram 1
415 Command(C_No+5,1) = {num2str(G_Mark)};
416 Command(C_No+5,2) = {'*RELEASE'};
417 Command(C_No+6,1) = {num2str(G_Mark+10)};
418 Command(C_No+6,2) = {'M_OUT(38)=1'};
419 Command(C_No+7,1) = {num2str(G_Mark+20)};
420 Command(C_No+7,2) = {'DLY 2'};
421 Command(C_No+8,1) = {num2str(G_Mark+30)};
422 Command(C_No+8,2) = {'MOV PREST'};
423 Command(C_No+9,1) = {num2str(G_Mark+40)};
424 Command(C_No+9,2) = {'MOV PWAIT'};
425 Command(C_No+10,1) = {num2str(G_Mark+50)};
426 Command(C_No+10,2) = {'RETURN'};

%Subprogram 2
427 Command(C_No+11,1) = {num2str(G_Mark+100)};
428 Command(C_No+11,2) = {'*COLLECT'};
429 Command(C_No+12,1) = {num2str(G_Mark+110)};
430 Command(C_No+12,2) = {'WAIT M_IN(8)=1'};
431 Command(C_No+13,1) = {num2str(G_Mark+120)};
432 Command(C_No+13,2) = {'MOV PCOLLECT, -30'};
433 Command(C_No+14,1) = {num2str(G_Mark+130)};
434 Command(C_No+14,2) = {'MOV PCOLLECT'};
435 Command(C_No+15,1) = {num2str(G_Mark+140)};
436 Command(C_No+15,2) = {'M_OUT(37)=1'};
437 Command(C_No+16,1) = {num2str(G_Mark+150)};
438 Command(C_No+16,2) = {'DLY 2'};
439 Command(C_No+17,1) = {num2str(G_Mark+160)};
440 Command(C_No+17,2) = {'WAIT M_IN(9)=0'};
441 Command(C_No+18,1) = {num2str(G_Mark+170)};
442 Command(C_No+18,2) = {'MOV PWAIT'};
443 Command(C_No+19,1) = {num2str(G_Mark+180)};
444 Command(C_No+19,2) = {'MOV PREST'};
445 Command(C_No+20,1) = {num2str(G_Mark+190)};
446 Command(C_No+20,2) = {'RETURN'};
```

The above function is used to generate the command list. The processes of this function are represented by the flowchart in Figure 6.12. In a typical command list, each line contains a line number followed by the command itself. If there are many repetitive commands, as is the case here, it is common practice to use subprograms. The above function generates a command list that contains a main program and two subprograms. The first subprogram is used to create the commands that release the collected fibres

in the mould, while the second generates the commands that collects the cut fibres from the pulling-and-cutting unit.

The function has two input parameters (line 377), namely, the fibre length of each layer in every element (*Fib_Len_Lay*), and the fibre length range (*FL_Rng*). The output of the function is the command list (*Command*), and it is initialised in line 380. In line 378, the number of fibre placement positions (*N_Pos*) is determined. The line number where the subprograms would begin (*G_Mark*) is calculated in line 379. As mentioned above, each command requires a line number. The variable *L_No* is assigned with the value of the first line number of the list (line 381). In the subsequent coding, each command is created by two lines of code. The first generates the line number, while the second creates the command itself.

According to Figure 6.12, the first step is to turn on the vacuum pump attached to the mould. The command for this is created using lines 382 and 383. The expression 'M_OUT(36)' in line 383 is associated with the output port of the robot controller that switches on the vacuum pump in question. The next steps would be to associate the fibre length with the output of the robot controller, collect the cut fibres from the pulling-and-cutting unit (second subprogram), move to the placement position, and release the fibres in the mould (first subprogram). A coded loop (lines 384 to 406) is executed, for each fibre placement position, to produce the commands that perform these tasks.

In line 385, the line number (*L_No*) is incremented by a value of 10, as this is common practice. The variable *C_No* in line 386 is used as a counter. In lines 387 and 388, the command that executes the first subprogram (discussed later) is created. This does not follow the procedure laid out in Figure 6.12 or discussed above. However, it was decided to execute this subprogram first to ensure that the vacuum pump on the fibre collection-placement tool is switched off, and, more importantly, that the robotic arm is positioned at the point where it waits for the fibres to be cut (PWAIT).

A function, **FibLen_2_MOut**, is executed in line 391. This function, which is discussed later, is used to associate the fibre length with its assigned output port on the robot controller. The code in lines 392 and 393 is used to generate the command that instructs the controller to activate the port given by the function. This would send a signal to the fibre pulling-and-cutting unit to pull the fibre tows according to the number of pulses associated with the fibre length, which is obtained from Table F.1 in Appendix F. The vacuum unit is switched on and the fibres are held in place. The blade cuts the fibres and the unit sends a signal to the controller to indicate that the cut fibres may be collected.

In lines 396 and 397, the command that runs the second subprogram (discussed later) is created. The fibres are pulled, cut and collected, and are ready to be placed in the mould. If the fibre collection-placement tool moves directly to the placement position, it could, while moving across the mould surface, damage the mould or dislodge the placed fibres. Therefore it was decided to move the tool to a point above the placement position and then advance it to the placement point. The commands for these movements are generated in lines 400 and 401, and, lines 404 and 405.

Once the commands for the all the fibre placements have been generated, the first subprogram, which releases the cut fibres in the mould, has to be executed to complete the last fibre placement. The command for this is created by lines 407 and 408. A command to switch off the vacuum pump attached to the mould is generated in lines 409 and 410, and in lines 411 and 412, a command that moves the robotic arm to a position away from the mould (PREST) is created. The command generated by lines 413 and 414 signifies the end of the command list.

As mentioned earlier, the first subprogram creates the commands to release the cut fibres in the mould. It is important to remember that at this point the fibres are in the placement position. The vacuum pump attached to the collection-placement tool is switched off by the command generated in lines 417 and 418. Thereafter commands to move the robotic arm to the PREST and PWAIT positions are created by lines 421 and 422, and, lines 423 and 424, respectively.

The second subprogram is used to generate commands to collect the cut fibres from the pulling-and-cutting unit. The command created in lines 429 and 430 instructs the robot controller to wait for a signal that would indicate that the fibres have been cut and are ready for collection. The robotic arm is moved to the collection position (PCOLLECT) by the commands generated in lines 431 to 434. At this point, the pneumatic cylinder on the collection-placement tool has to be extended to meet the pulling-and-cutting unit in order to collect the cut fibres. The command for this action is created by lines 435 and 436, and this sends a signal to the PLC to switch off the vacuum pump on the pulling-and-cutting unit and switch on the vacuum pump attached to the collection-placement tool. In lines 439 and 440, a command is generated that instructs the robot controller to wait for a signal that would indicate that the fibres have been collected. Commands are created by lines 441 and 442 to move the robotic arm to the PWAIT position, while lines 443 and 444 generates the command to move the robotic arm to the PREST position.

Function: FibLen_2_MOut

```
447 function MOutVal = FibLen_2_MOut(FL,RNG)
%Matches fibre length to preset robot controller output
448 FibLenSet = RNG(1):RNG(2);
449 MOutSet = 40: 40+length(FibLenSet);
450 [~,index] = ismember(FL,FibLenSet);
451 if index ~= 0
452     MOutVal = MOutSet(index);
453 end
```

The inputs (line 447) for the above function include the fibre length (*FL*) and the fibre length range (*RNG*). In line 448, a set containing all the possible fibre lengths, based on the fibre length range, is created (*FibLenSet*). A set containing the output ports of the robot controller (*MOutSet*) is produced in line 449. The code in line 450 is used to determine the position of the input fibre length (*FL*) in the fibre length set (*FibLenSet*). The output variable (*MOutVal*) is assigned the value at this position in the output port set (*MOutSet*) (lines 451 to 453). This value represents the output port of the robot controller.

The functions involved with the generation of the position and command lists have been examined. A Graphical User Interface (GUI) was also developed for this section and the Matlab code is shown and discussed below.

Graphical User Interface (GUI) code

As mentioned in Appendix B, a GUI consists of a figure file and a code file. The figure file contains the graphical layout, while the code file includes the various functions required for the operation of the GUI. The figure file for the GUI in this section is shown in Figure 6.14, and the code file, which replaces the main script file, is shown and discussed below.

```
1    function varargout = GUI_Robot(varargin)
% GUI_ROBOT M-file for GUI_Robot.fig
% GUI_ROBOT, by itself, creates a new GUI_ROBOT or raises the
existing singleton*.
% H = GUI_ROBOT returns the handle to a new GUI_ROBOT or the handle
To the existing singleton*.
% GUI_ROBOT('CALLBACK',hObject,eventData,handles,...) calls the
Local function named CALLBACK in GUI_ROBOT.M with the given input
arguments.
% GUI_ROBOT('Property','Value',...) creates a new GUI_ROBOT or
raises the existing singleton*.  Starting from the left, property
value pairs are applied to the GUI before GUI_Robot_OpeningFcn
gets called.  An unrecognized property name or invalid value makes
property application stop.  All inputs are passed to
GUI_Robot_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help GUI_Robot

% Begin initialization code - DO NOT EDIT
2    gui_Singleton = 1;
3    gui_State = struct('gui_Name',       mfilename, ...
                 'gui_Singleton',  gui_Singleton, ...
                 'gui_OpeningFcn', @GUI_Robot_OpeningFcn, ...
                 'gui_OutputFcn',  @GUI_Robot_OutputFcn, ...
                 'gui_LayoutFcn',  [] , ...
                 'gui_Callback',   []);
4    if nargin && ischar(varargin{1})
5        gui_State.gui_Callback = str2func(varargin{1});
6    end

7    if nargout
8        [varargout{1:nargout}] = gui_mainfcn(gui_State,
         varargin{:});
9    else
```

```
10      gui_mainfcn(gui_State, varargin{:});
11  end
% End initialization code - DO NOT EDIT


% --- Executes just before GUI_Robot is made visible.
12  function GUI_Robot_OpeningFcn(hObject, eventdata, handles,
    varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_Robot (see VARARGIN)

% Choose default command line output for GUI_Robot
13  handles.output = hObject;
14  M_M = find(strcmp(varargin, 'MainMenu'));
15  handles.MainMenuVar = varargin{M_M+1};
16  handles.HelpVal = 0;
17  handles.ModelVal = 0;
% Update handles structure
18  guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
19  function Pic_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Pic (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
% Hint: place code in OpeningFcn to populate Pic
20  axes(hObject)
21  imshow('Layers.jpg')

% --- Outputs from this function are returned to the command line.
22  function varargout = GUI_Robot_OutputFcn(hObject, eventdata,
    handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
23  varargout{1} = handles.output;


%---------P R O G R A M M I N G   S T A R T S   H E R E----------%
24  Pic_CreateFcn(handles.Pic, eventdata, handles);
25  set(handles.Robot_Import_Table,'Data',[]);
26  set(handles.Robot_Pos_Table,'Data',[]);
27  set(handles.Robot_Com_Table,'Data',[]);
28  set(handles.Data_1,'Style','edit');
29  set(handles.Data_2,'Style','edit');
30  set(handles.Data_3,'Style','edit');
```

```
31  set(handles.Data_4,'Style','edit');
32  set(handles.Data_5,'Style','edit');
33  set(handles.Data_6,'Style','edit');
34  set(handles.Data_7,'Style','edit');
35  set(handles.Data_8,'Style','edit');
36  set(handles.Data_9,'Style','edit');
37  uiwait(handles.Figure_Robot)

%-----Close Request Function - executes when plot is deleted-----%
38  function my_closereq(src, eventdata, hObject, handles)
39  delete(handles.ModelFig);
40  handles.ModelVal = 0;
41  guidata(hObject,handles);


%---I M P O R T   P A N E L   E X E C U T A B L E   B U T T O N S--%
% --- Executes on button press in Robot_Import_But.
42  function Robot_Import_But_Callback(hObject, eventdata, handles)
% hObject    handle to Robot_Import_But (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
43  [handles.FileName,PathName] = uigetfile('*.rslt','Select the
    results file');
44  if (handles.FileName ~= 0)
45      OpenFile = [PathName,handles.FileName];
46      RR = load(OpenFile,'-mat');
47      handles.Name = RR.Name;
48      handles.Material = RR.Material;
49      handles.Results = RR.Results;
50      handles.FEM = RR.FEM;

%Populate Import Panel table with values
51      N_El = size(handles.Results,1);
52      Results_Data = cell(N_El,4);
53      for v = 1:N_El
54          El_Lay = length(handles.Results(v).Angle);
55          El_Ang = num2str(handles.Results(v).Angle);
56          El_Thick = num2str(handles.Results(v).Thick);
57          Results_Data(v,:) = {v El_Lay El_Ang El_Thick};
58      end
59      set(handles.Robot_Import_Table,'Data',Results_Data);
60      set(handles.Robot_Info_2,'String',handles.Name{1});
61      handles.ModelFig = figure('NumberTitle','off','Position',
        [700 330 460 460]);
62      set(handles.ModelFig,'CloseRequestFcn',{@my_closereq,
        hObject,handles});
63      handles.ModelVal = 1;
64      plotfem(handles.FEM,'*-.',handles.ModelFig,'silent',
        'shaded','title','Model Plot Window');
65      guidata(hObject,handles);
66  end
```

```
% --- Executes on button press in Robot_Help.
67  function Robot_Help_Callback(hObject, eventdata, handles)
% hObject    handle to Robot_Help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
68  handles.HelpFig = Help_Robot;
69  handles.HelpVal = 1;
70  guidata(hObject,handles);

% --- Executes on button press in Robot_Exit.
71  function Robot_Exit_Callback(hObject, eventdata, handles)
% hObject    handle to Robot_Exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
72  Figure_Robot_CloseRequestFcn(handles.Figure_Robot, eventdata,
    handles)

% --- Executes on button press in Robot_Discard.
73  function Robot_Discard_Callback(hObject, eventdata, handles)
% hObject    handle to Robot_Discard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
74  set(handles.Robot_Info_2,'String','');
75  set(handles.Robot_Import_Table,'Data',[]);
76  set(handles.Robot_Pos_Table,'Data',[]);
77  set(handles.Robot_Com_Table,'Data',[]);
78  if ishandle(handles.ModelFig)
79      delete(handles.ModelFig);
80  end

% --- Executes on button press in Robot_Redraw.
81  function Robot_Redraw_Callback(hObject, eventdata, handles)
% hObject    handle to Robot_Redraw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
82  Data = get(handles.Robot_Import_Table,'Data');
83  if size(Data,1) == 0
84      errordlg('There is no data available to redraw
        figure','ERROR: No Data','modal');
85  elseif handles.ModelVal == 0
86      handles.ModelFig = figure('NumberTitle','off','Position',
        [700 330 460 460]);
87      set(handles.ModelFig,'CloseRequestFcn',
        {@my_closereq,hObject,handles});
88      handles.ModelVal = 1;
89      plotfem(handles.FEM,'*-.',handles.ModelFig,'silent',
        'shaded','title','Model Plot Window');
90      guidata(hObject,handles);
91  end
```

```
%----I N P U T   P A N E L   E X E C U T A B L E   B U T T O N S---%
% --- Executes on button press in Robot_Code.
92  function Robot_Code_Callback(hObject, eventdata, handles)
% hObject    handle to Robot_Code (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
93  Data = get(handles.Robot_Import_Table,'Data');
94  Scale = str2double(get(handles.Data_1,'String'));
95  T_Wid = str2double(get(handles.Data_2,'String'));
96  FL_Min = str2double(get(handles.Data_3,'String'));
97  FL_Max = str2double(get(handles.Data_4,'String'));
98  S_P_X = str2double(get(handles.Data_5,'String'));
99  S_P_Y = str2double(get(handles.Data_6,'String'));
100 S_P_Z = str2double(get(handles.Data_7,'String'));
101 Tool_H = str2double(get(handles.Data_8,'String'));
102 Tool_W = str2double(get(handles.Data_9,'String'));

103 if size(Data,1) == 0
104     errordlg('There are no results available to generate
        code','ERROR: No Results','modal');
105 elseif isnan(Scale)
106     errordlg('Please enter proper Scaling Factor','INPUT
        ERROR','modal');
107 elseif isnan(T_Wid)
108     errordlg('Please enter proper Tow Width','INPUT
        ERROR','modal');
109 elseif isnan(FL_Min)
110     errordlg('Please enter proper minimum Fibre Length','INPUT
        ERROR','modal');
111 elseif isnan(FL_Max)
112     errordlg('Please enter proper maximum Fibre Length','INPUT
        ERROR','modal');
113 elseif isnan(S_P_X)
114     errordlg('Please enter proper starting X Co-
        ordinate','INPUT ERROR','modal');
115 elseif isnan(S_P_Y)
116     errordlg('Please enter proper starting Y Co-
        ordinate','INPUT ERROR','modal');
117 elseif isnan(S_P_Z)
118     errordlg('Please enter proper starting Z Co-
        ordinate','INPUT ERROR','modal');
119 elseif isnan(Tool_H)
120     errordlg('Please enter proper Tool Height','INPUT
        ERROR','modal');
121 elseif isnan(Tool_W)
122     errordlg('Please enter proper Tool Width','INPUT
        ERROR','modal');
123 elseif Scale <= 0
124     errordlg('Please enter non-zero or positive value for
        Scaling Factor','INPUT ERROR','modal');
125 elseif T_Wid <= 0
126     errordlg('Please enter non-zero or positive value for Tow
```

```
        Width','INPUT ERROR','modal');
127 elseif FL_Min <= 0
128     errordlg('Please enter non-zero or positive value for
        minimum Fibre Length','INPUT ERROR','modal');
129 elseif FL_Max <= 0
130     errordlg('Please enter non-zero or positive value for
        maximum Fibre Length','INPUT ERROR','modal');
131 elseif Tool_H <= 0
132     errordlg('Please enter non-zero or positive value for Tool
        Height','INPUT ERROR','modal');
133 elseif Tool_W <= 0
134     errordlg('Please enter non-zero or positive value for Tool
        Width','INPUT ERROR','modal');
135 else
%Position List and Command List
136     [Fib_Len,handles.Pos_List] = Position_List(handles.FEM,
        handles.Results,Scale,T_Wid,[FL_Min FL_Max],
        [Tool_H Tool_W],[S_P_X S_P_Y S_P_Z]);
137     handles.Commands = Command_List(Fib_Len,[FL_Min FL_Max]);
%Populate tables
138     set(handles.Robot_Pos_Table,'Data',handles.Pos_List);
139     set(handles.Robot_Com_Table,'Data',handles.Commands);
140     guidata(hObject,handles);
141 end


%R O B O T  C O D E  P A N E L  E X E C U T A B L E  B U T T O N S%
% --- Executes on button press in Code_Save.
142 function Code_Save_Callback(hObject, eventdata, handles)
% hObject    handle to Code_Save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
143 [FileName1,PathName1] = uiputfile('*.pos','Choose destination
    of Positions file');
144 if (FileName1 ~= 0)
145     SaveFile1 = [PathName1,FileName1];
146     [N_Pos,~] = size(handles.Pos_List);
147     fid1 = fopen(SaveFile1,'w');
148     for PP = 1:N_Pos
149         fprintf(fid1,'%s=(%.2f,%.2f,%.2f,%.2f,%.2f,%.2f)(6,0)
            \n',handles.Pos_List{PP,:});
150     end
151     fclose(fid1);
152 end
153 [FileName2,PathName2] = uiputfile('*.mb4','Choose destination
    of Command file');
154 if (FileName2 ~= 0)
155     SaveFile2 = [PathName2,FileName2];
156     [N_Com,~] = size(handles.Commands);
157     fid2 = fopen(SaveFile2,'w');
158     for CC = 1:N_Com
159         fprintf(fid2,'%s %s\n',handles.Commands{CC,:});
```

```
160     end
161     fclose(fid2);
162 end

% --- Executes on button press in Code_Discard.
163 function Code_Discard_Callback(hObject, eventdata, handles)
% hObject    handle to Code_Discard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
164 set(handles.Robot_Pos_Table,'Data',[]);
165 set(handles.Robot_Com_Table,'Data',[]);


%-----------C L O S E   O R   E X I T   P R O G R A M-------------%
% --- Executes when user attempts to close Figure_Robot.
166 function Figure_Robot_CloseRequestFcn(hObject, eventdata,
    handles)
% hObject    handle to Figure_Robot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: delete(hObject) closes the figure
167 if handles.HelpVal == 1
168     if ishandle(handles.HelpFig)
169         delete(handles.HelpFig);
170     end
171 end
172 if handles.ModelVal == 1
173     if ishandle(handles.ModelFig)
174         delete(handles.ModelFig);
175     end
176 end
177 mainHandles = guidata(handles.MainMenuVar);
178 set(mainHandles.Check,'String','GO');
179 uiresume(handles.Figure_Robot)
180 delete(handles.Figure_Robot);
```

As mentioned previously, the GUI is a function assignment (line 1) and is capable of accepting various inputs (*varargin*) and generating multiple outputs (*varargout*). The code in lines 2 to 11 represents the initialisation of the GUI. Here the basic tasks of the GUI are set up for use and the input and output variables (if any) are recognised.

The code from line 12 to 18 is run prior to the GUI becoming visible. As in the case of the GUIs examined previously, the execution of other GUIs from the main graphical interface (Chapter 7) is prohibited via the code in lines 13 to 15, and line 37. Other GUIs may be executed once this GUI has been properly exited. In lines 16 and 17,

variables that assist in determining whether the help function was used or if the model plot window is open, is initialised. The newly created variables are stored in the GUI database via line 18. The function from line 19 to 21 is used to create the axes that displays the GUI's logo.

In line 24 the function that displays the logo is executed. All data from the tables on the Import Panel and the Robot Code Panel are cleared by the code in lines 25 to 27. In lines 28 to 36, the data boxes on the Input Panel are set to be editable. The function from line 38 to 41 is used to control the closing or exiting criteria of the model plot window.

The code in lines 42 to 91 is used to govern the operations of the Import Panel's five pushbuttons. The first of these buttons is "IMPORT" and its purpose is to load a set of saved results from the variable stiffness design method. In line 43, a window, where the user may browse for the saved results, is opened. If a file has been selected (line 44), it is loaded into the GUI via lines 45 and 46. The data stored in the saved file is assigned to variables in lines 47 to 50. Lines 51 to 60 are used to display some of this data in the table on the Import Panel. Further, a plot window is opened that displays the finite element model. This action is achieved by using lines 61 to 64 of the code. The variables used in lines 47 to 50 are stored in the GUI database via line 65.

The second of the five buttons on the Import Panel is "HELP". This button activates the help facility of the GUI via line 68. In line 69, the variable created above, in line 16, is assigned a value of 1 to establish that the help window has been opened. This information is required when the GUI is exited to determine whether the help window is still open. The code regarding the closing of the help window is discussed later.

The next two buttons on the Import Panel are "EXIT" and "DISCARD DATA". If the "EXIT" button is clicked, then the execution of the code in lines 71 and 72 is redirected to line 166 where the GUI is properly closed. The function for the "DISCARD DATA" button is from line 73 to 80. When this button is pressed, all data is cleared from the

table on the Import Panel as well as the two tables on the Robot Code Panel. Further, the plot window with the finite element model is also closed.

The last button on the Import Panel is “REDRAW FIGURE”. The button is used to recreate the plot of the finite element model, of the current results, if the plot window was previously closed. The function for this button is from line 81 to 91. The data is extracted from the table on the Import Panel via line 82. If no data is available, an error message is displayed (line 84). Otherwise, the model is replotted in a new window using lines 86 to 90.

The next panel from the GUI to be examined is the Input Panel. This panel contains a number of textboxes where the input parameters, for the generation of the position and command lists, are entered. These inputs include the scaling factor, tow width, minimum and maximum fibre lengths, starting position for fibre layup, and the dimensions of the collection-placement tool. Upon initialisation of the GUI, the data boxes contain default values for the setup used in this study.

There is one pushbutton on the Input Panel called “GENERATE”. When this button is pressed, the function contained in lines 92 to 141 is used to produce and display the position and command lists. Data is extracted from the table on the Import Panel as well as from the textboxes on the Input Panel using lines 93 to 102. The code in line 103 is used to determine if the table on the Import Panel was populated. If the table is empty, an error message is displayed (line 104). The data from the textboxes on the Input Panel are tested in lines 105 to 122 to evaluate if they are numeric. If this is not the case, suitable error messages are displayed that prompt the user to rectify the error. In lines 123 to 134, the data from the textboxes are tested to determine whether they are positive and non-zero. Here as well, error messages are displayed if any of the tests are not passed. If no errors are encountered or if the errors are corrected, the **Position_List** and **Command_List** functions in lines 136 and 137, respectively, are executed. The generated lists are displayed in the two tables on the Robot Code Panel via lines 138 and 139.

The Robot Code Panel contains two pushbuttons, namely, “SAVE” and “DISCARD”. The “SAVE” button is used to write the contents of the position list to a file with a .pos extension (lines 145 to 151), and the contents of the command list to a file with the extension .mb4 (lines 153 to 161). The code for the “DISCARD” button in lines 163 to 165 is used to clear all the data from the two tables on the Robot Code Panel.

The last function of the GUI code (lines 166 to 180) ensures that it is shutdown in a proper manner. The code in lines 167 to 171 is used to determine if the help window is open, and, if so, to close it. Similarly, the finite element plot window (if open) is closed via lines 172 to 176. As mentioned earlier, the GUI was locked to prevent execution of other GUIs from the main graphical interface. The code in lines 177 to 179 is used to unlock the GUI and close it via line 180.

The GUI as well as the other functions involved with the generation of the position and command lists have been examined. These lists may now be uploaded to the robot controller to perform the fibre layup.

APPENDIX H: MATLAB CODE FOR DATABASE OF MATERIALS

The GUIs for the CSM and VSM codes utilised a database that contained properties for different composite materials. As discussed in Chapter 7, a Matlab code was developed for the management of this database as well as databases for fibre and matrix properties. In this appendix, the line-by-line analysis of this Matlab code is performed. The complete code consisted of a GUI, and a function that calculated a composite material's properties via micromechanical methods.

Graphical User Interface (GUI) code

Unlike the others, this GUI has different layouts that depend on the operation being performed. These include viewing, editing, adding and deleting of materials and properties from the three databases. Figures 7.5 to 7.11 show the various layouts of the GUI. The Matlab code for this GUI is shown below.

```
1    function varargout = GUI_Database(varargin)
% GUI_DATABASE M-file for GUI_Database.fig
% GUI_DATABASE, by itself, creates a new GUI_DATABASE or raises the
  Existing singleton*.
% H = GUI_DATABASE returns the handle to a new GUI_DATABASE or the
  handle to the existing singleton*.
% GUI_DATABASE('CALLBACK',hObject,eventData,handles,...) calls the
  local function named CALLBACK in GUI_DATABASE.M with the given
  input arguments.
% GUI_DATABASE('Property','Value',...) creates a new GUI_DATABASE or
  raises the existing singleton*.  Starting from the left, property
  value pairs are applied to the GUI before GUI_Database_OpeningFcn
  gets called.  An unrecognized property name or invalid value makes
  property application stop.  All inputs are passed to
  GUI_Database_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
  one instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help GUI_Database

% Begin initialization code - DO NOT EDIT
2    gui_Singleton = 1;
3    gui_State = struct('gui_Name',       mfilename, ...
           'gui_Singleton',  gui_Singleton, ...
           'gui_OpeningFcn', @GUI_Database_OpeningFcn, ...
           'gui_OutputFcn',  @GUI_Database_OutputFcn, ...
           'gui_LayoutFcn',  [] , ...
           'gui_Callback',   []);
4    if nargin && ischar(varargin{1})
```

```
5           gui_State.gui_Callback = str2func(varargin{1});
6       end
7       if nargout
8           [varargout{1:nargout}] = gui_mainfcn(gui_State,
            varargin{:});
9       else
10          gui_mainfcn(gui_State, varargin{:});
11      end
% End initialization code - DO NOT EDIT

% --- Executes just before GUI_Database is made visible.
12      function GUI_Database_OpeningFcn(hObject, eventdata, handles,
        varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_Database (see VARARGIN)

% Choose default command line output for GUI_Database
13      handles.output = hObject;
14      handles.DB_I = get(handles.Database_Panel,'SelectedObject');
15      handles.Man_Aut = '0';
16      Pict_Axes_CreateFcn(handles.Pict_Axes, eventdata, handles);
17      Output_Panel_Add_Matrix_Callback(handles.Output_Panel_Add_Matri
        x, eventdata, handles);
18      Output_Panel_Add_Fibre_Callback(handles.Output_Panel_Add_Fibre,
        eventdata, handles);
19      M_M = find(strcmp(varargin, 'MainMenu'));
20      handles.MainMenuVar = varargin{M_M+1};
21      handles.HelpVal = 0;
% Update handles structure
22      guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
23      function Pict_Axes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Pict_Axes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
  called
% Hint: place code in OpeningFcn to populate Pict_Axes
24      axes(hObject)
25      imshow('Layers.jpg')

% --- Outputs from this function are returned to the command line.
26      function varargout = GUI_Database_OutputFcn(hObject, eventdata,
        handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
27      varargout{1} = handles.output;
```

```
%---------P R O G R A M M I N G   S T A R T S   H E R E----------%
28  uiwait(handles.Figure_Database)

%--- Makes all sub-panels invisible and resets databoxes
29  function clear_panels(hObject, eventdata, handles)
30  set(handles.Database_Panel,'Visible','off');
31  set(handles.Database_Panel,'SelectedObject',handles.DB_I);
32  set(handles.Material_Panel,'Visible','off');
33  set(handles.Output_Panel,'Visible','off');
34  set(handles.Output_Panel_Info_4,'String','');
35  set(handles.Output_Panel_Exec,'Visible','off');
36  set(handles.Output_Panel_Add_Manual,'Visible','off');
37  set(handles.Output_Panel_Add_Auto,'Visible','off');
38  set(handles.Output_Table,'Data',[]);
39  set(handles.Output_Table,'Visible','off');

%--- Makes add sub-panels invisible and resets databoxes
40  function clear_add(hObject, eventdata, handles)
41  set(handles.Output_Panel_Info_4,'String','');
42  set(handles.Output_Table,'Data',[]);
43  set(handles.Add_Panel,'Visible','off');
44  set(handles.Output_Panel_Add_Vol,'String','');
45  set(handles.Output_Panel_Add_Thick,'String','')


%--------F I G U R E   E X E C U T A B L E   B U T T O N S--------%
% --- Executes on button press in Database_Help.
46  function Database_Help_Callback(hObject, eventdata, handles)
% hObject    handle to Database_Help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
47  handles.HelpFig = Help_DatMan;
48  handles.HelpVal = 1;
49  guidata(hObject,handles);

% --- Executes on button press in Database_Close.
50  function Database_Close_Callback(hObject, eventdata, handles)
% hObject    handle to Database_Close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
51  Figure_Database_CloseRequestFcn(handles.Figure_Database,
    eventdata, handles);


%O P E R A T I O N S   P A N E L   E X E C U T A B L E   B U T T O N S%
% --- Executes on button press in OP_Button_View.
52  function OP_Button_View_Callback(hObject, eventdata, handles)
% hObject    handle to OP_Button_View (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
53  clear_panels(hObject, eventdata, handles);
54  clear_add(hObject, eventdata, handles);
```

```
55   set(handles.Output_Panel,'BackgroundColor',[0.0431373 0.517647
     0.780392]);
56   set(handles.Output_Panel,'Title','VIEWING PROPERTIES FOR');
57   set(handles.Output_Panel_Exec,'String','');
58   set(handles.Database_Panel,'visible', 'on');
59   handles.OP_Button = 'View';
60   guidata(hObject,handles);

% --- Executes on button press in OP_Button_Edit.
61   function OP_Button_Edit_Callback(hObject, eventdata, handles)
% hObject    handle to OP_Button_Edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
62   clear_panels(hObject, eventdata, handles);
63   clear_add(hObject, eventdata, handles);
64   set(handles.Output_Panel,'BackgroundColor',[0 0.498039 0]);
65   set(handles.Output_Panel,'Title','EDITING PROPERTIES FOR');
66   set(handles.Output_Panel_Exec,'String','OK');
67   set(handles.Database_Panel,'visible', 'on');
68   handles.OP_Button = 'Edit';
69   guidata(hObject,handles);

% --- Executes on button press in OP_Button_Add.
70   function OP_Button_Add_Callback(hObject, eventdata, handles)
% hObject    handle to OP_Button_Add (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
71   clear_panels(hObject, eventdata, handles);
72   clear_add(hObject, eventdata, handles);
73   set(handles.Output_Panel,'BackgroundColor',[0.74902 0
     0.74902]);
74   set(handles.Output_Panel,'Title','ADDING PROPERTIES TO');
75   set(handles.Output_Panel_Exec,'String','ADD');
76   set(handles.Database_Panel,'visible', 'on');
77   handles.OP_Button = 'Add';
78   guidata(hObject,handles);

% --- Executes on button press in OP_Button_Del.
79   function OP_Button_Del_Callback(hObject, eventdata, handles)
% hObject    handle to OP_Button_Del (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
80   clear_panels(hObject, eventdata, handles);
81   clear_add(hObject, eventdata, handles);
82   set(handles.Output_Panel,'BackgroundColor',[1 0 0]);
83   set(handles.Output_Panel,'Title','DELETING PROPERTIES FOR');
84   set(handles.Output_Panel_Exec,'String','DELETE');
85   set(handles.Database_Panel,'visible', 'on');
86   handles.OP_Button = 'Del';
87   guidata(hObject,handles);
```

```
%D A T A B A S E   P A N E L   S E L E C T A B L E   B U T T O N S%
% --- Executes when selected object is changed in Database_Panel.
88  function Database_Panel_SelectionChangeFcn(hObject, eventdata,
    handles)
% hObject    handle to the selected object in Database_Panel
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if
    none was selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
89  CH = get(eventdata.NewValue,'String');
90  switch CH
91      case ('Composite')
92          handles.FileName = 'Database-Comp.mat';
93      case ('Fibre')
94          handles.FileName = 'Database-Fib.mat';
95      case ('Matrix')
96          handles.FileName = 'Database-Mat.mat';
97  end
98  guidata(hObject,handles);
99  setMaterialListString(handles.Material_List,eventdata,handles);
100 set(handles.Material_List,'Value',1);
101 set(handles.Output_Panel,'visible','on');
102 set(handles.Output_Panel_Info_3,'string',CH);

103 if strcmp(handles.OP_Button,'Add') == 0
104     set(handles.Material_Panel,'Visible','on');
105     set(handles.Output_Table,'Visible','on');
106 end
107 if strcmp(handles.OP_Button,'View') == 0
108     set(handles.Output_Panel_Exec,'Visible','on');
109 end
110 if strcmp(handles.OP_Button,'Add') == 1
111     if strcmp(CH,'Composite')
112         set(handles.Output_Panel_Add_Manual,'Visible','on');
113         set(handles.Output_Panel_Add_Auto,'Visible','on');
114         set(handles.Output_Table,'Visible','off');
115     else
116         set(handles.Output_Panel_Add_Manual,'Visible','off');
117         set(handles.Output_Panel_Add_Auto,'Visible','off');
118         set(handles.Add_Panel,'Visible','off');
119         set(handles.Output_Table,'Visible','on');
120         Material_List_Callback(hObject, eventdata, handles);
121     end
122 end

% --- Sets content of Material_List with composite materials
123 function setMaterialListString(hObject,eventdata,handles)
124 list = load(handles.FileName,'Name');
125 set(hObject,'string',list.Name);
```

```
%---M A T E R I A L   P A N E L   S E L E C T A B L E   L I S T---%
% --- Executes during object creation, after setting all properties.
126 function Material_List_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Material_List (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
  called
% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
127 if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
128     set(hObject,'BackgroundColor','white');
129 end

% --- Executes on selection change in Material_List.
130 function Material_List_Callback(hObject, eventdata, handles)
% hObject    handle to Material_List (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns
  Material_List contents as cell array
  contents{get(hObject,'Value')} returns selected item from
  Material_List
131 MAT = get(handles.Material_List,'Value');
132 Material = load(handles.FileName);
133 if (strcmp(handles.OP_Button,'Edit')) ||
    (strcmp(handles.OP_Button,'Add'))
134     set(handles.Output_Panel_Info_4,'Style','edit');
135     ColEdit = true(1,length(Material.Att));
136 else
137     set(handles.Output_Panel_Info_4,'Style','text');
138     ColEdit = false(1,length(Material.Att));
139 end
140 if (strcmp(handles.OP_Button,'Add')) == 0
141     set(handles.Output_Panel_Info_4,'string',
        Material.Name(MAT));
142     set(handles.Output_Table,'Data',Material.Data(MAT,:));
143 else
144     set(handles.Output_Panel_Info_4,'string','');
145     set(handles.Output_Table,'Data',
        zeros(1,length(Material.Att)));
146 end

147 set(handles.Output_Table,'ColumnWidth','auto');
148 set(handles.Output_Table,'ColumnName',Material.Att);
149 set(handles.Output_Table,'ColumnEditable',ColEdit);


%-----O U T P U T   P A N E L   D R O P - D O W N   M E N U S-----%
% --- Executes during object creation, after setting all properties.
150 function Output_Panel_Add_Matrix_CreateFcn(hObject, eventdata,
    handles)
% hObject    handle to Output_Panel_Add_Matrix (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
  called
% Hint: popupmenu controls usually have a white background on
  Windows.
%       See ISPC and COMPUTER.
151 if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
152     set(hObject,'BackgroundColor','white');
153 end

% --- Executes on selection change in Output_Panel_Add_Matrix.
154 function Output_Panel_Add_Matrix_Callback(hObject, eventdata,
    handles)
% hObject    handle to Output_Panel_Add_Matrix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns
  Output_Panel_Add_Matrix contents as cell array
  contents{get(hObject,'Value')} returns selected item from
  Output_Panel_Add_Matrix
155 list = load('Database-Mat.mat','Name');
156 set(hObject,'string',list.Name);

% --- Executes during object creation, after setting all properties.
157 function Output_Panel_Add_Fibre_CreateFcn(hObject, eventdata,
    handles)
% hObject    handle to Output_Panel_Add_Fibre (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
  called
% Hint: popupmenu controls usually have a white background on
  Windows.
%       See ISPC and COMPUTER.
158 if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
159     set(hObject,'BackgroundColor','white');
160 end

% --- Executes on selection change in Output_Panel_Add_Fibre.
161 function Output_Panel_Add_Fibre_Callback(hObject, eventdata,
    handles)
% hObject    handle to Output_Panel_Add_Fibre (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns
    Output_Panel_Add_Fibre contents as cell array contents{get
(hObject,'Value')} returns selected item from Output_Panel_Add_Fibre
162 list = load('Database-Fib.mat','Name');
163 set(hObject,'string',list.Name);


%---O U T P U T   P A N E L   E X E C U T A B L E   B U T T O N S--%
% --- Executes on button press in Output_Panel_Add_Manual.
```

```
164 function Output_Panel_Add_Manual_Callback(hObject, eventdata,
    handles)
% hObject    handle to Output_Panel_Add_Manual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
165 clear_add(hObject, eventdata, handles);
166 set(handles.Output_Panel_Info_4,'Style','edit');
167 Material = load(handles.FileName);
168 set(handles.Output_Table,'Visible','on');
169 set(handles.Output_Table,'ColumnWidth','auto');
170 set(handles.Output_Table,'ColumnName',Material.Att);
171 set(handles.Output_Table,'ColumnEditable',true(1,length(Materia
    l.Att)));
172 set(handles.Output_Table,'Data',zeros(1,length(Material.Att)));
173 handles.Man_Aut = 'Manual-C';
174 guidata(hObject,handles);

% --- Executes on button press in Output_Panel_Add_Auto.
175 function Output_Panel_Add_Auto_Callback(hObject, eventdata,
    handles)
% hObject    handle to Output_Panel_Add_Auto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
176 clear_add(hObject, eventdata, handles);
177 set(handles.Output_Table,'Visible','off');
178 set(handles.Output_Panel_Info_4,'Style','edit');
179 set(handles.Add_Panel,'Visible','on');
180 Output_Panel_Add_Matrix_Callback(handles.Output_Panel_Add_Matri
    x,eventdata,handles);
181 Output_Panel_Add_Fibre_Callback(handles.Output_Panel_Add_Fibre,
    eventdata,handles);
182 handles.Man_Aut = 'Auto';
183 guidata(hObject,handles);

% --- Executes on button press in Output_Panel_Exec.
184 function Output_Panel_Exec_Callback(hObject, eventdata,
    handles)
% hObject    handle to Output_Panel_Exec (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
185 Material = load(handles.FileName);
186 index = get(handles.Material_List,'Value');

187 switch handles.OP_Button
188     case 'Edit'
189         if strcmp(get(handles.Output_Panel_Info_4,'string'),'')
190             errordlg('Please enter a Material Name',
                'ERROR: Name','modal');
191             GO = 0;
192         else
193             Material.Name(index) = get
                (handles.Output_Panel_Info_4,'string');
```

```
194             Material.Data(index,:) = get(handles.Output_Table,
                'Data');
195             GO = 1;
196         end
197     case 'Del'
198         quest = horzcat('Are you sure you want to delete',
                Material.Name(index),'from the database');
199         ANSWER = questdlg(quest,'Delete Confirmation',
                'YES','NO','NO');
200         switch ANSWER
201             case 'YES'
202                 Material.Name(index)='';
203                 Material.Data(index,:)=[];
204                 GO = 1;
205             otherwise
206                 GO = 0;
207         end
208     case 'Add'
209         pos = length(Material.Name) + 1;
210         if (strcmp(handles.FileName,'Database-Fib.mat')) ||
                (strcmp(handles.FileName,'Database-Mat.mat'))
211             handles.Man_Aut = 'Manual-FM';
212         end
213         if (strcmp(handles.Man_Aut,'0')) &&
                (strcmp(handles.FileName,'Database-Comp.mat'))
214             errordlg('Please choose "Manual" or "Auto"',
                'ERROR','modal');
215             GO = 0;
216         elseif strcmp(get(handles.Output_Panel_Info_4,
                'string'),'')
217             errordlg('Please enter a Material Name',
                'INPUT ERROR: Name','modal');
218         else
219             Material.Name{pos} = get
                (handles.Output_Panel_Info_4,'string');
220         end

221         switch handles.Man_Aut
222             case 'Manual-FM'
223                 if strcmp(get(handles.Output_Panel_Info_4,
                    'string'),'') == 0
224                     Material.Data(pos,:) = get
                        (handles.Output_Table,'Data');
225                     GO = 1;
226                 else
227                     GO = 0;
228                 end
229             case 'Manual-C'
230                 if strcmp(get(handles.Output_Panel_Info_4,
                    'string'),'') == 0
231                     Material.Data(pos,:) = get
                        (handles.Output_Table,'Data');
```

```
232                     set(handles.Output_Table,'Visible','off');
233                     GO = 1;
234                 else
235                     GO = 0;
236                 end
237             case 'Auto'
238                 GO = 0;
239                 Matrix = get
                    (handles.Output_Panel_Add_Matrix,'Value');
240                 Fibre = get
                    (handles.Output_Panel_Add_Fibre,'Value');
241                 if strcmp(get(handles.Output_Panel_Add_Vol,
                    'string'),'')
242                     errordlg('Please enter Volume Fraction',
                        'INPUT ERROR: Volume','modal');
243                     CC = 0;
244                 else
245                     Vol = str2num(get
                        (handles.Output_Panel_Add_Vol,
                        'String'))/100;
246                     CC = 1;
247                 end
248                 if strcmp(get(handles.Output_Panel_Add_Thick,
                    'string'),'')
249                     errordlg('Please enter Layer Thickness',
                        'INPUT ERROR: Thickness','modal');
250                     CC = 0;
251                 else
252                     Thick = str2num(get
                        (handles.Output_Panel_Add_Thick,'String'));
253                     CC = 1;
254                 end
255                 Mat_Val = load('Database-Mat.mat');
256                 Prop_M = Mat_Val.Data(Matrix,:);
257                 Mat_Val = load('Database-Fib.mat');
258                 Prop_F = Mat_Val.Data(Fibre,:);
259                 if CC == 1
260                     Material.Data(pos,:) = MicroMech(Prop_M,
                        Prop_F,Vol,Thick);
261                     GO = 1;
262                 end
263         end
264 End

265 if GO == 1
266     save(handles.FileName,'-struct','Material',
        '-append','Name','Data');
267     setMaterialListString
        (handles.Material_List,eventdata,handles);
268     set(handles.Material_List,'value',1);
269     clear_add(hObject, eventdata, handles);
270     set(handles.Output_Table,'Data',
```

```
        zeros(1,length(Material.Att)));
271     msgbox('Operation Completed','Confirm','modal');
272     handles.Man_Aut = '0';
273     guidata(hObject,handles);
274 end

% --- Executes on button press in Output_Panel_Close.
275 function Output_Panel_Close_Callback(hObject, eventdata,
    handles)
% hObject    handle to Output_Panel_Close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
276 clear_panels(hObject, eventdata, handles);
277 clear_add(hObject, eventdata, handles);


%----------C L O S E   O R   E X I T   P R O G R A M----------%
% --- Executes when user attempts to close Figure_Database.
278 function Figure_Database_CloseRequestFcn(hObject, eventdata,
    handles)
% hObject    handle to Figure_Database (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: delete(hObject) closes the figure
279 if handles.HelpVal == 1
280     if ishandle(handles.HelpFig)
281         delete(handles.HelpFig);
282     end
283 end
284 mainHandles = guidata(handles.MainMenuVar);
285 set(mainHandles.Check,'String','GO');
286 uiresume(handles.Figure_Database)
287 delete(handles.Figure_Database);
```

In line 1, various inputs (*varargin*) may be accepted and multiple outputs (*varargout*) may be generated. The basic tasks of the GUI are initialised via lines 2 to 11. Any input and output variables are recognised here.

The code from line 12 to 22 is used to create variables, obtain default values and lock the GUI, prior to it becoming visible. In line 14, the default database from the "DATABASE" panel is acquired, while the code in line 15 is used to initialise a variable (*handles.Man_Aut*) that is used later for the Adding Properties layout. The function that creates the axes for the GUI's logo, found in lines 23 to 25, is executed in line 16. The material names from the matrix and fibre databases are loaded via lines 17 and 18, respectively. Other GUIs, from the main graphical interface, are prevented

from being executed by the code in lines 19 and 20 as well as by line 28. A variable (*handles.HelpVal*) that is used to determine whether the help facility was used is initialised in line 21.

The first function (**clear_panels**), in lines 29 to 39 of the main program, is used to hide the "DATABASE" panel (line 30), the "MATERIAL" panel (line 32), and the output panel (line 33). The default database on the "DATABASE" panel is set by line 31 to the database obtained earlier in line 14. The code in lines 34 to 39 is used to render certain buttons and tables on the output panel invisible, as well as clearing the tables and textboxes of data. The next function (**clear_add**), in lines 40 to 45 of the code, is used to clear all data from the tables and textboxes associated with the Adding Properties layout.

The functions for the "HELP" and "EXIT" buttons are contained in lines 46 to 51. The help facility is launched by the code in line 47. A value of '1' is assigned to the variable *handles.HelpVal* in line 48, and this indicates that the help interface has been opened. This information is required later when the GUI is closed. When the "EXIT" button is pressed, line 51 is run and the execution of the code is redirected to the function in line 278, which ensures that the GUI is properly closed.

As discussed in Chapter 7, the user is able to view, edit, add or delete properties or materials from any of the databases. This is achieved via the pushbuttons on the "OPERATIONS" panel. The choice of button/operation determines the layout of the output panel, which includes the title and colour of the panel, as well as which of the buttons, textboxes, tables and drop-down menus are made available. The general layout of the output panel contains two textboxes that display the names of the selected database and material, a table that shows the properties of the selected material, and two pushbuttons. One of these pushbuttons is labelled "CLOSE" and is used to hide the "DATABASE", "MATERIAL" and output panels when a particular operation has been completed. The other pushbutton performs different tasks depending on the operation that was chosen. The title of this pushbutton changes with each operation, and hence, for ease of reference, will be termed the Exec button.

The code from line 52 to 87 contains the functions for the “VIEW”, “EDIT”, “ADD” and “DELETE” pushbuttons. In each case, the functions **clear_panels** and **clear_add**, which were discussed earlier, are executed to ensure that all data has been cleared and that the GUI’s appearance is as shown in Figure 7.1. For each particular operation, the output panel’s background colour is set, the panel’s title is assigned, the label of the Exec button is allocated, and the “DATABASE” panel is made visible. Depending on the function being executed, a value of ‘View’, ‘Edit’, ‘Add’ or ‘Del is assigned to a variable called *handles.OP_Button*. This variable is used later to further change the layout of the output panel, and is also used to determine the operation of the Exec button.

The function from line 88 to 122 deals with the selection of a database on the “DATABASE” panel. There are three radio buttons on the panel, and the code in lines 89 to 97 is used to determine which of these buttons is selected, and to assign the appropriate database file to the variable *handles.FileName*. In line 99, a function (shown in lines 123 to 125) is executed in order to load the material names, from the database stipulated by the variable *handles.FileName*, to the list on the “MATERIAL” panel. The default selection is set to the first name on the list via line 100. The code in lines 101 and 102 are used to make the output panel visible, and set the “DATABASE” textbox, on the output panel, to the value selected on the “DATABASE” panel.

When using the Add operation, the “MATERIAL” panel is not required. Therefore, in line 103, the value of the variable *handles.OP_Button* is determined and, if this value is not ‘Add’, the “MATERIAL” panel is made visible (line 104) and the table on the output panel is turned on (line 105). The Edit, Add and Delete operations require the use of the Exec button. The code in lines 107 to 109 is used to activate this button for those operations.

If the value of the variable *handles.OP_Button* is ‘Add’ (line 110) and the ‘Composite’ database is selected on the “DATABASE” panel (line 111), then two additional pushbuttons, namely, “Manual” and “Auto”, are activated/made visible on the output panel (lines 112 to 113). Further, the table on the output panel is turned off (line 114).

This layout of the output panel is shown in Figure 7.9. Alternatively, if the selected database was either ‘Fibre’ or ‘Matrix’ (line 115), the “Manual” and “Auto” buttons are deactivated/turned off and the table is made visible (lines 116 to 119). The function in line 120 is used to make the columns of this table editable. The GUI layout for this case is shown in Figure 7.8.

The “MATERIAL” panel is required for the View, Edit and Delete operations. The function in lines 126 to 129 is used to set the display properties of the list on this panel. The next function (lines 130 to 149) deals with the selection of materials from this list. In line 131, the name of the selected material is obtained, and the code in line 132 is used to load the database that was assigned earlier to the variable *handles.FileName*.

The next line of code (line 133) is used to determine whether the value of the variable *handles.OP_Button* is either ‘Edit’ or ‘Add’. If this is the case, the “MATERIAL” textbox on the output panel is made editable by the code in line 134. Further, a variable *ColEdit* is assigned a value of ‘true’ for each editable column of the table on the output panel (line 135). Alternatively, if the value of the variable *handles.OP_Button* is either ‘View’ or ‘Del’ (line 136), then, in line 137, the property of the “MATERIAL” textbox is changed to be non-editable, and, in line 138, the variable *ColEdit* is assigned a value of ‘False’ for each column of the table.

If the value of the variable *handles.OP_Button* is either ‘View’, ‘Edit’ or ‘Del’ (line 140), then the name of the material selected on the “MATERIAL” panel is displayed in the “MATERIAL” textbox on the output panel using line 141. In addition, the properties of the selected material are shown in the table of the output panel by the code in line 142. Alternatively, if the value of the variable *handles.OP_Button* is ‘Add’ (line 143), the “MATERIAL” textbox is left blank (line 144) and the entries of the table are all set to zero (line 145). In lines 147 to 149, the properties of the table on the output panel are set. These include the width, names of the columns, and whether the columns are editable or not (determined by the variable *ColEdit*).

(The functions in lines 150 to 163 are discussed later.)

As mentioned earlier, when the value of the variable *handles.OP_Button* is 'Add' and the 'Composite' database is selected on the "DATABASE" panel, the "Manual" and "Auto" pushbuttons are made visible. Pressing of the "Manual" button runs the function from line 164 to 174. The **clear_add** function is executed in line 165 in order to clear all data from the table and textboxes on the output panel. The "MATERIAL" textbox on this panel is set to be editable (line 166), and the database stipulated by the variable *handles.FileName* is loaded (line 167). The code in lines 168 to 172 is used to make the table on the output panel visible, set the width and name of each column on the table, and make the columns editable. (The layout of the output panel for this case is shown in Figure 7.10.) A value of 'Manual-C' is assigned to the variable *handles.Man_Aut* in line 173. This value, which is to be used later, implies that the material properties were manually entered for inclusion in the composite database.

If the "Auto" button is pressed, the function from line 175 to 183 is executed. In these lines the **clear_add** function is executed (line 176), the table on the output panel is turned off (line 177), and the "MATERIAL" textbox on the output panel is set to be editable (line 178). The code in line 179 is used to make a panel, containing two drop-down lists and two textboxes, visible. The textboxes are used for the fibre volume ratio and layer thickness values, and the drop-down lists contain the names of the matrix and fibre materials from their respective databases. (This layout of the output panel is shown in Figure 7.11.) In line 180, the names of the matrix materials are obtained by executing the function in lines 154 to 156. In this function the material names are loaded from the matrix database. The function in lines 150 to 153 is used to set the properties of the matrix drop-down list. Similarly, in line 181, the function in lines 161 to 163 is used to load the names of the fibre materials from the fibre database. The properties of the fibre drop-down list are set by the function in lines 157 to 160. For this case, the variable *handles.Man_Aut* is assigned a value of 'Auto' (line 182) to signify that the composite material properties were determined by micromechanical methods.

Up to this point, most of the code discussed above dealt with the layout of the GUI and the preparation of the graphical components for data input. Once data has been entered

or chosen, the relevant database needs to be amended/updated. This task is performed by means of the Exec button. As mentioned earlier, this button has different labels for each layout of the output panel. In the case of the Editing Properties layout (Figure 7.6), the Exec button is labelled as "OK". It is labelled as "DELETE" in the Deleting Properties layout (Figure 7.7) and as "ADD" in the different Adding Properties layouts (Figures 7.8 to 7.11). The View operation does not require the use of this button and therefore it is omitted from the Viewing Properties layout (Figure 7.5).

The function that governs the use of the Exec button begins at line 184 and ends at line 274. In the function, the database stipulated by the variable *handles.FileName* is loaded using line 185, and the code in line 186 determines the location of the selected material in the list on the "MATERIAL" panel. As stated before, the role of the Exec button changes depending on the layout of the GUI. The value of the variable *handles.OP_Button* is used to ascertain which layout is currently being used (line 187).

If the value of the variable *handles.OP_Button* is 'Edit' (line 188), then lines 189 to 196 is executed. The "MATERIAL" textbox on the output panel is tested to determine if it contains text (line 189). If the textbox is empty, an error message is displayed (line 190) that prompts the user to enter a material name. On the other hand, if the textbox contains text (line 192), then the code in lines 193 and 194 are respectively used to extract the material name from the textbox, and the material properties from the table. In lines 191 and 195, a variable *GO* is used to indicate whether errors have been encountered. A value of zero is assigned when an error is found (as in line 191), and, if there are no errors, a value of '1' is assigned (as in line 195).

When the Exec button is pressed and the value of the variable *handles.OP_Button* is 'Del' (line 197), a message is displayed (line 198) that asks the user to confirm or reject the deletion of the material selected on the "MATERIAL" panel. If the delete action is confirmed (line 201), the chosen material's name and properties are set to null (lines 202 to 203), and a value of '1' is assigned to the variable *GO* (line 204). On the other hand, if the delete action is rejected (line 205), the variable *GO* is set to zero (line 206).

If the value of the variable *handles.OP_Button* is ‘Add’ (line 208), the code in lines 209 to 264 is executed. The Add operation requires that data be appended to the currently loaded database. Therefore it is important to determine the position of the next data entry in the database in order to prevent any existing data from being overwritten. The code in line 209 is used to achieve this.

The next step is to confirm which database is loaded. If the database is the matrix or fibre one (line 210), then the variable *handles.Man_Aut* is assigned a value of ‘Manual-FM’ (line 211), which indicates that the material properties were manually entered for inclusion in the matrix or fibre database. On the other hand, if the composite database is loaded but neither the “Manual” nor “Auto” buttons were pressed (line 213), an error message is displayed (line 214) that prompts the user to select either the “Manual” or “Auto” button. Further, the variable *GO* is set to zero (line 215). However, if the “Manual” or “Auto” button was pressed but the “MATERIAL” textbox on the output panel is empty (line 216), an error message is displayed (line 217) that prompts the user to enter a material name. Alternatively, if a material name was entered (line 218), it is stored in the current database (line 219) at the position determined in line 209.

Once the current database has been confirmed, the method chosen for the input of data has to be determined. The variable *handles.Man_Aut* is used for this. If the value of this variable is ‘Manual-FM’ (as in line 222), the code in lines 223 to 228 is executed. The “MATERIAL” textbox on the output panel is tested to determine whether it contains text (line 223), and if so, the data from the table on the output panel is stored in the current database (line 224) at the position determined earlier in line 209. The variable *GO* is assigned a value of ‘1’ in line 225. If, however, the textbox did not contain any text (line 226), then the variable *GO* is set to zero (line 227).

If the value of the variable *handles.Man_Aut* is ‘Manual-C’ (line 229), the procedure is similar to the previous case. The “MATERIAL” textbox is tested for text (line 230), and if it does contain text, then line 231 is used to store the data from the table on the output panel in the current database. The table is turned off (line 232) and a value of

‘1’ is assigned to the variable *GO* (line 233). If the textbox did not contain any text (line 234), then the variable GO is set to zero (line 235).

In the case where the value for the variable *handles.Man_Aut* is ‘Auto’ (line 237), the code in lines 238 to 264 is executed. The variable *GO* is assigned a value of ‘0’ (line 238). The location of the selected matrix and fibre materials from their respective drop-down lists, on the output panel, is evaluated by the code in lines 239 and 240, respectively.

The “Fibre Ratio” textbox, on the output panel, is tested to determine whether it is empty (line 241), and if it is, an error message is displayed (line 242) that prompts the user to enter a value. A variable *CC*, which has the same role as the variable *GO* but is used only for this case where the variable *handles.Man_Aut* has the value of ‘Auto’, is set to zero in line 243. If the “Fibre Ratio” textbox contains any text (line 244), then the data from the textbox is stored using the variable *Vol* (line 245), and the variable *CC* is assigned a value of ‘1’ (line 246).

The procedure is similar for the “Thickness” textbox on the output panel. If the textbox does not contain text (line 248), an error message is displayed (line 249) that prompts the user to enter a value, and the variable *CC* is set to zero (line 250). On the other hand, if the textbox contains text (line 251), the variable *Thick* is used to store the data from the “Thickness” textbox (line 252), and the variable *CC* is assigned a value of ‘1’ in line 253.

The properties of the chosen matrix material is loaded from its database using lines 255 and 256. Similarly, lines 257 and 258 are used to load the chosen fibre material from the fibre database. If, in lines 241 to 254, no errors were encountered or the errors were corrected, the value of the variable *CC* is ‘1’, and the function **MicroMech** is executed in line 260. This function, which is discussed later, is used to create a composite from the selected matrix and fibre materials via micromechanical methods. The value of the variable *GO* is set to ‘1’ (line 261).

At this point (line 265), if no errors were encountered or the errors were corrected, or, in other words, the variable *GO* has a value of '1', then the amended database may be saved to file using line 266. The code in lines 267 and 268 is used to update the list of materials on the "MATERIAL" panel. The **clear_add function** is executed (line 269), and a message is displayed (line 271) that informs the user that the current operation is completed. The *handles.Man_Aut* variable is reset to its initial value (line 272).

The different layouts of the output panel each contain a "CLOSE" pushbutton. When this button is used, the function in lines 275 to 277 is run. The **clear_panels** and **clear_add** functions are executed, which returns the GUI's appearance to that shown in Figure 7.1.

The last function of the GUI code (lines 278 to 287) is used to terminate the interface. If the help window is open, the code in lines 279 to 283 is used to close it. The current GUI is unlocked via lines 284 to 286, and the GUI is terminated by line 287.

Function: MicroMech

```
288 function Comp_Prop = MicroMech(Mat_Prop,Fib_Prop,Vol,Thick)
%Finds properties of composite material by Micro Mechanical Methods

%-----------ELASTIC CONSTANTS-----------%
%Bulk modulus for fibre Kf = Ef/(3*(1-2*nu_f))
289 Kf = Fib_Prop(1)/(3*(1-2*Fib_Prop(4)));
%Bulk modulus for matrix Km = Em/(3*(1-2*nu_m))
290 Km = Mat_Prop(1)/(3*(1-2*Mat_Prop(4)));
%E1 = Ef*Vf + Em*Vm + Factor
291 E1_Factor = (4*(Mat_Prop(4)-Fib_Prop(4))^2*Vol*(1-
    Vol))/(Vol/Km+(1-Vol)/Kf+1/Fib_Prop(3));
292 E1 = Fib_Prop(1)*Vol + Mat_Prop(1)*(1-Vol) + E1_Factor;
%1/E2 = Vf/Ef + Vm/Em
293 e2 = Vol/Fib_Prop(1) + (1-Vol)/Mat_Prop(1);
294 E2 = 1/e2;
%G12 = Gf*(Gf*Vf + Gm*(1+Vm))/(Gf*(1+Vm) + GmVf)
295 G12 = Fib_Prop(3)*(Fib_Prop(3)*Vol + Mat_Prop(3)*(2-
    Vol))/(Mat_Prop(3)*Vol + Fib_Prop(3)*(2-Vol));
%nu12 = nu_f*Vf + nu_m*Vm + Factor
296 nu12_Factor = ((Mat_Prop(4)-Fib_Prop(4))*(1/Kf-1/Km)*Vol*(1-
    Vol))/(Vol/Km+(1-Vol)/Kf+1/Fib_Prop(3));
297 nu12 = Fib_Prop(4)*Vol + Mat_Prop(4)*(1-Vol) + nu12_Factor;
%nu21 = nu12*E2/E1
298 nu21 = nu12*E2/E1;
```

```
%-----------ULTIMATE STRENGTHS-----------%
%sig1_ult_T = sig_ult_f*Vf + eps_ult_f*Em*Vm
299 eps_ult_f = Fib_Prop(6)/(Fib_Prop(1)*1000);
300 sig1_ult_T = Fib_Prop(6)*Vol + eps_ult_f*Mat_Prop(1)*1000*(1-
    Vol);

%sig1_ult_C = smallest value of 3 sets of equations
%Ultimate Tensile Strains in Matrix Failure Mode
301 eps_ult_mT = Mat_Prop(6)/(Mat_Prop(1)*1000);
302 ds = sqrt(4*Vol/pi);
303 eps2_ult_t(1) = eps_ult_mT * (1-Vol^(1/3));
304 eps2_ult_t(2) = eps_ult_mT * (ds*(Mat_Prop(1)/Fib_Prop(1) - 1)
    + 1);
305 eps2_ult_T = min(eps2_ult_t);
306 sig1_ult_c(1) = E1*1000*eps2_ult_T/nu12;
%Shear/Extensional Fibre Microbuckling Failure Mode
307 S1_c = 2*(Vol + (1-Vol)*Mat_Prop(1)/Fib_Prop(1))*...
    sqrt(Vol*Mat_Prop(1)*1000*Fib_Prop(1)*1000/(3*(1-Vol)));
308 S2_c = Mat_Prop(3)*1000/(1-Vol);
309 sig1_ult_c(2) = min(S1_c,S2_c);
%Shear Stress Failure of Fibres Mode
310 sig1_ult_c(3) = 2*(Fib_Prop(10)*Vol + Mat_Prop(10)*(1-Vol));
%Smallest Value
311 sig1_ult_C = min(sig1_ult_c);

%sig2_ult_T = E2*eps2_ult_T
312 sig2_ult_T = E2*1000*eps2_ult_t(2);

%sig2_ult_C = E2*eps2_ult_C
313 eps_ult_mC = Mat_Prop(7)/(Mat_Prop(1)*1000);
314 eps2_ult_C = eps_ult_mC * (ds*(Mat_Prop(1)/Fib_Prop(1) - 1) +
    1);
315 sig2_ult_C = E2*1000*eps2_ult_C;

%tau12_ult = G12*gam12_ult_m
316 gam12_ult_m = Mat_Prop(10)/(Mat_Prop(3)*1000);
317 tau12_ult = G12*1000*(ds*(Mat_Prop(3)/Fib_Prop(3) - 1) +
    1)*gam12_ult_m;


%-----------SPECIFIC GRAVITY-----------%
318 rho_c = Fib_Prop(13)*Vol + Mat_Prop(13)*(1-Vol);


%-----------THERMAL EXPANSION-----------%
%alpha1 = 1/E1*(alpha_f*Ef*Vf + alpha_m*Em*Vm)
319 alpha1 = (Fib_Prop(11)*Fib_Prop(1)*Vol +
    Mat_Prop(11)*Mat_Prop(1)*(1-Vol))/E1;
%alpha2 = (1+nu_f)*alpha_f*Vf + (1+nu_m)*alpha_m*Vm - alpha1*nu12
320 alpha2 = (1+Fib_Prop(4))*Fib_Prop(11)*Vol +
    (1+Mat_Prop(4))*Mat_Prop(11)*(1-Vol) - alpha1*nu12;
```

```
%-----------MOISTURE EXPANSION-----------%
%beta1 = (Em/E1)*(rho_c/rho_m)*beta_m
321 beta1 = (Mat_Prop(1)/E1) * (rho_c/Mat_Prop(13)) * Mat_Prop(12);
%beta2 = (1+nu_m)*(rho_c/rho_m)*beta_m - beta1*nu12
322 beta2 = (1+Mat_Prop(4))*(rho_c/Mat_Prop(13))*Mat_Prop(12) -
    beta1*nu12;


%-----------OUTPUT-----------%
323 Comp_Prop = [Vol, E1, E2, G12, nu12, nu21, sig1_ult_T,
    sig1_ult_C, sig2_ult_T, sig2_ult_C, tau12_ult, alpha1, alpha2,
    beta1, beta2, rho_c, Thick];
```

The **MicroMech** function has four inputs (line 288), namely, the matrix properties (*Mat_Prop*), fibre properties (*Fib_Prop*), fibre volume ratio (*Vol*), and the thickness of a layer (*Thick*). As mentioned earlier, the purpose of this function is to evaluate the properties of a composite material via micromechanical methods. These properties include five material constants, five material limits, specific gravity, and two coefficients each of thermal and moisture expansion. The equations required to calculate these properties were discussed in Chapter 3 and will be identified during the line-by-line analysis.

The five material constants are computed in lines 289 to 298. First the bulk modulus for the fibre and matrix, which are determined by equation (3.33), are calculated using lines 289 and 290, respectively. These moduli are then used in lines 291 and 292 to determine Young's modulus in direction 1 (E_1). These two lines of code may be associated with equation (3.28). Young's modulus in direction 2 (E_2) is evaluated via lines 293 and 294, which correspond to equation (3.29). The code in line 295 is based on equation (3.30) and is used to calculate the shear modulus (G_{12}). Poisson's ratio in the 1-2 plane (ν_{12}) is computed using lines 296 and 297 (equation (3.31)), and line 298 (equation (3.32)) finds Poisson's ratio in the 2-1 plane (ν_{21}).

The next set of properties to be determined are the five material limits. The first of these is the ultimate longitudinal tensile strength ($(\sigma_1^T)_{ult}$) and it is calculated by lines 299 and 300, which are based on equation (3.34). As discussed in Chapter 3, the ultimate longitudinal compressive strength ($(\sigma_1^C)_{ult}$) is the minimum of three values.

The first of these values is determined using lines 301 to 306, which represent equations (3.36) to (3.40). The code in lines 307 to 309 is used to compute the second value (equations (3.41) to (3.43)). The last value, which is calculated by equation (3.44), is obtained using line 310. The minimum value is found using line 311 which corresponds to equation (3.35).

The next material limit is the ultimate transverse tensile strength ($(\sigma_2^T)_{ult}$) and it is evaluated using line 312 (equation (3.45)). The code in lines 313 to 315, which represents equation (3.46), are used to compute the ultimate transverse compressive strength ($(\sigma_2^C)_{ult}$). The last material limit is the ultimate shear strength ($(\tau_{12})_{ult}$) and it is calculated using lines 316 to 317 which are based on equation (3.47).

The specific gravity is found using line 318, which is based on equation (3.26). The coefficients of thermal expansion are calculated by lines 319 and 320, and these are associated with equations (3.48) and (3.49), respectively. The code in lines 321 and 322 represent equations (3.50) and (3.51), respectively, and are used to find the coefficients of moisture expansion. The output of the **MicroMech** function is given by line 323.

APPENDIX I: MATLAB CODE FOR MAIN GRAPHICAL INTERFACE

This appendix contains the examination of the Matlab code for the Composite Design Assistant (main) GUI discussed in Chapter 7.

Graphical User Interface (GUI) code

```
1    function varargout = CompDesMainMenu(varargin)
% COMPDESMAINMENU M-file for CompDesMainMenu.fig
% COMPDESMAINMENU, by itself, creates a new COMPDESMAINMENU or
  raises the existing singleton*.
% H = COMPDESMAINMENU returns the handle to a new COMPDESMAINMENU or
  the handle to the existing singleton*.
% COMPDESMAINMENU('CALLBACK',hObject,eventData,handles,...) calls
  the local function named CALLBACK in COMPDESMAINMENU.M with the
  given input arguments.
% COMPDESMAINMENU('Property','Value',...) creates a new
  COMPDESMAINMENU or raises the existing singleton*.  Starting from
  the left, property value pairs are applied to the GUI before
  CompDesMainMenu_OpeningFcn gets called.  An unrecognized property
  name or invalid value makes property application stop.  All inputs
  are passed to CompDesMainMenu_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
  one instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help CompDesMainMenu

% Begin initialization code - DO NOT EDIT
2    gui_Singleton = 1;
3    gui_State = struct('gui_Name',       mfilename, ...
                        'gui_Singleton',  gui_Singleton, ...
                        'gui_OpeningFcn', @CompDesMainMenu_OpeningFcn, ...
                        'gui_OutputFcn',  @CompDesMainMenu_OutputFcn, ...
                        'gui_LayoutFcn',  [] , ...
                        'gui_Callback',   []);
4    if nargin && ischar(varargin{1})
5        gui_State.gui_Callback = str2func(varargin{1});
6    end
7    if nargout
8        [varargout{1:nargout}] = gui_mainfcn(gui_State,
    varargin{:});
9    else
10       gui_mainfcn(gui_State, varargin{:});
11   end
% End initialization code - DO NOT EDIT

% --- Executes just before CompDesMainMenu is made visible.
12   function CompDesMainMenu_OpeningFcn(hObject, eventdata,
     handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to CompDesMainMenu (see
  VARARGIN)

% Choose default command line output for CompDesMainMenu
13  handles.output = hObject;
14  GUI_Splash;
% Update handles structure
15  guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
16  function varargout = CompDesMainMenu_OutputFcn(hObject,
    eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
17  varargout{1} = [];


%---------P R O G R A M M I N G   S T A R T S   H E R E----------%
18  set(handles.Check,'String','GO');
19  uiwait(handles.MainMenu);

%-----Close Request Function - executes when Help is deleted-----%
20  function my_closereq(src, eventdata, hObject, handles)
21  delete(handles.HelpFig);
22  handles.HelpVal = 0;
23  guidata(hObject,handles);

%----------Closes HELP window when required----------%
24  function Close_Help(hObject, eventdata, handles)
25  if isfield(handles,'HelpFig')
26      if handles.HelpVal == 1
27          delete(handles.HelpFig);
28          handles.HelpVal = 0;
29          guidata(hObject,handles);
30      end
31  end


%--------------E X E C U T A B L E   B U T T O N S--------------%
% --- Executes on button press in Conv_Meth.
32  function Conv_Meth_Callback(hObject, eventdata, handles)
% hObject    handle to Conv_Meth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
33  if strcmp(get(handles.Check,'String'),'GO')
34      Close_Help(handles.MainMenu, eventdata, handles);
```

```
35      set(handles.Check,'String','WAIT');
36      GUI_Conventional('MainMenu',handles.MainMenu);
37  else
38      warndlg('Please EXIT current operation first',
    'ATTENTION','modal');
39  end

% --- Executes on button press in FibAng_Meth.
40  function FibAng_Meth_Callback(hObject, eventdata, handles)
% hObject    handle to FibAng_Meth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
41  if strcmp(get(handles.Check,'String'),'GO')
42      Close_Help(handles.MainMenu, eventdata, handles);
43      set(handles.Check,'String','WAIT');
44      GUI_Fibre_Angle_Output('MainMenu',handles.MainMenu);
45  else
46      warndlg('Please EXIT current operation first',
    'ATTENTION','modal');
47  end

% --- Executes on button press in DB_Mang.
48  function DB_Mang_Callback(hObject, eventdata, handles)
% hObject    handle to DB_Mang (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
49  if strcmp(get(handles.Check,'String'),'GO')
50      Close_Help(handles.MainMenu, eventdata, handles);
51      set(handles.Check,'String','WAIT');
52      GUI_Database('MainMenu',handles.MainMenu);
53  else
54      warndlg('Please EXIT current operation first',
    'ATTENTION','modal');
55  end

% --- Executes on button press in Robot.
56  function Robot_Callback(hObject, eventdata, handles)
% hObject    handle to Robot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
57  if strcmp(get(handles.Check,'String'),'GO')
58      Close_Help(handles.MainMenu, eventdata, handles);
59      set(handles.Check,'String','WAIT');
60      GUI_Robot('MainMenu',handles.MainMenu);
61  else
62      warndlg('Please EXIT current operation first',
    'ATTENTION','modal');
63  end

% --- Executes on button press in Help.
64  function Help_Callback(hObject, eventdata, handles)
% hObject    handle to Help (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
65   if strcmp(get(handles.Check,'String'),'GO')
66       handles.HelpFig = Help_MainMenu;
67       handles.HelpVal = 1;
68       set(handles.HelpFig,'CloseRequestFcn',
    {@my_closereq,hObject,handles});
69       guidata(hObject,handles);
70   else
71       warndlg('Please EXIT current operation first',
    'ATTENTION','modal');
72   end

% --- Executes on button press in Exit.
73   function Exit_Callback(hObject, eventdata, handles)
% hObject    handle to Exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
74   MainMenu_CloseRequestFcn(handles.MainMenu, eventdata, handles);


%--------------C L O S I N G   W I N D O W---------------%
% --- Executes when user attempts to close MainMenu.
75   function MainMenu_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to MainMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
76   Close_Help(handles.MainMenu, eventdata, handles);
77   if strcmp(get(handles.Check,'String'),'GO')
78       uiresume(handles.MainMenu);
79   else
80       warndlg('Please EXIT current operation first',
    'ATTENTION','modal');
81   end
82   if ~isequal(get(hObject,'waitstatus'),'waiting')
83       delete(handles.MainMenu);
84   end
```

As discussed in the previous appendices, the code in line 1 is used to accept multiple inputs (*varargin*) and generate various outputs (*varargout*). These input and output variables are recognised in lines 2 to 11. The code in line 14 is used to execute a splash screen, which is shown in Figure I.1. Any outputs from this GUI to the command line is returned using the function in lines 16 and 17.

The GUIs examined in the previous appendices contained code that was used to prevent the execution of other GUIs from the main GUI. This is achieved by means of

a hidden textbox, on the main GUI, which may contain the value of either ‘GO’ or ‘WAIT’. The value of ‘GO’ allows any pushbutton on the main GUI to be used, and the value of ‘WAIT’ stops the execution of any GUI. The default value (‘GO’) of the hidden textbox is set by line 18, and the main GUI is prevented from being closed incorrectly or prematurely by the code in line 19.

Figure I.1: Splash screen for the Composite Design Assistant

As with the other GUIs, the main GUI also has a help facility (discussed later). The function in line 20 to 23 is executed when the help window is closed by the user. Here again, the variable *HelpVal* is used to determine if the help facility has been opened. When any of the other GUIs are run, the function in lines 24 to 31 is used to automatically close the help window (if open) in order to avoid conflict with the GUI being executed.

The Composite Design Assistant GUI has six pushbuttons as shown in Figure 7.12. The first of these is labelled “CONSTANT STIFFNESS METHOD” and is used to execute the GUI for the constant stiffness design method (CSM code). In line 33, a test is performed to determine whether the value in the hidden textbox is ‘GO’. If this is the case, the help window is closed (line 34), the value of the hidden textbox is set to ‘WAIT’ (line 35), and the GUI for the CSM code is launched (line 36). Further, in line 36, the value of the hidden textbox, which at this point is ‘WAIT’, is passed to the CSM GUI, where the value would be changed to ‘GO’ when the CSM GUI is properly terminated. If the test in line 33 revealed that the value of the hidden textbox is ‘WAIT’, a message is displayed (line 38) informing the user that another GUI is in operation.

The code for the “VARIABLE STIFFNESS METHOD” pushbutton (lines 40 to 47) is similar to that discussed above, with the only difference being in line 44, where the GUI for the variable stiffness design method (VSM code) is executed instead. Likewise, the code for the “DATABASE MANAGEMENT” (lines 48 to 55) and “ROBOTIC ARM CODING” (lines 56 to 63) pushbuttons differ in lines 52 and 60, respectively, where the relevant GUI is executed.

The code for the “HELP” pushbutton is contained in lines 64 to 72. The help window is opened only if the value of the hidden textbox is ‘GO’ (lines 65 to 69). If this is not the case, a message is displayed informing the user to exit the current operation (lines 70 to 72).

The last pushbutton of the main GUI is labelled “EXIT” and its function (lines 73 and 74) is redirected to line 75. Here, the function that correctly terminates the GUI is executed. In line 76, the help facility is closed (if open). If the value of the hidden textbox is ‘GO’ (line 77), the main GUI is unlocked (line 78), and the GUI is closed (lines 82 to 84). On the other hand, if the value of the hidden textbox is ‘WAIT’ (line 79), a message is displayed informing the user that another operation is running.