DEVELOPMENT OF AN ARTIFICIAL NEURAL NETWORK BASED ANALYSIS METHOD FOR SKIN-STRINGER STRUCTURES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ANIL CANKUR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
AEROSPACE ENGINEERING

SEPTEMBER 2017

Approval of the thesis:

DEVELOPMENT OF AN ARTIFICIAL NEURAL NETWORK BASED ANALYSIS METHOD FOR SKIN-STRINGER STRUCTURES

submitted by **ANIL CANKUR** in partial fulfillment of the requirements for the degree of **Master of Science in Aerospace Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Director, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Ozan Tekinalp
Head of Department, **Aerospace Engineering** _____

Assoc. Prof. Dr. Ercan Gürses
Supervisor, **Aerospace Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Altan Kayran
Aerospace Engineering Department, METU _____

Assoc. Prof. Dr. Ercan Gürses
Aerospace Engineering Department, METU _____

Assoc. Prof. Dr. Serdar Göktepe
Civil Engineering Department, METU _____

Assoc. Prof. Dr. Demirkan Çöker
Aerospace Engineering Department, METU _____

Assist. Prof. Dr. Barış Sabuncuoğlu
Mechatronics Engineering Department, UTAA _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ANIL CANKUR

Signature :

ABSTRACT

DEVELOPMENT OF AN ARTIFICIAL NEURAL NETWORK BASED ANALYSIS METHOD FOR SKIN-STRINGER STRUCTURES

Cankur, Anıl
M.S., Department of Aerospace Engineering
Supervisor : Assoc. Prof. Dr. Ercan Gürses

September 2017, 126 pages

The purpose of this work is to develop a tool for the design of skin-stringer structures. The tool to be developed needs to quickly identify the load carrying capacity and weight of these structures. For this purpose, finite element (FE) models of 1440 different skin-stringer structures were created with a script written in Python 2.7, and these models were analyzed using the commercial FE program ABAQUS. The script was used to construct the model, analyze the model, calculate the buckling load, the collapse load and the total mass, and collect the ABAQUS analysis results. The input parameters and analysis results of these 1440 model were used to create an artificial neural network (ANN) in the MATLAB NNTOOL toolbox for fast determination of the buckling load, the collapse load, and the weight of the skin-stringer assemblies that have not yet been created. The performance of the generated ANN was shown in comparison with FE results.

Keywords: Stiffened Panels, Buckling, Post-Buckling, Finite Element Analysis, Artificial Neural Network

ÖZ

KİRİŞLE GÜÇLENDİRİLMİŞ KABUK YAPILAR İÇİN YAPAY SİNİR AĞ BAZLI ANALİZ YÖNTEMİ GELİŞTİRİLMESİ

Cankur, Anıl
Yüksek Lisans, Havacılık ve Uzay Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Ercan Gürses

Eylül 2017 , 126 sayfa

Bu çalışmanın amacı, kirişle güçlendirilmiş kabuk yapıların tasarımına yönelik bir araç geliştirmektir. Geliştirilecek aracın bu yapıların yük taşıma kapasitesini ve ağırlığını hızlı şekilde belirlemesi gerekmektedir. Bu amaçla, Python 2.7 programı kullanılarak yazılmış bir betikle 1440 farklı güçlendirilmiş kabuk yapının sonlu elemanlar (SE) modeli oluşturulmuş ve ticari SE programı olan ABAQUS kullanılarak bu modeller analiz edilmiştir. Yazılan betik, modelin oluşturulması, modelin analizi, burkulma yükünün, çökme yükünün ve toplam ağırlığın elde edilmesi ve ABAQUS analiz sonuçlarının derlenmesi için kullanılmıştır. Bu 1440 modelin girdi parametreleri ve analiz sonuçları, hâlihazırda yaratılmamış olan güçlendirilmiş kabuk yapıların burkulma yükünü, çökme yükünü ve ağırlığını hızlı bir şekilde belirlemek için MATLAB NNTOOL araç kutusunda bir yapay sinir ağı (YSA) oluşturulmasında kullanılmıştır. Oluşturulan YSA'nın performansı SE sonuçlarıyla yapılan karşılaştırmalarla gösterilmiştir.

Anahtar Kelimeler: Güçlendirilmiş Kabuk Yapıları, Burkulma, Burkulma Sonrası, Sonlu Elemanlar Analizi, Yapay Sinir Ağı

To my family and friends

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Ercan Gürses for his advice, guidance, and reccomendations throughout the study.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Dr. Altan Kayran, Assoc. Prof. Dr. Serdar Göktepe, Assoc. Prof. Dr. Demirkan Çöker, and Assist. Prof. Dr. Barış Sabuncuoğlu, for their insightful comments and contributions.

I would like to thank my friends and colleagues for their motivation and support.

Finally, I must express my very profound gratitude to my parents Recep and Hanife Cankur, to my brother Onur Cankur and to my partner İrem Aksoy for providing me with unfailing support and continuous encouragement throughout my study and through the process of this research and writing. This accomplishment would not have been possible without them.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xii

LIST OF FIGURES xiii

LIST OF ABBREVIATIONS xvii

CHAPTERS

1 INTRODUCTION 1

2 LITERATURE REVIEW 5

3 BUCKLING AND POST-BUCKLING ANALYSIS 11

3.1 Buckling Analysis of Unstiffened Panels 12

3.1.1 Finite Element Model 18

3.1.2 Loading and Boundary Conditions 20

3.1.3 Finite Element Results 22

3.1.4 Comparison with Conventional Methods 25

3.2 Buckling and Post-Buckling Analysis of Skin-Stringer Assemblies 28

3.2.1 Geometry 29

3.2.2 Material Properties 32

3.2.3 Mesh Quality 34

3.2.4 Finite Element Solution Schemes 37

3.2.5 Finite Element Analysis Results 41

3.2.6 Effective Skin Width in Post-Buckling Region . . . 42

3.2.7 Finite Element Analysis Verification 48

4 ARTIFICIAL NEURAL NETWORK 51

4.1 Theory of Artificial Neural Network 52

4.2 Generation of Artificial Neural Network 54

4.3 Results of Artificial Neural Network 58

5 RESULTS AND EVALUATION 61

5.1 Comparison with Finite Element Analysis 62

5.2 Design Curves 64

6 CONCLUSION AND FUTURE WORKS 67

6.1 Conclusion 67

6.2 Future Works 68

REFERENCES 69

APPENDICES

A MATERIAL PROPERTIES 75

B COMPARISON OF SOLUTION METHODS 79

B.1 The Newton's Method 83

B.2 The Arc Length (Riks) Method 86

C PYTHON SCRIPTS 91

D MATLAB SCRIPTS 123

LIST OF TABLES

TABLES

Table 3.1 Eigenvalue results for different element sizes 19

Table 3.2 Translation and rotation restrictions for simply supported and clamped plates 21

Table 3.3 Collapse load results for different number of fasteners 30

Table 3.4 Collapse load results for different element sizes 34

Table 3.5 Buckling load results for different element sizes 35

Table 3.6 Structural properties obtained from the finite element analysis 41

Table 4.1 Geometric parameters that are fixed in all models 56

Table 4.2 Input parameters for the data set 56

Table 5.1 Deviations of the ANN results from the FEA results 63

Table 5.2 Input parameters for the additional analyses 63

Table 5.3 Results of the additional analyses 64

LIST OF FIGURES

FIGURES

Figure 1.1 Basic components of an airframe (adapted from Niu [2]) 2

Figure 1.2 Helicopter fuselage skin-stringers (left) and tail skin-stringers (right) 2

Figure 3.1 A buckled fuselage skin [33] 13

Figure 3.2 Panel geometric parameters 14

Figure 3.3 Compressive buckling factor with respect to panel aspect ratio a/b [8] (C:Clamped, SS: Simply supported, F: Free) 15

Figure 3.4 Shear buckling factor with respect to panel aspect ratio a/b [8] . . . 16

Figure 3.5 Finite element model of the panel 18

Figure 3.6 The first eigenvalues with respect to number of elements 19

Figure 3.7 Compressive loading of the panel 21

Figure 3.8 Panel subjected to shear loads 22

Figure 3.9 The first mode of the simply supported panel loaded in compression 23

Figure 3.10 The first mode of the simply supported panel loaded in shear 23

Figure 3.11 The first mode of the clamped panel loaded in compression 24

Figure 3.12 The first mode of the clamped panel loaded in shear 24

Figure 3.13 Comparison of compressive buckling factor for plates with simply supported edges. (Compared with continuous curve C of Figure 3.3) . . . 26

Figure 3.14 Comparison of shear buckling factor for plates with simply supported edges. (Compared with hinged edge curve of Figure 3.4) 26

Figure 3.15 Comparison of compressive buckling factor for plates with clamped edges. (Compared with dashed curve A of Figure 3.3) 27

Figure 3.16 Comparison of shear buckling factor for plates with clamped edges. (Compared with clamped edge curve of Figure 3.4) 27

Figure 3.17 Drawing of the skin-stringer assembly 28

Figure 3.18 Drawing of the stringer cross section 28

Figure 3.19 Z type stringer cross section 29

Figure 3.20 Fastener connections 30

Figure 3.21 Typical body structure [40] 31

Figure 3.22 Finite element model of the skin-stringer assembly 31

Figure 3.23 Stress-strain curves suggested by MMPDS [36] and obtained from Ramberg-Osgood Equation 33

Figure 3.24 Buckling patterns of the models with element sizes of 2 mm and 10 mm 35

Figure 3.25 Force-displacement curve of the models with different element sizes 36

Figure 3.26 Meshed skin-stringer assembly model 36

Figure 3.27 Enlarged view of the meshed skin-stringer assembly model 37

Figure 3.28 First four buckling mode shapes 38

Figure 3.29 Riks solver results with different initial imperfections 38

Figure 3.30 Disturbances after the buckling 39

Figure 3.31 Comparison of Static, General solver and Riks solver 40

Figure 3.32 Force-displacement curve of the skin-stringer assembly 42

Figure 3.33 Illustration of load distribution of the skin-stringer assembly after buckling (adapted from Collier [15]) 43

Figure 3.34 Distribution of loads on the skin-stringer assembly before the buckling with an applied load of 15.3 kN 44

Figure 3.35 Distribution of loads on the skin-stringer assembly after the buckling with an applied load of 38.7 kN 44

Figure 3.36 Distribution of loads on the skin-stringer assembly close to the collapse with an applied load of 111.9 kN 45

Figure 3.37 Distribution of loads on the skin-stringer assembly after the collapse with an applied load of 110.8 kN 45

Figure 3.38 Out-of-plane deformations a) before buckling b) after buckling c) close to collapse d) after collapse 46

Figure 3.39 Four stages of the assembly a) before buckling b) after buckling c) close to collapse d) after collapse 46

Figure 3.40 Total load, skin load, and stringer load during the analysis 47

Figure 3.41 Skin load and stringer load during the analysis as percentages of total load 47

Figure 3.42 Geometry of the test structure [22] 48

Figure 3.43 Force-displacement curve of the test structure analysis 49

Figure 4.1 A biological neuron and an artificial neuron (adapted from Gurney [48]) 52

Figure 4.2 Artificial neural network 52

Figure 4.3 Artificial neural network with 3:2:1 configuration 53

Figure 4.4 Function of an artificial neuron 54

Figure 4.5 The final configuration of the ANN 57

Figure 4.6 Performance plot of the ANN 59

Figure 4.7 Regression plots of the ANN for overall, training, validation and testing samples 60

Figure 4.8 Number of validation fails with respect to number of training iterations 60

Figure 5.1 The collapse load and the buckling load with respect to skin thickness 65

Figure 5.2 The collapse load and the buckling load with respect to stringer thickness 65

Figure A.1 Properties of 2024 T3 clad aluminum sheet [36] 75

Figure A.2 Properties of 2014 T6 aluminum sheet [36] 76

Figure A.3 Properties of 2024 T42 clad aluminum sheet [36] 77

Figure B.1 A body before and after deformation [53] 79
Figure B.2 Load - displacement curve of an unstable system [54] 82
Figure B.3 Representation of Newton's method [53] 84
Figure B.4 Drawback of Newton's method [53] 85
Figure B.5 Representation of arc length method [53] 87
Figure B.6 Representation of an incorrect solution direction [34] 89

LIST OF ABBREVIATIONS

σ_{cr}	Buckling stress
k_c	Compressive buckling factor
k_s	Shear buckling factor
b	Loaded edge length of the panel
E	Young's modulus
E_c	Compressive Young's modulus
v	Poisson's ratio
ρ	Density
t	Panel thickness
Q^N	Incremental load
K_0^{MN}	The stiffness matrix of the base state
K_Δ^{MN}	The stiffness matrix induced from the incremental load
λ	Eigenvalues
v^M	Eigenvectors
M , N	Degrees of freedom
N_{cr}	Buckling load
$N_{applied}$	Applied shell edge load
s	Fastener spacing
d	Fastener diameter
ϵ	Strain
σ	Stress
n	Ramberg-Osgood number
σ_0	Yield strength
σ_{tr}	True stress
σ_{eng}	Engineering stress
ϵtr	True strain
ϵ_{eng}	Engineering strain
σ_{ult}	Ultimate strength

CHAPTER 1

INTRODUCTION

Most airframe structures generally consist of thin skins and stiffeners. The basic components that crate an airframe can be listed with their primary functions as [1]

- Skin : Outer surface of an aircraft. The skin resists the aerodynamic forces on the aircraft.
- Stringer : A longitudinal beam structure that reinforces the skin. The main function of the stinger is to withstand the compression force acting on the airframe.
- Frame (or rib) : A transverse beam structure that reinforces the skin. The frame is a larger member than the stringer, and it gives the airframe its shape by its large geometry and high strength.

The components above are shown in Figure 1.1 which is adapted from Niu [2]. The basic components together form substructures such as skin-stringer assemblies shown in Figure 1.2. In the figure, there are images of fuselage skin-stringers (on the left) and tail boom skin-stringers (on the right) of a helicopter. Strengths of individual substructures determine the strength of an aircraft. The aircraft must be strong enough to ensure the safety of its passengers and payload. However, a design with too high strength may end up with extreme weights [3]. Therefore, design of the substructures requires an optimization of strength and weight.

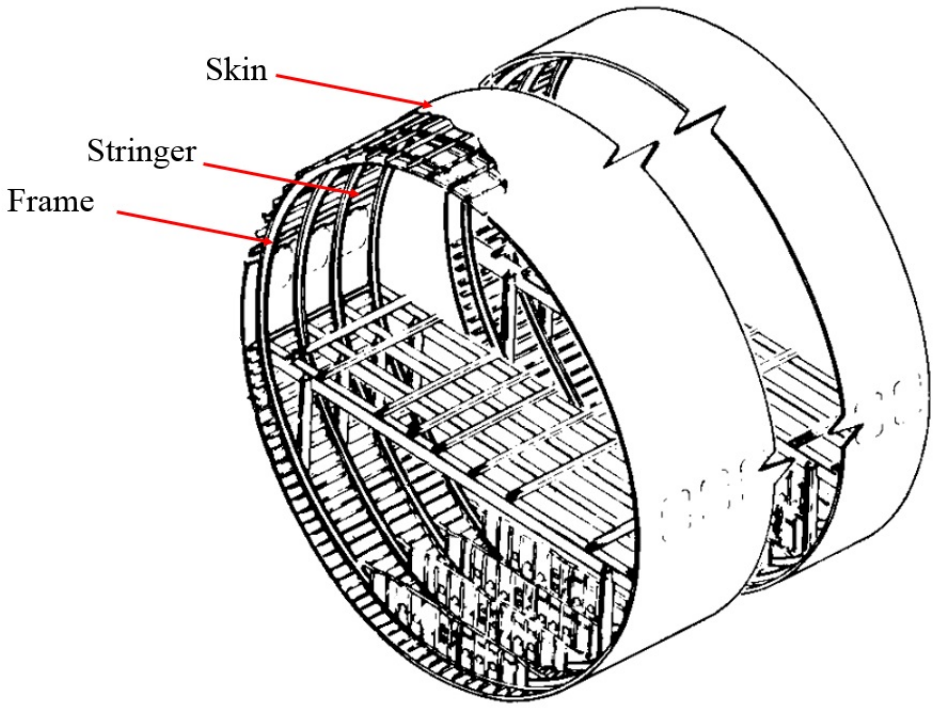


Figure 1.1: Basic components of an airframe (adapted from Niu [2])



Figure 1.2: Helicopter fuselage skin-stringers (left) and tail skin-stringers (right)

A design with optimum weight and strength can be obtained with a process that includes many trials. After each design, the weight and the strength of the aircraft are calculated. If the structure is too strong or not strong enough, then the design is changed with a new one. The strength can be calculated using different techniques. However, each technique aims to predict failure of the structure accurately. The prediction can be done by a structural testing of the structure or creating simulations that imitate the structure and its working conditions. Testing of the structure is usually a costly process. Therefore, many mathematical models that provide results based on test results of the similar structures are used in literature. However, these

models come with limitations and possible errors. To reduce the errors and limitations of the mathematical models, simulations based on finite element method (FEM) are most commonly preferred.

Finite element method requires a modeling process that includes creation of the structure geometry, imitation of the load and boundary conditions, configuration of the analysis parameters, and calculation of the desired outputs. These processes can consume a considerable amount of time depending of the complexity of the structure. Details of the models, such as geometric details, material properties, types of boundary conditions, can be quite complex. Also, nonlinearities in the model geometry and material properties can increase the calculation time significantly. In addition, many repetitions of finite element analyses and update of models during the optimization process can increase the computational time, further.

In this study, it is aimed to decrease the time of the optimization process of skin-stringer assemblies by creating a tool for structural analysis that reduces the time cost of the FEM. The structural analysis tool created for this purpose is based on artificial neural network (ANN) that predicts the load carrying capacity of a skin-stringer assembly considering the finite element analysis (FEA) results of similar structures. ANN is a computation tool that can classify and recognize patterns, and provide accurate prediction for the given inputs by learning from the past [4]. The tool in this study is created by analyzing 1440 different skin-stringer assemblies via FEA and training an ANN with their input parameters and corresponding analysis results. Then, the ANN is used to predict the load carrying capacity and the mass of skin-stringer assemblies other than already created 1440 different models.

FEM is a useful tool to obtain realistic predictions of failure. However, it can provide inaccurate and misleading results if the finite element model (FEM) does not reflect the reality. In order to use the FEM correctly and efficiently, a FEA procedure is proposed. After each analysis, the load carrying capacity of the structure is extracted from the results. The load carrying capacity is defined by two parameters in this study, which are the load that the skin of the skin-stringer assembly is buckled and the load that the assembly fails to resist. The reason for this approach is that the skin

buckling significantly affects the load carrying capacity of the structure. When the skin is buckled, it looses its ability to resist further loading and it transfers the excess loads to stringers. The load that the buckling occurs is referred as *the buckling load*. After the excess load is transferred to stringers, the stringers carry additional loads. If the external load is increased further, then after some time, the stringers also cannot resist more load and the structure collapses. The load that fails the assembly is called as *the collapse load*. The calculation of the load carrying capacity is explained in Chapter 3.

To create the ANN, 1440 skin-stringer FE analyses are conducted with different geometry and material parameters using a commercial FE program ABAQUS. After each analysis, the buckling load, the collapse load and the mass of the structure are extracted from the FEA results. Using the input parameters and the results of the 1440 FEA, a database is created and used in training of an ANN, which is created in MATLAB NNTOOL toolbox. Generation and training of the ANN are described in Chapter 4. In this chapter, theory of ANN is explained together with the generation and training processes. Also, the performance of the created ANN is examined. In Chapter 5, the results obtained by FEA and ANN tool are compared to verify the ANN tool. Conclusions and possible future works are given in Chapter 6.

CHAPTER 2

LITERATURE REVIEW

Shell structures are considered to be a very efficient way to transmit loads in structures especially due to their lightness. Therefore, they are used in many areas, particularly in aerospace industry. Traditional analytical methods are widely used in industry and are compiled as handbooks [5, 6, 7]. These methods are usually based on NACA methods that are created in 1957 [8]. A study of Edlund [9] about history of the shell structures and their use reveals that the buckling load prediction of the shell structures is a challenging problem since 1920s. The challenge mentioned in this study is the inconsistency between the ideal strength and the real strength of a shell structure. The study explains the sources of the inconsistencies as imperfections on the shell, elasto-plastic behavior of the structure, and energy sensitivity. Also, it has been seen that buckling modes that are close to each other can interact and reduce the real buckling load.

The buckling phenomenon was studied by Alinia et al. [10]. In the study, six stages of an unstiffened panel from the start of the loading to the failure of the panel were studied. It has been seen that the buckling occurs when a critical load is reached. The load was found to depend on elastic material properties and geometry of the panel. The study also examined the post-buckling behavior of thin plates. Load-displacement curves of thin plates until failure were examined and the effects of geometric parameters on post-buckling were discussed.

Parameters that effect buckling and post-buckling properties of shells have been studied in several works [9, 10, 11]. Choong and Ramm [12] studied these parameters by using numerical simulations. Initial conditions that affect the buckling

were examined and the structural behavior of the structure under buckling and post-buckling was studied. Numerical simulations and test results were compared to prove that the simulations are applicable to obtain structural properties of a shell structure. Another comparison for the buckling properties was done by Ferreira and Virtuoso [13]. In that study, two semi-analytical methods were developed and used to conduct a nonlinear stability analysis of shell plates. It was seen that the boundary conditions have major effects on structural behavior. In order to observe the changes in structural behavior due to boundary conditions, Muameleci [11] conducted a study that compares simply-supported and clamped edge conditions by using two different commercial finite element programs ABAQUS and NASTRAN. The study showed that geometric properties and boundary conditions have significant effects on buckling properties. The analyses done in the study were verified with test results and compared with traditional analysis methods. In a study conducted by Amani et al. [14], the effects of nonlinear material properties, geometric properties, and initial imperfections on buckling and post-buckling behavior were studied. Also, a comparison of traditional methods and the Finite Element Method (FEM) was performed for elastic buckling.

Aircraft structures are allowed to buckle for some load cases since the structure can still resist the flight loads without undergoing significant plastic deformation. Hence, after buckling, the load distribution of the structure changes. Collier [15] introduced an analytical method to determine ultimate load carrying capacity of skin-stringer assemblies. The method is based on redistribution of the loads after buckling. To verify the method, nonlinear finite element analyses (FEA) were conducted with ABAQUS finite element software. Another study based on load distribution was conducted by Aswini et al.[16]. An analytical method was developed for stiffened panels working under combined compression and shear loads. The study was focused on the stiffness loss of the skin panels due to buckling. In both studies, it has seen that additional loads are induced on the stiffening members after buckling. The collapse load of the structure was determined considering the effect of the additional loads. Zhang and Khan [17] proposed a new analytical method for nonlinear analysis of stiffened panels. The collapse patterns of the stiffened panels were observed. The results revealed that slenderness ratio of the stiffener can determine

the collapse pattern. The described method was verified by FEM and tests of 58 specimens. Another nonlinear analysis method was introduced by Stamatelos et al. [18]. In the study, the buckling and the post-buckling analyses of stiffened panels were conducted with an analytical method. The method was applied on assemblies with isotropic and orthotropic materials. The study stressed the importance of the boundary conditions on buckling load. The studies explained up to this point are mainly interested in parameters that affect the buckling and post-buckling stresses. Also, analytic and empirical methods were introduced and verified with test results or FEM.

In order to verify a method with FEM, one has to use the FEM with a great care.In order to obtain realistic results by FEA, many studies are focused on the finite element modeling of stiffened panels. A study conducted by Lynch et al. [19] can be an example to these studies. In the study, guidelines to determine the ultimate load carrying capacity of skin-stringer assemblies were proposed. The important aspects of finite element modeling of stiffened panels working under compression loads were specified. The guidelines described in the study were the element choice, the mesh quality, the skin-stringer connection idealization, the initial conditions, and the solution procedures. In a similar study by Murphy et al. [20], guidelines are created in a similar way for the skin-stringer assemblies working under shear loads. In the study, an additional guideline for material models in nonlinear finite element modeling was proposed . With a similar consideration, possible error sources were examined by Campbell et al. [21]. The study focused on idealization errors developed during FEA and controlling these errors by observing the effects of model parameters. The results revealed the modeling of boundary conditions is a critical parameter for the determination of the buckling load. Also, ultimate strength calculation of a stiffened panel is affected by the material model and idealization of the geometry of stiffeners. The finite element model creation process was studied by Wang [22] as well. Nonlinear finite element analyses were conducted with different solution procedures. Parameters that affect the buckling and post-buckling behavior were studied. These parameters were found to be the material model, the geometric properties, and the boundary conditions of the structure. The results were compared with test results to verify the FEA.

Design of an airframe requires many trials until an optimum design is achieved. Therefore, the time consumed for the design can be minimized by decreasing the number of the trials. To decrease this number, several methods have been developed and used for stiffened panels. For example, Caseiro et al. [23] proposed optimization algorithms to reach the optimum design faster. The optimization procedures suggested in the study were tested on sample designs. The results revealed that it is possible to create an effective analysis method that is based on an optimization algorithm. The optimum level of the design was determined by user defined criteria. Buckling analyses of the stiffened panels were performed using ABAQUS and outputs of the analyses were examined. If the optimum design is reached, the algorithm stops. If the design is not at the optimum level, the model parameters are changed and the new model is analyzed. A similar study was conducted on stiffened composite panels by Bisagni and Lanzi [24]. The study aimed to decrease analysis time by introducing an artificial neural network (ANN) tool that is created for weight reduction of a stiffened panel. The tool provides an optimum load carrying capacity and weight for a skin-stringer assembly. These studies show that it is possible to decrease design time of airframe structures by using optimization algorithms.

Optimum design of structures can be achieved by different optimization techniques. A possible method to use in optimization process is the ANN. In literature, different kinds of structures have been optimized using ANN. A study of optimization of a compression member conducted by Sheidaii and Bahraminejad [25] is an example of use of ANN in optimization. In the study, load-displacement relation of different types of columns were obtained using analytical methods. The results were utilized to form a data set to train an ANN. A study with a similar purpose was conducted by Yıldırım [26]. Yıldırım created a bolted flange design tool based on ANN. A data set was created with finite element model parameters and corresponding analysis results. The data set was used in training of ANN. The ANN results were compared with FEA and analytical methods. Comparisons revealed that the ANN results are accurate enough to be used in the design of bolted flanges. Another optimization study was conducted on anisotropic laminated composites [27]. In this study a generic algorithm and two ANNs were used to optimize the design of a laminated composite. It was stated that the use of these methods lead to accurate enough solutions and

decrease the time required for the design process. Gajewski et al. [28] studied the use of the ANN for the optimization of a thin-walled structure. FEA results of the structure were used to train the ANN. This study is another example of that the ANN is an appropriate tool in design and analysis of airframe structures.

Buckling and collapse loads of panels have also been studied to create analysis tools based on ANN. Sadovsky and Soares [29] obtained post-buckling strength of a thin rectangular plate by creating an ANN as a function of initial imperfections. The created ANN was found to be able to provide reasonable collapse load results. The post-buckling optimization by using ANN was utilized for stiffened panels in a study of Lanzi and Giavotto [30]. The study used different optimization methods including ANN to optimize composite stiffened panels subjected to axial compression. The results were verified by tests and it was seen that accurate results can be obtained for both the buckling load and the collapse load. Mallela and Upadhyay [31] also used the ANN to calculate the buckling load. The study was focused on buckling load prediction of composite stiffened panels working under shear loads. FEA results for different composite structures were collected in a database to train an ANN tool. An efficient tool that can be used in optimization was created in the study.

In this study, an ANN based structural analysis tool that can be used in optimization of stiffened panels is created. The ANN is utilized to predict the buckling mode, the collapse load and the mass of metallic skin-stringer structures. The ANN is trained by using a database created with the input parameters and the FEA results of 1440 metallic skin-stringer assemblies subjected to uniaxial compression. The first buckling load and the collapse load are extracted from the nonlinear FEA results of the assembly. By this approach, long preparation and analysis times of a nonlinear analysis are eliminated. Using the results of the same analysis for the buckling and the collapse load, the time required for the generation of the training database is significantly reduced. Also, the first buckling load is obtained with an enhanced accuracy by using nonlinear analysis instead of linear buckling analysis.

CHAPTER 3

BUCKLING AND POST-BUCKLING ANALYSIS

Finite element analyses (FEA) of a complex structures require significant computing power and time to obtain accurate results. Nonlinearity in geometry and material properties, part and element sizes, load and displacement transmitting elements and connections between parts can be considered as main factors that increase computing time.

In this study, it is aimed to decrease both modeling and analysis time of skin-stringer assemblies with the use of Artificial Neural Networks (ANN). To achieve this purpose, 1440 different skin-stringer finite element models (FEM) are created and analyzed. Number of input-output data sets is vital in training of an artificial neural network in order to obtain an accurate ANN. Therefore, a convergence study is conducted to determine element sizes in the models. Likewise, different solution methods and parameters are tested to increase the performance of the ANN.

For structural analysis of the skin-stringer assemblies, a commercial FE software ABAQUS is used. ABAQUS is able to run scripts written in Python. To this end, Python 2.7 is utilized to create finite element models with user defined geometry and material properties.

Before the analysis of the skin-stringer assembly, linear buckling analyses of unstiffened panels are performed. Skin panels with different length, width and thickness values are created as two dimensional shell parts. The panels are analyzed with simply-supported and clamped boundary conditions to observe buckling phenomenon.

Creation of a skin-stinger assembly model requires a more detailed process than an unstiffened panel analysis. The skin is created as a two dimensional shell. Its thickness is defined from section properties. The stringers are modeled as three dimensional solid parts to apply user defined cross section properties. Then, three dimensional stringers are transformed into two dimensional shell parts by creating mid-surfaces. Material parameters are assigned to these parts. Fasteners are defined from the stringer flanges to the skin panel. The parts are meshed with proper element types and sizes. Then the proper load and boundary conditions are assigned. After the model is created, the analysis method and settings are specified. Finite element model creation process is repeated with different properties to have multiple solutions.

3.1 Buckling Analysis of Unstiffened Panels

Buckling of a structure can be defined as a sudden unstable deflection that occurs in slender structures subject to compression or shear loads. This stability problem can arise for loads much lower than the failure loads of the materials that the structure is made of. The structure becomes unstable after buckling. Further loading causes unexpected deformations and failures [32]. Thus, it is important to consider the effects of buckled structure on an assembly and calculate the early failures caused by buckling.

Buckling is not always considered as a failure mode since a buckled panel can still satisfy the design specifications [6]. However, buckling should be allowed with a great care and be restricted to certain conditions. For example, the outer skin of a passenger aircraft should not buckle on ground to prevent passenger from observing a deformed skin. Figure 3.1 shows a fuselage skin of a Boeing B-52 which is buckled on ground due to a heavy cargo. The weight of the cargo created a shear loading at the landing gear station. The skin absorbed the in-plane shear loading, which resulted in buckling of the skin in a part of the fuselage. To prevent such situations, the buckling stress of the panel should be greater than a desired design value. The factors that affect the buckling stress is listed as [7],

- Type of loading
- Boundary conditions
- Sheet dimensions
- Material properties of the sheet



Figure 3.1: A buckled fuselage skin [33]

General buckling equations of thin panels are studied under different loading systems and boundary conditions and available to engineers. These equations can be summarized for shear loading and compression loading cases [5]. Although a general buckling equation is the same for the compression and shear loading, meanings and values of parameters in the equation are different. The buckling stress is computed as,

$$\sigma_{cr} = \frac{k\,\pi^2\,E}{12\,(1-v^2)}\,(\frac{t}{b})^2 \tag{3.1}$$

where k is the buckling factor which is represented as k_c for compression and k_s for shear stress calculation. Similarly, b is loaded edge length of the panel for compression and short dimension of the panel for shear stress calculation. Geometric parameters are given in Figure 3.2.

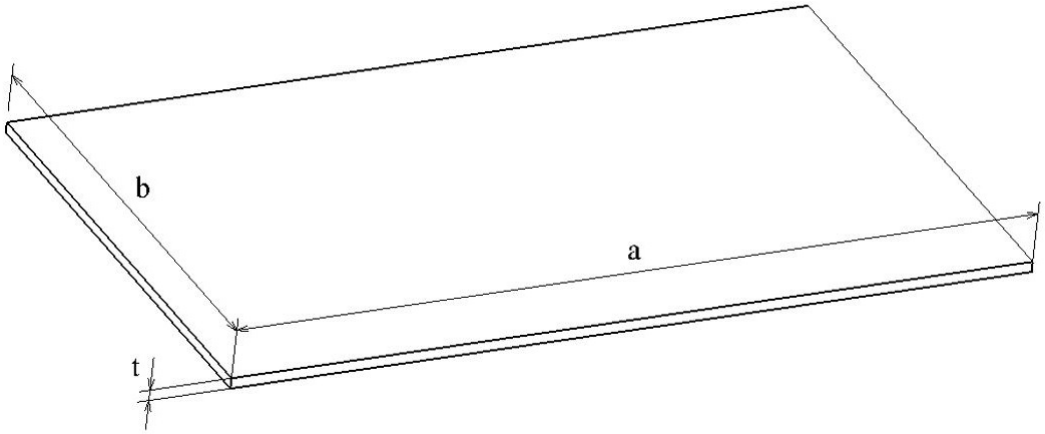


Figure 3.2: Panel geometric parameters

The other parameters are,

E : Modulus of elasticity

v : Poisson's ratio

t : Sheet thickness

Values for k_c and k_s depend on boundary conditions and dimensions of the panel. The values are obtained by test methods and given in a graphical form [8]. The compressive buckling factor k_c is given in Figure 3.3. The curves in the figure are obtained for simply supported, clamped, and free edge conditions. Five types of support combinations along unloaded edges are given in the figure. Support conditions of loaded edges are represented with dashed line for clamped edge conditions and continuous line for simply supported edge conditions. For example, dashed curve A is used for a panel with four clamped edges. The shear buckling factor k_s is given in Figure 3.4. The figure represents two types of support conditions: four simply supported (hinged) edges and four clamped edges. In Figure 3.4 symmetric and asymmetric modes are also shown. A symmetric mode means that the load deflection curve of the structure is symmetric about load axis. Similarly a mode shape is asymmetric if the load deflection curve is asymmetric about load axis.

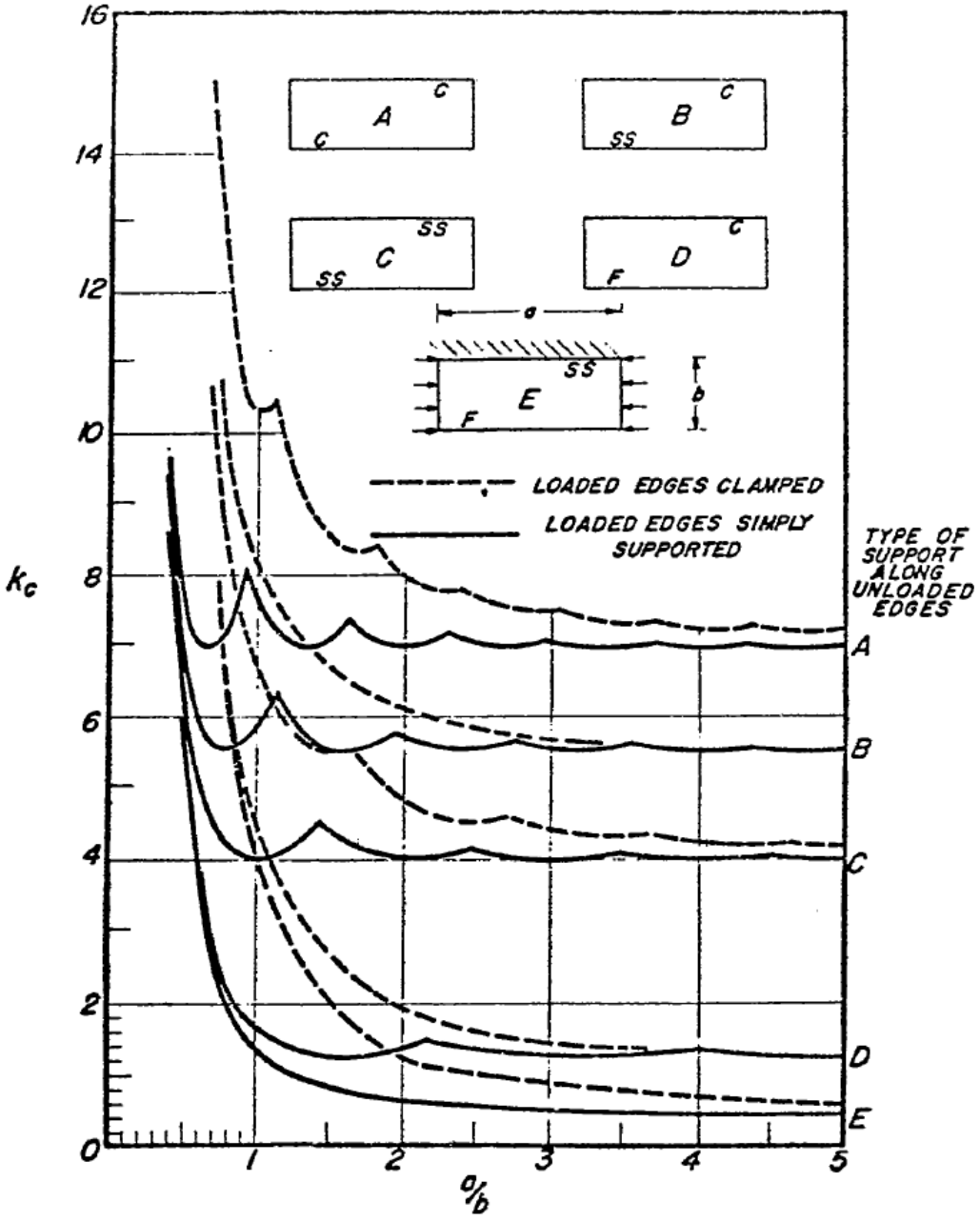


Figure 3.3: Compressive buckling factor with respect to panel aspect ratio a/b [8] (C:Clamped, SS: Simply supported, F: Free)

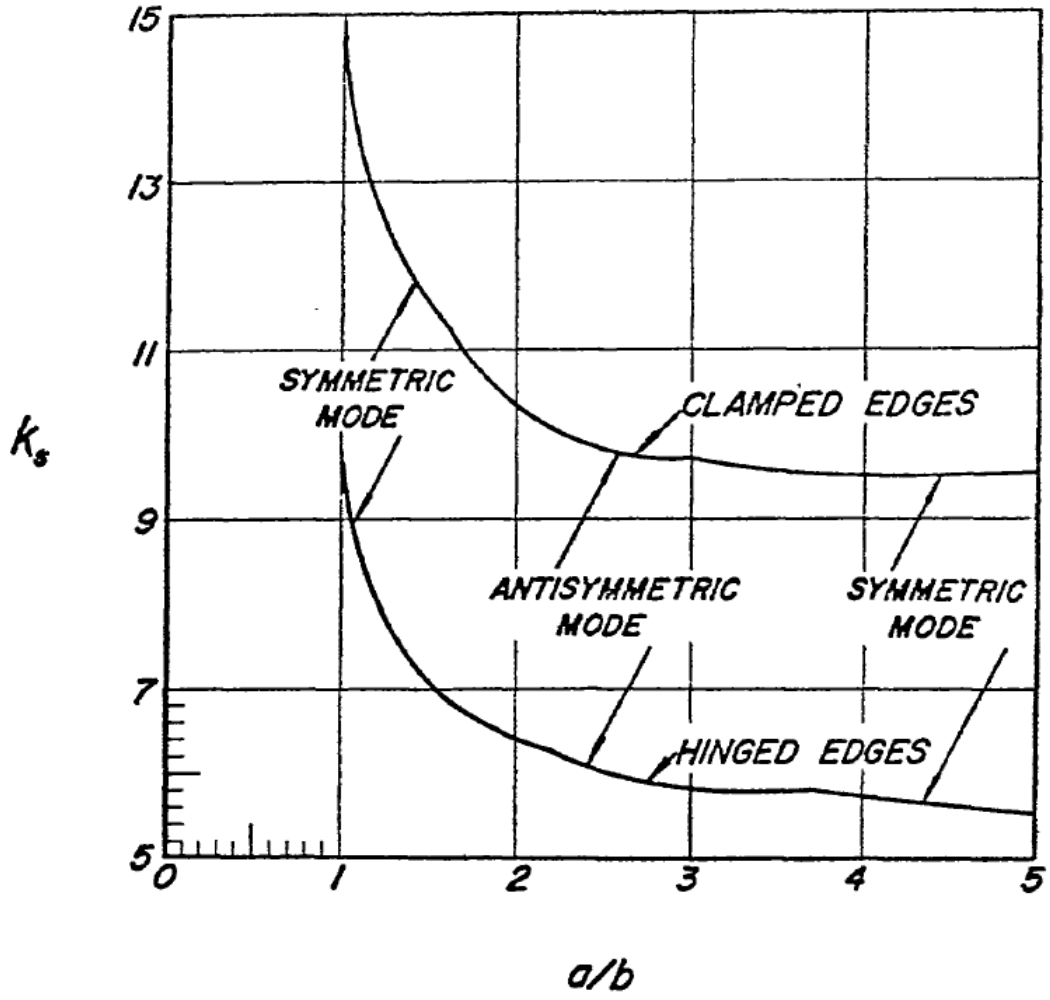


Figure 3.4: Shear buckling factor with respect to panel aspect ratio a/b [8]

The buckling stress can also be computed by using finite element analysis (FEA). To this end, ABAQUS linear buckling analysis solver "buckle" is used to determine the buckling load. This solution method is a linear perturbation procedure and is used to estimate critical buckling load of a structure [34]. The buckling analysis procedure seeks the load that results in a singular stiffness matrix. Thus, the following eigenvalue problem is solved,

$$K^{NM}\, v^{M} = 0 \tag{3.2}$$

where K^{NM} represents the stiffness matrix, and v^{M} represents nontrivial displacement solutions. Detailed calculation of K^{NM} is given in Appendix B.1.

In the eigenvalue prediction step, a load Q^{N} is defined. Q^{N} is scaled by load multipliers λ_i, therefore, its magnitude is unimportant. The general eigenvalue problem equation is,

$$(K_0^{NM} + \lambda_i\, K_\Delta^{NM})\, v_i^{M} = 0 \tag{3.3}$$

where,

K_0^{NM} : The stiffness matrix of the base state. The base state is defined as the last analysis step before the buckle step. If the buckle step is the first step, initial conditions defines the base step.

K_Δ^{NM} : The stiffness matrix induced from the incremental load Q^{N}

λ_i : Eigenvalues

v_i^{M} : Eigenvectors

M , N : Degrees of freedom

i : Buckling mode number

The analysis is used to extract the eigenvalues and the eigenvectors which represent the buckling load and the corresponding mode shape. The eigenvectors are normalized, thus, they do not represent the exact displacement values. The buckling load is obtained by scaling initial conditions with eigenvalues since the eigenvalue is actually a load multiplier.

3.1.1 Finite Element Model

ABAQUS, which is a commercial finite element analysis (FEA) program, is used to conduct analyses of an unstiffened panel. The modeling process is started with creating a geometry followed by meshing that geometry. Material properties are defined on the geometry. The process ends with definition of load and boundary conditions.

Analysis is conducted for flat thin panels under compression and shear loads separately. A flat panel is modeled as a shell part creating a rectangle of 200 mm wide and 400 mm long. The thickness is defined as 1 mm along the panel. The finite element model of the unstiffened panel with given dimensions can be seen in Figure 3.5. The shell is meshed with S4R elements which is a 4-node element with one integration point (reduced integration), having hourglass control and finite membrane strains. The element type S4R is recommended by ABAQUS in modeling of general purpose shells. For thin shells, S4R elements use discrete Kirchhoff thin shell theory and for thick shells thick shell theory is used [35]. As stated in ABAQUS documentation, theory to be used is determined according to the thickness of the element in calculations with S4R elements. Also, during the calculations, the changes in the thickness may cause the selected theory to change [34].

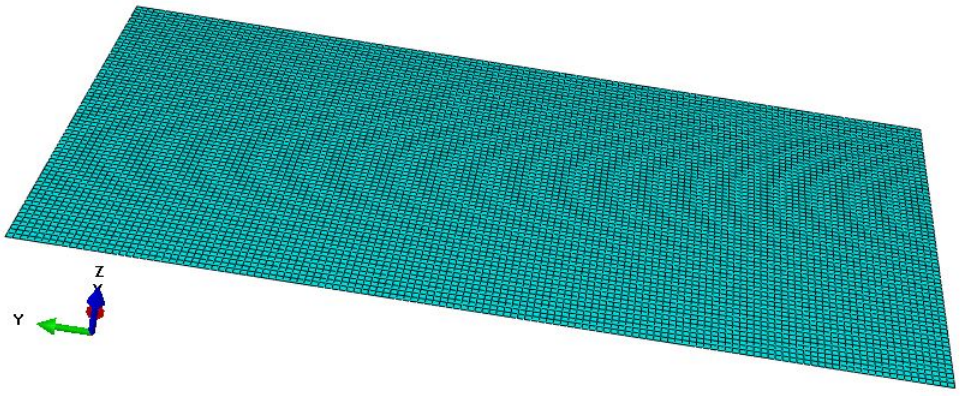


Figure 3.5: Finite element model of the panel

The element size used in the model is determined by a mesh convergence study. The panel is meshed with eight different element sizes. Then, the element size that provides an optimum result in terms of computational time and accuracy is chosen.

Element sizes used in the study and their results are given in Table 3.1. Also, Figure 3.6 shows the first eigenvalues with respect to number of elements. As the number of elements is increased, the first eigenvalue converges to a value which is approximately 6.787. The first eigenvalue does not change significantly for the element sizes of 3 mm and lower. To minimize that deviation without sacrificing extreme computation time, elements with 3 mm edge lengths are chosen. Consequently the unstiffened model is meshed with 8911 linear quadrilateral elements of type S4R.

Table 3.1: Eigenvalue results for different element sizes

Element Edge Size [mm]	Number of Elements	First Eigenvalue
0.5	320000	6.7872
1	80000	6.7888
2	20000	6.7933
3	8911	6.7980
4	5000	6.8024
5	3200	6.8067
8	1250	6.8280
10	800	6.8323

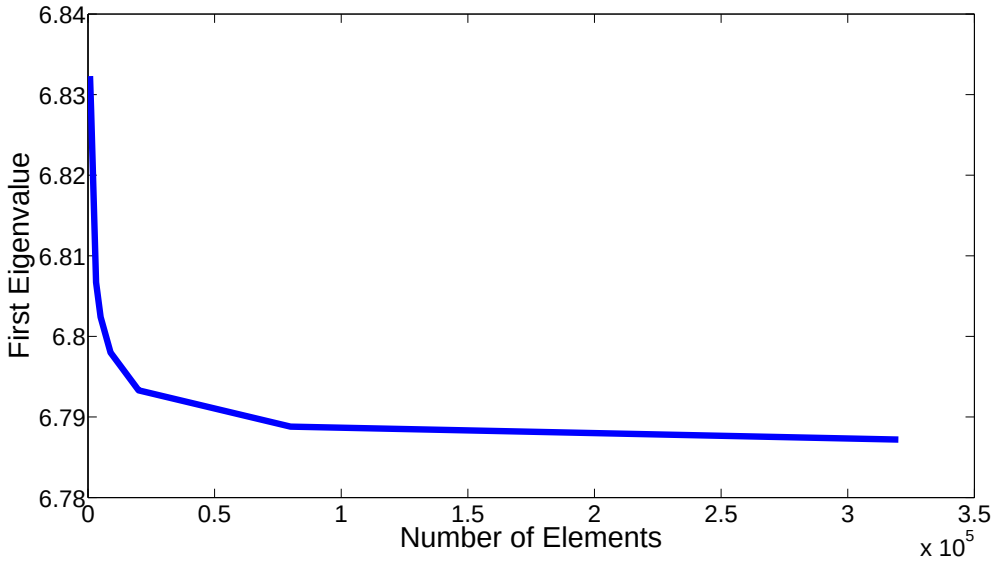


Figure 3.6: The first eigenvalues with respect to number of elements

The panel material is chosen as aluminum 2024-T3 due to its extensive utilization in aerospace industry. The following material properties are assigned for 2024-T3 aluminum alloy [36] (see Appendix A),

- Young's Modulus (E) : 73774 MPa
- Poisson's Ratio (v) : 0.33
- Density (ρ): 2.768 g/cc

To analyze the panel, "Buckle" step of the ABAQUS, which is a linear perturbation procedure, is utilized. ABAQUS offers two eigensolvers for the buckling analyses: Lanczos and Subspace, which is the default solver. In the documentation [34], it is stated that the Lanczos solver is faster but it has limitations and not as robust as the Subspace solver. Therefore, the Subspace solver is chosen in order not to encounter with a convergence problem.

In the Subspace solver, the number of requested eigenvalues, the vectors used per iterations and the maximum number of iterations can be specified. Since the concern of the analysis is to determine the lowest buckling load, the number of requested eigenvalues is specified as 1. Note that buckling analysis results of ABAQUS are arrenged from the minimum buckling load to the highest buckling load. ABAQUS suggests a suitable number of vectors for requested eigenvalues. For 1 eigenvalue, 2 vectors per iteration is chosen as suggested by ABAQUS. To avoid any convergence problem, the maximum number of iterations is chosen as 10000.

3.1.2 Loading and Boundary Conditions

The panel is subjected to unit compression and shear loads separately and the boundary conditions are chosen to represent simply-supported and clamped edge conditions. Simply supported boundary conditions can be simplified by allowing the edges to freely rotate in all directions [37] and restricting the translation in only out-of-plane direction. For modeling of clamped edge conditions, only axial translation is allowed, while all deflections in out-of-plane direction are restricted. Major difference between the simply supported and clamped edge condition is that

the rotation of each edge is restricted around corresponding edges [19] for clamped edge conditions while all the rotations are free for simply supported edge conditions.

The boundary conditions and the loads applied to the panel are shown in Figure 3.7 and Table 3.2. The conditions assigned are the same as in study of Muameleci [11]. In Table 3.2, restricted deflections are marked with "x". The compression loads are applied to panel edges AB and DC as unit shell edge loads. They are applied in negative y-direction to edge AB and in positive y-direction to edge CD.

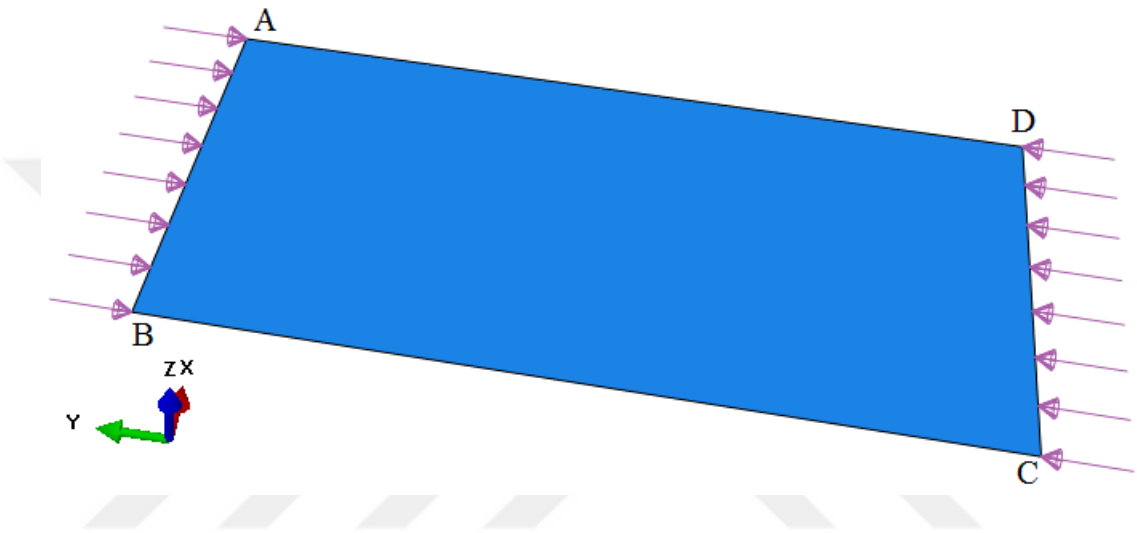


Figure 3.7: Compressive loading of the panel

Table 3.2: Translation and rotation restrictions for simply supported and clamped plates

	Simply Supported						Clamped					
	U1	U2	U3	R1	R2	R3	U1	U2	U3	R1	R2	R3
Edge AB			x						x	x		
Edge BC			x						x		x	
Edge CD			x						x	x		
Edge DA			x						x		x	
Point A												
Point B	x	x	x				x	x	x			
Point C												
Point D		x	x					x	x			

The shear load is applied to all four edges of the panel. It is applied in positive x-direction to edge AB, in negative y-direction to edge BC, in negative x-direction to edge CD and in positive y-direction to edge DA. The panel subjected to shear loads can be seen in Figure 3.8.

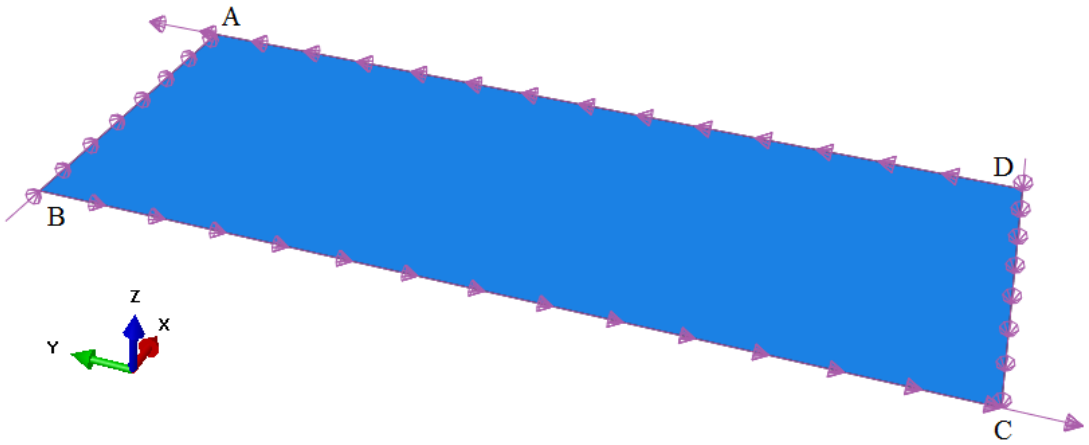


Figure 3.8: Panel subjected to shear loads

3.1.3 Finite Element Results

Buckling analysis calculates requested number of eigenvalues and mode shapes that represent the buckled view of the structures. The purpose of the analysis in this study is to determine the first buckling load. Therefore, only one eigenvalue is requested for each loading type and boundary conditions. Mode shapes obtained through the buckling analysis do not represent the actual displacement values but they are normalized to value of 1 at the node with maximum displacement. The mode shapes for the simply supported panel is given in Figure 3.9 and Figure 3.10 and for the clamped panel in Figure 3.11 and Figure 3.12.

For the simply supported panel in compression, the first eigenvalue is found to be 6.798 and for the simply supported panel in shear the first eigenvalue is found to be 11.146. These values increase to 13.397 and 17.472 when the panel boundary conditions are clamped. The reason for the increase is that the panel is more stable when the rotations along the edges are restricted. Therefore, the buckling, which is a stability problem, occurs at higher loads, resulting in higher eigenvalues.

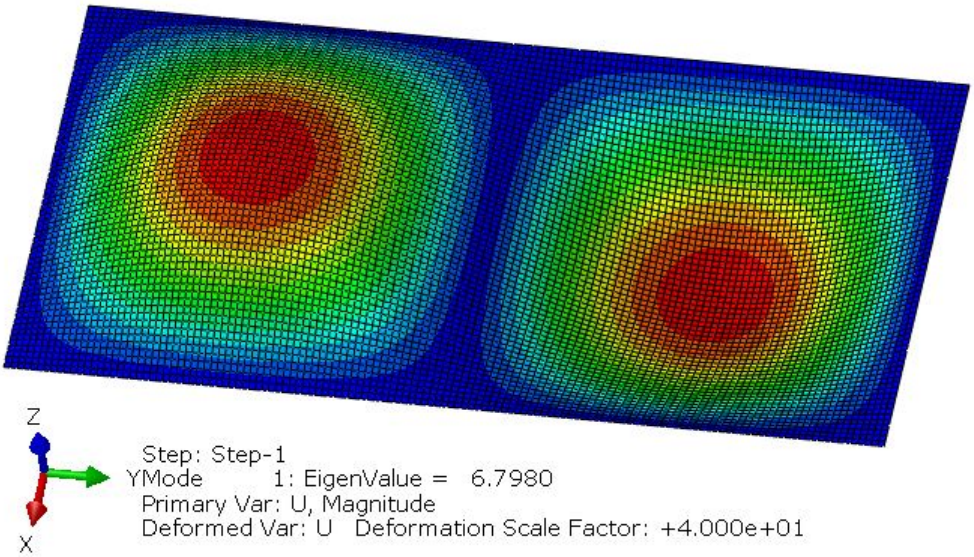


Figure 3.9: The first mode of the simply supported panel loaded in compression

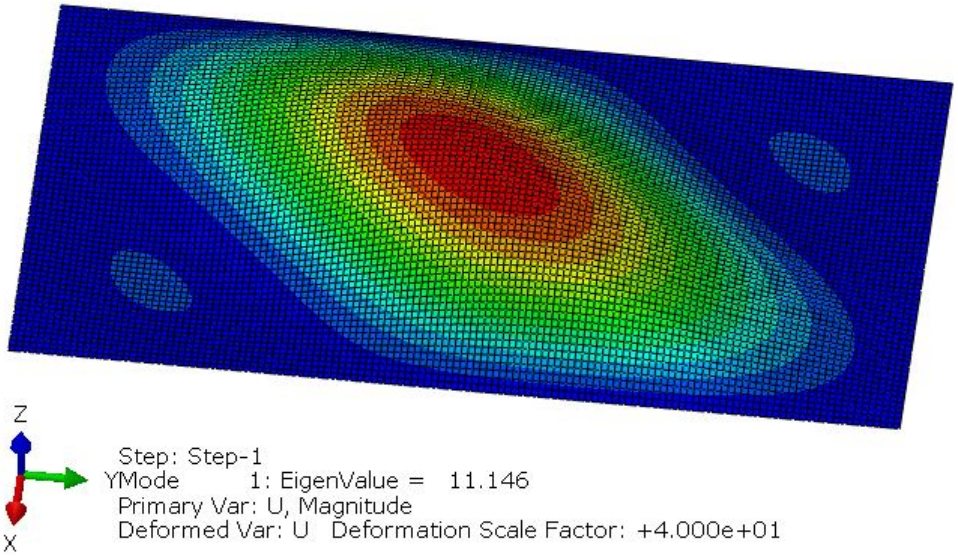


Figure 3.10: The first mode of the simply supported panel loaded in shear

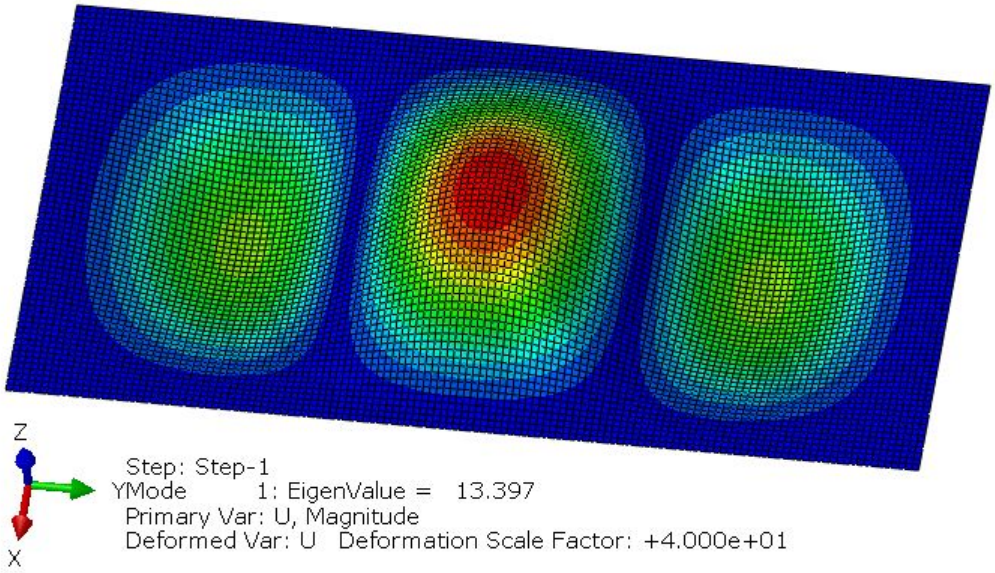


Figure 3.11: The first mode of the clamped panel loaded in compression

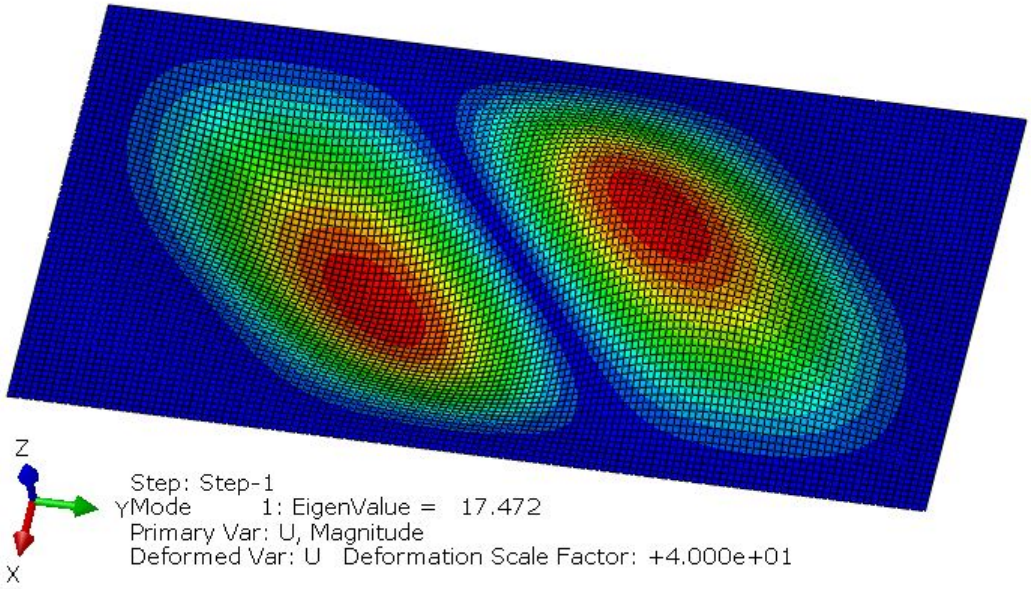


Figure 3.12: The first mode of the clamped panel loaded in shear

3.1.4 Comparison with Conventional Methods

Finite element results are verified with conventional analytical methods which are explained in Section 3.1. To compare the results, finite element modeling procedure given previous sections applied to panels with various lengths. Panels of 1 mm thickness and 200 mm width are created for lengths changing from 200 mm to 700 mm by 5 mm increments. The first eigenvalues are extracted from the analysis and the buckling factor of each panel is calculated by modifying Equation (3.1).

As explained in Section 3.1, the eigenvalue is a load multiplier that scales the initial conditions to critical loads. Therefore, the buckling load is calculated by scaling the applied load with the eigenvalue,

$$N_{cr} = N_{applied}\,\lambda \tag{3.4}$$

where N_{cr} is the critical buckling load, $N_{applied}$ is applied shell edge load and λ is the eigenvalue. The buckling stress of the panel is calculated from shell edge loads and the panel thickness as,

$$\sigma_{cr} = \frac{N_{cr}}{t} \tag{3.5}$$

Then, Equation (3.1) becomes,

$$\frac{N_{applied}\,\lambda}{t} = \frac{k\,\pi^2\,E}{12\,(1-v^2)}\,(\frac{t}{b})^2 \tag{3.6}$$

Finally, the buckling factor is,

$$k = \frac{12\,(1-v^2)}{\pi^2\,E}\,\frac{b^2}{t^3}\,N_{applied}\,\lambda \tag{3.7}$$

Calculated buckling factors are compared with the data in Figure 3.3 and Figure 3.4. The comparison curves are shown in Figures 3.13-3.16. In Figure 3.13 and Figure 3.14 all edges are simply supported, and in Figure 3.15 and Figure 3.16 all edges are clamped. The maximum difference in curves is found to be 4.36% which occurs in Figure 3.15 when the aspect ratio is 1.1. The maximum deviations in other comparison figures are 2.48% in Figure 3.13, 3.86% in Figure 3.14 and 2.8% in Figure 3.16. Consequently, it can be concluded that the FEA procedure in this study is consistent with the conventional method.

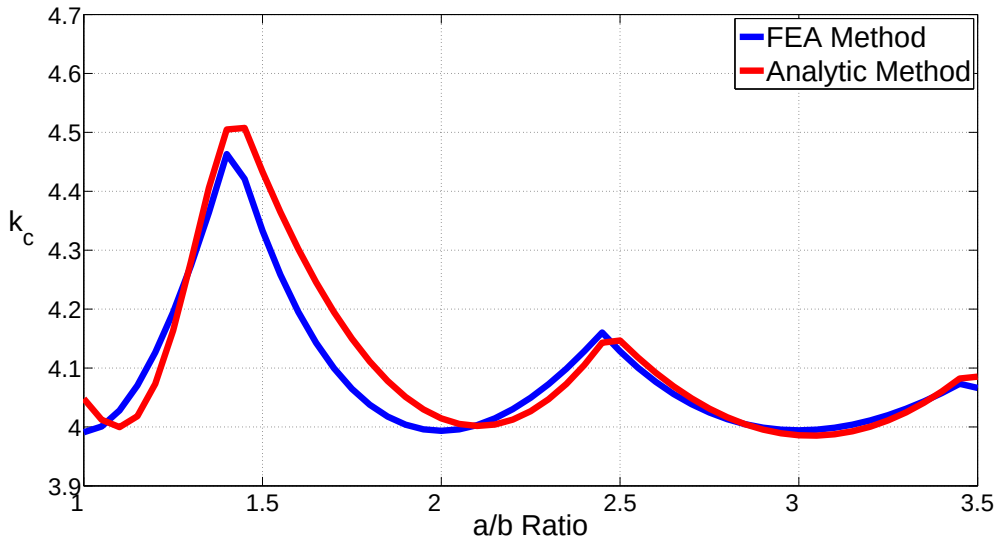


Figure 3.13: Comparison of compressive buckling factor for plates with simply supported edges. (Compared with continuous curve C of Figure 3.3)

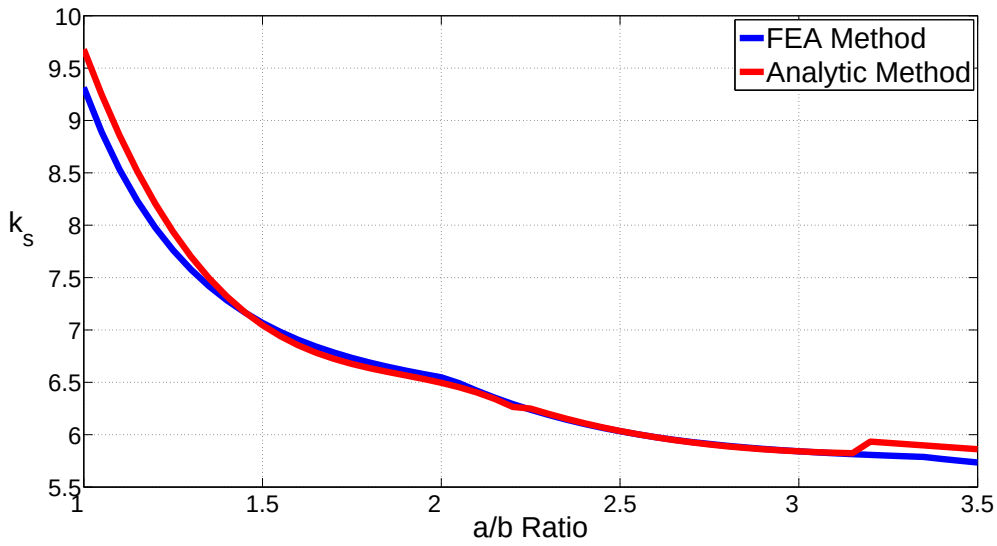


Figure 3.14: Comparison of shear buckling factor for plates with simply supported edges. (Compared with hinged edge curve of Figure 3.4)

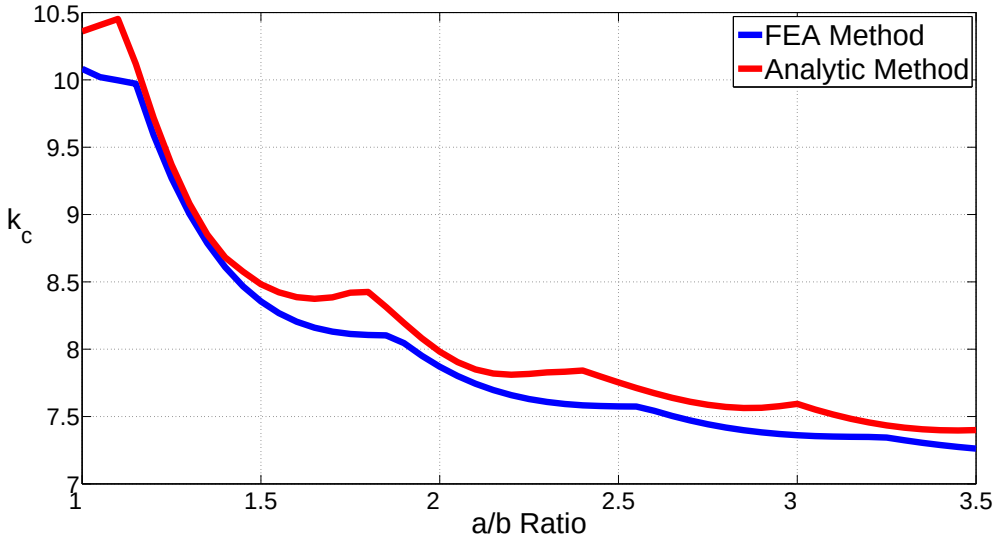


Figure 3.15: Comparison of compressive buckling factor for plates with clamped edges. (Compared with dashed curve A of Figure 3.3)

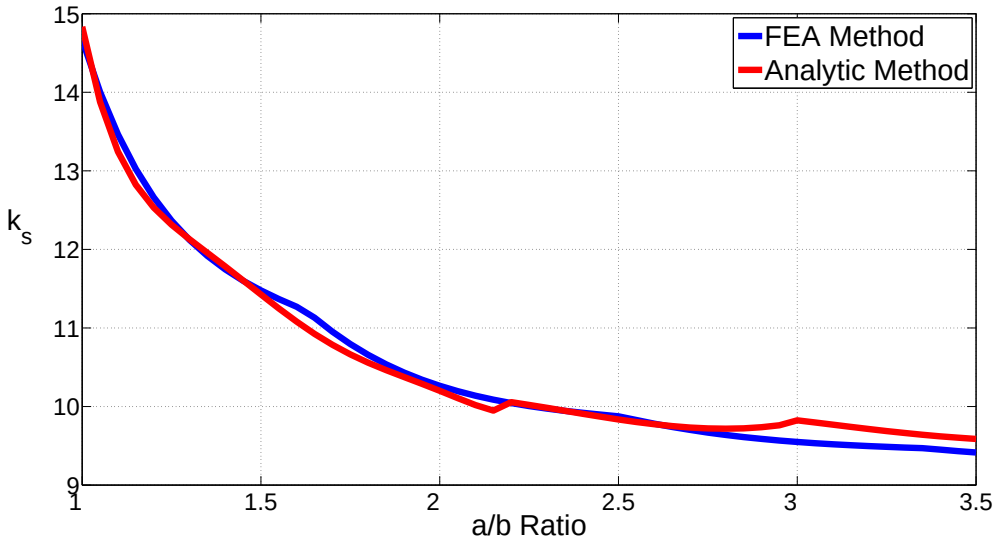


Figure 3.16: Comparison of shear buckling factor for plates with clamped edges. (Compared with clamped edge curve of Figure 3.4)

3.2 Buckling and Post-Buckling Analysis of Skin-Stringer Assemblies

As explained in Section 3.1.1, the first step of finite element analysis is the generation of a geometric model. In order to create a skin-stringer assembly model, a rectangular flat panel, six stringers and a frame are used. Drawing of the skin-stringer assembly is given in Figure 3.17, and the stringer cross section in Figure 3.18.

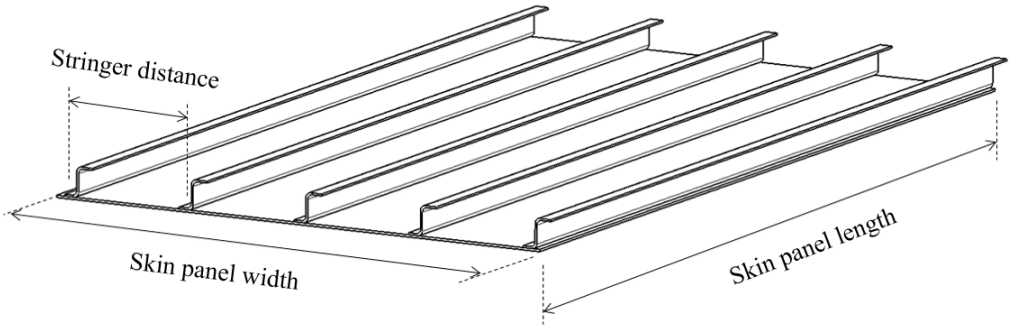


Figure 3.17: Drawing of the skin-stringer assembly

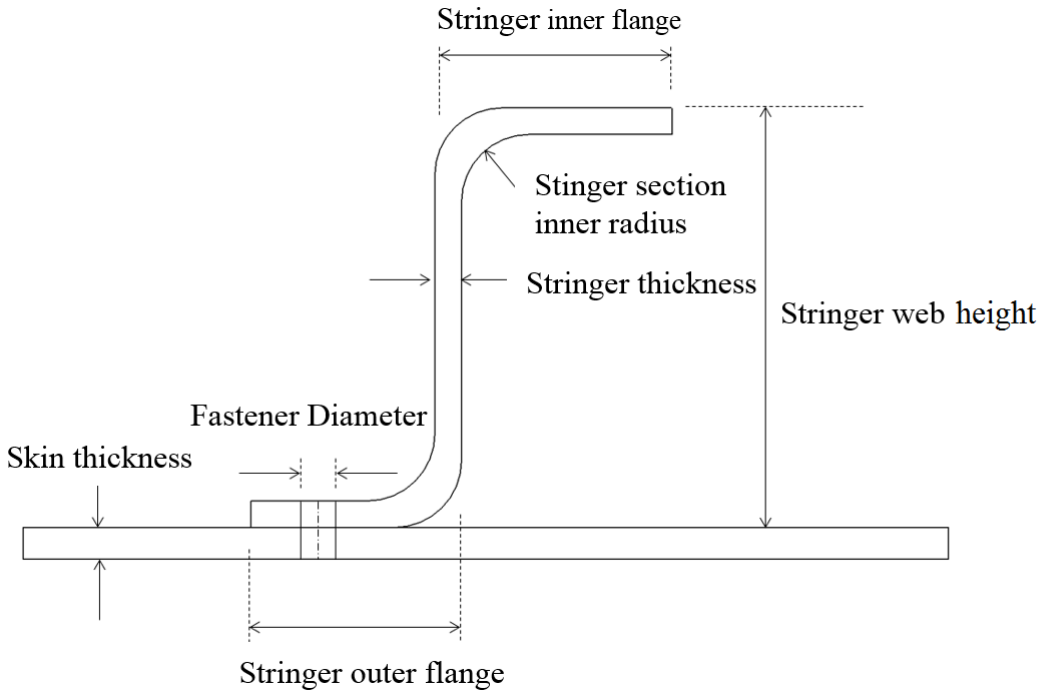


Figure 3.18: Drawing of the stringer cross section

The assembly dimensions are chosen as common aircraft design values. The stringers are connected to the skin panel with "Fastener" type connectors of

ABAQUS to simulate rivet connections. This type of connectors are composed of a rigid beam connected from its ends to the nearest nodes of the components to be connected [34]. Then, the geometry is meshed with shell elements. The mesh size is determined after conducting preliminary convergence studies.

3.2.1 Geometry

The stringer cross-section of the skin-stringer assembly is chosen as Z type which has one flange connected to the skin panel and one flange free at the other end of the stringer web as shown in Figure 3.19.

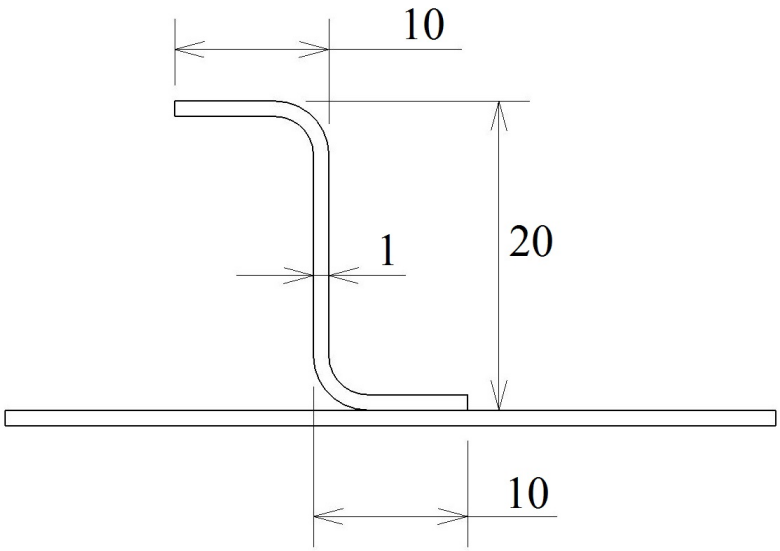


Figure 3.19: Z type stringer cross section

The Z type is one of the most common design configuration since it is structurally effective and easy to assemble [38]. The stringers with Z cross-section of 10 mm flange width and 20 mm height are created and connected to the skin panel, which is 410 mm wide and 450 mm long with fasteners of 3.2 mm diameter.

The fastener distance on an airframe structure is suggested to be equal to or greater than 4 times the diameter of the fastener ($s/d > 4$) in order to avoid any stress concentration around the rivet holes[2]. To understand the effect of the fastener spacing, a study that compares the collapse load of models with different fastener spacing is conducted. The fastener spacing to be assigned to the skin-stringer

assembly is determined by observing its effect on the collapse load. Nine different finite element models are created with the fastener spacing to fastener diameter (s/d) ratios given in Table 3.3. The fastener diameter is set as 3.2 mm to all models, which is one of the most common fastener size in aerospace industry.

Table 3.3: Collapse load results for different number of fasteners

Number of Fasteners	(s/d) Ratio	Collapse Load [kN]
34	4.0	113.06
33	4.1	112.83
32	4.3	112.65
31	4.4	112.36
30	4.5	112.06
29	4.7	108.89
27	5.0	107.85
23	6.1	103.25

The results show that models having (s/d) ratio between 4 to 4.5 result in similar collapse loads. However, the models with larger fastener spacing causes failing at significantly lower loads. Therefore, considering design parameters suggested in literature [39, 2], fasteners are located at the middle of the stinger flanges with s/d ratio of 4.5. The final configuration of the fasteners can be seen in Figure 3.20.

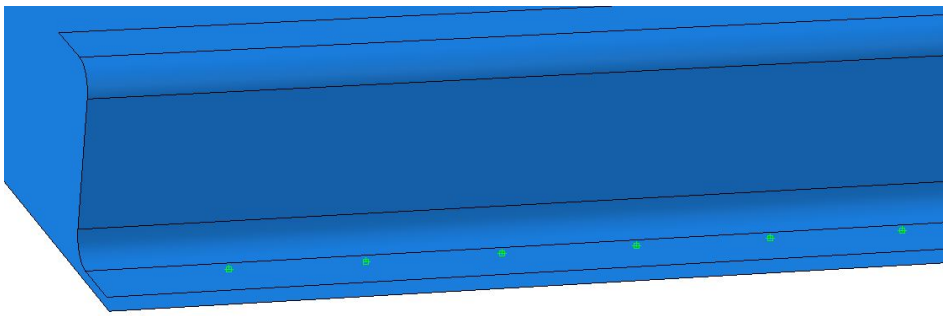


Figure 3.20: Fastener connections

In an aircraft structure, frame is bulkier and significantly stiffer compared to stringers and longerons (see Figure 3.21). Therefore, the frame can be idealized and assumed to be rigid in creation of the model [40]. A rigid frame of 440 mm width and 30 mm

height is connected to edges of the panel and the stringers. The finite element model generated is shown in Figure 3.22.

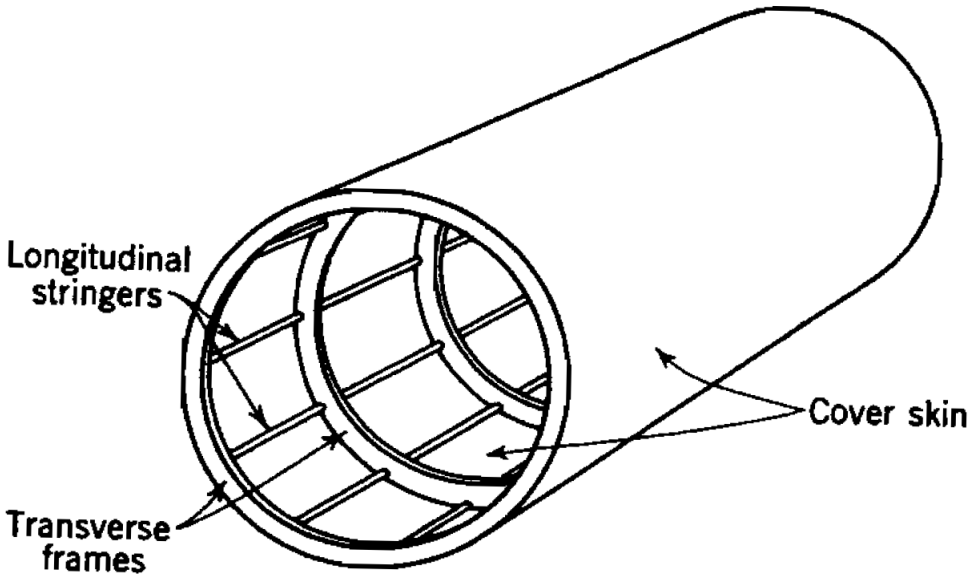


Figure 3.21: Typical body structure [40]

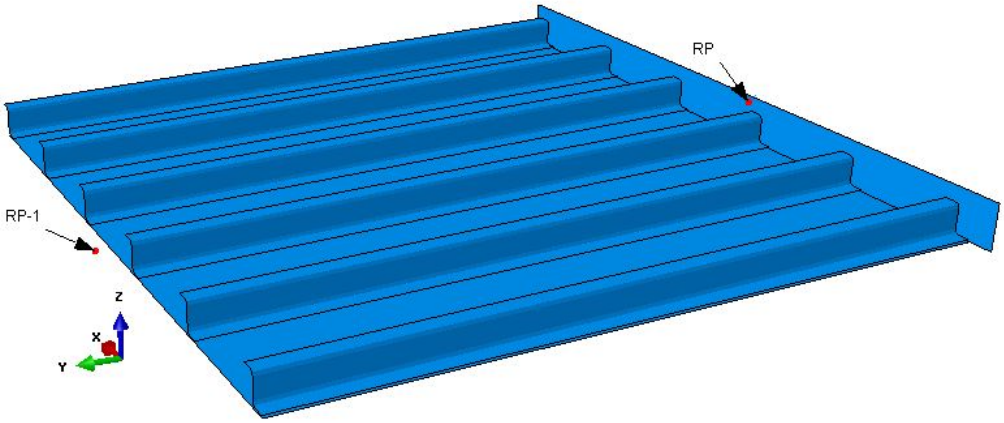


Figure 3.22: Finite element model of the skin-stringer assembly

To avoid the convergence problem explained in Section 3.2.3, compression is applied in a displacement controlled manner. The displacement is applied on a reference point on the frame. The reference point, which can be seen as "RP" in Figure 3.22, is created as a node of the frame, thus the point moves with the rigid frame in accordance with the displacement applied on it. The frame is connected to the stringers and the skin panel with a surface to surface tie constraint. The tie constraint is defined by

ABAQUS as a connection of the closest nodes on different surfaces equalizing the all active degrees of freedom [34]. Also, the tie constraint provides a rapid transition between different mesh sizes allowing the rigid frame to be meshed coarsely. The reference point “RP-1” is created to be symmetrical to mid-plane of stringers and is connected to each node on (0,y,z) plane via kinematic coupling type connection. The kinematic coupling is suggested for constraining a group of nodes to degrees of freedom of a master node. "RP-1" point is restrained in all directions and rotations as a boundary condition and assigned as the master node.

3.2.2 Material Properties

A homogeneous shell section of 1 mm thickness with aluminum 2024-T3 clad sheet properties is created to be assigned to the stringers and the skin panel. The frame is created as a rigid part, thus, material and section properties are not needed for the frame. 2024-T3 is a commonly used aluminum alloy in aerospace industry. The first digit 2 in its name indicates that the alloying element is the copper. 2000 series aluminums have optimum properties of a mild steel. Due to low corrosion resistance, 2000 series aluminums clad with high purity alloys. 2024 aluminum has 1.5 % magnesium, 0.6 % manganese alongside with 4.5 % copper as alloying elements. The physical properties of aluminum alloys can be altered by heat treatments. T3 in the name indicates the heat treatment temper of the material. T is used for materials strengthened by heat treatment. The designation continues with a number that designates the heat treatment process and condition. T3 stands for solution treated, cold worked, and naturally aged to a stable condition [41]. Material properties of the 2024-T3 clad sheet are listed as

- Young’s Modulus (E) : 65500 MPa
- Poisson’s Ratio (v) : 0.33
- Density (ρ): 2.768 g/cc
- Yield Strength (σ_0): 310 MPa
- Ramberg-Osgood Number (n): 15

A more detailed information about material properties of the 2024-T3 clad sheet is given in Appendix A.

The uniaxial material response of the material is described by the Ramberg-Osgood equation [42].

$$\epsilon = \frac{\sigma}{E} + 0.002\,(\frac{\sigma}{\sigma_0})^n \tag{3.8}$$

where ϵ is the uniaxial strain, σ is the uniaxial stress, E is the elastic modulus, σ_0 is the yield strength, and n is the Ramberg-Osgood number.

The stress-strain curve drawn as blue line in Figure 3.23 is digitized from stress-strain curve given in MMPDS [36] and the red line is obtained by the Ramberg-Osgood equation. In order to obtain an unloading curve parallel to the elastic loading curve of the material, isotropic hardening plasticity of ABAQUS is used without a hardening parameter.

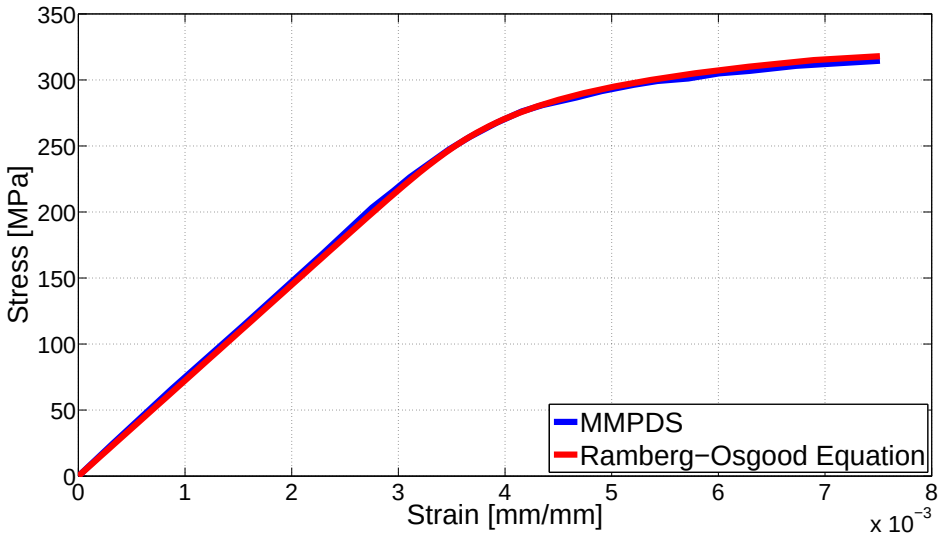


Figure 3.23: Stress-strain curves suggested by MMPDS [36] and obtained from Ramberg-Osgood Equation

There are two different modulus of elasticity of clad materials: primary modulus of elasticity and secondary modulus of elasticity. Cladding material contributes to the stiffness of the material up to its yield strength. However, the yield strength of the cladding material is usually very low. For example, primary modulus of elasticity of 2024-T3 clad aluminum, which is 72395 MPa, is valid up to 40 MPa. It is suggested that secondary modulus of elasticity must be used in design and analysis of a structure

made of a clad aluminum [36]. Therefore, secondary modulus of elasticity is assigned to the model as 65500 MPa as suggested.

ABAQUS requires true stress and corresponding true strain data to simulate plastic behavior of the structure. This information is provided by calculating engineering stress-strain data via (3.8) and converting the engineering stress-strain data to true stress-strain data as [43]

$$\sigma_{tr} = \sigma_{eng}(1 + \epsilon_{eng}) \tag{3.9}$$

$$\epsilon_{tr} = \ln(1 + \epsilon_{eng}) \tag{3.10}$$

3.2.3 Mesh Quality

All components of the skin-stringer assembly are modeled as shell elements. The element type used in Section 3.1.1 is assigned to the shell elements. To determine the suitable element sizes, a mesh convergence study is conducted. However, the rigid frame is not included in the study since it does not deform but transfers the applied deformation to other components. Seven models with different element sizes are examined by the means of the collapse load of the assembly. In total seven models with mesh sizes of 1 mm, 2 mm, 3 mm, 4 mm, 6 mm, 8 mm and 10 mm are created. The models are solved, and the buckling and the collapse loads are extracted from the results. Table 3.4 and Table 3.5 show the collapse and buckling loads for each mesh size, respectively.

Table 3.4: Collapse load results for different element sizes

Element Edge Size [mm]	Number of Elements	Collapse Load [kN]
1	281832	112.16
2	69207	112.24
3	31482	112.06
4	19229	113.53
6	8832	115.16
8	5676	113.74
10	3867	127.48

It can be seen that for the mesh sizes 1 mm, 2 mm, and 3mm, the collapse loads are almost constant and are roughly equal to 112 kN. For mesh sizes 4 mm, 6 mm, and 8 mm, the collapse loads are 114 kN with slight changes. The model with 10 mm mesh size leads to a collapse load of 127.5 kN which is significantly higher than the other results. The buckling patterns for the coarsest and the finest discretization exhibit some differences as shown in Figure 3.24.

Table 3.5: Buckling load results for different element sizes

Element Edge Size [mm]	Number of Elements	Buckling Load [kN]
1	281832	34.99
2	69207	37.99
3	31482	37.04
4	19229	36.70
6	8832	40.94
8	5676	45.19
10	3867	53.61

The buckling loads of the models with respect to number of elements are given in Table 3.5. In the figure, the buckling loads of the models having more than 15000 elements are similar.

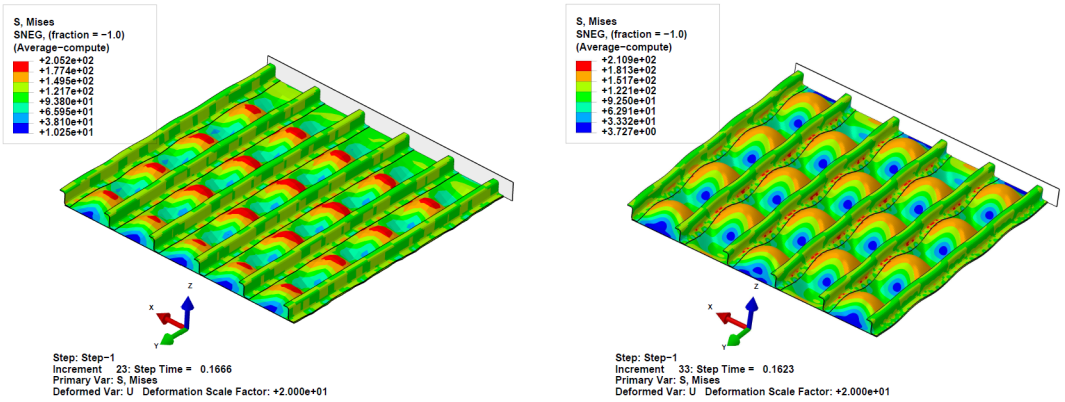


Figure 3.24: Buckling patterns of the models with element sizes of 2 mm and 10 mm

The force-deflection curves of the structure for different mesh sizes are given in Figure 3.25. The models with 3 mm or smaller element sizes lead to very close load

displacement curves, buckling loads and collapse loads. It should be noted that these models have at least three elements along the stringer flange widths.

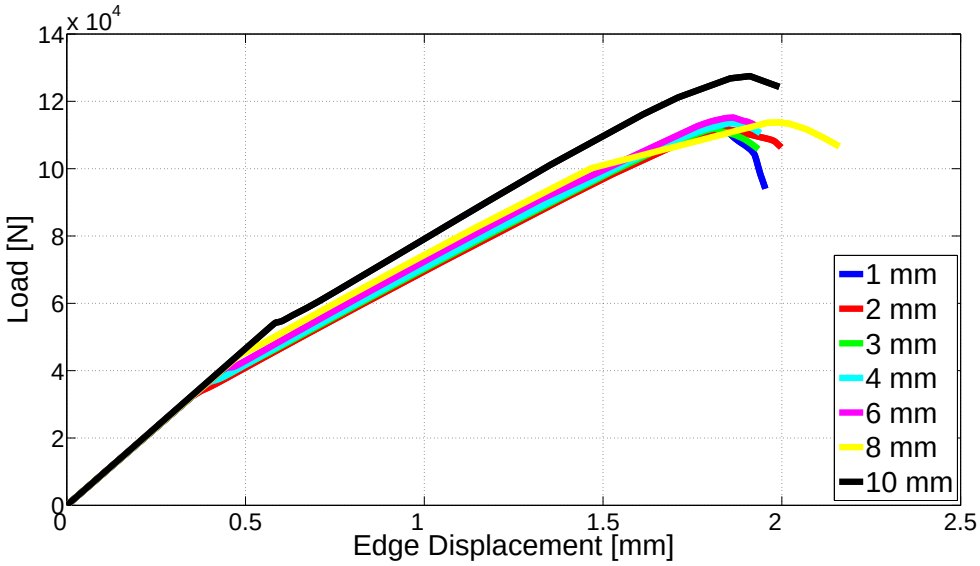


Figure 3.25: Force-displacement curve of the models with different element sizes

As a result of the mesh study the assembly is meshed with elements of 2.2 mm size that results in three elements along the stringer flange widths. The meshed geometry and an its enlarged view are shown in Figure 3.26 and Figure 3.27 respectively.

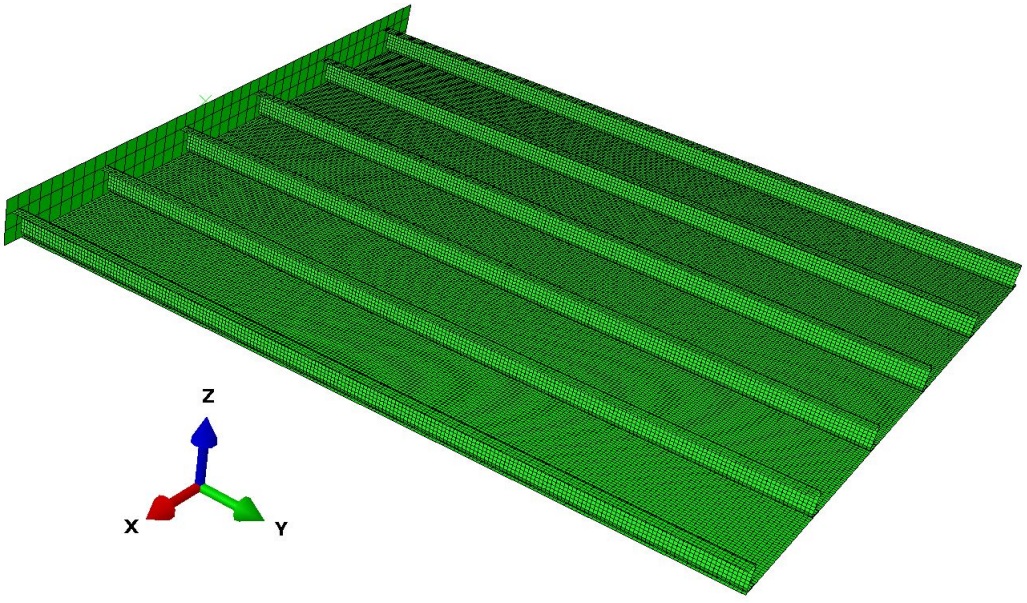


Figure 3.26: Meshed skin-stringer assembly model

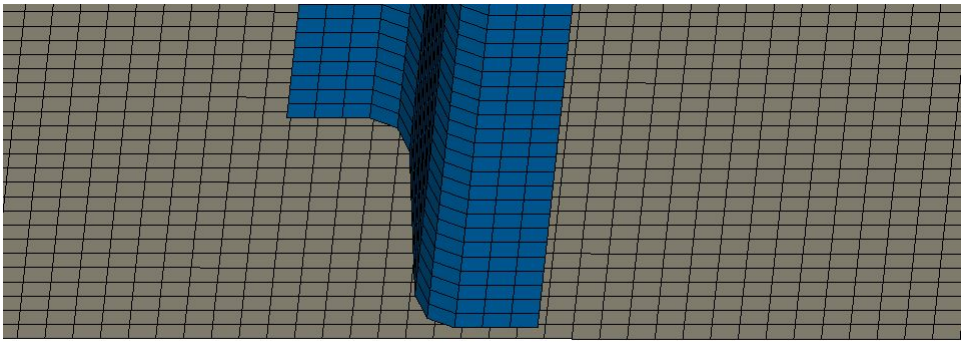


Figure 3.27: Enlarged view of the meshed skin-stringer assembly model

3.2.4 Finite Element Solution Schemes

In order to solve the buckling problem of the assembly introduced in the previous section, the *Riks* step and the *Static, General* step of ABAQUS are studied. A detailed description of the two solvers are given in Appendix B.

The Riks step is a solver that is utilized to solve nonlinear problems using arc-length method. The method is suggested for solving snap-through buckling analysis. However, if the solution involves bifurcation, the analyses may not be completed [34]. Buckling point in post-buckling analysis creates a bifurcation. Therefore, the branch that must be followed by the solver should be introduced by defining an initial imperfection, which is the buckling mode in this case.

In order to compare the Riks and the Static, General solvers of ABAQUS, the best configuration for the Riks solver should be determined. Therefore, a study is conducted to understand the effect of the initial imperfections on the Riks solver. The imperfections are defined by the results of an analysis obtained by the Buckle step of ABAQUS, which is the buckling solver used in Section 3.1. Due to the nature of a buckling problem, load applied on the structure cannot be displacement. Therefore, the applied displacement is replaced with unit force on the same location. First four mode shapes are requested from the analysis since the higher mode shapes are unlikely to occur. The linear buckling analysis executed with this configuration leads to the mode shapes given in Figure 3.28.

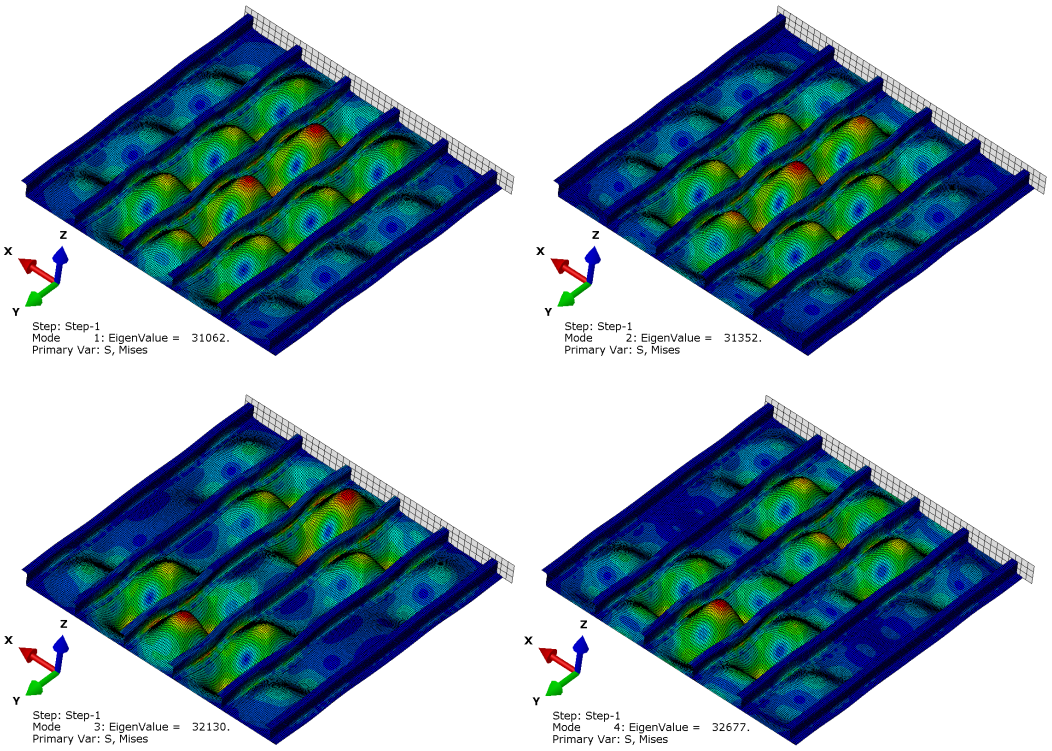


Figure 3.28: First four buckling mode shapes

The Riks solver is used by specifying separately the first three buckling modes as initial perfections. Also, a solution is obtained without any imperfection (the ideal model) to observe the effect of imperfections on convergence of the solver. The force-displacement curves are extracted and given in Figure 3.29.

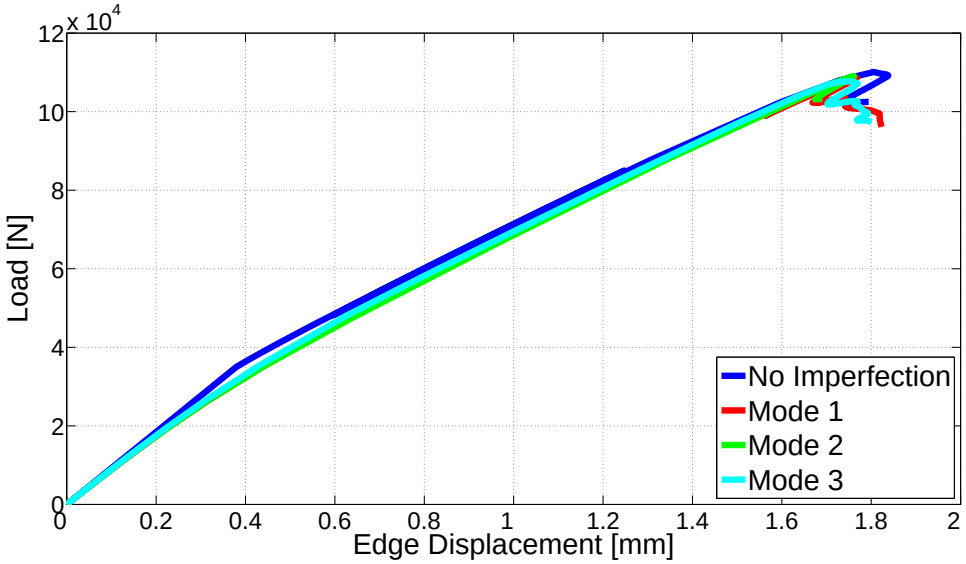


Figure 3.29: Riks solver results with different initial imperfections

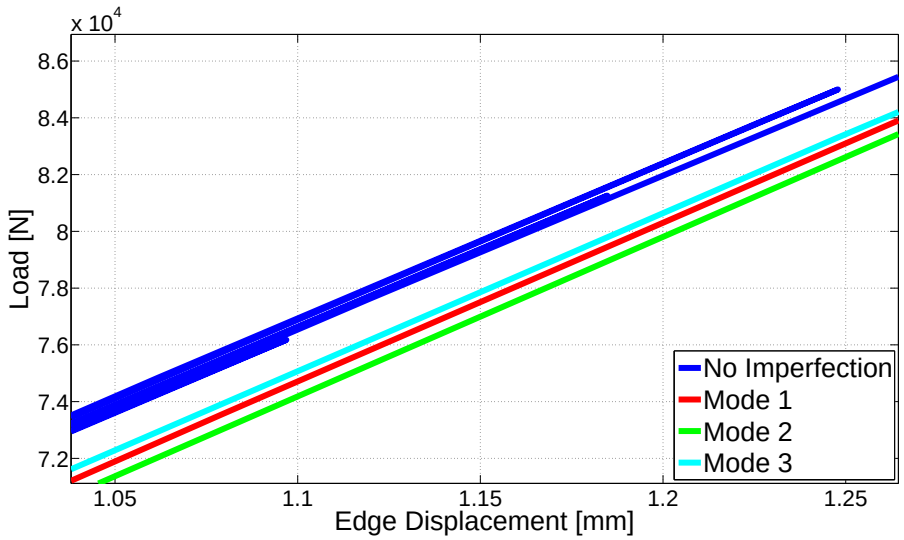


Figure 3.30: Disturbances after the buckling

As it can be seen in the figure, the response of the model without any imperfection is slightly stiffer than the models with imperfections. Also, the curve of the ideal model encounters with disturbances after the bifurcation at the buckling. The disturbances are shown in Figure 3.30 which is a magnified view of Figure 3.29. These disturbances are not seen in the results of the models with imperfections. Thus, it can be concluded that the Riks solver must be used with prior buckling analysis to avoid convergence problems and unrealistic analysis results. The imperfections defined by the mode shapes 1 and 2 result in almost identical force-displacement curves. This can be due to the similarities in mode shapes obtained by buckling analysis. The mode shape 3, which is substantially different from the first two shapes, guides the solver to the higher buckling load and the slightly lower collapse load. Considering the first mode shape is most likely to appear due to its lower buckling load, it is reasonable to use the Riks solver with the first mode shape.

The Static, General step employ the Newton's method to solve nonlinear problems. The method is more generic than the arc-length method. However, the solution may diverge after a critical point is reached (see Appendix B). To avoid this type of divergence problem, displacement control can be applied instead of load control to the structure. Due to geometric nature of a post-buckling analysis, instabilities are

developed during solution. If the instability is global, the Riks step can be used with an initial imperfection to achieve convergence. However, to overcome local instabilities, the Static, General solution with a stabilization procedure must be used. In the Static, General solution adaptive automatic stabilization option is used to increase the convergence of the solver. Automatic stabilization option creates an artificial damping leading dissipation of a part of released strain energy. If the artificial damping is too high, results may not be accurate [34]. Therefore, a tolerance is defined to control the created damping and released strain energy. Dissipated energy fraction is specified as 0.0002 and maximum ratio of stabilization to strain energy is defined as 0.05 as ABAQUS defaults. Also, nlgeom option is enabled to allow large displacements in step definition.

The Riks solver and the Static, General solver are compared by their load displacement curves. The force-displacements curves in Figure 3.31 are obtained by the Static, General solver and the Riks solver with initial imperfection. By introducing the initial imperfection with first buckling mode shape, the Riks solver overcomes the global instability at the buckling point. However, the solver also encounters with disturbances at the region near the collapse point due to the second instability. This situation leads the Riks solver to follow an incomprehensible loading path. Also, the disturbances at the collapse point require several additional iterations leading to high computational costs.

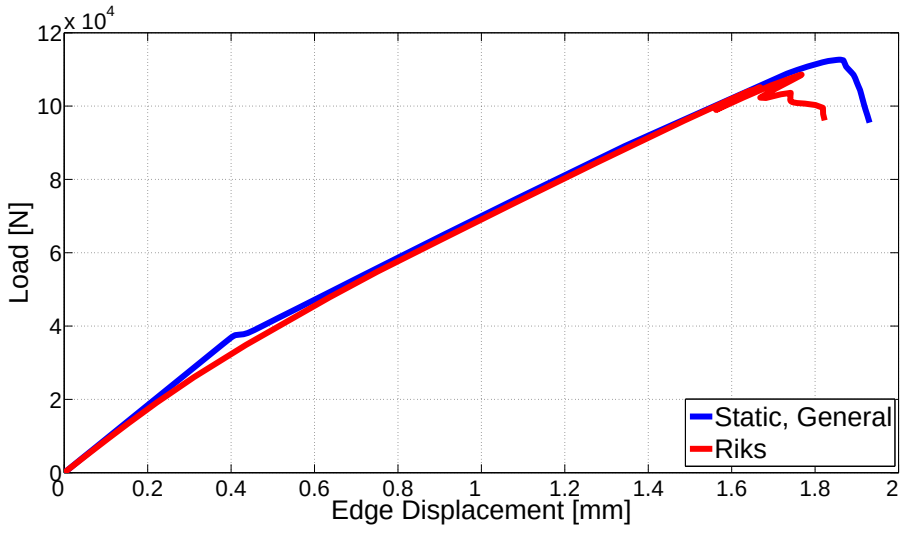


Figure 3.31: Comparison of Static, General solver and Riks solver

The solution step of the analysis is chosen to be the Static, General solver due to its robustness and suitability for post-buckling analysis. After the solution step is assigned, two additional information is requested as history outputs to be able to examine the force-displacement curve of the model. Reaction forces of point "RP-1", which is the application point of the boundary conditions, is requested with the displacement values of point "RP", which is the application point of displacement. To extract forces on the nodes, section forces are also requested. Finally, finite element creation process is completed and the analysis is run with the configurations stated in this section.

3.2.5 Finite Element Analysis Results

As it is stated in previous sections, the aim of the analysis is to determine the buckling load, the collapse load, and the mass of the skin-stringer assembly. For this purpose, the analysis results are examined and a force-displacement curve is obtained. The force data of the curve is obtained from the reaction forces at point "RP-1" and the displacement data is obtained from the point "RP". From Figure 3.32, one can see that the slope of the force-displacement curve is constant up to a point marked as circle 1 at a displacement value of 0.4 mm and a force value of 37 kN. From this point, the curve continues with a reduced slope. This reduction in the slope represents a loss of stiffness of the structure. The stiffness loss can be explained by the first buckling phenomenon. Before the buckling point, the skin panel and the stringers withstand the applied deformation with their combined stiffnesses. After the buckling, the buckled part of the skin panel looses its stiffness, while the unbuckled portion continues to contribute to the stiffness of the stringers. The effective portion of the skin panel can resist to external loading until the stringers collapse. The collapse point of the skin-stringer assembly can be seen in Figure 3.32 as circle 2. The skin-stringer assembly is found to posses the structural properties given in Table 3.6.

Table 3.6: Structural properties obtained from the finite element analysis

Buckling Load [kN]	37.10
Collapse Load [kN]	112.65
Mass [g]	775

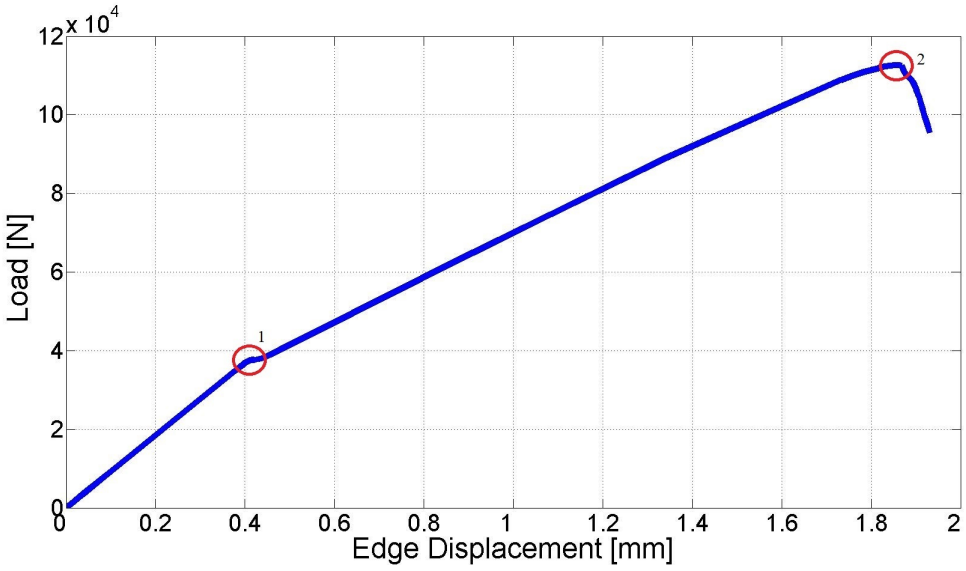


Figure 3.32: Force-displacement curve of the skin-stringer assembly

The buckling load is extracted by observing the rate of the slope and the collapse load is distinguished by spotting the peak value of the force. To extract the mass of the assembly, mass properties tool of the ABAQUS is utilized. ABAQUS calculates the mass of the system from material and geometry information by using the density and the volume.

3.2.6 Effective Skin Width in Post-Buckling Region

A skin-stringer assembly exhibits two distinct stiffness values when loaded in compression until failure. The first stiffness value merges from combination of stiffnesses of the skin panel and the stringers. This combined stiffness is effective until the skin panes buckles. After the buckling, buckled skin panel cannot carry more load leading to a reduced stiffness of the skin-stringer assembly. Therefore, the excess load is redistributed to the stringers, which have higher load carrying capacity [44]. From this moment, the skin-stringer assembly cannot use the stiffness of the buckled portion of the panel. However, stiffened portion of the skin panel remains unbuckled during the redistribution and carries additional load after buckling. The unbuckled region of the skin panel is called as the *effective skin* [45]. The redistribution of the load exposes the buckling mode of the structure. As the loading

continues, buckled shape of the structure becomes more distinctive. Also, the difference between load carried by the buckled portion and the effective portion of the skin panel increases [15]. The distribution of loads after buckling is illustrated in Figure 3.33, which is adapted from Collier et al. [15].

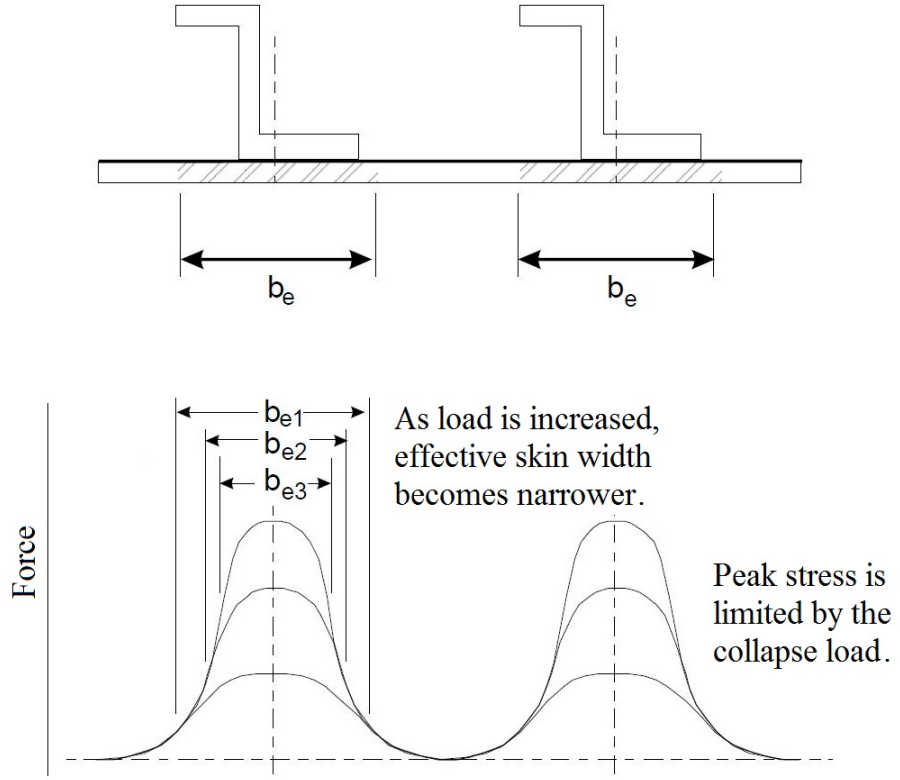


Figure 3.33: Illustration of load distribution of the skin-stringer assembly after buckling (adapted from Collier [15])

Development of the effective skin is observed from the finite element analysis results of the skin-stringer assembly. The nodal forces on the mid plane of the skin panel and the stringer flanges are extracted to observe the changes in the load distribution of the assembly. As it can be seen in Figure 3.34, the nodes on the skin panel and the stringer flanges carry the same load before the buckling. At the moment of buckling, the effective skin forms under the stringer flanges. In Figure 3.35, which shows the load distribution at the buckling, it can be observed that a difference arises in the load carried by the buckled portion of the skin and the stringer flanges. Close to the collapse, the difference becomes more distinctive as shown in Figure 3.36. When the structure reaches its limit of load carrying capacity, it collapses with a sudden loss of

stability as it can be seen in Figure 3.37. Load carried by the buckled portion of the the skin panel almost remains constant from the buckling to the collapse. Corresponding buckling patterns of the figures are given in Figure 3.38 with out-of-plane deformation contours. Also, Figure

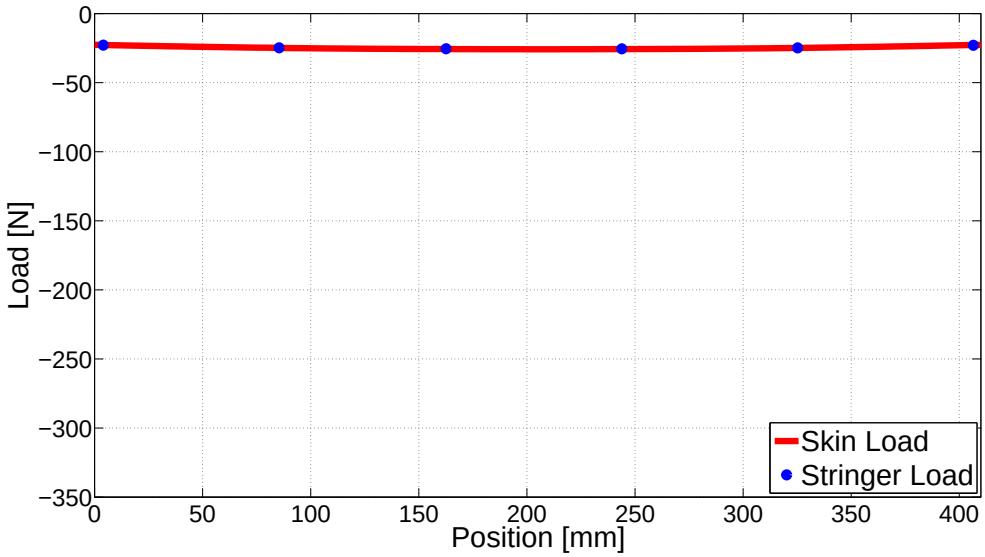


Figure 3.34: Distribution of loads on the skin-stringer assembly before the buckling with an applied load of 15.3 kN

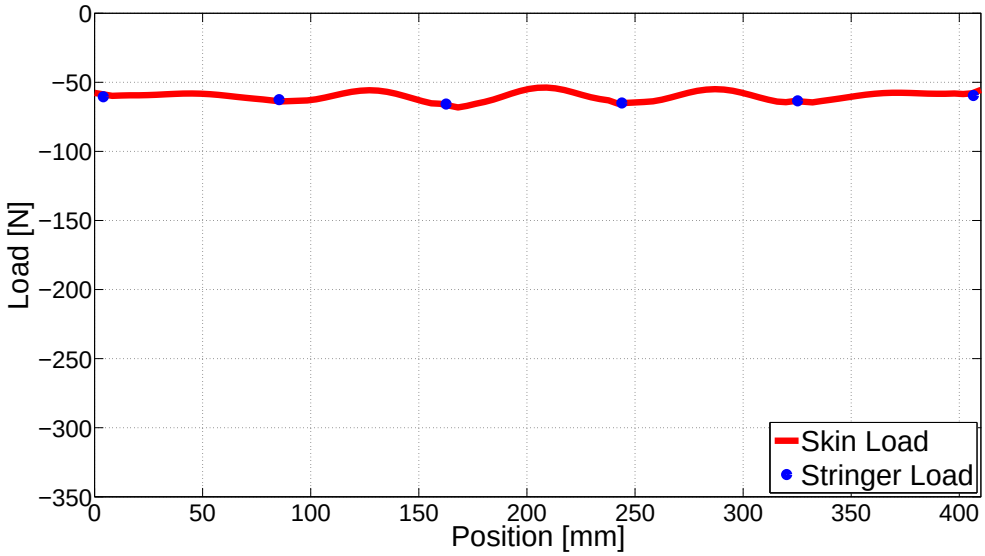


Figure 3.35: Distribution of loads on the skin-stringer assembly after the buckling with an applied load of 38.7 kN

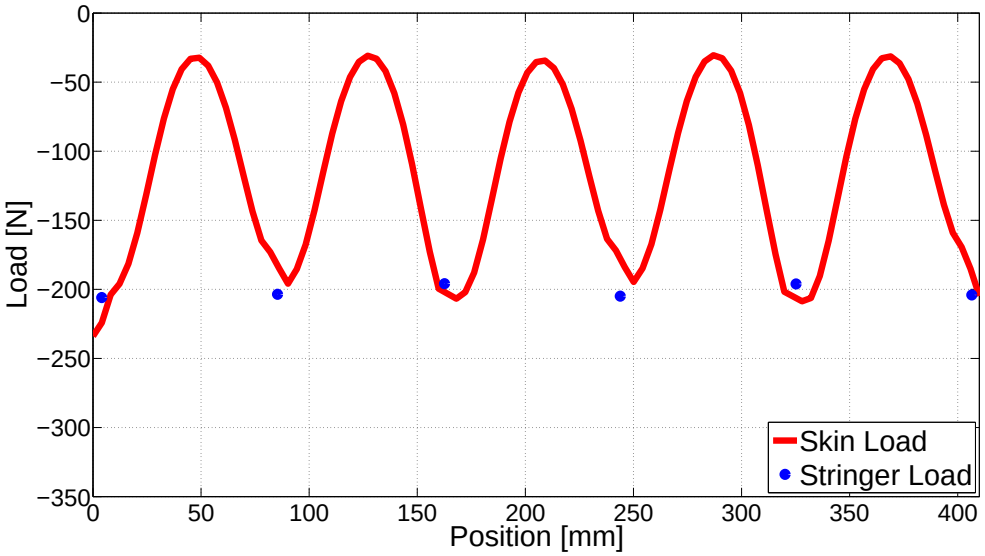


Figure 3.36: Distribution of loads on the skin-stringer assembly close to the collapse with an applied load of 111.9 kN

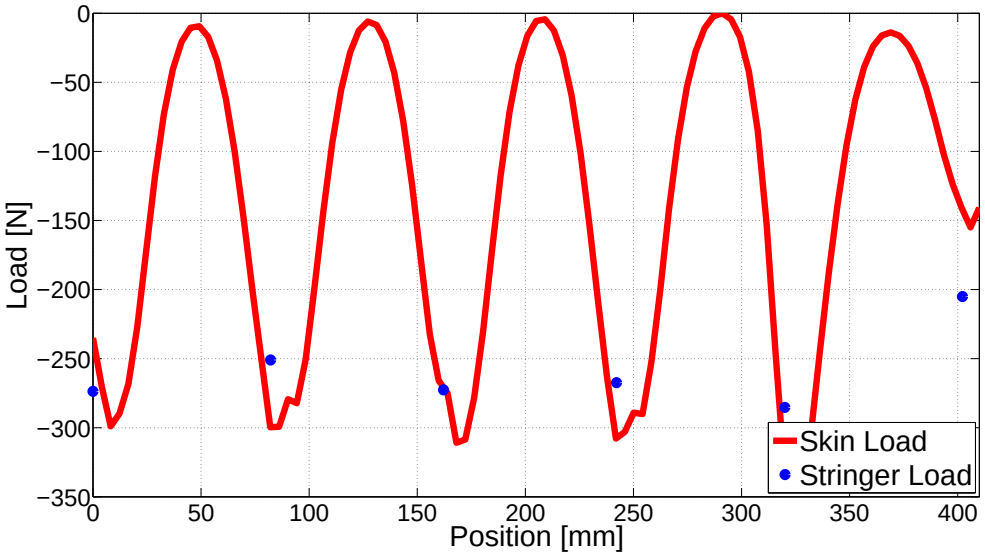


Figure 3.37: Distribution of loads on the skin-stringer assembly after the collapse with an applied load of 110.8 kN

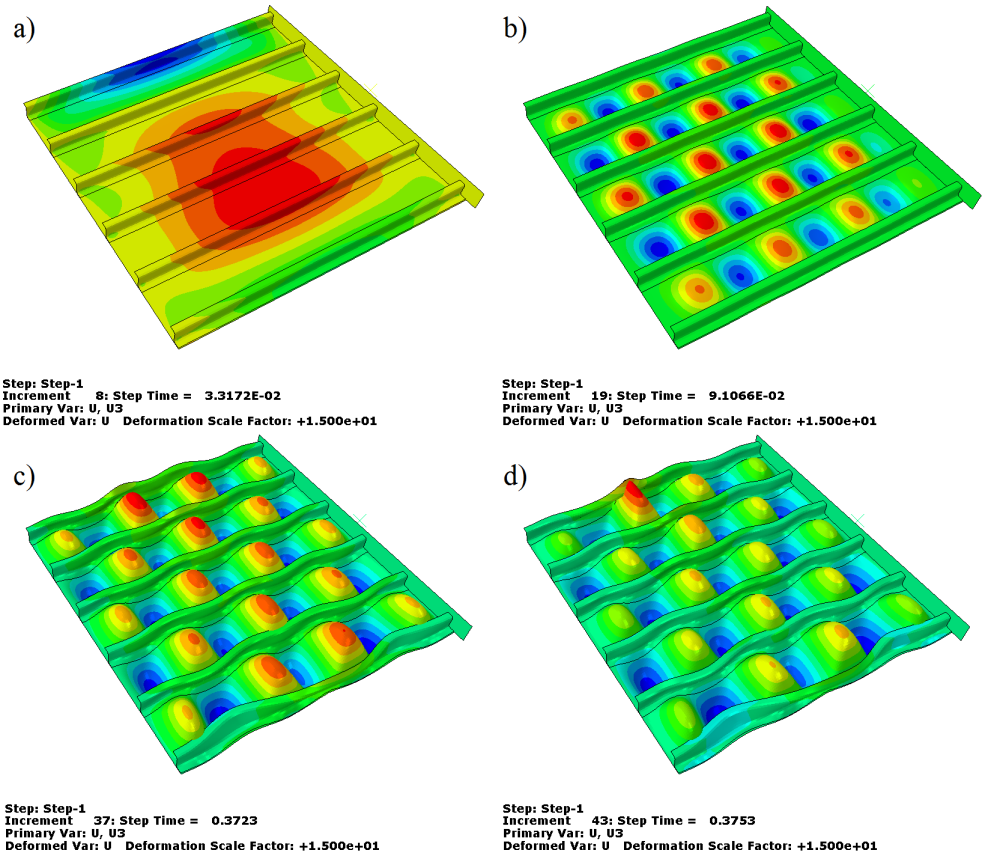


Figure 3.38: Out-of-plane deformations a) before buckling b) after buckling c) close to collapse d) after collapse

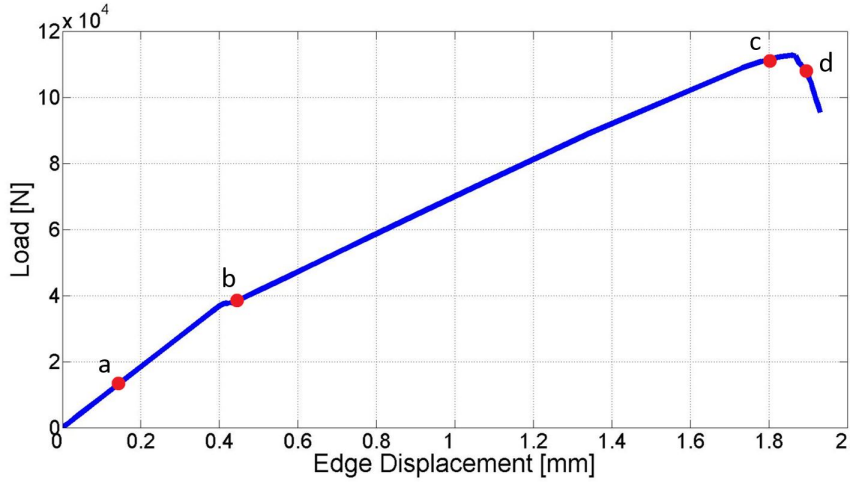


Figure 3.39: Four stages of the assembly a) before buckling b) after buckling c) close to collapse d) after collapse

In Figure 3.40, the parts of the load carried by the skin panel and stringers are shown separately. Note that the skin panel encounters a decrease in stiffness after buckling, while the stringers have the same stiffness until the collapse. The skin panel and the stringers collapse together after the structure loaded up to its load carrying capacity. The loads carried by the skin panel and the stringers are given as a percentage of the total applied load in Figure 3.41.

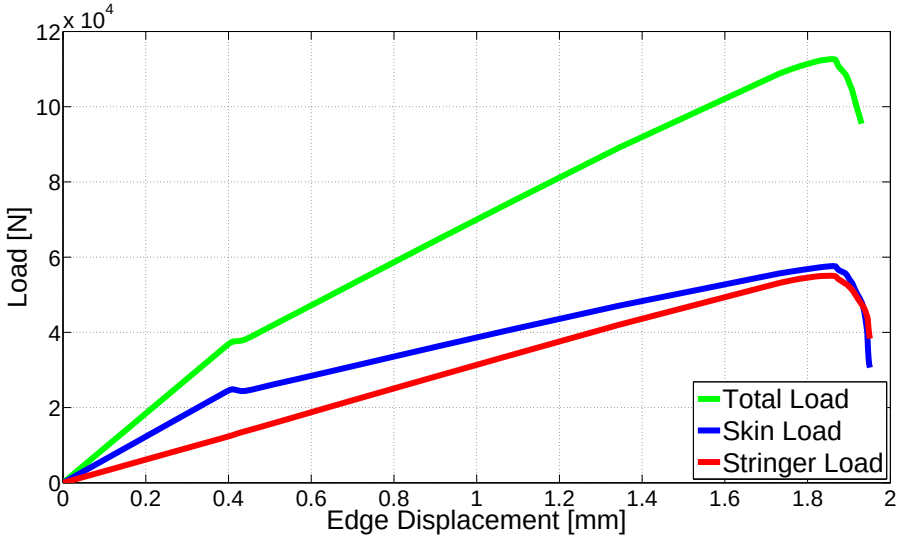


Figure 3.40: Total load, skin load, and stringer load during the analysis

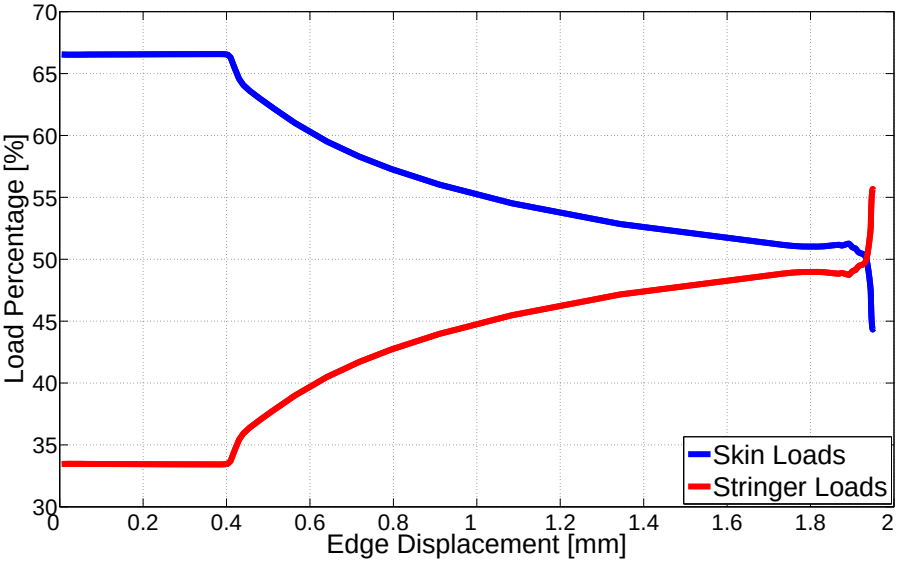


Figure 3.41: Skin load and stringer load during the analysis as percentages of total load

In the figure, initially, the force is distributed equally based on the cross section areas of the skin panel and the stiffeners. The stringers withstand one third of the total applied force since the area of the stringers is one third of the total area. After buckling, the force on the buckled portion of the skin panel is transferred to the stringers until the collapse.

3.2.7 Finite Element Analysis Verification

To verify the model and the analysis results, a study conducted by Wang [22] that involves test results of a similar structure is referred. A new finite element model is created to simulate the test structure. The test model dimensions are assigned as given in Figure 3.42. The fastener spacing is adjusted as discussed in Section 3.2.1.

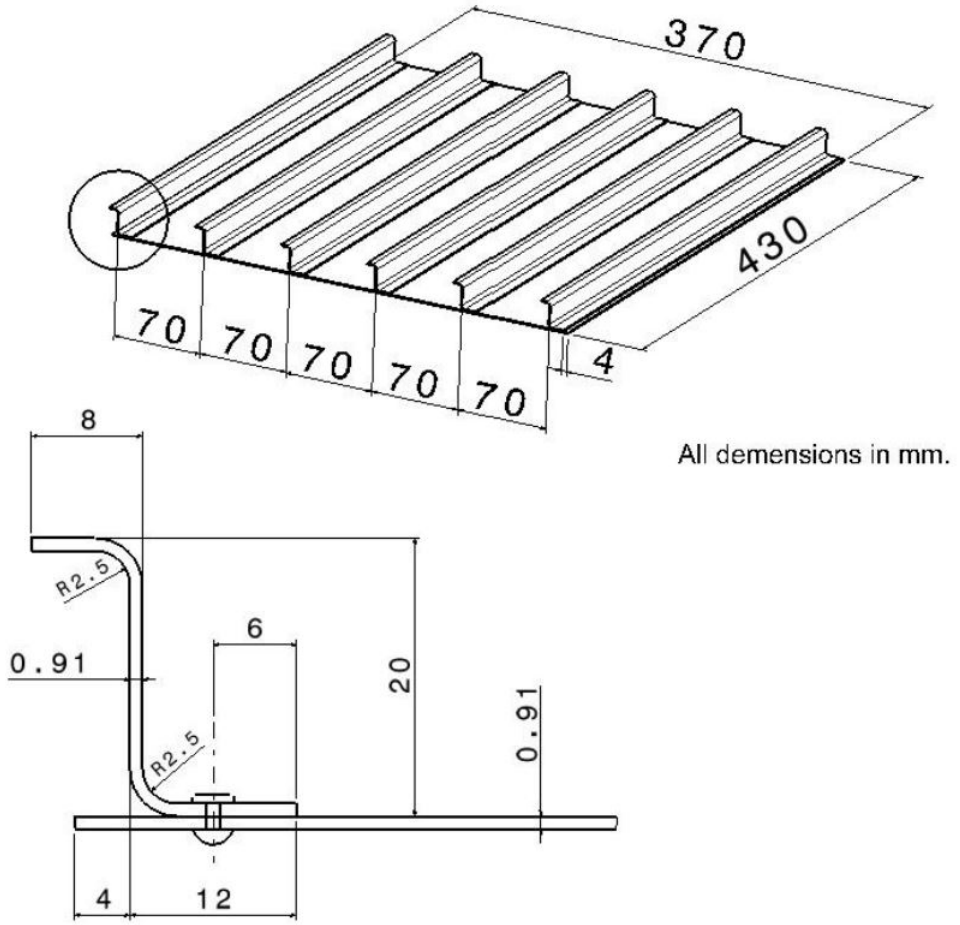


Figure 3.42: Geometry of the test structure [22]

The model is meshed to have three elements in the stringer flange widths as stated in Section 3.2.3. The skin panel and the stringers of the test structure are made of 2014-T6 aluminum. The material properties of the structure are obtained as explained in Section 3.2.2. The material properties of 2014-T6 can be listed as [36]

- Young's Modulus (E) : 73774 MPa
- Poisson's Ratio (υ) : 0.33
- Density (ρ): 2.796 g/cc
- Yield Strength (σ_0): 407 MPa
- Ramberg-Osgood Number (n): 17

The finite element model is analyzed and the results are compared with the test results. The test result shows that the mean collapse load of five specimens is 99.2 kN. The collapse load extracted from the finite element analysis is 98 kN, which is 1.2% different from the collapse load obtained from test results. In Figure 3.43 the result of the FEA is given with the data digitized from the FEA results of Wang [22], that is verified by test results. Therefore, it can be said that the finite element analysis method employed in this study is valid and realistic enough to simulate a real life case.

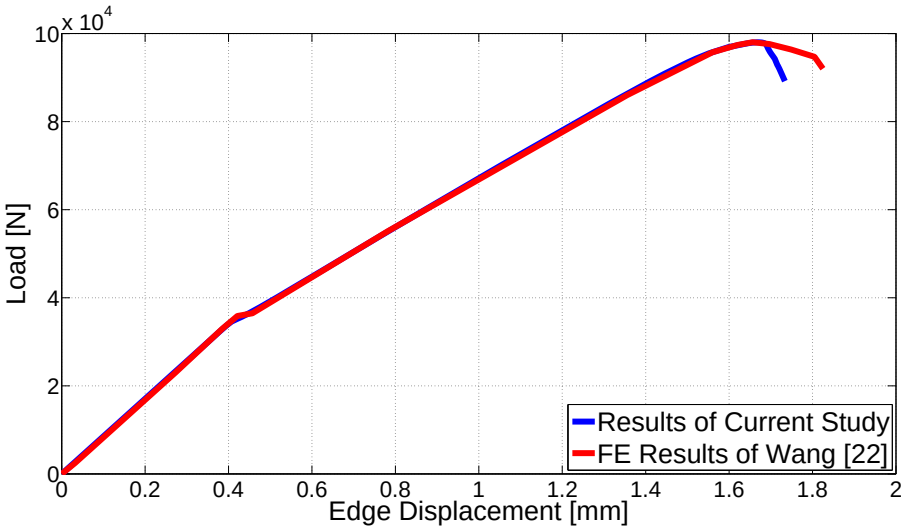


Figure 3.43: Force-displacement curve of the test structure analysis

CHAPTER 4

ARTIFICIAL NEURAL NETWORK

Artificial neural networks (ANN) are tools of pattern recognition and classification, which are originated from neural structure of central nervous system. They are mainly used for data estimation, pattern recognition, and optimization. High computing abilities of ANNs make them suitable for complex problems that are inconvenient to solve with traditional mathematical methods [46]. Haykin [47] describes ANN as a combination of simple processing units that works in parallel to gain knowledge from environment by learning the data gathered from the environment and storing it within the connections of the processing units. The processing units have ability to learn through modifying strength of the interconnections between them. Thus, the learning process is an example of adaptation, that is manipulated by environment [48]. The strength of interconnection is also called as "weight" in artificial version of the neural system. Similarly, the processing units, which are incompetent analogue of biological neurons, are named as "nodes" or "neurons". Feldman and Rojas [49] state that basic structures of a biological neuron are imitated in artificial neural network. The basic structures are considered as the synapses that store information, the cell body that combines the multiple information into a single output, the dendrites that transfer the information from synapses to cell body, and the axon that receives the output. The similarity between a biological neuron and a neuron of an artificial neural network can be seen in Figure 4.1, which is adapted from Gurney [48]. In the artificial neuron, each input is multiplied by a weight and sent to a function that is an imitation of a cell body. The neuron generates a signal if the input is above a threshold. Working together, the artificial neurons create an artificial neural network as illustrated in Figure 4.2.

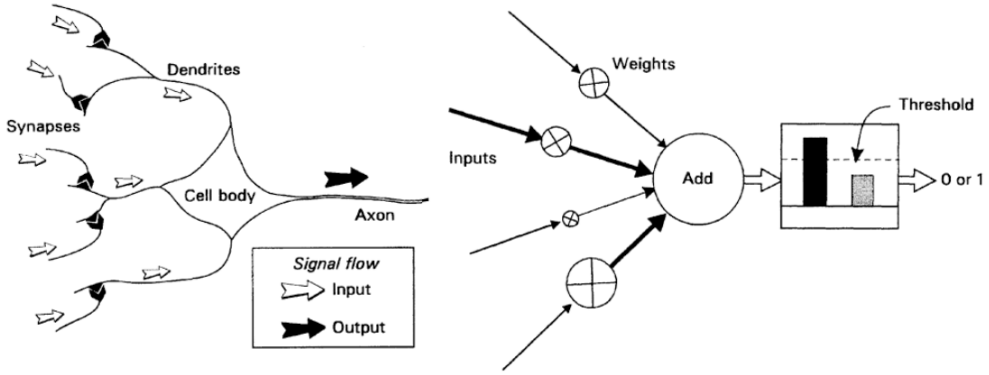


Figure 4.1: A biological neuron and an artificial neuron (adapted from Gurney [48])

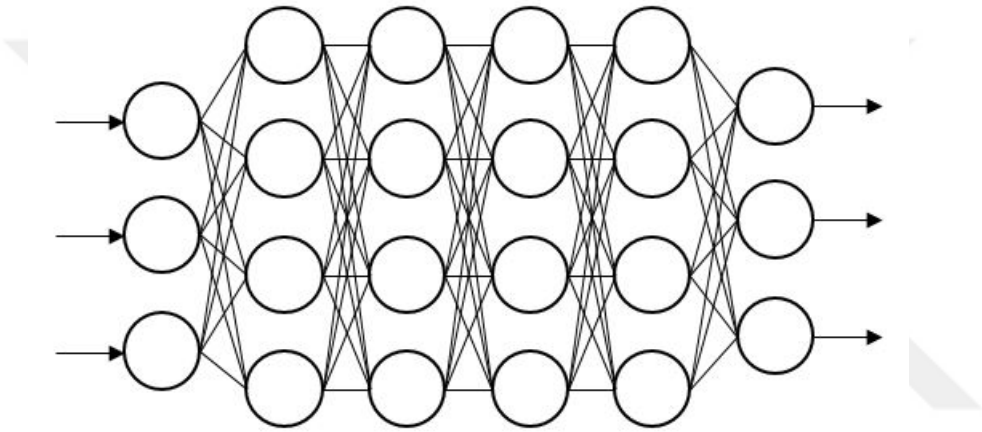


Figure 4.2: Artificial neural network

4.1 Theory of Artificial Neural Network

Artificial neural networks (ANNs) are computational modeling tools to model linear and nonlinear complex systems most traditional statistical approaches. ANNs have characteristic advantages such as being suitable for nonlinear problems, robustness, fault and noise tolerance, and parallel working ability. They can process imprecise and fuzzy information. Thus, they can provide accurate solutions for uncertain data. These capabilities deliver important benefits of excellent data fitting, adaptability and modeling of unlearned data [4].

Combination of two or more artificial neurons creates an artificial neural network. In

ANN, countless interconnections are possible in various ways resulting in countless artificial neural network topologies [50]. Individual artificial neurons in an artificial neural network are grouped to create a layer. Neurons of each layer are connected to the neurons of the next layer. In Figure 4.3, an artificial neural network with 3:2:1 configuration is shown. The ANN is composed of three layers: a three neurons input layer that is used to take three input parameters to the system, a two neurons output layer that returns two outputs as result of the computation process, and a hidden layer. The number of neurons in the input and the output layers are determined by the number of the input and the output parameters. Each neuron is associated with one input or output parameter. Therefore, computational ability of an ANN is determined by the number and the content of hidden layers. Number of neurons in hidden layers is determined by trial and error to adjust the ANN with desired capabilities [51].

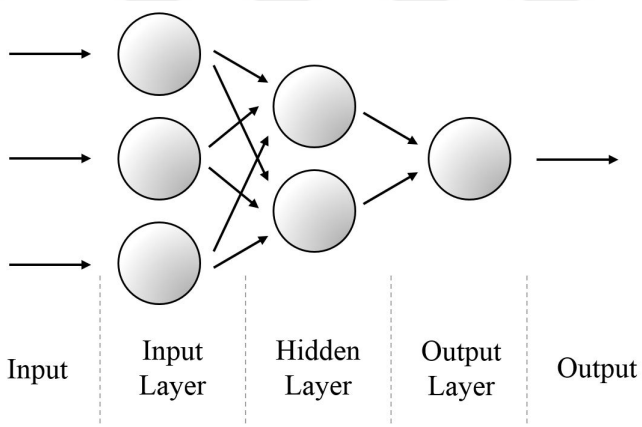


Figure 4.3: Artificial neural network with 3:2:1 configuration

MATLAB Neural Network Toolbox contains four types of neural network tools, which are the *function fitting* to train a neural network to fit a function, the *pattern recognition* to detect and classify an object by tracing its consistency with a defined pattern, the *data clustering* to group data based on similarity, and the *time series analysis* to predict the future behavior of an object according to information from the past. In this study, the function fitting tool is used to approximate the buckling load, the collapse load, and the mass of a skin-stringer assembly based on its geometric and material properties. In order to train the artificial neural network, MATLAB offers several algorithms such as the Bayesian regulation back-propagation, the Resilient back-propagation, etc. However, it is recommended to use the

Levenberg-Marquardt back-propagation algorithm since it is the fastest algorithm among the options that the toolbox offers [52].

Artificial neurons serve as building block in an ANN. Function of a neuron can be summarized as given in Figure 4.4. The input i is multiplied by a weight w and a bias b is added to the result. Result of these operations is processed in a transfer function f to obtain an output from the neuron [50].

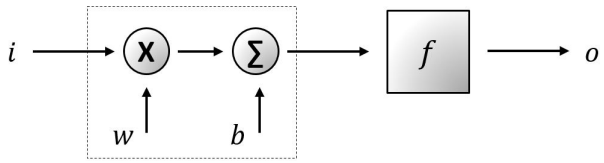


Figure 4.4: Function of an artificial neuron

A mathematical expression to describe the working principle of an artificial neuron can be given as

$$o = f(i\,w + b) \tag{4.1}$$

where o is the output, i is the input, w is the weight, b is the bias, and f is the transfer function. The weight and the bias are determined for each input during training process of the ANN. Therefore, the transfer function constitutes major characteristics of the neuron. Generally, options for the transfer function are a *step function*, a *linear function*, and a *sigmoid function*. The step function is a function that can only return one or zero and used in neurons that are involved in classification problems. Linear function is used in neurons that are utilized in linear transfer processes. Commonly, the sigmoid function is used in neurons of function fitting ANNs. It provides outputs between -1 and 1 [52].

4.2 Generation of Artificial Neural Network

As stated in previous sections, ANN requires a data set to be trained. Size of the data set is a vital parameter in the training process. It should be large enough to contain adequate information about a fitting function but also it should be small enough to offer acceptable time to create the data set and to train the ANN. With this

consideration in mind, a data set that includes 1440 FE analysis inputs and outputs are created. A script is prepared in Python 2.7 for this purpose. The script is utilized to create 1440 skin-stringer assembly finite element models and analyze them in accordance with the method described in this study.

The script is written in a way that allows the following parameters to be determined by the user,

- Skin panel thickness
- Stringer thickness
- Stringer height
- Stringer inner flange length
- Stringer outer flange length
- Stringer section inner radius
- Skin panel material
- Stringer material
- Skin panel length
- Skin panel width
- Distance between stringers
- Fastener diameter

The number of stringers is calculated by considering the skin width and the distance between stringers. Using the skin panel width and the distance between stringers given by the user, the maximum number or stringers is determined. The remaining space of the panel is distributed evenly to each side of the panel.

To minimize the computation time and sources, the script is not used in its maximum capacity. Parameters that are considered to have marginal effect on the buckling and the post-buckling properties are fixed to the values given in Table 4.1.

Table 4.1: Geometric parameters that are fixed in all models

Stringer section inner radius [mm]	2.5
Skin panel width [mm]	350
Fastener diameter [mm]	3.2

The values are chosen as average design values. The stringer section radius has an insignificant effect on sectional properties, such as inertia and area, compared to the flange lengths and the stringer height. The distance between stringers is a more critical design parameter than the panel width since the buckling occurs between the stringers. The fastener diameter is not a parameter to consider during skin-stringer design but it should be chosen from standard fastener types to be suitable to skin-stringer load carrying capacity. The following values of the parameters are used for the FE analyses. Consequently, the total number of analyses is calculated as 1440 with the values given in Table 4.2.

Table 4.2: Input parameters for the data set

Skin panel thickness [mm]	[1.00, 1.27, 1.42, 1.60]
Stringer thickness [mm]	[1.00, 1.27, 1.42, 1.60]
Stringer height [mm]	[20, 30]
Stringer inner flange length [mm]	[10, 15]
Stringer outer flange length [mm]	[10, 20]
Skin panel material	[AL2024 T3 clad, AL2024 T42 clad]
Stringer material	[AL2024 T3 clad, AL2024 T42 clad]
Skin panel length [mm]	[300, 350, 400]
Distance between stringers [mm]	[60, 70, 80]

In order to minimize the number of analyses, total number of values for each parameter is kept as low as possible. The skin and the stringer thicknesses have remarkable effects on the buckling load and the collapse load. Thus, relatively wide ranges of values are considered for these parameters. The stringer section properties are restricted to two values for each parameter. Material properties of the aluminum 2024 T42 is obtained as described in Section 3.2.2 and the detailed properties can be found in Appendix A.

The script composes a text file of input parameters before creating a model. Then it creates and analyze the model, and extracts the buckling load, the collapse load, and the mass of the analyzed structure to create another text file that includes the extracted data. Text files with the input and result data are processed by another Python 2.7 script to collect the data in two separate Microsoft Excel files for the input parameters and the output parameters. The Excel files are exported to MATLAB as two matrices. These two matrices are used to train the ANN.

In order to present the relation between the input parameters and desired output parameters of a skin-stringer assembly, function fitting ANN is defined. The ANN is configured to obtain the best performance before training with the created matrices. First parameter to configure in a setting of ANN is the number of hidden layers. Majority of the problems can be solved with a single hidden layer. Furthermore, the number of hidden layers significantly affects the duration of the training process. Therefore, single hidden layer is chosen for the ANN. Another important parameter is the number of artificial neurons in the hidden layer. As the number of neurons are increased, the function fitted by the ANN satisfies the data set more accurately. However, for very high number of neurons over-fitting may occur. Over-fitting the function results in inaccuracy for data points that are not in the training data set. Thus, the number of neurons must be selected carefully. In order to reach the optimum number of neurons, several configurations are tried and it is seen the ANN configuration with 1 hidden layer with 48 artificial neurons provides the best performance. The final configuration of the ANN is shown in Figure 4.5. In the figure, there are 16 inputs, 48 artificial neurons in the hidden layer, 3 neurons in the output layers, and 3 outputs. It should be noted that the output layers are assigned to each output by MATLAB, thus, they are not a configuration parameter. Hence, the ANN is created with 16:48:3 configuration.

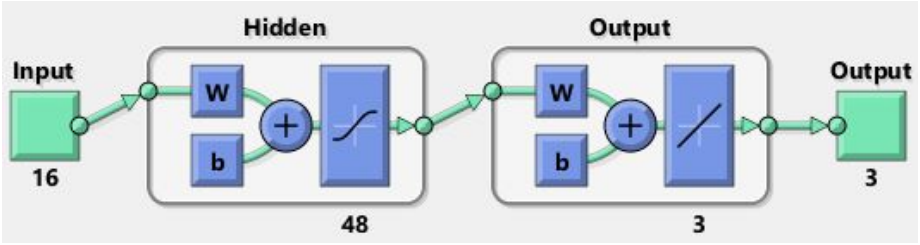


Figure 4.5: The final configuration of the ANN

Training of the ANN requires to divide the data set into three components that are called as the training set, the validation set, and the testing set. MATLAB divides the data set as 70% for training, 15% for validation, and 15% for testing by default [52]. The purpose of the training set is to determine weights and biases of the neurons. After each iteration of the training process, the validation set is employed to check the accuracy. The performance of the obtained ANN configuration is tested by the testing set. Thus, the testing set is not involved firsthand in the training process. In order not to deprive of many data points on training set, data set division is changed as 90% for the training set, 5% for the validation set, and 5% for the testing set. Sample points on data set are randomly assigned to each set. Therefore, training the same ANN with the same data set can result in different ANN performance. To obtain an ANN with better performance, the training process is repeated several times with the same data set.

The training performance is measured as the *Mean Squared Error* (MSE). The squared error measures the squared distance of a data point from the fitted line. Average of the squared distance values gives the MSE. Termination of the training process can be decided by several criteria. One of them is the number of iteration that is referred as the *Epoch*. The epoch limit is set to 5000 to prevent early terminations. Another termination criterion is the maximum number of iterations without any improvement, referred as the *Max Fail*. The Max Fail limit is assigned as 500.

4.3 Results of Artificial Neural Network

In determination of the ANN performance, the value of the mean squared error is not appropriate in this study. The reason is that the collapse load, the buckling load and the mass values are not on the same scale in terms of magnitude. The skin-stringer assembly analyzed in Section 3.2 has a buckling load of 37525 N, collapse load of 112653 N and mass of 775 grams. Numerical differences in these results may mislead the representation of the ANN performance. However, the decrease in the mean square error seen in Figure 4.6 reveals that the ANN performance increases over iterations.

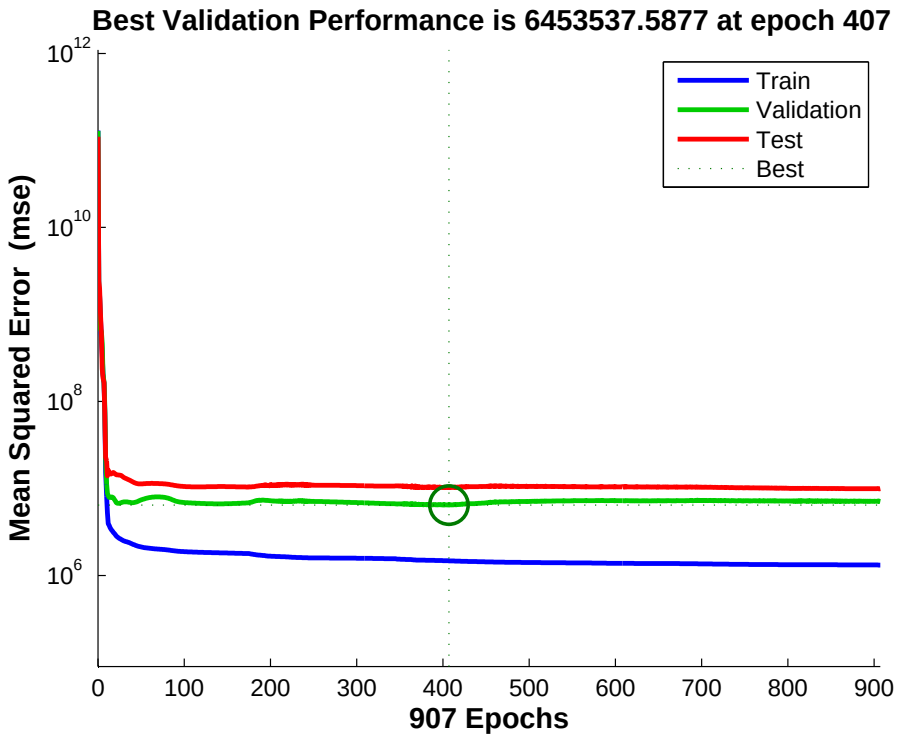


Figure 4.6: Performance plot of the ANN

To express the ANN performance with a numerical expression, input parameters of 1440 analysis are processed with the ANN and their results are compared with the FEA results. It is seen that, 95 analyses (6.6%) show 5% or more difference in any of the three outputs (the buckling load, the collapse load and the mass) in comparison to the FEM results. Only 8 analyses (0.56%) exhibit 10% or more difference. The maximum error is found to be 15.45% for the buckling load, 6.02% for the collapse load and 6.97% for the mass. Therefore, it can be said that the ANN performance is promising.

ANN performance can also be shown with the regression lines given in Figure 4.7. In the plots, vertical axis shows the ANN outputs for the data sample, and the horizontal axis shows the FEA results that are used in training of the ANN. The dashed lines in the figures represent the perfect fit and the blue lines are the fits that are created by the training process. It is seen that the blue lines are very close to the dashed lines in each figure.

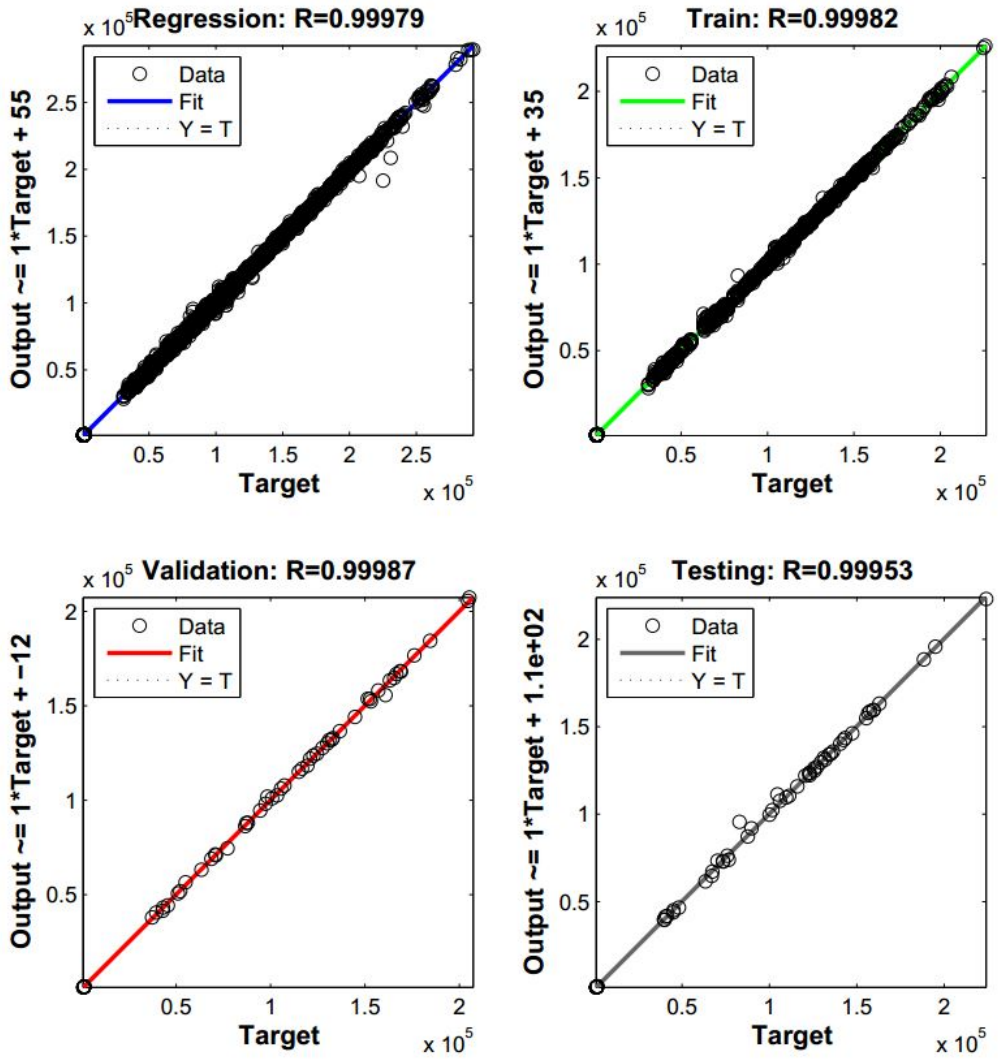


Figure 4.7: Regression plots of the ANN for overall, training, validation and testing samples

Figure 4.8 shows the cause for termination of the training. It is seen that the ANN performance does not increase after 407 iterations. Thus, the validation fails 500th times at the iteration 907.

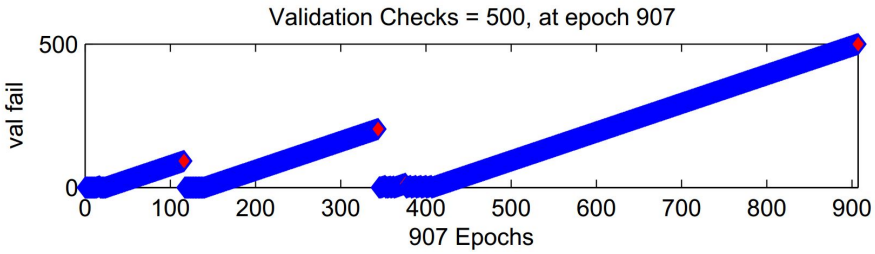


Figure 4.8: Number of validation fails with respect to number of training iterations

CHAPTER 5

RESULTS AND EVALUATION

The artificial neural network (ANN) that is created and trained in this study is verified by numerical tests. The tests are conducted with the data points that are both included and excluded in the training data set.

In order to obtain a result from the ANN, an array of input parameters is given as an input argument with the following order,

- Stringer height
- Stringer inner radius
- Stringer outer flange length
- Skin panel length
- Stringer inner flange length
- Skin panel ultimate strength
- Skin panel yield strength
- Skin panel Young's modulus
- Skin panel thickness
- Skin panel Ramberg-Osgood number
- Stringer ultimate strength
- Stringer yield strength

- Stringer Young's modulus
- Stringer thickness
- Stringer Ramberg-Osgood number
- Distance between stringers

The geometric parameter listed above were presented previously in Figure 3.17 and Figure 3.18.

An example of use of the developed ANN is as follows

net([20,2.5,10,300,10,407,241,65500,1,17,407,241,65500,1,17,60]')

Then, the result is displayed as

1.0e+04 *
5.3180
9.8104
0.0925

The result given above corresponds to the buckling load of 53.18 kN, the collapse load of 98.1 kN and the mass of 925 grams.

5.1 Comparison with Finite Element Analysis

The ANN is tested with the training data set during the training process. The results of the test are given in Table 5.1. The table shows the number of outputs that exhibits 5% and 10% difference to the finite element analysis (FEA) results. According to the table, 80 of the 1440 ANN outputs show a difference between 5% to 10% in the buckling load. Only 8 of the 1440 ANN outputs have a difference greater than 10%. Maximum difference is found to be 15.45% in the buckling load results, 6.02% in the collapse load results, and 6.97% in the mass results. These numbers decrease for the collapse load to 1 and 0 respectively. The total number of outputs that shows difference greater than 10% in any of the outputs is found to be 8. On order to test the ANN, additional finite element analyses are conducted. The input parameters of the additional analyses are given to ANN. The results obtained with both methods

are compared. The input parameters of the additional analyses are given in Table 5.2. In this table, σ_{ult} is the ultimate strength, σ_0 is the yield strength, E is the Young's modulus, and n is the Ramberg-Osgood number.

Table 5.1: Deviations of the ANN results from the FEA results

Error	$< 5\%$	$5\% - 10\%$	$> 10\%$
Buckling Load	1352	80	8
Collapse Load	1439	1	0
Mass	1432	8	0

Table 5.2: Input parameters for the additional analyses

FEA	1	2	3	4	5	6	7
Stringer thickness [mm]	1.3	1.27	1.27	1.27	1.5	2	2
Stringer height [mm]	20	25	25	20	20	20	20
Inner flange length [mm]	10	10	10	10	10	10	10
Outer flange length [mm]	10	12	12	10	10	10	10
Stringer inner radius [mm]	2.5	2.5	2.5	2.5	2.5	2.5	2.5
Skin σ_{ult} [MPa]	421	421	421	421	421	421	407
Skin σ_0 [MPa]	310	310	310	310	310	310	241
Skin E [GPa]	65.5	65.5	65.5	65.5	65.5	65.5	65.5
Skin n [-]	15	15	15	15	15	15	17
Skin length [mm]	350	350	350	370	450	350	350
Skin thickness [mm]	0.8	1	1	1	1.3	1	1
Stringer σ_{ult} [MPa]	421	421	407	421	421	421	407
Stringer σ_0 [MPa]	310	310	241	310	310	310	241
Stringer E [GPa]	65.5	65.5	65.5	65.5	65.5	65.5	65.5
Stringer n [-]	15	15	17	15	15	15	17
Stringer distance [mm]	70	70	70	65	70	70	70

In Table 5.2 the input parameters for the additional FE analyses are given. Table 5.3 shows the FE results, the ANN results and relative errors obtained for these inputs. The maximum difference of ANN results to FEM results is found as 2.36%.

Table 5.3: Results of the additional analyses

	Buckling Load [kN]			Collapse Load [kN]			Mass [g]		
FEA	ANN	FEA	% Err	ANN	FEA	% Err	ANN	FEA	% Err
1	53.18	52.90	0.53	98.10	97.34	0.78	924.66	931.89	0.78
2	66.00	65.03	1.50	106.12	105.58	0.51	924.10	931.89	0.84
3	53.12	52.90	0.43	108.68	109.94	1.15	909.94	931.89	2.36
4	64.85	65.03	0.26	119.05	119.25	0.16	948.98	931.89	1.83
5	36.68	37.29	1.63	84.14	83.77	0.44	893.02	873.45	2.24
6	38.08	37.29	2.11	91.08	92.72	1.77	859.66	873.45	1.58
7	36.51	37.29	2.11	92.38	91.56	0.90	868.26	873.45	0.59

5.2 Design Curves

Comparisons performed in this chapter revealed that the ANN can be used in nonlinear structural analysis of skin-stringer assemblies that require fast results. For example, design curves that includes many analysis results can be quickly generated with the ANN. Figure 5.1 is given as an example design curve obtained with the generated ANN. In the figure, change in the collapse load and the buckling load with respect to the skin thickness of a skin-stringer assembly can be observed. Similarly, Figure 5.2 shows the collapse load and the buckling load change with respect to skin thickness. It can be seen from the figures that both the collapse load and the buckling load increase with increasing thickness. Also, the skin thickness is found to be more effective than the stringer thickness in determination of the buckling load, while the stringer thickness is more effective on the collapse load.

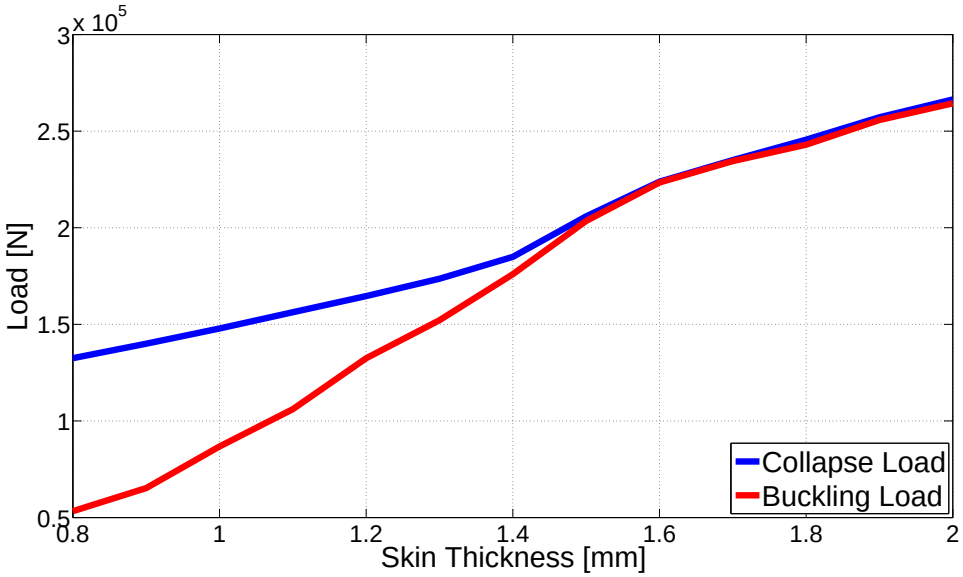


Figure 5.1: The collapse load and the buckling load with respect to skin thickness

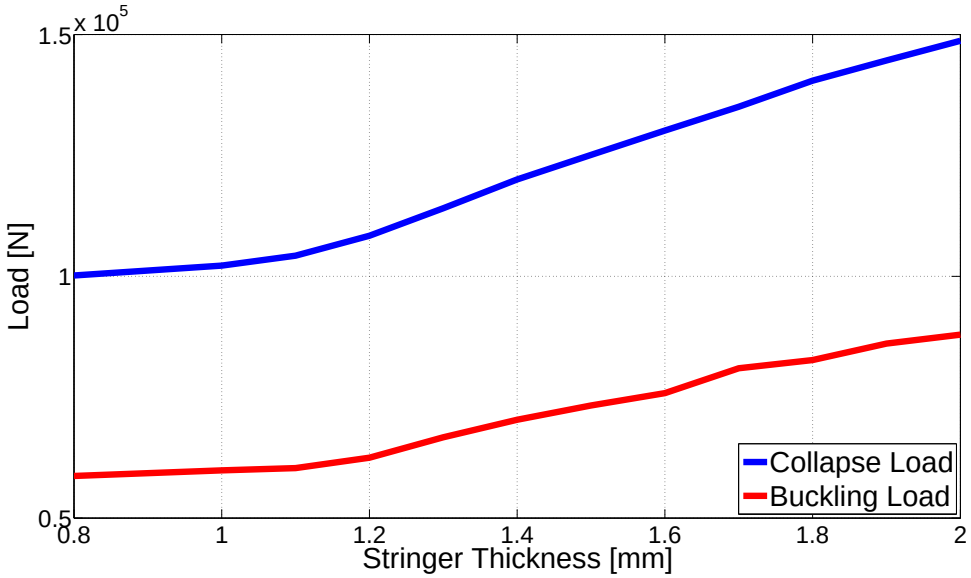


Figure 5.2: The collapse load and the buckling load with respect to stringer thickness

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

An artificial neural network based analysis method is developed in this study. It is aimed to decrease analysis time of structures that are composed of skin-stringer assemblies by using ANN. The capability of ANN on modeling of nonlinear problems is utilized to develop a structural analysis method. Therefore, the high computing time that is required for a nonlinear analysis is eliminated with the use of the ANN. It was shown that the use of ANN to conduct nonlinear structural analysis of skin-stringer assemblies can provide fast and accurate solutions.

To train the ANN, a database that includes input and target data points is required. This database is created with multiple skin-stringer assembly input parameters and nonlinear finite element analysis outputs of these assemblies. Input parameters are chosen to cover the important parameters that affect the buckling and post-buckling properties of skin-stringer assemblies. 1440 skin-stringer assembly finite element models are created by assigning a range of values to the input parameters. Then, nonlinear analyses of those assemblies are conducted and the results that are critical for the optimization process are extracted. These results are chosen as the buckling load, the collapse load, and the mass of the structure. A data set is created with the input parameters and corresponding analysis results. Finally, the ANN is trained with the created data set and performance of the ANN is tested.

Performance of the ANN is tested with two different data sets. The first test is conducted with the data set that is used to train the ANN. Deviation of the ANN

results from the FEA results are calculated and it was shown that the deviation is low for the data points in training set, thus, the ANN results are consistent with the analysis results. To conduct the second test, additional data samples are created by input parameters that are not in the training data set. Results of ANN and FEA for the additional data samples are collected and compared. The comparison proved that the ANN performs well also for the data points that are not included in the training data set. Therefore, the ANN can be used for structural analysis of the skin-stringer assemblies as long as the input parameters stay in the limits of the input parameter set.

6.2 Future Works

The analysis method for skin-stringer assemblies introduced in this study is applied for a single loading type. However, the method can also be used for different loading types such as bending and shear. The procedure explained in this study can be applied to the models of different loading types to broaden the scope of the ANN. Similarly, the analysis method can be applied to skin-stringers that are composed of different materials. The models in this study are created with commonly used aluminum material properties. However, the method can be applied to the models of composite materials, which are also frequently used in aerospace industry. Also, additional input parameters can be studied and used in training of the ANN. To achieve this, additional finite element models are required. Then, a new data set can be created with the input parameters and corresponding FEA results of the additional finite element models. An ANN that is trained with the new data set can provide desired results for the new input parameters. The accuracy of the ANN results depends on the range of values and the size of the data set that the ANN is trained with. Therefore, bigger data sets with larger range of values in training set can be tested. By enlarging the data set created in this study, accuracy of the ANN can be improved.

REFERENCES

[1] T. H. G. Megson. *Aircraft Structures for Engineering Students*. Number 1 in Elsevier aerospace engineering series. Elsevier Science, 2014.

[2] M. C. Y. Niu. *Airframe Structural Design : Practical Design Information And Data On Aircraft Structures*. Airframe book series. Adaso Adastra Engineering Center, 1999.

[3] S. Gudmundsson. *General Aviation Aircraft Design - Applied Methods and Procedures*. Elsevier Science, 2014.

[4] I. A. Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, 2000.

[5] E. F. Bruhn. *Analysis and design of Flight Vehicle Structures*. S.R. Jacobs, 1973.

[6] C. Niu and M.C.Y. Niu. *Airframe Stress Analysis and Sizing*. Airframe book series. Conmilit Press, 1999.

[7] J.C. Flabel. *Practical Stress Analysis for Design Engineers: Design and Analysis of Aerospace Vehicle Structures*. Lake City Publishing Company, 1997.

[8] G. George and B. Herbert. Handbook of structural stability part i: Buckling of flat plates. TN 3781, NACA, National Advisory Committee for Aeronautics; Washington, DC, United States, July 1957.

[9] B. L. O. Edlund. Buckling of metallic shells: Buckling and postbuckling behaviour of isotropic shells, especially cylinders. *Structural Control and Health Monitoring*, 14(4):693–713, 2007.

[10] M.M. Alinia, H.R. Habashi, and A. Khorram. Nonlinearity in the postbuckling behaviour of thin steel shear panels. *Thin-Walled Structures*, 47(4):412–420, 2009.

[11] M. Muameleci. Linear and nonlinear buckling analyses of plates using the finite element method. Master's Thesis, Linköping University, Linköping, Sweden, June 2014.

[12] K.K. Choong and E. Ramm. Simulation of buckling process of shells by using the finite element method. *Thin-Walled Structures*, 31(1):39–72, 1998.

[13] P. S. Ferreira and F. B.E. Virtuoso. Semi-analytical models for the post-buckling analysis and ultimate strength prediction of isotropic and orthotropic plates under uniaxial compression with the unloaded edges free from stresses. *Thin-Walled Structures*, 82:82–94, 2014.

[14] M. Amani, B.L.O. Edlund, and M.M. Alinia. Buckling and postbuckling behavior of unstiffened slender curved plates under uniform shear. *Thin-Walled Structures*, 49(8):1017–1031, 2011.

[15] C. Collier, P. Yarrington, P. Gustafson, and B. Bednarcyk. Local Post Buckling: An Efficient Analysis Approach for Industry Use. In *50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Palm Springs, California, 4-7 May 2009.

[16] N. Aswini, H.-G. Reimerdes, and N.K. Gupta. Postbuckling responses of the panels subjected to combined compression and shear stresses. *Procedia Engineering*, 173:25–32, 2017. Plasticity and Impact Mechanics.

[17] S. Zhang and I. Khan. Buckling and ultimate capability of plates and stiffened panels in axial compression. *Marine Structures*, 22(4):791–808, 2009.

[18] D.G. Stamatelos, G.N. Labeas, and K.I. Tserpes. Analytical calculation of local buckling and post-buckling behavior of isotropic and orthotropic stiffened panels. *Thin-Walled Structures*, 49(3):422–430, 2011.

[19] C. Lynch, A. Murphy, M. Price, and A. Gibson. The computational post buckling analysis of fuselage stiffened panels loaded in compression. *Thin-Walled Structures*, 42(10):1445–1464, 2004.

[20] A. Murphy, M. Price, C. Lynch, and A. Gibson. The computational post-buckling analysis of fuselage stiffened panels loaded in shear. *Thin-Walled Structures*, 43(9):1455–1474, 2005.

[21] J. Campbell, L. Hetey, and R. Vignjevic. Non-linear idealisation error analysis of a metallic stiffened panel loaded in compression. *Thin-Walled Structures*, 54:44–53, 2012.

[22] Y. Wang. Virtual testing of post-buckling behaviour of metallic stiffened panel. Master's Thesis, Cranfield University, Cranfield, England, 2011.

[23] J.F. Caseiro, R.A.F. Valente, A. Andrade-Campos, and J.W. Yoon. Elasto-plastic buckling of integrally stiffened panels (isp): An optimization approach for the design of cross-section profiles. *Thin-Walled Structures*, 49(7):864–873, 2011.

[24] C. Bisagni and L. Lanzi. Post-buckling optimisation of composite stiffened panels using neural networks. *Composite Structures*, 58(2):237–247, 2002.

[25] M. R. Sheidaii and R. Bahraminejad. Evaluation of compression member buckling and post-buckling behavior using artificial neural network. *Journal of Constructional Steel Research*, 70:71–77, 2012.

[26] A. Yıldırım and A. Kayran. Development of bolted flange design tool based on finite element analysis and artificial neural network. Master's Thesis, Middle East Technical University, Ankara, September 2015.

[27] H. M. Gomes, A. M. Awruch, and P. A. M. Lopes. Reliability based optimization of laminated composite structures using genetic algorithms and artificial neural networks. *Structural Safety*, 33(3):186–195, 2011.

[28] J. Gajewski, P. Golewski, and T. Sadowski. Geometry optimization of a thin-walled element for an air structure using hybrid system integrating artificial neural network and finite element method. *Composite Structures*, 159:589–599, 2017.

[29] Z. Sadovský and C. G. Soares. Artificial neural network model of the strength of thin rectangular plates with weld induced initial imperfections. *Reliability Engineering & System Safety*, 96(6):713–717, 2011. ESREL 2009 Special Issue.

[30] L. Lanzi and V. Giavotto. Post-buckling optimization of composite stiffened panels: Computations and experiments. *Composite Structures*, 73(2):208–220, 2006. International Conference on Buckling and Postbuckling Behavior of Composite Laminated Shell Structures.

[31] U. K. Mallela and A. Upadhyay. Buckling load prediction of laminated composite stiffened panels subjected to in-plane shear using artificial neural networks. *Thin-Walled Structures*, 102:158–164, 2016.

[32] D. Bushnell. *Computerized buckling analysis of shells*. Mechanics of Elastic Stability. Springer Netherlands, 1985.

[33] D. Bushnell and B. Bushnell. Shell buckling. https://shellbuckling.com, 2015.

[34] Dassault Systèmes. *Abaqus 6.13 Documentation*, April 2013.

[35] A. Shahbazian and Y. C. Wang. Calculating the global buckling resistance of thin-walled steel members with uniform and non-uniform elevated temperatures under axial compression. *Thin-Walled Structures*, 49:1415–1428, May 2011.

[36] United States Federal Aviation Administration and Battelle Memorial Institute. Columbus Laboratories. MMPDS-03: Metallic Materials Properties Development and Standardization Federal Aviation Administration, Virginia, United States, 2006.

[37] A. Aalberg, M. Langseth, and P.K. Larsen. Stiffened aluminium panels subjected to axial compression. *Thin-Walled Structures*, 39(10):861–885, 2001.

[38] A Ambri and R Kaur. Spars and stringers- function and designing. *International Journal of Aerospace and Mechanical Engineering*, 1(1):58–61, 2014.

[39] S. Grihon, M. Samuelides, A. Merval, A. Remouchamps, M. Bruyneel, B. Colson, and K. Hertel. Fuselage Structure Optimisation. In E. Kesseler and M. D. Guenov, editors, *Progress in Astronautics and Aeronautics - Advances in Collaborative Civil Aeronautical Multidisciplinary Design Optimization*, pages 193–247. American Institute of Aeronautics and Astronautics, 2009.

[40] R.M. Rivello. *Theory and analysis of flight structures*. McGraw-Hill series in aeronautical and aerospace engineering. McGraw-Hill, 1969.

[41] Federal Aviation Administration. Aviation maintenance technician handbook, US: Department of Transportation, Oklahoma City, United States 2008.

[42] W. Ramberg and W. R. Osgood. Description of stress-strain curves by three paramaters. TN 902, NACA, National Advisory Committee for Aeronautics; Washington, DC, United States, July 1943.

[43] D. Roylance. Stress-strain curves. 3.11 Mechanics of Materials. Fall 1999. Massachusetts Institute of Technology: MIT OpenCourseWare, $https://ocw.mit.edu/courses/materials-science-and-engineering/3-11-mechanics-of-materials-fall-1999/modules/MIT3_11F99_ss.pdf$ License: Creative Commons BY-NC-SA.

[44] M. Mert and A. Kayran. Post-buckling load redistribution of stiffened panels in aircraft wingbox structures. In *8th Ankara International Aerospace Conference*, METU, Ankara, Turkey, 10 - 12 September 2015. AIAC-2015-116.

[45] S. Veen. Stiffened panels in compression: Redirecting loads toward high-strength stiffeners. In Daniel Coatta, editor, *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, 19 - 22 April 2004. AIAA 2004-2051.

[46] S. Gholizadeh, A. Pirmoz, and R. Attarnejad. Assessment of load carrying capacity of castellated steel beams by neural networks. *Journal of Constructional Steel Research*, 67(5):770–779, 2011.

[47] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[48] K. Gurney. *An Introduction to Neural Networks*. Taylor & Francis, 2003.

[49] J. Feldman and R. Rojas. *Neural Networks: A Systematic Introduction*. Springer Berlin Heidelberg, 1996.

[50] A. Krenker, A. Kos, and J. Bešter. *Introduction to the Artificial Neural Networks*. INTECH Open Access Publisher, 2011.

[51] H. M. Gomes and A. M. Awruch. Comparison of response surface and neural network with other methods for structural reliability analysis. *Structural Safety*, 26(1):49–67, 2004.

[52] M.H. Beale, M.T. Hagan, and H.B. Demuth. *Neural Network Toolbox: For Use with MATLAB;[User's Guide]*. MathWorks, 1992.

[53] N. Vasios. Nonlinear analysis of structures: The arc length method. Master's Thesis, Harvard University, Massachusetts, United States, 2015.

[54] M. Życzkowski. Post-buckling analysis of non-prismatic columns under general behaviour of loading. *International Journal of Non-Linear Mechanics*, 40(4):445–463, 2005.

[55] E. Riks. An incremental approach to the solution of snapping and buckling problems. *International Journal of Solids and Structures*, 15(7):529–551, 1979.

[56] M.A. Crisfield. A fast incremental/iterative solution procedure that handles "snap-through". *Computers & Structures*, 13(1):55–62, 1981.

APPENDIX A

MATERIAL PROPERTIES

Thickness, in.	0.008-0.009		0.010-0.062		0.063-0.128		0.129-0.249	
Basis	A	B	A	B	A	B	A	B
Mechanical Properties:								
F_{tu}, ksi:								
L	59	60	60	61	62	63	63	64
LT	58	59	59	60	61	62	62	63
ST	...	...	...	...	...	...	...	...
F_{ty}, ksi:								
L	44	45	44	45	45	47	45	47
LT	39	40	39	40	40	42	40	42
ST	...	...	...	...	...	...	...	...
F_{cy}, ksi:								
L	36	37	36	37	37	39	37	39
LT	42	43	42	43	43	45	43	45
ST	...	...	...	...	...	...	...	...
F_{su}, ksi	37	37	37	38	38	39	39	40
F_{bru}[c], ksi:								
(e/D = 1.5)	96	97	97	99	101	102	102	104
(e/D = 2.0)	119	121	121	123	125	127	127	129
F_{bry}[c], ksi:								
(e/D = 1.5)	68	70	68	70	70	73	70	73
(e/D = 2.0)	82	84	82	84	84	88	84	88
e, percent (S-basis):								
LT	10	...	[d]	...	15	...	15	...
E, 10^3 ksi:								
Primary	10.5							
Secondary	9.5				10.0			
E_c, 10^3 ksi:								
Primary	10.7							
Secondary	9.7				10.2			

Figure A.1: Properties of 2024 T3 clad aluminum sheet [36]

Thickness, in.	0.020-0.039		0.040-0.249	
Basis	A	B	A	B
Mechanical Properties:				
F_{tu}, ksi:				
L	65	67	67	68
LT	64	66	66	67
ST	...	...	...	...
F_{ty}, ksi:				
L	58	60	59	60
LT	57	59	58	59
ST	...	...	...	...
F_{cy}, ksi:				
L	58	60	59	60
LT	59	61	60	61
ST	...	...	...	...
F_{su}, ksi	39	40	40	41
F_{bru}, ksi:				
(e/D = 1.5)	97	100	100	102
(e/D = 2.0)	123	127	127	129
F_{bry}, ksi:				
(e/D = 1.5)	81	84	83	84
(e/D = 2.0)	93	96	94	96
e, percent (S-basis):				
LT	6	...	7	...
E, 10^3 ksi	10.5			
E_c, 10^3 ksi	10.7			
G, 10^3 ksi	4.0			
μ	0.33			

Figure A.2: Properties of 2014 T6 aluminum sheet [36]

Thickness, in.	0.008-0.009		0.010-0.062		0.063-0.249		0.250-0.499	0.500-1.000[b]	1.001-2.000[b]	2.001-3.000[b]
Basis	A	B	A	B	A[c]	B[c]	S[c]	S[c]	S[c,d]	S
Mechanical Properties:										
F_{tu}, ksi:										
L	55	57	57	59	60	62	60	59	58	...
LT	55	57	57	59	60	62	60	59	58	56
F_{ty}, ksi:										
L	34	35	34	35	36	38	36	36	36	...
LT	34	35	34	35	36	38	36	36	36	36
F_{cy}, ksi:										
L	38	39	38	39	40	42	39	38	35	...
LT	37	38	37	38	39	41	39	39	39	...
F_{su}, ksi	33	34	34	35	36	37	36	35	35	...
F_{bru}, ksi:										
(e/D = 1.5)	88	91	91	94	96	99	95	90	83	...
(e/D = 2.0)	109	113	113	117	119	123	119	117	115	...
F_{bry}, ksi:										
(e/D = 1.5)	60	61	60	61	63	67	63	63	63	...
(e/D = 2.0)	72	74	72	74	76	80	76	76	76	...
e, percent (S-basis):										
LT	10	...	[e]	...	15	...	12	8	[e]	4
E, 10^3 ksi:										
Primary	10.5				10.5		10.7			
Secondary	9.5				10.0		10.2			
E_c, 10^3 ksi:										
Primary	10.7				10.7		10.9			
Secondary	9.7				10.2		10.4			

Figure A.3: Properties of 2024 T42 clad aluminum sheet [36]

APPENDIX B

COMPARISON OF SOLUTION METHODS

In order to compare the Riks solver and the Static, General solver, a boundary value problem is defined. Using the boundary value problem, formulation and implementation of the Newton's method and the Riks method, which is also called as arc length method, are explained. Vasios [53] explained the arc length method by solving the boundary value problem retraced in this section.

Consider a body B surrounded by surface S, and subjected to body forces b, traction forces $\hat{t}$ per unit surface S_t and displacements $\hat{u}$ on surface S_u. After a time period Δt, volume of the body changes from V_0 to V and density of it changes from ρ_0 to ρ.

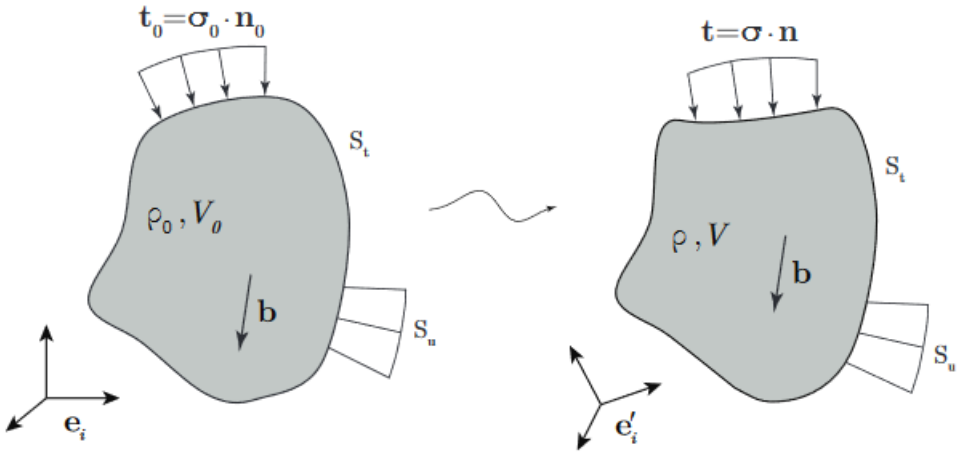


Figure B.1: A body before and after deformation [53]

Notice that $u = \hat{u}$ is known on S_u, $\hat{t}$ is known on S_t. Also, $S_u \cap S_t = \emptyset$ and $S_u \cup S_t = S$.

The balance of linear momentum reads,

$$\rho a = \nabla\sigma + \rho b \tag{B.1}$$

where a is the acceleration. The body is assumed not to be affected by a dynamic phenomenon, hence the acceleration a vanishes. Thus,

$$0 = \nabla\sigma + \rho b \tag{B.2}$$

The constitutive equation is,

$$\sigma = \hat{\sigma}\left(\varepsilon\right), \tag{B.3}$$

where

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{B.4}$$

is the strain tensor. Equations (B.2)-(B.4) together with boundary conditions constitute the strong form of the boundary value problem. An alternative formulation which is the weak form of the boundary value problem can be constructed through the Galerkin Method. The procedure is as follows,

The balance equation (Equation B.2) is multiplied by a virtual displacement field $\delta u\left(x,t\right)$ which satisfies the homogeneous form of the essential boundary conditions,

$$\delta u\left(x,t\right) = \{\delta u \in H^1 \vee \delta u = 0 \text{ on } S_u\}, \tag{B.5}$$

where H^1 functions fulfill

$$\int_B \left\|\delta u\right\|^2 dV < \infty = 0 \quad \text{in } B. \tag{B.6}$$

Thus,

$$\delta u\left[\nabla\sigma + \rho b\right] = 0 \quad \text{in } B \tag{B.7}$$

Integration of (B.7) over the domain B gives

$$\int_B \delta u\left[\nabla\sigma + \rho b\right] = 0 \quad \text{in } B \tag{B.8}$$

Then integration by parts and the Gauss integral theorem

$$\delta u\nabla\sigma = \nabla\left(\sigma^T\delta u\right) - \left(\nabla\delta u\right) : \sigma \tag{B.9}$$

are used to obtain

$$\int_B \delta u\left[\nabla\sigma + \rho b\right] dV = \int_B [\nabla\delta u : \sigma - \delta u\rho b - \nabla(\sigma^T\delta u)]dV, \tag{B.10}$$

where

$$\int_B \nabla \left(\sigma^T \delta u\right) dV = \int_S \left(\delta u \sigma\right) n dA \tag{B.11}$$

$$\delta u = 0 \text{ on } S_u$$

$$\sigma n = \hat{t} \text{ on } S_t$$

Therefore, the equation becomes

$$\sigma\left(u, \delta u\right) = \int_B \delta\varepsilon : \sigma dV - \int_B \delta u : \rho b dV - \int_{S_t} \delta u \hat{t} \mathrm{dA} \tag{B.12}$$

Equation (B.12) is also known as the principal of virtual work and provides the basis for the finite element approximation.

The virtual work of internal forces is

$$G_{\text{int}}\left(u, \delta u\right) = \int_B \delta\varepsilon : \sigma dV, \tag{B.13}$$

and the virtual work of external forces is

$$G_{ext}\left(\delta u\right) = \int_B \delta u : \rho b dV + \int_{S_t} \delta u \hat{t} dA. \tag{B.14}$$

In a finite element domain, the following approximation is introduced

$$u\left(x, t\right) = N\left(x\right) d^e \in B^e, \tag{B.15}$$

where N(x) is the interpolation matrix that stores shape functions and d^e is the displacement matrix. Then, the strain field is computed as the derivative of the displacement field

$$\varepsilon\left(x, t\right) = B\left(x\right) d^e \in B^e, \tag{B.16}$$

where $B\left(x\right)$ is the matrix containing the spatial derivatives of the shape functions. Similarly, the test function δu and its symmetric gradient $\delta\varepsilon$ are

$$\delta u\left(x\right) = N\left(x\right) \delta d^e \tag{B.17}$$

$$\delta\varepsilon\left(x\right) = B\left(x\right) \delta d^e \tag{B.18}$$

Substituting each term that are derived into the weak form equation (B.12) leads to

$$R(u) = \bigwedge_e^{n_{\text{elem}}} \delta {d^e}^T \left(\int_{B^e} B^T \left[\sigma\right] \mathrm{dV} - \int_{B^e} N^T \left[b\right] \mathrm{dV} - \int_{S^e} N^T \left[\hat{t}\right] \mathrm{dA}\right) = 0 \tag{B.19}$$

where $R(u)$ is the residual vector, $\bigwedge$ is assembly operator and n_{elem} is the number of elements. Brackets [] are used to indicate Voigt notation. Therefore, this equation can be written in a more compact form

$$R\left(u\right) = \delta d^{e^T}\left(f^e_{\text{int}}\left(u\right) - f^e_{\text{ext}}\right) = 0 \tag{B.20}$$

$$f^e_{\text{int}}\left(u\right) = \int_{B^e} B^T\left[\sigma\right] \text{dV} \tag{B.21}$$

$$f^e_{ext} = \int_{B^e} N^T\left[b\right] dV + \int_{S^e} N^T\left[\hat{t}\right] dA, \tag{B.22}$$

Nonlinear finite element solutions are used to solve nonlinear systems of equations with one or more critical points. A critical point can be defined as the point where the external loads on a body is more than the load carrying capacity of that body. In this manner, the system considered encounters with an instability problem, which can be post-buckling response of the structure in a mechanical loading-unloading problem. In a typical instability problem, the loading is assumed to be depending on a scalar parameter so that it is proportional. During the solution period, load or displacement may decrease as an unstable static response which can be seen in Figure B.2, yet, the displacement response does not include any skipping or furcation.

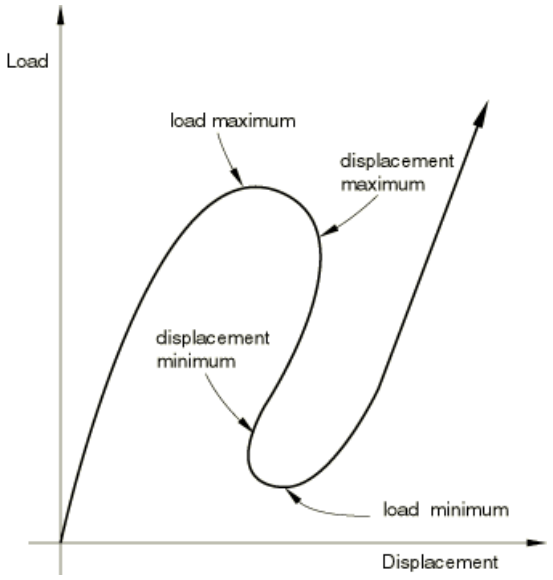


Figure B.2: Load - displacement curve of an unstable system [54]

B.1 The Newton's Method

In the Newton's method, the external load f^e_{ext} is increased from zero to the desired value gradually and the desired value is kept constant during the analysis. Recall the system of equations that is derived

$$R(u) = \delta d^{e^T} \left(f^e_{\text{int}}(u) - f^e_{\text{ext}} \right) = 0. \tag{B.23}$$

For any value of δd^{e^T}, $f^e_{\text{int}}(u) = f^e_{\text{ext}}$ and therefore,

$$R(u) = f^e_{\text{int}}(u) - f^e_{\text{ext}} = 0 \tag{B.24}$$

Introducing a scalar quantity λ, change in external load can be controlled.

For simplicity, expression f is used instead of f^e_{ext} and superscript e is not used later in this document. Then (B.24) becomes

$$R(u) = f_{\text{int}}(u) - \lambda f = 0 \tag{B.25}$$

Concerns of this system of equations are the variations in variables u and λ. Therefore, the variables are assumed to change by increments of Δu and $\Delta\lambda$

$$\lambda^{'} = \lambda_0 + \Delta\lambda \tag{B.26}$$

$$u^{'} = u_0 + \Delta u. \tag{B.27}$$

Thus the residuum becomes

$$R(u') = f_{\text{int}}(u_0 + \Delta u) - (\lambda_0 + \Delta\lambda) f = 0. \tag{B.28}$$

Applying Taylor series expansion to $f_{\text{int}}(u_0 + \Delta u)$ one obtains

$$f_{\text{int}}(u_0 + \Delta u) = f_{\text{int}}(u_0) + \left[\frac{\partial f^{\text{int}}(u)}{\partial u} \right]_{u_0} \Delta u. \tag{B.29}$$

Introducing the "Stiffness Matrix" as $[K_T] = \left[\frac{\partial f^{\text{int}}(u)}{\partial u} \right]$, (B.29) becomes,

$$f_{\text{int}}(u_0 + \Delta u) = f_{\text{int}}(u_0) + [K_T]_{u_0} \Delta u. \tag{B.30}$$

Combining (B.28) and (B.30), it is obtained

$$f_{\text{int}}(u_0) - \lambda_0 f + [K_T]_{u_0} \Delta u - \Delta\lambda f = 0, \tag{B.31}$$

where $f_{\text{int}}(u_0) - \lambda_0 f = 0$. Then the displacement increment is found as

$$\Delta u = [K_T]_{u_0}^{-1} \Delta\lambda f. \tag{B.32}$$

$(u_0 + \Delta u, \lambda_0 + \Delta\lambda)$ point is on the equilibrium path if (B.28) is satisfied. However, due to the Taylor series expansion, the equation results in a non-zero residual vector. To achieve the convergence, a correction is applied. The corrected point $(u_0 + \Delta u + \delta u, \lambda_0 + \Delta\lambda)$ will satisfy the residual vector at the point $R\left(u^{'} + \delta u\right)$,

$$f_{\text{int}}(u_0 + \Delta u) + [K_T]_{u_0+\Delta u}\, \delta u - (\lambda_0 + \Delta\lambda) f = 0 \tag{B.33}$$

$$[K_T]_{u_0+\Delta u}\, \delta u = -\left[f_{\text{int}}(u_0 + \Delta u) - (\lambda_0 + \Delta\lambda)\, f\right] = -R(u') \tag{B.34}$$

Thus, the correction is found as

$$\delta u = -\left[K_T\right]_{u'}^{-1}\ R(u')\,. \tag{B.35}$$

The displacement correction is calculated until the norm of the residual vector is less than a tolerance value. A schematic representation of the Newton iterations is given in Figure B.3.

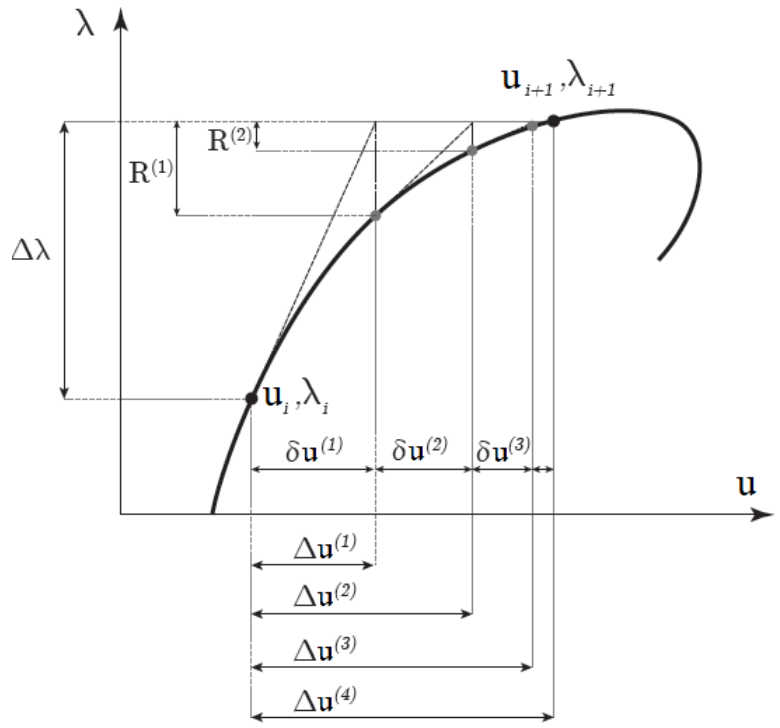


Figure B.3: Representation of Newton's method [53]

The Newton's method is implemented in many commercial FEA programs due to its quadratic convergence property which decreases the number of iterations required. However, the fast convergence of Newton's method comes with a drawback. The method can follow the equilibrium path until a critical point is reached. The behavior of the Newton's method at a critical point is presented in Figure B.4. To overcome this drawback, displacement control can be chosen by increasing the displacement instead of load.

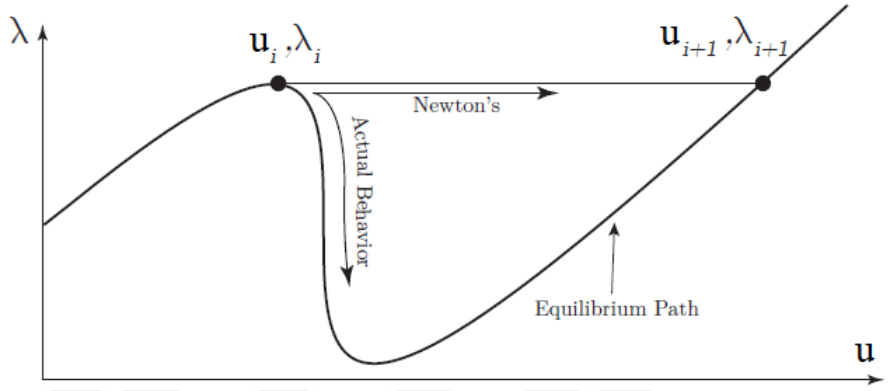


Figure B.4: Drawback of Newton's method [53]

Following pseudocode is given as a summary of the Newton's method.

1. Initiate increment
2. $\Delta u = \left([K_T]_{u_0}^{-1}\ \Delta\lambda f\right)$
3. Obtain the updated point (u', λ') as $u' = u + \Delta u$, $\lambda' = \lambda + \Delta\lambda$
4. Check for convergence $\left\|R\left(u'\right)\right\| < tolerance$
    a. If the criterion is met, go to 2
    b. If it is not,
        i. $\delta u = \left([K_T]_{u'}^{-1}\ \Delta\lambda f\right)$
        ii. $u' = u_0 + \Delta u + \delta u\ ,\ \ \lambda' = \lambda_0 + \Delta\lambda$
        iii. Obtain the updated point (u', λ') as $u' = u + \Delta u$
        iv. Check for convergence $\left\|R\left(u', \lambda'\right)\right\| < tolerance$

B.2 The Arc Length (Riks) Method

The arc length method used in the Riks solver is proposed by Riks [55] in 1979. The method presents an incremental solution of buckling and aims to eliminate the snapping problem that is explained in the previous section. Recall the system of equations that is derived in (B.25)

$$R(u) = f_{\text{int}}(u) - \lambda f = 0 \tag{B.36}$$

In arc length method, the main difference is that both Δu and $\Delta\lambda$ are unknown. Therefore,

$$R(u', \lambda') = f_{\text{int}}(u_0 + \Delta u) - (\lambda_0 + \Delta\lambda) f = 0 \tag{B.37}$$

$(u_0 + \Delta u, \lambda_0 + \Delta\lambda)$ point is on the equilibrium path if (B.37) is satisfied. If it is not, a correction is applied to the point. The corrected point $(u_0 + \Delta u + \delta u, \lambda_0 + \Delta\lambda + \delta\lambda)$ satisfies the equation

$$R(u'', \lambda'') = f_{\text{int}}(u_0 + \Delta u + \delta u) - (\lambda_0 + \Delta\lambda + \delta\lambda) f = 0 \tag{B.38}$$

Taylor series expansion reads,

$$f_{\text{int}}(u_0 + \Delta u) + \left[\frac{\partial f_{\text{int}}(u)}{\partial u}\right]_{u_0+\Delta u} \delta u - (\lambda_0 + \Delta\lambda + \delta\lambda) f = 0. \tag{B.39}$$

Here, the Jacobian matrix $[K_T]$ is introduced as,

$$[K_T]_{u_0+\Delta u} = \left[\frac{\partial f_{\text{int}}(u)}{\partial u}\right]_{u_0+\Delta u} \tag{B.40}$$

Then, the equation becomes,

$$[K_T]_{u_0+\Delta u} \delta u - \delta\lambda f = -\left[f_{\text{int}}(u_0 + \Delta u) - (\lambda_0 + \Delta\lambda) f\right] = -R(u', \lambda'). \tag{B.41}$$

Considering the displacement vector u has $N \times 1$ dimensions, there are N equations. However, N unknowns for δu and 1 unknown for $\delta\lambda$ require an additional equation. At this point, arc length Equation completes the system,

$$(\Delta u + \delta u)^T (\Delta u + \delta u) + \psi^2 (\Delta\lambda + \delta\lambda)^2 \left(f^T f\right) = \Delta l^2. \tag{B.42}$$

where ψ and Δl are user defined parameters. Δl defines the radius of the search for the next equilibrium point. ψ defines the shape of the search. For example, if $\psi = 1$,

then the point $(\Delta u^{'}, \Delta\lambda')$ belongs to a circle with a radius of Δl. For any arbitrary ψ value, (B.42) is a hyper-ellipse in displacement-load space.

Residual vector R is

$$R = f_{\text{int}}\left(u_0 + \Delta u\right) - (\lambda_0 + \Delta\lambda) f. \tag{B.43}$$

Here, we introduce

$$A = -\left(\Delta u^T \Delta u + \psi^2\, \Delta\lambda^2 \left(f^T f\right) - \Delta l^2\right) \tag{B.44}$$

for a compact representation. Then the system to be solved can be written in a more compact form

$$\begin{bmatrix} [K_T] & -f \\ 2\Delta u^T & 2\psi^2 \Delta\lambda (f^T f) \end{bmatrix} \cdot \begin{bmatrix} \delta u \\ \delta\lambda \end{bmatrix} = - \begin{bmatrix} R \\ A \end{bmatrix}. \tag{B.45}$$

The system can be solved for δu and $\delta\lambda$ to update the point $(u_0 + \Delta u, \lambda_0 + \Delta\lambda)$ to obtain the point $\left(u^{'}, \lambda'\right)$. The update algorithm continues until the convergence is achieved. A schematic representation of the arc length method is given in Figure B.5,

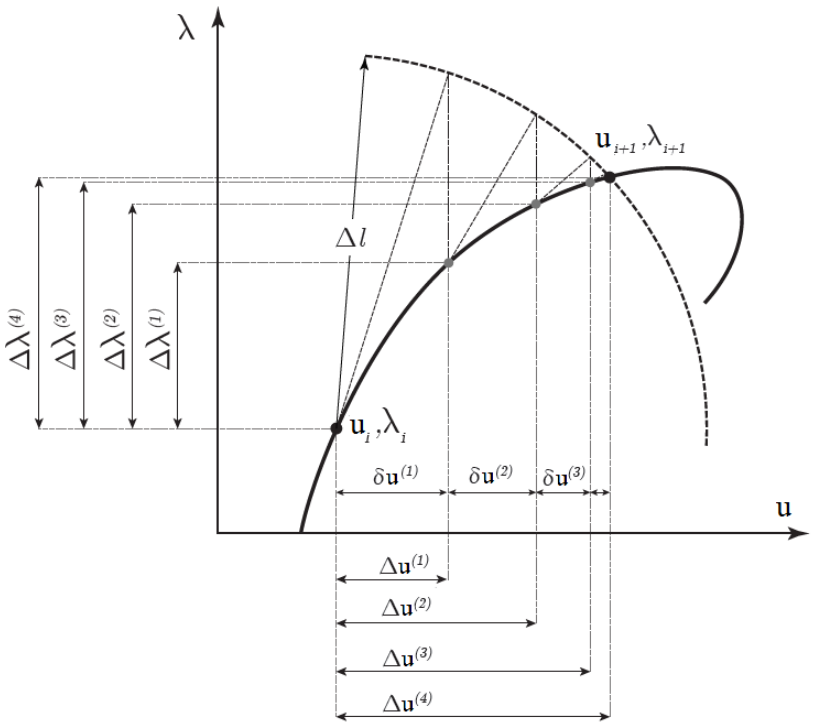


Figure B.5: Representation of arc length method [53]

This method is used in ABAQUS as a solution step which is called as Riks method. However, the method used in the program is modified because an alternative approach is required in arc length method in order to implement the method to a commercial product. In 1981, the modification is introduced by Crisfield [56],

$$\begin{aligned}\delta u = - \ & [K_T]^{-1}_{u_0+\Delta u} \ \left(f_{\text{int}}\left(u_0+\Delta u\right) - \left(\lambda_0+\Delta\lambda\right) f\right) \\ & + \delta\lambda \left(\ [K_T]^{-1}_{u_0+\Delta u} \ \right)\end{aligned} \tag{B.46}$$

$$\delta u = \delta\bar{u} + \delta\lambda\delta u_t \tag{B.47}$$

where, $\delta\bar{u}$ and δu_t depend on known information which are,

$$\bar{u} = - \ [K_T]^{-1}_{u_0+\Delta u} \ \left(f_{\text{int}}\left(u_0+\Delta u\right) - \left(\lambda_0+\Delta\lambda\right) f\right) \tag{B.48}$$

$$\delta u_t = \left([K_T]^{-1}_{u_0+\Delta u} \ f\right) \tag{B.49}$$

Substituting in arc length equation (B.42), a quadratic equation for $\delta\lambda$ is obtained,

$$A_1\delta\lambda^2 + A_2\delta\lambda + A_3 = 0. \tag{B.50}$$

where

$$A_1 = \delta u^T \ \delta u + \psi^2 \left(f^T \ f\right) \tag{B.51}$$

$$A_2 = 2\left(\Delta u + \delta\bar{u}\right)^T \ \delta_t + 2\psi^2\Delta\lambda \left(f^T \ f\right) \tag{B.52}$$

$$A_3 = \left(\Delta u + \delta\bar{u}\right)^T \ \left(\Delta u + \delta\bar{u}\right) + \psi^2\Delta\lambda^2 \left(f^T \ f\right) - \Delta l^2 \tag{B.53}$$

Then, $\delta\lambda$ is known and the displacement can be updated to complete the iteration.

Following this modification, the program calculates $\delta\bar{u}$ and δu_t by using the existing solver as the stiffness matrix is the same with any other method.

Since the solution pattern of the method is a circle or a hyper-ellipse, the method will intersect the solution curve at two points $(\delta u_1, \delta\lambda_1)$ and $(\delta u_2, \delta\lambda_2)$. Therefore, the correct point has to be selected. If the incorrect point is chosen, the direction of the solution fails to evolve forward as represented in Figure B.6.

By the following rule, the point to update the algorithm is chosen correctly. Thus, the solution will converge in most of the cases even there is a snap-back such as the one in Figure B.6.

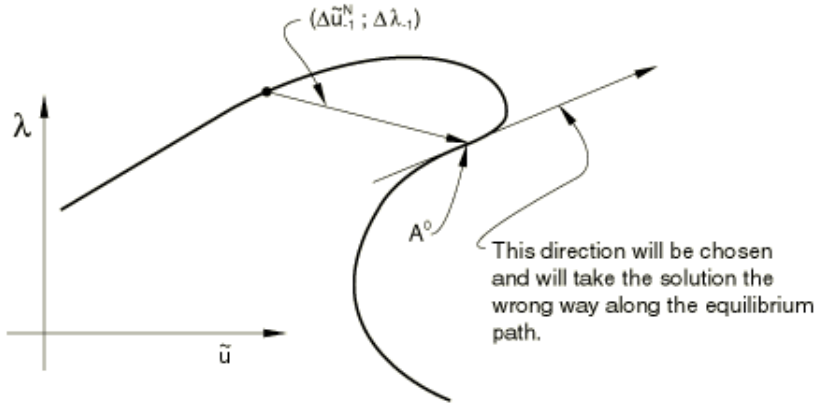


Figure B.6: Representation of an incorrect solution direction [34]

Following equation is evaluated for both points

$$D_i = (\Delta u + \delta u_i, \lambda + \Delta\lambda + \delta\lambda_i)\ (\Delta u, \lambda + \Delta\lambda)\,, \tag{B.54}$$

alternatively

$$D_i = (\Delta u + \delta u_i)^T\ (\Delta u) + \psi^2 \Delta\lambda\,(\Delta\lambda + \delta\lambda)\left(f^T\ f\right) \tag{B.55}$$

where, $i = 1, 2$.

Then, the direction which leads the larger D_i values is chosen as the correct direction.

APPENDIX C

PYTHON SCRIPTS

The skin-stringer finite element models are created with the following script.

```
#Import necessary modules
import time
start = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
from abaqus import *
from abaqusConstants import *
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
import math
import odbAccess
import job
from jobMessage import *

#Enter the input parameter values
ss_props = []
ss_props.append({"ir": 2.5, "oc": 10.0, "ic": 10.0, "h": 20.0})
ss_props.append({"ir": 2.5, "oc": 15.0, "ic": 10.0, "h": 20.0})
```

```
ss_props.append({"ir": 2.5, "oc": 10.0, "ic": 20.0, "h": 30.0})
ss_props.append({"ir": 2.5, "oc": 10.0, "ic": 10.0, "h": 30.0})
sts = [1.0, 1.27, 1.42, 1.6]
ts = [1.0, 1.27, 1.42, 1.6]
ws = [350.0]
sds = [60.0, 70.0, 80.0]
ls = [300.0, 350.0, 400.0]
sms = [0, 1]
stms = [0, 1]
sss = [0, 1, 2, 3]
fast_dia = 3.2

#Create the function that creates a finite element models with the
#given input values
def work_on_job(jname, skin_thickness, str_thickness, inner_radius,
                outercap, innercap, height, width, str_spacing,
                length, skin_material, stringer_material, fast_dia):
    path = ""
    # Calculate and arrange the inputs

    #Arrange material inputs
    # 0 -->2024 T42 Clad
    # 1 -->2024 T3 Clad
    # 2 -->7075 T7351
    mats = [{"name": 'AL2024T42CLAD', "density": 2.768e-09,
             "poisson": 0.33, "modulus": 65500.0, "Ftu": 407,
             "Fty": 241,"nt": 17},
            {"name": 'AL2024T3CLAD', "density": 2.768e-09,
             "poisson": 0.33, "modulus": 65500.0, "Ftu": 421,
             "Fty": 310,"nt": 13},
            {"name": 'AL7075T7351', "density": 2.796e-09,
             "poisson": 0.33, "modulus": 71705.0, "Ftu": 510,
             "Fty": 434,"nt": 28}]

    skin_material_name = mats[skin_material]["name"]
    skin_poisson = mats[skin_material]["poisson"]
    skin_density = mats[skin_material]["density"]
    skin_modulus = mats[skin_material]["modulus"]
    skin_Ftu = mats[skin_material]["Ftu"]
    skin_Fty = mats[skin_material]["Fty"]
```

```
skin_nt = mats[skin_material]["nt"]

str_material_name = mats[stringer_material]["name"]
str_poisson = mats[stringer_material]["poisson"]
str_density = mats[stringer_material]["density"]
str_modulus = mats[stringer_material]["modulus"]
str_Ftu = mats[stringer_material]["Ftu"]
str_Fty = mats[stringer_material]["Fty"]
str_nt = mats[stringer_material]["nt"]

#Define the file directory to work on
path = path + "\\" if path != "" else ""

#Calculate geometric parameters
outer_radius = inner_radius + str_thickness
outercap = outercap - outer_radius
innercap = innercap - outer_radius
num_of_str = int(math.ceil(width / str_spacing))
fast_rad = fast_dia / 2.0
fast_spacing = 4.5 * fast_dia
num_of_fast = int(math.ceil(length / fast_spacing))
initial_space = (width - str_spacing * float(num_of_str - 1)) / 2.0

#Write the calculated and arranged input parameters on a file
definition = {'Inner Radius': inner_radius, 'Outercap': outercap,
              'Innercap': innercap,
              'Height': height,
              'Stringer Thickness': str_thickness,
              'Skin Thickness': skin_thickness, 'Width': width,
              'Length': length,
              'Stringer Spacing': str_spacing,
              'Skin Modulus': skin_modulus,
              'Skin Ftu': skin_Ftu, 'Skin Fty': skin_Fty,
              'Skin nt': skin_nt,
              'Stringer Ftu': str_Ftu, 'Stringer Fty': str_Fty,
              'Stringer nt': str_nt,
              'Fastener Diameter': fast_dia}
text = open(path + jname + ".def", "w")
text.write("%20s" % (jname))
for item in sorted(definition.items()):
```

```
    text.write("\n")
    text.write("%20s%20.2f" % (item[0], item[1]))

text.close()

#Create the stringer section
mdb.models['Model-1'].ConstrainedSketch(name='Sketch-1',
                                        sheetSize=200.0)
mdb.models['Model-1'].sketches['Sketch-1'].Line(
    point1=(-15.0, 20.0), point2=(
        0.0, 20.0))
mdb.models['Model-1'].sketches['Sketch-1'].HorizontalConstraint(
    addUndoState=
    False, entity=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[2])
mdb.models['Model-1'].sketches['Sketch-1'].Line(
    point1=(0.0, 20.0), point2=(0.0, 1.25))
mdb.models['Model-1'].sketches['Sketch-1'].VerticalConstraint(
    addUndoState=
    False, entity=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[3])
mdb.models['Model-1'].sketches[
    'Sketch-1'].PerpendicularConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[2],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[3])
mdb.models['Model-1'].sketches['Sketch-1'].Line(
    point1=(0.0, 1.25), point2=(15.0, 1.25))
mdb.models['Model-1'].sketches['Sketch-1'].HorizontalConstraint(
    addUndoState=
    False, entity=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[4])
mdb.models['Model-1'].sketches[
    'Sketch-1'].PerpendicularConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[3],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[4])
mdb.models['Model-1'].sketches['Sketch-1'].Line(
```

```
    point1=(15.0, 1.25), point2=(15.0, -1.25))
mdb.models['Model-1'].sketches['Sketch-1'].VerticalConstraint(
    addUndoState=
    False, entity=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[5])
mdb.models['Model-1'].sketches[
    'Sketch-1'].PerpendicularConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[4],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[5])
mdb.models['Model-1'].sketches['Sketch-1'].Line(
    point1=(15.0, -1.25), point2=(-5.0, -1.25))
mdb.models['Model-1'].sketches['Sketch-1'].HorizontalConstraint(
    addUndoState=
    False, entity=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[6])
mdb.models['Model-1'].sketches[
    'Sketch-1'].PerpendicularConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[5],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[6])
mdb.models['Model-1'].sketches['Sketch-1'].Line(
    point1=(-5.0, -1.25), point2=(-5.0, 18.3272705078125))
mdb.models['Model-1'].sketches['Sketch-1'].VerticalConstraint(
    addUndoState=False, entity=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[7])
mdb.models['Model-1'].sketches[
    'Sketch-1'].PerpendicularConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[6],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[7])
mdb.models['Model-1'].sketches['Sketch-1'].Line(
    point1=(-5.0, 18.3272705078125)
    , point2=(-15.0, 18.3272705078125))
mdb.models['Model-1'].sketches['Sketch-1'].HorizontalConstraint(
    addUndoState=False, entity=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[8])
```

```
mdb.models['Model-1'].sketches[
    'Sketch-1'].PerpendicularConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[7],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[8])
mdb.models['Model-1'].sketches['Sketch-1'].Line(point1=(-15.0,
    18.3272705078125),point2=(-15.0, 20.0))
mdb.models['Model-1'].sketches['Sketch-1'].VerticalConstraint(
    addUndoState=
    False, entity=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[9])
mdb.models['Model-1'].sketches[
    'Sketch-1'].PerpendicularConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[8],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[9])
mdb.models['Model-1'].sketches['Sketch-1'].DistanceDimension(
    entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[6],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[4],
    textPoint=(20.9896659851074, 0.890911102294922), value=9.1)
mdb.models['Model-1'].sketches['Sketch-1'].dimensions[
    0].setValues(value=str_thickness)
mdb.models['Model-1'].sketches['Sketch-1'].DistanceDimension(
    entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[7],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[3],
    textPoint=(-1.90814971923828, 1.40000152587891),
    value=str_thickness)
mdb.models['Model-1'].sketches['Sketch-1'].DistanceDimension(
    entity1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[8],
    entity2=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[2],
    textPoint=(10.9400672912598, 20.1090888977051),
    value=str_thickness)
```

```
mdb.models['Model-1'].sketches['Sketch-1'].FilletByRadius(curve1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[4],
    curve2=mdb.models['Model-1'].sketches['Sketch-1'].geometry[3],
    nearPoint1=(1.75966453552246,-6.74870109558105),
    nearPoint2=(-0.146638870239258,-4.84146022796631),
    radius=inner_radius)
mdb.models['Model-1'].sketches['Sketch-1'].FilletByRadius(curve1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[7],
    curve2=mdb.models['Model-1'].sketches['Sketch-1'].geometry[8],
    nearPoint1=(-1.17310905456543,17.3852443695068),
    nearPoint2=(-2.78613662719727,18.9257106781006),
    radius=inner_radius)
mdb.models['Model-1'].sketches['Sketch-1'].FilletByRadius(curve1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[6],
    curve2=mdb.models['Model-1'].sketches['Sketch-1'].geometry[7],
    nearPoint1=(1.0264720916748,-7.77567768096924),
    nearPoint2=(-0.806512832641602,-6.52863502502441),
    radius=outer_radius)
mdb.models['Model-1'].sketches['Sketch-1'].FilletByRadius(curve1=
    mdb.models['Model-1'].sketches['Sketch-1'].geometry[3],
    curve2=mdb.models['Model-1'].sketches['Sketch-1'].geometry[2],
    nearPoint1=(-0.366598129272461,17.3852443695068),
    nearPoint2=(-2.12626075744629,20.1727504730225),
    radius=outer_radius)
mdb.models['Model-1'].sketches['Sketch-1'].delete(objectList=(
    mdb.models['Model-1'].sketches['Sketch-1'].dimensions[5],))
mdb.models['Model-1'].sketches['Sketch-1'].delete(objectList=(
    mdb.models['Model-1'].sketches['Sketch-1'].dimensions[4],))
mdb.models['Model-1'].sketches['Sketch-1'].delete(objectList=(
    mdb.models['Model-1'].sketches['Sketch-1'].dimensions[2],))
mdb.models['Model-1'].sketches['Sketch-1'].delete(objectList=(
    mdb.models['Model-1'].sketches['Sketch-1'].dimensions[0],))
mdb.models['Model-1'].ConstrainedSketch(name='__edit__',
    objectToCopy=mdb.models['Model-1'].sketches['Sketch-1'])
mdb.models['Model-1'].sketches['__edit__'].HorizontalDimension(
    textPoint=(-8.83351516723633, 22.038459777832), value=outercap,
    vertex1=mdb.models['Model-1'].sketches['__edit__'].vertices[0],
    vertex2=mdb.models['Model-1'].sketches['__edit__'].vertices[23])
mdb.models['Model-1'].sketches['__edit__'].HorizontalDimension(
    textPoint=(8.37011528015137, -10.5820636749268), value=innercap,
```

```
    vertex1=
    mdb.models['Model-1'].sketches['__edit__'].vertices[18],
    vertex2=
    mdb.models['Model-1'].sketches['__edit__'].vertices[4])
mdb.models['Model-1'].sketches['__edit__'].VerticalDimension(
    textPoint=(-7.26954936981201, -1.0966625213623), value=height,
    vertex1=mdb.models['Model-1'].sketches['__edit__'].vertices[23],
    vertex2=mdb.models['Model-1'].sketches['__edit__'].vertices[18])
mdb.models['Model-1'].sketches.changeKey(fromName='__edit__',
    toName='Sketch-1')
mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
                                        sheetSize=200.0)

#Create a three dimensional part from the defined section
del mdb.models['Model-1'].sketches['__profile__']
mdb.models['Model-1'].Part(dimensionality=THREE_D, name='Part-1',
    type=DEFORMABLE_BODY)
mdb.models['Model-1'].parts['Part-1'].BaseSolidExtrude(
    depth=length, sketch=
    mdb.models['Model-1'].sketches['Sketch-1'])
del mdb.models['Model-1'].sketches['Sketch-1']

#Transform the stringer into a shell part
mdb.models['Model-1'].parts['Part-1'].AssignMidsurfaceRegion(
    cellList=
    mdb.models['Model-1'].parts[
        'Part-1'].cells.getSequenceFromMask(mask=('[#1 ]',), ))
mdb.models['Model-1'].parts['Part-1'].OffsetFaces(faceList=
mdb.models['Model-1'].parts['Part-1'].faces.getSequenceFromMask(
    mask=('[#3031 ]',), ), targetFaces=
mdb.models['Model-1'].parts['Part-1'].faces.getSequenceFromMask(
    mask=('[#f80 ]',), ), targetFacesMethod=HALF_OF_AVERAGE,
    trimToReferenceRep=False)

#Create the skin as a shell part
mdb.models['Model-1'].ConstrainedSketch(name='Sketch-1',
                                        sheetSize=200.0)
mdb.models['Model-1'].sketches['Sketch-1'].Spot(point=(0.0, 0.0))
mdb.models['Model-1'].sketches['Sketch-1'].FixedConstraint(entity=
   mdb.models['Model-1'].sketches['Sketch-1'].vertices[0])
```

```
mdb.models['Model-1'].sketches['Sketch-1'].rectangle(
    point1=(0.0, 0.0),point2=(16.25, 12.5))
mdb.models['Model-1'].sketches['Sketch-1'].ObliqueDimension(
    textPoint=(7.59596252441406, -5.28302001953125), value=width,
    vertex1=mdb.models['Model-1'].sketches['Sketch-1'].vertices[4],
    vertex2=mdb.models['Model-1'].sketches['Sketch-1'].vertices[5])
mdb.models['Model-1'].sketches['Sketch-1'].ObliqueDimension(
    textPoint=(-9.02708435058594, 5.61320495605469), value=length,
    vertex1=mdb.models['Model-1'].sketches['Sketch-1'].vertices[1],
    vertex2=mdb.models['Model-1'].sketches['Sketch-1'].vertices[2])
mdb.models['Model-1'].Part(dimensionality=THREE_D, name='Part-2',
    type=DEFORMABLE_BODY)
mdb.models['Model-1'].parts['Part-2'].BaseShell(sketch=
    mdb.models['Model-1'].sketches['Sketch-1'])
del mdb.models['Model-1'].sketches['Sketch-1']

#Create the frame as a rigid shell part
mdb.models['Model-1'].ConstrainedSketch(name='Sketch-1',
                                        sheetSize=200.0)
mdb.models['Model-1'].sketches['Sketch-1'].rectangle(point1=
    (-38.75, 7.5),point2=(2.5, -12.5))
mdb.models['Model-1'].sketches['Sketch-1'].VerticalDimension(
    textPoint=(-50.0636901855469, -7.91519546508789),
    value=height * 1.5, vertex1=
    mdb.models['Model-1'].sketches['Sketch-1'].vertices[0],
    vertex2=mdb.models['Model-1'].sketches['Sketch-1'].vertices[1])
mdb.models['Model-1'].sketches['Sketch-1'].ObliqueDimension(
    textPoint=(-24.5992050170898, -22.7561836242676),
    value=width + 30, vertex1=
    mdb.models['Model-1'].sketches['Sketch-1'].vertices[1],
    vertex2=mdb.models['Model-1'].sketches['Sketch-1'].vertices[2])
mdb.models['Model-1'].Part(dimensionality=THREE_D, name=
'Part-3', type=DISCRETE_RIGID_SURFACE)
mdb.models['Model-1'].parts['Part-3'].BaseShell(sketch=
    mdb.models['Model-1'].sketches['Sketch-1'])

del mdb.models['Model-1'].sketches['Sketch-1']

#Regenerate the parts
mdb.models['Model-1'].parts['Part-1'].regenerate()
```

```
mdb.models['Model-1'].parts['Part-2'].regenerate()
mdb.models['Model-1'].parts['Part-3'].regenerate()

#Create materials
mdb.models['Model-1'].Material(name='AL2024T42CLAD')
mdb.models['Model-1'].materials['AL2024T42CLAD'].Density(
    table=((2.768e-09,),))
mdb.models['Model-1'].materials['AL2024T42CLAD'].Elastic(
    table=((65500.0, 0.33),
           ))
mdb.models['Model-1'].materials['AL2024T42CLAD'].Plastic(
    table=((242.3687328,
        0.0), (243.4122628, 0.000155512),
        (244.4562016, 0.000311393),(245.5006554, 0.000468069),
        (246.5457367, 0.000625988),(247.5915624, 0.000785604),
        (248.6382535, 0.000947384),(249.6859346, 0.001111798),
        (250.7347335, 0.001279323),(251.7847813, 0.00145044),
        (252.8362121, 0.001625635),(253.8891632, 0.001805396),
        (254.9437744, 0.001990217),(256.0001886,0.002180592),
        (257.0585514, 0.002377018),(258.1190113, 0.002579996),
        (259.1817192, 0.002790027),(260.2468289, 0.003007615),
        (261.3144969, 0.003233267),(262.3848822, 0.003467488),
        (263.4581463, 0.003710788),(264.5344537, 0.003963676),
        (265.6139711, 0.004226664),(266.696868,0.004500262),
        (267.7833162, 0.004784983),(268.8734903, 0.005081342),
        (269.9675675, 0.005389851),(271.0657272, 0.005711024),
        (272.1681516, 0.006045378),(273.2750253, 0.006393425),
        (274.3865355, 0.006755682),(275.5028718, 0.007132664),
        (276.6242264, 0.007524884),(277.750794,0.007932858),
        (278.8827717, 0.0083571),(280.0203591, 0.008798124),
        (281.1637585, 0.009256442),(282.3131744, 0.009732566),
        (283.468814, 0.010227009),(284.6308868, 0.010740281),
        (285.7996049, 0.011272891),(286.9751829, 0.011825348),
        (288.1578378, 0.012398158),(289.3477892,0.012991827),
        (290.5452588, 0.013606858),(291.7504714, 0.014243754),
        (292.9636536, 0.014903014),(294.185035, 0.015585138),
        (295.4148475, 0.016290621),(296.6533253, 0.017019957),
        (297.9007052, 0.017773637),(299.1572265, 0.018552152),
        (300.4231311, 0.019355986),(301.6986629, 0.020185625),
        (302.9840689, 0.021041547),(304.279598, 0.021924232),
```

(305.5855019, 0.022834153),(306.9020346, 0.023771782),
(308.2294528, 0.024737587),(309.5680155, 0.02573203),
(310.917984, 0.026755573),(312.2796224, 0.027808672),
(313.6531971, 0.02889178), (315.038977, 0.030005344),
(316.4372333, 0.031149808),(317.8482401, 0.032325612),
(319.2722734, 0.03353319),(320.7096122, 0.034772973),
(322.1605375, 0.036045386),(323.6253332, 0.037350848),
(325.1042855, 0.038689775),(326.5976828, 0.040062576),
(328.1058165, 0.041469655),(329.62898, 0.042911411),
(331.1674694, 0.044388236),(332.7215833, 0.045900518),
(334.2916227, 0.047448638),(335.8778911, 0.049032969),
(337.4806943, 0.050653882),(339.100341, 0.052311736),
(340.7371419, 0.05400689),(342.3914104, 0.05573969),
(344.0634626, 0.057510479),(345.7536166,0.059319592),
(347.4621933, 0.061167358),(349.1895161, 0.063054097),
(350.9359107, 0.064980123),(352.7017053, 0.066945742),
(354.4872309, 0.068951252),(356.2928205, 0.070996945),
(358.1188099, 0.073083105),(359.9655373, 0.075210006),
(361.8333434, 0.077377916),(363.7225715,0.079587095),
(365.633567, 0.081837795),(367.5666782, 0.084130257),
(369.5222558, 0.086464718),(371.5006528, 0.088841404),
(373.5022249, 0.091260532),(375.5273302, 0.093722312),
(377.5763293, 0.096226945),(379.6495853, 0.098774624),
(381.7474637, 0.101365531),(383.8703327,0.10399984),
(386.0185627, 0.106677719),(388.192527, 0.109399323),
(390.392601, 0.1121648),(392.6191628, 0.114974288),
(394.872593, 0.117827919),(397.1532746, 0.120725811),
(399.4615932, 0.123668077), (401.7979368, 0.126654819),
(404.162696, 0.12968613),(406.5562639, 0.132762094),
(408.9790359, 0.135882786),(411.4314103, 0.139048271),
(413.9137875, 0.142258605),(416.4265706, 0.145513836),
(418.9701652, 0.148814002),(421.5449794, 0.152159131),
(424.1514238, 0.155549242),(426.7899115, 0.158984347),
(429.460858, 0.162464447),(432.1646815, 0.165989533),
(434.9018027, 0.169559589),(437.6726447, 0.173174589),
(440.477633, 0.176834497),(443.317196, 0.180539271),
(446.1917642, 0.184288858),(449.1017708, 0.188083195),
(452.0476517, 0.191922213),(455.0298449, 0.195805833),
(458.0487912, 0.199733968),(461.104934, 0.20370652),
(464.1987189, 0.207723386),(467.3305942, 0.211784452),

```
        (470.5010108, 0.215889598),(473.710422, 0.220038693),
        (476.9592837, 0.224231601),(480.2480542, 0.228468175),
        (483.5771945, 0.232748262),(486.9471679, 0.237071701),
        (490.3584404, 0.241438323),(493.8114805, 0.245847951),
        (497.3067592, 0.250300401),(500.84475, 0.254795481),
        (504.425929, 0.259332993),(508.0507747, 0.263912731),
        (511.7197683, 0.268534482),(515.4333934, 0.273198026),
        (519.1921362, 0.277903137),(522.9964853, 0.282649582),
        (526.8469321, 0.287437121),(530.7439702,0.292265508),
        (534.6880961, 0.297134492),(538.6798084, 0.302043813),
        (542.7196087, 0.306993209),(546.8080007, 0.311982409),
        (550.945491,0.317011138),(555.1325885, 0.322079114),
        (559.3698047, 0.327186052),(563.6576538, 0.332331659),
        (567.9966522, 0.33751564),(572.3873192, 0.342737693),
        (576.8301764, 0.347997512),(581.325748, 0.353294786),
        (585.8745608, 0.358629199),(590.4771441, 0.364000432),
        (595.1340298, 0.369408162),(599.8457522, 0.37485206),
        (604.6128483, 0.380331796),(609.4358576, 0.385847034),
        (614.315322, 0.391397435)))
mdb.models['Model-1'].Material(name='AL2024T3CLAD')
mdb.models['Model-1'].materials['AL2024T3CLAD'].Density(
    table=((2.768e-09,),))
mdb.models['Model-1'].materials['AL2024T3CLAD'].Elastic(
    table=((65500.0, 0.33),
           ))
mdb.models['Model-1'].materials['AL2024T3CLAD'].Plastic(
    table=((312.0871756, 0.0), (313.1247443, 9.84829e-05),
        (314.1625508, 0.000197082),(315.2006674, 0.000296025),
        (316.2391934, 0.000395622),(317.2782505, 0.000496252),
        (318.3179808, 0.000598349),(319.3585455, 0.000702405),
        (320.4001238, 0.000808959),(321.4429119, 0.000918597),
        (322.4871226, 0.00103195),(323.5329849, 0.001149688),
        (324.5807434, 0.001272524),(325.6306582, 0.001401205),
        (326.6830043, 0.001536519),(327.738072, 0.001679286),
        (328.7961661, 0.001830361),(329.8576058, 0.001990633),
        (330.9227252, 0.002161021),(331.9918723, 0.002342477),
        (333.0654096, 0.002535981),(334.1437137, 0.002742545),
        (335.2271752, 0.002963208),(336.316199, 0.003199036),
        (337.4112037, 0.003451123),(338.5126221, 0.003720588),
        (339.6209008, 0.004008578),(340.7365003, 0.004316262),
```

(341.8598951, 0.004644834),(342.9915735, 0.004995512),
(344.1320378, 0.005369537),(345.281804, 0.005768169),
(346.4414022, 0.006192694),(347.6113763, 0.006644414),
(348.792284, 0.007124653),(349.984697, 0.007634753),
(351.1892011, 0.008176077),(352.4063957, 0.008750003),
(353.6368945, 0.009357925),(354.881325, 0.010001255),
(356.1403289, 0.010681421),(357.4145617, 0.011399862),
(358.7046931, 0.012158032),(360.0114071, 0.0129574),
(361.3354014, 0.013799443),(362.6773883, 0.014685652),
(364.0380941, 0.015617525),(365.4182594, 0.016596572),
(366.8186389, 0.017624307),(368.2400019, 0.018702256),
(369.6831318, 0.019831946),(371.1488265, 0.021014913),
(372.6378984, 0.022252694),(374.1511742, 0.02354683),
(375.6894952, 0.024898864),(377.2537173, 0.026310338),
(378.8447109, 0.027782794),(380.4633611, 0.029317773),
(382.1105677, 0.030916813),(383.787245, 0.032581445),
(385.4943225, 0.034313199),(387.232744, 0.036113595),
(389.0034686, 0.037984146),(390.80747,0.039926355),
(392.645737, 0.041941715),(394.5192734, 0.044031708),
(396.429098, 0.046197802),(398.3762446, 0.048441449),
(400.3617623, 0.050764088),(402.3867153, 0.053167139),
(404.4521829, 0.055652004),(406.55926, 0.058220064),
(408.7090565, 0.060872681),(410.9026978,0.063611192),
(413.1413248, 0.066436912),(415.4260937, 0.069351131),
(417.7581764, 0.07235511),(420.1387601, 0.075450085),
(422.569048, 0.078637261),(425.0502587, 0.081917814),
(427.5836265, 0.085292887),(430.1704015, 0.088763593),
(432.8118497, 0.092331006),(435.5092529, 0.095996171),
(438.2639088, 0.099760091),(441.077131, 0.103623735),
(443.9502492, 0.107588033),(446.8846092, 0.111653875),
(449.8815727,0.115822111),(452.9425178, 0.120093548),
(456.0688386, 0.124468953),(459.2619456, 0.128949047),
(462.5232656, 0.133534511),(465.8542417,0.138225977),
(469.2563333, 0.143024034),(472.7310166, 0.147929224),
(476.2797838, 0.152942043),(479.9041442, 0.158062939),
(483.6056232,0.163292312),(487.3857632, 0.168630515),
(491.2461232, 0.174077852),(495.188279, 0.179634577),
(499.213823, 0.185300897),(503.3243647, 0.191076969),
(507.5215304, 0.1969629),(511.8069633, 0.202958749),
(516.1823237, 0.209064523),(520.6492889, 0.215280183),

```
        (525.2095534, 0.221605639),(529.8648286, 0.228040753),
        (534.6168433, 0.234585338),(539.4673437, 0.241239158),
        (544.4180928, 0.24800193),(549.4708716,.254873324),
        (554.6274779, 0.261852962),(559.8897273, 0.26894042),
        (565.2594528, 0.276135228),(570.738505, 0.283436872),
        (576.328752, 0.290844792),(582.0320796, 0.298358387),
        (587.8503913, 0.30597701),(593.7856083, 0.313699975),
        (599.8396697, 0.321526554),(606.0145323, 0.329455978),
        (612.312171, 0.337487441),(618.7345783, 0.345620098),
        (625.283765, 0.353853067),(631.9617599, 0.36218543)))
mdb.models['Model-1'].Material(name='AL7075T7351')
mdb.models['Model-1'].materials['AL7075T7351'].Density(
    table=((2.796e-09,),))
mdb.models['Model-1'].materials['AL7075T7351'].Elastic(
    table=((71705.0, 0.33),
           ))
mdb.models['Model-1'].materials['AL7075T7351'].Plastic(
    table=((437.4948182,0.0), (438.5650685, 0.000141831),
        (439.6356341, 0.000283709),(440.70659, 0.000425803),
        (441.7780602, 0.000568392),(442.8502181, 0.000711864),
        (443.9232868, 0.000856716),(444.9975394, 0.00100355),
        (446.0732991, 0.001153077),(447.1509398, 0.001306113),
        (448.2308866, 0.001463582),(449.3136159, 0.001626511),
        (450.3996561, 0.001796034),(451.4895879, 0.001973389),
        (452.5840448, 0.00215992),(453.6837135, 0.002357074),
        (454.7893344, 0.0025664),(455.9017021, 0.002789554),
        (457.0216658, 0.003028291),(458.1501298, 0.003284472),
        (459.2880538, 0.003560056),(460.4364537, 0.003857106),
        (461.5964018, 0.004177783),(462.7690274, 0.00452435),
        (463.9555172, 0.004899168),(465.157116, 0.005304695),
        (466.3751268, 0.005743486),(467.6109116, 0.006218194),
        (468.8658918, 0.006731564),(470.1415486, 0.007286435),
        (471.4394236, 0.007885738),(472.7611192, 0.008532494),
        (474.1082992, 0.009229813),(475.482689,0.00998089),
        (476.8860766, 0.010789006),(478.3203126, 0.011657522),
        (479.7873108, 0.012589881),(481.2890491, 0.013589602),
        (482.8275692, 0.014660277),(484.4049781, 0.015805572),
        (486.0234474, 0.017029219),(487.6852151, 0.018335015),
        (489.3925849, 0.019726818),(491.1479276, 0.021208542),
        (492.9536808, 0.022784156),(494.8123504, 0.024457676),
```

```
        (496.7265098, 0.026233161),(498.6988018, 0.028114712),
        (500.7319378, 0.030106462),(502.8286992, 0.032212573),
        (504.9919376, 0.03443723),(507.2245752, 0.036784636),
        (509.5296053, 0.039259006),(511.9100929, 0.041864559),
        (514.3691753, 0.044605514),(516.9100623, 0.047486082),
        (519.536037, 0.050510458),(522.2504559, 0.053682818),
        (525.0567499, 0.057007308),(527.9584245, 0.060488038),
        (530.9590603, 0.064129074),(534.0623134, 0.067934434),
        (537.2719162, 0.071908074),(540.5916778, 0.076053886),
        (544.0254842, 0.080375687),(547.5772992, 0.084877214),
        (551.2511647, 0.089562114),(555.051201, 0.094433935),
        (558.9816079,0.099496124),(563.0466644, 0.104752012),
        (567.25073, 0.110204815),(571.5982445, 0.115857618),
        (576.0937289, 0.121713374),(580.7417858, 0.127774896),
        (585.5471, 0.134044849),(590.5144387, 0.140525743),
        (595.6486524, 0.147219931),(600.9546751, 0.154129598),
        (606.4375247, 0.161256761),(612.1023039, 0.168603259),
        (617.9542006, 0.176170753),(623.9984879, 0.183960717),
        (630.2405254, 0.19197444),(636.6857589, 0.200213018),
        (643.3397215, 0.208677353),(650.2080337, 0.217368153),
        (657.2964044, 0.226285926),(664.6106306, 0.235430982),
        (672.1565987, 0.244803432),(679.9402846, 0.254403186),
        (687.9677541, 0.264229956),(696.2451637, 0.274283255),
        (704.778761, 0.284562399),(713.5748849, 0.29506651),
        (722.6399667, 0.305794516),(731.98053, 0.316745156),
        (741.6031915, 0.327916984),(751.5146614, 0.33930837),
        (761.721744, 0.350917507),(772.2313381, 0.362742414)))

#Create the stringer section
mdb.models['Model-1'].HomogeneousShellSection(density=str_density,
    idealization=NO_IDEALIZATION,integrationRule=SIMPSON,
    material=str_material_name,name='stringer_section',
    numIntPts=5,poissonDefinition=DEFAULT,preIntegrate=OFF,
    temperature=GRADIENT,thickness=str_thickness,thicknessField='',
    thicknessModulus=None,thicknessType=UNIFORM,useDensity=OFF)
mdb.models['Model-1'].sections['stringer_section'].setValues(
    density=str_density, idealization=NO_IDEALIZATION,
    integrationRule=SIMPSON, material= str_material_name, numIntPts=5,
    preIntegrate=OFF, thickness=str_thickness, thicknessField='',
    thicknessType=UNIFORM, useDensity=ON, poisson=str_poisson,
```

```
    poissonDefinition=VALUE)

#Create the skin section
mdb.models['Model-1'].HomogeneousShellSection(
    density=skin_density, idealization=NO_IDEALIZATION,
    integrationRule=SIMPSON, material=skin_material_name,
    name='skin_section',
    numIntPts=5, poissonDefinition=DEFAULT, preIntegrate=OFF,
    temperature=
    GRADIENT, thickness=skin_thickness, thicknessField='',
    thicknessModulus=None,
    thicknessType=UNIFORM, useDensity=OFF)
mdb.models['Model-1'].sections['skin_section'].setValues(
    density=skin_density,
    idealization=NO_IDEALIZATION, integrationRule=SIMPSON,
    material=skin_material_name, numIntPts=5, preIntegrate=OFF,
    thickness=skin_thickness, thicknessField='', thicknessType=UNIFORM,
    useDensity=ON, poisson=skin_poisson, poissonDefinition=VALUE)

#Assign the stringer section
mdb.models['Model-1'].parts['Part-1'].Set(faces=
    mdb.models['Model-1'].parts['Part-1'].faces,name='Set-1')
mdb.models['Model-1'].parts['Part-1'].SectionAssignment(
    offset=0.0,offsetField='', offsetType=MIDDLE_SURFACE, region=
    mdb.models['Model-1'].parts['Part-1'].sets['Set-1'],
    sectionName='stringer_section', thicknessAssignment=FROM_SECTION)

#Assign the skin section
mdb.models['Model-1'].parts['Part-2'].Set(faces=
    mdb.models['Model-1'].parts['Part-2'].faces[0:1], name='Set-1')
mdb.models['Model-1'].parts['Part-2'].SectionAssignment(
    offset=0.0,offsetField='', offsetType=MIDDLE_SURFACE, region=
    mdb.models['Model-1'].parts['Part-2'].sets['Set-1'],
    sectionName='skin_section', thicknessAssignment=FROM_SECTION)

#Create the assembly by adding the instances
mdb.models['Model-1'].rootAssembly.DatumCsysByDefault(CARTESIAN)
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON,
    name='Part-1-1',part=mdb.models['Model-1'].parts['Part-1'])
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON,
```

```
    name='Part-2-1',part=mdb.models['Model-1'].parts['Part-2'])

#Arrange the part positions
mdb.models['Model-1'].rootAssembly.rotate(angle=180.0,
    axisDirection=(0.0,-397.0,0.0),
    axisPoint=(11.09, 214.126358,217.59),
    instanceList=('Part-1-1',))
mdb.models['Model-1'].rootAssembly.ParallelFace(fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['Part-2-1'].faces[0],
    flip=OFF, movablePlane=mdb.models['Model-1'].rootAssembly.instances[
    'Part-1-1'].faces[0])
x1 = mdb.models['Model-1'].rootAssembly.instances[
    'Part-2-1'].vertices[0].pointOn[0][0] - \
    mdb.models['Model-1'].rootAssembly.instances[
    'Part-1-1'].vertices[11].pointOn[0][0]
x2 = mdb.models['Model-1'].rootAssembly.instances[
    'Part-2-1'].vertices[0].pointOn[0][1] - \
    mdb.models['Model-1'].rootAssembly.instances[
    'Part-1-1'].vertices[11].pointOn[0][1]
x3 = mdb.models['Model-1'].rootAssembly.instances[
    'Part-2-1'].vertices[0].pointOn[0][2] - \
    mdb.models['Model-1'].rootAssembly.instances[
    'Part-1-1'].vertices[11].pointOn[0][2]
mdb.models['Model-1'].rootAssembly.translate(
    instanceList=('Part-1-1',),vector=(x1, x2, x3))
mdb.models['Model-1'].rootAssembly.translate(
    instanceList=('Part-1-1',),
    vector=(initial_space - innercap - outer_radius + 6.0, 0.0,
            0.5 * (skin_thickness + str_thickness)))

#Create multiple stringers with linear pattern
mdb.models['Model-1'].rootAssembly.LinearInstancePattern(
    direction1=(1.0, 0.0,0.0), direction2=(0.0, -1.0, 0.0),
    instanceList=('Part-1-1',), number2=1,
    number1=num_of_str, spacing2=length, spacing1=str_spacing)

#Create the slave sets for load and boundary conditions
counter = 0
RFedges = mdb.models['Model-1'].rootAssembly.instances[
              'Part-2-1'].edges[2:3]
```

```
DISPedges = mdb.models['Model-1'].rootAssembly.instances[
                'Part-2-1'].edges[0:1]
for part in mdb.models['Model-1'].rootAssembly.instances.keys():
    if part.startswith("Part-1"):
        counter += 1
        id1 = mdb.models[
            'Model-1'].rootAssembly.DatumPointByOffset(point=
                mdb.models['Model-1'].rootAssembly.instances[
                part].vertices[10], vector=(
                (innercap + outer_radius) / 2.0, 0.0, 0.0)).id
        id2 = mdb.models[
            'Model-1'].rootAssembly.DatumPointByOffset(point=
                mdb.models['Model-1'].rootAssembly.instances[
                   part].vertices[11], vector=(
                (innercap + outer_radius) / 2.0, 0.0, 0.0)).id
        mdb.models[
            'Model-1'].rootAssembly.AttachmentPointsAlongDirection(
            createPtAtEndPt=False, createPtAtStartPt=False,
            endPoint=mdb.models['Model-1'].rootAssembly.datums[id1]
            , name='Attachment Points-' + str(counter),
            numPtsBetweenPts=num_of_fast, pointCreationMethod=
            NUM_PTS_BETWEEN_PTS,
            setName='Attachment Points-' + str(
                counter) + '-Set-1', startPoint=
            mdb.models['Model-1'].rootAssembly.datums[id2])
        mdb.models[
            'Model-1'].rootAssembly.engineeringFeatures.PointFastener(
            connectionType=BEAM_MPC, directionVector=(
                mdb.models['Model-1'].rootAssembly.instances[
                    part].InterestingPoint(
                    mdb.models['Model-1'].rootAssembly.instances[
                        part].edges[7], MIDDLE),
                mdb.models['Model-1'].rootAssembly.instances[
                    part].vertices[4]),
            name='Fasteners-' + str(counter),
            physicalRadius=fast_rad, region=
            mdb.models['Model-1'].rootAssembly.sets[
                'Attachment Points-' + str(counter) + '-Set-1'])
        for edge in mdb.models['Model-1'].rootAssembly.instances[
            part].edges:
```

```
            if edge.index in [0, 4, 7, 10, 13]:
                RFedges = RFedges + mdb.models[
                    'Model-1'].rootAssembly.instances[part].edges[
                    edge.index:edge.index + 1]
        for edge in mdb.models['Model-1'].rootAssembly.instances[
            part].edges:
            if edge.index in [2, 6, 9, 12, 15]:
                DISPedges = DISPedges + mdb.models[
                    'Model-1'].rootAssembly.instances[part].edges[
                    edge.index:edge.index + 1]

#Create the master sets for load and boundary conditions
mdb.models['Model-1'].parts['Part-3'].DatumPointByMidPoint(
    point1=mdb.models['Model-1'].parts['Part-3'].InterestingPoint(
        mdb.models['Model-1'].parts['Part-3'].edges[0], MIDDLE),
    point2=mdb.models['Model-1'].parts['Part-3'].InterestingPoint(
        mdb.models['Model-1'].parts['Part-3'].edges[2], MIDDLE))
mdb.models['Model-1'].parts['Part-3'].DatumPointByOffset(point=
    mdb.models['Model-1'].parts['Part-3'].datums[2],vector=(0.0,0.0,20.0))
mdb.models['Model-1'].parts['Part-3'].ReferencePoint(point=
    mdb.models['Model-1'].parts['Part-3'].datums[3])
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name=
'Part-3-1', part=mdb.models['Model-1'].parts['Part-3'])
mdb.models['Model-1'].rootAssembly.ParallelEdge(fixedAxis=
    mdb.models['Model-1'].rootAssembly.instances[
        'Part-1-1'].edges[7],flip=OFF, movableAxis=
    mdb.models['Model-1'].rootAssembly.instances['Part-3-1'].edges[1])

point = mdb.models['Model-1'].rootAssembly.instances[
    'Part-3-1'].InterestingPoint(
    mdb.models['Model-1'].rootAssembly.instances[
        'Part-3-1'].edges[2], MIDDLE)

x1 = mdb.models['Model-1'].rootAssembly.instances[
    'Part-2-1'].vertices[0].pointOn[0][0] - \
    mdb.models['Model-1'].rootAssembly.instances[
    'Part-3-1'].vertices[2].pointOn[0][0]
x2 = mdb.models['Model-1'].rootAssembly.instances[
    'Part-2-1'].vertices[0].pointOn[0][1] - \
    mdb.models['Model-1'].rootAssembly.instances[
```

```
    'Part-3-1'].vertices[2].pointOn[0][1]
x3 = mdb.models['Model-1'].rootAssembly.instances[
    'Part-2-1'].vertices[0].pointOn[0][2] - \
    mdb.models['Model-1'].rootAssembly.instances[
    'Part-3-1'].vertices[2].pointOn[0][2]
mdb.models['Model-1'].rootAssembly.translate(
    instanceList=('Part-3-1',),vector=(x1 - 15, x2, x3))
id = mdb.models['Model-1'].rootAssembly.DatumPointByOffset(point=
mdb.models['Model-1'].rootAssembly.instances[
    'Part-2-1'].InterestingPoint(
    mdb.models['Model-1'].rootAssembly.instances[
    'Part-2-1'].edges[2], MIDDLE),vector=(0.0, 10.0, skin_thickness)).id

#Mesh the stringers
mdb.models['Model-1'].parts['Part-1'].seedPart(
    deviationFactor=0.1,minSizeFactor=0.1, size=innercap / 3.0)
mdb.models['Model-1'].parts['Part-1'].generateMesh()

#Mesh the skin
mdb.models['Model-1'].parts['Part-2'].seedPart(
    deviationFactor=0.1,minSizeFactor=0.1, size=innercap / 3.0)
mdb.models['Model-1'].parts['Part-2'].generateMesh()

#Mesh the Frame
mdb.models['Model-1'].parts['Part-3'].seedPart(deviationFactor=
    0.1,minSizeFactor=0.1,size=10.0)
mdb.models['Model-1'].parts['Part-3'].generateMesh()
mdb.models['Model-1'].rootAssembly.regenerate()

for item in mdb.models['Model-1'].rootAssembly.instances[
    'Part-3-1'].referencePoints.items():
    disprp = item[1]

#Connect the master and the slave sets
mdb.models['Model-1'].rootAssembly.Surface(name='DISPsurf',
    side2Faces=mdb.models['Model-1'].rootAssembly.instances[
    'Part-3-1'].faces[0:1])

mdb.models['Model-1'].rootAssembly.ReferencePoint(
    point=mdb.models['Model-1'].rootAssembly.datums[id])
```

```
mdb.models['Model-1'].rootAssembly.Set(edges=RFedges,name='RFset')
mdb.models['Model-1'].rootAssembly.Set(edges=DISPedges,name='DISPset')

mdb.models['Model-1'].rootAssembly.Set(name='RF',referencePoints=(
    mdb.models['Model-1'].rootAssembly.referencePoints[
    mdb.models['Model-1'].rootAssembly.referencePoints.items()[0][0]],))
mdb.models['Model-1'].rootAssembly.Set(name='DISP',
    referencePoints=(disprp,))

mdb.models['Model-1'].Coupling(controlPoint=
    mdb.models['Model-1'].rootAssembly.sets['RF'], couplingType=KINEMATIC,
    influenceRadius=WHOLE_SURFACE,localCsys=None,name='RFconstraint',
    surface=mdb.models['Model-1'].rootAssembly.sets['RFset'], u1=ON, u2=ON,
    u3=ON, ur1=ON, ur2=ON, ur3=ON)

mdb.models['Model-1'].Tie(adjust=ON, master=
mdb.models['Model-1'].rootAssembly.surfaces['DISPsurf'], name=
    'DISPconstraint',positionToleranceMethod=COMPUTED, slave=
    mdb.models['Model-1'].rootAssembly.sets['DISPset'], thickness=ON,
    tieRotations=ON)

#Create the analysis step
mdb.models['Model-1'].StaticStep(adaptiveDampingRatio=0.05,
    continueDampingFactors=False,extrapolation=PARABOLIC,
    initialInc=0.001,maxNumInc=1500, name='Step-1',
    nlgeom=ON, previous='Initial',stabilizationMagnitude=0.0002,
    stabilizationMethod=DISSIPATED_ENERGY_FRACTION)

#Define the boundary conditions
mdb.models['Model-1'].EncastreBC(createStepName='Initial',
    localCsys=None,name='BC-1', region=mdb.models[
    'Model-1'].rootAssembly.sets['RF'])

#Define the load
mdb.models['Model-1'].DisplacementBC(amplitude=UNSET,
    createStepName='Step-1',distributionType=UNIFORM,
    fieldName='', localCsys=None,name='BC-2',
    region=mdb.models['Model-1'].rootAssembly.sets['DISP'], u1=SET, u2=5,
    u3=SET, ur1=SET, ur2=SET,ur3=SET)
```

```
    #Request outputs from the analysis step
    mdb.models['Model-1'].HistoryOutputRequest(
        createStepName='Step-1', name='OUTPUT_RF', rebar=EXCLUDE, region=
        mdb.models['Model-1'].rootAssembly.sets['RF'],
        sectionPoints=DEFAULT,
        variables=('RF1', 'RF2', 'RF3', 'RM1', 'RM2', 'RM3'))

    mdb.models['Model-1'].HistoryOutputRequest(
        createStepName='Step-1', name='OUTPUT_U', rebar=EXCLUDE, region=
        mdb.models['Model-1'].rootAssembly.sets['DISP'],
        sectionPoints=DEFAULT,
        variables=('U1', 'U2', 'U3', 'UR1', 'UR2', 'UR3'))

    #Create the job
    mdb.Job(atTime=None, contactPrint=OFF, description='',
            echoPrint=OFF, explicitPrecision=SINGLE, getMemoryFromAnalysis=True,
            historyPrint=OFF,memory=90, memoryUnits=PERCENTAGE, model='Model-1',
            modelPrint=OFF, multiprocessingMode=DEFAULT, name=jname,
            nodalOutputPrecision=SINGLE,numCpus=8, numDomains=8, numGPUs=0,
            queue=None,resultsFormat=ODB, scratch='', type=ANALYSIS,
            userSubroutine='', waitHours=0, waitMinutes=0)

    #Print the reaction force values into .dat file
    mdb.models['Model-1'].keywordBlock.synchVersions(
        storeNodesAndElements=False)
    pos = mdb.models['Model-1'].keywordBlock.sieBlocks.index('*End Step') - 1
    mdb.models['Model-1'].keywordBlock.insert(pos,
                          '\n*NODEPRINT,NSET=RF,SUMMARY=NO,FREQUENCY= 1\nRF2')

    #Create the ABAQUS input file
    import os
    mdb.jobs[jname].writeInput()
    mdb.close()

#Run the function with each input parameter
for t in ts:
    skin_thickness = t
    for st in sts:
        str_thickness = st
        if not st < t:
```

```
for ss in sss:
    props = ss_props[ss]
    inner_radius = props["ir"]
    outercap = props["oc"]
    innercap = props["ic"]
    height = props["h"]
    for w in ws:
        width = w
        for sd in sds:
            str_spacing = sd
            for l in ls:
                length = l
                for sm in sms:
                    skin_material = sm
                    for stm in stms:
                        stringer_material = stm
                        jname = "ss_" + str(
                            ss) + "_st_" + str(
                            sts.index(st)) + "_t_" + str(
                            ts.index(t)) + "_w_" + str(
                            ws.index(w)) + "_l_" + str(
                            ls.index(l)) + "_sd_" + str(
                            sds.index(sd)) + "_stm_" + str(
                            stm) + "_sm_" + str(sm)
                        work_on_job(jname, skin_thickness,
                                    str_thickness,
                                    inner_radius,
                                    outercap, innercap,
                                    height, width,
                                    str_spacing, length,
                                    skin_material,
                                    stringer_material,
                                    fast_dia)
```

Following script is utilized to analyze the models and collect the results from the output files.

```
import time
start = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
from abaqus import *
```

```
from abaqusConstants import *
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
import math
import odbAccess
import job
from jobMessage import *

#Define the input parameters of the models to solve
ss_props = []
ss_props.append({"ir": 2.5, "oc": 10.0, "ic": 10.0, "h": 20.0})
ss_props.append({"ir": 2.5, "oc": 15.0, "ic": 10.0, "h": 20.0})
ss_props.append({"ir": 2.5, "oc": 10.0, "ic": 20.0, "h": 30.0})
ss_props.append({"ir": 2.5, "oc": 10.0, "ic": 10.0, "h": 30.0})
sts = [1.0, 1.27, 1.42, 1.6]
ts = [1.0, 1.27, 1.42, 1.6]
ws = [350.0]
sds = [60.0, 70.0, 80.0]
ls = [300.0, 350.0, 400.0]
sms = [0, 1]
stms = [0, 1]
sss = [0, 1, 2, 3]

#Create the function that analyses the finite element models with the
#given input values and collects the results
def work_on_job(jname, skin_thickness, str_thickness, inner_radius,
                outercap, innercap, height, width, str_spacing,
                length, skin_material, stringer_material, fast_dia):
```

```
#Define the file directory to work on
path = ""

#Submit the ABAQUS input file
os.system("abaqus job=" + jname)
import time

#Create a file to observe solution process
process = open(path + jname + ".pro", "a")

#Write the process file that the job is submitted
process.write("Submitted\n")
process.close()

#Create a function to determine the line that is
#necessary to track the reaction force values
#on the .dat file
def nodeout_index():
    dat_file = open(path + jname + ".dat", "r")
    lines = dat_file.readlines()
    for line in lines:
        if "N O D E   O U T P U T" in line:
            nodeprint_position = lines.index(line)
            break
        else:
            nodeprint_position = None
    return nodeprint_position

#Wait 20 minutes before tracking the solution
time.sleep(1200)

#Find the reaction force results on .dat file and
#wait 20 seconds
#Wait 50 seconds if the .dat file does not contain
#reaction force information
nodeout_position = None
while nodeout_position == None:
    try:
        nodeout_position = nodeout_index()
        time.sleep(20)
```

```
    except:
        time.sleep(50)

#Write the line number on process file
process = open(path + jname + ".pro", "a")
process.write("Dat. file start line is found.\n")

#Write the last read reaction force and the maximum reaction force read
#during the analysis on the process file
#If the reaction force start to decrease, terminate the analysis process
process.write(
    '%15s%15s%15s\n' % ("TERMINATE   ", "MAX FORCE   ", "LAST FORCE  "))
process.close()
kill = False
process_list = []
while kill == False:
    try:
        dat_file = open(path + jname + ".dat", "r")
        process = open(path + jname + ".pro", "a")
        datlist = []
        lines = dat_file.readlines()
        lines = lines[nodeout_position:]
        for line in lines:
            line = line.rstrip("\n")
            line = line.split(" ")
            nums = []
            for word in line:
                try:
                    word = float(word)
                    nums.append(abs(word))
                except:
                    pass
            if len(nums) == 2:
                if nums[0] == 1.0:
                    datlist.append(nums[1])
        maxdat = max(datlist)
        lastdat = datlist[len(datlist) - 1]
        if maxdat > lastdat:
            kill = True
        else:
```

```
                kill = False
            dat_file.close()
            additional_line = str(
                '%15s%15.2f%15.2f' % (str(kill), maxdat, lastdat))
            if additional_line not in process_list:
                process.write(additional_line + "\n")
                process_list.append(additional_line)
            process.close()
        except:
            dat_file.close()
            process.write("Failed" + "\n")
            process.close()
            time.sleep(30)
os.system("abaqus terminate job=" + jname)

#Write the analysis is terminated on process file
process = open(path + jname + ".pro", "a")
process.write("Kill command is triggered.\n")
process.close()

#Wait five seconds before opening the .odb file
time.sleep(5)
myOdb = odbAccess.openOdb(path=path + jname + ".odb", readOnly=True)
step = myOdb.steps["Step-1"]

#Collect deformation and reaction force data from the .odb file
u_al = [];u_val = [];rf_al = [];rf_val = []
for region in step.historyRegions.keys():
    if region.startswith("Node "):
        if "U2" in step.historyRegions[region].historyOutputs.keys():
            for data in step.historyRegions[region].historyOutputs[
                "U2"].data:
                u_al.append(data[0])
                u_val.append(data[1])
        else:
            for data in step.historyRegions[region].historyOutputs[
                "RF2"].data:
                rf_al.append(data[0])
                rf_val.append(abs(data[1]))
```

```
#Collect collapse load data
fail_load = max(rf_val)
print "fail load          : %20.2f" % (fail_load)
fail_al = rf_al[rf_val.index(fail_load)]

#Collect collapse deformation data
fail_disp = u_val[rf_val.index(fail_load)]
print "fail displacement : %20.6f" % (fail_disp)
fail = (fail_load, fail_disp, fail_al)

#Collect mass data
mass = myOdb.steps["Step-1"].mass * 1000
myOdb.close()

#Create a output file and write the obtained data
text = open(path + jname + ".out", "w")
text.write("%60s" % (jname))
text.write("\n")
text.write('%8s%4s%17s%2s%17s%5s%12s%10s' % (
" ", "Disp", " ", "RF", " ", "Slope", " ", "Slope Rate"))
text.write("\n")

#Calculate the buckling load from the slope change
#of the force-displacement data
slope = 0.0
slope_list = [0.0]
slope_rate_list = [0.0, 0.0]
buckle_found = False
buckle_load = 0.0
for i in range(0, len(rf_val)):
    if i > 0:
        slope = (rf_val[i] - rf_val[i - 1]) / (u_val[i] - u_val[i - 1])
        slope_list.append(slope)
    if i > 1:
        slope_rate = abs(slope_list[i] - slope_list[i - 1]) / slope_list[i]
        slope_rate_list.append(slope_rate)
        if slope_rate > 0.1 and buckle_found == False:
            buckle_load = rf_val[i]
            buckle_found = True
    text.write('%20.6f%20.2f%20.2f%20.2f' % (
```

```
            u_val[i], rf_val[i], slope, slope_rate_list[i]))
            text.write("\n")

        #Write the buckling load, collapse load and the mass on the output file
        slope_list = slope_list[:-1]
        slope_rate_list = slope_rate_list[:-1]
        text.write("\n")
        text.write("%20s%20.2f" % ("Buckle Load :", buckle_load))
        text.write("\n")
        text.write("%20s%20.2f" % ("Fail Load   :", fail_load))
        text.write("\n")
        text.write("%20s%20.6f" % ("Mass        :", mass))
        text.write("\n")
        text.close()
        process = open(path + jname + ".pro", "a")
        process.write("Output file is created.\n")
        end = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        process.write(end)
        process.close()
        mdb.close()

#Run the function with each input parameter
outs = []
fast_dia = 3.2
for t in ts:
    skin_thickness = t
    for st in sts:
        str_thickness = st
        if not st < t:
            for ss in sss:
                props = ss_props[ss]
                inner_radius = props["ir"]
                outercap = props["oc"]
                innercap = props["ic"]
                height = props["h"]
                for w in ws:
                    width = w
                    for sd in sds:
                        str_spacing = sd
                        for l in ls:
```

```
length = l
for sm in sms:
    skin_material = sm
    for stm in stms:
        stringer_material = stm
        jname = "ss_" + str(ss) + "_st_" + str(
            sts.index(st)) + "_t_" + str(
            ts.index(t)) + "_w_" + str(
            ws.index(w)) + "_l_" + str(
            ls.index(l)) + "_sd_" + str(
            sds.index(sd)) + "_stm_" + str(
            stm) + "_sm_" + str(sm)
        import os

        for file in os.listdir("F:\scriptworks"):
            if file.endswith(".out"):
                outs.append(file)
        if jname + ".out" not in outs:
            try:
                work_on_job(jname, skin_thickness,
                            str_thickness,
                            inner_radius, outercap,
                            innercap, height, width,
                            str_spacing, length,
                            skin_material,
                            stringer_material,
                            fast_dia)
            except:
                text = open(jname + ".out", "w")
                text.write('error')
                text.write("\n")
```

The training data sets are created with the following script.

```
import os
defs=[]
names=[]

#Collect .def files
for file in os.listdir("F:\\nnworks"):
```

```
    if file.endswith(".def"):
        defs.append(file)

#Create a function to collect output data
def get_crits():
    resdict={}
    outs = read_outs()
    for out in outs:
        text = open("F:\\nnworks\\"+out,"r")
        lines = text.readlines()
        job = lines[0].split(" ")
        for word in job:
            if not word=="":
                jname=word.rstrip("\n")
                resdict[jname]={}
        for line in lines:
            if "Buckle Load :" in line:
                resdict[jname]["buckle"]=getload(line)
            if "Fail Load   :" in line:
                resdict[jname]["fail"]=getload(line)
            if "Mass        :" in line:
                resdict[jname]["mass"]=getload(line)
    return resdict

#Create a function to read the given line
#and obtain any float on the line
def get_prop(line):
    line=line.rstrip("\n")
    line = line.split(" ")
    ret = None
    for word in line:
        try:
            ret = float(word)
        except:
            pass
    return ret

#Read .def files and collect input parameter values
res_array = []
for ssdef in defs:
```

```
    res=[]
    text = open("F:\\nnworks\\"+ssdef,"r")
    lines = text.readlines()
    job = lines[0].split(" ")
    for word in job:
        if not word=="":
            jname=word.rstrip("\n")
            names.append(jname)
    counter = 0
    for line in lines:
        if counter > 1:
            prop = get_prop(line)
            if not prop == None:
                res.append(prop)
        counter += 1
    res[2] = res[2] + res[1] + res[13]
    res[4] = res[4] + res[1] + res[13]
    res_array.append(res)

#Collect output values
outputs=get_crits()

#Match input and output parameters and write on two MS Excel
#spreadsheets in order
import openpyxl
wb = openpyxl.Workbook()
wb2 = openpyxl.Workbook()
ws = wb.create_sheet(0)
ws2 = wb2.create_sheet(0)
for i in range(0,len(res_array)):
    ws2.cell(row=i+1, column=1).value=outputs[names[i]]["buckle"]
    ws2.cell(row=i+1, column=2).value=outputs[names[i]]["fail"]
    ws2.cell(row=i+1, column=3).value=outputs[names[i]]["mass"]
    for j in range(0,len(res_array[i])):
        ws.cell(row=i+1, column=j+1).value=res_array[i][j]

#Save the MS Excel spreadsheets
wb.save(r"F:\nnworks\def.xlsx")
wb2.save(r"F:\nnworks\out.xlsx")
```

APPENDIX D

MATLAB SCRIPTS

Following script is utilized to train ANNs with different number of neurons.

```
%Clear the workspace
clc; clear all; close all;

%Format the values to be printed
format longG
format bank

%Open MS Excel spreadsheets and create data sets
out = xlsread('F:\nnworks\out.xlsx');
def = xlsread('F:\nnworks\def.xlsx');

%Define the error limit
err_limit = 5;

%Create ANNs with different number of neurons
count = 0;
for neurons = 1:64
net = fitnet(neurons);
%Configure the training parameters
net.divideParam.trainRatio = 0.90;
net.divideParam.valRatio = 0.05;
net.divideParam.testRatio = 0.05;
net.trainParam.max_fail = 500;
net.trainParam.epochs = 5000;

%Train the ANN
[net,tr] = train(net,def',out');
```

```
%Compare the ANN results and the FEA results
ctot = 0;
cmass = 0;
cbuck = 0;
cfail = 0;
for j = 1:1440
    for i = 1:16
        net_in(i) = def(j,i);
    end
    net_res = net(net_in');
    real = [out(j,1),out(j,2),out(j,3)]';
    err_list = [100*abs(net_res(1)-real(1))/real(1),100* ...
                abs(net_res(2)-real(2))/real(2),100*abs(net_res(3) ...
                -real(3))/real(3)]';

    inc = 0;
    if err_list(1)>err_limit
        cbuck = cbuck+1;
        inc = 1;
    end
    if err_list(2)>err_limit
        cfail = cfail+1;
        inc = 1;
    end
    if err_list(3)>err_limit
        cmass = cmass+1;
        inc = 1;
    end
    if inc == 1
        ctot = ctot + 1;
    end
end

%Print the ANN performance information
disp(' ')
fprintf('# of neurons is %d\n', neurons)
fprintf('# of ANN results %d percent different than FEA\n', err_limit)
disp('Buckling  Failure     Mass')
fprintf('%8d %8d %8d\n',[cbuck,cfail,cmass])
```

```
disp(' ')
count = count + 1;
neuron_nums(count) = neurons;
ctot_nums(count) = ctot;
fprintf('Neurons = %d Total Fails: %d\n', [neurons, ctot])
end
```

The ANN performance is measured by using the following script.

```
%Clear the workspace
clc; clear all; close all;
%Format the values to be printed
format longG

%Import the ANN to compare with the FEA
anns = dir(fullfile('F\ANN\checkers','*.mat'));

%Open MS Excel spreadsheets and create data sets
out = xlsread('F\checkers\out.xlsx');
def = xlsread('F\checkers\def.xlsx');

%Define the error limit
err_limit = 5;

%Compare the ANN results and the FEA results
count = 0;
for k=1:length(anns)
    clearvars -except anns out def err_limit count k
    load(anns(k).name)
    ctot = 0;
    cmass = 0;
    cbuck = 0;
    cfail = 0;
    for j = 1:7
        for i = 1:16
            net_in(i) = def(j,i);
        end
        net_res = net(net_in');
        real = [out(j,1),out(j,2),out(j,3)]';
        err_list = [100*abs(net_res(1)-real(1))/real(1),100* ...
```

```
                    abs(net_res(2)-real(2))/real(2),100*abs(net_res(3) ...
                    -real(3))/real(3)]';
        err_array(j,1,:) = net_res;
        err_array(j,2,:) = real;
        err_array(j,3,:) = err_list;
        inc = 0;
        if err_list(1)>err_limit
            cbuck = cbuck+1;
            inc = 1;
        end
        if err_list(2)>err_limit
            cfail = cfail+1;
            inc = 1;
        end
        if err_list(3)>err_limit
            cmass = cmass+1;
            inc = 1;
        end
        if inc == 1
            ctot = ctot + 1;
        end
    end

        %Print the ANN performance information
    if ctot==0
        disp(' ')
        disp(anns(k).name)
        disp('ANN Result FEM Result    % Error ')
        for j=1:7
            fprintf('%10.2f %10.2f %10.2f\n',[err_array(j,:,1),...
                                err_array(j,:,2),err_array(j,:,3)]')
        end
        fprintf('# of ANN results %d percent different than FEA\n', err_limit)
        disp('Buckling  Failure     Mass')
        fprintf('%8d %8d %8d\n',[cbuck,cfail,cmass])
        count = count + 1;
        ctot_nums(count) = ctot;
        fprintf('Total Fails: %d\n', ctot)
    end
end
```