İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

STRUCTURAL IDENTIFICATION AND STATIC AEROELASTIC OPTIMIZATION OF ARW-2 WING WITH MULTIDISCIPLINARY CODE COUPLING

M.Sc. Thesis by
Ahmet AYSAN

Department : Aeronautics and Astronautics Engineering

Programme : Aeronautics and Astronautics Engineering

June 2009

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

STRUCTURAL IDENTIFICATION AND STATIC AEROELASTIC OPTIMIZATION OF ARW-2 WING WITH MULTIDISCIPLINARY CODE COUPLING

M.Sc. Thesis by
Ahmet AYSAN
(511071101)

Date of submission : 04 May 2009
Date of defence examination: 08 June 2009

Supervisor (Chairman) : Assis. Prof. Dr. Melike NİKBAY (ITU)
Members of the Examining Committee : Prof. Dr. Metin Orhan KAYA (ITU)
Prof. Dr. Serdar ÇELEBİ (ITU)

June 2009

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ÇOK DİSİPLİNLİ KOD EŞLEME İLE ARW-2 KANADININ YAPISAL TANIMLANMASI VE STATİK AEROELASTİK OPTİMİZASYONU

YÜKSEK LİSANS TEZİ
Ahmet AYSAN
(511071101)

Tezin Enstitüye Verildiği Tarih : 04 Mayıs 2009
Tezin Savunulduğu Tarih : 08 Haziran 2009

Tez Danışmanı : Yrd. Doç. Dr. Melike NİKBAY (İTÜ)
Diğer Jüri Üyeleri : Prof. Dr. Metin Orhan KAYA (İTÜ)
Prof. Dr. Serdar ÇELEBİ (İTÜ)

Haziran 2009

FOREWORD

I would like to express my deep appreciation and thanks to my advisor, Assistant Professor Melike Nikbay for her support and assistance. I am also grateful to her for providing me full-time scholarship.

I would like to thank to my colleague Arda Yanangönül for both his support in my work and always being with me . I would never proceed in my studies without his guidance.

Many thanks to The Scientific and Technological Research Council of Turkey – TÜBİTAK for financial support. This study is financed by the TÜBİTAK project titled "Analysis and Reliability Based Design Optimization of Fluid-Structure Interaction Problems Subject to Instability Phenomena" with the grant number 105M235.

I also would like to thank to ITU Institute of Science and Technology and the academic staff of Aeronautics and Astronautics Faculty of Istanbul Technical University for helping me with their valuable experience. I would like to thank to Informatics Institute of Istanbul Technical University for providing their facilities.

Many friends of mine helped me during my graduate studies. I appreciate them for their friendship.

Finally, I would like to express my deep appreciation to my family for their support, love and patience.

June 2009

Ahmet Aysan

Astronautics and Aeronautics Engineering

TABLE OF CONTENTS

Page

ABBREVIATIONS

AGARD : Advisory Group for Aerospace Research and Development
ALE : Arbitrary Lagrangian Euleria
ARW : Aeroelastic Research Wing
CAD : Computer Aided Drafting
CFD : Computational Fluid Dynamics
CSD : Computational Structural Dynamics
FE : Finite Element
MDO : Multi-Disciplinary Optimization
MLS : Moving Least Squares
MOGA : Multi Objective Genetic Algorithm
MpCCI : Mesh Based Parallel Code Coupling Interface
NSGA : Non Dominated Sorting Genetic Algorithm
SQP : Sequential Quadratic Programming

LIST OF TABLES

Page

LIST OF FIGURES

Page

STATIC AEROELASTIC OPTIMIZATION OF ARW-2 WING WITH MULTIDISCIPLINARY CODE COUPLING

SUMMARY

In last two decades, interest in multidisciplinary design analysis and optimization has increased substantially. Besides of experimental studies, computational studies via academic and commercial codes took a place in literature. There are also some extended experimental database of a test case like ARW-2 wing which is selected as design model in this study.

In the first step of this study, an accurate computational wing model which has similar responses with experimental wing model is created. Due to the lack of some properties of experimental wing in the literature, an inverse engineering problem with multi objective optimization tools has been defined. The purpose of this effort is to identify missing properties of the wing. After this identification, the computational wing model is validated with experimental data for both static and dynamic responses.

In the second step of this study, a static aeroelastic optimization problem has been set by using previously validated ARW-2 computational wing model. The objectives of the problem are maximization of lift over drag ratio and minimization of weight. The problem is constrained with aerodynamic and structural criteria. As optimization algorithm, NSGA-II is used to govern optimization process. According to the results, pareto set for the optimum designs are acquired and the optimum design is selected with a satisfactory improvement on the design.

During the study, calculations are performed by using commercial codes. As a finite element solver ABAQUS 6.7-1 is used while FLUENT 6.3.26 is used to solve flow equation. To couple these flow and structural solvers, MpCCI 3.0.6 is used. An advanced optimization software ModeFrontier 4.0 is used to solve optimization problems.

ÇOK DİSİPLİNLİ KOD EŞLEME İLE ARW-2 KANADININ STATİK AEROELASTİK OPTİMİZASYONU

ÖZET

Son yıllarda, çok disiplinli tasarım analizleri ve optimizasyona olan ilgi oldukça artmıştır. Deneysel çalışmaların yanısıra, akademik ve ticari kodlar kullanarak yapılan sayısal çalışmalar literatürde yerini almıştır. Bu çalışmada model olarak seçilen ARW-2 kanadı gibi geniş deneysel veritabanı olan test durumları yer almaktadır.

Bu çalışmanın ilk aşamasında, deneysel kanat modeli ile benzer cevaplara sahip olan sayısal bir kanat modeli oluşturulmuştur. Deneysel kanadın bir takım özelliklerinin literatürdeki eksiklerinden ötürü, tersine mühendislik ile çok amaçlı bir optimizasyon problemi kurulmuştur. Bu denemenin amacı, kanadın eksik özelliklerinin teşhis edilmesidir. Bu teşhisten sonra, sayısal kanat modeli statik ve dinamik cevaplarına göre deneysel kanat modeli ile doğrulanmıştır.

Çalışmanın ikinci aşamasında, daha önce doğrulanan ARW-2 sayısal kanat modeli kullanılarak bir statik aeroelastik optimizasyon problemi tanımlanmıştır. Problemin amacı kanadın taşıma/sürükleme oranını maksimize etmek ve ağırlığını minimize etmektir. Problem aerodinamik ve yapısal kriter ile kısıtlanmıştır. Optimizasyon algoritması olarak NSGA-II optimizasyon prosesini yürütmek üzere kullanılmıştır. Elde edilen sonuçlara göre, optimum tasarımlar için pareto kümesi elde edilmiş ve optimum tasarım, tasarımda tatmin edici bir iyileştirme ile seçilmiştir.

Çalışma süresince, sayısal hesapların yapılmasında ticari kodlardan faydalanılmıştır. Sonlu elemanlar yöntemi çözücüsü olarak ABAQUS 6.7-1 kullanılırken, akış denklemlerini çözmek için FLUENT 6.3.26 kullanılmıştır. Bu iki kodun eşlenmesinde ise MpCCI 3.0.6'dan faydalanılmıştır. Gelişmiş bir optimizasyon yazılımı olan ModeFrontier 4.0, optimizasyon problemlerini çözmek üzere kullanılmıştır.

1. INTRODUCTION

1.1 Motivation

The goal is to perform an aeroelastic optimization study based on ARW-2 (Aeroelastic Research Wing) wing model for steady-state conditions while both aerodynamic and structural parameters can be used as optimization variables. Since some of the structural properties of ARW-2 composite wing is missing in literature, firstly, a reliable 3-D computational ARW-2 wing model is needed to be identified in an inverse approach and validated with experimental results. The missing material properties and thicknesses of the skin, ribs, axial bars and spars are defined as optimization variables of an multi-objective optimization problem based on structural mechanics. The objectives are minimization of the errors in the first five modal frequencies, in mode shapes, in pre-defined static bending and torsional responses of the wing model. An isotropic skin approach is used for simplicity. ModeFrontier is used as an optimization tool and Abaqus as a FE structural solver. In the second step, the computational ARW-2 model's aeroelastic response is validated with the experimental results. By coupling Fluent and Abaqus softwares through MPCCI, static aeroelastic analysis for Mach number 0.8 at angle of attack changing between -1 to 3 degrees are performed for fluid-structure interaction validation. In the third step, a multidisciplinary optimization study is performed on the verified computational ARW-2 model in order to improve the lift/drag performance and static displacement criteria of the wing while trying to reduce its weight. The angle of attack, the thicknesses of ribs and spars are defined as design variables while a multiobjective genetic algorithm (MOGA) is employed in the aeroelastic optimization framework.

1.2 Background

1.2.1 Aeroelasticity

Aeroelasticity is a field of study that concerns the interaction between the deformation of an elastic structure in an airstream and the resulting aerodynamic force. This interdisciplinary study can be illustrated by Figure 1.1.

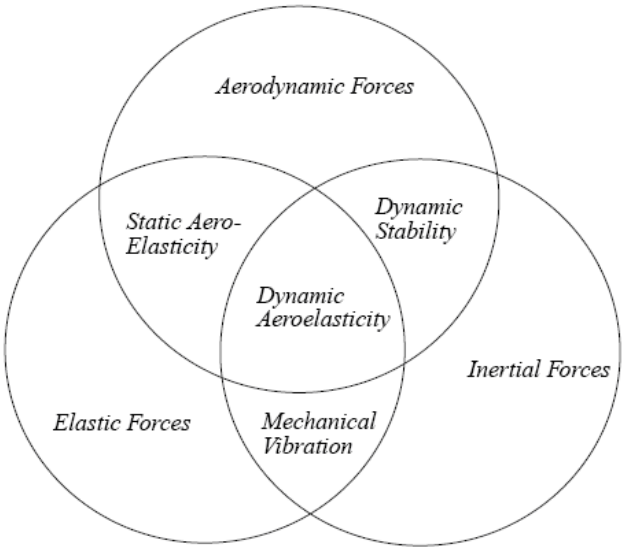

Figure 1.1 : Schematic of the field of aeroelasticity [1]

The interaction of aerodynamic loading caused by steady flow and consequent elastic deformation of the surface is called static aeroleasticity. This area has two types of design problems. The most usual problem is the effect of elastic deformation on the airloads in normal operating conditions. Flight stability, quality of control, influence on performance and load distribution are related to these effects. Another problem about static aeroelasticity is the instability of the structure which is called "divergence".

The most commonly posed problems about aeroelasticity are stability problems. The deformation of the structure leads to a different aerodynamic load distribution on the structure. The increase in the load leads to an increase in the deflection of the structure and may lead to a failure in structure. When inertial forces have less effect, we refer to this as a static aeroelasticity instability (divergence). On the other hand, when the inertial forces are important, the resulting aeroelastic instability is called "flutter".

1.2.2 Multidisciplinary Optimization (MDO)

Many studies in aerospace industry need to be considered as multidisciplinary problems due to their complexity and interaction between other disciplines like aerodynamics, structural dynamics, heat transfer, vibration, control, etc. Developing new and complex methodologies depends on the interaction of these different disciplines, so that entire system is considered as a coupled system.

Coupled systems have complexity in their nature. One design requirement can be an input by a discipline whereas this variable can be an output of another discipline. This complexity may induce contradiction among disciplines. For example, one aircraft design may be good from the point of view of structural dynamics as this design is useless from another point of view. Designing an aircraft with high weight would decrease the flexibility and suppress aeroelastic instabilities, however it decreases aerodynamic performance of the aircraft. A systematic approach to solve this kind of coupled problems is defined as "Multi-disciplinary Optimization (MDO)". [2]

1.3 Outline

This thesis provides two major studies mainly about multi objective optimization by using ARW-2 experimental wing model as a test case. First of all, background of computational aeroelasticity and multidisciplinary optimization is provided in chapter 2. In chapter 3, the computational framework and the methodology developed and used in this study is described. In chapter 4, an inverse engineering problem by using multi objective optimization tools are presented in order to completely identify the test case ARW-2 computational wing model. In chapter 5, a static aeroelastic optimization problem is defined and the results are presented.

2. LITERATURE REVIEW

2.1 Computational Aeroelasticity

In a computational framework, aeroelastic analysis requires a simultaneous analysis of fluid and structural equations. To further improve the performance of the air vehicle, implementation of multi-disciplinary optimization techniques into the computational design process will be beneficial. The topic of computational aeroelasticity has flourished in the last few decades with the great advances in computer technology and algorithms. The Euler/ NavierStokes flow solvers have been widely employed for the fluid domain as in the works of Farhangnia [3], Bhardwaj [4], Karpel [5], Newman [6], Garcia and Gruswamy [7], Liu [8], Cai [9], Kamakoti [10], Farhat [11-14], Gordnier [12], Feng and Soulaimani [13]. Recently, reduced order models have been applied to fluid domain by Dowell [15] , Lieu [16] and Haddadpour [17]. Structural analysis of the aeroelastic problem is performed by using modal equations as in the works of Karpel [5], Farhangnia [3], Garcia and Gruswamy [7], Liu [8]. The structural finite element method is employed in the studies presented by Liu [8], Farhat and Lesoinne [11], Gordnier [12], Bhardwaj [4], Relvas and Suleman [18], Gordnier [12]. To transform the physical data between fluid and structure, three different methods of coupling can be used. These are loosely coupled [19] , closely coupled [8-13] and fully coupled methods [11].

Computational aeroelasticity with commercial codes are becoming more common recently because of the industrial needs. Kuntz and Menter (20) used the commercial software packages to perform an aeroelastic analysis of the AGARD 445.6 wing with the high fidelity non-linear finite element solver ANSYS and the general purpose finite volume based CFD code CFX-5. Mesh based Parallel Code Coupling Interface (MpCCI) is used for the interfacing and data transfer between CSD and CFD solvers. Love et al (19) used the Lockheed's unstructured CFD solver SPLITFLOW and the MSC/Nastran CSD solver for the aeroelastic computations of an F-16 model in a max-g pull-up maneuver. They used a loosely coupled method for the analysis. Data

transfers between the codes are done by using Multi-Disciplinary Computing Environment (MDICE). Heinrich [21] used the DLR'S unstructured TAU code with MSC/Nastran finite element solver for the aeroelastic analysis of an A340 like aircraft. MpCCI is used for the loosely coupling of these codes.

Cavagna [22] used an interfacing method that can be applied on unmatching meshes based on Moving Least Squares (MLS). They used Fluent for the fluid solver and the MSC-Nastran for the structural solver for the aeroelastic analysis of the AGARD 445.6 wing. They used a user defined function (UDF) to implement the grid deformation and scheme for the Crank-Nicolson algorithm for Fluent.

Thirifay and Geuzaine [23] studied the AGARD 445.6's aeroelastic problems both with steady and the unsteady approximations in a loosely coupled method. In their study they used a three dimensional unstructured CFD solver developed in CANAERO and a CSD solver "the SAMCEF Mecano code" for their analysis. They used the ALE method for the moving mesh method. MpCCI is used for the aeroelastic code coupling tool.

Yosibash [24] designed an interface to couple a parallel spectral/hp element fluid solver "Nektar" with the hp-FEM solid solver "StressCheck" for the direct numerical solution (DNS) over AGARD 445.6 wing. ALE formulation is used for the fluid structure coupling. They used the one-way coupling method with linear assumption for the structural response and the two-way coupling method which considers the non-linear effects of the structure. The ALE formulation of the Navier-Stokes equations are also used in Svacek's work [25]. The Reynolds averaged Navier Stokes (RANS) system of equations with the Spallart-Almaras turbulence model were used to compare the results with NASTRAN code solutions. Fazelzahed [26] highlighted the effects of an external force and mass parameters such as the mass ratio and their locations on the flutter speed and frequency by performing numerical simulations. Unsteady aerodynamic pressure loadings were taken into account and the resulting partial differential equations are converted into a set of eigenvalue equations through the extended Galerkin's approach.

Stanford et al. [27] used a design model of MAV (Micro Air Vehicle) with a low aspect-ratio by using an aeroelastic code to couple a Navier-Stokes solver and a finite element solver. For the steady laminar flow field, they solved 3-D incompressible viscous Navier-Stokes equations and interpolated the computed wing pressures to the FEA to solve the displacements using the structural membrane model. They

interpolated the displacement onto the model and remeshed CFD grid using a mesh/slave moving-grid scheme. After repeating these steps until convergence is achieved, they compared their results with the experimental data to validate the computational model. Lim [28] studied the aeroelastic stability of a bearingless rotor with a composite flexbeam. Numerical results were compared with both previously published experimental results and theoretical values.

Xie [29], in his work, emphasized the importance of nonlinear aeroelastic stability for the high-altitude long-endurance (HALE) aircraft model by using MSC/NASTRAN as a FEM software and an unsteady aerodynamic code with planar doublet lattice method.

In Pahlavanloo's study [30], AGARD 445.6 wing model was used for dynamic aeroelastic simulations by using EDGE code which is previously validated with experiments. In this study, flutter boundary for AGARD wing in subsonic and supersonic regions were presented and additional validation of aeroelastic implementation of EDGE was provided. Edward [31] performed generalized aeroelastic analysis method to apply on three cases which are restrained, unrestrained and a wing model. A computer code for the generalization of a doublet lattice method was applied to the calculation for the wing model for both incompressible and subsonic flow conditions. To check accuracy of the code, for all cases aeroelastic flutter, divergence speed and frequencies were compared with published results.

Jian-min [32] investigated aeroelastic characteristics of an airship by coupling a SIMPLE method based finite volume code and a finite element code. They developed a nonlinear finite element method to solve the structure equations of the airship and derived the flow solver based on the Reynolds-averaged Navier-Stokes equations. A Thin Plate Spline (TPS) is used as the interface to exchange the data between fluid and structure codes.

A nonlinear aeroelastic analysis of a two-dimensional airfoil was presented in Sarkar's [33] study. Due to structural damage potential of stall aeroelastic instability, aeroelastic instability and nonlinear dynamic response were investigated by considering two different oscillation models one of which is pitching oscillation and the other one is flap-edgewise oscillation. A quasi-steady Onera model was used to calculate the nonlinear aerodynamic load in the dynamic stall regime. Another nonlinear aeroelastic analysis was presented by Shams et al [34]. They used the second-order form of nonlinear general flexible Euler-Bernoulli beam equations for

structural modeling. Aerodynamic loading on the model which is “Human Powered Aircraft’s” (HPA) long, highly flexible wing were determined by using unsteady linear aerodynamic theory based on “Wagner function”. The nonlinear integro differentials aeroelastic equations were obtained from the combination of these two types of formulations. Although their linear study for a test case had a good agreement with experiments, the nonlinear model did not satisfy the experimental data. Silva [35] presented an improvement to the development of CFD based unsteady aerodynamic reduced-order model in his study. This improvement involves the simultaneous excitation of the structural modes of the CFD based unsteady aerodynamic system. CFL3Dv6.4 code which solves the three-dimensional, thin layer, Reynolds-averaged Navier-Stokes equations with an upwind finite volume formulation. The second-order backward time differencing with subiterations was used for static and dynamic aeroelastic calculations. Another nonlinear aerolasticity study in supersonic and hypersonic regimes was performed by Abbas et al [36]. Their study shows that the freeplay in the pitching degree-of-freedom and soft/hard cubic stifness in the pitching and plunging degrees-of-freedom have significant effects on the limit cycle oscillations exhibited by the aeroelastic system in the supersonic and hypersonic regimes. They also investigated the effect of the radius of gyration, the frequency ratio and post-flutter regimes on the aeroelastic system behaviour by using Euler equations and CFL3D code. They concluded that the non linear aerodynamic stiffness induces damaging effects for aeroelastic system at high Mach numbers.

Computational aeroeleasticity has been also used in many applications other than aerospace engineering. Chattot [37], in his study, used his previously validated code and performed aeroelastic simulation of wind turbine to observe its vortex model. Baxevanou [38] developed a novel aeroelastic numerical model which combines a Navier–Stokes CFD solver with an elastic model and two coupling schemes for the study of the aeroelastic behaviour of wind turbine blades undergoing classical flutter. In the conclusion, the capabilities of the numerical model were presented to perform an aeroelastic analysis accurately. Moreover, Braun [39] has performed CFD and aeroelastic analysis on the CAARC (Commonwealth Advisory Aeronautical Council) standard tall building by using numerical simulation techniques. A major topic was referred to one of the first attempts to simulate the aeroelastic behavior of a

tall building employing complex CFD techniques. Numerical results were compared with numerical and wind tunnel measurements with some important concluding remarks about the simulation.

Recently, as a former study of this thesis, a robust aeroelastic optimization methodology was developed by multidisciplinary code coupling approach employing common softwares such as Fluent and Abaqus with Modefrontier and MpCCI as in the work of Nikbay [76] and Öncü [77] for the aeroelastic optimization of AGARD 445.6 wing. After this methodology was successfully established, the current work focuses on aeroelastic optimization of a more complicated 3D wing model of ARW 2 which has inner rib, spar and axial bar elements.

2.2 Multidisciplinary Optimization

Aircraft design is a complex engineering process that depends on the interaction of different disciplines so that the system of these disciplines must be thought as a coupled system. For instance, design of an aircraft wing with low weight would improve the aerodynamics performance but this will increase the flexibility of the wing which may lead to aeroelastic instability. Such a system can be solved by aeroelastic optimization.

Therefore, the contradictory situations in aircraft design optimization process disciplines such as aerodynamics, structural dynamics, propulsion, flight controls, etc. must be thought as a whole system to find the optimized design. Moreover design requirements enhanced with the developments in computer technology. The increased complexity and the computational cost issues regarding multi-disciplinary design leaded to a concept referred as “Multi-Disciplinary Optimization (MDO)”. MDO which is a growing field of study has been particularly applied to aerospace engineering problems.

As the capabilities of computational studies grow, the fidelity level of engineering numerical analysis increase as well. These multifidelity models range from low fidelity simple physics models to high-fidelity detailed computational simulation models. Including higher-fidelity analyses in the design process, for example through the use of computational fluid dynamic (CFD) analyses, leads to an increase in complexity and computational expense. As a result, design optimization, which

requires large numbers of analyses of objectives and constraints, becomes more expensive for some systems. Robinson [40] presented a methodology for improving the computational efficiency of high-fidelity design, by using variable fidelity and variable complexity in a design optimization framework. to minimize expensive high-fidelity models at reduced computational cost, Surrogate-based-optimization methods were used. The methods are useful in problems for which two models of the same physical system exist: a high-fidelity model which is accurate and expensive, and a low-fidelity model which is less costly but less accurate. Three methods were demonstrated on a fixed-complexity analytical test problem and a variable complexity wing design problem. The SQP-like method consistently outperformed optimization in the high-fidelity space and the other variable complexity methods. On the wing design problem, the combination of the SQP-like method and corrected space mapping achieved 58% savings in high-fidelity function calls over optimization directly in the high-fidelity space. These savings provided a reduction in computational cost.

Alonso [41] presented a new approach for software architecture of a high-fidelity multidisciplinary design framework that facilitates the reuse of existing components, the addition of new ones, and the scripting of MDO procedures. The necessary components of a high-fidelity aero-structural design environment for complete aircraft configurations were implemented, and were demonstrated with two separate aero-structural analyses: a supersonic jet and a launch vehicle. An aero-structural solver that uses high-fidelity models for both disciplines as well as an accurate coupling procedure was the core of the effort. The Euler or Navier–Stokes equations were solved for aerodynamics side and a detailed finite-element model was used for the primary structure. In Kodiyalam's [42] study, Multidisciplinary Design Optimization of a vehicle system for safety, NVH (noise, vibration and harshness) and weight, in a scalable HPC (High Performance Environment) environment, was addressed. HPC, utilizing several hundred processors in conjunction with approximation methods, formal MDO strategies and engineering judgement were used to obtain superior design solutions with significantly reduced elapsed computing times. MDO solution time through HPC was significant in improvement engineering productivity, so reinforcement the vehicle design were made possible. Korngold [43] presented a new algorithm to perform multidisciplinary optimization.

Coupled nonhierarchic systems with discrete variable was efficiently optimized. Through formulation of first and second order "Global Sensitivity Equations", the global approximation was optimized using branch and bound or simulated annealing. The approximation was to decompose the system into the disciplines and use designed experiments within the disciplines to build local response to the discipline analysis. This algorithm based on established statistical methods was implemented very well in an example problem.

Multidisciplinary optimization techniques were also used in more realistic problems. For example, Venter [44] used particle swarm optimization method in his study for multidisciplinary optimization of a transport aircraft. A new algorithm for multidisciplinary optimization problems were introduced and the recommendations for the use of the algorithm in future applications were provided. This algorithm was applied to the multidisciplinary design of a typical long-range transport aircraft wing of the Boeing 767 class. The wing was optimized relative to a reference wing. This was an unconstrained problem which has a purpose of maximization the range for the wing by varying the aspect ratio, the depth-to-chord ratio, the number of internal spars and ribs and the wing cover construction. Gantois [45] performed multi-disciplinary design of a large-scale civil aircraft wing by taking into account the manufacturing cost. A multi level MDO process was constucted and implemented through a hierarchical system. Calculation of the sensitivities and minimisation of the operating costs, by taking variations of the 6 primary design parameters, was done by the sequential quadratic programming algorithm E04UCF (Mark17) from the NAG Fortran Library. This algorithm uses a quadratic approximation for the objective function and employs linearised constraints. Drag sensitivities are obtained from response surfaces created from CFD calculations. Thus, the possibility of combination optimization parameters normally used in aircraft studies, relating to weight and aerodynamic performance with a realistic cost component. The complex, multidisciplinary nature of aerospace design problems have exposed a need to model and manage uncertainities. A new method for propagating this uncertainity to find robust design solution was developed and described in DeLaurentis' [46] study. Both the uncertainity modeling and efficient robust design technique were demonstrated on an example problem involving the design of a supersonic transport aircraft using the relaxed static stability technology. This study has been found to be an important

aspect of modern aerospace problems, where emphasis on life-cycle disciplines will introduce new uncertainities and require robust solutions.

A specific field of study in multi-disciplinary optimization is aeroelastic optimization. Barcelos [47] presented a general optimization methodolgy for fluid structure interaction problems based on turbulent flow models. The overall optimization methodology was applied to the shape and thickness optimization of a detailed wing model. The optimization results based on an inviscid and turbulent flow model were compared. Using an approriate formulation of the optimization problem, the optimization results based on the inviscid model can provide a general idea about the overall layout of the optimum wing configuration. Another example for aeroelastic optimization is Librescu's [48] study about the optimization of thin walled subsonic wings against divergence. The objective of the study was maximization of the divergence speed while maintaining the total structural mass at a constant value by using a new mathematical approach. A study of an investigation into a minimum weight optimal design and aeroelastic tailoring of an aerobatic aircraft wing structure was conducted by Guo [49]. After validating numerical model considering experimental results, by employing gradient-based optimization method the investigation was focused on aeroelastic tailoring of the wing box.

3. COMPUTATIONAL FRAMEWORK

3.1 Design Model

The design model provides an interface between the analysis model and optimization model. In general there is a relation between the physical design parameters and the abstract optimization variables. A structural or an aerodynamic parameter can be directly associated to an abstract optimization variable. Thicknesses of the structure or angle of attack of a wing could be an abstract optimization variable. In some cases, this relation becomes more complicated. Shape optimization could be an example for this approach due to the design variables of the shape of the structure or the boundary of the fluid domain.

In this study, to create parametric 3D wing model, CATIA V5 R17 software was used. CATIA (Computer Aided Three Dimensional Interactive Application) is a multi-platform CAD/CAM/CAE commercial software suite developed by the French company Dassault Systems and marketed worldwide by IBM. Written in the C++ programming language, CATIA is the cornerstone of the Dassault Systemes product lifecycle management software suite.

The software was created in the late 1970s and early 1980s to develop Dassault's Mirage fighter jet, then was adopted in the aerospace, automotive, shipbuilding, and other industries.

3.2 Analysis Model

3.2.1 FE Analysis (ABAQUS)

In the optimization process, finite element (FE), computational fluid dynamics (CFD) calculations and the coupling of these two codes are performed depending on the variation of the structural and aerodynamic variables.

In this study ABAQUS, a finite element based solver is used as the structural solver. All of the structural analyses are done by using the linear static analysis approximation. Finite element method (FEM) is based on dividing a whole structure into smaller cells. The solution procedure for a FEM in structural analysis can be given as follows;

The first step is the processing step. In this step building of the finite element model, the constraints and loads are defined. Moreover, mesh is prepared in this step. Next step is FEA solver step. In this step assembling of the model and the solving of the system of equations are done. Last step is the post-processing step. In this step the results are sorted and displayed. The equations of motion for a structure can be written as follows in a generalized way;

$$[M]\{\ddot{u}\}+[D]\{\dot{u}\}+[K]\{u\}=\{F_a\}+\{F_e\} \quad \textbf{(3.1)}$$

Where;

$[M]$: Mass matrix

$[D]$: Damping matrix

$[K]$: Stiffness matrix

$\{F_a\}$: Aerodynamic force column matrix

$\{F_e\}$: External load column matrix

$\{u\}$: Displacement column matrix

Since the analysis will be performed in static analysis the time related terms with the time derivatives of the equation (3.1) will be neglected. Moreover, in the aeroelastic analysis only the aerodynamic forces will be taken into account.

Therefore, by using the assumptions above the system of linear equations generated by the finite element method can be written as follows;

$$[K]\{u\}=\{F_e\} \quad \textbf{(3.2)}$$

Displacements and stresses induced by aerodynamic loads from the flow solver, will be calculated by ABAQUS

3.2.2 CFD Analysis (FLUENT)

In this study aerodynamic loads will be calculated by FLUENT commercial computational fluid dynamics solver. FLUENT is used for modeling fluid flow both for structured and unstructured grids by using Navier-Stokes/Euler equations [79]. A finite volume based approach is used to define the discrete equations. In our case as the flow will be in transonic regime and the compressibility effects should be taken into account the coupled solver will be used [79].

The fluid solver of the FLUENT solves the governing equations of continuity, momentum and energy simultaneously [79]. In this study, flow will be assumed as inviscid and Euler equations will be used. This is a valid approximation for high Reynolds number flows according to the Prandtl's boundary layer analysis. Moreover, according to the Barcelos and Maute [47] inviscid flow models gives acceptable results for maximizing the lift/drag optimization problems for transonic cruise conditions.

The general Euler equations, in conservation form can be written as follows;

$$\frac{\partial w}{\partial t} + \vec{\nabla}.\vec{F} = 0 \qquad \textbf{(3.3)}$$

Where $\vec{F}$ is the flux vector with cartesian components. The fluid state conservative variable, w is defined as

$$w = \begin{Bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ E \end{Bmatrix} \qquad \textbf{(3.4)}$$

Governing equations are non-linear and coupled. In FLUENT in order to get convergence several iterations are performed as the equations of continuity, energy and momentum are solved simultaneously.

3.2.3 Aeroelastic Coupling (MpCCI)

In order to couple FE code (ABAQUS) and CFD code (FLUENT), MpCCI (mesh based parallel code coupling interface) is used. MpCCI gives user the opportunity of using high fidelity simulation codes for different disciplines. The advantage of using MpCCI is that it enables the exchange of data transfer between nonmatching meshes of CFD and CSD codes in a multiphysics simulation [81]. MpCCI supports several types of coupling regions and spaces. Line, surface or volume coupling depending on the elements definitions can be done in two or three dimensional space.

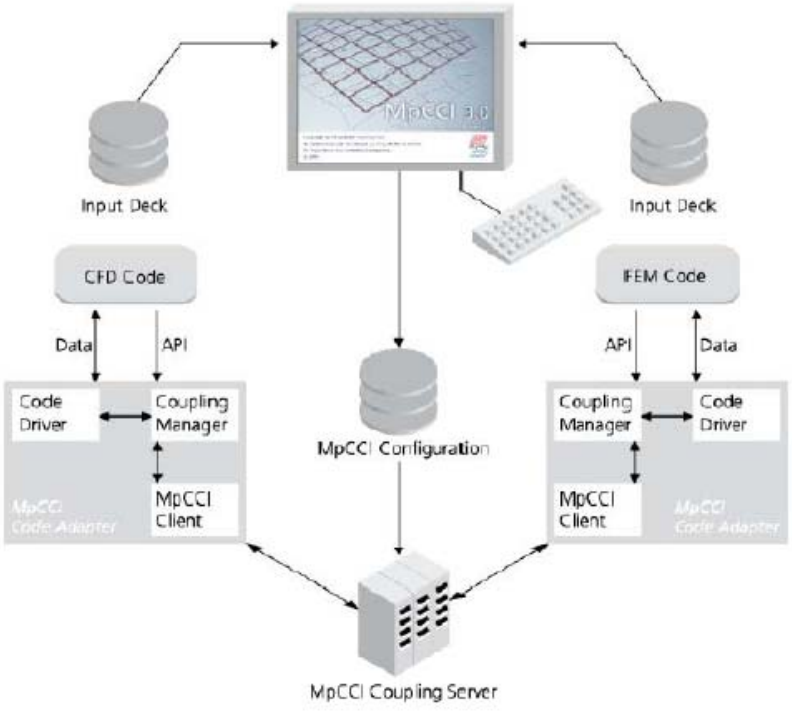

Figure 3.1 : MpCCI Coupling Process [81]

In MpCCI, data exchange process are made in three steps. First of all, to make the contact search easier the elements are split into triangles in 2D or tetrahedras in 3D.(a) Search for the elements is done by using the “Bucket Search” algorithm of MpCCI [81]. Then, each triangle is bounded by a box which includes the triangle.(b) After that step, “buckets” are formed by dividing the space into smaller squares or cubes. (c) Finally, a list is formed by listing the closer triangles to the bucket to use for the further steps.

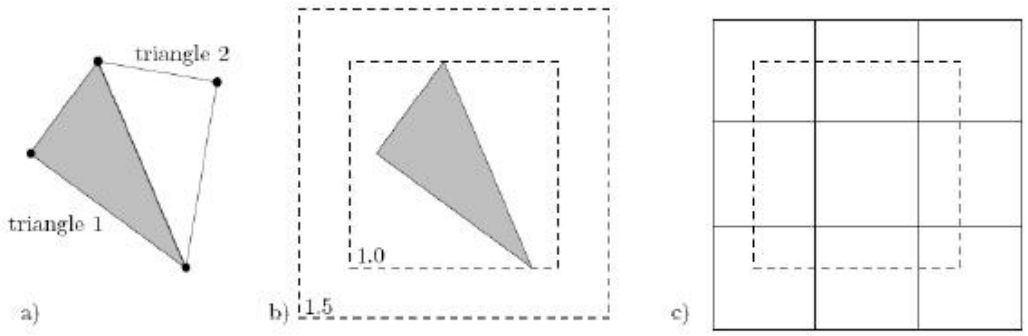

Figure 3.2 : Pre Contact Search [81]

Point-element relationships are used in the minimal distance algorithm. A list of triangles which belong to elements was formed in the pre-contact search step. In this step, the best triangle corresponding to the best element is determined and chosen. Relative positions of the triangles and the node P is used in this process [81]. Projection of the point P is taken onto the surface of each triangle.

Interpolation of the quantitites (displacement, force, pressure,…) can be done by using a flux or filed interpolation method [81]. In flux interpolation, preserving of the integral is done by adapting the value to the element sizes. This method is used for example for forces. In field interpolation, a conservative transfer is ensured by keeping the value of the elements. It is used for example for pressures.

Performing code coupling with MpCCI is done in four steps;

- Preparation of Model Files

In this step FLUENT and ABAQUS models are prepared separately. The definition of the coupling surfaces (upper wing, lower wing, tip) are given in this step. Then, model files are written in input files for the CFD and CSD codes.

- Definition of the Coupling Process

The most important step of the aeroelastic coupling process is the definition of the coupling process step. FLUENT and ABAQUS models of the wing are chosen via user interface. Then, coupling regions described above, transfer quantities (nodal displacements from the CSD code and the pressure values from the CFD code) and the coupling algorithms are selected.

- Running the Co-Simulation

In this step aeroelastic analysis are performed. MpCCI controls the rest of the coupling process till to the specified coupling iterations or time.

- Post-Processing

Finally, the results for both CFD and the CSD code are examined by using the codes own post-processing tools or the post-processing tools of MpCCI.

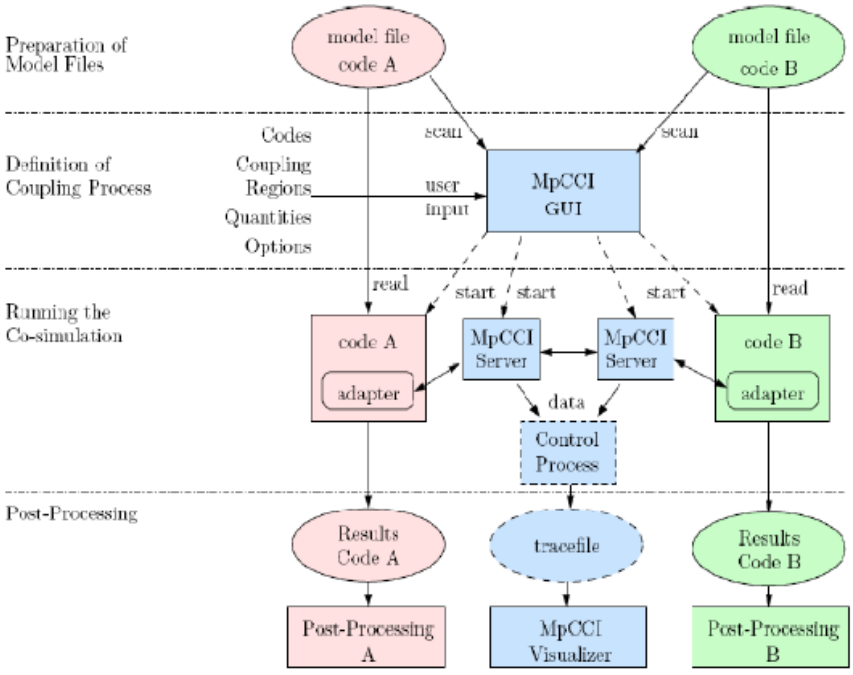

Figure 3.3 : Code Coupling Process [81]

After all the models are prepared, the solution procedure for the aeroelastic coupling can be divided into steps. The solution strategy described below is performed until a specified coupling time or iterations. CFD code calculates the surface pressures and maps these pressures as nodal forces to the CSD code. CSD code calculates the deformation of the structure under these pressure loads. Calculated nodal displacement values are mapped onto the CFD modal as mesh displacements and mesh is updated and CFD code performs the analysis.

3.3 Optimization Model

In mathematics, the simplest case of optimization, or mathematical programming, refers to the study of problems in which one seeks to minimize or maximize a real function by systematically choosing the values of variables from within an allowed

set. This is actually a small subset of this field which comprises a large area of applied mathematics and generalizes to study of means to obtain "best available" values of some objective function given a defined domain where the elaboration is on the types of functions and the conditions and nature of the objects in the problem domain.

Adding more than one objective to an optimization problem adds complexity. For example, if one wants to optimize a structural design, a design that is both light and rigid might be required. Because of the conflict of these two objectives, a trade-off exists. There will be one lightest design, one stiffest design, and an infinite number of designs that are some compromise of weight and stiffness.

A typical multi-objective optimization problem can be formulated as;

$$\begin{aligned} &\min_{s\in S} z(s) = \min_{s\in S}\left\{z_1(s), z_2(s),...., z_{n_z}(s)\right\} \\ &h(s) = 0 \qquad h(s) \in \Re^{n_h} \\ &g(s) \geq 0 \qquad g(s) \in \Re^{n_g} \\ &S = \left\{s \in \Re^{n_s},\ s_L \leq s \leq s_U\right\} \end{aligned} \tag{3.5}$$

Where s is a set of n_s abstract parameters constrained by lower and upper bounds s_L and s_U, z is the set of objective functions of the problem. h is a set of n_h equality constraints and g is a set of n_g inequality constraints.

The set of optimization variables are the parameters that affect the optimization problem. These variables can be both geometrical variables and boundary conditions. For instance, the optimization problem described in the next chapter has optimization variables that are the thicknesses of ribs,spars,skins and axialbars and material properties of an aircraft wing. However, the variables in the optimization problem performed in the fifth chapter are the thicknesses of ribs, spars, skin and axialbars and the angle of attack of the wing.

A constraint is a condition that must be satisfied during the design. Feasible design means that design satisfies the constraints, contrarily infeasible design means that the design does not satisfy the constraints. For example, designer may want the lift over drag ratio of a wing to be maximum or not want the maximum stress to exceed the

value of the material's yield stress value or an unreasonable displacement over the wing.

Objective function is the goal of the optimization problem that we want to minimize or maximize. Most of the optimization problems are single objective or can be made single objective by defining weight factors for the multi-objective functions. In the following chapters study there are two objective functions in both optimization problems and for the multiobjective optimization problems the algorithm will try to find the pareto front.

Optimization problems can be generally solved by either gradient based algorithms or evolutionary algorithms. In this study multi objective optimization problem will be solved by genetic algorithm which is an evolutionary algorithm. Evolutionary algorithms or genetic algorithms use the evolution theory to perform optimization. A population evolves over generations to adapt to an environment by selection, mutation and crossover [51]. There are three important terms related to the genetic algorithms which are fitness, individual and genes. Fitness refers to the objective function, individual refers to the design candidate and genes refer to the design variables.

Multiobjective (MO) optimization tries to find the components of a vector-valued objective function whereas the single objective optimization tries to find the single valued objective function [52]. In MO problems, solution is a set of solutions called "pareto-optimal set".

The application of the evolutionary algorithms to a MO optimization problem can be solved by using a multiobjective genetic algorithm (MOGA). Genetic algorithms are capable of finding the global optima within complex design spaces whereas gradient based algorithms can find the local optima points. Genetic algorithms can be used almost for every optimization problem, where gradient based algorithms may have some limitations. Gradient based algorithms needs the gradient information to determine the search direction that's why they need the existence of derivatives. Genetic algorithms do not need to start from a point whereas gradient based algorithms need a starting point. Genetic algorithms do not operate on design variables directly. They use binary representation of the parameters.

In this study, MOGA (multiobjective genetic algorithm) and NSGA-II (Non-dominated Sorting Genetic Algorithm) were used as optimization algorithms. NSGA-II is a fast and elitist multi-objective evolutionary algorithm. Its main features are[80]:

- A fast non-dominated sorting procedure is implemented. Sorting the individuals of a given population according to the level of non-domination is a complex task: non-dominated sorting algorithms are in general computationally expensive for large population sizes. The adopted solution performs a clever sorting strategy.
- NSGA-II implements elitism for multiobjective search, using an elitism preserving approach. Elitism is introduced storing all non-dominated solutions discovered so far, beginning from the initial population. Elitism enhances the convergence properties towards the true Pareto-optimal set.
- A parameter-less diversity preservation mechanism is adopted. Diversity and spread of solutions is guaranteed without use of sharing parameters. It is used the crowding distance, which estimates the density of solutions in the objective space, and the crowded comparison operator, which guides the selection process towards a uniformly spread Pareto frontier.
- The constraint handling method does not make use of penalty parameters. The algorithm implements a modified definition of dominance in order to solve constrained multi-objective problems efficiently.
- NSGA-II allows both continuous ("real-coded") and discrete ("binary coded") design variables. The original feature is the application of a genetic algorithm in the field of continuous variables.

On the other hand MOGA is an efficient multi-objective genetic algorithm that uses a smart multi-search elitism. This new elitism operator is able to preserve some excellent solutions without bringing premature convergence to local-optimal frontiers.

For simplicity, MOGA requires only very few user-provided parameters, several other parameters are internally settled in order to provide robustness and efficiency to the optimizer. The algorithm attempts a total number of evaluations that is equal to

the number of points in the design of experiment table (the initial population) multiplied by the number of generations.

The size of each run is usually defined by the computing resources available. A rule of thumb would suggest possibly to accumulate an initial DOE of at least 16 design configuration and possibly more than $2(n_{\mathrm{var}iable}n_{objective})$, where $n_{\mathrm{var}iable}$ is number of variable and $n_{objective}$ is number of objectives.

All of these optimization algorithms and process were employed by a commercial software called “ModeFrontier” which is a multi-objective optimization and design environment, developed to couple CAD(Computer Aided Drafting)/CAE(Computer Aided Engineering) tools, Finite Element Structural Analysis and Computational Fluid Dynamics (CFD) software. ModeFRONTIER is a GUI driven software written in Java that wraps around the CAE tool, performing the optimization by modifying the value assigned to the input variables, and analyzing the outputs as they can be defined as objectives and/or constraints of the design problem. The logic of the optimization loop can be set up in a graphical way, building up a "workflow" structure by means of interconnected nodes. Serial and parallel connections and conditional switches are available. ModeFRONTIER builds automatic chains and steers many different external application programs using scripting (DOS script, UNIX shell, Python Programming Language, Visual Basic, JavaScript, etc...) and direct integrations nodes (with many CAE/CAD and other application programs). In this study, DOS scripts were used to implement several commercial software (Fluent, Abaqus and MpCCI)

4. IDENTIFICATION OF AEROELASTIC RESEARCH WING (ARW-2)

In this chapter, the structural validation of ARW-2 (Aeroelastic Research Wing) by employing multi-objective optimization techniques in an inverse approach is considered. The objective is to identify a reliable 3-D computational ARW-2 wing model and validate it with experimental results published by NASA "Drones for Aerodynamic Structural Testing (DAST)" program. The purpose of this effort is to create an isotropic computational model of ARW-2 wing which will be used in static and dynamic aeroelastic studies. However, the structural definition of the composite wing is not complete in literature. The thicknesses of ribs, spars, skin and axial bars of the wing are missing geometrical properties. Furthermore the material data given in the literature is not enough to establish a composite FE model of wing's skin. To remedy these deficiencies, a computational model which will have the similar structural response with the experimental wing is required. In the first stage of this research, an isotropic skin approach is used for simplicity. The errors in the first five modal frequencies, mode shapes, pre-defined static bending and torsional responses of the wing model is minimized simultaneously as the objectives of a multi-objective optimization problem. The missing material properties and missing thicknesses of the skin, ribs, axial bars and spars are computed as optimization variables and identified in an inverse approach. In the second step; the computational ARW-2 model's structural response is validated with the experimental results. In this study, commercial software "ModeFrontier" is used as a multi-disciplinary optimization tool, "Abaqus" as a FE solver.

Many research studies used ARW-2 model for aeroleastic code validation purposes. Sandford provided geometrical and structural properties of ARW-2 experimental wing model[53]. In another study of him the steady state pressure measurements on the wing model were presented[54]. Other than these two important studies, there are also many studies that presents experimental data and background about the ARW-2 wing model [53-73]. By using these experimental data, a few computational studies

were performed. In Bhardwaj's Ph.D. thesis, an aeroelastic coupling procedure was developed which performs static aeroelastic analysis using CFD and CSD code with little code integration[4]. ARW-2 wing model was used for demonstration of the aeroelastic coupling procedure by using ENSAERO (NASA Ames Research Center CFD code) and a finite element wing-box code which was developed as a part of his study. The results were compared with experimental data from an experimental study conducted at NASA Langley Research Center. In his study, Bhardwaj created ARW 2 wing model with isotropic skin instead of composite skin. In present thesis, like Bhardwaj's approach, ARW-2 wing model is created with isotropic skin for simplicity. The wing model is validated both with the experimental and computational data which was presented by Bhardwaj. However, the thicknesses of ribs, spars, skin and axial bars of the wing are missing geometrical properties. Furthermore the material data given in the literature is not enough to establish a composite FE model of the wing's skin.

4.1 Geometrical and Structural Properties of ARW-2

At NASA Langley Research Center, "Drones for Aerodynamics and Structural Testing – DAST" program intended to generate an extensive database of measured steady and unsteady pressures for a supercritical wing model so that these results could be used in computational studies for validation purposes. At the beginning of the program, wing models were produced as rigid as possible in order to provide simple comparisons for transonic aerodynamic computations. Next, a flexible wing configuration was tested as part of this NASA program. Increasing flexibility of the experimental wing provided more realistic data for comparison of aeroelastic computational results with measurements.

The elastic wing configuration is known as DAST ARW-2 which has an aspect ratio of 10.3, a leading-edge sweepback angle of 28.8^{o}, and a supercritical airfoil. It is produced with two inboard and one outboard trailing-edge control surfaces. Only the outboard control surface was deflected to generate steady and unsteady flow over the wing. The wing contour was performed from three different supercritical airfoils. The wing primary structure consists of two main spars, one of which is at 25 % and the other at 62 % of local chord. Ribs were placed perpendicular to the rear spar every 13.2 in. except for the outboard wing-tip rib, which is also served as a spar end

fitting. The spars and ribs were machined from 7075-T73 aluminum alloy. The wing skin was made of fiberglass material with honeycomb panels sandwiched between the middle two layers of fiberglass for areas of skin not located over the spars or ribs. The number of layers of fiberglass used to make the skin varied from 13 at the inboard end to 27 at the outboard end, with approximately 25 % of the layers at ± 45 deg orientation. Figure 4.1 shows the wing in the wind tunnel. Figure 4.2 shows right and top views and Figure 4.3 demonstrates the planform area of the wing.

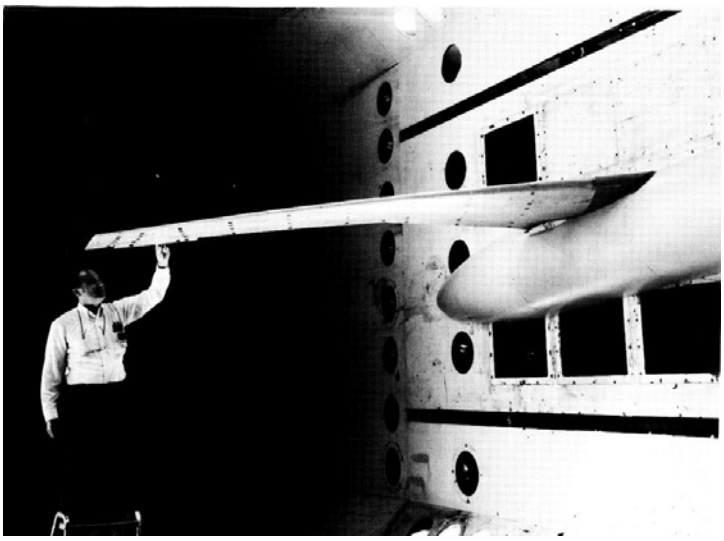

Figure 4.1 : Aeroelastic Research Wing (ARW-2) [53]

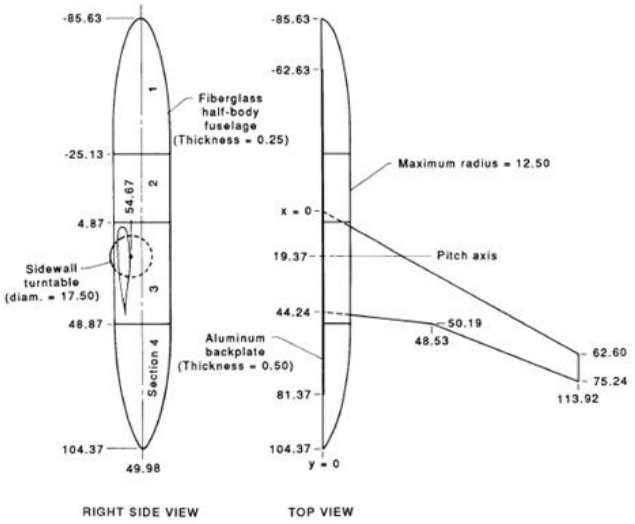

Figure 4.2 : Right Side and Top Views of the wing [53]

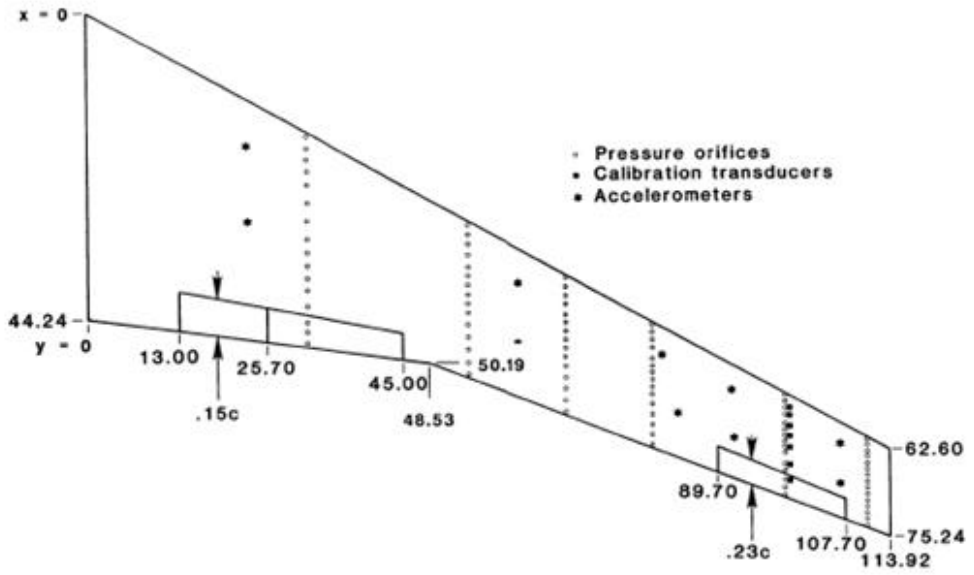

Figure 4.3 : Planform Area of the wing [53]

4.2 ARW-2 Wing Finite Element Model

ARW-2 wing geometrical model has been created by using CATIA V5 R17 software. The wing has three different supercritical airfoils. Through the coordinate data of the airfoils given in a NASA Technical Report of Sandford [53], firstly supercritical airfoils were created. Skin of the wing was created by assembling airfoils via "*Generative Shape Design*" module of CATIA V5 R17. Also, the ribs and spars are located according to the coordinates given in the NASA Technical Report [53] as it is illustrated in Figure 4.4 and 4.5. In Figure 4.6 and 4.7, wing surface model and structural model created in CATIA V5 are shown.

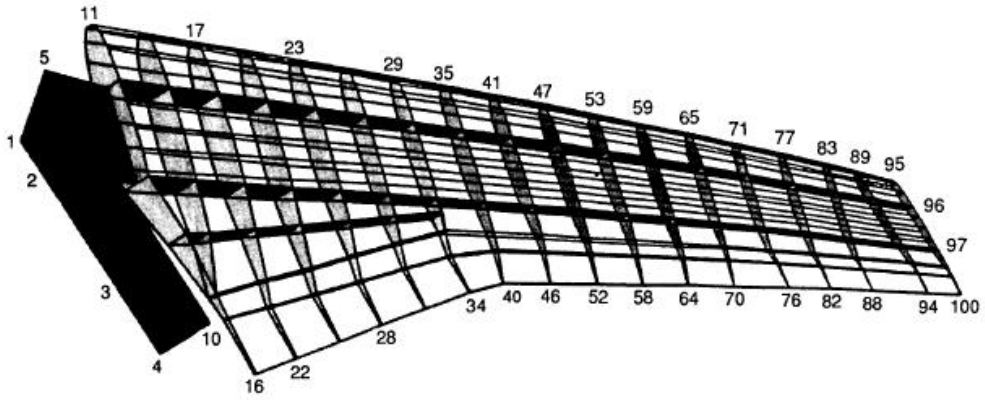

Figure 4.4 : Structural Model of the Wing [53]

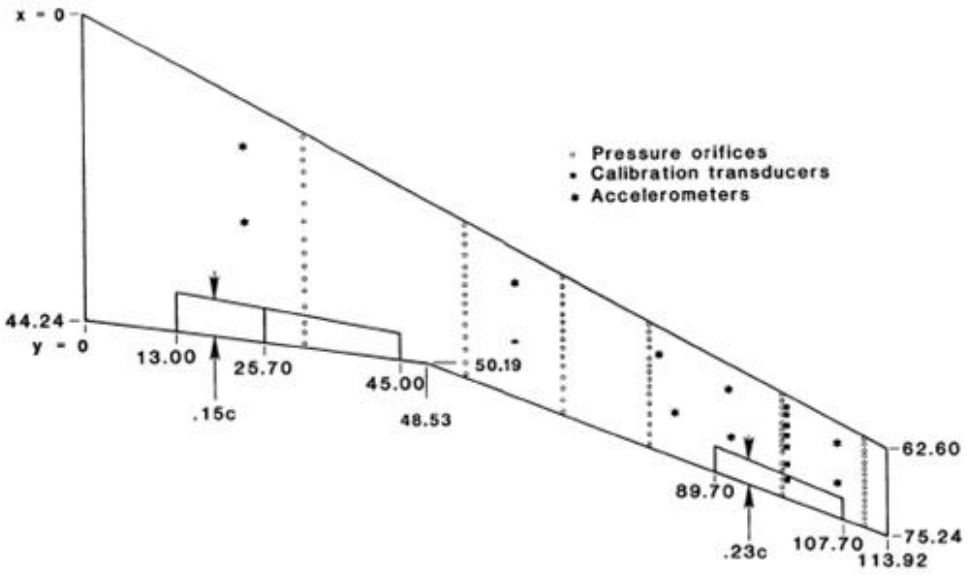

Figure 4.5 : Locations of ribs and spars [53]

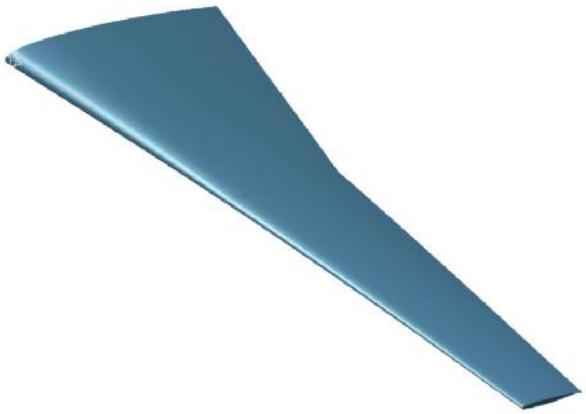

Figure 4.6 : 3D CAD model of the wing created in CATIA

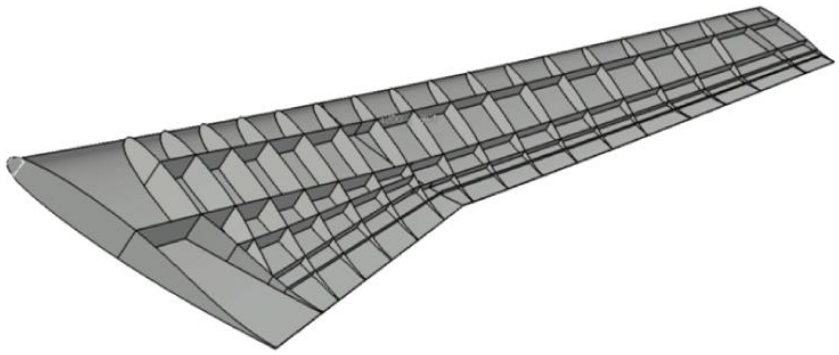

Figure 4.7 : Inner strucutral model of ARW-2 wing

Finite element (FE) model of the wing created in Abaqus 6.7-1 has 26,000 quadrilateral elements. As discussed before, the thicknesses of ribs, spars, skin and axial bars of the wing are missing geometrical properties. Furthermore, the material data given in the literature is not enough to establish a 3-D FE model of wing's composite skin. All these missing geometrical and material properties are defined as "variables" in Abaqus parametrically. The aim is to reach a computational model, which will have the same structural response with the experimental wing by iterating these variables. This leads to an inverse engineering problem where the benefits of numerical optimization methods can be used conveniently.

4.3 Application of Multi-Objective Optimization

In the first stage of this effort, instead of a composite skin model, an isotropic skin approach was used for simplicity. In order to obtain a reliable computational wing model, an inverse engineering optimization problem is set. For structural validation purpose, the modal frequencies, mode shapes, pre-defined static bending and torsional responses are considered. In the optimization problem, the objective is to minimize the average of relative errors in first five modal frequencies and in static bending displacements at wing tip on front and rear spars. This leads to a multi objective optimization problem with 2 objectives.

Optimization variables are defined as the material properties such as Young Modulus, mass density and Poisson's ratio and the missing geometrical properties of wing such as thicknesses of the ribs, axial bars and spars. The optimization problem is formulated as;

Objective Functions : $\min(z_1)$, $\min(z_2)$

$$z_1 = \frac{\sum_{i=1}^{5} \frac{\left| f_i^{\exp} - f_i^{comp} \right|}{f_i^{\exp}}}{5} \text{x}100$$

$$z_2 = \frac{\frac{\left| u_{forward}^{comp} - u_{forward}^{\exp} \right|}{u_{forward}^{\exp}} + \frac{\left| u_{rear}^{comp} - u_{rear}^{\exp} \right|}{u_{rear}^{\exp}}}{2} \text{x}100$$

Constraints : $r_l^{axial} > t_l^{axial}$

Optimization Variables : t_i^{rib} $, i = 1,2,....,17$

t_j^{spar} $, j = 1,2,....,5$

t_l^{axial} $, l = 1,2,....,4$

r_l^{axial}

$E_{skin}, E_{spar}, E_{axial}, E_{rib}$

$m_{skin}, m_{spar}, m_{axial}, m_{rib}$

$\nu_{skin}, \nu_{spar}, \nu_{axial}, \nu_{rib}$

Where t_i^{rib}, t_j^{spar}, t_l^{axial} are thicknesses of ribs, spars and axial bars respectively and r_l^{axial} is the radius of axial bars. $E_{skin}, E_{spar}, E_{axial}, E_{rib}$ are young modulus, $m_{skin}, m_{spar}, m_{axial}, m_{rib}$ are mass density and $\nu_{skin}, \nu_{spar}, \nu_{axial}, \nu_{rib}$ are the poisson's ratio of skin, spars, axial bars and ribs.

The reference for modal frequencies and shapes are taken from the NASA Technical Report and the displacements are taken from Bhardwaj's Ph.D. thesis [4]. Errors of modal frequencies and pre-defined static bending response of the wing model are

minimized simultaneously as the objectives of this multi-objective optimization problem.

The missing material properties and missing thicknesses of the ribs, axial bars and spars are computed as optimization variables and identified in an inverse approach. This optimization problem is set in commercial software ModeFrontier which is used as a multi-disciplinary optimization tool. FE solver Abaqus is used to compute modal frequencies and displacements. Figure 4.8 shows ModeFrontier flow chart which is developed for this optimization problem.

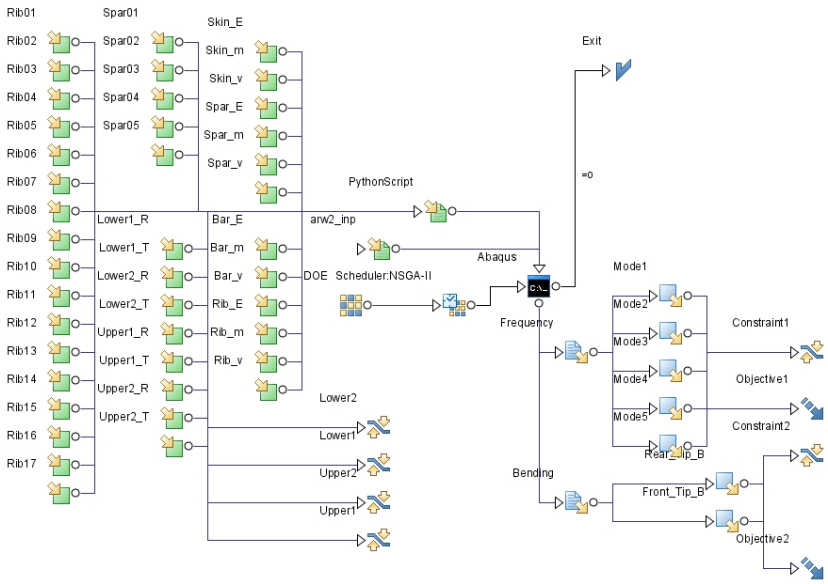

Figure 4.8 : ModeFrontier Flow Chart

In Figure 4.8, variables of the optimization problem are located at the left side of the figure. In the center of the figure, Abaqus node is located to perform both modal analysis and static analysis. Modal analysis computes first five natural frequencies while static analysis computes the displacements of ARW-2 wing with a 100 lb applied upward at the wing tip on the front spar. Objectives are located at the right side of the Figure 4.8. The computational and experimental results are compared with the modal and bending behaviors of the wing by minimization of the error percentages between computational and experimental data. MOGT (Multi objective Game Theory) and NSGA II (Non-dominated Sorting Genetic Algorithm II) are used as optimization algorithms.

4.4 Results

First of all, MOGT (Multi objective Game Theory) is used and pareto optimal set among 145 designs are taken as DOE (Design of Experiment) for the further optimization analysis with NSGA II (Non-dominated Sorting Genetic Algorithm II). By using NSGA II, 5000 different designs are evaluated and in table 4.1 pareto optimal set is tabularized. Error of natural frequency is considered as major objective. In order to arrive at a decision, weight of relative error of natural frequency and relative error of wing tip displacement are considered as 60% and 40%, respectively. In this point of view, among the paretos shown in table 4.1, one of them is selected as the best design. Table 4.2 shows the best design with objective functions and their errors.

Table 4.1: Paretos

Paretos	Relative Errors of Natural Frequencies	Relative Errors of Wing Tip Displacements
1	2.73 %	7.44 %
2	2.86 %	7.69 %
3	3.10 %	7.69 %
4	2.75 %	7.68 %
5	3.18 %	7.98 %
6	3.45 %	7.78 %
7	2.74 %	7.72 %
8	2.47 %	7.72 %
9	1.49 %	7.72 %
10	1.05 %	7.73 %
11	3.83 %	4.93 %
12	3.23 %	5.78 %
13	**1.01 %**	**6.60 %**

Table 4.2: Optimum Design and Relative Errors

	Experimental Data	Computational Data	Relative Error
Mode 1	8.31 Hz.	7.83 Hz.	5.70 %
Mode 2	31.28 Hz.	31.08 Hz.	0.64 %
Mode 3	36.00 Hz.	36.52 Hz.	1.44 %
Mode 4	62.37 Hz.	62.32 Hz.	0.08 %
Mode 5	66.97 Hz.	66.92 Hz.	0.07 %
Forward Spar Wing Tip Displacement (in)	1.90 Inch	1.80 Inch	5.26 %
Rear Spar Wing Tip Displacement (in)	2.01 Inch	1.85 Inch	7.96 %

Although the error percentages are satisfactory, mode shapes and torsional response of the wing model has to be verified in order to prove the reliability of the wing configuration according to the optimum design. Therefore, mode shapes which are published in the NASA Technical Report are compared with those of the found optimum design [53]. Figure 4.9 shows the experimental wing's mode shapes and Figure 4.10 shows the computational wing's mode shapes for comparison.

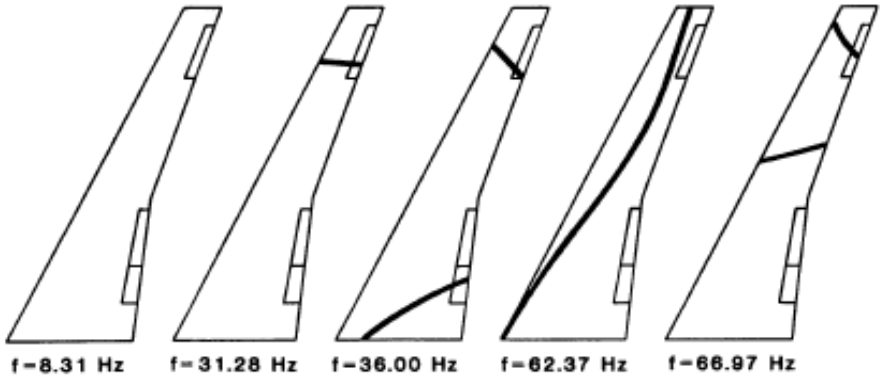

Figure 4.9 : Mode Shapes of the Experimental Wing

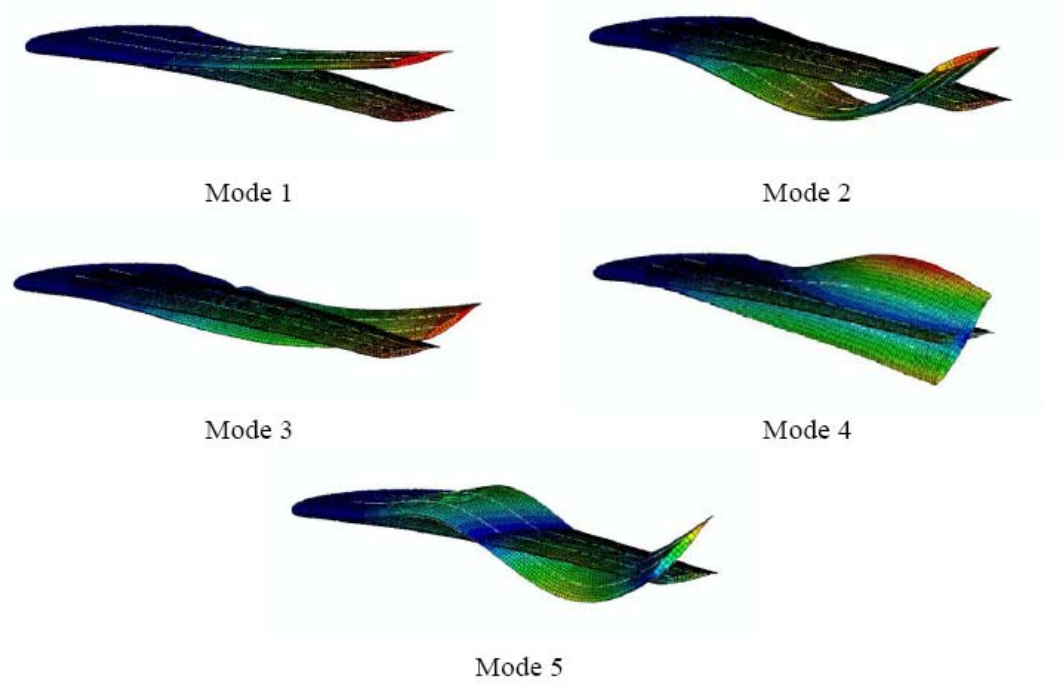

Figure 4.10 : Mode Shapes of the Computational Wing

Bending is one of the most important structural criteria that has to be validated for the computational wing model. Bhardwaj, in his Ph.D. thesis has validated ARW-2 wing model with isotropic wing skin by comparing bending and twisting behaviors of the composite skin ARW-2 wing [4]. Figure 4.11 shows the comparison between the displacements of the front spar of the composite skin and isotropic ARW-2 wings with a 100 lb load applied upward at the wing tip on the front spar, besides Figure 4.12 shows the displacements of the front spar of ARW-2 computational wing model. Figure 4.13 shows the displacements of the rear spar of the composite skin and isotropic ARW-2 wings with a 100 lb load applied upward at the wing tip on the front spar, and the Figure 4.14 shows the displacemets of rear spar of ARW-2 computational wing model. The bending behavior of ARW-2 wing model produced at this study is coherent with the behavior of composite ARW-2 wing and Bhardwaj's wing model. On the other hand, torsional responses of the wing model are compared with the experimental results. Displacements of the front and rear spars of the ARW-2 wing with 1 lb applied upward at the wing tip on the front spar and 1 lb applied downward at the wing tip on the rear spar has been calculated for the identified wing model. To compare torsional responses of the wing model, Figure 4.15, 4.16, 4.17 and 4.18 show the deflections of the front and rear spars of the Bhardwaj's isotropic, composite skin and identified ARW-2 wing model. Figure 4.15 and 4.16 show the displacements of the front spar while figure 4.17 and 4.18 show the displacements of the rear spar. A good agreement is obtained for both bending and torsional behaviors of ARW-2 wing model.

In this chapter, ARW-2 finite element wing model has been created and missing properties are identified in an inverse engineering approach by using multi-objective optimization tools. Inverse engineering problem is implemented successfully by comparing modal and static bending responses of the wing. Furthermore, mode shapes and torsional responses of wing has been compared with literature to prove the reliability of ARW-2 wing model profoundly. It can be concluded from this chapter that ARW-2 wing model is enabled to be used for further studies.

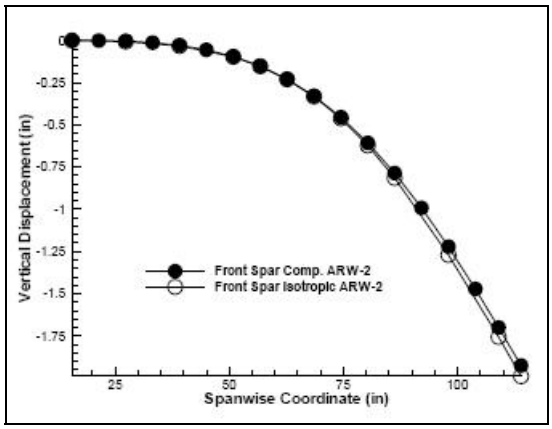

Figure 4.11 : Displacement of the Front spar of the Composite Skin and the Isotropic Wing Subjected to a 100 lb Vertical Load Applied at the tip [4]

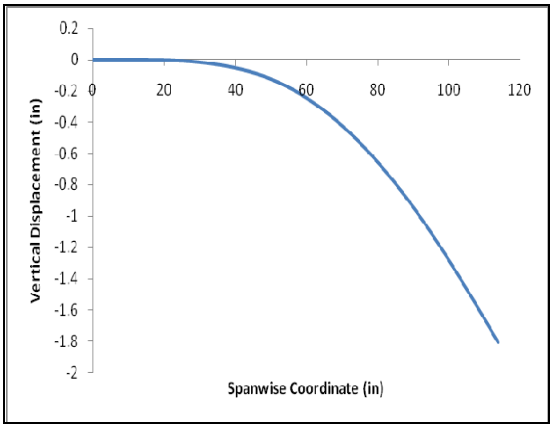

Figure 4.12 : Displacement of the front spar of the Isotropic Wing Model Subjected to 100 lb Vertical Load Applied at the tip (Present Study)

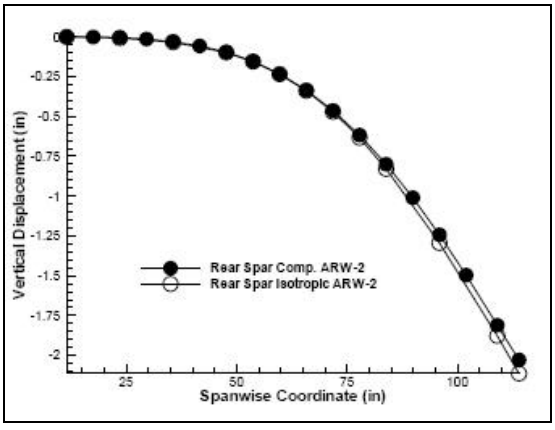

Figure 4.13 : Displacement of the Rear Spar of the Composite Skin and the Isotropic Wing Subjected to a 100 lb Vertical Load Applied at the tip [4]

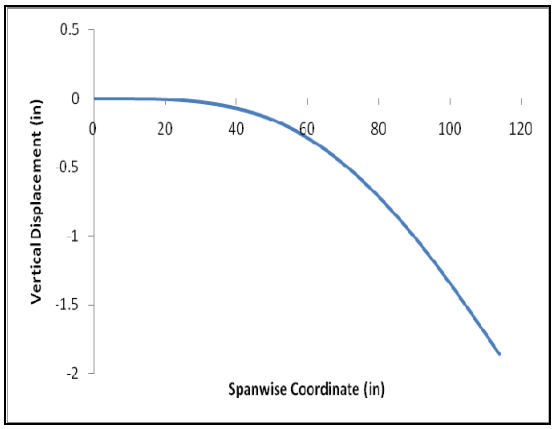

Figure 4.14 : Displacement of the Rear Spar of the Isotropic Wing Model Subjected to 100 lb Vertical Load Applied at the tip (Present Study)

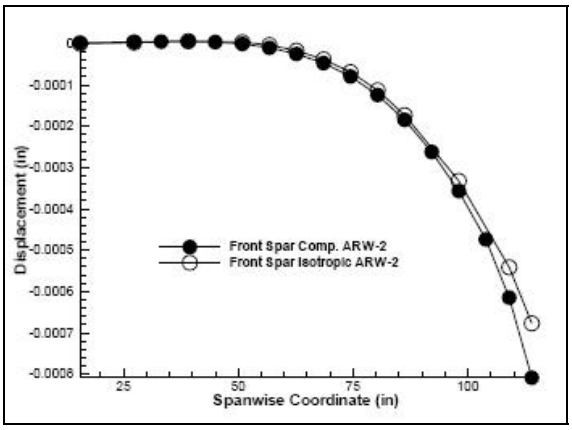

Figure 4.15 : Displacement of the Front Spar of the Composite Skin and the Isotropic Wing Subjected to a Twisting Load Applied at the tip [4]

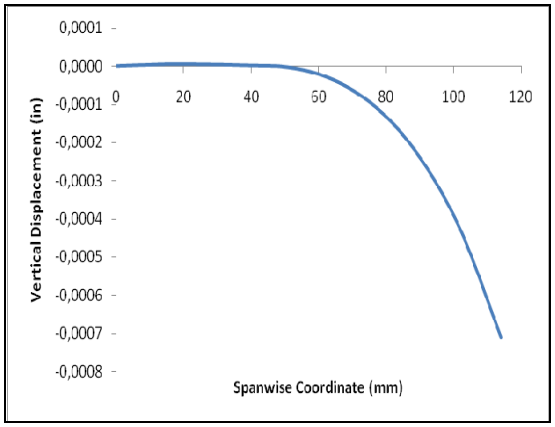

Figure 4.16 : Displacement of the Front Spar of the Isotropic Wing Model Subjected to a Twisting Load Applied at the tip (Present Study)

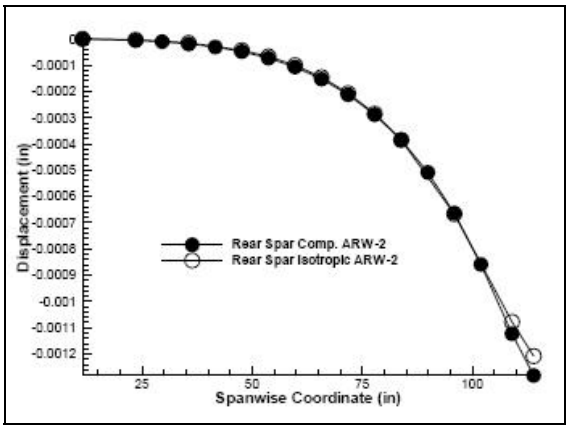

Figure 4.17 : Displacement of the Rear Spar of the Composite Skin and the Isotropic Wing Subjected to a Twisting Load Applied at the tip [4]

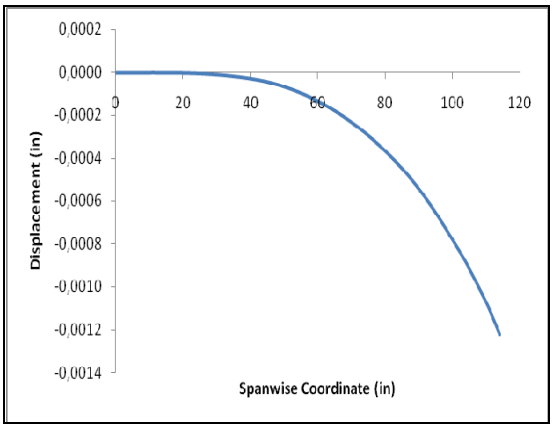

Figure 4.18 : Displacement of the Rear Spar of the Isotropic Wing Model Subjected to Twisting Load Applied at the tip (Present Study)

5. STATIC AEROELASTIC ANALYSIS AND OPTIMIZATION OF ARW-2 WING MODEL

5.1 Static Aeroelastic Analysis and Validation

In the previous chapter, missing material properties and thicknesses of ribs, spars, skin and axial bars of computational ARW-2 wing model has been identified as an inverse engineering problem by using multi-objective optimization tools. After structural and aerodynamic responses of ARW-2 isotropic model have been validated, static aeroelastic analysis is needed to be performed to ensure the reliability of the computational model. Experimental data from the work of Sandford et al. [54] is used for comparison of aeroelastic displacements at the wing tip. Several boundary conditions with 0.8 Mach number and -1 to 3 degrees angle of attack are used for static aeroelastic computations.

In aeroelastic analysis, mesh based parallel code coupling interface MpCCI 3.0.6 is used to exchange the pressure and displacement information between Fluent and Abaqus for a two-way loose coupling at each aeroelastic iteration.

All of the structural analyses are done by linear static analysis approximation with 26,000 hexahedral elements. The equations of motion for a structure can be written as follows in a generalized way;

$$[M]\{\ddot{u}\}+[D]\{\dot{u}\}+[K]\{u\}=\{F_a\}+\{F_e\} \quad \textbf{(5.1)}$$

Since the analysis will be performed as static analysis the time terms with the time derivatives of equation 5.1 will be neglected. Moreover, in the aeroelastic analysis only the aerodynamic forces will be taken into account. Therefore, the system of linear equations generated by the finite element method can be written as follows;

$$[K]\{u\}=\{F_a\} \quad \textbf{(5.2)}$$

Displacements will be calculated by Abaqus-6.7.1 [78] via using the aerodynamic loads calculated from the flow solver, Fluent [79]. In this study MpCCI [81] (Mesh based Parallel Code Coupling Interface) is used as an aeroelastic coupling interface. MpCCI gives the opportunity to couple high fidelity simulation codes for multi physics simulations. The advantage of using MpCCI is that it enables the exchange of data transfer between nonmatching meshes of CFD and CSD codes in a multi physics simulation. In this study a staggered algorithm is used for the aeroelastic coupling which is shown in figure 5.1.

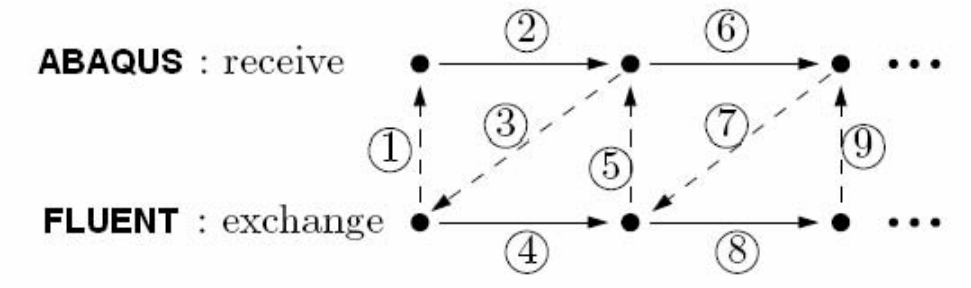

Figure 5.1 : Staggered Algorithm for the Aeroelastic Coupling

To validate the aeroelastic wing model, static aeroelastic analyses with five different flow conditions are performed via MpCCI. The CFD and CSD models of the wing are introduced to MpCCI and transfer quantities (surface pressures from the CFD and nodal displacements from the CSD) were selected. In order to comprehend that the solution is converged, the residuals of continuity, x-velocity, y-velocity, z-velocity and energy equations calculated by Fluent are plotted in Figure 5.2. The figure shows that the solutions converged since the residuals are very low. The aeroelastic results are compared with the measurements of Sandford et al. [54] in which the wing tip displacements were given. Figure 5.3 shows wing-tip deflections for both experimental data and results of computational analyses at Mach number 0.8 and -1 to 3 degrees angle of attack.

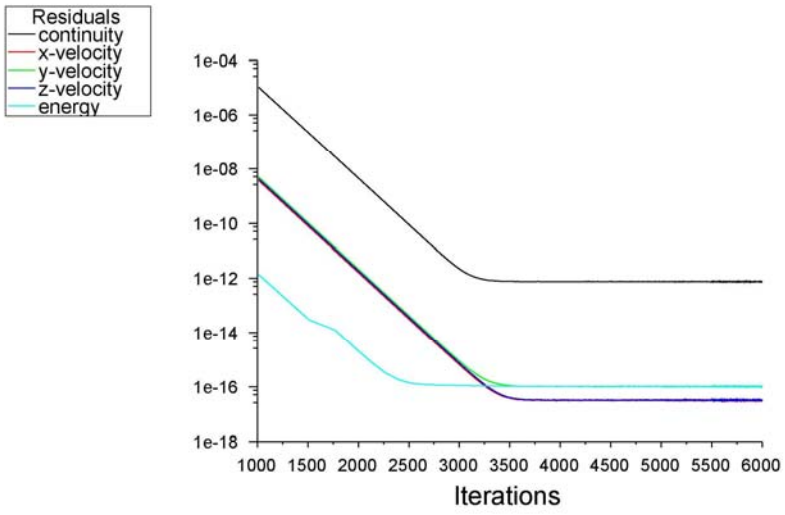

Figure 5.2 : Residuals

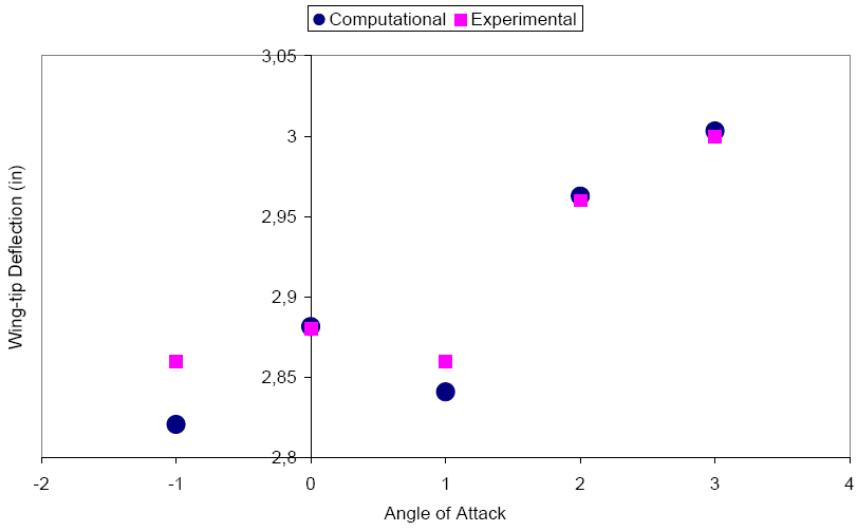

Figure 5.3 : Wing-tip Deflections at M = 0.8

Table 5.1 shows the relative errors for computational model relative to experimental data. The correlation of the static aeroelastic response is remarkably good.

Table 5.1: Relative Errors Related to Static Aeroelastic Response

Angle of Attack	Experimental Data (mm)	Computational Data (mm)	Relative Error
-1	2.86	2.821	1.37 %
0	2.88	2.881	0.05 %
1	2.86	2.841	0.07 %
2	2.96	2.963	0.09 %
3	3.00	3.003	0.10 %

5.2 Static Aeroelastic Optimization

The structurally and aeroelastically validated ARW-2 wing isotropic model will be optimized in a multiobjective and multi-disciplinary optimization framework developed for steady-state conditions while both aerodynamic and structural parameters can be used as optimization variables. Numerical optimization is an iterative scheme to reach the most desired design within a design space bounded by the lower and upper limits of optimization variables[76].

For aeroelastic optimization problems, the structural and aerodynamic optimization parameters can be cross-sectional and thickness dimensions of the structural elements, the shape optimization parameters such as sweep, dihedral, and twist angle of the wing, control nodes in the design elements representing the geometry, and the Mach number and angles of attack. The optimization criteria q in equation 5.3 which include the objective functions and the constraints can cover the aerodynamic performance factors such as lift and drag, as well as the structural behavior descriptors such as mass, displacements, stresses and strains. In general, for aeroelastic optimization, the optimization criteria q depends on the aeroelastic response of the system characterized by the structural displacement vector u and the fluid state vector w, which in turn are functions of the physical design parameters, or rather the abstract optimization variables s. Therefore,

$$q = q(s,u,w)$$
$$u = u(s); \qquad w = w(s) \tag{5.3}$$

Optimization studies require a high number of sequential analyses automatically and needs longer computational time as compared to only-analysis studies. For that reason, a serious research has been focusing on development of more efficient optimization algorithms for problems with large analysis size. For only optimization purposes simpler analysis models can be preferred in the iterative process and parametric geometries can be used to reduce the number of optimization variables that can sufficiently describe a problem. In this study, the multi-objective and multidisciplinary optimization methodology which was presented in the work of Nikbay et al. [76] will be extended to ARW-2 wing configuration [74].

5.2.1 Formulation of Optimization Problem

For the optimization of ARW-2 wing, thicknesses of ribs and spars and angle of attack are selected as design variables. There are two objective functions which are maximizing the lift over drag ratio and minimizing the weight of the wing. There are three aeroelastic constraints. The first constraint emposes that the maximum deformation of the wing tip should not exceed 0.381 m. The second constraint controls that the lift L can change only by an amount greater or equal to the variation of the weight induced by the variation of the thicknesses of structural elements. The third constraint limits that the Von Mises stress should not exceed the yield strength ($9.05*10^7$ Pa). The overall optimization problem can be formulated as;

$$\min_{s\in S}\{M(s)\}, \qquad \max_{s\in S}\left\{\frac{L}{D}(s)\right\} \tag{5.4}$$

$$g_1(s)=\frac{2\Delta W}{\Delta L}-1\le 0 \qquad g_1(s)\in\Re \tag{5.5}$$

$$g_2(s)=\frac{u_{\max}}{0.381}-1\le 0 \qquad g_2(s)\in\Re \tag{5.6}$$

$$g_3(s)=\frac{\sigma_{\max}}{9.05\text{x}10^7}-1\le 0 \qquad g_3(s)\in\Re \tag{5.7}$$

$$s=(\alpha,k_1,k_2,k_3) \qquad -2\le\alpha\le 6 \qquad -0.25\le k_1,k_2,k_3\le 0.25 \tag{5.8}$$

where $M(s)$ is the total mass of the wing, $\frac{L}{D}(s)$ is the lift over drag value for the wing, $u_{\max}$ is the maximum deflection at the wing tip, ΔL is the change in the lift of the wing, ΔW is the change in the weight of the wing, $\sigma_{\max}$ is the maximum von misses stress, α is the angle of attack. k_1, k_2 and k_3 are the abstract optimization parameters used to vary the thicknesses of ribs (t_i , t_j) and thicknesses of spars (T_k). To reduce the number of optimization variables, 17 ribs are divided into 2 groups as; first half has index i and second half has index j. At the $(n+1)^{\text{th}}$ optimization iteration the values of the design variables are updated as;

$$t_i^{(n+1)} = t_i^{(0)} + t_i^{(0)}.k_1^{(n+1)} \qquad i = 1,2,...,7 \tag{5.9}$$

$$t_j^{(n+1)} = t_j^{(0)} + t_j^{(0)}.k_2^{(n+1)} \qquad j = 8,9,...,17 \tag{5.10}$$

$$T_k^{(n+1)} = T_k^{(0)} + T_k^{(0)}.k_3^{(n+1)} \qquad k = 1,2,....,5 \tag{5.11}$$

where $t_i^{(0)}$, $t_j^{(0)}$ and $T_k^{(0)}$ are the thickness values for the ribs and spars at the initial design. Modefrontier software has both gradient based and gradient-free algorithms. In this work multi-objective genetic algorithm MOGA is chosen as the optimization driver[52]. The scheduler used in this study is the implementation of Deb [75] which is "Non-dominated Sorting Genetic Algorithm NSGA-II" which is a fast and elitist multiobjective evolutionary algorithm[80].

5.2.2 Optimization Framework

For the aeroelastic optimization problem several commercial softwares were coupled during the optimization process in this work. Fluent-6.3.26 and Abaqus-6.7.1 are used to solve inviscid 3D Euler equations and to compute the structural response of the aeroelastic system respectively. Mesh based parallel code coupling interface MPCCI-3.0.6 is used to exchange the pressure and displacement information between Fluent and Abaqus. Modefrontier 4.1 is used as a multi-objective and multidisciplinary optimization software. In order to perform an optimization study a workflow should be prepared in Modefrontier to govern the optimization process. In this workflow the optimization variables (with their upper and lower bounds and incrementations), scheduler, design of experiments, objectives, constraints, output variables and the softwares are defined. Optimization workflow is prepared to automate the multiobjective multidisciplinary optimization problem. Once the workflow is run, it controls the optimization process automatically by using the well prepared script files and models. Figure 5.4 shows the work flow of this optimization problem. All script files prepared for this optimization problem are given in Appendix.

In this work flow, Modefrontier's script files drive Abaqus and Fluent codes in batch mode. In each optimization iteration, Modefrontier updates the thickness parameters of the wing and create a new input file for Abaqus and also updates the angle of attack parameter and creates a new case file for Fluent through the journal file of

Fluent. The weight criteria is a direct output of Abaqus node. However, during fluid structure iteraction iterations, MpCCI exchanges the displacement and pressure values between Abaqus and Fluent before Abaqus evaluates the displacement criteria finally.

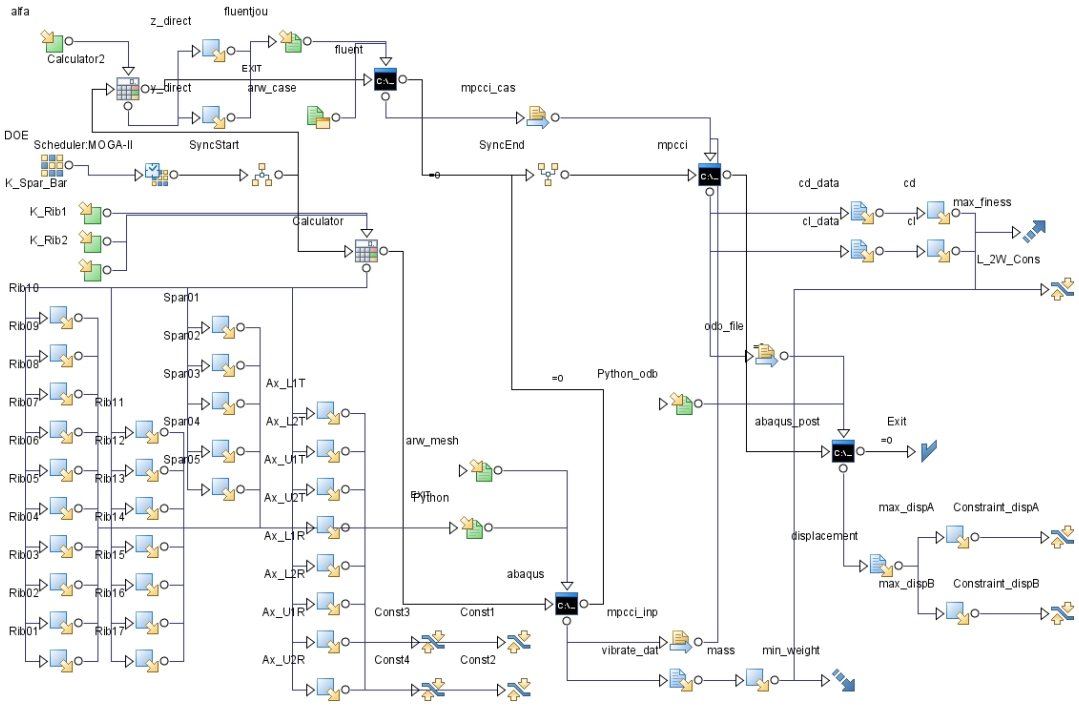

Figure 5.4 : Workflow of the optimization problem

5.2.3 Optimization Results

6 design of experiments (DOE) with "Sobol sequence" are used and 100 generations for the MOGA-II are defined. Sobol sequence distributes the experiments uniformly in the design space [80]. Finally, a total number of 129 designs are generated for the optimization problem. Solution of the problem took 188 hours on a workstation with Intel(R) Core(TM)2 CPU 6700 @ 2.66 GHz processor, with 2 GB of RAM on Microsoft Windows XP operating system. 36 designs were found to be feasible that satisfy the constraint condition given in the optimization problem and 12 designs are unfeasible that did not satisfy the constraint condition. Moreover, there are 9 error designs that did not give any solution because of modeling or computational errors resulted in the optimization workow. As a result, 2 designs are found in the pareto front set for this optimization problem. These pareto designs are tabularized in table 5.2. First of the pareto designs is preferred as the best wing configuration due to

lower W value. Table 5.3 shows a comparison between the initial and optimum design of ARW-2 wing.

Table 5.2: Paretos

Angle of Attack	k_1	k_2	k_3	$\frac{L}{D}$	Weight (W) (Newton)
0	-0.22	-0.25	-0.25	11.836	507.42
0	-0.08	-0.23	-0.25	11.836	561.81

Table 5.3: Comparison of optimum and initial designs of ARW-2

	Angle of Attack	k_1	k_2	k_3	$\frac{L}{D}$	Weight (W) (Newton)
Initial Design	2.5	0	0	0	10.76	597.284
Optimum Design	0	-0.22	-0.25	-0.25	11.836	507.42
Improvements					% 9.09	% 15.04

5.3 Conclusion

Firstly, aeroelastic response of ARW-2 computational wing model with isotropic skin is verified with experimental data of the real ARW-2 wing model which had composite skin. Secondly, an aeroelastic optimization study is performed on ARW-2 wing geometry to increase lift over drag ratio and decrease the weight of the wing while imposing constraints on wing tip displacement, maximum Von Mises stress and the relation between lift and weight. Angle of attack and thicknesses of ribs and spars are used as optimization parameters. Moreover, a multi-objective genetic algorithm MOGA-II is used to control the optimization process. At the end of the aeroelastic optimization study, the solution for a pareto-optimum set is given and the best design configuration has been chosen. In this study, commonly used commercial analysis tools such as Gambit, Fluent, Abaqus are employed as structural and flow solvers. Thus, a strong and easy to apply multi-disciplinary optimization methodology is successfully applied by implementing advanced optimization techniques directly into the everyday used tools employed in aerospace industry. A loose coupling software MpCCI is utilized for fluid-structure interaction while advanced optimization software Modefrontier is used for MDO.

6. CONCLUSION AND RECOMMENDATIONS

One of the purposes of this study is to identify a complete configuration of ARW-2 computational wing model for further use. Due to its missing properties in literature, FE structural model of the wing is determined by an inverse engineering through optimization. Thus, an accurate and validated ARW-2 computational wing model is created with its identified properties such as material and geometrical properties.

Another purpose of the study is to develop a methodology for static aeroelastic optimization problems by using commercial codes. After the ARW-2 computational wing model is identified and validated successfully with the experimental results, an optimization problem which constraints wing tip deflection, maximum stress and a relation between lift and weight is set concerning a couple process of CSD and CFD codes in transonic regime. Both the validation and optimization results are satisfactory.

6.1 Application of The Work

In this thesis, structural and aeroelastic validation and aeroelastic optimization studies were performed by using highly complicated experimental wing model of ARW-2 wing. In chapter 4, the missing geometrical and material properties of this wing were identified by using multi-objective optimization tools. ARW-2 wing model was generated by using a CAD software CATIA. As an optimization driver, an advanced optimization software ModeFrontier was used via NSGA II optimization algorithm. As FE solver, Abaqus 6-7.1 was employed to calculate structural response of the wing. Although the structural analysis was performed in Abaqus, HyperMesh was used to generate structural grid.

In chapter 5, an aeroelastic optimization problem for which ModeFrontier was used was set. Considering transonic regime (M = 0.8), a coupling software MpCCI was implemented to the optimization process. Abaqus was used to perform structural

analysis as Fluent was used to compute steady-flow equations. MpCCI was used to couple Fluent and Abaqus software in a loosely coupling way. CATIA is used to create wing model and HyperMesh is used to generate mesh. CATIA and HyperMesh were not implemented to the optimization process since any shape parameter such as taper ratio and swept angle of the wing was not defined as optimization variable.

6.2 Recommendations and Future Work

All of the experiences gained during this effort could be asset for further studies. The identification and validation of ARW-2 wing model has been performed successfully and the reliability of the computational model was satisfied. Thus, this identified ARW-2 computational wing model could be used for any computational purposes with confidence. In addition, other computational models of ARW-2 wing could be created, if the identification method described in this study is applied. For example, a computational ARW- wing model with composite skin will be soon identified with the same methodolgy.

The future work for this study is to extend the study to dynamic aeroelasticity with applications related to aeroelastic optimization either to suppress dynamic instability of flutter or to predict flutter speed. Next, more complicated optimization problems involving more advanced geometries and materials with composite modeling will be studied to furthermore enhance virtual prototyping using MDO techniques. The resulting large size optimization problems will be solved by parallel computing in the MDO framework. Optimization problem employed in this study can be improved by adding more constraints or changing the objectives of the problem. The methodolgy developed in this study can be used for any other optimization problem in which commercial or academic codes are used.

REFERENCES

[1] **Hodges H. D., and Pierce G. A.,** 2002: Introduction to Structural Dynamics and Aeroelasticity, Cambridge Aerospace Series.

[2] **Sobieszczanski-Sobieski and J., Haftka, R. T.,** 1996, Multidisciplinary aerospace design optimization: Survey of recent developments, *34th Proceedings of the AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, January.

[3] **Farhangnia M., Guruswamy G., and Biringen S.**, 1996, Transonic-buffet associated aeroelasticity of a supercritical wing. In 34th Aerospace Science Meeting and Exhibit, January 15-18 1996, Reno, NV. AIAA.

[4] **Bhardwaj M. K.,** 1997, A CFD/CSD Interaction Methodology for Aircraft Wings. PhD thesis, Virginia Polytechnic Institute and State University, USA.

[5] **Karpel M., Yaniv S., and Livshits D. S.,** 1996, Integrated solution for computational static aeroelastic problems. In 6th AIAA, NASA and ISSMO Symposium on Multidisiplinary Analysis and Optimization, 4-6 September 1996, Bellevue, WA. AIAA.

[6] **Newman III J. C., Newman P. A., Taylor III A. C., and Hou G.J.-W.,** 1999, Efficient nonlinear static wing analysis. Computers and Fluids, 28:615-628.

[7] **Garcia J.A. and Guruswamy G.P.,** 1999, Static aeroelastic characteristics of an advanced wing with a control surface using navier-stokes equations. In 37th Aerospace Science Meeting and Exhibit, 11-14 January,1999, Reno, NV. AIAA.

[8] **Liu F., Cai J., Zhu Y., Wong A.S.F., and Tsai H.M.**, 2000, Calculation of wing utter by a coupled cfd and csd method. In 38th AIAA Aerospace Sciences Meeting and Exhibit, 10-13 January 2000, Reno, NV. AIAA.

[9] **Cai J., Liu F., and Tsai H.M.,** 2001, Static aero-elastic computation with a coupled cfd and csd method. In 39th AIAA Aerospace Sciences Meeting and Exhibit, 8-11 January 2001, Reno, NV. AIAA.

[10] **Kamakoti R., Yongsheng L., and Regisford S.,** 2002, Computational aeroelasticity using a pressure-based solver. In 40th Aerospace Sciences Meeting and Exhibit, 14-17 January, 2002, Reno, NV. AIAA.

[11] **Farhat C. and Lesoinne M.,** 2000, Two effcient staggered procedures for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic problems. Comp. Meth. Appl.Mech.Eng., 182:499 - 515.

[12] **Gordnier R. and Fithen R.,** 2003, Coupling of a nonlinear finite element structural method with a navier-stokes solver. Computers and Structures, 81:75-89.

[13] **Lim Feng Z. and Soulaimani A.,** 2006, Nonlinear aeroelasticity computations in transonic flows using tightly coupling algoritms. In 2006 ASME Pressure Vessels and Piping Division Conference, 2327 July 2006, Vancouver, BC, Canada. ASME.

[14] **Farhat C., Lesoinne M., and Maman N.,** 1995, Mixed explicit/implicit time integration of coupled aeroelastic problems: Three-field formulation, geometric conservation and distributed solution. Int J. Num. Meth.Fluids, 21:807-835.

[15] **Dowell E.H., Thomas J.P., and Hall K.C.,** 2004, Transonic limit cycle oscillation analysis using reduced order aerodynamic models. Journal of Fluids and Structures, 19:17-17.

[16] **Lieu T., Farhat C., and Lesoinne M.,** 2006, Reduced-order fluid-structure modeling of a complete aircraft configuration. Computer Methods in Applied Mechanics and Engineering, 195:5730-5742.

[17] **Haddadpour H., Behbahani-Nejad M., and Firouz-Abadi R.D.,** 2007, Reduced-order aerodynamic model for aeroelastic analysis of complex configurations in incompressible flow. Journal of Aircraft, 44:1015-1019.

[18] **Relvas A. and Suleman A.,** 2006, Fluid-structure interaction modelling of nonlinear aeroelastic structures using the finite element corotational theory. Journal of Fluids and Structures, 22:59-75.

[19] **Love M., Garza T.D.L., Charlton E., and Egle D.,** 2000, Computational aeroelasticity in high performance aircraft flight loads. In International Council of the Aeronautical Sciences (ICAS) 2000 Congress.

[20] **Kuntz M. and Menter F.R.,** 2004, Simulation of uid-structure interactions in aeronautical applications. In European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS 2004, 24-28 July 2004, Jyvaskyla, number 133-144.

[21] **Heinrich, R., Ahrem, R., Guenther, G., Kersken, H. P., Krueger, W. And Neumann, J.**, 2001, Aeroelastic computation using the AMANDA simulation environment, *Proceedings of the CEAS Conference on Multidisciplinary Design and Optimization* (DGLR-Bericht 2001–2005), June 25–26, pp. 19–30.

[22] **Cavagna, L., Quaranta, G., Mantegazza, P., Merlo, E., Marchetti, D. And Martegani, M.**, 2005, Preliminary assessment of the complete aeroelastic simulation of the M-346 in transonic condition with a CFD Navier-Stokes solver, *XVIII Congresso Nazionale AIDAA*, Volterra.

[23] **Thirifay, F. and Geuzaine, P.**, 2004, Numerical Simulations of Fluid-Structure Interaction Problems using MpCCI, *5th MpCCI User Forum*, March, Schloss Birlinghoven, Snakt Augustin, Germany.

[24] **Yosibash, Z., Kirby, R. M., Myers, M., Szabo, B. and Karniadakis, G.**, 2003, High-order finite elements for fluid-stucture interaction

problems, *44th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference*, 7-10 April, Norfolk, Virginia.

[25] **Svácek, P.**, 2005, Application of finite element method in aeroelasticity, Journal of Computational and Applied Mathematics, 215, pp. 586-594

[26] **Fazelzadeh S.A., Mazidi A., Kalantari H.**, 2009, Bending-torsional flutter of wings with an attached mass subjected to a follower force, Journal of Sound and Vibration 323 : 148-162

[27] **Stanford B., Ifju P., Albertani R., Shyy W.**, 2008, Fixed membrane wings for micro air vehicles: Experimental characterization, numerical modeling, and tailoring, Progress in Aerospace Sciences 44 : 258-294

[28] **Lim I., Lee I.**, 2009, Aeroelastic analysis of bearingless rotors with a composite flexbeam, Composite Structures, 88 : 570-578

[29] **Xie C. C., Leng, J. Z., Yang C.**, 2007, Geometrical nonlinear aeroelastic stability analysis of a composite high-aspect-ratio wing, Aeroelasticity Branch, Aircraft Design Institute, Beijing University of Aero. & Astro., Beijing 100083, China

[30] **Pahlavanloo P.**, 2007, Dynamic Aeroelastic Simulation of the AGARD 445.6 wing using Edge, Swedish Defence Research Agency Defence and Security, Systems and Technology Technical Report No:FOI-R-2259—SE

[31] **Edwards J. W., Wieseman C. D.**, 2008, Flutter and Divergence Analysis Using the Generalized Aeroelastic Analysis Method, JOURNAL OF AIRCRAFT, Vol. 45, No. 3, May–June 2008

[32] **Jian-min L., Chuan-jing L., Lei-ping X.**, 2008, INVESTIGATION OF AIRSHIP AEROELASTICITY USING FLUIDSTRUCTURE INTERACTION, Journal of Hydrodynamics, 20 (2) : 164-171

[33] **Sarkar S., Bijl H.**, 2008, Nonlinear aeroelastic behavior of an oscillating airfoil during stall-induced vibration, Journal of Fluids and Structures 24 (2008) 757–777

[34] **Shams S., Sadr Lahidjani M.H., Haddadpour H.**, 2008, Nonlinear aeroelastic response of slender wings based on Wagner function, Thin-Walled Structures 46 (2008) 1192– 1203

[35] **Silva, W. A.**, 2008, Simultaneous Excitation of Multiple-Input/Multiple-Output CFD-Based Unsteady Aerodynamic Systems, JOURNAL OF AIRCRAFT, Vol. 45, No. 4.

[36] **Abbas L. K., Chen Q., O'Donnell K., Valentine D. , Marzocca P.**, 2007, Numerical studies of a non-linear aeroelastic system with plunging and pitching freeplays in supersonic/hypersonic regimes, Aerospace Science and Technology 11 : 405–418

[37] **Chattot J.**, 2007, Helicoidal vortex model for wind turbine aeroelastic simulation, Computers and Structures 85 : 1072–1079

[38] **Baxevanou C.A., Chaviaropoulos P.K., Voutsinas S.G., Vlachos N.S.**, 2008, Evaluation study of a Navier– Stokes CFD aeroelastic model of wind

turbine airfoils in classical flutter, Journal of Wind Engineering and Industrial Aerodynamics 96 : 1425– 1443

[39] **Braun A. L., Awruch A. M.**, 2009, Aerodynamic and aeroelastic analyses on the CAARC standard tall building model using numerical simulation, Computers and Structures 87 : 564–581

[40] **Robinson T. D., Willcox K. E., Eldred M. S., Haimes R.**, 2001, Multifidelity Optimization for Variable-Complexity Design, American Institute of Aeronautics and Astronautics, Aerospace Computational Design Laboratory Massachusetts Institute of Technology, Cambridge, MA, 02139

[41] **Alonso J. J., LeGresley P., van der Weide E., Joaquim R. R. A. M., J. J. Reuther**, 2004, pyMDO: A Framework for High-Fidelity Multi-Disciplinary Optimization, 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference August 30 – September 1, 2004, Albany, New York

[42] **Kodiyalam S., Yang R.J., Gu L., Tho C.-H.**, 2004, Multidisciplinary design optimization of a vehicle system in a scalable, high performance computing environment

[43] **Korngold J. C., Gabriele G. A.**, 1997, Multidisciplinary Analysis and Optimization of Discrete Problems Using Response Surface Methods, Journal of Mechanical Design, Vol. 119:427-433

[44] **Venter G., Sobieszczanski-Sobieski J.**, 2004, Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization, Struct Multidisc Optim 26, 121–131

[45] **K. Gantois, A.J. Morris**, 2004, The multi-disciplinary design of a large-scale civil aircraft wing taking account of manufacturing costs, Struct Multidisc Optim 28, 31–46.

[46] **DeLaurentis D. A., Mavris D. N**., 2001, Uncertainty Modeling and Management in Multidisciplinary Analysis and Synthesis, Aerospace Systems Design Laboratory (ASDL) School of Aerospace Engineering, Georgia Institute of Technology Atlanta, GA 30332-0150

[47] **Barcelos, M. and Maute, K.**, 2008, Aeroelastic design optimization for laminar and turbulent flows, Computer Methods in Applied Mechanics and Engineering, 197 :1813-1832.

[48] **Librescu L., Maalawi K.Y.**, 2008, Aeroelastic design optimization of thin-walled subsonic wings against divergence, Thin-Walled Structures 47 : 89– 97

[49] **Guo S.**, 2007, Aeroelastic optimization of an aerobatic aircraft wing structure, Aerospace Science and Technology 11 : 396–404

[50] **Nikbay, M.**, 2002, Coupled sensitivity analysis by discrete-analytical direct and adjoint methods with applications to aeroelastic optimization and sonic boom minimization, PhD thesis, Department of Aerospace Engineering, University of Colorado.

[51] **Ghosh, A. and Dehuri, S.**, 2004, Evolutionary Algorithms for Multi-Criterion Optimization: A Survey, International Journal of computing & Information Sciences, 2 (1) : 38-57.

[52] **Sasaki, D., Obayashi, S. and Nakahashi, K.**, 2002, Navier-Stokes optimization of supersonic wings with four objectives using evolutionary algorithm, Journal of Aircraft, 39 (4):621-629.

[53] **Sandford M.C., Seidel D.A., Escktrom C.V., and Spain C.V.,** 1989, Geometrical and Structural properties of an Aeroelastic Research Wing (ARW-2). NASA Technical Memorandum 4110.

[54] **Sandford M.C., Seidel D.A., and Escktrom C.V.,** 1994, Steady Pressure Measurements on an aeroelastic Research Wing (ARW-2). NASA Technical Memorandum 109046.

[55] **Sandford, Maynard Seidel C., David A. and Eckstrom, Clinton V.,** 1987, Measured Unsteady Transonic Aerodynamic Characteristics of an Elastic Supercritical Wing. J. Aircraft, vol. 24, no. 4, pp. 225-230.

[56] **Seidel, David A. Eckstrom, Clinton V. and Sandford, Maynard C.,** 1989, Transonic Region of High Dynamic Response Encountered on an Elastic Supercritical Wing. J. Aircraft, vol. 26, no. 9, pp. 870-875

[57] **Eckstrom, Clinton V. Seidel, David A. and Sandford, Maynard C.,** 1990, Unsteady Pressure and Structural Response Measurements on an Elastic Supercritical Wing. J. Aircraft, vol. 27, no. 1, pp. 75-80.

[58] **Seidel, David A. Sandford, Maynard C. and Eckstrom, Clinton V.,** 1991, Unsteady-Pressure and Dynamic-Deflection Measurements on an Aeroelastic Supercritical Wing. NASA TM-4278.

[59] **Eckstrom, Clinton V.; Seidel, David A.; and Sandford, Maynard C.,** 1994, Measurement of Unsteady Pressure and Structural Response for an Elastic Supercritical Wing. NASA TP 3443

[60] **Seidel, David A. Adams, William M., Jr. Eckstrom, Clinton V. and Sandford, Maynard C.,** 1989, Investigation and Suppression of High Dynamic Response Encountered on an Elastic Supercritical Wing. "Transonic Unsteady Aerodynamics and Aeroelasticity 1987". NASA CP-3022, Part 2, pp. 427-448

[61] **Byrdsong, Thomas A. Adams, Richard R. and Sandford, Maynard C.,** 1990, Close-Range Photogrammetry Measurement of Static Deflections for an Aeroelastic Supercritical Wing. NASA TM-4194.

[62] **Byrdsong, Thomas A. Adams, Richard R. and Sandford, Maynard C.,** 1990, Close-Range Photogrammetry Measurement of Static Deflections for an Aeroelastic Supercritical Wing. Supplement to NASA TM-4194.

[63] **Eckstrom, Clinton V.,** 1983, Loads Calibrations of Strain Gage Bridges on the DAST Project Aeroelastic Research Wing (ARW-2). NASA TM-87677.

[64] **Murrow, H. N. and Eckstrom, C. V.,** 1979, Drones for Aerodynamic and Structural Testing (DAST)-A Status Report. J. Aircraft, vol. 16, no. 8, pp. 521-526.

[65] **Byrdsong, Thomas A. and Brooks, Cuyler W., Jr.,** 1980, Wind Tunnel Investigation of Longitudinal and Lateral-Directional Stability and Control Characteristics of a 0.237-Scale Model of a Remotely Piloted Research Vehicle With a Thick, High-Aspect-Ratio Supercritical Wing. NASA TM-81790.

[66] **Byrdsong, Thomas A. and Brooks, Cuyler W., Jr.,** 1983, Wind Tunnel Investigation of Aerodynamic Loading on a 0.237-Scale Model of a Remotely Piloted Research Vehicle With a Thick, High-Aspect-Ratio Supercritical Wing. NASA TM-84614.

[67] **Adams, William M., Jr. Tiffany, Sherwood H and Bardusch, Richard E.**, 1987, Active Suppression of an "Apparent Shock Induced Instability". AIAA Paper 87-0881.

[68] **Chapin, William G.,** 1983, Dynamic-Pressure Measurements Using an Electronically Scanned Pressure Module. NASA TM-84650.

[69] **Roos, Frederick W.,** 1978, Shock Oscillations and Pressure Fluctuation Measurements on Supercritical and Conventional Airfoils. "Advanced Technology Airfoil Research, Volume 2". NASA CP-2046, pp. 201-219

[70] **Cohen, David E.,** 1998, Trim Angle of Attack of Flexible Wings Using Non-Linear Aerodynamics. PhD Dissertation in Aerospace Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[71] **Whitlow, Woodrow, Jr. and Bennett, Robert M.,** 1982, Application of a Transonic Potential Flow Code to the Static Aeroelastic Analysis of Three-Dimensional Wings, AIAA Paper 82-0689.

[72] **Bennett, Robert M. Seidel, David A. and Sandford, Maynard C.,** 1985, Transonic Calculations for a Flexible Supercritical Wing and Comparison with Experiment. AIAA Paper 85-0665.

[73] **Bhardwaj, Manoj K. Kapania, Rakesh K. Reichenbach, Eric and Guruswamy, Guru P,** 1998, A Computational Fluid Dynamics/Computational Structural Dynamics Interaction Methodology for Aircraft Wings. AIAA Journal, vol. 36, no. 12, pp. 2179-2186

[74] **Nikbay M. and Aysan A.**, 2009, Identicifation of structural and aeroelastic properties of a Computational ARW-2 wing model by using multi-objective optimization techniques. In International Forum on Aeroelasticity and Structural Dynamics, Seattle, WA. IFASD.

[75] **Deb K., Pratap A., Agarwal S., and Meyarivan T.,** 2000, A Fast and Elitist Multi-Objective Genetic Algorithm NSGA-II. KanGAL Report 2000001.

[76] **Nikbay M., Oncu L., and Aysan A.,** 2008, A multi-disciplinary code coupling approach for analysis and optimization of aeroelastic systems. In 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 10 - 12 Sep 2008 , Victoria, British Columbia, Canada. AIAA.

[77] **Oncu L.,** 2008, Multidisiplinary Design Optimization of Aerospace Structures with Static Aeroelastic Criteria. MS Thesis, Istanbul Technical University, Turkey, 2008

[78] Abaqus 6.7 version documentation. Simulia

[79] Fluent 6.3.26 Version Documentation. ANSYS Inc

[80] Modefrontier V4 Version Documentation. Esteco

[81] MpCCI 3.0.6 Version Documentation. Fraunhofer-Institute SCAI

APPENDICES

APPENDIX A.1 : Abaqus Python Script

APPENDIX A.2 : Fluent Journal File

APPENDIX A.3 : Properties of ARW-2 Computational Model

APPENDIX A.1

```
# -*- coding: mbcs -*-
# Abaqus/CAE Version 6.7-1 replay file
# Internal Version: 2007_05_01-12.35.33 79448
# from driverUtils import executeOnCaeGraphicsStartup
# executeOnCaeGraphicsStartup()
#: Executing "onCaeGraphicsStartup()" in the site directory ...
from abaqus import *
from abaqusConstants import *
session.Viewport(name='Viewport: 1', origin=(0.0, 0.0), width=195.41015625,
    height=197.16796875)
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
from caeModules import *
from driverUtils import executeOnCaeStartup
executeOnCaeStartup()
Mdb()
#: A new model database has been created.
#: The model "Model-1" has been created.
session.viewports['Viewport: 1'].setValues(displayedObject=None)
mdb.ModelFromInputFile(name='deviation',
    inputFileName='deviation.inp')
#: The model "deviation" has been created.
#: The part "PART-1" has been imported from the input file.
#: The model "deviation" has been imported from an input file.
#: Please scroll up to check for error and warning messages.
a = mdb.models['deviation'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
del mdb.models['Model-1']
a = mdb.models['deviation'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
a = mdb.models['deviation'].rootAssembly
del a.features['PART-1-1']
a = mdb.models['deviation'].rootAssembly
del a.features['Datum csys-1']
del mdb.models['deviation'].rootAssembly.sets['LVL10000']
p = mdb.models['deviation'].parts['PART-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5352.97,
    farPlane=9703.74, width=3084.17, height=2143.61, cameraPosition=(7519.99,
    -3466.37, -532.259), cameraUpVector=(-0.0861421, 0.35512, 0.930843))
session.viewports['Viewport: 1'].view.setValues(nearPlane=5384.53,
    farPlane=9672.19, width=3102.36, height=2156.25, viewOffsetX=40.8618,
    viewOffsetY=59.7212)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5590.68,
    farPlane=9401.03, width=3221.13, height=2238.8, cameraPosition=(8438.48,
    9.77291, 2493.57), cameraUpVector=(-0.629375, 0.134674, 0.765343),
    viewOffsetX=42.4262, viewOffsetY=62.0076)
mdb.saveAs(pathName='arw')
#: The model database has been saved to "arw.cae".
session.viewports['Viewport: 1'].view.setValues(nearPlane=5718.77,
    farPlane=9272.94, width=2008.49, height=1395.97, viewOffsetX=-23.9007,
    viewOffsetY=109.813)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5734.08,
    farPlane=9257.63, width=2013.87, height=1399.71, viewOffsetX=206.599,
    viewOffsetY=-69.4471)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5734.11,
    farPlane=9257.6, width=2013.88, height=1399.72, viewOffsetX=122.944,
    viewOffsetY=-110.256)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5792.58,
    farPlane=9162.09, width=2034.42, height=1413.99, cameraPosition=(8174.23,
    1539.49, 3172.96), cameraUpVector=(-0.720976, 0.0832939, 0.687936),
    viewOffsetX=124.198, viewOffsetY=-111.38)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5790,
    farPlane=9164.67, width=2033.51, height=1413.36, viewOffsetX=128.263,
    viewOffsetY=32.8905)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5789.31,
    farPlane=9165.36, width=2163.05, height=1503.4, viewOffsetX=144.087,
    viewOffsetY=39.232)
```

```
session.viewports['Viewport: 1'].view.setValues(nearPlane=5773.73,
    farPlane=9180.94, width=2157.23, height=1499.35, viewOffsetX=655.14,
    viewOffsetY=-476.686)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6030.7,
    farPlane=8923.97, width=194.339, height=135.073, viewOffsetX=525.113,
    viewOffsetY=-434.388)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6032.32,
    farPlane=8922.35, width=194.392, height=135.109, viewOffsetX=403.341,
    viewOffsetY=-343.513)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=366.314,
    viewOffsetY=-296.244)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=287.533,
    viewOffsetY=-227.902)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6042.3,
    farPlane=8912.38, width=111.57, height=77.545, viewOffsetX=296.363,
    viewOffsetY=-238.324)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6043.23,
    farPlane=8911.44, width=111.587, height=77.557, viewOffsetX=262.152,
    viewOffsetY=-207.157)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=220.886,
    viewOffsetY=-170.413)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=190.361,
    viewOffsetY=-138.192)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=142.312,
    viewOffsetY=-106.762)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6039.88,
    farPlane=8914.79, width=128.82, height=89.5344, viewOffsetX=14.1378,
    viewOffsetY=-4.47018)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6040.96,
    farPlane=8913.72, width=128.843, height=89.5504, viewOffsetX=-37.423,
    viewOffsetY=49.0504)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6044.67,
    farPlane=8910, width=94.6165, height=65.7618, viewOffsetX=-46.2223,
    viewOffsetY=50.5403)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6045.46,
    farPlane=8909.21, width=94.6289, height=65.7704, viewOffsetX=-72.3064,
    viewOffsetY=71.9272)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6042.93,
    farPlane=8911.74, width=107.05, height=74.4035, viewOffsetX=-135.369,
    viewOffsetY=108.718)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6043.82,
    farPlane=8910.85, width=107.066, height=74.4145, viewOffsetX=-162.942,
    viewOffsetY=138.999)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-203.946,
    viewOffsetY=175.447)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-231.499,
    viewOffsetY=197.034)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-277.168,
    viewOffsetY=229.468)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-309.602,
    viewOffsetY=255.394)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-339.433,
    viewOffsetY=276.655)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5854.3,
    farPlane=9100.37, width=1637.37, height=1138.03, viewOffsetX=-187.223,
    viewOffsetY=379.293)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5842.64,
    farPlane=9112.04, width=1634.11, height=1135.76, viewOffsetX=131.032,
    viewOffsetY=30.8542)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#ffffffff:340 #7fffffff ]', ), )
p.Set(elements=elements, name='Upper')
#: The set 'Upper' has been created (10911 elements).
set1 = mdb.models['deviation'].parts['PART-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5842.62,
    farPlane=9112.05, width=1634.1, height=1135.76, viewOffsetX=129.376,
    viewOffsetY=-181.066)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5675.56,
    farPlane=9397.21, width=1587.38, height=1103.29, cameraPosition=(8534.81,
    -1128.06, -1528.92), cameraUpVector=(-0.116427, 0.122274, 0.985644),
    viewOffsetX=125.677, viewOffsetY=-175.889)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5681.49,
    farPlane=9391.28, width=1589.05, height=1104.44, viewOffsetX=-64.1688,
    viewOffsetY=-209.882)
```

```
session.viewports['Viewport: 1'].view.setValues(nearPlane=5745.71,
    farPlane=9331.13, width=1607.01, height=1116.93, cameraPosition=(8468.29,
    -667.63, -2251.86), cameraUpVector=(-0.0230808, 0.109438, 0.993726),
    viewOffsetX=-64.8942, viewOffsetY=-212.255)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5743.43,
    farPlane=9333.41, width=1606.37, height=1116.49, viewOffsetX=94.6297,
    viewOffsetY=-111.264)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5716.11,
    farPlane=9360.73, width=1924.83, height=1337.83, viewOffsetX=65.2702,
    viewOffsetY=-83.6425)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5702.12,
    farPlane=9374.72, width=1920.12, height=1334.55, viewOffsetX=63.165,
    viewOffsetY=-155.418)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5767.01,
    farPlane=9290.02, width=1941.97, height=1349.74, cameraPosition=(8878.49,
    410.116, -1114.21), cameraUpVector=(-0.194233, 0.120276, 0.973554),
    viewOffsetX=63.8838, viewOffsetY=-157.187)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5764.26,
    farPlane=9292.78, width=1941.05, height=1349.1, viewOffsetX=42.2206,
    viewOffsetY=-180.711)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5630,
    farPlane=9390.75, width=1895.84, height=1317.68, cameraPosition=(8762.03,
    2535.28, -675.796), cameraUpVector=(-0.276317, 0.0458166, 0.959974),
    viewOffsetX=41.2372, viewOffsetY=-176.502)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:340 #80000000 #ffffffff:371 #7fffffff ]', ), )
p.Set(elements=elements, name='Lower')
#: The set 'Lower' has been created (11904 elements).
session.viewports['Viewport: 1'].view.setValues(nearPlane=5635.69,
    farPlane=9385.06, width=1897.76, height=1319.01, viewOffsetX=173.949,
    viewOffsetY=-178.603)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5613.27,
    farPlane=9347.33, width=1890.21, height=1313.76, cameraPosition=(8351.98,
    2924.59, 2059.48), cameraUpVector=(-0.650446, 0.177618, 0.738493),
    viewOffsetX=173.257, viewOffsetY=-177.892)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5614.23,
    farPlane=9346.39, width=1890.53, height=1313.99, viewOffsetX=295.874,
    viewOffsetY=-30.434)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5717.8,
    farPlane=9225.42, width=1925.41, height=1338.23, cameraPosition=(8080.4,
    2350.94, 3104.16), cameraUpVector=(-0.739782, 0.31419, 0.594985),
    viewOffsetX=301.332, viewOffsetY=-30.9955)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5713.41,
    farPlane=9229.81, width=1923.93, height=1337.2, viewOffsetX=316.695,
    viewOffsetY=17.7602)
leaf = dgm.Leaf(leafType=DEFAULT_MODEL)
session.viewports['Viewport: 1'].partDisplay.displayGroup.replace(leaf=leaf)
set1 = mdb.models['deviation'].parts['PART-1'].sets['Lower']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:721 #ff800000 #ffffffff #3fffff ]',
    ), )
p.Set(elements=elements, name='Tip')
#: The set 'Tip' has been created (63 elements).
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:712 #80000000 #ffffffff:8 #7fffff ]', ), )
p.Set(elements=elements, name='Root')
#: The set 'Root' has been created (280 elements).
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:723 #ffc00000 #ffffffff:9 #1f ]',
    ), )
p.Set(elements=elements, name='Tra')
#: The set 'Tra' has been created (303 elements).
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:733 #ffffffe0 #ffffffff:10 #fffff ]', ), )
p.Set(elements=elements, name='Spar1')
#: The set 'Spar1' has been created (367 elements).
```

```
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:744 #fff00000 #ffffffff:8 #1fff ]',
    ), )
p.Set(elements=elements, name='Spar2')
#: The set 'Spar2' has been created (281 elements).
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:753 #ffffe000 #ffffffff:14 #7f ]',
    ), )
p.Set(elements=elements, name='AuxSpar')
#: The set 'AuxSpar' has been created (474 elements).
set1 = mdb.models['deviation'].parts['PART-1'].sets['AuxSpar']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Root']
set3 = mdb.models['deviation'].parts['PART-1'].sets['Spar1']
set4 = mdb.models['deviation'].parts['PART-1'].sets['Spar2']
set5 = mdb.models['deviation'].parts['PART-1'].sets['Tip']
set6 = mdb.models['deviation'].parts['PART-1'].sets['Tra']
leaf = dgm.LeafFromSets(sets=(set1, set2, set3, set4, set5, set6, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5830.87,
    farPlane=8925.41, width=1963.48, height=1364.69, viewOffsetX=249.6,
    viewOffsetY=-29.619)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5748.32,
    farPlane=8949.46, width=1935.68, height=1345.37, cameraPosition=(7534.62,
    3658.96, 3308.21), cameraUpVector=(-0.806623, 0.146993, 0.572497),
    viewOffsetX=246.066, viewOffsetY=-29.1997)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:768 #ffffff80 #ffffffff:58 #7ffff ]', ), )
p.Set(elements=elements, name='Ribs')
#: The set 'Ribs' has been created (1900 elements).
set1 = mdb.models['deviation'].parts['PART-1'].sets['Ribs']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5634.85,
    farPlane=9268.28, width=1897.47, height=1318.81, viewOffsetX=85.4888,
    viewOffsetY=-40.1581)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5825.44,
    farPlane=9154.96, width=1961.65, height=1363.42, cameraPosition=(8491.08,
    707.6, 2501.76), cameraUpVector=(-0.628073, 0.229744, 0.743466),
    viewOffsetX=88.3804, viewOffsetY=-41.5164)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5817.35,
    farPlane=9163.05, width=1958.93, height=1361.53, viewOffsetX=88.2576,
    viewOffsetY=-41.4587)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5676.92,
    farPlane=9261.35, width=1911.65, height=1328.66, cameraPosition=(8095.58,
    2591.4, 2952.43), cameraUpVector=(-0.717606, 0.0645619, 0.693451),
    viewOffsetX=86.1271, viewOffsetY=-40.4579)
p = mdb.models['deviation'].parts['PART-1']
s = p.elements
side2Elements = s.getSequenceFromMask(mask=(
    '[#ffffffff:39 #7f #0:23 #fffff800 #ffffffff:2 #fc01ffff #ffffffff:3',
    ' #fffff #0:24 #fffff800 #ffffffff:33 #1fffffff #0 #fffffe00',
    ' #ffffffff:3 #ff0fffff #ffffffff:12 #7fffff #fffffffc #ffffffff:12 #3ffff',
    ' #0 #fffffffe #ffffffff:20 #7ffff #0:14 #ffff0000 #ffffffff:139',
    ' #7fffffff ]', ), )
p.Surface(side2Elements=side2Elements, name=',')
#: The surface ',' has been created (8756 mesh faces).
p = mdb.models['deviation'].parts['PART-1']
s = p.elements
side1Elements = s.getSequenceFromMask(mask=(
    '[#0:39 #ffffff80 #ffffffff:23 #7ff #0:2 #3fe0000 #0:3',
    ' #fff00000 #ffffffff:24 #7ff #0:33 #e0000000 #ffffffff #1ff',
    ' #0:3 #f00000 #0:12 #ff800000 #3 #0:12 #fffc0000',
    ' #ffffffff #1 #0:20 #fff80000 #ffffffff:14 #ffff ]', ), )
p.Surface(side1Elements=side1Elements, name='Surf-2')
#: The surface 'Surf-2' has been created (2155 mesh faces).
a=mdb.models['deviation'].parts['PART-1']
a.SurfaceByMerge(name='Upper', surfaces=(a.surfaces[','], a.surfaces['Surf-2'],
    ))
mdb.models['deviation'].parts['PART-1'].deleteSurfaces(surfaceNames=(',',
    'Surf-2', ))
session.viewports['Viewport: 1'].view.setValues(nearPlane=5694.98,
    farPlane=9243.3, width=1933.46, height=1343.82, viewOffsetX=77.6875,
```

```
    viewOffsetY=37.0261)
p = mdb.models['deviation'].parts['PART-1']
s = p.elements
side1Elements = s.getSequenceFromMask(mask=(
    '[#0:39 #ffffff80 #ffffffff:23 #7ff #0:2 #3fe0000 #0:3',
    ' #fff00000 #ffffffff:24 #7ff #0:33 #e0000000 #ffffffff #1ff',
    ' #0:3 #f00000 #0:12 #ff800000 #3 #0:12 #fffc0000',
    ' #ffffffff #1 #0:20 #fff80000 #ffffffff:14 #ffff ]', ), )
side2Elements = s.getSequenceFromMask(mask=(
    '[#ffffffff:39 #7f #0:23 #fffff800 #ffffffff:2 #fc01ffff #ffffffff:3',
    ' #fffff #0:24 #fffff800 #ffffffff:33 #1fffffff #0 #fffffe00',
    ' #ffffffff:3 #ff0fffff #ffffffff:12 #7fffff #fffffffc #ffffffff:12 #3ffff',
    ' #0 #fffffffe #ffffffff:20 #7ffff #0:14 #ffff0000 #ffffffff:139',
    ' #7fffffff ]', ), )
p.Surface(side1Elements=side1Elements, side2Elements=side2Elements,
    name='Upper')
#: The surface 'Upper' has been edited (10911 mesh faces).
set1 = mdb.models['deviation'].parts['PART-1'].sets['Lower']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5707.59,
    farPlane=9263.45, width=1937.73, height=1346.79, viewOffsetX=77.8594,
    viewOffsetY=37.108)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6025.17,
    farPlane=8716.22, width=2045.55, height=1421.73, cameraPosition=(-1284.77,
    4653.61, -5836.01), cameraUpVector=(0.70178, 0.55534, 0.446209),
    viewOffsetX=82.1916, viewOffsetY=39.1728)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6011.61,
    farPlane=8729.79, width=2040.95, height=1418.53, viewOffsetX=79.9387,
    viewOffsetY=-47.7641)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5756.55,
    farPlane=9045.93, width=1954.36, height=1358.35, cameraPosition=(-2353.82,
    -3522.53, -4578.83), cameraUpVector=(0.170674, 0.871574, -0.459596),
    viewOffsetX=76.5471, viewOffsetY=-45.7376)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5767.45,
    farPlane=9035.04, width=1958.06, height=1360.92, viewOffsetX=425.85,
    viewOffsetY=-49.7919)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5767.47,
    farPlane=9035.01, width=1958.07, height=1360.93, viewOffsetX=340.546,
    viewOffsetY=-101.372)
p = mdb.models['deviation'].parts['PART-1']
s = p.elements
side1Elements = s.getSequenceFromMask(mask=(
    '[#0:362 #ffffffc0 #ffffffff:3 #7ffff #0:21 #ffffff00 #ffffffff:2',
    ' #7fffffff #0:7 #ffffffff:3 #1fffff #0:24 #ff800000 #ffffffff:3',
    ' #7f #0:6 #ff000000 #ffffffff:3 #1 #0:25 #fffffc00',
    ' #ffffffff:2 #3ffff #0:6 #ff800000 #ffffffff:3 #7 #0:12',
    ' #fffe0000 #ffffffff:16 #7fffff #0:7 #fffffffe #ffffffff:2 #fff',
    ' #0:29 #ffffffe0 #3fff #0:14 #3fffff00 #0:3 #ffe00000',
    ' #ffffffff #ffff #fc000000 #ffffffff #3f #0:16 #ffffff80',
    ' #ffffffff #1 #0:4 #80000000 #ffffffff:29 #ffff #0:2',
    ' #ffffff00 #ffffffff:7 #fff #0:10 #ffffff80 #ffffffff:25 #3ffffff ]', ), )
p.Surface(side1Elements=side1Elements, name='Surf-2')
#: The surface 'Surf-2' has been created (3743 mesh faces).
p = mdb.models['deviation'].parts['PART-1']
s = p.elements
side2Elements = s.getSequenceFromMask(mask=(
    '[#0:340 #80000000 #ffffffff:21 #3f #0:3 #fff80000 #ffffffff:21',
    ' #ff #0:2 #80000000 #ffffffff:7 #0:3 #ffe00000 #ffffffff:24',
    ' #7fffff #0:3 #ffffff80 #ffffffff:6 #ffffff #0:3 #fffffffe',
    ' #ffffffff:25 #3ff #0:2 #fffc0000 #ffffffff:6 #7fffff #0:3',
    ' #fffffff8 #ffffffff:12 #1ffff #0:16 #ff800000 #ffffffff:7 #1',
    ' #0:2 #fffff000 #ffffffff:29 #1f #ffffc000 #ffffffff:14 #c00000ff',
    ' #ffffffff:3 #1fffff #0 #ffff0000 #3ffffff #0 #ffffffc0',
    ' #ffffffff:16 #7f #0 #fffffffe #ffffffff:4 #7fffffff #0:29',
    ' #ffff0000 #ffffffff:2 #ff #0:7 #fffff000 #ffffffff:10 #7f',
    ' #0:25 #fc000000 #ffffffff:31 #7fffffff ]', ), )
p.Surface(side2Elements=side2Elements, name='Surf-3')
#: The surface 'Surf-3' has been created (8161 mesh faces).
a=mdb.models['deviation'].parts['PART-1']
a.SurfaceByMerge(name='Lower', surfaces=(a.surfaces['Surf-2'],
    a.surfaces['Surf-3'], ))
mdb.models['deviation'].parts['PART-1'].deleteSurfaces(surfaceNames=('Surf-2',
    'Surf-3', ))
p = mdb.models['deviation'].parts['PART-1']
s = p.elements
```

```
side1Elements = s.getSequenceFromMask(mask=(
    '[#0:362 #fffffc0 #ffffffff:3 #7ffff #0:21 #ffffff00 #ffffffff:2',
    ' #7fffffff #0:7 #ffffffff:3 #1fffff #0:24 #ff800000 #ffffffff:3',
    ' #7f #0:6 #ff000000 #ffffffff:3 #1 #0:25 #fffffc00',
    ' #ffffffff:2 #3ffff #0:6 #ff800000 #ffffffff:3 #7 #0:12',
    ' #fffe0000 #ffffffff:16 #7fffff #0:7 #fffffffe #ffffffff:2 #fff',
    ' #0:29 #ffffffe0 #3fff #0:14 #3fffff00 #0:3 #ffe00000',
    ' #ffffffff #ffff #fc000000 #ffffffff #3f #0:16 #ffffff80',
    ' #ffffffff #1 #0:4 #80000000 #ffffffff:29 #ffff #0:2',
    ' #ffffff00 #ffffffff:7 #fff #0:10 #ffffff80 #ffffffff:25 #3fffff ]', ), )
side2Elements = s.getSequenceFromMask(mask=(
    '[#0:340 #80000000 #ffffffff:21 #3f #0:3 #fff80000 #ffffffff:21',
    ' #ff #0:2 #80000000 #ffffffff:7 #0:3 #ffe00000 #ffffffff:24',
    ' #7fffff #0:3 #ffffff80 #ffffffff:6 #ffffff #0:3 #fffffffe',
    ' #ffffffff:25 #3ff #0:2 #fffc0000 #ffffffff:6 #7fffff #0:3',
    ' #ffffffff8 #ffffffff:12 #1ffff #0:16 #ff800000 #ffffffff:7 #1',
    ' #0:2 #fffff000 #ffffffff:29 #1f #ffffc000 #ffffffff:14 #c00000ff',
    ' #ffffffff:3 #1fffff #0 #ffff0000 #3ffffff #0 #ffffffc0',
    ' #ffffffff:16 #7f #0 #fffffffe #ffffffff:4 #7fffffff #0:29',
    ' #ffff0000 #ffffffff:2 #ff #0:7 #fffff000 #ffffffff:10 #7f',
    ' #0:25 #fc000000 #ffffffff:31 #7fffffff ]', ), )
p.Surface(side1Elements=side1Elements, side2Elements=side2Elements,
    name='Lower')
#: The surface 'Lower' has been edited (11904 mesh faces).
session.viewports['Viewport: 1'].view.setValues(nearPlane=5767.47,
    width=1958.07, height=1360.93, viewOffsetX=356.417, viewOffsetY=-204.533)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5671.54,
    farPlane=9233.68, width=1925.5, height=1338.29, cameraPosition=(5767.72,
    6109.89, -3285.66), cameraUpVector=(-0.478641, 0.393739, 0.784775),
    viewOffsetX=350.488, viewOffsetY=-201.131)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5675.64,
    farPlane=9229.58, width=1926.89, height=1339.26, viewOffsetX=321.457,
    viewOffsetY=-162.231)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5631.56,
    farPlane=9321.03, width=1911.93, height=1328.86, cameraPosition=(7825.97,
    3536.76, 2849.74), cameraUpVector=(-0.711028, -0.0310708, 0.702477),
    viewOffsetX=318.96, viewOffsetY=-160.971)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5633.44,
    farPlane=9319.14, width=1912.57, height=1329.3, viewOffsetX=328.755,
    viewOffsetY=-172.651)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5535.67,
    farPlane=9426.89, width=1879.38, height=1306.23, cameraPosition=(8073.3,
    4059.8, 1485.32), cameraUpVector=(-0.513995, -0.145636, 0.84534),
    viewOffsetX=323.049, viewOffsetY=-169.654)
set1 = mdb.models['deviation'].parts['PART-1'].sets['Tip']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Tra']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5539.84,
    farPlane=9422.71, width=1880.8, height=1307.22, viewOffsetX=346.159,
    viewOffsetY=-183.121)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5491.09,
    farPlane=9456.25, width=1864.25, height=1295.72, cameraPosition=(7687.25,
    5046.25, 359.806), cameraUpVector=(-0.303354, -0.249571, 0.919614),
    viewOffsetX=343.113, viewOffsetY=-181.51)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5699.08,
    farPlane=9248.25, width=284.194, height=197.525, viewOffsetX=86.8153,
    viewOffsetY=-122.523)
p = mdb.models['deviation'].parts['PART-1']
s = p.elements
side2Elements = s.getSequenceFromMask(mask=(
    '[#0:721 #ff800000 #ffffffff #3fffff ]', ), )
p.Surface(side2Elements=side2Elements, name='Tip')
#: The surface 'Tip' has been created (63 mesh faces).
session.viewports['Viewport: 1'].view.setValues(nearPlane=5717.38,
    farPlane=9229.95, width=153.562, height=106.731, viewOffsetX=88.1244,
    viewOffsetY=-135.643)
p = mdb.models['deviation'].parts['PART-1']
s = p.elements
side2Elements = s.getSequenceFromMask(mask=(
    '[#0:723 #ffc00000 #ffffffff:9 #1f ]', ), )
p.Surface(side2Elements=side2Elements, name='Tra')
#: The surface 'Tra' has been created (303 mesh faces).
session.viewports['Viewport: 1'].view.setValues(nearPlane=5676.01,
    farPlane=9271.33, width=507.457, height=352.701, viewOffsetX=44.4887,
    viewOffsetY=-97.8188)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5671.86,
```

```
    farPlane=9274.03, width=507.087, height=352.443, cameraPosition=(7617.91,
    5168.88, 66.7488), cameraUpVector=(-0.247693, -0.263823, 0.932226),
    viewOffsetX=44.4562, viewOffsetY=-97.7473)
a=mdb.models['deviation'].parts['PART-1']
a.SurfaceByMerge(name='Couple', surfaces=(a.surfaces['Lower'],
    a.surfaces['Tip'], a.surfaces['Tra'], a.surfaces['Upper'], ))
session.viewports['Viewport: 1'].view.setValues(nearPlane=5700.22,
    farPlane=9245.68, width=274.491, height=190.781, viewOffsetX=7.14359,
    viewOffsetY=-85.4685)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5703.17,
    farPlane=9242.68, width=274.633, height=190.879, cameraPosition=(7564.61,
    5243.36, -276.982), cameraUpVector=(-0.190391, -0.267398, 0.94459),
    viewOffsetX=7.14728, viewOffsetY=-85.5127)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-3.14798,
    viewOffsetY=-10.6634)
p = mdb.models['deviation'].parts['PART-1']
s = p.elements
side1Elements = s.getSequenceFromMask(mask=(
    '[#0:39 #ffffff80 #ffffffff:23 #7ff #0:2 #3fe0000 #0:3',
    ' #fff00000 #ffffffff:24 #7ff #0:33 #e0000000 #ffffffff #1ff',
    ' #0:3 #f00000 #0:12 #ff800000 #3 #0:12 #fffc0000',
    ' #ffffffff #1 #0:20 #fff80000 #ffffffff:14 #ffff #0:161',
    ' #fffffc0 #ffffffff:3 #7ffff #0:21 #ffffff00 #ffffffff:2 #7fffffff',
    ' #0:7 #ffffffff:3 #1fffff #0:24 #ff800000 #ffffffff:3 #7f',
    ' #0:6 #ff000000 #ffffffff:3 #1 #0:25 #fffffc00 #ffffffff:2',
    ' #3ffff #0:6 #ff800000 #ffffffff:3 #7 #0:12 #fffe0000',
    ' #ffffffff:16 #7fffff #0:7 #fffffffe #ffffffff:2 #fff #0:29',
    ' #fffffe0 #3fff #0:14 #3fffff00 #0:3 #ffe00000 #ffffffff',
    ' #ffff #fc000000 #ffffffff #3f #0:16 #ffffff80 #ffffffff',
    ' #1 #0:4 #80000000 #ffffffff:29 #ffff #0:2 #ffffff00',
    ' #ffffffff:7 #fff #0:10 #ffffff80 #ffffffff:25 #3ffffff ]', ), )
side2Elements = s.getSequenceFromMask(mask=(
    '[#ffffffff:39 #7f #0:23 #fffff800 #ffffffff:2 #fc01ffff #ffffffff:3',
    ' #fffff #0:24 #fffff800 #ffffffff:33 #1fffffff #0 #fffffe00',
    ' #ffffffff:3 #ff0fffff #ffffffff:12 #7fffff #fffffffc #ffffffff:12 #3ffff',
    ' #0 #fffffffe #ffffffff:20 #7ffff #0:14 #ffff0000 #ffffffff:161',
    ' #3f #0:3 #fff80000 #ffffffff:21 #ff #0:2 #80000000',
    ' #ffffffff:7 #0:3 #ffe00000 #ffffffff:24 #7fffff #0:3 #ffffff80',
    ' #ffffffff:6 #ffffff #0:3 #fffffffe #ffffffff:25 #3ff #0:2',
    ' #fffc0000 #ffffffff:6 #7fffff #0:3 #fffffff8 #ffffffff:12 #1ffff',
    ' #0:16 #ff800000 #ffffffff:7 #1 #0:2 #fffff000 #ffffffff:29',
    ' #1f #ffffc000 #ffffffff:14 #c00000ff #ffffffff:3 #1fffff #0',
    ' #ffff0000 #3ffffff #0 #fffffffc0 #ffffffff:16 #7f #0',
    ' #ffffffffe #ffffffff:4 #7fffffff #0:29 #ffff0000 #ffffffff:2 #ff',
    ' #0:7 #fffff000 #ffffffff:10 #7f #0:25 #fc000000 #ffffffff:31',
    ' #7fffffff #0:8 #ff800000 #ffffffff:11 #1f ]', ), )
p.Surface(side1Elements=side1Elements, side2Elements=side2Elements,
    name='Couple')
#: The surface 'Couple' has been edited (23181 mesh faces).
session.viewports['Viewport: 1'].view.setValues(nearPlane=5642.51,
    farPlane=9303.34, width=781.103, height=542.894, viewOffsetX=16.57,
    viewOffsetY=48.0843)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5636.54,
    farPlane=9309.31, width=780.277, height=542.32, viewOffsetX=152.528,
    viewOffsetY=-12.8392)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5641.27,
    farPlane=9313.45, width=780.931, height=542.775, cameraPosition=(7853.65,
    4776.12, 450.949), cameraUpVector=(-0.302516, -0.275963, 0.91232),
    viewOffsetX=152.656, viewOffsetY=-12.85)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5641.19,
    farPlane=9313.54, width=780.92, height=542.767, viewOffsetX=236.521,
    viewOffsetY=-24.7179)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5614.33,
    farPlane=9340.4, width=980.074, height=681.186, viewOffsetX=235.395,
    viewOffsetY=-24.6002)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5614.93,
    farPlane=9339.79, width=980.179, height=681.259, viewOffsetX=235.42,
    viewOffsetY=-116.96)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5591.06,
    farPlane=9363.66, width=1156.38, height=803.726, viewOffsetX=234.419,
    viewOffsetY=-116.463)
mdb.save()
#: The model database has been saved to "arw.cae".
set1 = mdb.models['deviation'].parts['PART-1'].sets['Lower']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Spar1']
set3 = mdb.models['deviation'].parts['PART-1'].sets['Tip']
set4 = mdb.models['deviation'].parts['PART-1'].sets['Tra']
```

```
leaf = dgm.LeafFromSets(sets=(set1, set2, set3, set4, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=ON,
    engineeringFeatures=ON)
session.viewports['Viewport: 1'].view.setValues(cameraPosition=(1390.98,
    8603.95, -227.551), cameraUpVector=(0, 0, 1), cameraTarget=(1390.98,
    1190.06, -227.551), viewOffsetX=0, viewOffsetY=0)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6805.06,
    farPlane=8330.57, width=912.714, height=634.368, viewOffsetX=493.441,
    viewOffsetY=132.97)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6812.46,
    farPlane=8323.17, width=913.705, height=635.057, viewOffsetX=-132.748,
    viewOffsetY=141.446)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-705.781,
    viewOffsetY=139.595)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-1175.13,
    viewOffsetY=109.046)
p = mdb.models['deviation'].parts['PART-1']
e = p.elementEdges
elementEdges = e.getSequenceFromMask(mask=(
    '[#0:23467 #1 #0 #4:2 #0:10 #4:2 #0:2',
    ' #4:2 #0:3 #2 #0:2 #2:2 #0:8 #2:2', ' #0:2 #2:2 #0:9 #2 #0:2 #2:2 #0:8',
    ' #2:2 #0:2 #2:2 #0:11 #1 #0 #4:2', ' #0:7 #1:2 #0:6 #1:2 #8:3 #0:6 #8:2',
    ' #0:3 #8:2 #0:4 #8:2 #0:4 #8:2 #0:4', ' #8:3 #0:6 #1 #0:2 #4:2 #0:4 #1',
    ' #0:2 #4:2 #0:4 #1 #0:2 #4:2 #0:4', ' #4:2 #0:4 #4:2 #0:4 #1 #0:2 #4:2',
    ' #0:4 #4:2 #0:4 #4:2 #0:4 #1 #0:2', ' #4:2 #0:4 #4:2 #0:4 #4:2 #0:4 #1',
    ' #0:2 #4:2 #0:4 #4:2 #0:4 #4:2 #0:4', ' #1 #0:2 #4:2 #0:4 #1 #0:2 #1',
    ' #0:2 #4:2 #0:4 #1 #0:2 #1 #0:2', ' #1 #0:2 #1 #0:2 #1 #0:2 #1',
    ' #0:2 #1 #0:2 #4:2 #0:4 #1 #0:2', ' #1 #0:2 #1 #0:2 #1 #0:2 #4:2',
    ' #0:4 #1 #0:2 #1 #0:2 #1 #0:2', ' #1 #0:2 #4 #0:2 #1 #0 #1',
    ' #0 #1 #0 #1 #0 #1 #0', ' #1 #0 #1 #0 #1 #0:2 #4:2',
    ' #0:3 #2 #0 #2 #0 #2 #0', ' #2 #0 #2 #0 #2 #0 #2 ]', ), )
p.Stringer(elementEdges=elementEdges, name='Stringer-1')
session.viewports['Viewport: 1'].view.setValues(nearPlane=6795.99,
    farPlane=8339.65, width=977.627, height=679.485, viewOffsetX=536.172,
    viewOffsetY=192.964)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6803.88,
    farPlane=8331.75, width=978.763, height=680.275, viewOffsetX=90.5506,
    viewOffsetY=201.122)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-396.352)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-941.763,
    viewOffsetY=165.423)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-1475.27,
    viewOffsetY=136.665)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6357.06,
    farPlane=8778.57, width=4625.64, height=3214.99, viewOffsetX=-604.141,
    viewOffsetY=267.256)
p = mdb.models['deviation'].parts['PART-1']
e = p.elementEdges
elementEdges = e.getSequenceFromMask(mask=(
    '[#0:23462 #4 #0:2 #1:2 #0:8 #1:2 #0:2',
    ' #1:2 #0:14 #8 #0 #8:2 #0:10 #8:2', ' #0:2 #8:2 #0:8 #8 #0 #8:2 #0:10',
    ' #8 #0 #8 #0 #8 #0 #8', ' #0 #4:3 #0:6 #8 #0 #4 #0:5',
    ' #4 #0 #4 #0:7 #8 #0 #8', ' #0 #8 #0:5 #2 #0:4 #2:2 #0:2',
    ' #8 #0 #8 #0:3 #8 #0 #8', ' #0:4 #8 #0 #8:2 #0:4 #4 #0:2',
    ' #1 #0 #1 #0:3 #4 #0:2 #1', ' #0 #1 #0:3 #4 #0:2 #1 #0',
    ' #1 #0:3 #1 #0 #1 #0:3 #1', ' #0 #1 #0:3 #4 #0:2 #1 #0',
    ' #1 #0:3 #1 #0 #1 #0:3 #1', ' #0 #1 #0:3 #4 #0:2 #1 #0',
    ' #1 #0:3 #1 #0 #1 #0:3 #1', ' #0 #1 #0:3 #4 #0:2 #1 #0',
    ' #1 #0:3 #1 #0 #1 #0:3 #1', ' #0 #1 #0:3 #4 #0:2 #1 #0',
    ' #1 #0:3 #4 #0:2 #4 #0:2 #1', ' #0 #1 #0:3 #4 #0:2 #4 #0:2',
    ' #4 #0:2 #4 #0:2 #4 #0:2 #4', ' #0:2 #4 #0:2 #1 #0 #1 #0:3',
    ' #4 #0:2 #4 #0:2 #4 #0:2 #4', ' #0:2 #1 #0 #1 #0:3 #4 #0:2',
    ' #4 #0:2 #4 #0:2 #4 #0:2 #4', ' #0 #4 #0 #4 #0 #4 #0',
    ' #4 #0 #4 #0 #4 #0 #4', ' #0 #1:2 #0:2 #1:2 #8 #0 #8',
    ' #0 #8 #0 #8 #0 #8 #0', ' #8 #0 #8 ]', ), )
p.Stringer(elementEdges=elementEdges, name='Stringer-2')
set1 = mdb.models['deviation'].parts['PART-1'].sets['Spar1']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Spar2']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6660.81,
    farPlane=8356.44, width=1097.77, height=762.99, viewOffsetX=550.379,
    viewOffsetY=207.945)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6669.63,
    farPlane=8347.62, width=1099.22, height=764, viewOffsetX=54.3964,
    viewOffsetY=56.757)
```

```
session.viewports['Viewport: 1'].view.setValues(nearPlane=6669.63,
    farPlane=8347.62, width=1099.22, height=764, viewOffsetX=-394.426,
    viewOffsetY=141.398)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6669.63,
    farPlane=8347.62, width=1099.22, height=764, viewOffsetX=-896.706,
    viewOffsetY=53.4155)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6707.67,
    farPlane=8309.58, width=762.647, height=530.067, viewOffsetX=-958.055,
    viewOffsetY=81.9445)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6713.88,
    farPlane=8303.36, width=763.353, height=530.558, viewOffsetX=-1451.6,
    viewOffsetY=75.8332)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6200.76,
    farPlane=8816.49, width=4898.91, height=3404.91, viewOffsetX=-521.762,
    viewOffsetY=134.126)
p = mdb.models['deviation'].parts['PART-1']
e = p.elementEdges
elementEdges = e.getSequenceFromMask(mask=('[#0:23829 #4 #0 #4 #0:2 #4 #0',
    ' #4 #0 #4 #0:2 #4 #1 #0', ' #1 #0 #1 #0:2 #1 #0 #1',
    ' #0 #1 #0 #1 #0:2 #1 #0', ' #1 #0 #1 #0 #1 #0:2 #1',
    ' #0 #1 #0 #1 #0:2 #1 #0', ' #1 #0 #1 #0 #1 #0 #1',
    ' #0 #8 #0:3 #1 #0 #1 #0', ' #1 #0 #1 #0 #1 #0 #1',
    ' #0 #4 #0:3 #1 #0 #1 #0:3', ' #4 #0 #4 #0 #4 #0:2 #4',
    ' #0:2 #1:3 #0 #1 #0 #1 #0:2', ' #1 #0 #1 #0 #1 #0:2 #1',
    ' #0 #1 #0 #1 #0 #8 #0:3', ' #1 #0 #1 #0:3 #1:2 #0:2 #4',
    ' #0 #4 #0:2 #1:3 #0 #1 #0', ' #1 #0 #1 #0 #1 #0 #1',
    ' #0 #1 #0:2 #4:2 #0:4 #4 #0:2', ' #4:2 #0:3 #1 #0 #1:2 #0 #4:2',
    ' #0:2 #1 #0 #4:2 #0:3 #4 #0:2', ' #4 #0 #1 #0:3 #1 #0 #1',
    ' #0:5 #8 #0:2 #8 #0:5 #2:2 #0:4', ' #2 #0 #4 #0:2 #4 #0:6 #4:2',
    ' #0:5 #1 #0:5 #2 #0:2 #2:2 #0:7', ' #2 #0:2 #2 #0:12 #1 #0 #4:2',
    ' #0:10 #1:2 ]', ), )
p.Stringer(elementEdges=elementEdges, name='Stringer-3')
session.viewports['Viewport: 1'].view.setValues(nearPlane=6691.43,
    farPlane=8325.82, width=878.782, height=610.784, viewOffsetX=657.601,
    viewOffsetY=204.393)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6698.55,
    farPlane=8318.7, width=879.717, height=611.435, viewOffsetX=350.801,
    viewOffsetY=231.35)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6698.55,
    width=879.718, height=611.435, viewOffsetX=-160.808, viewOffsetY=196.589)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-768.678,
    viewOffsetY=211.741)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-1179.57,
    viewOffsetY=143.11)
p = mdb.models['deviation'].parts['PART-1']
e = p.elementEdges
elementEdges = e.getSequenceFromMask(mask=('[#0:23828 #1 #0 #1 #0 #2 #1',
    ' #0 #1 #0 #1 #0 #2 #1', ' #0:2 #4 #0 #4 #0 #1:2 #0',
    ' #4 #0 #4 #0 #4 #0 #1:2', ' #0 #4 #0 #4 #0 #4 #0',
    ' #1:2 #0 #4 #0 #4 #0 #2', ' #1 #0 #4 #0 #4 #0 #4',
    ' #0 #4 #0 #4 #0:2 #1:2 #0', ' #4 #0 #4 #0 #4 #0 #4',
    ' #0 #4 #0 #4 #0:2 #1:2 #0', ' #4 #0 #4 #1:2 #0 #1 #0',
    ' #1 #0 #1:2 #0 #4:2 #0:3 #4', ' #0 #4 #0 #1:2 #0 #4 #0',
    ' #4 #0 #1:2 #0 #4 #0 #4', ' #0 #4 #0:2 #1:2 #0 #4 #0',
    ' #4:3 #0:2 #2 #1 #0 #1 #0', ' #4:2 #0:3 #4 #0 #4 #0 #4',
    ' #0 #4 #0 #4 #0 #4 #0:2', ' #4 #0:2 #1 #0 #1 #0:3 #4',
    ' #0:2 #1:3 #0:4 #4 #0:2 #1:2 #0', ' #4 #0:2 #1:3 #0 #4:2 #0:3 #4:3',
    ' #0:5 #8:3 #0:5 #8:2 #0:4 #8:2 #0:8', ' #1 #0 #1 #0 #1 #0:5 #1',
    ' #8 #0:4 #1:2 #0:6 #8 #0 #8', ' #0 #8 #0:6 #8 #0 #8 #0',
    ' #8 #0:2 #4 #0 #4 #0 #4', ' #0:7 #4 #0:2 #1:2 ]', ), )
p.Stringer(elementEdges=elementEdges, name='Stringer-4')
session.viewports['Viewport: 1'].view.setValues(nearPlane=6221.1,
    farPlane=8796.15, width=4765.64, height=3312.29, viewOffsetX=40.6804,
    viewOffsetY=109.338)
mdb.models['deviation'].Material(name='AxialBar')
mdb.models['deviation'].materials['AxialBar'].Density(table=((2.7e-09, ), ))
mdb.models['deviation'].materials['AxialBar'].Elastic(table=((40000.0, 0.4), ))
mdb.models['deviation'].Material(name='Spars')
mdb.models['deviation'].materials['Spars'].Density(table=((3.6e-09, ), ))
mdb.models['deviation'].materials['Spars'].Elastic(table=((39250.0, 0.23022),
    ))
mdb.models['deviation'].Material(name='Skins')
mdb.models['deviation'].materials['Skins'].Density(table=((2.7e-09, ), ))
mdb.models['deviation'].materials['Skins'].Elastic(table=((31000.0, 0.35087),
    ))
mdb.models['deviation'].Material(name='Ribs')
mdb.models['deviation'].materials['Ribs'].Density(table=((3.6e-09, ), ))
```

```
mdb.models['deviation'].materials['Ribs'].Elastic(table=((60000.0, 0.39998), ))
mdb.models['deviation'].PipeProfile(name='Upper1', r=17.1, t=3.1)
mdb.models['deviation'].PipeProfile(name='Upper2', r=13.8, t=1.0)
mdb.models['deviation'].PipeProfile(name='Lower1', r=13.0, t=1.0)
mdb.models['deviation'].PipeProfile(name='Lower2', r=5.8, t=1.0)
mdb.models['deviation'].BeamSection(name='Lower1', profile='Lower1',
    integration=DURING_ANALYSIS, poissonRatio=0.0, material='AxialBar',
    temperatureVar=LINEAR)
mdb.models['deviation'].BeamSection(name='Lower2', profile='Lower2',
    integration=DURING_ANALYSIS, poissonRatio=0.0, material='AxialBar',
    temperatureVar=LINEAR)
mdb.models['deviation'].BeamSection(name='Upper1', profile='Upper1',
    integration=DURING_ANALYSIS, poissonRatio=0.0, material='AxialBar',
    temperatureVar=LINEAR)
mdb.models['deviation'].BeamSection(name='Upper2', profile='Upper2',
    integration=DURING_ANALYSIS, poissonRatio=0.0, material='AxialBar',
    temperatureVar=LINEAR)
mdb.models['deviation'].HomogeneousShellSection(name='Skin_Tra',
    preIntegrate=OFF, material='Skins', thickness=30.0,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Skin_Lower',
    preIntegrate=OFF, material='Skins', thickness=1.0,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Skin_Upper',
    preIntegrate=OFF, material='Skins', thickness=1.0,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Skin_Tip',
    preIntegrate=OFF, material='Skins', thickness=50.0,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Skin_Root',
    preIntegrate=OFF, material='Skins', thickness=20.0,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib01', preIntegrate=OFF,
    material='Ribs', thickness=9.8, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib02', preIntegrate=OFF,
    material='Ribs', thickness=26.0, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib03', preIntegrate=OFF,
    material='Ribs', thickness=4.0, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib04', preIntegrate=OFF,
    material='Ribs', thickness=20.0, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib05', preIntegrate=OFF,
    material='Ribs', thickness=25.0, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib06', preIntegrate=OFF,
    material='Ribs', thickness=30.4, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib07', preIntegrate=OFF,
    material='Ribs', thickness=9.2, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib08', preIntegrate=OFF,
    material='Ribs', thickness=2.4, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib09', preIntegrate=OFF,
    material='Ribs', thickness=5.8, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
```

```
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib10', preIntegrate=OFF,
    material='Ribs', thickness=1.2, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib11', preIntegrate=OFF,
    material='Ribs', thickness=1.4, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib12', preIntegrate=OFF,
    material='Ribs', thickness=6.0, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib13', preIntegrate=OFF,
    material='Ribs', thickness=19.6, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib14', preIntegrate=OFF,
    material='Ribs', thickness=7.8, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib15', preIntegrate=OFF,
    material='Ribs', thickness=2.2, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib16', preIntegrate=OFF,
    material='Ribs', thickness=40.4, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Rib17', preIntegrate=OFF,
    material='Ribs', thickness=41.2, poissonDefinition=DEFAULT,
    thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
    nodalThicknessField='', integrationRule=SIMPSON, numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Spar01',
    preIntegrate=OFF, material='Spars', thickness=1.0,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Spar02',
    preIntegrate=OFF, material='Spars', thickness=4.0,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Spar03',
    preIntegrate=OFF, material='Spars', thickness=4.0,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Spar04',
    preIntegrate=OFF, material='Spars', thickness=2.0,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
mdb.models['deviation'].HomogeneousShellSection(name='Spar05',
    preIntegrate=OFF, material='Spars', thickness=5.2,
    poissonDefinition=DEFAULT, thicknessModulus=None, temperature=GRADIENT,
    useDensity=OFF, nodalThicknessField='', integrationRule=SIMPSON,
    numIntPts=5)
set1 = mdb.models['deviation'].parts['PART-1'].sets['Spar2']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6080.72,
    farPlane=9216.12, width=4658.1, height=3237.54, viewOffsetX=-82.9433,
    viewOffsetY=-81.9077)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5246.46,
    farPlane=9373.18, width=4019.02, height=2793.36, cameraPosition=(8056.21,
    2007.03, 2914.6), cameraUpVector=(-0.435767, 0.129486, 0.890697),
    viewOffsetX=-71.5637, viewOffsetY=-70.6702)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5521.58,
    farPlane=9098.08, width=2423.64, height=1684.52, viewOffsetX=-44.4642,
    viewOffsetY=-120.725)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5539.61,
    farPlane=9080.04, width=2431.55, height=1690.02, viewOffsetX=120.451,
    viewOffsetY=-32.4299)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
```

```
elements = e.getSequenceFromMask(mask=('[#0:827 #fff80000 #ffffffff:3 #7ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Upper1', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:827 #fff80000 #ffffffff:3 #7ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Upper1', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:835 #fffffff8 #ffffffff:2 #7ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Upper2', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:835 #fffffff8 #ffffffff:2 #7ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Upper2', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:831 #fffff800 #ffffffff:3 #7 ]', ),
    )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Lower1', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:831 #fffff800 #ffffffff:3 #7 ]', ),
    )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Lower1', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:838 #fffff800 #ffffffff:3 #7f ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Lower2', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:838 #fffff800 #ffffffff:3 #7f ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Lower2', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#ffffffff:340 #7fffffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Upper', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#ffffffff:340 #7fffffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Upper', offset=0.0)
set1 = mdb.models['deviation'].parts['PART-1'].sets['Lower']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:340 #80000000 #ffffffff:371 #7fffffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Lower', offset=0.0)
```

```
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:340 #80000000 #ffffffff:371 #7fffffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Lower', offset=0.0)
set1 = mdb.models['deviation'].parts['PART-1'].sets['Lower']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Tip']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:721 #ff800000 #ffffffff #3fffff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Tip', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:721 #ff800000 #ffffffff #3fffff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Tip', offset=0.0)
set1 = mdb.models['deviation'].parts['PART-1'].sets['AuxSpar']
set2 = mdb.models['deviation'].parts['PART-1'].sets['Ribs']
set3 = mdb.models['deviation'].parts['PART-1'].sets['Root']
set4 = mdb.models['deviation'].parts['PART-1'].sets['Spar1']
set5 = mdb.models['deviation'].parts['PART-1'].sets['Spar2']
set6 = mdb.models['deviation'].parts['PART-1'].sets['Tra']
leaf = dgm.LeafFromSets(sets=(set1, set2, set3, set4, set5, set6, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:723 #ffc00000 #ffffffff:9 #1f ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Tra', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:723 #ffc00000 #ffffffff:9 #1f ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Tra', offset=0.0)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5554.15,
    farPlane=9066.07, width=2189.64, height=1521.88, viewOffsetX=-75.2251,
    viewOffsetY=-71.5781)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:712 #80000000 #ffffffff:8 #7fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Root', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:712 #80000000 #ffffffff:8 #7fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Skin_Root', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:733 #ffffffe0 #ffffffff:10 #fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Spar01', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:733 #ffffffe0 #ffffffff:10 #fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
```

```
p.SectionAssignment(region=region, sectionName='Spar01', offset=0.0)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5644.41,
    farPlane=8975.81, width=1737.34, height=1207.51, viewOffsetX=-58.0833,
    viewOffsetY=-45.6684)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:744 #fff00000 #ffffffff:8 #1fff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Spar02', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:744 #fff00000 #ffffffff:8 #1fff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Spar02', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:753 #ffffe000 #1fff #7fe00000 #0 #3ffffc00 ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Spar03', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:753 #ffffe000 #1fff #7fe00000 #0 #3ffffc00 ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Spar03', offset=0.0)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5631.69,
    farPlane=8988.53, width=1733.42, height=1204.79, viewOffsetX=129.967,
    viewOffsetY=-31.5154)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5627.21,
    farPlane=8870.11, width=1732.04, height=1203.83, cameraPosition=(7770.96,
    444.394, 3474.56), cameraUpVector=(-0.483216, 0.148934, 0.862741),
    viewOffsetX=129.864, viewOffsetY=-31.4903)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:754 #7ffe000 #801ffe00 #c0001fff #c00003ff #1ff801ff #fc007ff0',
    ' #fe0001ff #fffc003f #ff000000 #1fffff #c000fffe #f8003fff #ffe000ff',
    ' #7f80001 ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Spar04', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:754 #7ffe000 #801ffe00 #c0001fff #c00003ff #1ff801ff #fc007ff0',
    ' #fe0001ff #fffc003f #ff000000 #1fffff #c000fffe #f8003fff #ffe000ff',
    ' #7f80001 ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Spar04', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:754 #f8000000 #1ff #3fffe000 #0 #e007fe00 #3ff800f',
    ' #1fffe00 #3ffc0 #ffffff #ffe00000 #3fff0001 #7ffc000 #1fff00',
    ' #f807fffe #7f ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Spar05', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:754 #f8000000 #1ff #3fffe000 #0 #e007fe00 #3ff800f',
    ' #1fffe00 #3ffc0 #ffffff #ffe00000 #3fff0001 #7ffc000 #1fff00',
    ' #f807fffe #7f ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Spar05', offset=0.0)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5627.38,
    farPlane=8869.93, width=1732.09, height=1203.87, viewOffsetX=-126.349,
```

```
    viewOffsetY=63.2738)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:817 #fffff800 #7ffffff #ff000000 #3fffff #0:3 #ffc00000',
    ' #ffffffff:2 #7ffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib01', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:817 #fffff800 #7ffffff #ff000000 #3fffff #0:3 #ffc00000',
    ' #ffffffff:2 #7ffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib01', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:818 #f8000000 #ffffff #ffc00000 #ffffffff:3 #3fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib02', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:818 #f8000000 #ffffff #ffc00000 #ffffffff:3 #3fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib02', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:812 #ffc00000 #ffffffff:4 #7ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib03', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:812 #ffc00000 #ffffffff:4 #7ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib03', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:808 #ffffff00 #ffffffff:3 #3fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib04', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:808 #ffffff00 #ffffffff:3 #3fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib04', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:804 #fffffff8 #ffffffff:3 #ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib05', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:804 #fffffff8 #ffffffff:3 #ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib05', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:800 #ffc00000 #ffffffff:3 #7 ]', ),
    )
```

```
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib06', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:800 #ffc00000 #ffffffff:3 #7 ]', ),
    )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib06', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:797 #fffffff0 #ffffffff:2 #3fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib07', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:797 #fffffff0 #ffffffff:2 #3fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib07', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:793 #ffff0000 #ffffffff:3 #f ]', ),
    )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib08', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:793 #ffff0000 #ffffffff:3 #f ]', ),
    )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib08', offset=0.0)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5627.38,
    farPlane=8869.93, width=1732.1, height=1203.87, viewOffsetX=0.00469208,
    viewOffsetY=43.9698)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=593.166,
    viewOffsetY=-205.228)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:790 #fffffe00 #ffffffff:2 #ffff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib09', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:790 #fffffe00 #ffffffff:2 #ffff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib09', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:786 #ff800000 #ffffffff:3 #1ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib10', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:786 #ff800000 #ffffffff:3 #1ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib10', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:783 #ffc00000 #ffffffff:2 #7fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
```

```
p.SectionAssignment(region=region, sectionName='Rib11', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:783 #ffc00000 #ffffffff:2 #7fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib11', offset=0.0)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=826.57,
    viewOffsetY=-321.052)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5676.22,
    farPlane=8821.09, width=1282.23, height=891.194, viewOffsetX=627.059,
    viewOffsetY=-263.076)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:780 #fe000000 #ffffffff:2 #3fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib12', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:780 #fe000000 #ffffffff:2 #3fffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib12', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:778 #fffffffe #ffffffff #1ffffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib13', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=(
    '[#0:778 #fffffffe #ffffffff #1ffffff ]', ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib13', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:775 #fffff800 #ffffffff:2 #1 ]', ),
    )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib14', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:775 #fffff800 #ffffffff:2 #1 ]', ),
    )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib14', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:772 #ff800000 #ffffffff:2 #7ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib15', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:772 #ff800000 #ffffffff:2 #7ff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib15', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:770 #ffff0000 #ffffffff #7fffff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib16', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
```

```
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:770 #ffff0000 #ffffffff #7fffff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib16', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:768 #ffffff80 #ffffffff #ffff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib17', offset=0.0)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:768 #ffffff80 #ffffffff #ffff ]',
    ), )
region = regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.SectionAssignment(region=region, sectionName='Rib17', offset=0.0)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5686.35,
    farPlane=8810.97, width=1284.52, height=892.784, viewOffsetX=628.177,
    viewOffsetY=-263.545)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5604.29,
    farPlane=8893.02, width=1879.79, height=1306.52, viewOffsetX=619.112,
    viewOffsetY=-259.742)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5607.76,
    farPlane=8889.55, width=1880.95, height=1307.33, viewOffsetX=619.495,
    viewOffsetY=-259.903)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5759.14,
    farPlane=8738.17, width=751.855, height=522.566, viewOffsetX=636.218,
    viewOffsetY=-266.919)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5756.58,
    farPlane=8740.73, width=751.521, height=522.334, viewOffsetX=635.935,
    viewOffsetY=-266.8)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5714.38,
    farPlane=8782.93, width=1063.96, height=739.489, viewOffsetX=631.273,
    viewOffsetY=-264.844)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5715.39,
    farPlane=8781.92, width=1064.15, height=739.621, viewOffsetX=631.385,
    viewOffsetY=-264.891)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5608.97,
    farPlane=8888.34, width=1836.96, height=1276.75, viewOffsetX=619.629,
    viewOffsetY=-259.959)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5613.37,
    farPlane=8883.95, width=1838.4, height=1277.75, viewOffsetX=620.115,
    viewOffsetY=-260.163)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5505.77,
    farPlane=8991.54, width=2604.15, height=1809.97, viewOffsetX=608.229,
    viewOffsetY=-255.176)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5512.06,
    farPlane=8985.25, width=2607.13, height=1812.04, viewOffsetX=-141.252,
    viewOffsetY=58.8667)
set1 = mdb.models['deviation'].parts['PART-1'].sets['Lower']
leaf = dgm.LeafFromSets(sets=(set1, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
set1 = mdb.models['deviation'].parts['PART-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, ))
session.viewports['Viewport: 1'].partDisplay.displayGroup.either(leaf=leaf)
p = mdb.models['deviation'].parts['PART-1']
e = p.elements
elements = e.getSequenceFromMask(mask=('[#0:827 #fff80000 #ffffffff:14 #7f ]',
    ), )
region=regionToolset.Region(elements=elements)
p = mdb.models['deviation'].parts['PART-1']
p.assignBeamSectionOrientation(region=region, method=N1_COSINES, n1=(0.0, 0.0,
    -1.0))
#: Beam orientations have been assigned to the selected regions.
mdb.save()
#: The model database has been saved to "arw.cae".
a = mdb.models['deviation'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
a1 = mdb.models['deviation'].rootAssembly
a1.DatumCsysByDefault(CARTESIAN)
p = mdb.models['deviation'].parts['PART-1']
a1.Instance(name='PART-1-1', part=p, dependent=ON)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
```

```
    adaptiveMeshConstraints=ON)
mdb.models['deviation'].FrequencyStep(name='Step-1', previous='Initial',
    numEigen=5)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Step-1')
session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON, connectors=ON, adaptiveMeshConstraints=OFF)
session.viewports['Viewport: 1'].view.setValues(cameraPosition=(1551.44,
    910.229, 7314.48), cameraUpVector=(0, 1, 0))
set1 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, ))
session.viewports['Viewport: 1'].assemblyDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6733.49,
    farPlane=7975.51, width=3879.57, height=2696.44, viewOffsetX=-911.915,
    viewOffsetY=-279.077)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6890.72,
    farPlane=7818.27, width=2574.56, height=1789.41, viewOffsetX=-1072.19,
    viewOffsetY=-188.378)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6910.23,
    farPlane=7798.76, width=2581.85, height=1794.48, viewOffsetX=-897.348,
    viewOffsetY=-322.32)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6066.22,
    farPlane=8545.79, width=2266.51, height=1575.3, cameraPosition=(-2369.59,
    1269.38, 6182.49), cameraUpVector=(-0.0788834, 0.991242, -0.105909),
    viewOffsetX=-787.747, viewOffsetY=-282.952)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6568.1,
    farPlane=8093.16, width=2454.02, height=1705.63, cameraPosition=(-179.299,
    1088.55, 7107.43), cameraUpVector=(-0.104985, 0.993227, -0.0497881),
    viewOffsetX=-852.92, viewOffsetY=-306.361)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6546.55,
    farPlane=8114.71, width=2445.97, height=1700.04, viewOffsetX=-887.295,
    viewOffsetY=-325.181)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6334.13,
    farPlane=8303.01, width=2366.61, height=1644.88, cameraPosition=(-1196.38,
    1138.13, 6782.69), cameraUpVector=(0.0323912, 0.999272, -0.0201586),
    viewOffsetX=-858.504, viewOffsetY=-314.63)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6521.57,
    farPlane=8115.56, width=963.186, height=669.449, viewOffsetX=-1091.26,
    viewOffsetY=-56.6488)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6529.34,
    farPlane=8107.79, width=964.334, height=670.246, viewOffsetX=-1072.04,
    viewOffsetY=-272.641)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6494.79,
    farPlane=8142.34, width=1307.03, height=908.429, viewOffsetX=-1036.31,
    viewOffsetY=-246.978)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6484.47,
    farPlane=8152.66, width=1304.95, height=906.986, viewOffsetX=-975.166,
    viewOffsetY=-472.671)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6473.49,
    farPlane=8163.64, width=1474.36, height=1024.73, viewOffsetX=-949.085,
    viewOffsetY=-460.922)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6462.49,
    farPlane=8174.64, width=1471.85, height=1022.99, viewOffsetX=-929.577,
    viewOffsetY=-482.507)
session.viewports['Viewport: 1'].view.setValues(cameraPosition=(-1196.38,
    1138.13, 6782.69), cameraUpVector=(-0.380919, 0.906497, -0.182109))
session.viewports['Viewport: 1'].view.setValues(nearPlane=6484.74,
    farPlane=8152.39, width=1226.71, height=852.605, viewOffsetX=-992.473,
    viewOffsetY=-487.656)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6494.53,
    farPlane=8142.6, width=1228.56, height=853.891, viewOffsetX=-1054.96,
    viewOffsetY=-25.3491)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6562.76,
    farPlane=8074.37, width=668.673, height=464.752, viewOffsetX=-1112.86,
    viewOffsetY=-78.7983)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6568.23,
    farPlane=8068.9, width=669.23, height=465.139, viewOffsetX=-1029.03,
    viewOffsetY=-154.127)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6460.6,
    farPlane=8176.53, width=1578.18, height=1096.89, viewOffsetX=-1143.73,
    viewOffsetY=-260.105)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6448.86,
    farPlane=8188.27, width=1575.31, height=1094.9, viewOffsetX=-967.684,
    viewOffsetY=291.009)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6407.55,
    farPlane=8229.58, width=2004.77, height=1393.39, viewOffsetX=-966.606,
    viewOffsetY=338.132)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6392.89,
```

```
    farPlane=8244.24, width=2000.19, height=1390.2, viewOffsetX=-676.627,
    viewOffsetY=136.732)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6249.28,
    farPlane=8441.31, width=1955.26, height=1358.97, cameraPosition=(-603.881,
    -1561.57, 6549.69), cameraUpVector=(-0.563055, 0.817458, 0.121374),
    viewOffsetX=-661.427, viewOffsetY=133.66)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6254.95,
    farPlane=8435.63, width=1957.03, height=1360.21, viewOffsetX=-715.563,
    viewOffsetY=203.18)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6254.96,
    farPlane=8435.62, width=1957.04, height=1360.21, cameraPosition=(-603.881,
    -1561.57, 6549.69), cameraUpVector=(-0.125315, 0.94252, 0.309761),
    viewOffsetX=-715.564, viewOffsetY=203.18)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-1082.39,
    viewOffsetY=-94.2426)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6309.36,
    farPlane=8381.22, width=1639.63, height=1139.6, viewOffsetX=-1114.24,
    viewOffsetY=25.4386)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6297.21,
    farPlane=8393.37, width=1636.47, height=1137.4, viewOffsetX=-946.292,
    viewOffsetY=-302.899)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6185.24,
    farPlane=8505.35, width=2636.9, height=1832.74, viewOffsetX=-817.475,
    viewOffsetY=-346.516)
set1 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Lower']
set2 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].assemblyDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6395.02,
    farPlane=8295.57, width=841.411, height=584.81, viewOffsetX=-1231.1,
    viewOffsetY=131.082)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6401.84,
    farPlane=8288.75, width=842.308, height=585.434, viewOffsetX=-1171.83,
    viewOffsetY=-42.0184)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6332.43,
    farPlane=8358.16, width=1454.07, height=1010.63, viewOffsetX=-1175.76,
    viewOffsetY=27.7708)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6321.57,
    farPlane=8369.01, width=1451.58, height=1008.9, viewOffsetX=-1148.74,
    viewOffsetY=-189.94)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-1145.8,
    viewOffsetY=-167.879)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6757.84,
    farPlane=7945.37, width=1551.76, height=1078.53, cameraPosition=(773.41,
    -241.889, 7182.97), cameraUpVector=(-0.129596, 0.98143, 0.141423),
    viewOffsetX=-1224.88, viewOffsetY=-179.465)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6745.04,
    farPlane=7958.17, width=1548.82, height=1076.49, viewOffsetX=-1220.99,
    viewOffsetY=-259.155)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6747.74,
    farPlane=8000.83, width=1549.45, height=1076.92, cameraPosition=(2177.01,
    -634.011, 7124.84), cameraUpVector=(-0.463913, 0.857246, 0.223415),
    viewOffsetX=-1221.48, viewOffsetY=-259.259)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6747.66,
    farPlane=8000.91, width=1549.43, height=1076.91, viewOffsetX=-1201.06,
    viewOffsetY=108.086)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6579.35,
    farPlane=8194.76, width=1510.78, height=1050.05, cameraPosition=(2448.38,
    -1663.99, 6795.13), cameraUpVector=(-0.33555, 0.867378, 0.367507),
    viewOffsetX=-1171.1, viewOffsetY=105.39)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6584.29,
    farPlane=8189.82, width=1511.92, height=1050.84, viewOffsetX=-1263.89,
    viewOffsetY=-55.3734)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6584.29,
    farPlane=8189.81, width=1511.92, height=1050.84, viewOffsetX=-1084.67,
    viewOffsetY=159.084)
session.viewports['Viewport: 1'].view.setValues(cameraPosition=(350.339,
    630.188, 7366.18), cameraUpVector=(0, 1, 0), cameraTarget=(350.339,
    630.188, -47.6956), viewOffsetX=0, viewOffsetY=0)
session.viewports['Viewport: 1'].view.setValues(nearPlane=7012.38,
    farPlane=7800.01, width=2334.1, height=1622.28, viewOffsetX=-25.3026,
    viewOffsetY=2.20023)
set1 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Lower']
set2 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].assemblyDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6995.4,
```

```
    farPlane=7816.99, width=2328.45, height=1618.35, viewOffsetX=69.1233,
    viewOffsetY=-68.5785)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6916.79,
    farPlane=8118.55, width=2302.29, height=1600.17, cameraPosition=(-220.483,
    14.6936, 7318.5), cameraUpVector=(-0.0174268, 0.996487, 0.0819126),
    viewOffsetX=68.3466, viewOffsetY=-67.8079)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6920.09,
    farPlane=8115.25, width=2303.39, height=1600.94, viewOffsetX=187.399,
    viewOffsetY=-56.1716)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6631.48,
    farPlane=9127.69, width=2207.33, height=1534.17, cameraPosition=(-1865.8,
    -2661.68, 6214.72), cameraUpVector=(-0.00416913, 0.885759, 0.464128),
    viewOffsetX=179.583, viewOffsetY=-53.8289)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6643.58,
    farPlane=9115.58, width=2211.36, height=1536.97, viewOffsetX=278.492,
    viewOffsetY=55.8566)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6816.92,
    farPlane=8942.24, width=843.124, height=586.001, viewOffsetX=-38.5574,
    viewOffsetY=269.977)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6823.77,
    farPlane=8935.39, width=843.971, height=586.59, viewOffsetX=-6.95788,
    viewOffsetY=23.1275)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6754.13,
    farPlane=9005.04, width=1457.88, height=1013.28, viewOffsetX=31.7655,
    viewOffsetY=103.564)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6743.21,
    farPlane=9015.96, width=1455.52, height=1011.64, viewOffsetX=103.974,
    viewOffsetY=-72.0918)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6378.09,
    farPlane=8593.69, width=1376.71, height=956.864, cameraPosition=(1324.69,
    -4420.59, 5291.39), cameraUpVector=(-0.490844, 0.586429, 0.644339),
    viewOffsetX=98.3442, viewOffsetY=-68.1883)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6509.13,
    farPlane=8462.66, width=433.615, height=301.378, viewOffsetX=-27.1302,
    viewOffsetY=-223.217)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6512.71,
    farPlane=8459.08, width=433.854, height=301.544, viewOffsetX=-24.0681,
    viewOffsetY=-210.592)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6685.06,
    farPlane=8055.71, width=445.336, height=309.524, cameraPosition=(1364.73,
    -1959.52, 6824.71), cameraUpVector=(-0.453981, 0.809599, 0.372088),
    viewOffsetX=-24.705, viewOffsetY=-216.165)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6683.56,
    farPlane=8057.2, width=445.236, height=309.455, viewOffsetX=-35.0748,
    viewOffsetY=-290.999)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6516.8,
    farPlane=8223.97, width=1816.45, height=1262.49, viewOffsetX=44.3894,
    viewOffsetY=-68.1547)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6503.4,
    farPlane=8237.37, width=1812.71, height=1259.9, viewOffsetX=130.618,
    viewOffsetY=-69.8512)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6932.11,
    farPlane=8251.06, width=1932.21, height=1342.95, cameraPosition=(-592.768,
    -412.966, 7231.58), cameraUpVector=(-0.142403, 0.982223, 0.122307),
    viewOffsetX=139.229, viewOffsetY=-74.4559)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6916.94,
    farPlane=8266.22, width=1927.98, height=1340.02, viewOffsetX=138.924,
    viewOffsetY=89.7906)
session.viewports['Viewport: 1'].view.setValues(nearPlane=7005.64,
    farPlane=7909.14, width=1952.71, height=1357.2, cameraPosition=(-37.5807,
    770.994, 7354.68), cameraUpVector=(-0.148235, 0.988595, -0.0265733),
    viewOffsetX=140.706, viewOffsetY=90.9421)
set1 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Lower']
set2 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].assemblyDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6931.4,
    farPlane=7983.39, width=2643.29, height=1837.18, viewOffsetX=170.621,
    viewOffsetY=235.231)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6912.4,
    farPlane=8002.38, width=2636.04, height=1832.14, viewOffsetX=74.0059,
    viewOffsetY=39.6208)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6601.29,
    farPlane=9560.22, width=2517.4, height=1749.68, cameraPosition=(-3525.79,
    -1643.16, 5849.17), cameraUpVector=(-0.0371787, 0.940378, 0.338094),
    viewOffsetX=70.6751, viewOffsetY=37.8376)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6616.14,
```

```
    farPlane=9545.38, width=2523.06, height=1753.62, viewOffsetX=132.185,
    viewOffsetY=163.181)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6801.11,
    farPlane=8479.58, width=2593.6, height=1802.65, cameraPosition=(-825.001,
    -739.507, 7143.13), cameraUpVector=(0.053723, 0.979269, 0.19531),
    viewOffsetX=135.881, viewOffsetY=167.743)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6986.13,
    farPlane=8294.56, width=1053.12, height=731.958, viewOffsetX=-117.705,
    viewOffsetY=368.472)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6994.62,
    farPlane=8286.07, width=1054.4, height=732.848, viewOffsetX=-58.0239,
    viewOffsetY=-29.5533)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6921.18,
    farPlane=8359.51, width=1711.59, height=1189.62, viewOffsetX=14.2195,
    viewOffsetY=-86.1161)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6908.46,
    farPlane=8372.23, width=1708.45, height=1187.43, viewOffsetX=21.1172,
    viewOffsetY=-156.927)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6145.38,
    farPlane=8430.49, width=1519.74, height=1056.27, cameraPosition=(2227.11,
    -2971.28, 6154.93), cameraUpVector=(-0.267589, 0.795895, 0.543091),
    viewOffsetX=18.7847, viewOffsetY=-139.594)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6279.04,
    farPlane=8296.82, width=652.986, height=453.849, viewOffsetX=-40.4579,
    viewOffsetY=-232.08)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6284.37,
    farPlane=8291.49, width=653.541, height=454.234, viewOffsetX=-17.9792,
    viewOffsetY=-243.534)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6067.33,
    farPlane=8404.94, width=630.97, height=438.546, cameraPosition=(2746.25,
    -3527.41, 5603.8), cameraUpVector=(-0.448786, 0.618353, 0.645159),
    viewOffsetX=-17.3583, viewOffsetY=-235.123)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6070.26,
    farPlane=8402.01, width=631.275, height=438.758, viewOffsetX=-96.0362,
    viewOffsetY=-183.43)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6131.08,
    farPlane=8379.03, width=637.6, height=443.155, cameraPosition=(2587.72,
    -3427.57, 5739.71), cameraUpVector=(-0.446176, 0.642514, 0.622979),
    viewOffsetX=-96.9984, viewOffsetY=-185.268)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6098.31,
    farPlane=8411.79, width=875.364, height=608.409, viewOffsetX=-96.4799,
    viewOffsetY=-184.278)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6098.9,
    farPlane=8411.2, width=875.449, height=608.468, viewOffsetX=-202.927,
    viewOffsetY=-175.426)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6063.53,
    farPlane=8446.57, width=1137.29, height=790.454, viewOffsetX=-201.75,
    viewOffsetY=-174.409)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6064.38,
    farPlane=8445.72, width=1137.45, height=790.568, viewOffsetX=-201.778,
    viewOffsetY=-174.433)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6238.71,
    farPlane=8312.49, width=1170.15, height=813.294, cameraPosition=(2232.45,
    -2711.65, 6297), cameraUpVector=(-0.263109, 0.819228, 0.509548),
    viewOffsetX=-207.578, viewOffsetY=-179.447)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6090.46,
    farPlane=8460.74, width=2400.28, height=1668.28, viewOffsetX=-155.056,
    viewOffsetY=80.9321)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6073.26,
    farPlane=8477.94, width=2393.51, height=1663.57, viewOffsetX=-6.69122,
    viewOffsetY=75.8535)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6480.11,
    farPlane=8239.88, width=2553.85, height=1775.02, cameraPosition=(1307.9,
    -1535.1, 6977.97), cameraUpVector=(-0.18034, 0.932789, 0.312062),
    viewOffsetX=-7.13947, viewOffsetY=80.9349)
set1 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Lower']
set2 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].assemblyDisplay.displayGroup.either(leaf=leaf)
a = mdb.models['deviation'].rootAssembly
n1 = a.instances['PART-1-1'].nodes
nodes1 = n1.getSequenceFromMask(mask=(
    '[#ffffffff:39 #3fffff #0:580 #1c000001 #ff000000 #1 #fe000000',
    ' #ffffffff:8 #1ffff #3fc000 #f0000000 #1fff #0:20 #f0000000',
    ' #ffffffff:34 #ffffff #1c00000 #0:11 #700000 #10000c0 #0:29',
    ' #7ffff #fffffe #0:2 #fffff800 #ffffffff #fff ]', ), )
region = regionToolset.Region(nodes=nodes1)
```

```
mdb.models['deviation'].EncastreBC(name='BC-1', createStepName='Step-1',
    region=region)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6460.08,
    farPlane=8259.91, width=2545.96, height=1769.53, viewOffsetX=16.098,
    viewOffsetY=-89.5619)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5115.25,
    farPlane=8754.32, width=2015.95, height=1401.16, cameraPosition=(4832.95,
    -3655.22, 4015.16), cameraUpVector=(-0.0758582, 0.643096, 0.762019),
    viewOffsetX=12.7468, viewOffsetY=-70.9173)
set1 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, ))
session.viewports['Viewport: 1'].assemblyDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5181.45,
    farPlane=8688.12, width=2042.05, height=1419.29, viewOffsetX=530.147,
    viewOffsetY=-34.5942)
session.viewports['Viewport: 1'].view.setValues(nearPlane=4995.68,
    farPlane=8873.89, width=3655.37, height=2540.61, viewOffsetX=742.674,
    viewOffsetY=226.375)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=OFF, bcs=OFF,
    predefinedFields=OFF, connectors=OFF)
mdb.Job(name='vibrate', model='deviation', type=ANALYSIS,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, description='',
    parallelizationMethodExplicit=DOMAIN, multiprocessingMode=DEFAULT,
    numDomains=1, userSubroutine='', numCpus=1, preMemory=512.0,
    standardMemory=512.0, standardMemoryPolicy=MODERATE, scratch='',
    echoPrint=OFF, modelPrint=OFF, contactPrint=OFF, historyPrint=OFF)
mdb.jobs['vibrate'].submit(consistencyChecking=OFF)
#: The job input file "vibrate.inp" has been submitted for analysis.
#: Job vibrate: Analysis Input File Processor completed successfully.
#: Job vibrate: Abaqus/Standard completed successfully.
#: Job vibrate completed successfully.
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    adaptiveMeshConstraints=ON)
del mdb.models['deviation'].steps['Step-1']
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Initial')
mdb.models['deviation'].StaticStep(name='Step-1', previous='Initial',
    maxNumInc=400)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Step-1')
session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON, connectors=ON, adaptiveMeshConstraints=OFF)
session.viewports['Viewport: 1'].view.setValues(width=3655.37, cameraPosition=(
    1198.75, 1399.87, 7241.94), cameraUpVector=(0, 1, 0), cameraTarget=(
    1198.75, 1399.87, -171.924), viewOffsetX=0, viewOffsetY=0)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6528.88,
    farPlane=8035.03, width=4802.8, height=3338.12, viewOffsetX=-267.633,
    viewOffsetY=-510.937)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6614.25,
    farPlane=7949.66, width=4041.29, height=2808.84, viewOffsetX=-436.297,
    viewOffsetY=-473.859)
set1 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, ))
session.viewports['Viewport: 1'].assemblyDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6778.82,
    farPlane=7785.09, width=2857.34, height=1985.95, viewOffsetX=-732.372,
    viewOffsetY=-535.019)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6800.18,
    farPlane=7763.73, width=2866.34, height=1992.21, viewOffsetX=-458.791,
    viewOffsetY=-763.224)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6800.25,
    farPlane=7763.66, width=2866.37, height=1992.23, viewOffsetX=-412.33,
    viewOffsetY=-882.301)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5771.4,
    farPlane=8759.49, width=2432.7, height=1690.81, cameraPosition=(-2325.65,
    -1278.12, 5775.54), cameraUpVector=(-0.0967619, 0.927763, 0.360407),
    viewOffsetX=-349.946, viewOffsetY=-748.813)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5825.37,
    farPlane=8705.52, width=2455.45, height=1706.62, viewOffsetX=-303.463,
    viewOffsetY=-723.474)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6391.46,
    farPlane=8190.26, width=2694.06, height=1872.47, cameraPosition=(-613.497,
    451.568, 6954.21), cameraUpVector=(-0.0173398, 0.99168, 0.127556),
    viewOffsetX=-332.952, viewOffsetY=-793.779)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6579.06,
    farPlane=8002.66, width=1096.2, height=761.898, viewOffsetX=-753.764,
    viewOffsetY=-399.192)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6570.75,
    farPlane=8010.97, width=1094.81, height=760.935, viewOffsetX=-710.661,
```

```
    viewOffsetY=-761.408)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6481.09,
    farPlane=8100.63, width=1884.62, height=1309.88, viewOffsetX=-580.122,
    viewOffsetY=-697.209)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6467.23,
    farPlane=8114.49, width=1880.59, height=1307.07, viewOffsetX=-355.955,
    viewOffsetY=-1017.72)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6467.21,
    farPlane=8114.51, width=1880.58, height=1307.07, viewOffsetX=-321.658,
    viewOffsetY=-1053.92)
session.viewports['Viewport: 1'].view.setValues(cameraPosition=(-613.497,
    451.568, 6954.21), cameraUpVector=(-0.34847, 0.936628, 0.0360202))
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-847.535,
    viewOffsetY=-554.719)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5855.86,
    farPlane=8100.89, width=1702.81, height=1183.51, cameraPosition=(1657.44,
    -4161.04, 4709.8), cameraUpVector=(-0.402782, 0.584855, 0.704068),
    viewOffsetX=-767.417, viewOffsetY=-502.281)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5878.39,
    farPlane=8078.36, width=1709.36, height=1188.07, viewOffsetX=-784.225,
    viewOffsetY=-22.752)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6065.04,
    farPlane=7891.71, width=275.579, height=191.537, viewOffsetX=-945.978,
    viewOffsetY=-238.638)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6067.33,
    farPlane=7889.43, width=275.683, height=191.609, viewOffsetX=-941.029,
    viewOffsetY=-238.169)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6215.62,
    farPlane=7824.36, width=282.421, height=196.293, cameraPosition=(1702.88,
    -3162.09, 5650.42), cameraUpVector=(-0.451843, 0.682807, 0.574119),
    viewOffsetX=-964.029, viewOffsetY=-243.99)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6214.74,
    farPlane=7825.24, width=282.382, height=196.265, viewOffsetX=-990.214,
    viewOffsetY=-321.775)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6505.6,
    farPlane=7722.44, width=295.598, height=205.451, cameraPosition=(868.49,
    -1805.1, 6505.23), cameraUpVector=(-0.468727, 0.805179, 0.363293),
    viewOffsetX=-1036.56, viewOffsetY=-336.835)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6503.87,
    farPlane=7724.17, width=295.519, height=205.396, viewOffsetX=-1047.06,
    viewOffsetY=-446.63)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6284.97,
    farPlane=7943.07, width=2076.82, height=1443.46, viewOffsetX=-970.471,
    viewOffsetY=-369.792)
set1 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Lower']
set2 = mdb.models['deviation'].rootAssembly.instances['PART-1-1'].sets['Upper']
leaf = dgm.LeafFromSets(sets=(set1, set2, ))
session.viewports['Viewport: 1'].assemblyDisplay.displayGroup.either(leaf=leaf)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6269.85,
    farPlane=7958.19, width=2071.82, height=1439.99, viewOffsetX=-1018.51,
    viewOffsetY=-211.469)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6756.45,
    farPlane=7936.47, width=2232.62, height=1551.75, cameraPosition=(570.191,
    2107.37, 7181.28), cameraUpVector=(-0.0516678, 0.993643, -0.100023),
    viewOffsetX=-1097.56, viewOffsetY=-227.881)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6716.27,
    farPlane=7976.64, width=2521.98, height=1752.87, viewOffsetX=-525.897,
    viewOffsetY=-236.469)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6698.08,
    farPlane=7994.83, width=2515.15, height=1748.12, viewOffsetX=-904.166,
    viewOffsetY=-702.163)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6824.01,
    farPlane=7868.9, width=1468.26, height=1020.5, viewOffsetX=-958.302,
    viewOffsetY=-571.248)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6835.63,
    farPlane=7857.28, width=1470.77, height=1022.23, viewOffsetX=-962.915,
    viewOffsetY=-1052.05)
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-879.468,
    viewOffsetY=-1040.13)
session.viewports['Viewport: 1'].view.setValues(cameraPosition=(570.191,
    2107.37, 7181.28), cameraUpVector=(-0.467155, 0.875414, -0.124164))
session.viewports['Viewport: 1'].view.setValues(viewOffsetX=-1117.89,
    viewOffsetY=-482.816)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6972.12,
    farPlane=7720.79, width=409.087, height=284.33, viewOffsetX=-1161.18,
    viewOffsetY=-641.462)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6975.5,
```

```
    farPlane=7717.41, width=409.286, height=284.468, viewOffsetX=-1161.74,
    viewOffsetY=-641.773)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6809.05,
    farPlane=7728.34, width=399.52, height=277.68, cameraPosition=(186.338,
    579.448, 7126.51), cameraUpVector=(-0.396858, 0.916625, 0.047986),
    viewOffsetX=-1134.02, viewOffsetY=-626.459)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6758.83,
    farPlane=7778.56, width=840.109, height=583.905, viewOffsetX=-1135.32,
    viewOffsetY=-538.255)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6752.39,
    farPlane=7785, width=839.308, height=583.349, viewOffsetX=-1134.24,
    viewOffsetY=-537.742)
session.viewports['Viewport: 1'].view.setValues(nearPlane=6044.47,
    farPlane=7981.9, width=751.315, height=522.191, cameraPosition=(2338.96,
    -2884.95, 5769.92), cameraUpVector=(-0.433826, 0.68925, 0.580284),
    viewOffsetX=-1015.33, viewOffsetY=-481.365)
session.viewports['Viewport: 1'].view.setValues(nearPlane=5733.63,
    farPlane=8292.75, width=3388.57, height=2355.18, viewOffsetX=-869.583,
    viewOffsetY=-402.388)
a = mdb.models['deviation'].rootAssembly
n1 = a.instances['PART-1-1'].nodes
nodes1 = n1.getSequenceFromMask(mask=(
    '[#ffffffff:39 #3fffff #0:580 #1c000001 #ff000000 #1 #fe000000',
    ' #ffffffff:8 #1ffff #3fc000 #f0000000 #1fff #0:20 #f0000000',
    ' #ffffffff:34 #ffffff #1c00000 #0:11 #700000 #10000c0 #0:29',
    ' #7ffff #fffffe #0:2 #fffff800 #ffffffff #fff ]', ), )
region = regionToolset.Region(nodes=nodes1)
mdb.models['deviation'].EncastreBC(name='BC-1', createStepName='Step-1',
    region=region)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=OFF, bcs=OFF,
    predefinedFields=OFF, connectors=OFF)
mdb.Job(name='couple', model='deviation', type=ANALYSIS,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, description='',
    parallelizationMethodExplicit=DOMAIN, multiprocessingMode=DEFAULT,
    numDomains=1, userSubroutine='', numCpus=1, preMemory=256.0,
    standardMemory=256.0, standardMemoryPolicy=MODERATE, scratch='',
    echoPrint=OFF, modelPrint=OFF, contactPrint=OFF, historyPrint=OFF)
mdb.jobs['couple'].writeInput(consistencyChecking=OFF)
#: The job input file has been written to "couple.inp".
session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=OFF,
    engineeringFeatures=OFF)
p = mdb.models['deviation'].parts['PART-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
mdb.save()
#: The model database has been saved to "arw.cae".
```

APPENDIX A.2

```
file
read-case
arw_steady.cas
q
define
bc
pff
farfield
no
0
no
0.8
no
216
no
0
no
0.999390827
no
0.034899496
q
q
solve
monitors
force
dc
yes
5
6
7
8
no
yes
no
no
0
0.999390827
0.034899496
lc
yes
no
yes
no
no
0
0.034899496
0.999390827
q
q
q
file
write-case
arw_steady2.cas
q
exit
```

APPENDIX A.3

Table A.1 : Material Properties of ARW-2

	Young Modulus (MPa)	Mass Density (kg/m^3)	Poisson's Ratio
Skin	31000	2700	0.35087
Axial Bars	40000	2700	0.4
Spars	39250	3600	0.23022
Ribs	60000	3600	0.39998

Table A.2 : Geometrical Properties of ARW-2

	Thickness (mm)	Radius (mm)
Rib 1	9.8	
Rib 2	26.0	
Rib 3	4.0	
Rib 4	20.0	
Rib 5	25.0	
Rib 6	30.4	
Rib 7	9.2	
Rib 8	2.4	
Rib 9	5.8	
Rib 10	1.2	
Rib 11	1.4	
Rib 12	6.0	
Rib 13	19.6	
Rib 14	7.8	
Rib 15	2.2	
Rib 16	40.4	
Rib 17	41.2	
Spar 1	1.0	
Spar 2	4.0	
Spar 3	4.0	
Spar 4	2.0	
Spar 5	5.2	
Axial Bar 1	17.1	3.1
Axial Bar 2	13.8	1.0
Axial Bar 3	13.0	1.0
Axial Bar 4	5.8	1.0

CURRICULUM VITA

Candidate's full name: Ahmet AYSAN

Place and date of birth: GÖLCÜK – 09.11.1984

Permanent Address: Gülbağ Mah. Avni Dilligil Sok. No : 38/4 Mecidiyeköy-İSTANBUL

Universities and
Colleges attended: Istanbul Technical University

Publications:

- Nikbay, M., Öncü, L., Aysan, A., A multi-disciplinary code coupling approach for analysis and optimization of aeroelastic systems. In 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 10 – 12 Sep 2008, Victoria, British Columbia, Canada, AIAA, 2008.