

# A CFD/CSD INTERACTION METHODOLOGY FOR AIRCRAFT WINGS

By  
Manoj K. Bhardwaj

A DISSERTATION SUBMITTED TO THE FACULTY OF  
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY  
IN  
AEROSPACE ENGINEERING

---

Rakesh K. Kapania, Chairman

---

Eric R. Johnson

---

William H. Mason

---

Bernard Grossman

---

Liviu Librescu

October 1997  
Blacksburg, Virginia

# A CFD/CSD INTERACTION METHODOLOGY FOR AIRCRAFT WINGS

by

Manoj K. Bhardwaj

Committee Chairman: Rakesh K. Kapania

Aerospace Engineering

(ABSTRACT)

With advanced subsonic transports and military aircraft operating in the transonic regime, it is becoming important to determine the effects of the coupling between aerodynamic loads and elastic forces. Since aeroelastic effects can contribute significantly to the design of these aircraft, there is a strong need in the aerospace industry to predict these aero-structure interactions computationally.

To perform static aeroelastic analysis in the transonic regime, high fidelity computational fluid dynamics (CFD) analysis tools must be used in conjunction with high fidelity computational structural dynamics (CSD) analysis tools due to the nonlinear behavior of the aerodynamics in the transonic regime. There is also a need to be able to use a wide variety of CFD and CSD tools to predict these aeroelastic effects in the transonic regime. Because source codes are not always available, it is necessary to couple the CFD and CSD codes without alteration of the source codes. In this study, an aeroelastic coupling procedure is developed which will perform static aeroelastic analysis using any CFD and CSD code with little code integration. The aeroelastic coupling procedure is demonstrated on an F/A-18 Stabilator using NASTD (an in-house McDonnell Douglas CFD code) and NASTRAN. In addition, the Aeroelastic Research Wing (ARW-2) is used for demonstration of the aeroelastic coupling procedure by using ENSAERO (NASA Ames Research Center CFD code) and a finite element wing-box code (developed as a part of this research). The results obtained from the present study are compared with those available from an experimental study conducted at NASA Langley Research Center and a study conducted at NASA Ames Research Center using ENSAERO and modal superposition. The results compare

well with experimental data.

Parallel computing power is used to investigate parallel static aeroelastic analysis because obtaining an aeroelastic solution using CFD/CSD methods is computationally intensive. A parallel finite element wing-box code is developed and coupled with an existing parallel Euler code to perform static aeroelastic analysis. A typical wing-body configuration is used to investigate the applicability of parallel computing to this analysis. Performance of the parallel aeroelastic analysis is shown to be poor; however with advances being made in the arena of parallel computing, there is definitely a need to continue research in this area.

# Acknowledgements

A variety of people have contributed to the work in this study. Mr. Vic Spain (Lockheed Martin) showed great patience in helping me in my understanding of the material. The help of Dr. Guru Guruswamy from NASA Ames Research Center and Dr. Chansup Byun (MCAT Institute) was invaluable. My friend, Mehrdad Farhangnia, gave great support at the time most needed. In addition, Mr. Rudy Yurkovich and Mr. Eric Reichenbach from McDonnell Douglas Aerospace-East contributed significantly to the direction of this research. I would also like to thank Drs. Grossman, Mason, Librescu, and Johnson for their technical guidance. And finally, I would like to thank my advisor and committee chairman, Dr. Kapania. His guidance helped me in realizing my goals. Without him, none of this would have been possible.

# Dedications

I have been very fortunate to have some of the most loving, passionate, and supporting people as my family. My sister, Minni Powell, and my brother, Jimmy Powell, smiled at every accomplishment, encouraged me at every turn, and supported me in every decision. Without them, I'd have been studying something else, somewhere else. My uncle, Raghu Bhardwaj, and my aunt, Sushma Bhardwaj have also been supportive in my goals. They too deserve my thanks. My cousins, Alok and Anuj, have been very interesting to talk to. They have been great sources of amusement and laughter. Urvashi, my wife, has given meaning to my life. With her love and heavenly smile, I now know why I have worked so hard. And finally the two most important people in my life, my father, Brahm Tej Bhardwaj, and my mother, Chander Kanta Bhardwaj, deserve more than just the gratitude I give them. Thanking them is just the beginning. They have worked and worked so I can have a better life. They didn't let me get lazy in achieving minor milestones. They have supported, loved, and encouraged me to bring out my best; it worked. I hope to be able to return to them what they have given me. My parents are the base of my heart and soul. I am truly lucky.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>Dedications</b>	<b>v</b>
<b>1 Literature Review</b>	<b>1</b>
1.1 Aeroelasticity . . . . .	1
1.2 Previous CFD/CSD work . . . . .	4
1.3 Interface Mappings . . . . .	7
1.3.1 Finite-Plate Spline . . . . .	8
1.3.2 Multiquadric-Biharmonic . . . . .	9
1.3.3 Thin-Plate Spline . . . . .	9
1.3.4 Inverse Isoparametric Mapping . . . . .	9
1.3.5 Non-Uniform B-Splines . . . . .	10
1.3.6 Infinite-Plate Spline . . . . .	10
1.4 Convergence of the Aeroelastic Solution . . . . .	11
1.4.1 Exterior Grid Deformation . . . . .	12
1.5 Parallel Computing . . . . .	13
<b>2 Aeroelastic Coupling Procedure</b>	<b>16</b>
<b>3 Analysis of Aircraft Wings</b>	<b>29</b>
3.1 F/A-18 Stabilator . . . . .	29

3.1.1	CFD and CSD Modeling . . . . .	29
3.1.2	Aeroelastic Coupling Procedure . . . . .	30
3.2	Aeroelastic Research Wing (ARW-2) . . . . .	32
3.2.1	CFD and CSD Modeling . . . . .	32
3.2.2	Aeroelastic Coupling Procedure . . . . .	33
<b>4</b>	<b>Results</b>	<b>46</b>
4.1	F/A-18 Stabilator . . . . .	46
4.2	Aeroelastic Research Wing (ARW-2) . . . . .	47
4.2.1	Validation of the ARW-2 Wing Finite Element Model . . . . .	48
4.2.2	Rigid Steady State Solution . . . . .	48
<b>5</b>	<b>Parallel Aeroelastic Analysis</b>	<b>80</b>
5.1	Governing Aerodynamic Equations . . . . .	80
5.2	Aeroelastic Equations of Motion . . . . .	81
5.2.1	Wing-box Model . . . . .	81
5.3	Parallelization of the Aeroelastic Equations . . . . .	82
5.4	Structural Analysis . . . . .	83
5.4.1	A Square Panel . . . . .	84
5.4.2	A Cantilevered Beam . . . . .	84
5.4.3	Box Beam . . . . .	84
5.5	Aeroelastic Analysis . . . . .	85
5.5.1	Aerodynamic Modeling . . . . .	85
5.5.2	Structural Modeling . . . . .	85
5.5.3	Typical Wing-body Configuration . . . . .	86
5.6	Structural Analysis Results . . . . .	86
5.6.1	A Square Panel . . . . .	86
5.6.2	Cantilevered Beam . . . . .	87
5.6.3	Box Beam . . . . .	88
5.7	Aeroelastic Analysis Results . . . . .	88
5.7.1	Typical Wing-body Configuration . . . . .	88
5.8	Conclusions . . . . .	90

<b>6 Conclusions</b>	<b>113</b>
<b>Bibliography</b>	<b>115</b>
<b>Appendices</b>	<b>122</b>
<b>A Finite Element Wing-box Source Code</b>	<b>122</b>
<b>B Aeroelastic Coupling Procedure Source Code</b>	<b>188</b>
<b>Vita</b>	<b>198</b>

# List of Figures

2.1	Diagram of 2-D Airfoil . . . . .	25
2.2	Convergence of Solution for 2-D Airfoil . . . . .	26
2.3	Mapping of a CFD Grid Point to a CSD Triangle . . . . .	27
2.4	Area Coordinates of a CFD Grid Point within a CSD Triangle . . . . .	28
3.1	CFD Grid for the F/A-18 Stabilator . . . . .	35
3.2	Finite Element Model of the F/A-18 Stabilator . . . . .	36
3.3	Mapping of CFD Points to Structural Triangles for the F/A-18 Stabilator	37
3.4	Spline Points Used for Mapping for F/A-18 Stabilator . . . . .	38
3.5	CFD Surface Grid of the F/A-18 Stabilator . . . . .	39
3.6	Cosine Spacing Function Used to Deform Exterior Grid . . . . .	40
3.7	$k = \text{Constant}$ Face of the CFD Grid of the F/A-18 Stabilator . . . . .	41
3.8	CFD Grid of the ARW-2 Wing . . . . .	42
3.9	Finite Element Model of the ARW-2 Wing . . . . .	43
3.10	Mapping of Structural Triangles to CFD Points for the ARW-2 Wing	44
3.11	Spline Points Used in Aeroelastic Coupling of ARW-2 Wing . . . . .	45
4.1	Convergence of the Wing Tip of the F/A-18 Stabilator . . . . .	51
4.2	Convergence of the Trailing Edge Tip of the F/A-18 Stabilator . . . . .	52
4.3	Final Converged and Initial Undeformed F/A-18 Stabilator . . . . .	53
4.4	$C_p$ Variation on the Upper Surface of the Rigid F/A-18 Stabilator . . . . .	54
4.5	$C_p$ Variation on the Upper Surface of the Flexible F/A-18 Stabilator . . . . .	55
4.6	Mach Number Variation on the Upper Surface of the Rigid F/A-18 Stabilator . . . . .	56

4.7	Mach Number Variation on the Upper Surface of the Flexible F/A-18 Stabilator . . . . .	57
4.8	Displacement of the Front Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a 100 lb Vertical Load Applied at the Tip . . . . .	58
4.9	Displacement of the Rear Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a 100 lb Vertical Load Applied at the Tip . . . . .	59
4.10	Displacement of the Auxiliary Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a 100 lb Vertical Load Applied at the Tip . . . . .	60
4.11	Displacement of the Front Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a Twisting Load Applied at the Tip . . . . .	61
4.12	Displacement of the Rear Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a Twisting Load Applied at the Tip . . . . .	62
4.13	Twisting of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a Twisting Load Applied at the Tip . . . . .	63
4.14	$L_2$ Norm of the Residual of the Navier-Stokes Equations for the Rigid Steady State Solution at $\alpha = 1$ deg . . . . .	64
4.15	$L_2$ Norm of the Residual of the Navier-Stokes Equations for the Rigid Steady State Solution at $\alpha = 2$ deg . . . . .	65
4.16	Comparison of $C_p$ Variation for Rigid Steady State Solution at the 70.7% Semi-span Location for $\alpha = 1$ deg . . . . .	66
4.17	$L_2$ Norm of the Residual of the Navier-Stokes Equations for the Flexible Steady State Solution at $\alpha = 1$ deg . . . . .	67
4.18	$L_2$ Norm of the Residual of the Navier-Stokes Equations for the Flexible Steady State Solution at $\alpha = 2$ deg . . . . .	68
4.19	Comparison of $C_p$ Variation of Experimental Data Versus Computational Results at the 70.7% Semi-span Location for $\alpha = 1$ deg . . . . .	69
4.20	Section Lift Coefficient Variation Along Span for $\alpha = 1$ deg . . . . .	70
4.21	Section Lift Coefficient Variation Along Span for $\alpha = 2$ deg . . . . .	71
4.22	$C_p$ Variation on the Upper Surface of the Rigid and Flexible ARW-2 Wing, at $\alpha = 1$ deg . . . . .	72

4.23	$C_p$ Variation on the Upper Surface of the Rigid and Flexible ARW-2 Wing, at $\alpha = 2$ deg . . . . .	73
4.24	$C_p$ Variation for $\alpha = 2$ deg at the 70.7% Semi-span Location . . . . .	74
4.25	Comparison of the Experimental and Computational Front Spar Deflections of the ARW-2 Wing at $\alpha = 1$ deg . . . . .	75
4.26	Comparison of the Experimental and Computational Rear Spar Deflections of the ARW-2 Wing at $\alpha = 1$ deg . . . . .	76
4.27	Comparison of the Experimental and Computational Front Spar Deflections of the ARW-2 Wing at $\alpha = 2$ deg . . . . .	77
4.28	Comparison of the Experimental and Computational Rear Spar Deflections of the ARW-2 Wing at $\alpha = 2$ deg . . . . .	78
4.29	Comparison of the Rear Spar Deflections Using Modal Analysis Versus Finite Element Analysis of the ARW-2 Wing at $\alpha = 1$ deg . . . . .	79
5.1	Allman's Triangular Element . . . . .	91
5.2	Schematic of the Solution Procedure and the Coupling between Fluid and Structure Domains . . . . .	92
5.3	Uni-Partitioning Scheme of the Fluid Domain . . . . .	93
5.4	Discretization of a Square Panel with Linearly Varying Edge Normal Stress . . . . .	94
5.5	Discretization of a Cantilever Beam with Tip Load . . . . .	95
5.6	A Box Beam Subjected to Axial Loads . . . . .	96
5.7	Aerodynamic Surface Grid for a Typical Wing-body Configuration . . . . .	97
5.8	Aerodynamic Surface Grid of Wing . . . . .	98
5.9	Top View of the Structural Discretization of the Wing . . . . .	99
5.10	Structural Modeling of the Entire Wing . . . . .	100
5.11	Structural Discretization of the Spars and Ribs of the Wing . . . . .	101
5.12	Stress Contours Due to a Tip Load on the Cantilevered Beam . . . . .	102
5.13	Wing Leading Edge Tip History Versus Iteration Step . . . . .	103
5.14	$L_2$ Norm of the Residual of the Energy Equation . . . . .	104
5.15	Comparison of the Stresses in the Axial Bars of Box Beam . . . . .	105

5.16 $C_p$ Variation on the Upper Surface of a Rigid Wing . . . . .	106
5.17 $C_p$ Variation on the Upper Surface of a Flexible Wing . . . . .	107
5.18 Initial Undeformed and Final Converged Wing-box Tip Section . . . .	108
5.19 Initial Undeformed and Final Converged Wing-box Leading Edge (Front Spar) . . . . .	109
5.20 Bending Stress Variation on the Upper Surface of a Flexible Wing (psi)	110
5.21 CPU Time for the Execution of the Various Parts of the Finite Element Wing-box Code . . . . .	111

# List of Tables

5.1 Comparison of Allman's Triangular Element, Constant Strain Triangle,  
and Linear Strain Triangle for Analysis of Cantilevered Beam . . . . 112

# Chapter 1

## Literature Review

### 1.1 Aeroelasticity

Aeroelasticity is defined as *phenomena which exhibit appreciable reciprocal interaction (static or dynamic) between aerodynamic forces and the deformations induced thereby in the structure of a flying vehicle, its control mechanisms, or its propulsion system* [1]. Aeroelastic problems would not exist if airplanes were perfectly rigid [2]. The primary focus of this research is static aeroelasticity, i.e. the interactions between elastic and aerodynamic forces. Control system reversal, control surface effectiveness, divergence, and load distribution are some of the areas in which static aeroelasticity plays an important role.

Traditionally, aircraft designers have viewed aeroelastic effects as undesirable. To avoid aeroelastic phenomena, the flexibility of the wing was decreased, but this added weight to the structure. Recently, there has been an increased interest in taking advantage of these aeroelastic effects for roll control, load alleviation, and drag reduction while reducing the wing weight as seen in the Active Flexible Wing (AFW) [3,4] and the Active Aeroelastic Wing (AAW) [5] programs.

In the AFW program, large amounts of aeroelastic twist in fighter aircraft type wings are permitted to increase maneuverability. Roll performance is degraded as a

direct result in the form of aileron reversal over a large portion of the flight envelope. The problem is alleviated using multiple leading and trailing edge wing control surfaces in various combinations. The AAW program also examines multiple control surface blending in increasing roll performance, especially in the transonic regime. Miller [6] predicted that savings of at least 15 percent of take-off gross weight could be achieved for an advanced fighter configuration by taking advantage of fluid-structure interactions.

In addition, the accurate prediction of wind tunnel model static aeroelastic deformations is becoming increasingly important for transonic testing of transport aircraft [7]. A computational fluid dynamics (CFD) code in conjunction with a computational structural dynamics (CSD) code is used to predict the aeroelastic deformations under given flight conditions for a 1-g wing. The deformations are “subtracted” from the original 1-g wing to obtain the “jig” wing model. The jig wing will deform to the designed 1-g wing under the given flight conditions. Experimental data are obtained in wind tunnel tests using the jig wing. Thus, the correlation between experimental and computational data is improved since aeroelastic deformations are taken into account.

Whether viewed as undesirable or desirable, it is becoming more important to predict static aeroelastic behavior in the transonic regime of transport and fighter aircraft. Advanced CFD tools are necessary to capture the nonlinear behavior of the aerodynamics in the transonic regime (shocks, vortices, separation). In transonic flow, the nonlinear nature of the aerodynamics makes load prediction difficult. The loads an airfoil experiences are dependent of the accurate prediction of the shock waves [8]. Numerous studies have also shown the need for using advanced CSD analysis tools, Ref. [7,9–15], for obtaining the structural response of the aircraft in aeroelastic solutions. Hooker *et al.* [7] stated the need for a complete definition of all the wind tunnel model cut-outs when performing static aeroelastic analyses which significantly improved the correlation between CFD predicted and wind tunnel measured wing surface pressures. The finite element method (FEM), which is fundamentally based on discretization, has proven to be computationally efficient to solve aerospace structures problems [13].

The coupling of high fidelity CFD and CSD tools to solve aeroelastic problems has received interest only in the past few years. Large amounts of computational power is required to make the use of such tools feasible. However, continuous improvements in computer speed, memory, and architecture have made solving these computationally intensive problems more cost effective.

Both uncoupled and coupled methods for solving these nonlinear systems of equations [16] exist. Aeroelastic problems of aerospace vehicles are often dominated by flow nonlinearities and at times by large structural deformations. Therefore, coupled approaches are necessary to solve such problems accurately [9].

Coupled approaches for solving aeroelastic problems are usually categorized in two ways: fully or loosely coupled. The loosely coupled approaches can be integrated or modular. Integrated, loosely coupled methods alter the source code of either the CSD or CFD analysis tool by including the coupling schemes in either code. Though the codes are integrated, the CFD and CSD equations are not being altered and are solved as one system, but remain loosely coupled. Modular, loosely coupled methods do not integrate the coupling schemes into either the CFD or CSD code. This allows the use of a variety of CFD/CSD codes.

Fully or strongly (single domain) coupled approaches require the solution of the CFD and CSD equations simultaneously which necessitates the reformulation of the equations of each discipline [17]. The numerical matrices associated with the structures are orders of magnitude stiffer than those associated with fluids. Thus, it is numerically inefficient or even impossible to solve both systems using a monolithic numerical scheme [9]. Recently, there have been renewed attempts to solve both fluids and structures in a single computational domain [18, 19]. However, they have been limited to simple two-dimensional problems and have not proven to be better than the loosely coupled approach. The drop in convergence rate from the rigid case to the flexible case in Ref. [19] indicates another weakness of a fully coupled approach.

Guruswamy and Yang [16] demonstrated a loosely coupled approach to aeroelasticity. The fluids and structures were modeled independently and exchanged boundary information to obtain aeroelastic solutions. The fluids were modeled using finite-difference based transonic small perturbation (TSP) equations. The structures were

modeled using finite element equations. The two disciplines were used to solve aeroelastic problems of two-dimensional airfoils. This loosely coupled or domain decomposition approach was shown to be efficient and accurate. This approach has been extended to three-dimensional problems and is incorporated into advanced aeroelastic codes as XTRAN3S [20], ATRAN3S [21], and CAP-TSD [22]. Guruswamy [23, 24] also demonstrated the same technique by modeling fluids with Euler/Navier-Stokes equations on moving grids. Matching the CFD grid displacements with the CSD or finite element model response maintains the accuracy of this loosely coupled approach.

The CFD and CSD codes are usually integrated using this loosely coupled approach. The two disciplines exchange information at the boundaries in an efficient manner since the codes have been tightly integrated. Several papers have presented techniques for calculating aeroelastic solutions using this approach.

## 1.2 Previous CFD/CSD work

Chipman *et al.* [25] obtained transonic loads on a flexible supercritical transport wing. The transonic aerodynamics were modeled using GAC/AMES [26] which uses a finite-difference formulation of the modified small perturbation equation. The wing is structurally discretized using a beam model. The fluid and structure models are coupled loosely. The solution is obtained in an iterative scheme using underrelaxation. However, shock location and separation were not predicted accurately. More accurate aerodynamic modeling is required to obtain the transonic loads.

Batina *et al.* [27, 28] obtained transonic aeroelastic solutions coupling an unstructured grid Euler method with modal structures. The CFD and CSD equations were loosely coupled, but were tightly integrated into a CFD code, CFL3D. Mode shapes were used to obtain the structural response which reduces the number of equations to be solved. The coupling of the CFD and CSD equations is simpler since the mode shapes can be interpolated to the CFD grid. Hence, the forces are obtained at CFD grid points on the wing and do not need to be mapped to the CSD nodes. The structural system of equations is solved to obtain the generalized coordinates at the

CFD grid points. The disadvantages of using modal analysis [29] (mode superposition method) is the assumption that the wing deformations can be described by a linear combination of a finite set of modes. The increasing use of composite materials for aeroelastic tailoring and highly sensitive nature of transonic flows makes the linearization assumption inherent in modal techniques less attractive [11]. And if the number of modes needs to be increased, then a separate modal identification technique has to be solved. Conversely, there is no assumption of harmonic motion when using finite element equations and detailed stresses may be obtained directly from the solution. The storage and CPU time required are increased, but more reliable results are obtained.

Ref. [27, 28] use unstructured grid technology to obtain the CFD solution. The time required to obtain steady-state solutions on unstructured grids is two to five times more expensive than using structured grids with the same number of cells [30], therefore the CFD codes used in this research obtain the transonic flowfield on structured grids.

Purcell *et al.* [31] presented a loosely coupled approach to solving aeroelastic problems in the transonic regime. Any CFD analysis tool can be coupled with a particular CSD analysis tool, ELFINI [32], to calculate aeroelastic solutions. The algorithms which map the displacements and loads between the CFD and CSD codes are integrated into ELFINI which restricts the use of a variety of CSD codes.

Aerodynamicists have been forced to utilize vortex flow to enhance fighter wing performance in the transonic regime, leading to localized high loads since only small portions of the wing are generating the overall lift. To aid in identifying these areas, Schuster *et al.* [33] obtained static aeroelastic solutions of fighter aircraft by coupling Navier-Stokes flow equations with finite element equations. Finite element equations were used since they tend to give more reliable results than modal analysis. Details of the mappings required to interpolate displacements and loads were not given in Ref. [33]. The CFD and CSD codes were loosely coupled and integrated in a single code, restricting the use of a wide variety of CFD and CSD codes.

Other similar work [7, 13, 14, 34–36] has also attacked the problem of aeroelasticity by using loosely coupled high fidelity CFD and CSD methods. Often the coupling

is integrated, not allowing a wide variety of CFD and CSD codes to be used. The CSD analysis, in some of this work, is performed using a modal analysis approach; this makes the exchange of boundary information easier. The loads need only to be calculated on the CFD grid points. As a direct result, not many algorithms have been presented for accurate transformation of pressures on the CFD grid to loads on the CSD nodes. Future work, however, requires the use of detailed finite element models and the use of direct finite element equations, not modal analyses. Therefore, an accurate load transformation scheme is needed.

Macmurdy *et al.* [11] obtained a static aeroelastic solution on an intermediate complexity wing (ICW) using Euler flow equations (ENSAERO) coupled with finite element equations. The finite element wing-box was modeled using an Air Force in-house structural analysis code, ANALYZE [37]. Static aeroelastic solutions were obtained by loosely coupling ENSAERO with ANALYZE in a modular manner. This modularity allows a variety of CFD/CSD codes to be used. The twist and leading edge plunge are obtained from the structural response which is then applied to the CFD grid. The loads are calculated at the CFD grid points and are transferred to the CSD nodes using various schemes. The schemes do not transfer the loads accurately since some of the information is extrapolated. The calculated stresses on the wing were not accurate since constant strain triangles (CST) were used to model the wing skin and due to the inaccurate load transfer schemes.

Tzong *et al.* [10] presented a general method for calculating aero-structure interactions. An interface method based on finite element technology was used to exchange information between the CFD and CSD codes. The CFD analysis was performed using OVERFLOW [38] and a Douglas panel code [39]. The CSD response was calculated using a McDonnell Douglas Corporation finite element code. The interface method maps each CFD grid point to a host finite element. The displacements and loads are transferred between the CFD grid point and the CSD nodes using the shape functions of the host finite element. The disadvantage of this approach is that the shape functions of the finite elements in the model might not be available to the user. In addition, the necessary degrees of freedom might not be contained in the host finite element to transfer the boundary information accurately. This interface method

has been integrated into the finite element code at Douglas. This again restricts the user's ability to use a wide variety of CFD and CSD codes.

Two ways of transferring the pressures on the CFD grid to the CSD nodes are possible [10]. In the first transfer method, the pressures on the CFD grid are interpolated onto the CSD model and are integrated to obtain the forces on the CSD nodes. Ref. [10] stated that the inconsistency between the CFD and CSD models makes this conversion improper. The pressures can be converted to the CSD model, but the loads may not be integrated accurately since information about the true surface areas is often not available from the CSD model. In the second method, the forces at the CFD grid points are calculated by using the CFD grid information and then are transferred to the CSD nodes. This transfer calculates loads on the CSD nodes more accurately and is easier to implement. This is the method used in this research.

### 1.3 Interface Mappings

In this research, static aeroelastic solutions are calculated using a loosely coupled and modular approach. This allows a wide variety of CFD and CSD codes to be used in calculating static aeroelastic solutions and does not require the source codes. The one disadvantage is that the process is not as efficient as an integrated approach.

In the loosely coupled modular approach, boundary information between the CFD and CSD codes is exchanged through the codes' native files. Native files are the files required by the code as input and the files to which the output is written. The forces are obtained from the output of pressures from the CFD code. A pressure mapping algorithm transfers the pressures from the CFD grid to the CSD nodes. The CSD code calculates the response of the structure. The resulting output, the displacements, are interpolated to the CFD grid using a displacement mapping algorithm. The CFD code calculates the flow field about this new CFD grid. The procedure is repeated in an iterative manner until a specified convergence criterion is met.

Two mappings are necessary to obtain static aeroelastic solutions in a loosely coupled and modular manner. The pressures on the wing CFD grid have to be transferred to forces on the CSD nodes and the displacements on the CSD nodes

have to be interpolated to the CFD grid points on the wing. In this research, the forces at the CFD grid point are obtained and then transferred to the CSD nodes. The mapping used is described in chapter /refchapter:mapping.

The mapping of the displacements from the CSD nodes to the CFD grid requires an interpolation scheme. Smith *et al.* [40] presented a review of the methodologies used to do this mapping in interfacing CFD/CSD codes. A significant literature review and an industry/government survey narrowed the search to six schemes: (i) the Infinite-plate spline; (ii) Finite-plate spline; (iii) Multiquadric-Biharmonic; (iv) Thin-Plate Spline; (v) Inverse Isoparametric Mapping; and (vi) Non-Uniform B-Splines; These methods were analyzed by a series of mathematical test cases and selected applications. The six schemes were rated based on accuracy, smoothness, diminishing variation, robustness, extrapolation, CPU memory, and CPU time. Next, brief descriptions of the methods taken from Ref. [40] are stated.

### 1.3.1 Finite-Plate Spline

The finite-plate spline (FPS), originally developed by Appa [41], employs uniform plate elements to represent a given planform. The virtual surface created by these plate elements is constrained to pass through both the structural and aerodynamic grid points. The constraints are applied at the element level, and a proper choice of shape functions are necessary. The shape functions of the elements relate the displacements at the CSD nodes to the CFD grid points. The structural nodes of the virtual mesh do not have to coincide with either the CFD or CSD grid, but are usually a subset of the CSD grid. The FPS has the advantage of accommodating changes in fluid and structural models easily. The approach conserves the work done by the aerodynamic forces when obtaining the global node force vector. It is versatile enough to model realistic body geometries because it is finite element based. The known disadvantages of the FPS are few since the method is new. Using this method, a mapping matrix,  $3m \times 3n$ , is generated, where  $m$  is the number of aerodynamic grid points and  $n$  is the number of structural nodes. Therefore, large amounts of CPU

time and memory are required to generate and store this matrix. This method is recommended for mainframes or supercomputers with large memory cores. Guruswamy *et. al* [13,14] have used the finite-plate spline with extensions to 3-D, but no other details were found in literature.

### 1.3.2 Multiquadric-Biharmonic

The multiquadric-biharmonic (MQ) scheme was developed to perform interpolation of various topographies. The scheme is used to represent a surface using quadratic basis functions. The system of equations is biharmonic because they can always be solved. The MQ method is stable and consistent with respect to a user-defined parameter which controls the shape of the basis functions. The method produces an infinitely differentiable function that preserves monotonicity and convexity. This method has not been applied to aeroelastic computations.

### 1.3.3 Thin-Plate Spline

The thin-plate spline (TPS) method allows a representation of an irregular surface by using functions which minimize an energy functional. This method is similar to the MQ method, but the problem is approached from an engineering or physical representation of the surface. This method can be applied to 1-D, 2-D, or 3-D problems by varying the functional. The 2-D method minimizes bending energy of a thin-plate which is exactly the same as the infinite-plate spline method. The TPS has been used in aeroelastic applications, but has been limited to the 2-D method as an infinite-plate spline.

### 1.3.4 Inverse Isoparametric Mapping

The inverse isoparametric mapping (IIM) method is based on finite element analysis where an isoparametric element uses the same shape functions to interpolate both coordinate and displacement vectors. This method is limited in that it is unable to extrapolate data. The method has been used in aeroelastic applications as seen in

Refs. [42, 43].

### 1.3.5 Non-Uniform B-Splines

Splines in their simplest form are used to represent curves in 2-D space. Spline functions can be polynomial or rational in type. Polynomial splines are piecewise polynomial functions of a specified degree. The B-splines (basic splines), a subclass of polynomial splines, are linearly independent and span the space of univariate polynomial splines. Any polynomial spline function can be represented as a series B-splines. Only polynomial cubic splines were investigated in Ref. [40]. B-splines are not currently used in aeroelastic applications.

### 1.3.6 Infinite-Plate Spline

The method of infinite-plate spline (IPS) is used extensively in programs such as ASTROS, MSC/NASTRAN, XTRAN3S, ENS3DAE, and CLF3DAE. The method is based on a superposition of the solutions of the partial differential equation of equilibrium for an infinite plate. Using solutions of the equilibrium equation, a concentrated set of loads are calculated that give rise to the deflections at the data points. The concentrated forces are substituted back into the solution providing a smooth surface that passes through the data. The deflections at the CFD grid points are easily calculated using the deflections at the CSD nodes.

Of the presented methods, the IPS method was chosen to interpolate displacements from the CSD nodes to the CFD grid in this study. The IPS method is referred to as the Harder and Desmarais surface spline [44]. The IPS method provides reasonable results without having the requirement that the input grid be a rectangular array. In addition, its ease of use and implementation make it one of the better methods as can be seen by its use in several codes. Details of this method are given in chapter 2. More details of the other methods discussed above can be found in the excellent analyses given in Ref. [40].

## 1.4 Convergence of the Aeroelastic Solution

In a loosely coupled approach, the static aeroelastic solution is obtained in the transonic regime in an iterative manner. The pressures on the CFD grid are mapped to forces on the CSD nodes. The structural system of equations is solved and the displacements are used to deform the CFD grid. The pressures on the deformed grid are obtained and the process repeated. This iterative technique obtains the solution for certain static aeroelastic cases. In this work, the structural system of equations is solved using direct methods. The CFD equations are solved using various techniques depending on the code. If the quality of the CFD grid is retained after deformation, the limitation on this approach will depend on the capability of the CFD code. If the deformations become too large, the CFD grid will lose its quality. The deformations will be large near divergence dynamic pressures. Therefore, this method is limited in that it will not predict divergence. However, the method will provide the static aeroelastic response, load distributions, under given conditions which a designer can use to optimize an aircraft design.

Due to the oscillatory nature of convergence of an aeroelastic solution, the time to obtain an aeroelastic solution can be reduced in various ways. This oscillatory convergence is due to the iterative scheme applied to obtain a static aeroelastic solution. The loads are calculated using the CFD solution on a rigid wing and applied to the CSD model. The swept back wing will bend and twist negatively. These deformations are applied to the CFD grid. The pressures are decreased since the angle of attack has decreased. The loads are again applied to the CSD model. Since the loads are less than the rigid loads, the wing twists less. These deformations are applied to the CFD grid again. The loads have increased since the angle of attack has increased since the last calculation. This is repeated until convergence which will be oscillatory as explained above.

Several researchers have investigated either artificial structural damping [45] or under-relaxation techniques (Ref. [10, 25]) to converge the solution faster and/or to keep it stable. In this research, an initial rigid steady state solution of the lifting

surface is used to decrease the time to calculate a static aeroelastic solution as opposed to starting impulsively from free stream boundary conditions. In addition, the CFD solution is not fully converged after each grid deformation before exchanging information with the structural analysis code for the case when NASTD is used. This has the same effect as an underrelaxation scheme and has been used effectively as seen in Ref. [15].

### 1.4.1 Exterior Grid Deformation

In the loosely coupled approach to obtain static aeroelastic solutions, the exterior CFD grid has to be deformed using the deflections on the wing surface. There are two ways of doing this: (i) regenerate a completely new CFD exterior grid or (ii) deform the existing CFD grid. In most research, the existing CFD grid is deformed. These methods redistribute points along grid lines that are in the radial direction normal to the surface. The grid points are distributed by moving them along these grid lines by displacing them a value equal to the surface value times some spacing parameter. Guruswamy [9] used a normalized arc length as the spacing parameter. Batina [27] represented the exterior grid using a spring network. The stiffness of the spring is inversely proportional to the length of the side of the CFD cell. This prevents the CFD grid from losing its quality. In this research, only vertical displacements are taken into account. Therefore, a simple cosine spacing function is used to deform the exterior grid, details of which are given in chapter 2.

Static aeroelastic solutions are obtained in this research assuming a linear structural model. The loads obtained from the pressures are applied to the original finite element model to obtain the displacements. The finite element is not regenerated using the displacements in the previous iteration although this capability is not difficult to include in the aeroelastic coupling procedure.

In this work, an aeroelastic coupling procedure is presented by which static aeroelastic solutions of aircraft wings are obtained. The aeroelastic coupling procedure requires only the grid point coordinates of the CFD and CSD grids to create the interface mappings. To demonstrate this procedure, a static aeroelastic solution of the

F/A-18 Stabilator is calculated by using Euler flow equations as available in NASTD (an in-house McDonnell Douglas Aerospace - East code) and finite element equations as available in the structural analysis tool NASTRAN [46]. The solution is obtained in the highly nonlinear transonic range at Mach 0.95, one degree angle of attack. Next, two different CFD and CSD codes are used to obtain a static aeroelastic solution for the Aeroelastic Research Wing (ARW-2). Navier-Stokes equations, as available in ENSAERO [47], are coupled with a finite element wing-box code to obtain a static aeroelastic solution in the transonic regime at Mach 0.85, at one and two degrees angle of attack. The flexible solutions are also compared with experimental results, and a good agreement is obtained. The examples use direct finite element equations, not modal analysis equations, to obtain the structural response. The advantage of the aeroelastic coupling procedure is shown by using two different sets of CFD/CSD codes to perform static aeroelastic analyses.

## 1.5 Parallel Computing

There is an increasing need to reduce turn around time between converged aeroelastic solutions due to the computational intensity of using high fidelity CFD and CSD methods to perform aeroelastic analyses. The most recent trends show that the performance of serial machines is saturating due to the physical limit imposed by the speed of light [48]. Therefore, the next step in high performance computing is the use of many processors in parallel to reduce computational times.

The speed of current microprocessors is one order of magnitude less than the speed of the fastest serial computers [48]; however, microprocessors cost much less. Using a network of these microprocessors, researchers can obtain raw computing power comparable to the fastest serial machines at a lower cost. Therefore serial codes can be “ported” to parallel machines and run faster.

However, in order to take advantage of parallel computing, a code must be parallelizable, i.e. the underlying physical problem should be amenable to being broken down efficiently among a group of processors. Using an analogy, if one person paints four walls in a room, the work could be done much faster using more people. So this

work is highly parallelizable. Alternatively, there might be only one can of paint, therefore this process might become highly inefficient if all the people are waiting idly to use the can of paint. In addition, the work must be broken down evenly among the workers, or some workers will wait idly while other workers finish the job. This brings forth an important point, the issues involved in parallel algorithms vary significantly from those for serial algorithms. In addition, the parallel computer hardware itself can greatly influence the development of parallel algorithms. Unfortunately, there are different models of parallel computers varying from shared memory architectures to distributed memory architectures. Also, the type of network used in the parallel machine can affect the performance of the code. So when developing parallel algorithms, there are many factors that need to be considered which normally do not come into play when developing sequential algorithms.

Significant advances have been made for single-discipline use of both CFD and CSD, with computations being made on complete aircraft. However, due to the lack of computational power, only a limited amount of work has been performed in coupling these two disciplines for multidisciplinary applications. As mentioned before, serial machines are reaching their physical limits, so there has been increased interest in parallel computing for aeroelastic analyses. Aeroelasticity, like other multidisciplinary applications, contains inherent parallelism which can be exploited using parallel computing. With respect to aeroelastic analysis, the ability to run CFD and CSD codes independently using some type of communication has great potential for parallel computing. And if propulsion is needed in the analysis, then coupling routines can be created, and a number of processors can be assigned to the propulsion code. One way of gaining advantage is to run the CSD code on one processor and the CFD code on another processor and have the two codes communicate directly with some code integration efforts. In addition, the CSD and CFD codes themselves can be parallel algorithms designed to run on many processors.

The current paradigm of doing aeroelastic analysis would seem inefficient for this application. In other words, while the CFD code obtains an intermediate solution (not converged), the CSD code sits idle, and vice-versa. However, work has been done performing parallel aeroelastic analysis whereby both disciplines start with an

initial guess and exchange information at rendezvous points and iteratively converge towards a flexible solution. The effects of varying the initial guess were not studied. The CSD code assumes a constant pressure field over the wing as an initial guess. The CFD code starts from the rigid steady state solution. In this manner, neither code will sit idle for a long time. However, if the process of obtaining a CSD solution is ten times faster than obtaining a CFD solution, then the processors devoted to the CSD code will remain idle. This will be inefficient. Thus load balancing would have to be implemented. For example, if both codes are parallel, and one code is ten times faster, then the slower code will have assigned to it, ten times more processors. This assumes that both codes are scalable, i.e. with increasing number of processors the time to obtain the solution decreases proportionally.

In this study, an existing parallel CFD code is coupled with a parallel CSD code, and parallel aeroelastic analysis capability is investigated. The Intel iPSC/860 hypercube, a multiple-instruction, multiple-data (MIMD) machine, is used to demonstrate the parallel aeroelastic analysis. However, there is no parallel CSD code available. So a major portion of this study was devoted to the creation of a parallel finite element wing-box code. A typical wing-body type configuration is used for demonstration purposes using Euler flow equations in the parallel version of ENSAERO.

The details of this research are given as follows. Chapter 2 describes the aeroelastic coupling procedure which is applied to the F/A-18 Stabilator and the Aeroelastic Research Wing (ARW-2). The details of the two lifting surfaces are outlined in chapter 3, followed by the results of the static aeroelastic analysis in chapter 4. Investigation of parallel computing as applied to aeroelastic analyses is described in chapter 5. Finally, the work is summarized in chapter 6.

## Chapter 2

# Aeroelastic Coupling Procedure

A general coupling procedure is presented by which static aeroelastic solutions of wings may be obtained using a wide variety of CFD/CSD codes. Many methods presented to do this require specific codes or assume the source codes are available for alteration. The aeroelastic coupling procedure uses two mappings to exchange information between CFD and CSD codes through the codes' native files; thus no code integration is required.

A static aeroelastic solution of a wing is obtained using the following aeroelastic coupling procedure:

- 1 Obtain an intermediate or rigid steady state CFD solution for the wing
- 2 Calculate the pressures at the CFD grid points on the aerodynamic surface
- 3 Map pressures at the CFD grid points to forces on the CSD nodes
- 4 Obtain the structural response of the wing
- 5 Map displacements at the CSD nodes to the displacements on the CFD grid points of the aerodynamic surface
- 6 Deform the entire CFD grid
- 7 Repeat steps 1-6 until preselected convergence criteria is met

The above steps are repeated in an iterative manner until a converged solution is obtained. This fixed-point iteration scheme is used for its simplicity and for its application to obtaining loosely coupled CFD/CSD solutions. To use a method which converges faster, like Newton's method [49], large amounts of computational time would have to be spent in calculating sensitivities of pressure with respect to deformations. Direct finite element analysis, not modal analysis, determines the structural response, thus the number of unknowns makes this process inefficient. Therefore Newton's method is computationally too expensive to make this approach feasible.

To help converge to the solution faster using this fixed-point iteration scheme, a rigid steady state solution is obtained before initiating the aeroelastic coupling procedure. An intermediate, not fully converged CFD solution was sometimes used to decrease the time to converge to a solution. This iterative scheme is demonstrated on a 2-D airfoil.

A simplified aeroelastic system [50] of an airfoil mounted to a wall by a torsional spring is used to demonstrate the fixed-point iteration scheme. The equation of equilibrium for the 2-D airfoil (Fig. 2.1) is

$$K\theta = Lec + M_o \quad (2.1)$$

where  $K$  is the stiffness of the rotational spring,  $L$  is the lift at the aerodynamic center,  $M_o$  is the moment about the aerodynamic center,  $e$  is the distance from the aerodynamic center to the elastic axis in percent chord, and  $c$  is the chord length of the airfoil. For simplicity, assume  $M_o = 0$ . The lift is calculated as

$$L = \frac{1}{2}\rho V^2 S \frac{\partial C_L}{\partial \alpha} (\alpha + \theta) \quad (2.2)$$

where  $\rho$  is the free stream density,  $V$  is the free stream velocity,  $S$  is the reference wing area,  $\frac{\partial C_L}{\partial \alpha}$  is the lift curve slope,  $\alpha$  is the rigid angle of attack, and  $\theta$  is the rotation of the airfoil due to its flexibility. Substituting  $M_o = 0$  and the expression for lift into Eqn. 2.1, the divergence velocity is calculated to be

$$V_d^2 = \frac{K}{Sec \frac{\partial C_L}{\partial \alpha}} \quad (2.3)$$

The divergence velocity  $V_d$  is the velocity at which  $\theta \rightarrow \infty$ . The iterative scheme in the aeroelastic coupling procedure is applied to this example to calculate the rotation  $\theta$  at a specified free stream velocity  $V$ .

An initial guess of  $\theta_0 = 0.1$  deg is used to calculate  $\theta_i$ . For specified values of  $V$  and  $\theta_i$ , the lift is calculated using Eqn. 2.2. The new rotation angle is then calculated by using Eqn. 2.1. Rewriting the equation,

$$\theta_{i+1} = \frac{\frac{1}{2}\rho V^2 Sec \frac{\partial C_L}{\partial \alpha}}{K} (\alpha + \theta_i) \quad (2.4)$$

Now let  $V^2 = \gamma V_d^2$ , Eqn. 2.4 becomes,

$$\theta_{i+1} = \gamma(\alpha + \theta_i) \quad (2.5)$$

Let  $g(\theta) = \gamma(\alpha + \theta)$ . This fixed-point iteration scheme will converge if  $|\frac{\partial g}{\partial \theta}| < 1$ . Taking the partial derivative of  $g$  with respect to  $\theta$ , get

$$\frac{\partial g}{\partial \theta} = \gamma \quad (2.6)$$

Thus, the fixed-point iteration scheme will converge if  $|\gamma| < 1$ . Physically, if the free stream velocity,  $V$ , is less than  $V_d$ , the solution will converge.

To validate the above analysis, various values of  $\gamma$  are chosen, and the solution  $\theta$  is calculated using the fixed-point scheme of Eqn. 2.5. As shown in Fig. 2.2, the fixed-point scheme works for all values of  $\gamma < 1$ . Note, the case where  $\gamma > 1$ ,  $\theta$  diverges quickly. If the free stream velocity is less than  $V_d$ , the fixed-point scheme will converge. Since swept back wings are not inhibited by divergence, but rather flutter, the iterative scheme will converge. The limitations of the aeroelastic coupling procedure are in the modeling of the deformations on the CFD grid. The iterative scheme will not be a limiting factor if the velocities close to divergence are avoided.

In obtaining the static aeroelastic solution of a wing, either a fully converged rigid steady state solution is obtained or an intermediate solution is obtained before initiating the aeroelastic coupling procedure. In this research, both methods were used. However, the aeroelastic solution converges faster if the aeroelastic coupling is started with the CFD rigid steady state solution as opposed to starting impulsively from free stream boundary conditions. This is shown in Sec. 5.7.1. The CFD solution

is calculated using any CFD code. Then, the pressures are calculated at the CFD grid points of the wing. The forces are calculated at each CFD grid point using the pressures and calculated areas. The forces at the CFD grid points of the wing are then mapped onto the CSD nodes. Therefore, given the pressures on the aerodynamic surface, the mapping will transform the pressures on the CFD grid to forces on the CSD or finite element model. To explain the coupling, the grid is assumed to be structured with indices  $i, j$  varying along the wing surface and the index,  $k$ , varying in the normal direction.

The area on which the pressure acts and the unit normal are calculated using the aerodynamic surface CFD grid. The load at the CFD grid point  $i, j$  is calculated as

$$\begin{Bmatrix} G_x^{i,j} \\ G_y^{i,j} \\ G_z^{i,j} \end{Bmatrix} = S^{i,j} p_a^{i,j} \begin{Bmatrix} n_x^{i,j} \\ n_y^{i,j} \\ n_z^{i,j} \end{Bmatrix} \quad (2.7)$$

where  $G_x^{i,j}$ ,  $G_y^{i,j}$ , and  $G_z^{i,j}$  are the forces in the  $x$ ,  $y$ , and  $z$  directions, respectively.  $S^{i,j}$  is the area on which the pressure,  $p_a^{i,j}$ , acts, and  $n_x^{i,j}$ ,  $n_y^{i,j}$ , and  $n_z^{i,j}$  are the  $x$ ,  $y$ , and  $z$  components of the unit normal for CFD point  $i, j$ .  $S^{i,j}$  and  $\{n^{i,j}\}$  are calculated by taking the cross products between adjacent CFD points.  $S^{i,j}$  is calculated using four neighboring points (Fig. 2.3), namely  $(i-1, j)$ ,  $(i+1, j)$ ,  $(i, j-1)$ , and  $(i, j+1)$ , as

$$S^{i,j} = \frac{1}{4} (|\vec{r}^{i-1,j} \times \vec{r}^{i,j+1}| + |\vec{r}^{i,j+1} \times \vec{r}^{i+1,j}| + |\vec{r}^{i+1,j} \times \vec{r}^{i,j-1}| + |\vec{r}^{i,j-1} \times \vec{r}^{i,j+1}|) \quad (2.8)$$

where  $\vec{r}^{a,b}$  is the distance between CFD grid points  $i, j$  and  $a, b$ . The unit normal,  $\{n^{i,j}\}$ , for CFD point  $i, j$  is

$$\begin{Bmatrix} n_x^{i,j} \\ n_y^{i,j} \\ n_z^{i,j} \end{Bmatrix} = \left( \frac{1}{4} \right) \frac{(\vec{r}^{i-1,j} \times \vec{r}^{i,j+1}) + (\vec{r}^{i,j+1} \times \vec{r}^{i+1,j}) + (\vec{r}^{i+1,j} \times \vec{r}^{i,j-1}) + (\vec{r}^{i,j-1} \times \vec{r}^{i,j+1})}{(|\vec{r}^{i-1,j} \times \vec{r}^{i,j+1}| + |\vec{r}^{i,j+1} \times \vec{r}^{i+1,j}| + |\vec{r}^{i+1,j} \times \vec{r}^{i,j-1}| + |\vec{r}^{i,j-1} \times \vec{r}^{i,j+1}|)} \quad (2.9)$$

If the CFD grid point  $i, j$  lies on the trailing edge, wing root, or wing tip, then only the CFD grid points which exist as its neighbors are used in Eqn. 2.8 and 2.9.

Next, each CFD grid point is mapped to a structural triangle. Using Fig. 2.3, step 1 shows the area used to obtain the force at CFD grid point  $i, j$  as indicated by

the dotted box and explained above. Steps 2-3 in Fig. 2.3 are designed to find the structural triangle associated with the CFD grid point. Here it is assumed that the CFD grid is denser than the CSD grid. The four closest structural nodes are obtained using the upper or lower surface structural grid depending on which surface the CFD grid point is located. All possible triangles are formed using the four CSD nodes. Triangles that do not contain the CFD point as an interior point are eliminated. The area coordinates of the CFD point  $i, j$  with respect to the structural triangle determine whether the point is an interior point. If the area coordinates sum to  $1.0 \pm 0.01$ , the CFD grid point is interior to the structural triangle. Area coordinates are explained later. From Fig. 2.3, there are four triangles and triangles 1 and 2 do not contain the CFD grid point and therefore are eliminated. Of the remaining triangles, the distance,  $v_i$ , between the CFD grid point  $i, j$  and each CSD node of triangle  $m$  is calculated as,

$$v_i^m = \sqrt{(x_p^m - x_a)^2 + (y_p^m - y_a)^2 + (z_p^m - z_a)^2} \quad \text{For } i = 1, 3 \quad (2.10)$$

where  $(x_a, y_a, z_a)$  are the coordinates of the CFD grid point  $i, j$  and  $(x_p^m, y_p^m, z_p^m)$  are the coordinates of CSD node  $p$  of triangle  $m$ . The largest vertex distance for each triangle  $m$  is obtained as

$$w_{max}^m = \max(v_1^m, v_2^m, v_3^m) \quad (2.11)$$

where  $\max$  is the maximum of the values  $v_1^m, v_2^m, v_3^m$ . The triangle with the smallest value of  $w_{max}$  is the “smallest” structural triangle for CFD point  $i, j$ ; thus the forces at CFD grid point  $i, j$  are mapped to this triangle. Using Fig. 2.3, triangle 3 has CSD node 4 as the farthest node from the CFD grid point  $i, j$ . Triangle 4 has CSD node 2 as the farthest node the CFD grid point  $i, j$ . In this example, triangle 4 has the smallest of the largest vertex distances, so it is chosen as the mapped CSD triangle for this CFD grid point.

Figure 2.4 shows that the force at the CFD grid point  $i, j$  is distributed to the CSD nodes of triangle 4 since it is the “smallest” structural triangle for CFD point  $i, j$ . The weight factors used are the area coordinates of the CFD grid point  $i, j$  within the structural triangle. Thus, the loads at the CSD nodes of the triangle are,

$$\begin{Bmatrix} \vec{F}_{n1}^{i,j} \\ \vec{F}_{n2}^{i,j} \\ \vec{F}_{n3}^{i,j} \end{Bmatrix} = \begin{bmatrix} L_1 & 0 & 0 \\ L_1 & 0 & 0 \\ L_1 & 0 & 0 \\ 0 & L_2 & 0 \\ 0 & L_2 & 0 \\ 0 & L_2 & 0 \\ 0 & 0 & L_3 \\ 0 & 0 & L_3 \\ 0 & 0 & L_3 \end{bmatrix} \begin{Bmatrix} G_x^{i,j} \\ G_y^{i,j} \\ G_z^{i,j} \end{Bmatrix} \quad (2.12)$$

where  $\vec{F}_{n1}^{i,j}$ ,  $\vec{F}_{n2}^{i,j}$  and  $\vec{F}_{n3}^{i,j}$  are the forces at nodes 1, 2, and 3 of the structural triangle due to the load at CFD point  $i, j$ .  $L_i$  are the area coordinates of the CFD grid point  $i, j$  within the structural triangle. Here  $n1, n2$ , and  $n3$  correspond to the actual node numbers of the CSD nodes to which the load at CFD point  $i, j$  is distributed. If a CSD node,  $n_i$ , is not part of the structural triangle which contains the CFD grid point  $i, j$ , then  $\vec{F}_{n_i}^{i,j} = \vec{0}$ .

The area coordinates  $L_i$  are obtained as follows. Three separate lines are drawn from the CFD grid point  $i, j$  to each CSD node of the structural triangle (Fig. 2.4). The three triangles formed have areas  $A_2, A_3$ , and  $A_4$ . The total area of the structural triangle is  $A = A_2 + A_3 + A_4$ . The area coordinates are then defined as  $L_i = \frac{A_i}{A}$ . If  $F$  is the force at CFD grid point  $i, j$ , then node  $i$  of the CSD triangle will have a force of  $L_i F$ . Area coordinates are helpful since no additional moment or twist needs to be applied to compensate for the transfer of the load.

Four CSD nodes were used to show this mapping algorithm, but this number can be increased to  $n_{clo}$ . The number chosen depends on the density of the structural grid. It is possible not to find a structural triangle for a CFD grid point if this number is too low. For example, if all four nodes in the previous example are to the same side of the CFD grid point, then none of the formed triangles would contain the CFD grid point. In this research,  $n_{clo} = 20$  was used. This number was validated by graphically viewing the mapping of the CFD grid points to the structural triangles for various choices  $n_{clo}$ .

The global force vector,  $\vec{F}_{n_i}^G$ , for each structural node  $n_i$  is calculated as follows:

$$\vec{F}_{n_i}^G = \sum_{i=1}^{i_{max}} \sum_{j=1}^{j_{max}} \vec{F}_{n_i}^{i,j} \quad \text{For } n_i = n1, n2, \dots, n_{max} \quad (2.13)$$

where  $i_{max}, j_{max}$  are the number of points on the wing CFD grid only, and  $n_{max}$  is the number of CSD nodes.

The global force vector,  $\{f_s\}$ , for the finite element model is obtained by,

$$\{f_s\} = \left\{ \begin{array}{c} \vec{F}_{n1} \\ \vec{F}_{n2} \\ \cdot \\ \cdot \\ \cdot \\ \vec{F}_{n_{max}} \end{array} \right\} \quad (2.14)$$

The structural response of the system is calculated using the forces obtained above on the CSD nodes. The following system of equations are solved,

$$[K]\{u_s\} = \{f_s\} \quad (2.15)$$

where  $\{u_s\}$  are the displacements at the CSD nodes, and  $\{K\}$  is the stiffness matrix of the CSD or finite element model. This can be solved by any structural analysis tool to obtain the displacements,  $\{u_s\}$ , on the CSD nodes.

The displacements,  $\{u_a\}$ , on the aerodynamic portion of the CFD grid are calculated using the structural response,  $\{u_s\}$ . A surface spline [44] is used to interpolate the displacements from the CSD nodes to the CFD grid points. Reasonable accuracy [51] is obtained as long as extrapolation is avoided. The surface spline equation is derived from the governing equations of a plate of infinite extent that deforms in bending only. The surface spline equation is

$$W_j = a_0 + a_1 x_j + a_2 y_j + \sum_{i=1}^N F_i r_{ij}^2 \ln r_{ij}^2 \quad \text{for } j=1, N \quad (2.16)$$

where  $W_j$  is the displacement at CSD node  $j$ ,  $r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$ , and  $N$  is the number of points where the displacements,  $W_j$ , are known. Note,

$$\lim_{r \rightarrow 0} r^2 \ln r^2 = 0 \quad (2.17)$$

Thus, the term  $r_{ij}^2 \ln r_{ij}^2$  is set to zero when  $r_{ij} = 0$ , though  $\ln r_{ij}|_{r_{ij}=0}$  does not exist. Eqn. 2.16 has  $N+3$  unknowns, but there are only  $N$  equations. Therefore, the three additional equations are

$$\sum_{i=1}^N F_i = \sum_{i=1}^N x_i F_i = \sum_{i=1}^N y_i F_i = 0 \quad (2.18)$$

These are the equations of equilibrium.

The surface spline system of equations becomes

$$[A^s]\{c\} = \{u_{spl}\} \quad (2.19)$$

where  $[A^s]$  is dependent on the coordinates of the spline points,  $\{c\}$  is the vector of unknown coefficients of the surface spline equation, and  $\{u_{spl}\}$  are the displacements at the spline points. In the preprocessing stage, some of the structural nodes are chosen as the spline points.  $[A^s]$  is formed using the coordinates of the chosen spline points. The spline point displacements,  $\{u_{spl}\}$ , are extracted from the structural response,  $\{u_s\}$ , as

$$\{u_{spl}\} = [E]\{u_s\} \quad (2.20)$$

Here  $[E]$  is a  $n_{spl} \times n_{max}$  matrix where  $n_{spl}$  is the number of spline points and  $n_{max}$  is the number of CSD nodes.  $[E]$  is composed of zeroes and ones.  $[A^s]$  is decomposed using an LU factorization. The coefficients of the surface spline,  $\{c\}$ , are solved by forward and backward substitutions.

The displacements at the CFD surface grid points,  $\{u_a\}$ , are calculated by using the coordinates of the CFD grid points within the surface spline equation. The exterior CFD grid is deformed using the CFD surface grid displacements,  $\{u_a\}$ , but the deformation of the exterior CFD grid depends on the aerodynamic analysis tool. Two separate codes for fluid analysis are used in this research. One of the codes, ENSAERO [47], has a built in scheme to move the grid once the CFD surface grid is deformed. The other code, NASTD [52], does not have a scheme to move the grid. So a simple grid moving scheme was applied when NASTD was used. This is explained in Sec. 3.1.2.

The aeroelastic coupling procedure is demonstrated by calculating a flexible solution of an F/A-18 Stabilator (horizontal tail) using Euler flow equations in NASTD

(an in-house McDonnell Douglas Aerospace East CFD code) coupled with an advanced structural analysis tool, NASTRAN. Also, a flexible solution of the the Aeroelastic Research Wing (ARW-2) is calculated by using Navier-Stokes flow equations in ENSAERO in conjunction with a finite element wing-box code (Ref. [12]); the wing-box code is developed as a part of this research.

In summary, a CFD/CSD interface approach consisting of two mappings has been presented. One mapping transfers loads from the CFD grid points to the CSD nodes. The second mapping interpolates the displacements from the CSD nodes to the CFD grid points. Both of these mappings require only the coordinates of the CFD surface grid and the CSD nodes. Therefore, the mappings can be created in the preprocessing stage. The mappings communicate essential information using the native files of the respective codes; thus no code integration is required. Hence, static aeroelastic solutions of wings can be obtained using a wide variety of CFD and CSD codes.

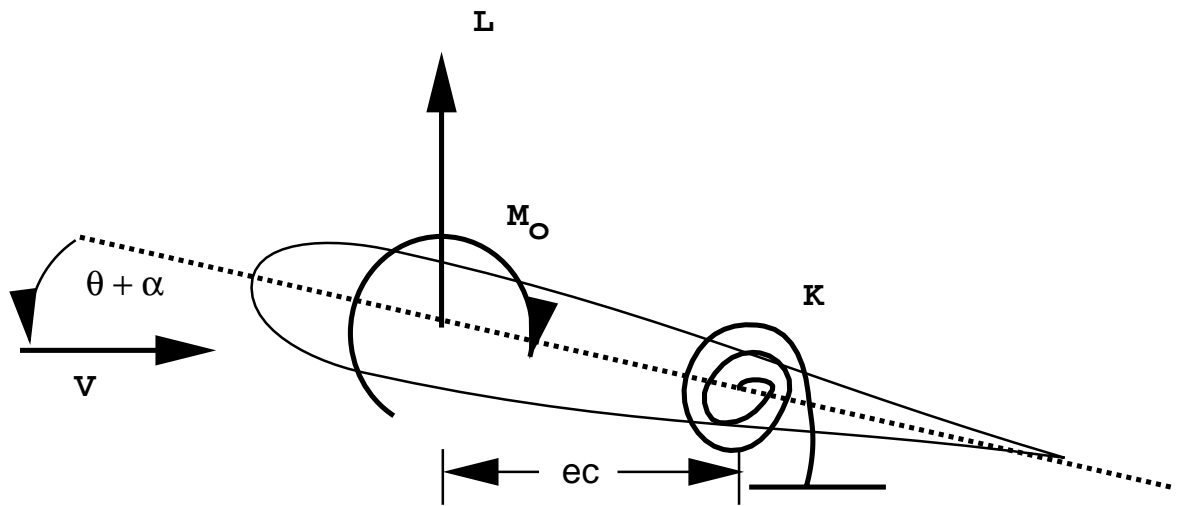


Figure 2.1: Diagram of 2-D Airfoil

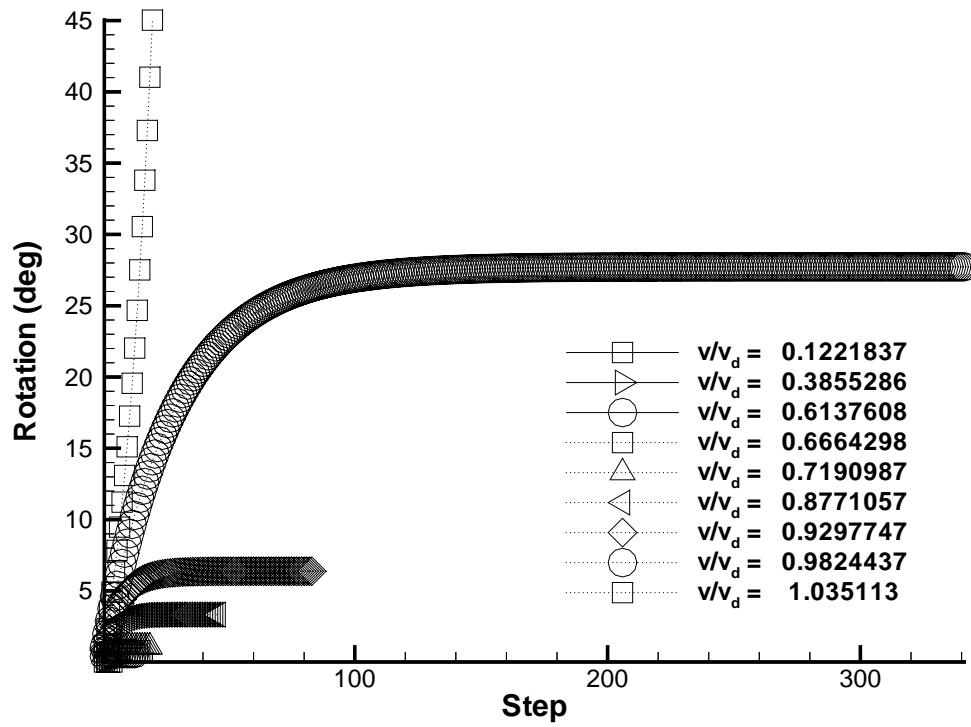
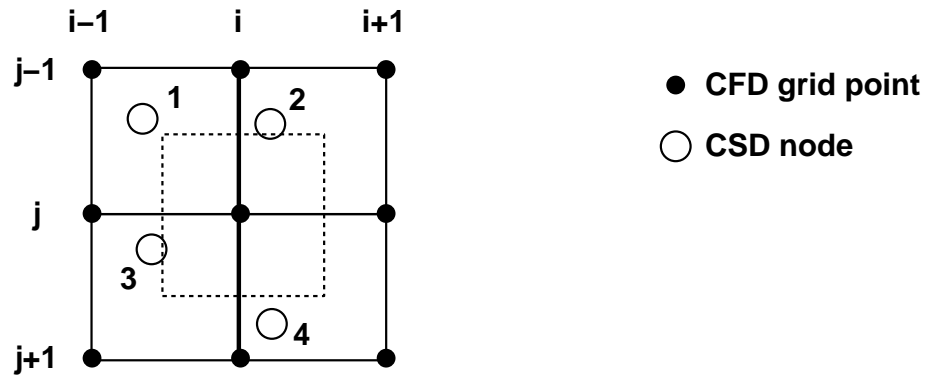
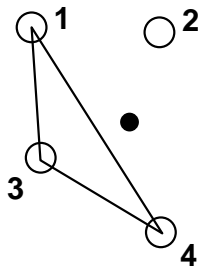


Figure 2.2: Convergence of Solution for 2-D Airfoil

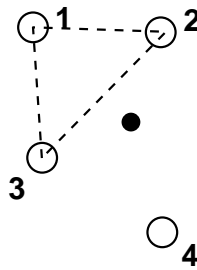
Step 1



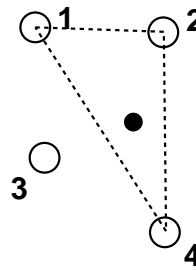
Triangle 1



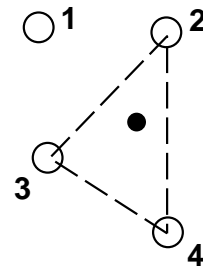
Triangle 2



Triangle 3



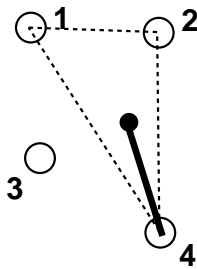
Triangle 4



Step 2

Step 3

Triangle 3



Triangle 4

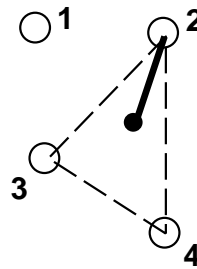


Figure 2.3: Mapping of a CFD Grid Point to a CSD Triangle

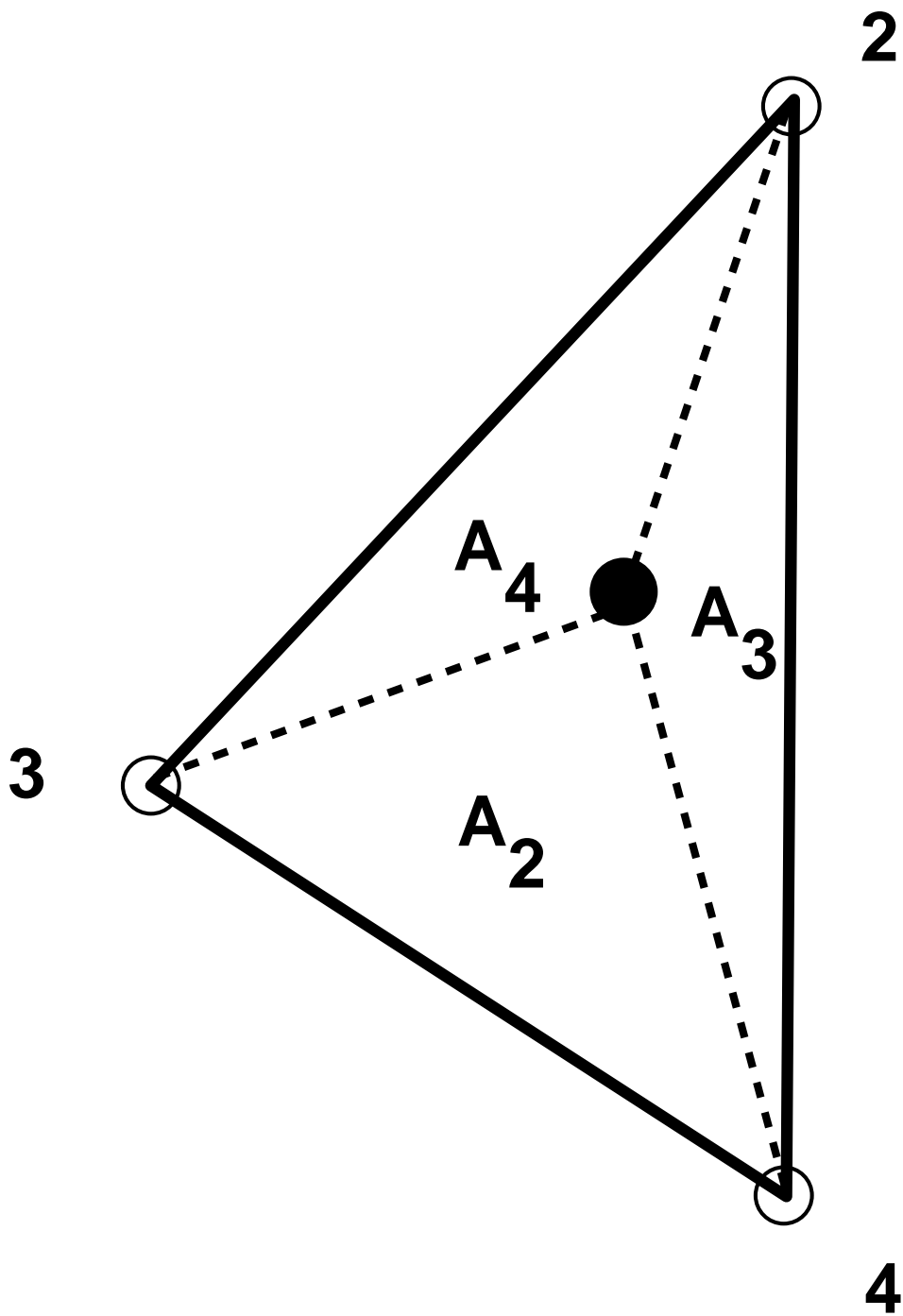


Figure 2.4: Area Coordinates of a CFD Grid Point within a CSD Triangle

# Chapter 3

## Analysis of Aircraft Wings

Details of the static aeroelastic analyses on two lifting surfaces, the F/A-18 Stabilator and the Aeroelastic Research Wing (ARW-2), are presented in this chapter. To demonstrate the aeroelastic coupling procedure, static aeroelastic solutions of the F/A-18 Stabilator and the Aeroelastic Research Wing (ARW-2) are obtained in the transonic regime. A complex finite element model of both of the wings' structures is used in conjunction with an advanced CFD tool to capture the aeroelastic interactions. The static aeroelastic solution of the two modern complex wings is compared with experimental and other available computational data.

### 3.1 F/A-18 Stabilator

#### 3.1.1 CFD and CSD Modeling

For the F/A-18 Stabilator, Euler flow equations, as available in NASTD, are used to demonstrate the aeroelastic coupling procedure. The analysis is performed at sea-level, one degree angle of attack, and Mach 0.95. The CFD grid of the F/A-18 Stabilator, as seen in Fig. 3.1, is approximately 800,000 grid points.

A general purpose finite element program, NASTRAN, is implemented to analyze the structure. The stiffness matrix produced by NASTRAN is used to obtain the

displacements for given aerodynamic loads. NASTRAN is utilized to obtain the stiffness matrix,  $[K]$ , of the structure. Another simple code is used to solve the structural system of equations using the  $[K]$  matrix produced by NASTRAN. Therefore, during the linear aeroelastic analysis procedure, NASTRAN is not directly involved, since the stiffness matrix does not change during the procedure. The finite element model of the F/A-18 Stabilator, as seen in Fig. 3.2, consists of 2000 nodes and 12000 d.o.f.

### 3.1.2 Aeroelastic Coupling Procedure

The first step in the aeroelastic coupling procedure is obtaining the CFD solution for the lifting surface. For this case, the rigid steady state solution is obtained before the aeroelastic analysis cycle begins. Once the CFD solution is obtained, the forces on the CSD grid are calculated using the preprocessed mapping. The mapping of the CFD points to the structural triangles, previously discussed in chapter 2, is shown in Fig. 3.3. Here the mapped structural triangle for each CFD point is presented. The structural triangle does not refer to an actual structural element. So shape functions are not necessary, and if linear displacements are assumed over each element, then energy is conserved during the mapping. The actual structure of the wing does not extend to the wing root, but this was done to avoid computational problems. This was required by the CFD code NASTD.

Once the forces on the CSD grid are known, the structural response,  $\{u_s\}$ , is obtained by solving the structural system of equations. The spline points for the Stabilator are chosen to be a subset of the structural nodes and some far field points of the CFD grid. The choice of these spline points is subjective. The spline points are chosen as to be distributed evenly across the planform of the surface. So, after the displacements on the nodes,  $\{u_s\}$ , are obtained, the displacements,  $\{u_{spl}\}$ , at the spline points are extracted. The spline points for this case are shown in Fig. 3.4. The reason for this choice can be seen when looking at Fig. 3.5, which is the surface grid of the F/A-18 Stabilator. The surface grid includes the aerodynamic surface and the points extending beyond the wing tip in the spanwise direction, and the points extending beyond the trailing edge in the chordwise direction. The points on

the aerodynamic surface grid are chosen so the displacements vary smoothly from the aerodynamic surface to the farfield. The right hand side of the surface spline system of equations is known, so the surface spline coefficients are obtained. Next the deflections on the CFD surface grid are calculated using the surface spline equation.

Next, the exterior grid is deformed. The CFD grid for this case has the  $i$  index varying circumferentially around the wing section, the  $j$  index varying in the normal direction, and the  $k$  index varying along the span. Once the surface deflections are known at  $j = 1$ , a cosine spacing function is used to deform the exterior grid at each spanwise ( $k = \text{constant}$  face) location. The spacing function is dependent on the location along the normal direction, i.e. the  $j$  index. Fig. 3.6 show the plot of the spacing function used, given as

$$\alpha_s^j = \cos \frac{\pi(j-1)}{2(j_{max}-1)} \quad \text{for } j = 1, j_{max} \quad (3.1)$$

where  $j_{max}$  are the maximum number of points extending in the radial direction normal to the surface. Using the displacements at the CFD surface grid, i.e.  $j = 1$ , the exterior grid is deformed at each  $k = \text{constant}$  surface, by multiplying the surface displacement by the spacing parameter,  $\alpha_s$ , i.e. the new vertical coordinate at some  $j$  section is,

$$z_{i,k}^{new} = z_{i,k}^{rigid} + \alpha_s^j u_{i,k}^{j=1} \quad (3.2)$$

Only the vertical displacements are taken into account. Note that the  $z_{i,k}^{rigid}$  coordinates are used and not the  $z$  coordinates from the previous iteration. To avoid overlapping of the CFD grid, a minimum spacing criteria,  $\alpha_{min}$  is chosen as,

$$\alpha_{min} = fs * (\alpha_s^1 - \alpha_s^2) \quad (3.3)$$

where  $\alpha_s^1 = 1$  and  $fs$  is subjectively chosen to prevent loss of grid quality. For this analysis,  $fs$  is chosen in the range of 1-2.  $\alpha_s^2$  depends on  $j_{max}$ . This assumes the grid is stretching smoothly away from the surface. If the spacing between two consecutive points is smaller than  $\alpha_{min}$ , if  $z_{i,k}^{j+1} - z_{i,k}^j < \alpha_{min}$ , then  $\alpha_s$  is set to one for that entire  $j$  section. In this example, all the points within the  $j = 26$  boundary are moved the same amount as the aerodynamic surface at  $j = 1$ . All the points exterior to  $j = 26$ ,

i.e.  $26 < j < j_{max}$ , are moved using Eqn. 3.1. This enforces that the outer boundaries of the CFD grid do not move. This is done to take advantage of distributed computing capabilities in the future where the grid can be broken into many zones. So, to avoid problems with grid mismatching at the zonal boundaries, the zonal boundaries are fixed. In this case, the CFD grid is broken into two zones, but distributed computing was not used. Zone 1 consists of the Stabilator CFD grid, and zone 2 consists of the region extending downstream from the trailing edge. The CFD solution is obtained in zone 1. The boundary information is used to calculate a CFD solution in zone 2. This process is repeated until convergence of some specified criteria is obtained.

After the CFD grid is deformed, the aeroelastic coupling procedure is repeated until some specified convergence criteria is met. Initially, the rigid steady state solution was obtained before exchanging information with the CSD code. After initiating the aeroelastic coupling procedure, the CFD solution was not fully converged before exchanging information with the CSD code. The number of iterations during each cycle was about  $200 \pm 10$ . This has a similar effect as an underrelaxation scheme. A Hewlett-Packard workstation was used to perform the calculations.

## 3.2 Aeroelastic Research Wing (ARW-2)

### 3.2.1 CFD and CSD Modeling

The Aeroelastic Research Wing (ARW-2), a supercritical airfoil with aspect ratio of 10.3 and a leading edge sweep of  $28.8^\circ$ , is used to validate the force and displacement mappings. In addition, it also provides as a validation tool for the finite element wing-box code. The strong conservation law form of the thin-layer Reynolds-averaged Navier-Stokes equations are used to calculate the fluid flow about the ARW-2 wing as available in ENSAERO. The structural response is calculated by the finite element wing-box code (see Section 5.2). The two codes are coupled using the aeroelastic coupling procedure presented in chapter 2.

The CFD code uses a C-H type grid with a grid size of 171 (circumferentially) x 51 (spanwise) x 45 (normal) points. The wing CFD grid is shown in Fig. 3.8. The

wing has a grid size of 139 (circumferentially) x 39 (spanwise) points. The fluid flow equations are solved for Mach 0.85, an angle of attack,  $\alpha$ , of 1 and 2 degrees, and a free stream dynamic pressure,  $q$ , of 200 psf.

The finite element wing-box model of the ARW-2 wing uses Allman's triangular elements in conjunction with axial bars to represent the wing's spars, ribs, and skins. Figure 3.9 shows the spars and ribs of the ARW-2 wing. The wing is discretized into a 11 x 13 mesh, 312 nodes, 1872 d.o.f. The ARW-2 wing consists of composite fiberglass skins, but the finite element wing-box code does not yet have composite capability. An equivalent isotropic wing is created by matching bending and twisting properties with the ARW-2 wing made of composite fiberglass skins.

### 3.2.2 Aeroelastic Coupling Procedure

The aeroelastic coupling procedure is more integrated using ENSAERO and the finite element wing-box code since the source code of ENSAERO is available. The pressures from ENSAERO are obtained and mapped to forces on the finite element model. The mapping of the structural triangle to the CFD point is shown in Fig. 3.10. The finite element wing-box code then solves for the structural response,  $\{u_s\}$ . The displacements at the spline points (Fig. 3.11),  $\{u_{spl}\}$ , are extracted. The surface spline coefficients are calculated, and the displacements at the CFD grid points are obtained. Again, only the vertical displacements are used. This version of ENSAERO only requires vertical displacements at the CFD aerodynamic surface, i.e. the wing surface. Then ENSAERO regenerates the exterior grid and the pressures are recalculated. The process is repeated until a convergence of the solution is reached. Convergence of the CFD solution is monitored by whatever criteria the CFD code uses. Here the  $L_2$  norm of the residuals of the CFD equations is used as the criteria for convergence. The loads were also compared to ensure convergence. Convergence of the CSD solution is checked by examining the tip displacement after each aeroelastic cycle and the displacements at other locations, as seen in chapter 4.1.

Since only the vertical displacements are taken into account for the F/A-18 Stabilator and ARW-2 wing, the quality of the CFD grid can be poor. ENSAERO uses

the vertical deflections to calculate a rigid body rotation and a deflection so as to avoid this problem when dealing with the ARW-2 wing. This was also done for the F/A-18 Stabiltor using NASTD. This means that chordwise rigidity is assumed for the wing. This is a good approximation for the ARW-2 wing. Byrdsong *et al.* measured experimental data for the flexible ARW-2, where it was stated that the ARW-2 has sufficiently chordwise rigidity.

The aeroelastic solution is obtained at Mach 0.85,  $\alpha = 1$  and 2 degrees,  $q = 200$  psf, and compared with experimental results. In addition, the results are also compared with another similar work, which uses modal analysis as opposed to the direct finite element analysis used in this study. A Cray-90 was used to obtain the solution for this case.

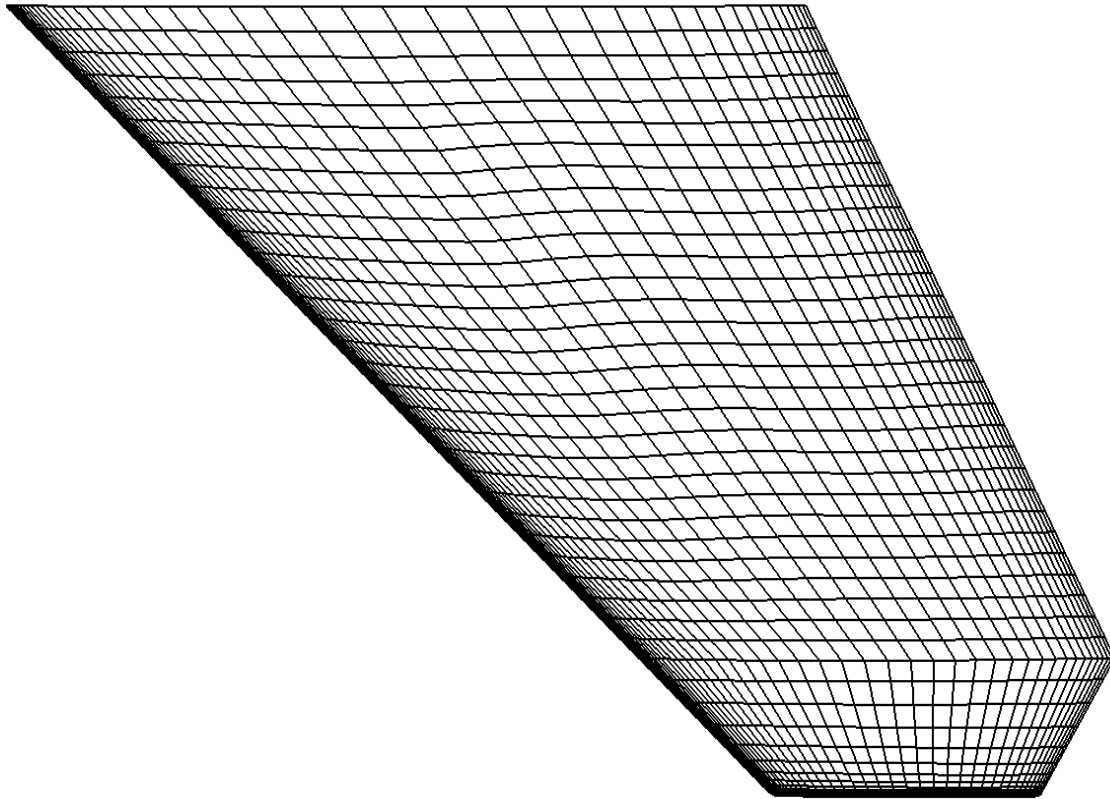


Figure 3.1: CFD Grid for the F/A-18 Stabilator

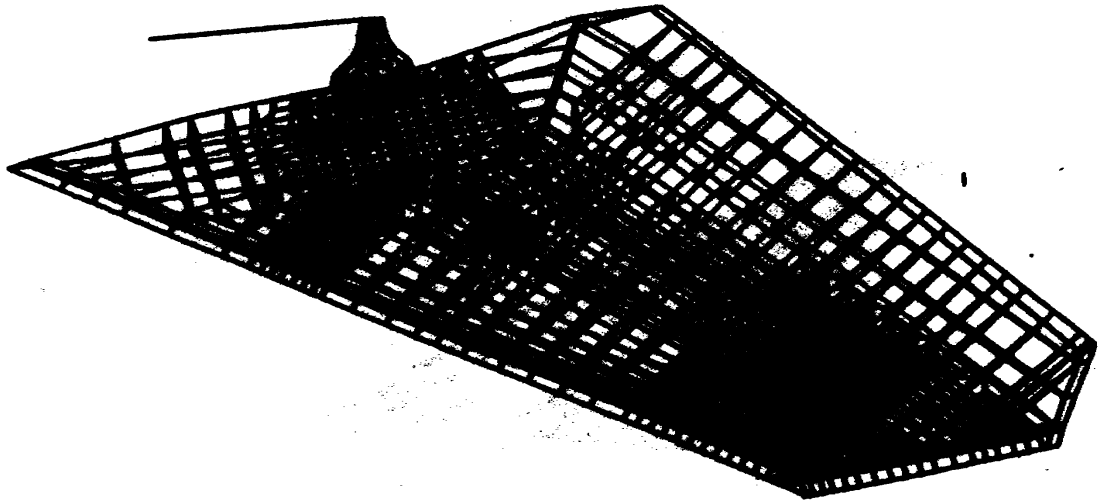


Figure 3.2: Finite Element Model of the F/A-18 Stabilator

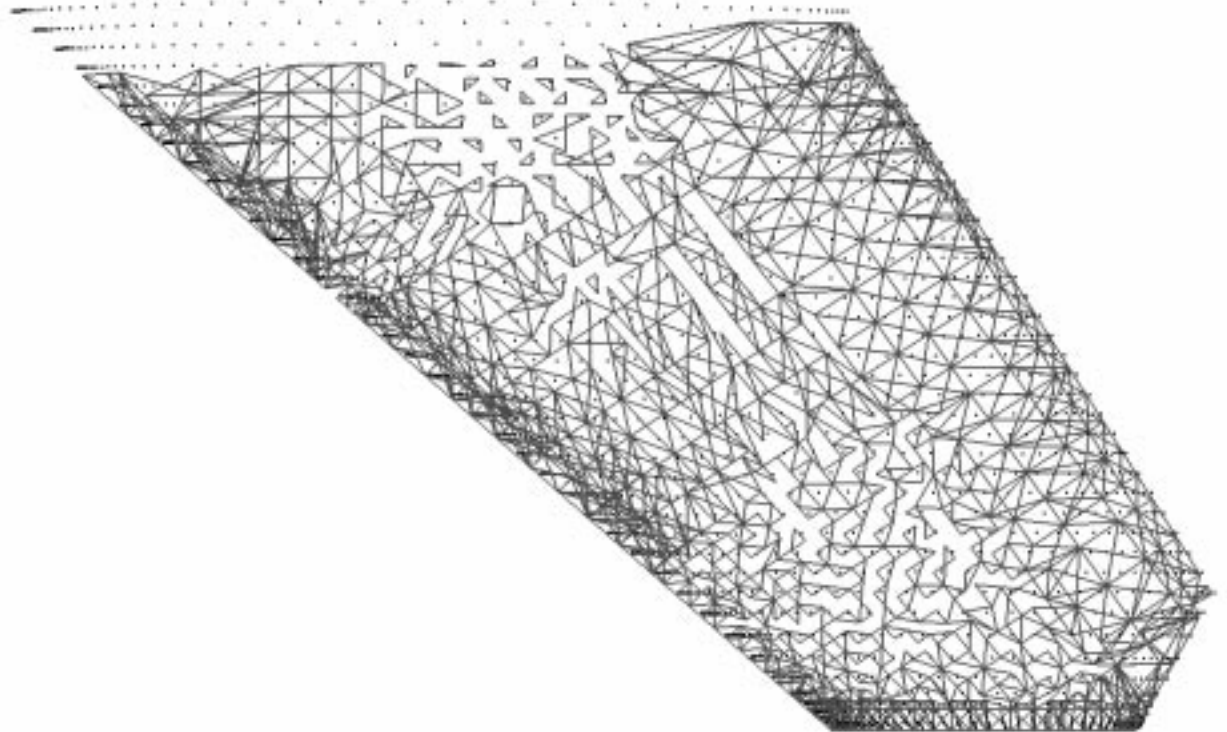


Figure 3.3: Mapping of CFD Points to Structural Triangles for the F/A-18 Stabilator

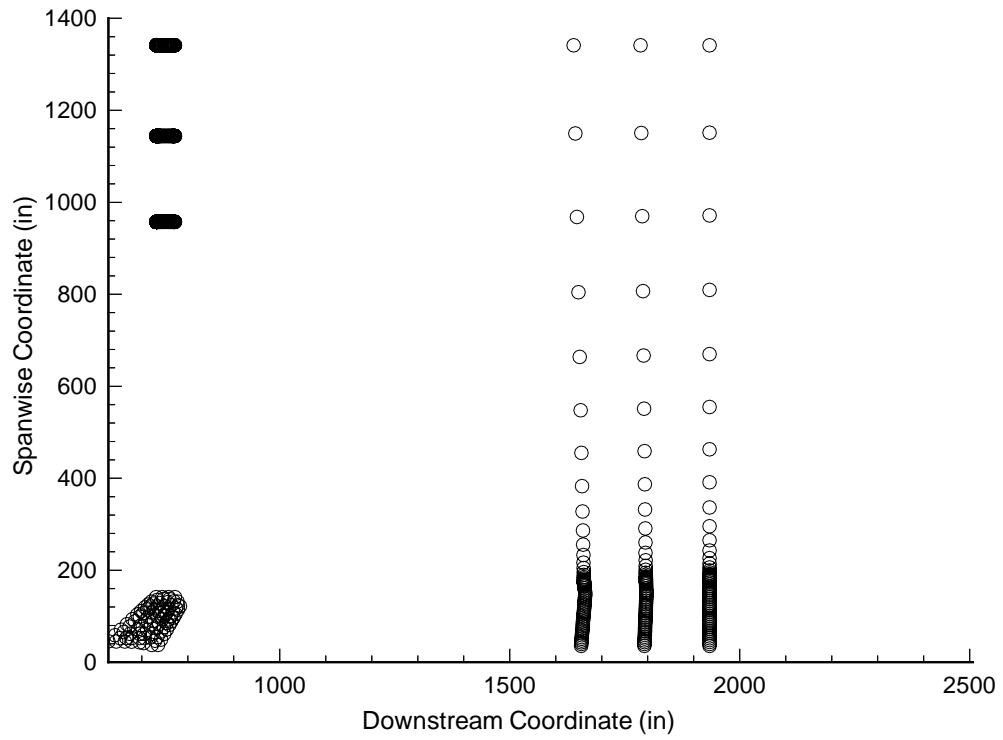


Figure 3.4: Spline Points Used for Mapping for F/A-18 Stabilator

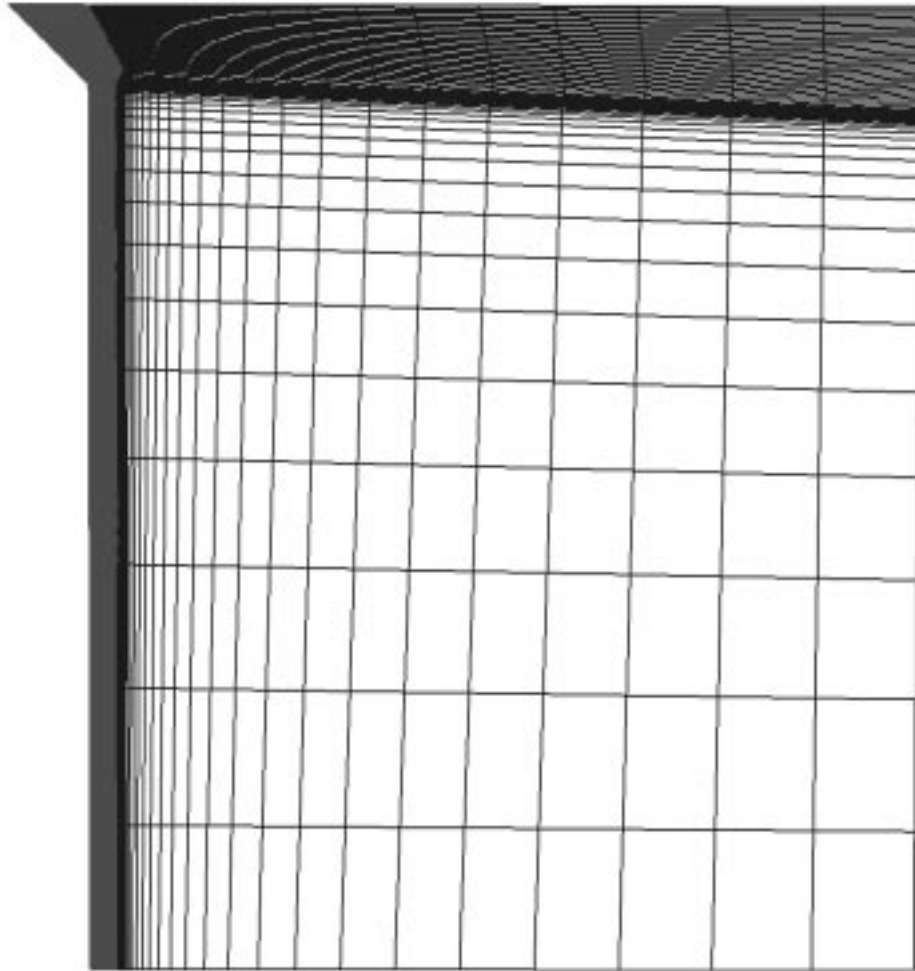


Figure 3.5: CFD Surface Grid of the F/A-18 Stabilator

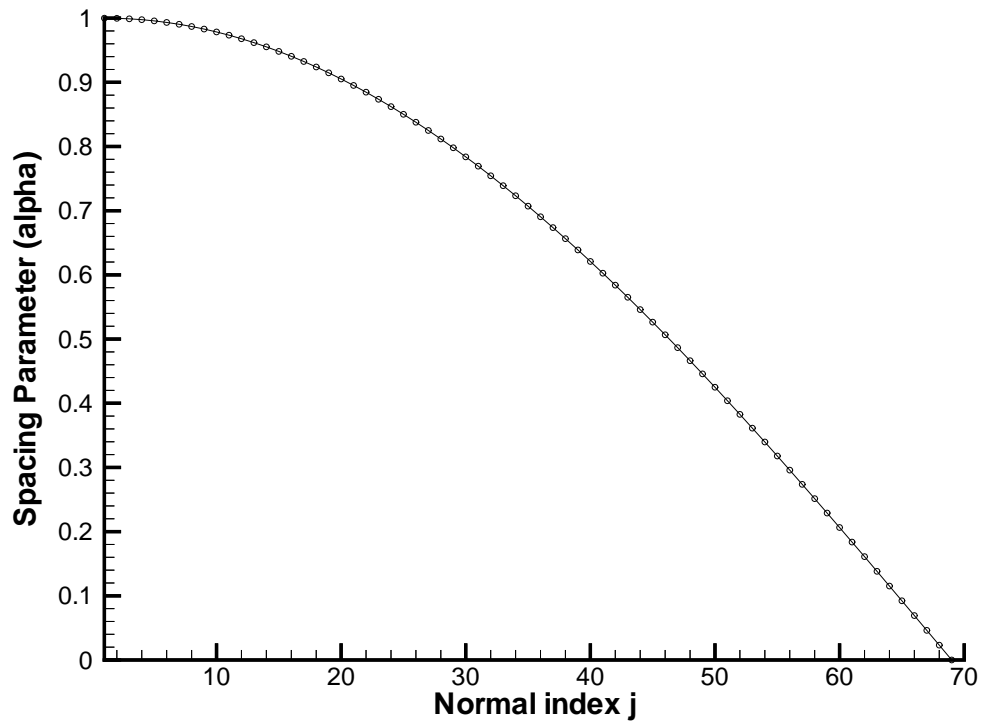


Figure 3.6: Cosine Spacing Function Used to Deform Exterior Grid

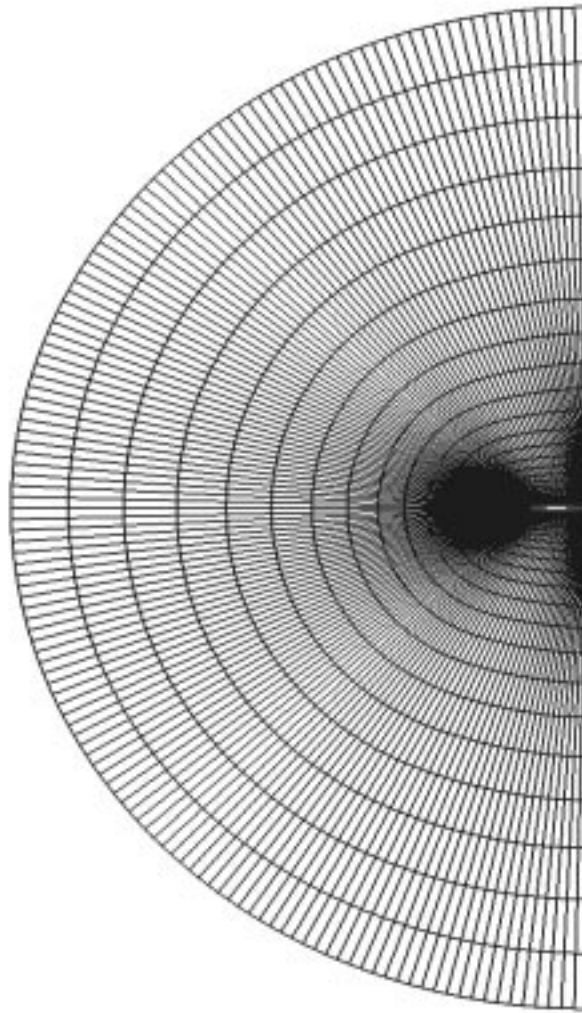


Figure 3.7:  $k = \text{Constant}$  Face of the CFD Grid of the F/A-18 Stabilator

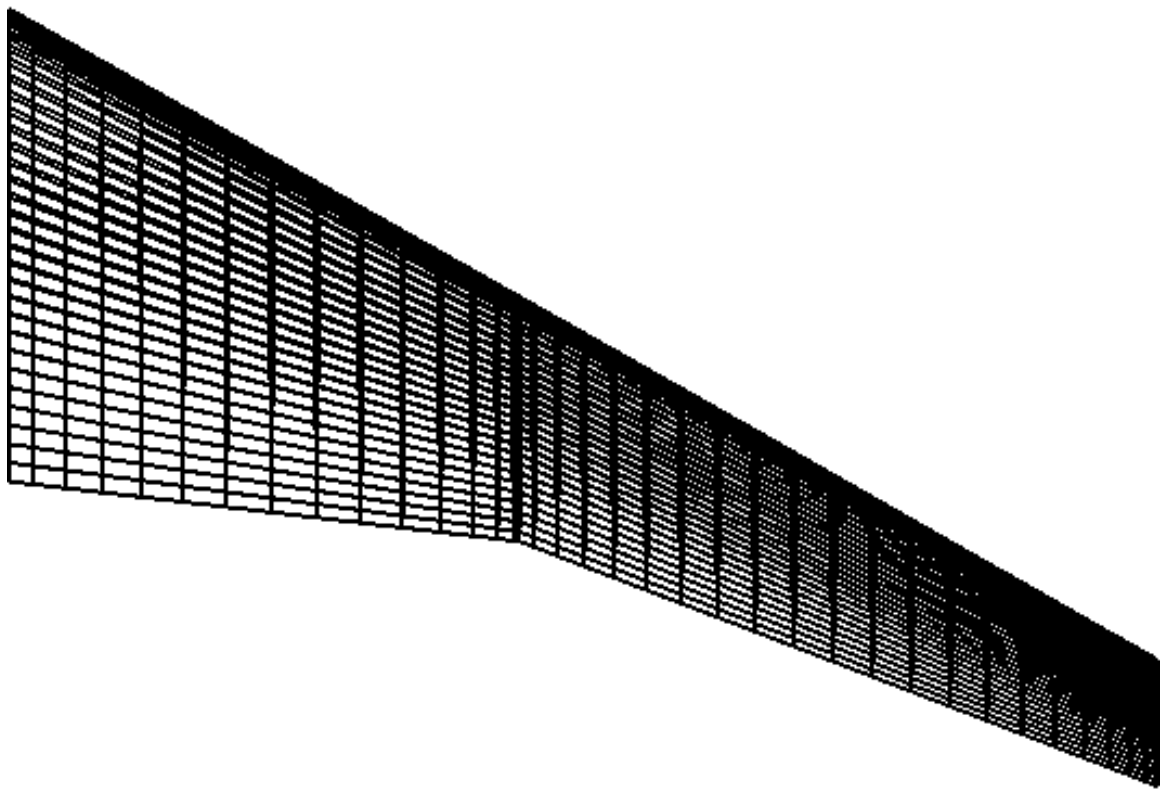


Figure 3.8: CFD Grid of the ARW-2 Wing

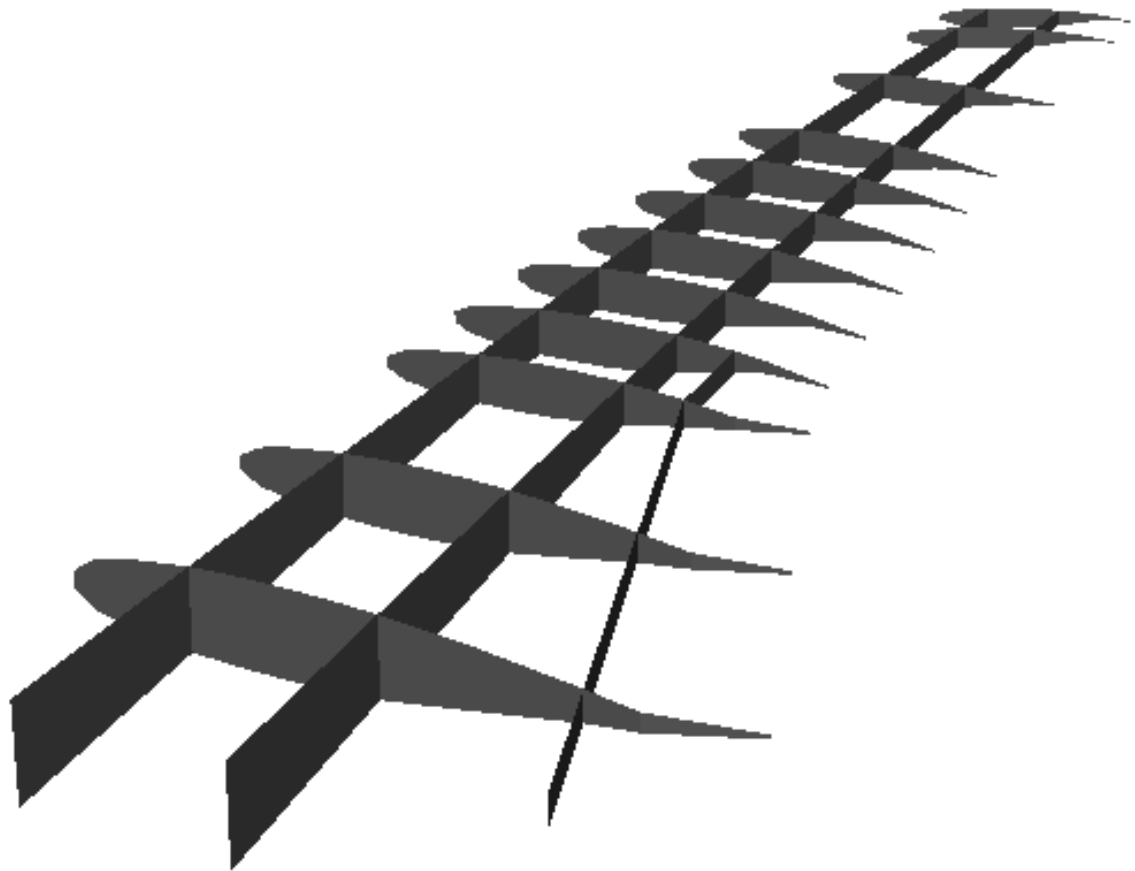


Figure 3.9: Finite Element Model of the ARW-2 Wing

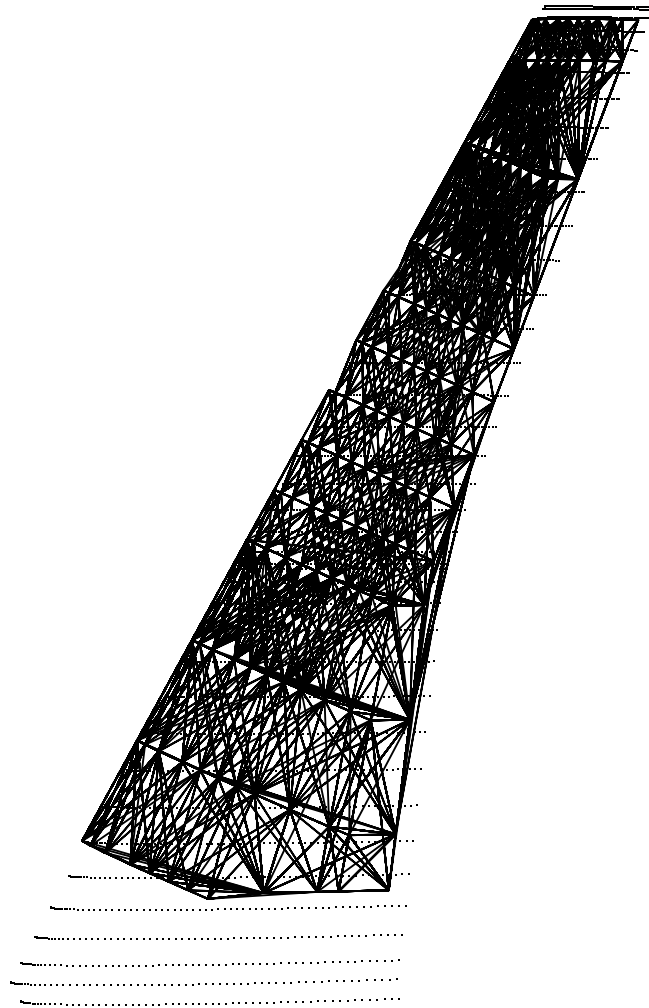


Figure 3.10: Mapping of Structural Triangles to CFD Points for the ARW-2 Wing

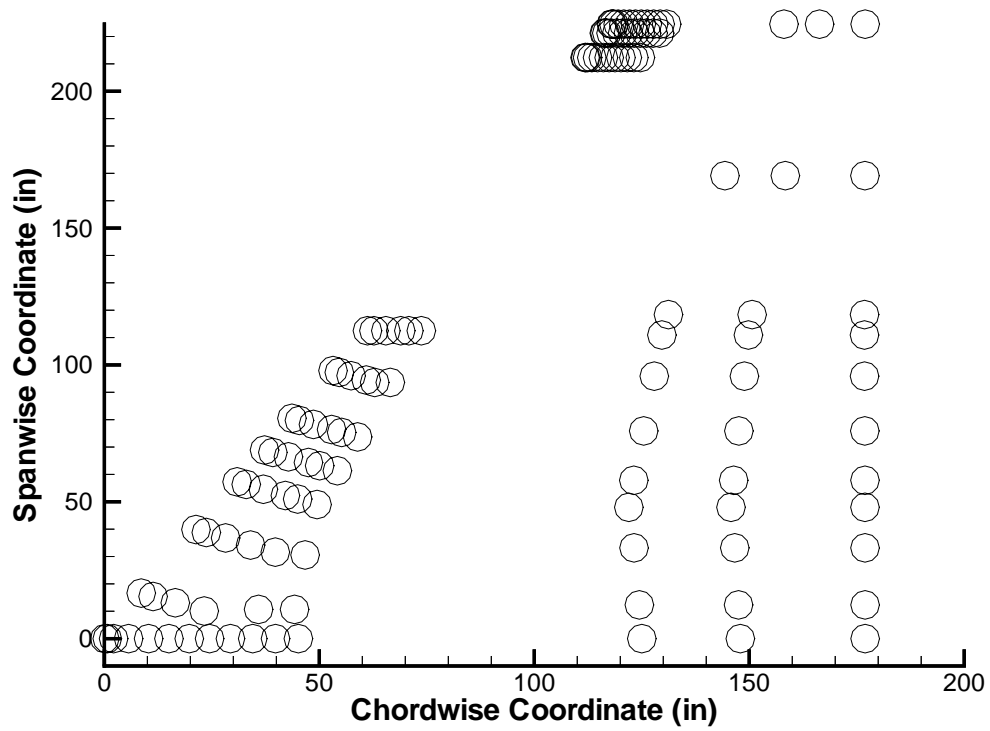


Figure 3.11: Spline Points Used in Aeroelastic Coupling of ARW-2 Wing

# Chapter 4

## Results

### 4.1 F/A-18 Stabilator

The convergence of the aeroelastic solution for the F/A-18 Stabilator is monitored in several ways. The  $L_2$  norm of the residuals of the continuity, momentum, and energy equations are examined. The loads on the wing surface are also examined. Examining the two mentioned criteria helps assure that the CFD solution is converged. In the CSD solution, the displacements at various locations are examined to assure convergence. One of the convergence checks for the structural analysis is shown in Fig. 4.1, where the deflection of the wing tip of the F/A-18 Stabilator is plotted after each cycle of the aeroelastic coupling procedure. In addition, Fig. 4.2 shows the convergence of the trailing edge tip of the F/A-18 Stabilator. The structural solution converges very quickly. This is because the rigid steady state solution was obtained prior to initiating the aeroelastic coupling procedure. In addition, the aeroelastic effect is not significant; the largest displacement on the F/A-18 Stabilator is 1.55 inches.

The final converged flexible F/A-18 Stabilator is shown in Fig. 4.3 with the initial undeformed rigid F/A-18 Stabilator. The largest deflection occurs at the trailing edge tip of the F/A-18 Stabilator, approximately 1.5 inches. The pressure coefficient variation of the flexible versus rigid F/A-18 Stabilator is shown in Fig. 4.4 and 4.5. In addition, the Mach number variation of the flexible versus rigid F/A-18 Stabilator

is shown in Fig. 4.6 and 4.7. A presence of a shock can be seen in the figures. The pressure coefficient variation and Mach variation contours show that the aeroelastic effect is not significant, but this may be due to the fact that the aeroelastic solution is obtained at one degree angle of attack at 0.3 psi, and not much aeroelastic interaction is occurring. From a previous analytical study (performed at McDonnell Douglas) using CAP-TSD, a transonic small disturbance CFD code, coupled with modal analysis structures, the largest deflection of the F/A-18 Stabilator was calculated to be 1.56 inches. The deflection using NASTD coupled with NASTRAN is also about 1.5 inches. The present results do compare well with existing data. Unfortunately, more details of the comparisons are not available.

Next, the Aeroelastic Research Wing (ARW-2), is used to determine the accuracy of the entire aeroelastic coupling procedure, since experimental static aeroelastic data exist for it.

## 4.2 Aeroelastic Research Wing (ARW-2)

The ARW-2 wing is composed of spars and ribs made of isotropic materials while the skins are made of composite materials. The finite element wing-box code used in this study does not have composite capability, therefore an isotropic ARW-2 wing model was created. The isotropic ARW-2 wing model was developed by matching the structural properties of the spars and ribs with the composite skin ARW-2 wing. The thicknesses of the spars, ribs, and skins are generated to match the composite skin ARW-2 wing. To match the twisting properties of the isotropic ARW-2 wing with that of the composite skin ARW-2 wing, the thicknesses of the skins and the cross-sectional areas of the axial bars are altered. This is done so as not to change the bending behavior of the isotropic ARW-2 wing while trying to match the twisting behavior with that of the composite skin ARW-2 wing. To validate the isotropic ARW-2 wing model, two different loading conditions are applied to both wings, and the structural response is compared. The structural response of the composite skin ARW-2 wing was obtained using Engineering Analysis Language (EAL) at NASA Langley Research Center.

### 4.2.1 Validation of the ARW-2 Wing Finite Element Model

For validation, the bending and twisting behaviors of the isotropic ARW-2 wing are compared with the bending and twisting behaviors of the composite skin ARW-2 wing. The structural response of the composite skin ARW-2 wing is calculated using Engineering Analysis Language (EAL). Figures 4.8, 4.9, and 4.10 show the comparisons between the displacements of the front, rear, and auxiliary spars of the composite skin and isotropic ARW-2 wings with a 100 lb load applied upward at the wing tip on the front spar. The bending behaviors of the composite skin and isotropic ARW-2 wings are in good agreement. Note, the auxiliary spar does not extend to the wing tip although the displacements are shown for the entire span.

Figures 4.11, 4.12, and 4.13 show the deflections and twisting of the front and rear spars of the isotropic and composite skin ARW-2 wings with a 1 lb upward and a 1 lb downward force applied to the front and rear spars, respectively, at the wing tip. Fairly good agreement is obtained with the composite skin ARW-2 wing. Details of the composite skin ARW-2 wing finite element model can be obtained in Ref. [53].

### 4.2.2 Rigid Steady State Solution

The next step after the validation of the ARW-2 wing finite element model is to obtain the rigid steady state solution for the two cases, i.e. the one and two degree angle of attack cases. Intermediate rigid steady state solutions were obtained by using Navier-Stokes flow equations as available in ENSAERO. Convergence of the rigid steady state solutions is checked by examining the  $L_2$  norm of the residuals of the fluids equations. Figures 4.14 and 4.15 show the convergence of the relative  $L_2$  norm of the residual of the Navier-Stokes equations for the one and two degree cases, respectively. The  $L_2$  norm has not been sufficiently reduced, but this is done since a completely converged solution is not necessary to start the aeroelastic coupling. In addition,  $C_p$  variation for the rigid steady state solution at  $\alpha = 1$  degree is compared with computational results from Ref. [54]. This study and the Farhangnia *et al.* [54] study start with the same rigid steady state solution of the ARW-2 using ENSAERO. Farhangnia *et al.* use the first five mode shapes as opposed to the direct finite element

equations used in the work. Since final results are compared later, the starting points are compared by examining Fig. 4.16. This shows the  $C_p$  variation at the 70.7% semi-span location. Because both studies used ENSAERO to obtain the rigid steady state solution, the results match as expected.

After the rigid steady state solutions are obtained, static aeroelastic analysis is performed by including the finite element wing-box code and the preprocessed mappings. Convergence is checked by examining CFD and CSD solution criteria. The CFD solution convergence is checked by examining the  $L_2$  norm of the residual of the fluids equations, while the CSD solution is checked by examining displacements at various locations on the wing structure. Figures 4.17 and 4.18 show the  $L_2$  norm during the aeroelastic analysis. Flexible steady state solutions are obtained at  $\alpha = 1$  and 2 deg. The spikes coincide with the restarting of ENSAERO.  $C_p$  variation at the 70.7% semi-span location, for the flexible ARW-2 wing, is shown in Fig. 4.19 and plotted with experimental data from Ref. [54]. The  $C_p$  variation compares well with the experimental data. The shock location for the experimental data is 5% of chord aft of the computational data.

Figures 4.20 and 4.21 show the section lift coefficient along the span for the rigid and flexible wings. Note the decrease in lift when flexibility of the wing is included in the analysis. The ability to predict the effects of flexibility on load distribution in the transonic regime would be a useful tool. The designer can use this to help improve the design of the wing in the preliminary stages.

Figures 4.22 and 4.23 show the  $C_p$  variation on the upper surface of the flexible and rigid ARW-2 wing for the one and two degree angle of attack cases. Due to the flexibility, the shock location has moved aft in both cases. The  $C_p$  plot at the 70.7% semi-span location is shown in Fig. 4.24 verifies this for  $\alpha = 2$  deg case. For  $\alpha = 1$  deg case, the shock movement is less.

Figures 4.25 and 4.26 show the deflections of the front and rear spars, respectively, for the one degree angle of attack case. Experimental data from Byrdsong *et al.* [55] is also shown. The wing tip for the one degree case deflects approximate six inches, while the wing tip for the two degree case deflects approximately eight inches. Good agreement is obtained.

Figures 4.27 and 4.28 show the deflections of the front and rear spars for the two degree angle of attack case as compared with experimental data from Byrdsong *et al.* [55]. Again, good agreement is obtained using direct finite element data coupled with Navier-Stokes flow equations.

In addition, Fig. 4.29 also shows aeroelastic data from Farhangnia *et al.* [54] where modal analysis was used for structural analysis for the one degree case. Modal analysis results are about 25% in error at the wing tip, where the first five mode shaped were used. Finite element equations results are 3% in error compared to experimental data. Here it is shown the increased accuracy of using direct finite element displacement data as opposed to modal analysis data. Again, the accuracy of the aeroelastic coupling procedure and the finite element wing-box code are demonstrated successfully.

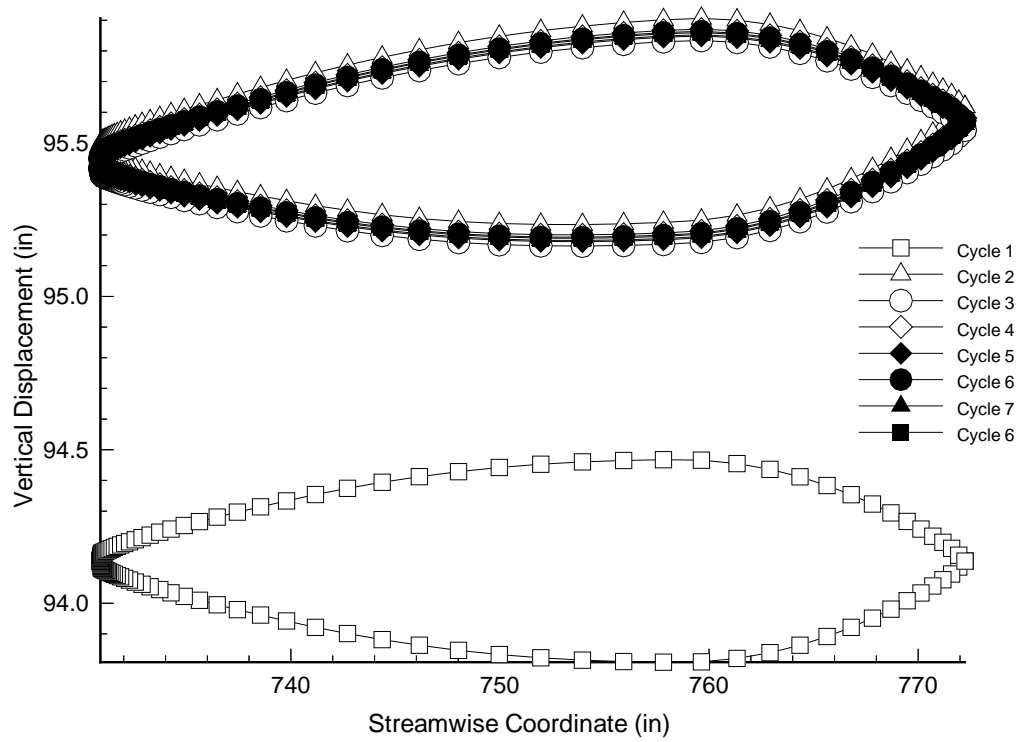


Figure 4.1: Convergence of the Wing Tip of the F/A-18 Stabilator

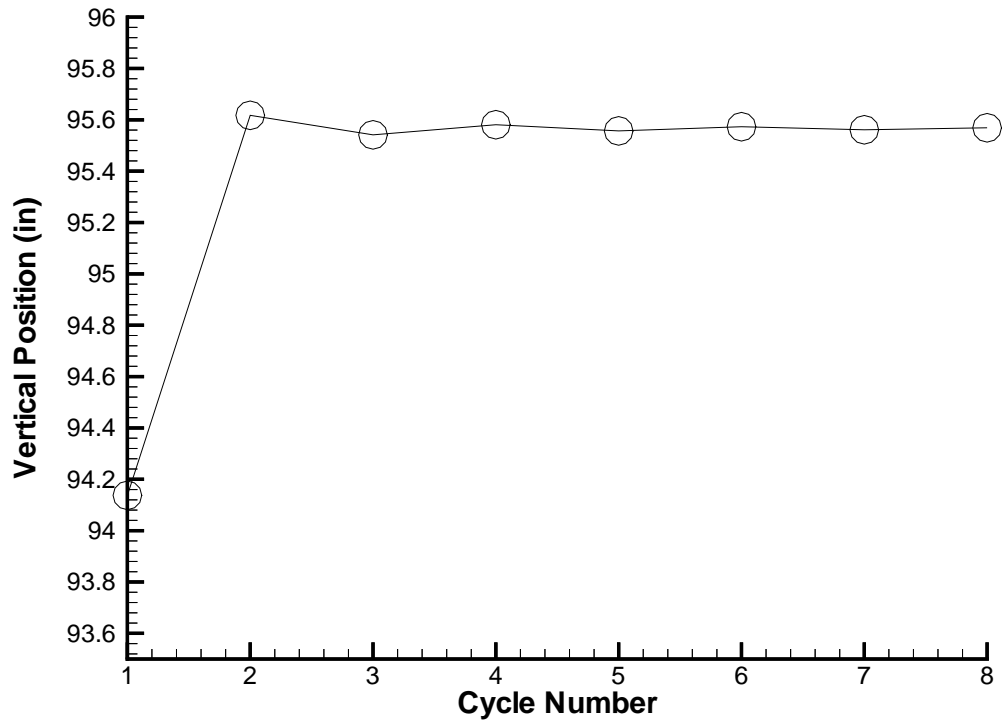
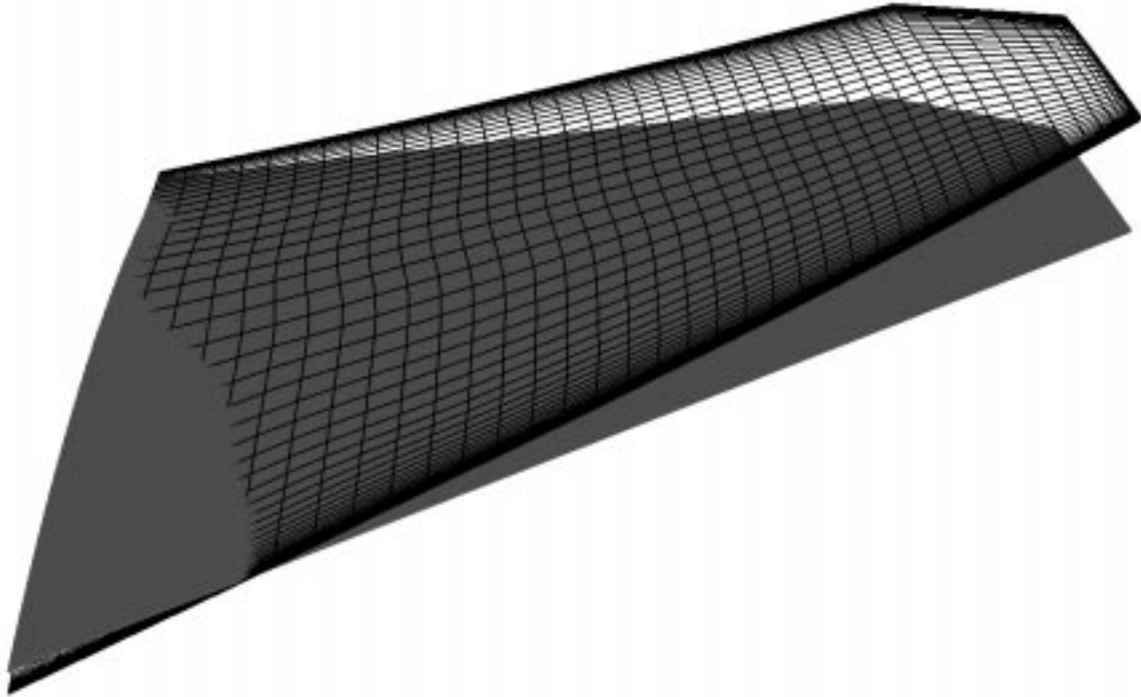


Figure 4.2: Convergence of the Trailing Edge Tip of the F/A-18 Stabilator



Deflections scaled by a factor of 10

Figure 4.3: Final Converged and Initial Undeformed F/A-18 Stabilator

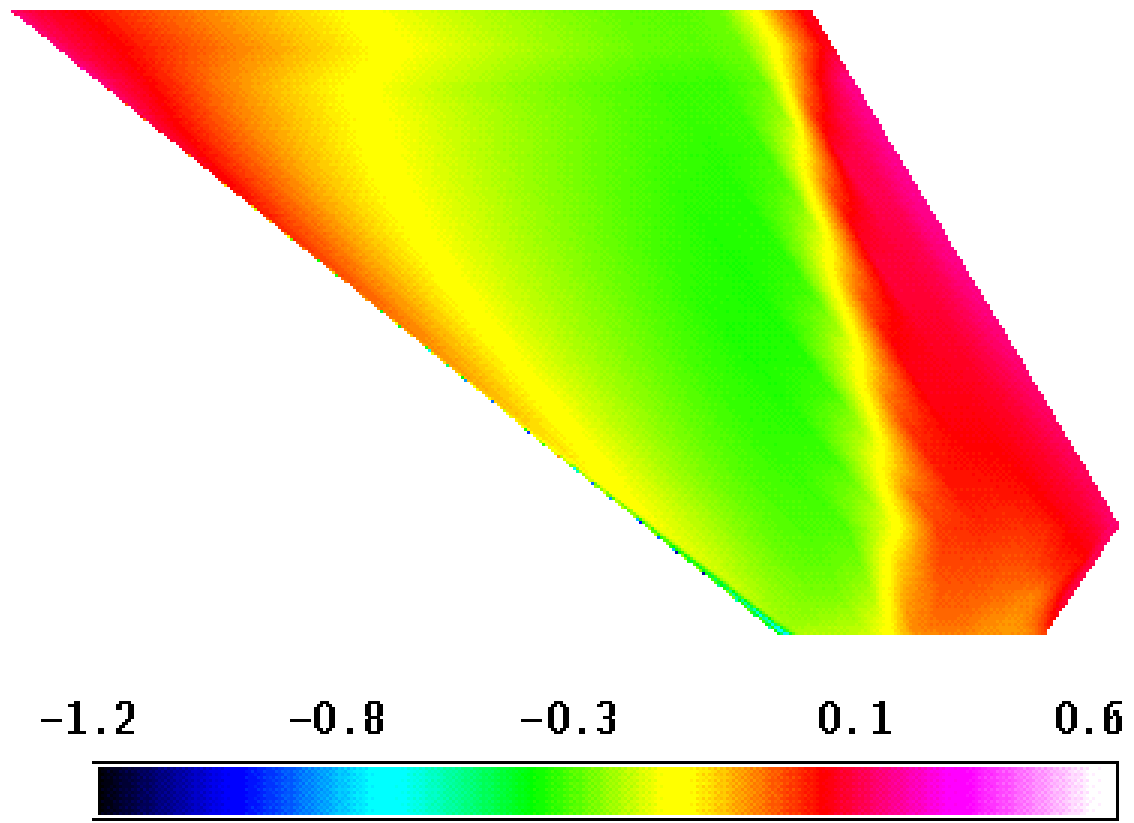


Figure 4.4:  $C_p$  Variation on the Upper Surface of the Rigid F/A-18 Stabilator

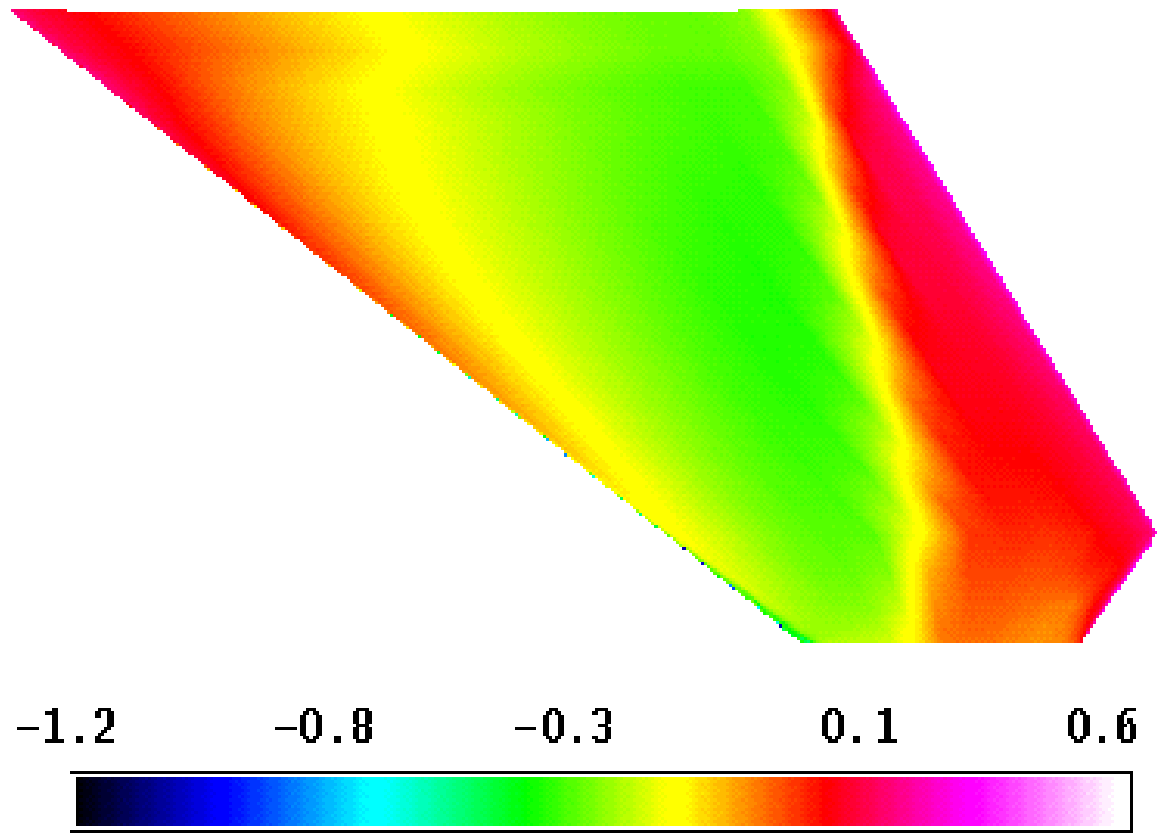


Figure 4.5:  $C_p$  Variation on the Upper Surface of the Flexible F/A-18 Stabilator

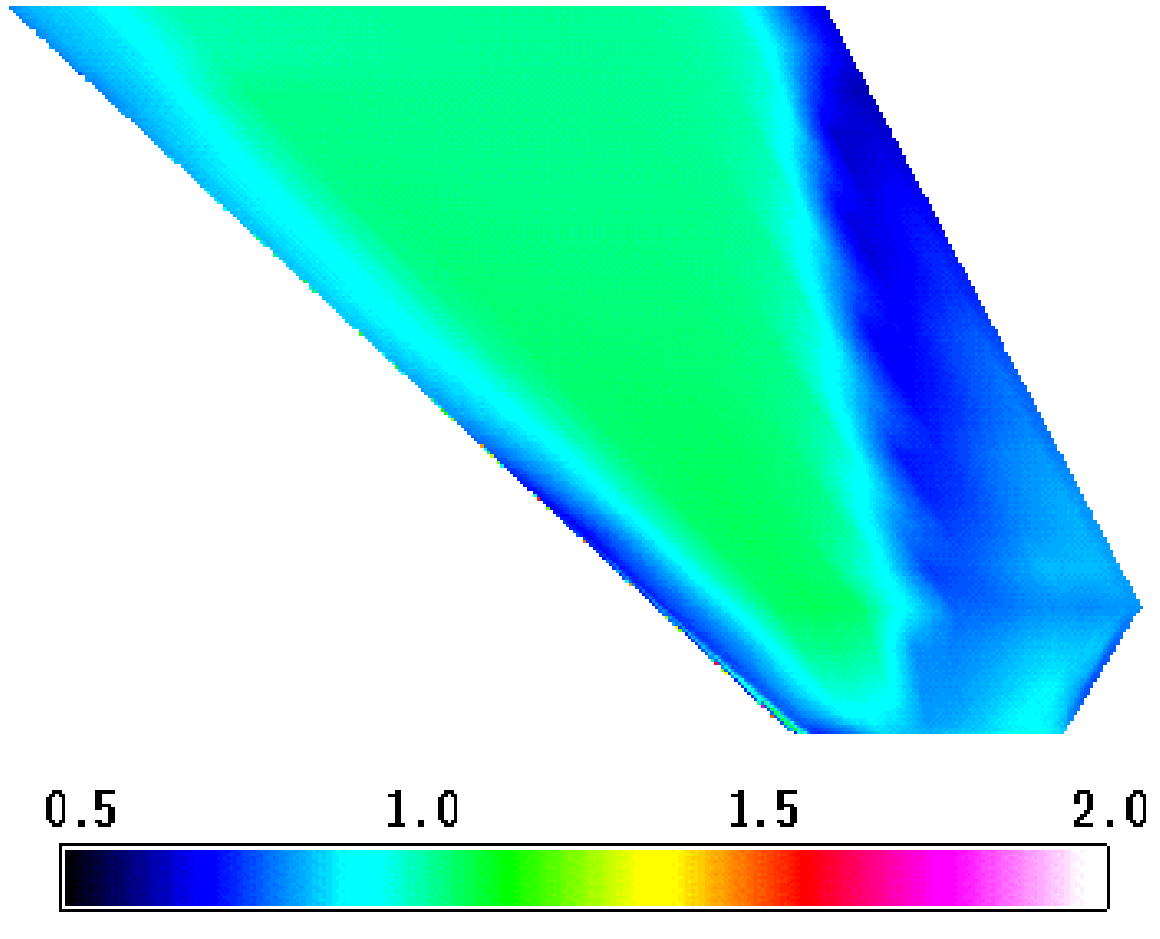


Figure 4.6: Mach Number Variation on the Upper Surface of the Rigid F/A-18 Stabilator

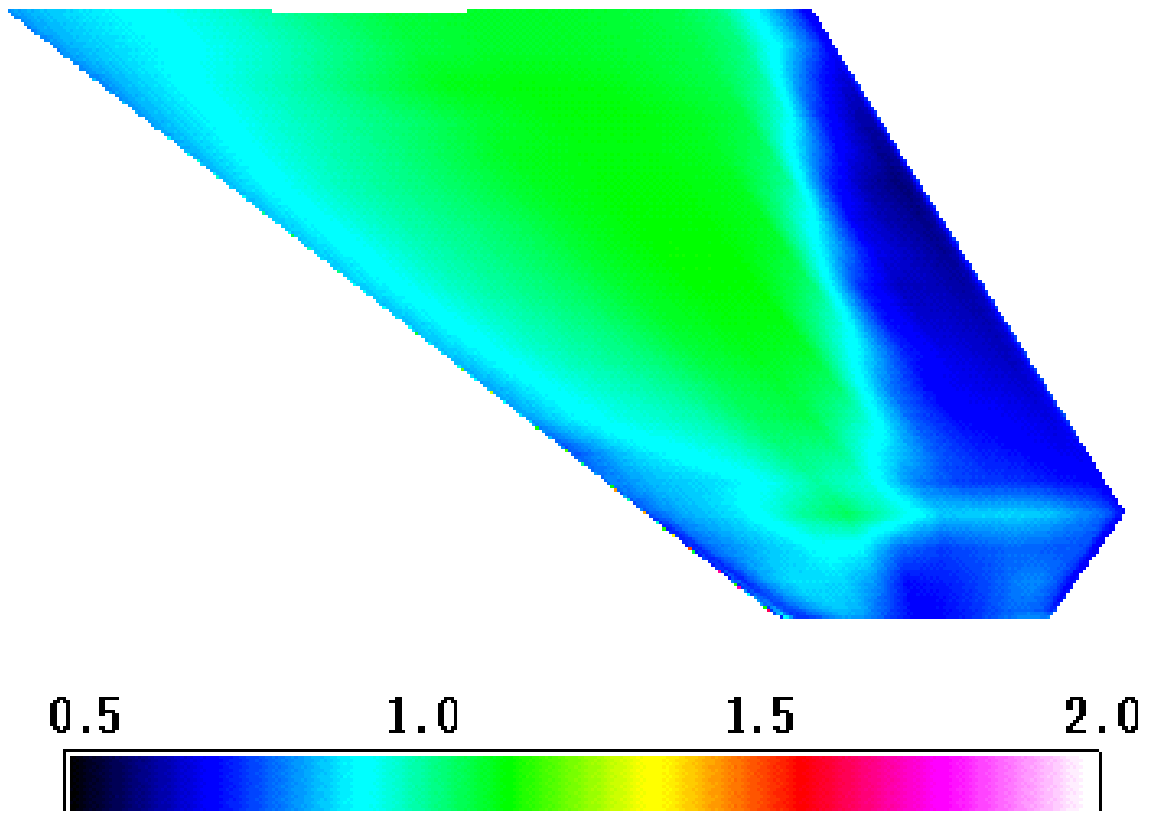


Figure 4.7: Mach Number Variation on the Upper Surface of the Flexible F/A-18 Stabilator

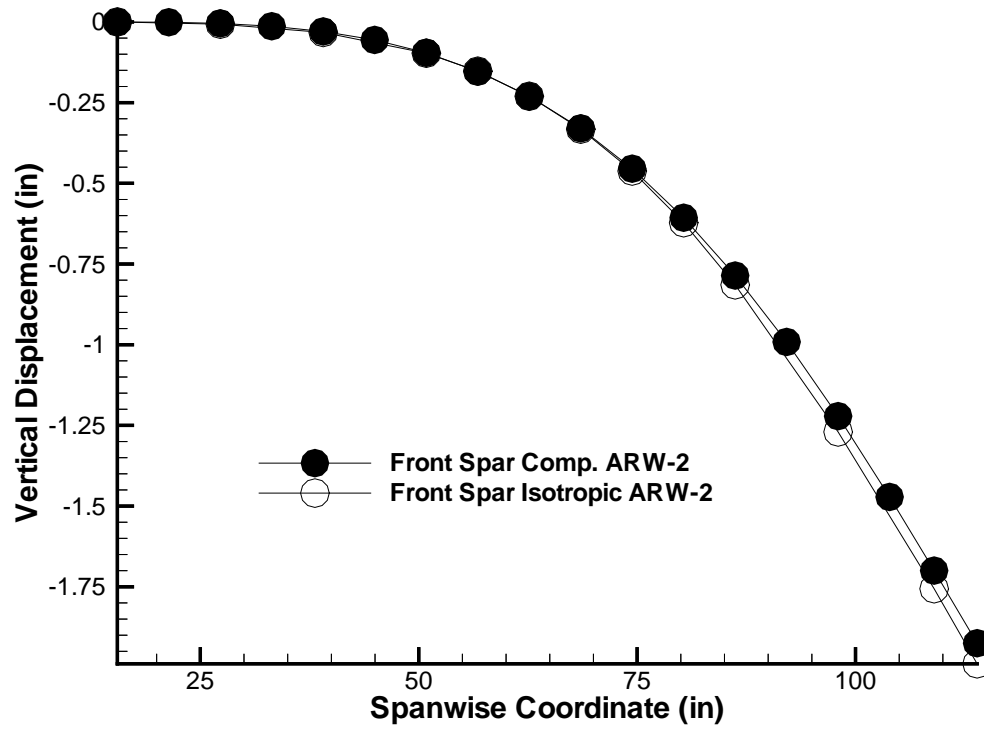


Figure 4.8: Displacement of the Front Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a 100 lb Vertical Load Applied at the Tip

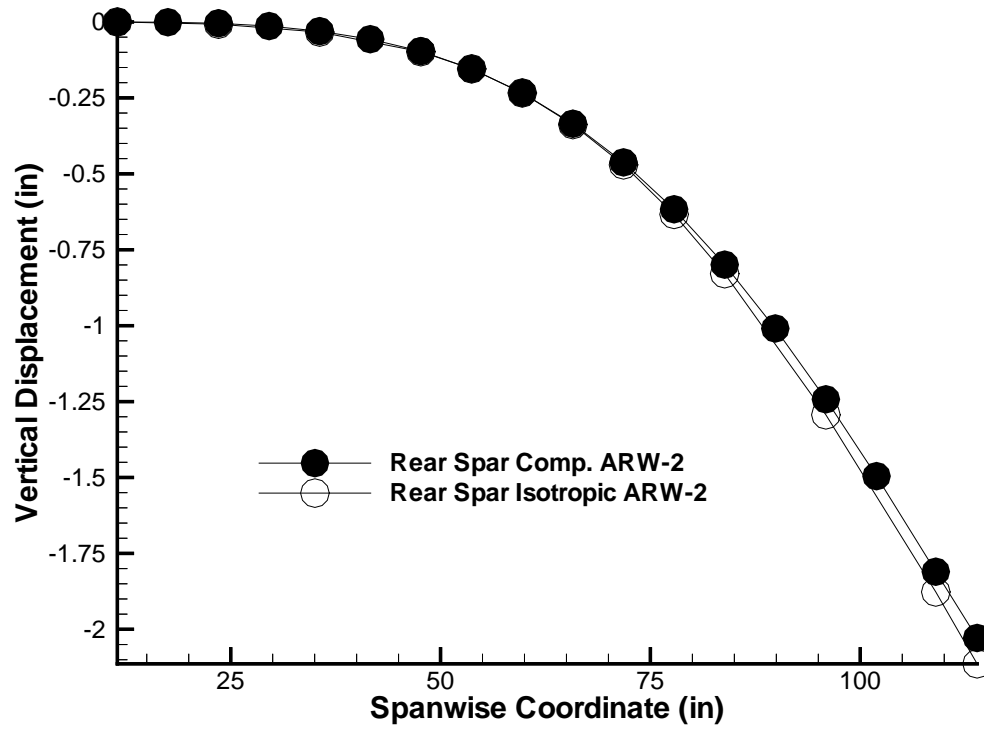


Figure 4.9: Displacement of the Rear Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a 100 lb Vertical Load Applied at the Tip

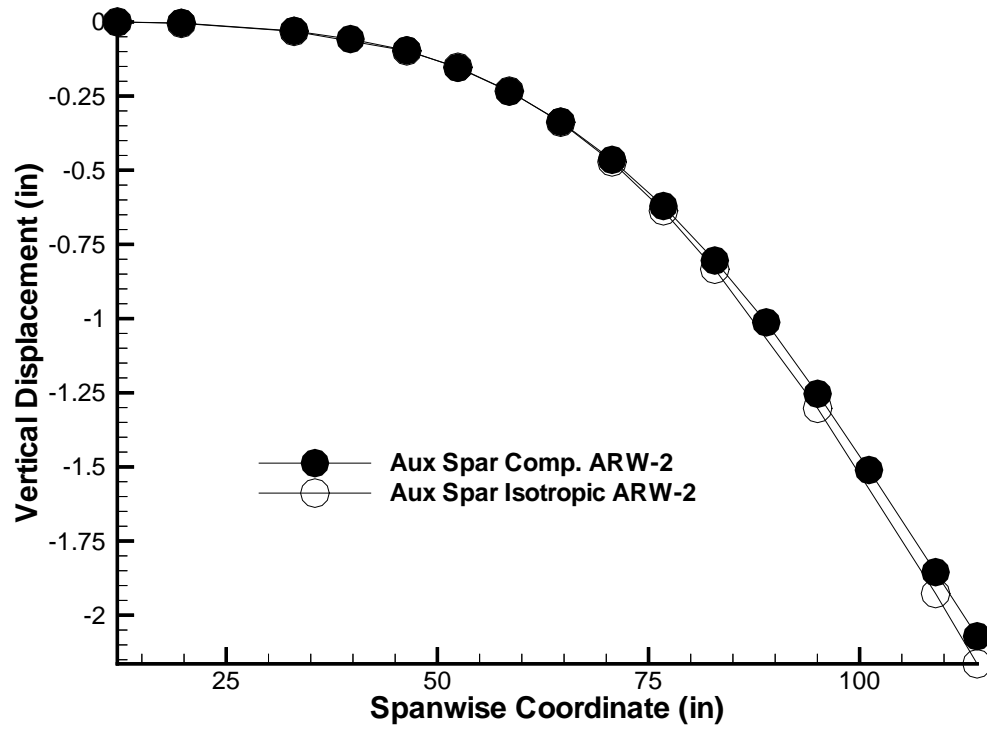


Figure 4.10: Displacement of the Auxiliary Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a 100 lb Vertical Load Applied at the Tip

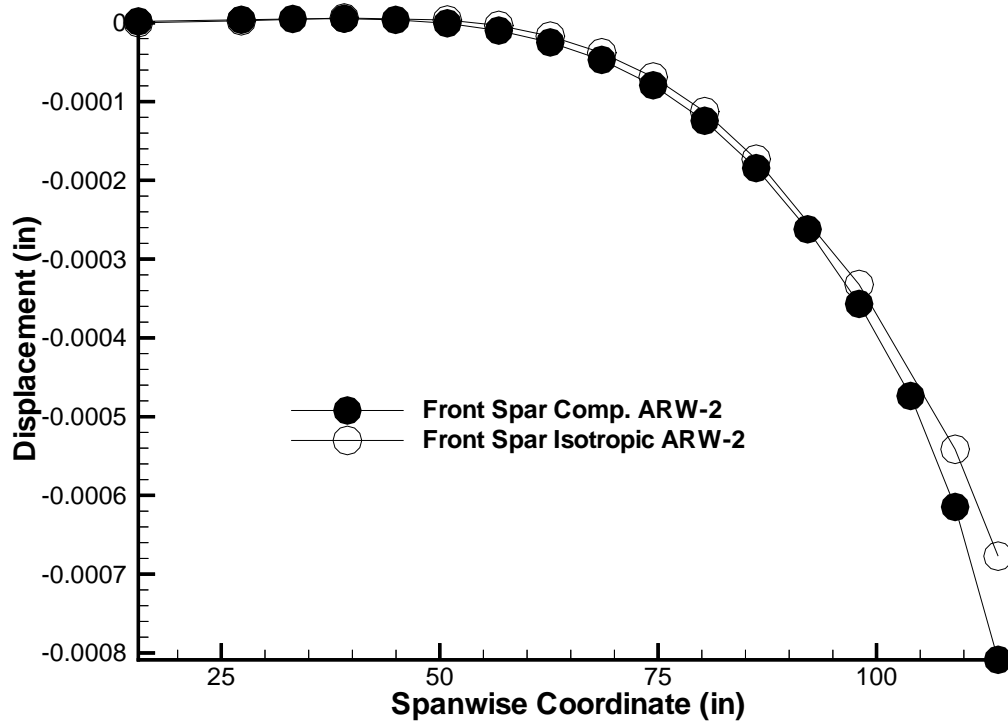


Figure 4.11: Displacement of the Front Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a Twisting Load Applied at the Tip

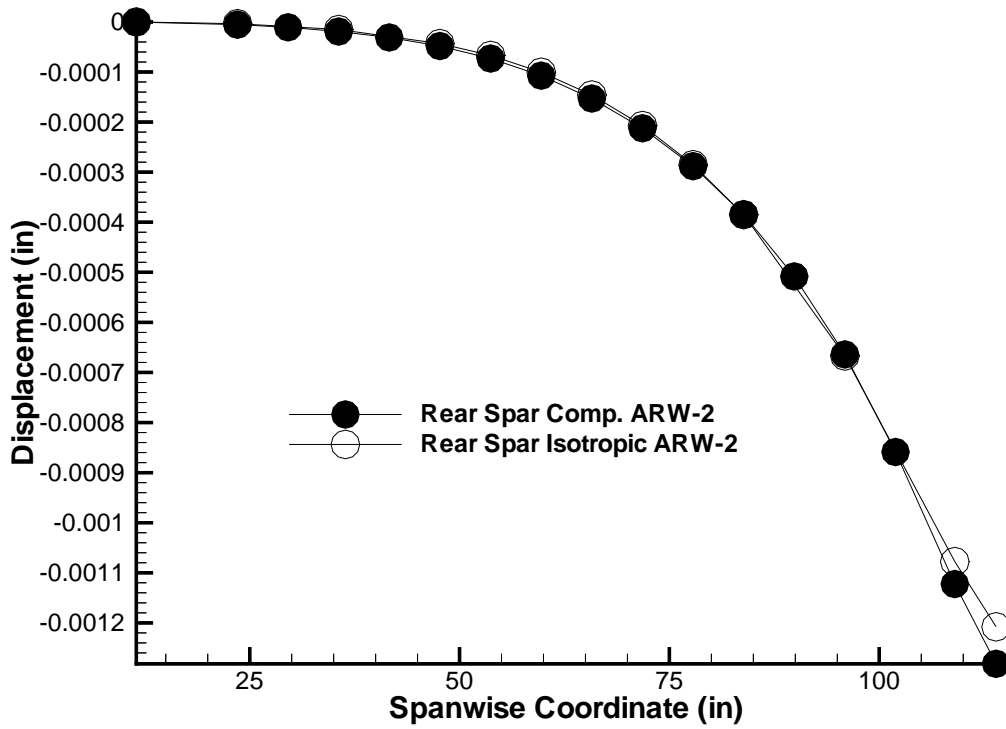


Figure 4.12: Displacement of the Rear Spar of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a Twisting Load Applied at the Tip

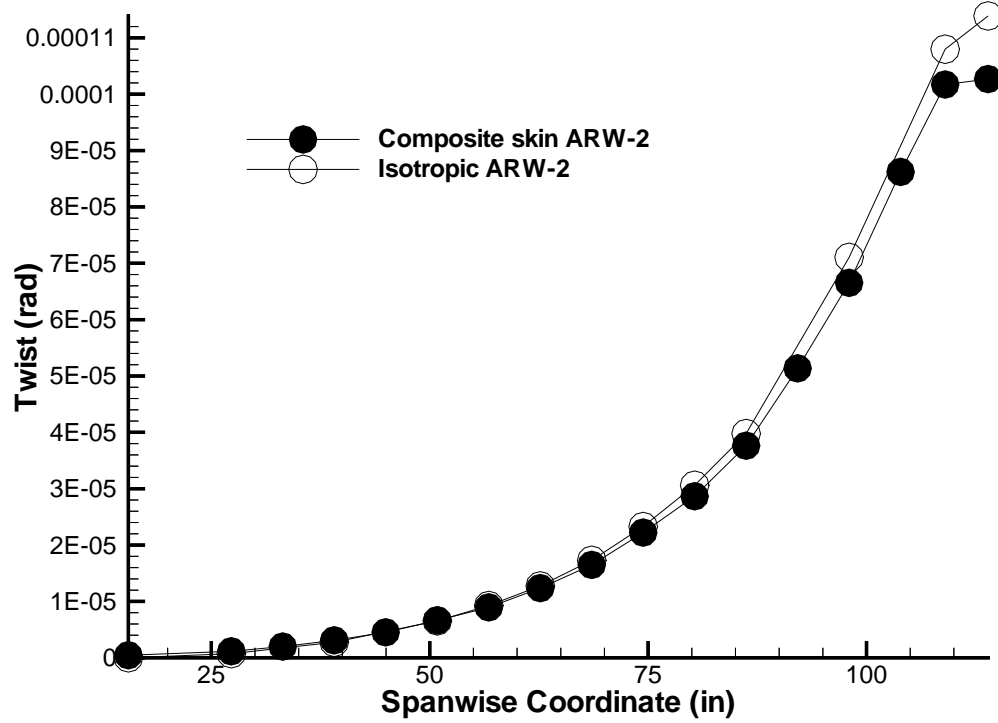


Figure 4.13: Twisting of the Composite Skin and the Isotropic ARW-2 Wing Subjected to a Twisting Load Applied at the Tip

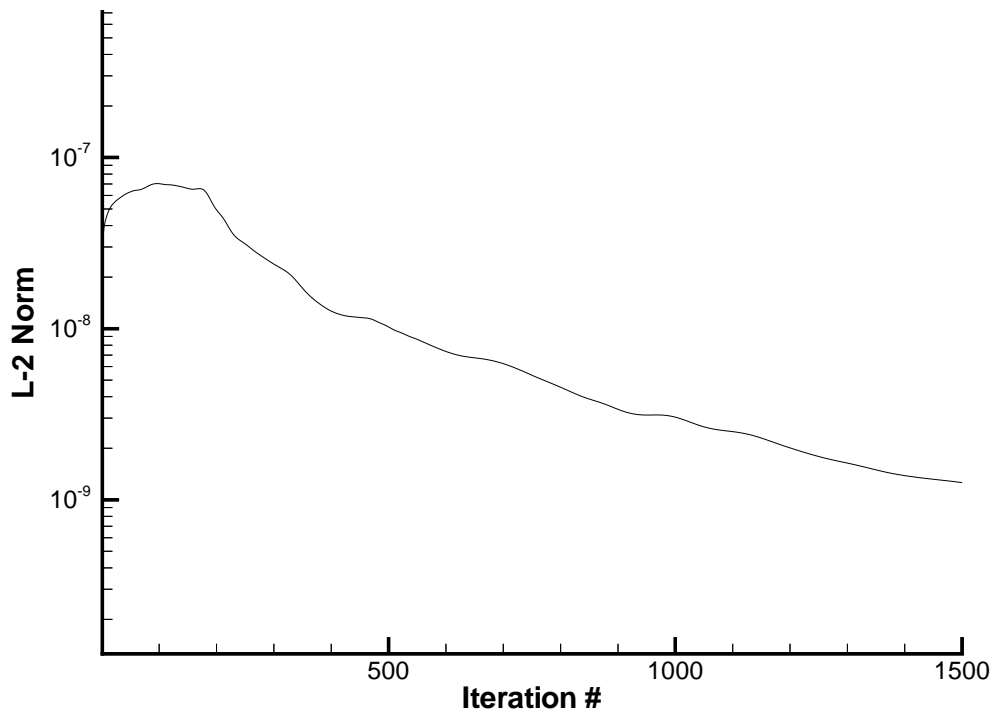


Figure 4.14:  $L_2$  Norm of the Residual of the Navier-Stokes Equations for the Rigid Steady State Solution at  $\alpha = 1$  deg

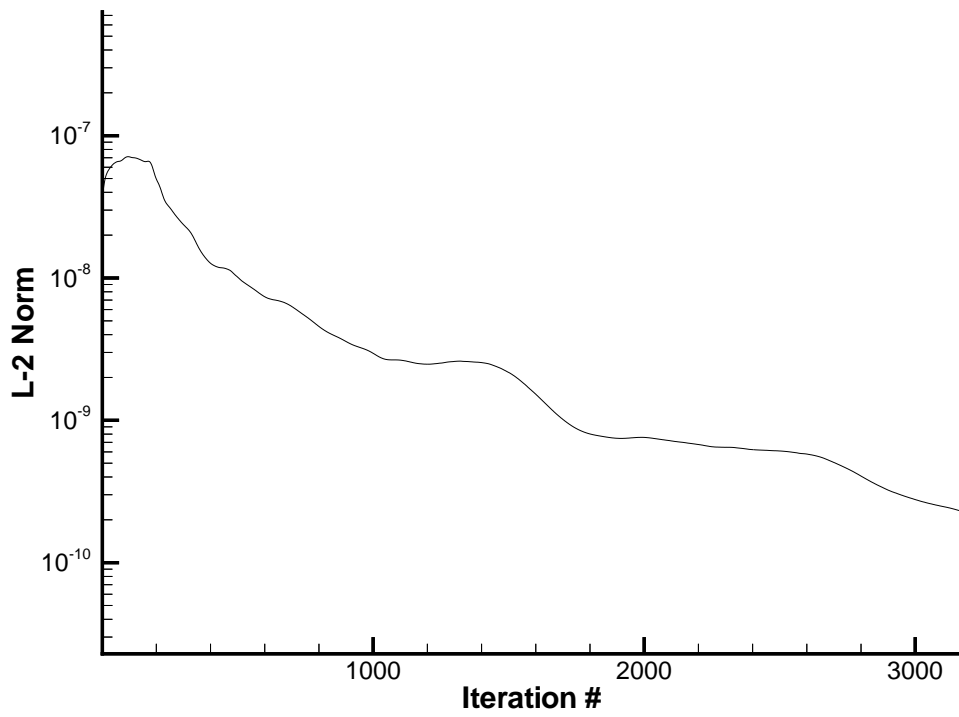


Figure 4.15:  $L_2$  Norm of the Residual of the Navier-Stokes Equations for the Rigid Steady State Solution at  $\alpha = 2$  deg

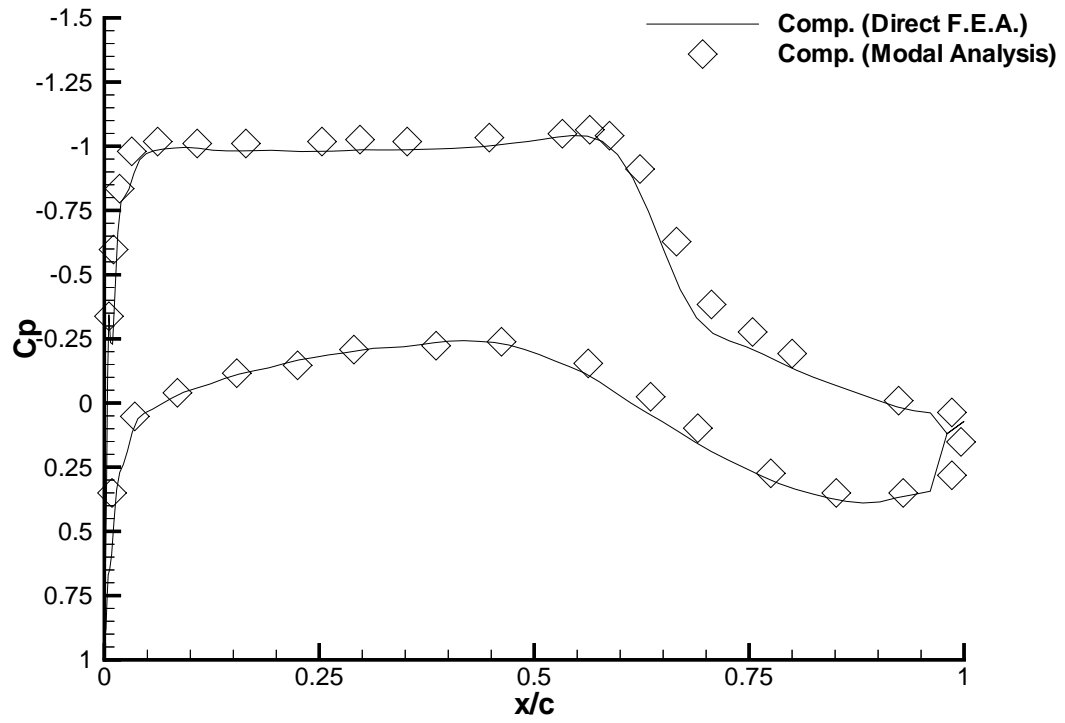


Figure 4.16: Comparison of  $C_p$  Variation for Rigid Steady State Solution at the 70.7% Semi-span Location for  $\alpha = 1$  deg

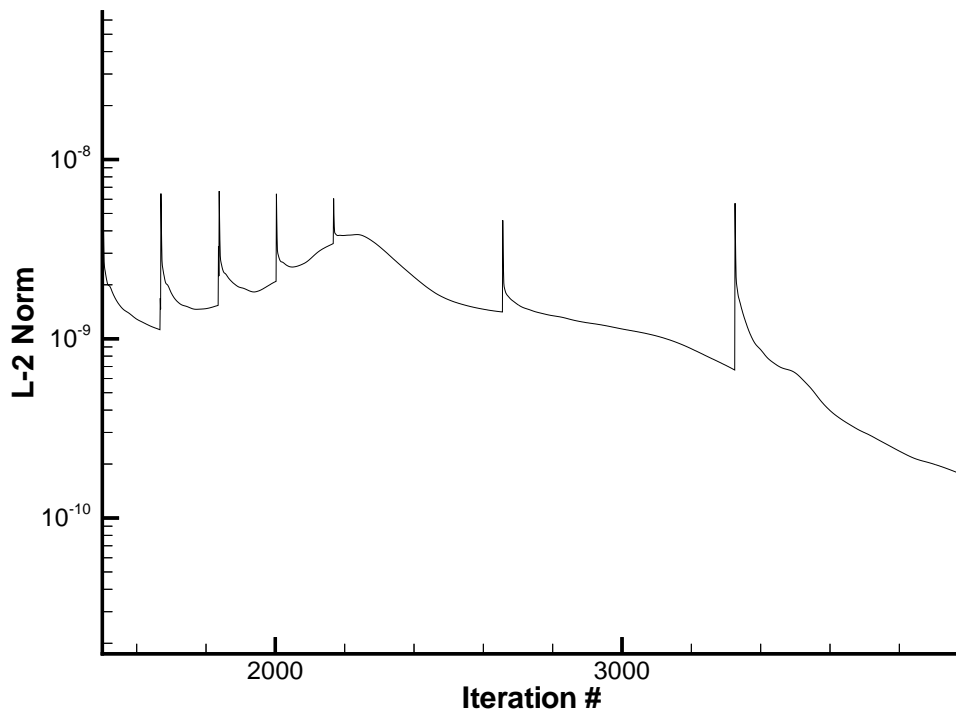


Figure 4.17:  $L_2$  Norm of the Residual of the Navier-Stokes Equations for the Flexible Steady State Solution at  $\alpha = 1$  deg

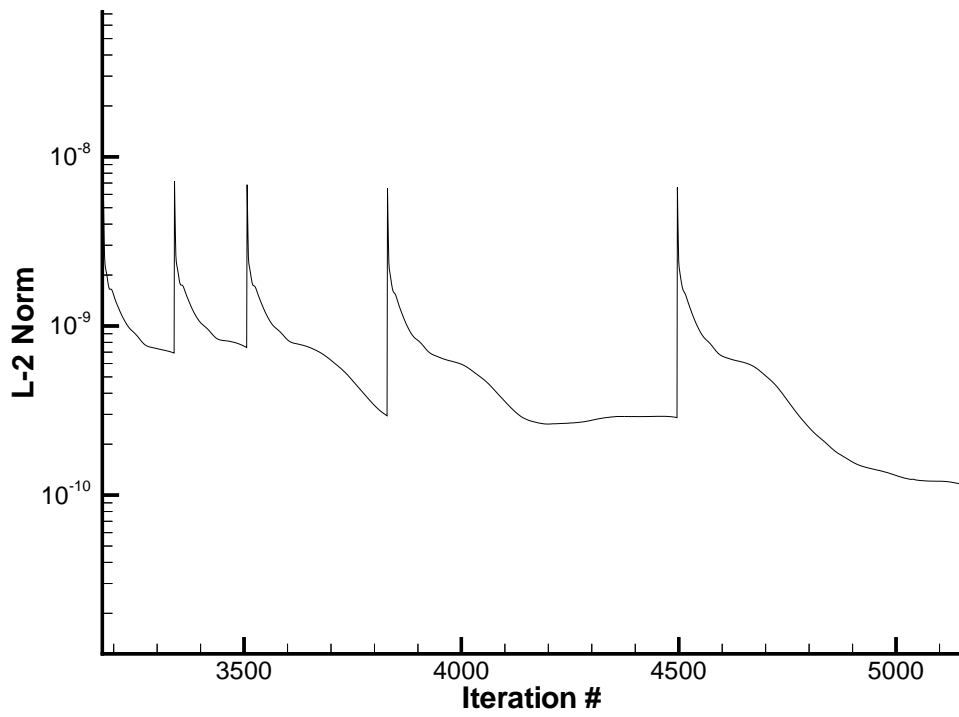


Figure 4.18:  $L_2$  Norm of the Residual of the Navier-Stokes Equations for the Flexible Steady State Solution at  $\alpha = 2$  deg

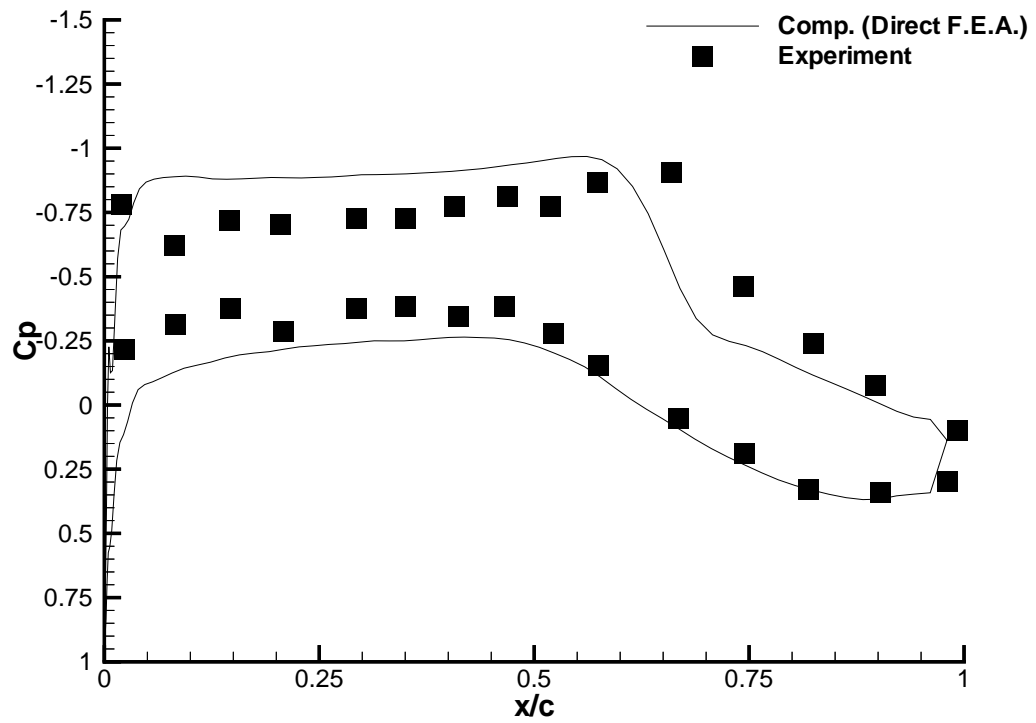


Figure 4.19: Comparison of  $C_p$  Variation of Experimental Data Versus Computational Results at the 70.7% Semi-span Location for  $\alpha = 1$  deg

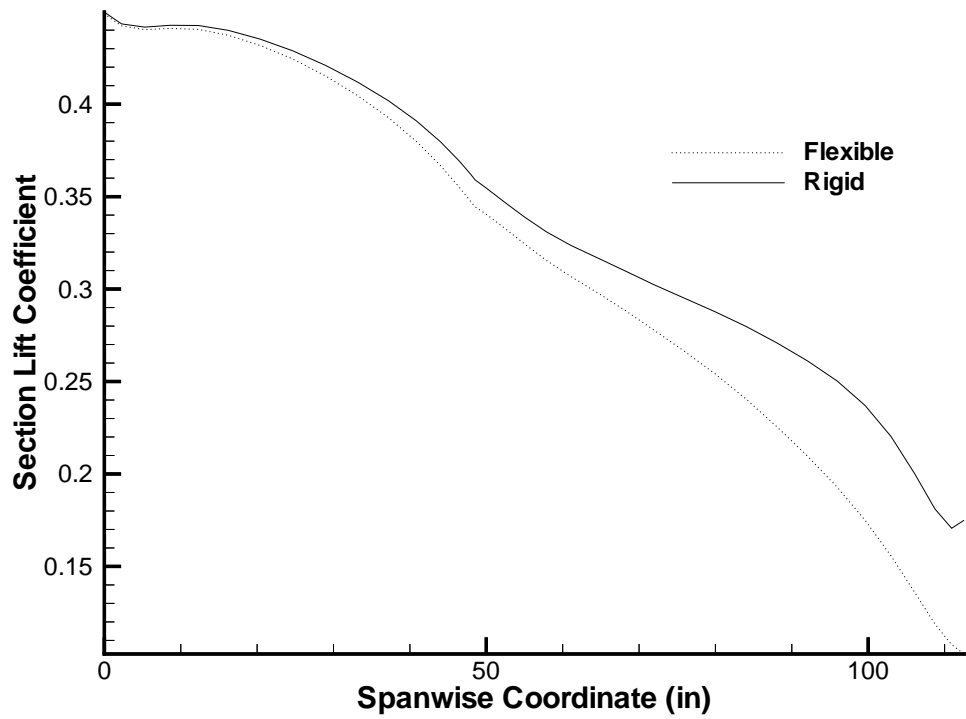


Figure 4.20: Section Lift Coefficient Variation Along Span for  $\alpha = 1$  deg

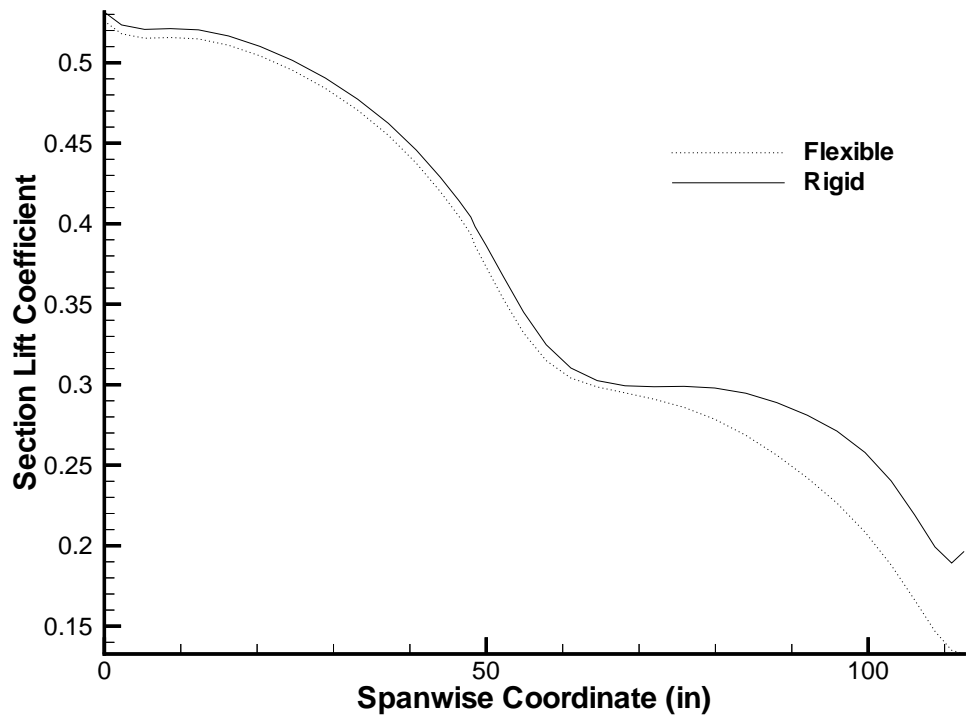


Figure 4.21: Section Lift Coefficient Variation Along Span for  $\alpha = 2$  deg

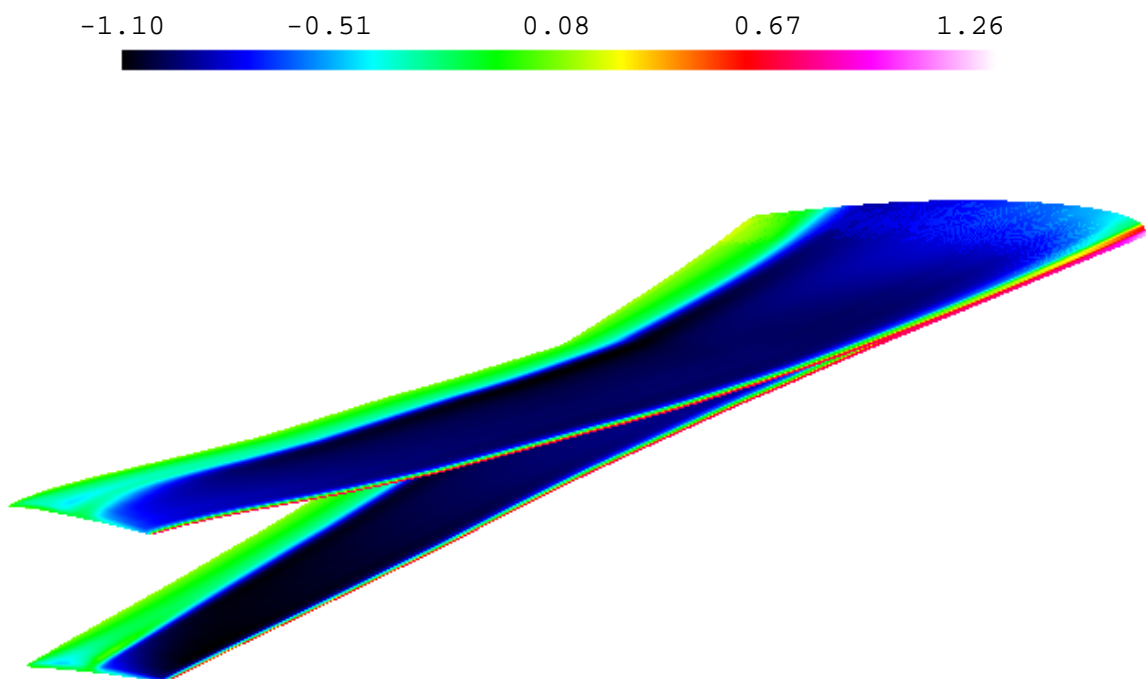


Figure 4.22:  $C_p$  Variation on the Upper Surface of the Rigid and Flexible ARW-2 Wing, at  $\alpha = 1$  deg

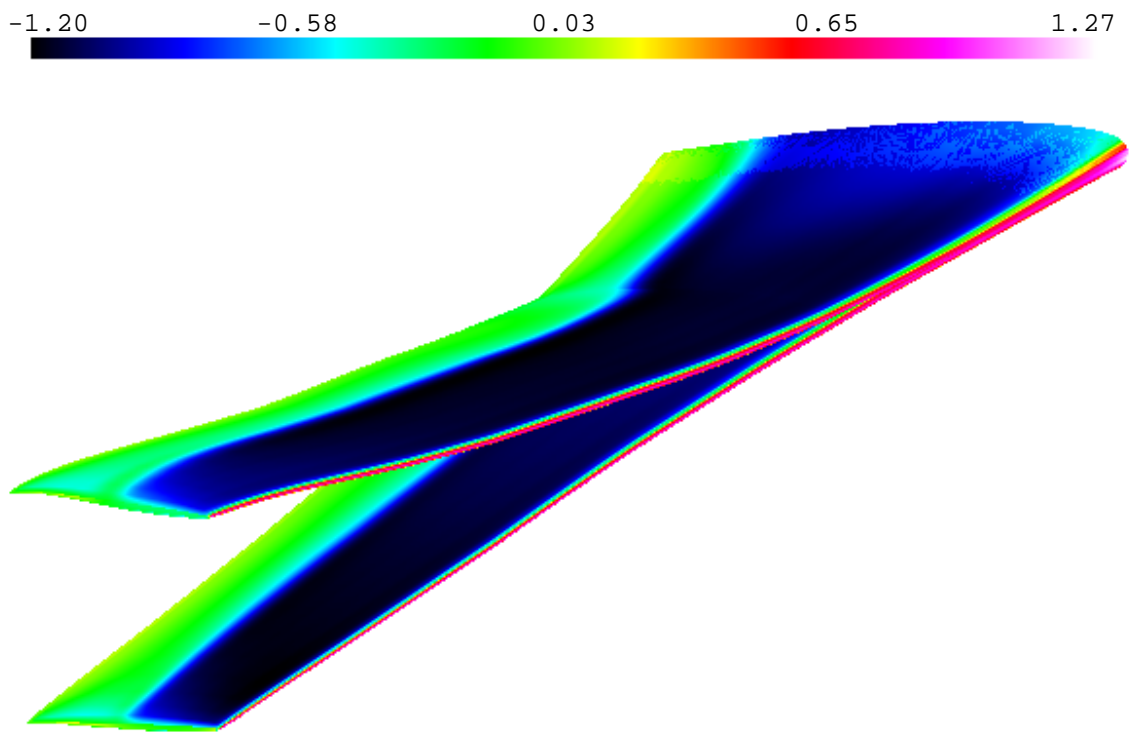


Figure 4.23:  $C_p$  Variation on the Upper Surface of the Rigid and Flexible ARW-2 Wing, at  $\alpha = 2$  deg

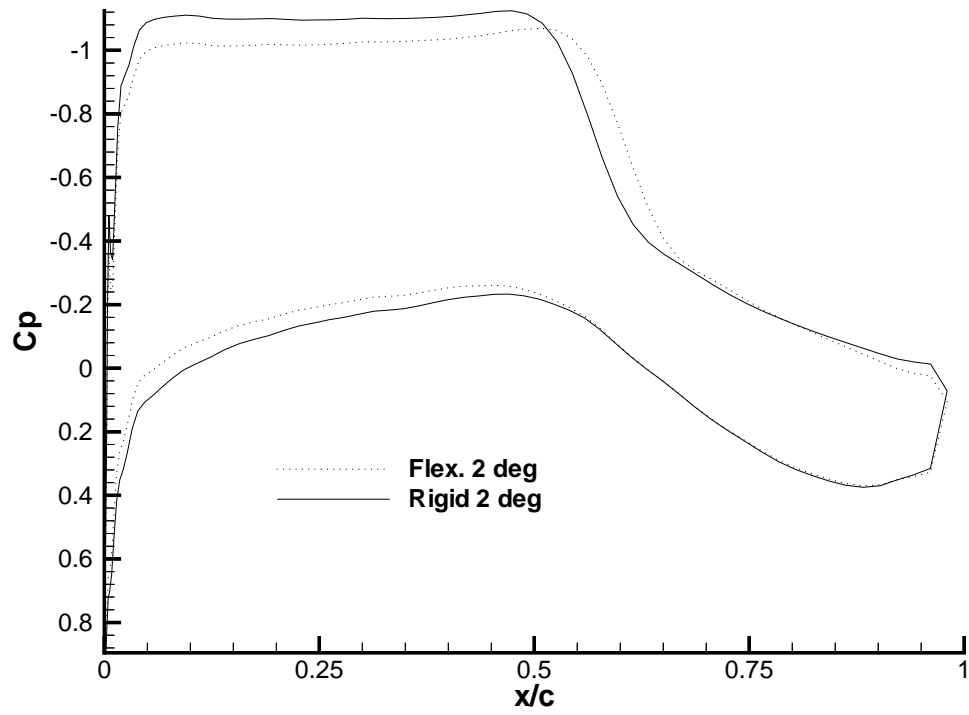


Figure 4.24:  $C_p$  Variation for  $\alpha = 2$  deg at the 70.7% Semi-span Location

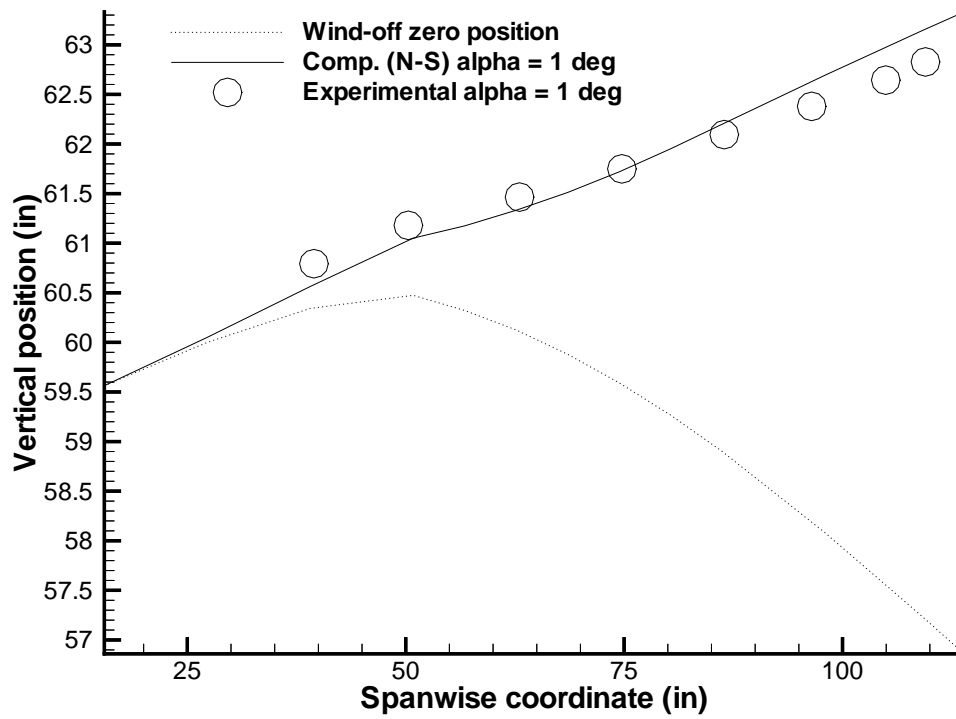


Figure 4.25: Comparison of the Experimental and Computational Front Spar Deflections of the ARW-2 Wing at  $\alpha = 1$  deg

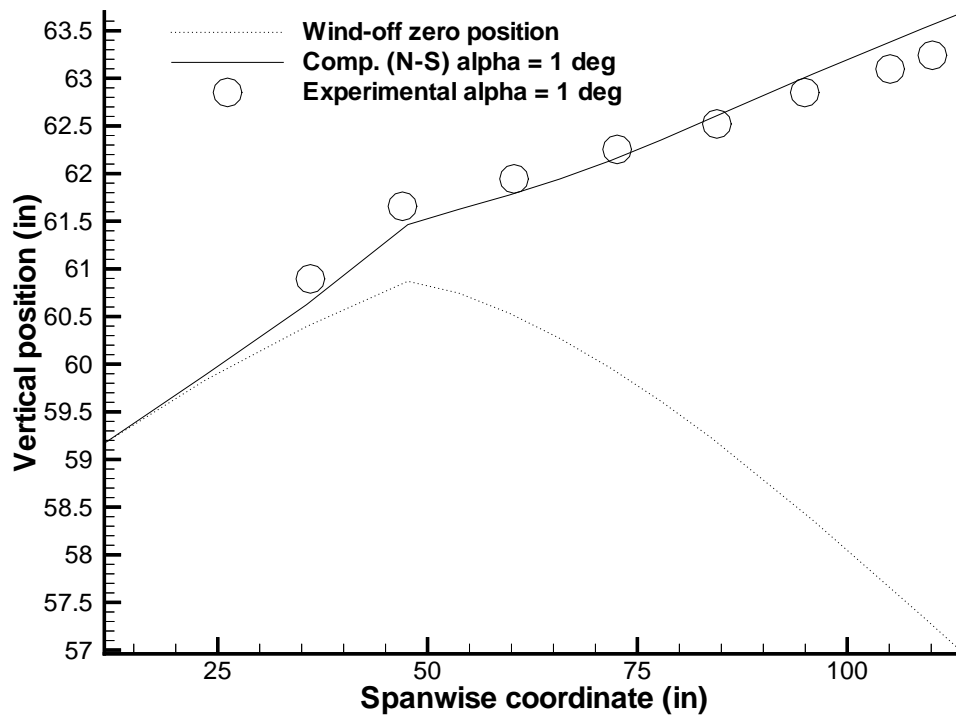


Figure 4.26: Comparison of the Experimental and Computational Rear Spar Deflections of the ARW-2 Wing at  $\alpha = 1$  deg

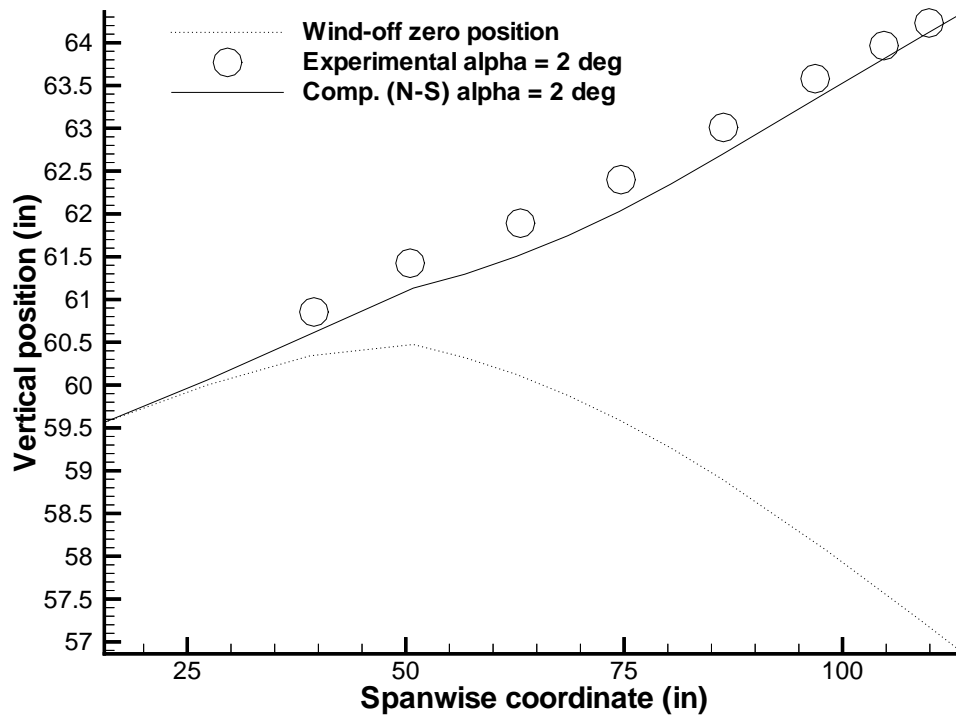


Figure 4.27: Comparison of the Experimental and Computational Front Spar Deflections of the ARW-2 Wing at  $\alpha = 2$  deg

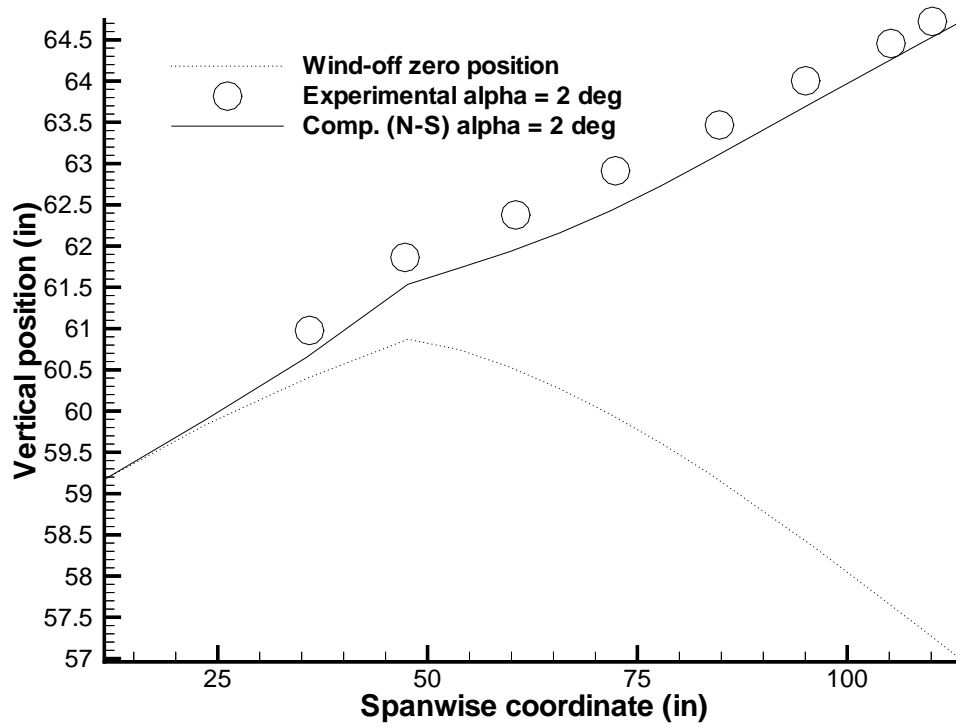


Figure 4.28: Comparison of the Experimental and Computational Rear Spar Deflections of the ARW-2 Wing at  $\alpha = 2$  deg

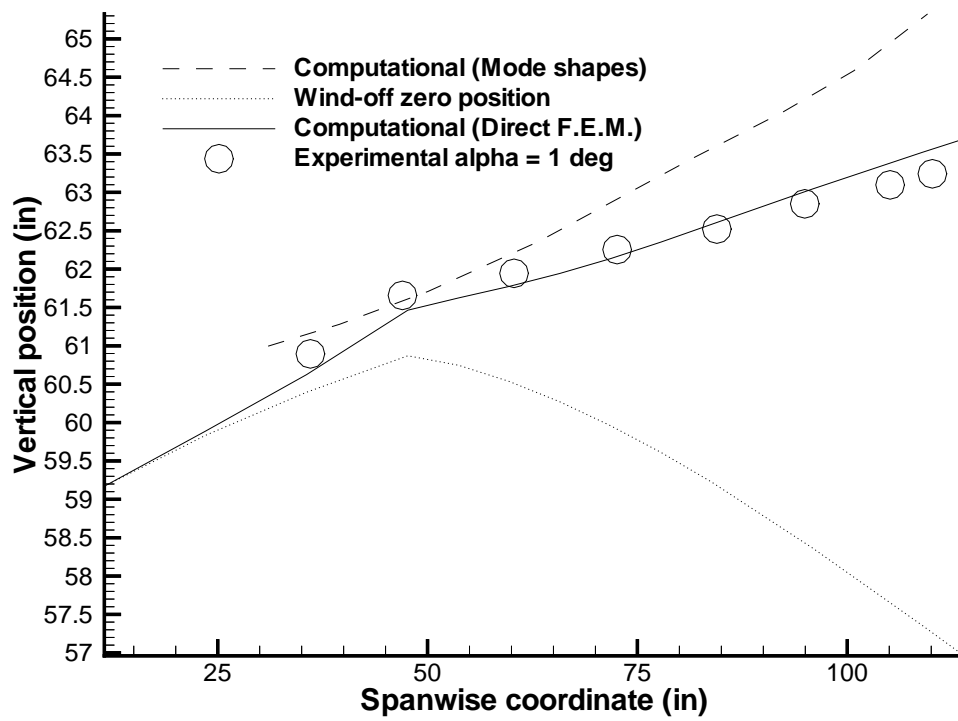


Figure 4.29: Comparison of the Rear Spar Deflections Using Modal Analysis Versus Finite Element Analysis of the ARW-2 Wing at  $\alpha = 1$  deg

# Chapter 5

## Parallel Aeroelastic Analysis

Due to the large number of floating point operations involved with CFD/CSD coupling, a parallel computing approach to aeroelastic analysis is investigated. An advanced parallel CFD code, ENSAERO, is coupled with a parallel wing-box model which uses Allman's triangular element [56] in conjunction with axial bars for static aeroelastic analysis. This study uses a wing-box model which calculates structural properties in parallel on the iPSC/860 (discussed in later section). The structural code incorporates a direct solver which calculates the solution to the structural equations of motion in parallel. Also, the structural and fluids models are run concurrently on the iPSC/860 on separate cubes and through the use of intercubes communication, the two separate disciplines are directly coupled to solve aeroelastic problems.

### 5.1 Governing Aerodynamic Equations

The strong conservation law form of the Euler equations is used for shock capturing purposes. The Euler equations in generalized coordinates can be written as [57]

$$\partial_\tau \hat{Q} + \partial_\xi \hat{E} + \partial_\eta \hat{F} + \partial_\zeta \hat{G} = 0 \quad (5.1)$$

where  $\hat{Q}$ ,  $\hat{E}$ ,  $\hat{F}$ , and  $\hat{G}$  are flux vectors in the generalized coordinates. To solve Eqn. 5.1, ENSAERO has time-accurate methods based on central-difference schemes. In

this work, the central-difference scheme used is based on the implicit approximate factorization algorithm of Beam and Warming [58] with modifications for diagonalization suggested by Pulliam and Chaussee [59].

## 5.2 Aeroelastic Equations of Motion

The governing equations of motion for structures can be written as

$$[M]\{\ddot{q}\} + [C]\{\dot{q}\} + [K]\{q\} = \{Z\} \quad (5.2)$$

where  $[M]$ ,  $[C]$ , and  $[K]$  are the global mass, damping, and stiffness matrices, respectively;  $\{Z\}$  is the aerodynamic force vector corresponding to finite element degrees of freedom. For a given time  $t$ ,  $\{Z\}$  is computed by solving the fluid equations.

### 5.2.1 Wing-box Model

Wing-box modeling discretizes the wing into spars, ribs, and skins to represent the structure of the wing. Allman's triangular element (Fig. 5.1), with three nodes at the vertices, is used as a part of the wing-box model to represent the skins, spars, and ribs. Each node has three degrees of freedom including two in-plane displacements and an in-plane rotation. The element can represent all constant strain states exactly, assuring convergence to an exact solution with consistent mesh refinement. The element is more accurate for stress analysis than the constant strain triangle. Axial bars are used in conjunction with Allman's element to formulate the wing-box model used in this study to represent the spar caps.

The structural system of equations is solved by the  $LDL^t$  method, parallelized by Farhat et al. (Ref. [60]). The solver assumes that the stiffness matrix is stored in a skyline fashion, to reduce the storage requirements. The solver decomposes the stiffness matrix by columns among the processors. The processors communicate to perform an  $LDL^t$  factorization. The solution of the system of equations is obtained by a forward and backward substitution in parallel.

### 5.3 Parallelization of the Aeroelastic Equations

A domain decomposition or loosely coupled approach enables solution methods for fluid and structural equations to be developed independently. Fluid and structural equations are modeled in separate computational domains. Each domain is mapped individually onto a group of processors, referred to as a cube on the Intel iPSC/860. The iPSC/860 allows  $2^n$  processors to be used per code, termed a cube. The structures code runs on  $2^n$  processors and the fluids code runs on  $2^n$  processors. The codes are independent; however, coupling of the disciplines requires the exchange of the interface boundary data. When this exchange of data is performed using parallel iteration, idle time is reduced as opposed to using a serial iteration (Fig. 5.2). This is due to the fact that both sides have idle time when running sequentially, while only one side has idle time during parallel iteration. This exchange between the fluid and structural domains is accomplished through an intercubes communication mechanism. This intercubes communication facility enables different processors in each cube to communicate directly on the iPSC/860.

The Euler equations for fluids domain are solved by using 3-D uni-partitioning of the computational domain. The uni-partitioning scheme denotes that one grid subdomain is assigned to each of the processors. The arrangement of the processors, described in Fig. 5.3, show how the grid is decomposed. The arrows show how the processors, representing the various grid sub-domains, communicate by exchanging boundary information. The arrows denote bi-directional data communication. There are a variety of concurrent algorithms available for solving the system of equations for fluids. Pipelined Gaussian Elimination [61] was chosen for this study.

For the structural domain, each node or processor assembles the portion of the stiffness matrix required by the parallel solver. Therefore, there is no need of actually assembling the global stiffness matrix on any one node and passing it to the remaining nodes. This method involves redundant calculations, but is quicker than having global communications, especially as the system of equations becomes large. This can also help alleviate memory problems that might occur due to large storage requirements. In other words, the parallel solver will break down the stiffness matrix by columns.

If the processors are numbered from 0 to  $np - 1$ , where  $np$  is the total number of processors, then process  $n$  will only require column  $n + 1, (n + 1) + np, (n + 1) + 2 * np...$  and so on. Therefore, instead of having the solver break down the stiffness matrix, only the portion required by that particular processor is calculated and stored. So, if the matrix size is  $m \times m$ , and there are  $np$  processors, each processor will only have to store  $m \times (m/np)$  matrix. And as stated earlier, this can help alleviate memory problems with analysis of large scale finite element problems.

The structures and fluids codes only communicate at rendezvous points to exchange vital information. The structures code calculates the deflections on the structural grid and using the deflections, calculates a leading edge displacement and angle of twist. The fluids code uses the displacements and twist angles to deform the aerodynamic grid.

The forces are first calculated at the grid points of the aerodynamic grid which must then be transferred to the structural nodes. This is done in the structures code, by finding the structural element that contains the aerodynamic grid point. Using the area coordinates of the CFD grid point within the element, the loads are transferred to the structural nodes. A problem arises when the aerodynamic point lies outside the structural grid. Currently, the load on an exterior aerodynamic point is transferred to the closest structural node. Moment and twist produced by this transfer is neglected, but the effect is negligible as long as the planforms of the CFD and CSD models are the same. It is suggested that extrapolation be avoided by properly modeling the structural finite element model to match planforms with the CFD wing grid.

## 5.4 Structural Analysis

Details of the validation of the finite element wing-box code are presented in the next few sections. A square panel, a cantilever beam, and a box beam were subjected to various loads, and the structural responses were obtained. The results are compared with available analytical and published data.

### 5.4.1 A Square Panel

To validate the finite element wing-box code, the response of an isotropic square panel of length  $L$ , (taken from Ref. [56]) under a linearly varying edge normal stress of amplitude  $\sigma_o$  given by

$$\sigma_x|_{x=\pm(L/2)} = 2\left(\frac{y}{L}\right)\sigma_o \quad (5.3)$$

is obtained. Poisson's ratio is 0.3. Due to symmetry, only  $\frac{1}{4}$  of the panel is discretized, as shown in Fig. 5.4, using a 4 x 4 mesh, a total of 32 elements. The antisymmetric boundary conditions for the corner of the panel are  $u=0$  along AB and the symmetric boundary conditions are  $u=0$  along line AD. To suppress the zero displacement mode,  $\omega_i$  is set to zero at corner A.

### 5.4.2 A Cantilevered Beam

Another example is used to validate the finite element wing-box code. A cantilevered beam (Fig. 5.5) of length  $L=48$  inches, height  $H=12$  inches, and a thickness  $t=1$  inch, is modeled using a 5 x 17 mesh, a total of 128 elements, 255 degrees of freedom. A tip load of  $W = 40,000$  lbs is applied as a parabolic shear stress, taken from Ref. [56],

$$\tau_{xy}|_{x=L} = \frac{3W}{2Ht} \left[1 - 4\left(\frac{y}{H}\right)^2\right] \quad (5.4)$$

where Young's modulus is 30,000 ksi; Poisson's ratio is 0.25. The beam is clamped at  $x=0$ , that is  $u, v$ , and  $\omega$  are zero at  $x=0$ . The consistent load vector is used to obtain the loads at the tip of the beam, at  $x=48$  inches.

### 5.4.3 Box Beam

Next, a more complex example is chosen for validation of the structural code. A box beam with axial rods and membrane elements is examined. It is discretized into 2 x 30 mesh, a total of 540 elements. The box beam has axial loads acting on the rods and also producing a bending moment, Fig. 5.6. The stresses in the axial bars are compared with those obtained analytically.

## 5.5 Aeroelastic Analysis

### 5.5.1 Aerodynamic Modeling

The CFD code uses an H-O type grid with grid dimensions of 95 (axial) x 79 (circumferential) x 32 (normal) points. The wing-body grid is shown in Fig. 5.7. The fuselage is assumed to be rigid and the wing is considered flexible. The wing surface grid (Fig. 5.8) has dimensions of 40 (chordwise) x 49 (spanwise). Though a typical wing-body configuration is selected for demonstration purposes, the code can be easily applied to more realistic configurations such as the High Speed Civil Transport (HSCT) type wing-body configuration.

One of the major difficulties in using the Euler equations for computational aerodynamics lies in the area of grid generation. For steady flows, advanced techniques such as blocked zonal grids [62] are currently being used. However, grid generation techniques for aeroelastic calculations which involve moving components are still in the early stages of development. In this work, the moving grid is generated using an algebraic scheme. The grid is designed so that flow phenomena such as shock waves, vortices, etc. and their movement on the wing-body configurations are accurately simulated. The grid is generated at every iteration based on the aeroelastically deformed position of the structure. Details of the implementation of this grid-generation technique on parallel computers is given in Ref. [42].

### 5.5.2 Structural Modeling

The wing is structurally discretized using Allman's triangular elements and axial bars. The code has the ability to discretize the wing into an  $m \times n$  mesh, where  $m$  is number of spars, and  $n$  is the number of ribs. The wing is discretized as to match the physical structure of the wing. The spars and ribs of a wing are represented by the combination of the AT elements with axial bars, while the skin is represented by the AT elements only.

### 5.5.3 Typical Wing-body Configuration

A typical wing-body configuration is chosen for parallel static aeroelastic analysis using both fluid and structural codes. The wing is bi-convex with thickness varying from 6% chord at the root to 4% chord at the tip. The wing is discretized into an  $8 \times 10$  mesh. The top view of the structurally discretized wing is shown in Fig. 5.9. Figure 5.10 shows the entire wing structure which includes the upper and lower surface skins, the spars, and ribs, but not the axial bars, which run spanwise on both surfaces. Figure 5.11 shows only the spars and ribs.

The wing-body configuration is examined at Mach 0.9 at  $\alpha = 1.0$  degree angle of attack; the free stream dynamic pressure  $q$  is 0.3 psi. The  $L_2$  norms of the residuals of the fluids equations are examined to validate solution convergence. The loads on the CFD and structural grids are examined to assure proper transfer of loads from fluids to structures. The loads also indicate the convergence of the CFD/CSD solution. The displacements along the span and tip are also examined.

Scalability of the structural code is also examined by increasing the number of processors. Although the results shown for the static aeroelastic analysis are performed on the iPSC/860, the parallel performance of the structures code is studied on the Intel Paragon. With computer technology increasing so rapidly, especially in the area of parallel computing, the IBM SP-2 was replaced the iPSC/860 at NASA Ames. Due to different message passing standards on the two machines, considerable time has to be spent before the fluids and structures codes will be IBM SP-2 ready. Therefore, performance of the CFD/CSD coupling on the Intel iPSC/860 is not available. However, the Intel Paragon was available to test the performance of the structures code.

## 5.6 Structural Analysis Results

### 5.6.1 A Square Panel

As stated earlier, to validate the finite element wing-box code, an isotropic square panel under pure bending loads (eqn. 5.3) is examined. Due to the symmetry of the

problem, only  $\frac{1}{4}$  of the panel is analyzed, as shown in Fig. 5.4. The bending stress at point C,  $\frac{\sigma_x}{\sigma_o}$ , is obtained and compared with exact solution. The exact solution is 1.000 and the computational results show  $\frac{\sigma_x}{\sigma_o}$  to be 0.998. The displacement in the x-direction at point C,  $\frac{Eu_c}{\sigma_o L}$  is 0.498 while the exact solution is 0.500. Good agreement is obtained.

### 5.6.2 Cantilevered Beam

Allman's triangular (AT) element is compared with the constant strain triangle (CST) [63] and the linear strain triangle (LST) [64,65], by using a cantilevered beam analysis. A cantilever beam with a parabolic shear tension (eqn. 5.4) is examined, and the solutions compared. The beam is modeled by a 5 x 17 mesh, a total of 128 elements, and 255 degrees of freedom, as seen in Fig. 5.12. The stress contours,  $\sigma_x$  are also plotted in Fig. 5.12. The comparison of AT, CST, and LST is summarized in table 5.1. The stress,  $\sigma_x$ , computed at x=12 in and y=6 in, using the finite element wing-box code, is 59.6 ksi. Using the constant strain triangle [63], the stress is calculated to be 53.5 ksi. Using the LST, the stress is calculated to be 60.0 psi. The AT element is accurate to 0.7% of the solution, while the CST element is almost 11% in error. The LST element produced excellent results, with negligible error.

The tip deflection,  $v$ , at x=48 in, y=0 in, is also compared using AT, CST, and LST elements, and the calculated deflections are 0.3471 in, 0.3115 in, and 0.3556 in, respectively. The AT element calculated the tip deflection to 2.4% error, while the CST element is 12.5% in error. The LST element produced good results again with negligible error.

The stress,  $\sigma_x$ , and tip deflection,  $v$ , obtained for comparisons in Olsen and Bear-den [66] are 60.0 ksi and 0.3558 in, respectively. The LST element is more accurate than the AT element, but the AT element provides good accuracy for the mesh shown in Fig. 5.12, for the degrees of freedom involved. So, for the 5 x 17 mesh used, the LST element mesh is a 594 degree of freedom model as compared to the AT mesh, which is only a 255 degree of freedom model. The stress and displacement results show the accuracy of the AT element.

### 5.6.3 Box Beam

A more complicated example is chosen to validate the finite element wing-box code. A box beam modeled by using a 2 x 30 mesh of AT elements, and axial bars is put under an axial loading at the tip of the box beam, therefore causing a bending moment. Results from the analysis of the box beam are shown in Fig. 5.15. Stresses in the three axial bars, A, B, and C, (Fig. 5.6) on the top surface of the box beam are plotted along with the analytical results. Even though only three curves can be seen, there are six curves plotted. It can be seen that the stresses match well with those predicted analytically in this shear lag example. It is noted again that the box beam is under axial loads acting on one side of the box beam, causing a moment. Accuracy of the AT element has been shown by the above examples using the finite element wing-box code. Next, the results of the aeroelastic analysis are shown on a typical wing-body configuration.

## 5.7 Aeroelastic Analysis Results

### 5.7.1 Typical Wing-body Configuration

A typical wing-body configuration is analyzed using Euler flow equations as available in ENSAERO, in conjunction with the finite element wing-box code. The deflections for the flexible wing are calculated in two ways. One method is to start impulsively from the free stream boundary conditions and the other is to start from the rigid steady state solution. Figure 5.13 shows the wing leading edge tip history versus iteration step for both cases. The total number of iterations required are about the same for the two methods when including the number of iterations required to determine the rigid steady state solution. If many calculations are needed, e.g. parametric studies, then using the rigid steady state solution will reduce the computational time required. It is also noted that both methods converge toward the same solution. The rigid steady state solution can help avoid the transients that cause havoc in the convergence of some problems.

The convergence of the static aeroelastic analysis is shown through the  $L_2$  norm of

the residual of the energy equation. Figure 5.14 indicates the convergence of the fluids equations versus the iteration step on a semi-log scale. Also, as a validation of the force transfer scheme used to transfer loads from the aerodynamic grid to the structural mesh, the summation of the loads in the z-direction on the aerodynamic grid and finite element model were compared; they remained identical at each iteration. The comparisons of the loads also shows the convergence of the CFD/CSD solution. The moments were not compared.

Figures 5.16 and 5.17 show the pressure coefficient variation on the upper surface of a rigid and flexible wing, respectively. The rigid wing pressure contours are calculated using fluids code only. The flexible wing pressure contours are calculated using the fluids and structures codes. The wing deflects less than one inch. This is due to the high stiffness of the wing structure and can be seen by comparing the pressure coefficient contours for the rigid and flexible wings.

The initial undeflected state of the wing-box tip section and the final converged state are shown in Fig. 5.18. It can be seen that the trailing edge deflects more than the leading edge. Also, in Fig. 5.19, the initial undeflected and final statically converged wing-box leading edge displacements are shown. Again, it can be seen that the wing does not deflect much. The tip is displaced by less than 1% of the root chord. The stress variation,  $\sigma_{yy}$ , on the upper surface of the wing is shown in Fig. 5.20. It can be seen in Fig. 5.20 that the maximum compressive stresses occur at the wing root. The entire upper wing surface is in compression, while near the tip the stresses are nearly zero.

The parallel performance of the structures code is summed up in Fig. 5.21. There are five lines indicating assembly, factorization, forward substitution, backward substitution, and total times. The factorization part of the structures code dominates the time as the number of processors increases. This is probably due to the increasing number of communications needed as more processors are involved. The communication to computation speeds are also important when looking at the performance of the code. This can be seen by the fact that one processor can factor much faster than anything greater than one processor. Therefore, it might be concluded that parallel computers are not worth the research effort, but that would be short sighted. One

reason for the results in Fig. 5.21 is that the communication speed of the computer used is much slower than the computational speed of the computer. Communication speeds are increasing with the use of high performance networks, e.g. IBM SP-2, and the ability of parallel computers will be even more demonstrable in the near future. In addition, when parallelizing a code, there is an overhead cost which involves the time it takes to communicate between the processors. If the problem size is fixed, and the number of processors is increased, then eventually, there will be so many communications and little computation, and the overhead cost will dominate the results. But if the overhead cost is small compared to that for the computations, then good parallel performance results can be obtained. Parallel performance of the aeroelastic analysis is not shown due to the unavailability of the iPSC/860.

## 5.8 Conclusions

Parallel aeroelastic analysis was performed on a typical wing-body configuration using a parallel CFD code in conjunction with a parallel CSD code. A parallel CSD code is developed in this study using Allman's triangular element in conjunction with axial bars to represent the wing-box structure of the wing. Reasonable results were obtained, however the performance of the CSD code on a parallel machine seemed poor. But this can be easily attributed to the fact the the problem size was not large enough to overcome the cost of parallelization of the code, the overhead cost. Parallel computing is the future of scientific computing. It is this researcher's belief that most large scale applications will be created for parallel computing paradigms in the near future.

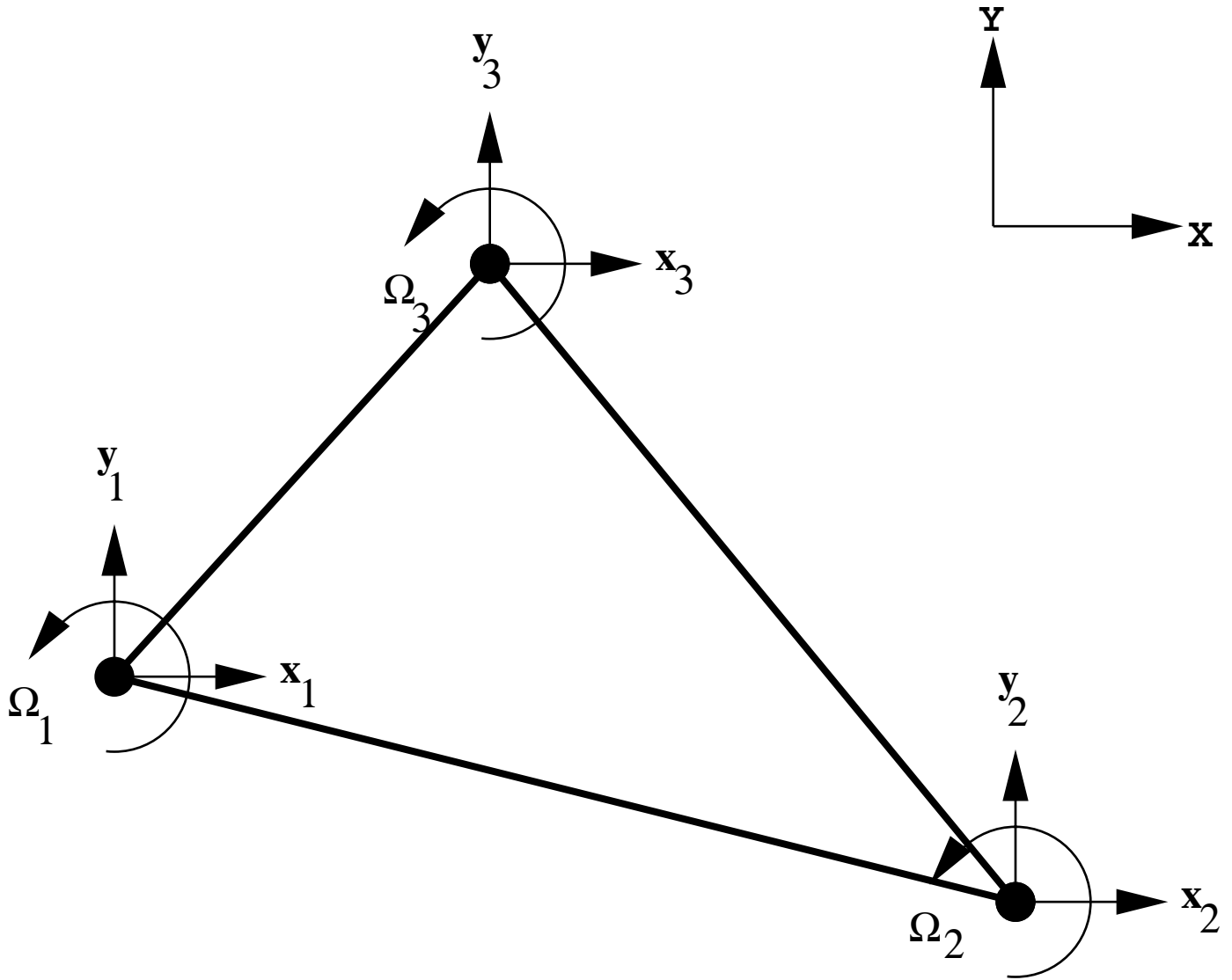


Figure 5.1: Allman's Triangular Element

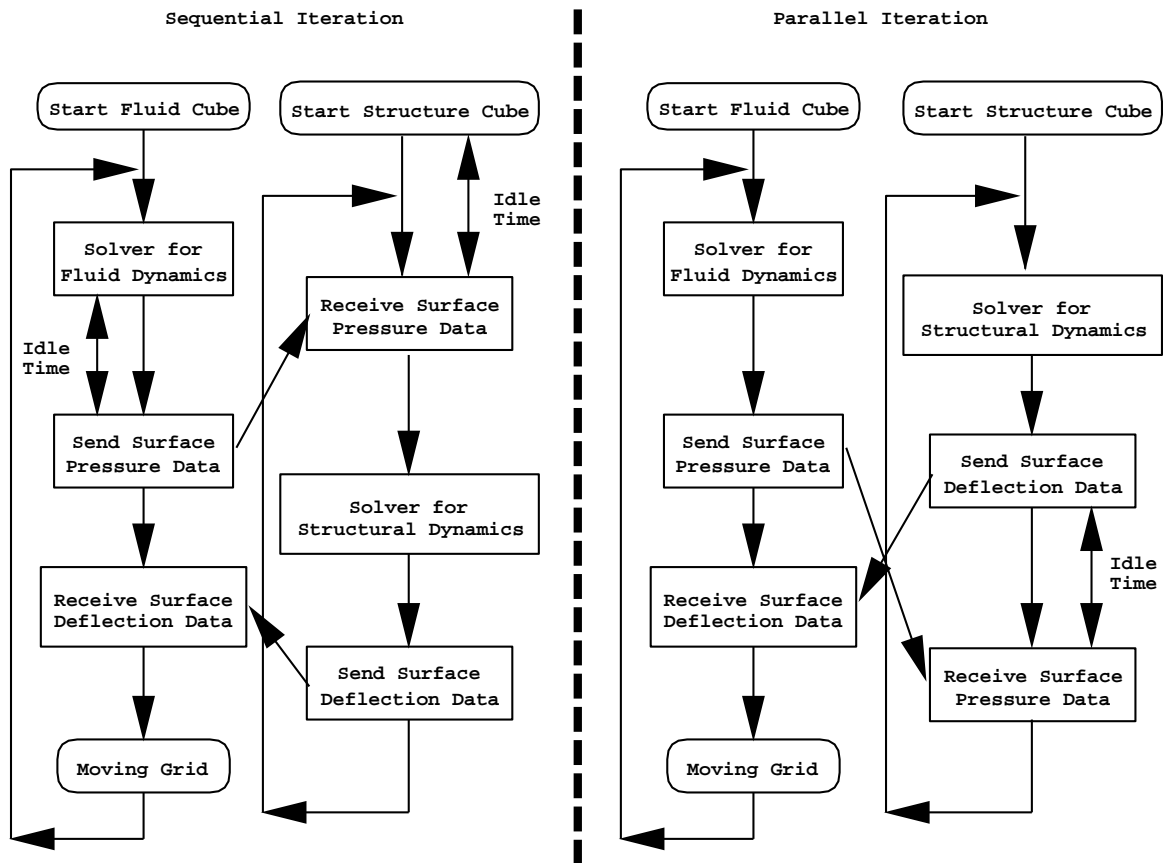


Figure 5.2: Schematic of the Solution Procedure and the Coupling between Fluid and Structure Domains

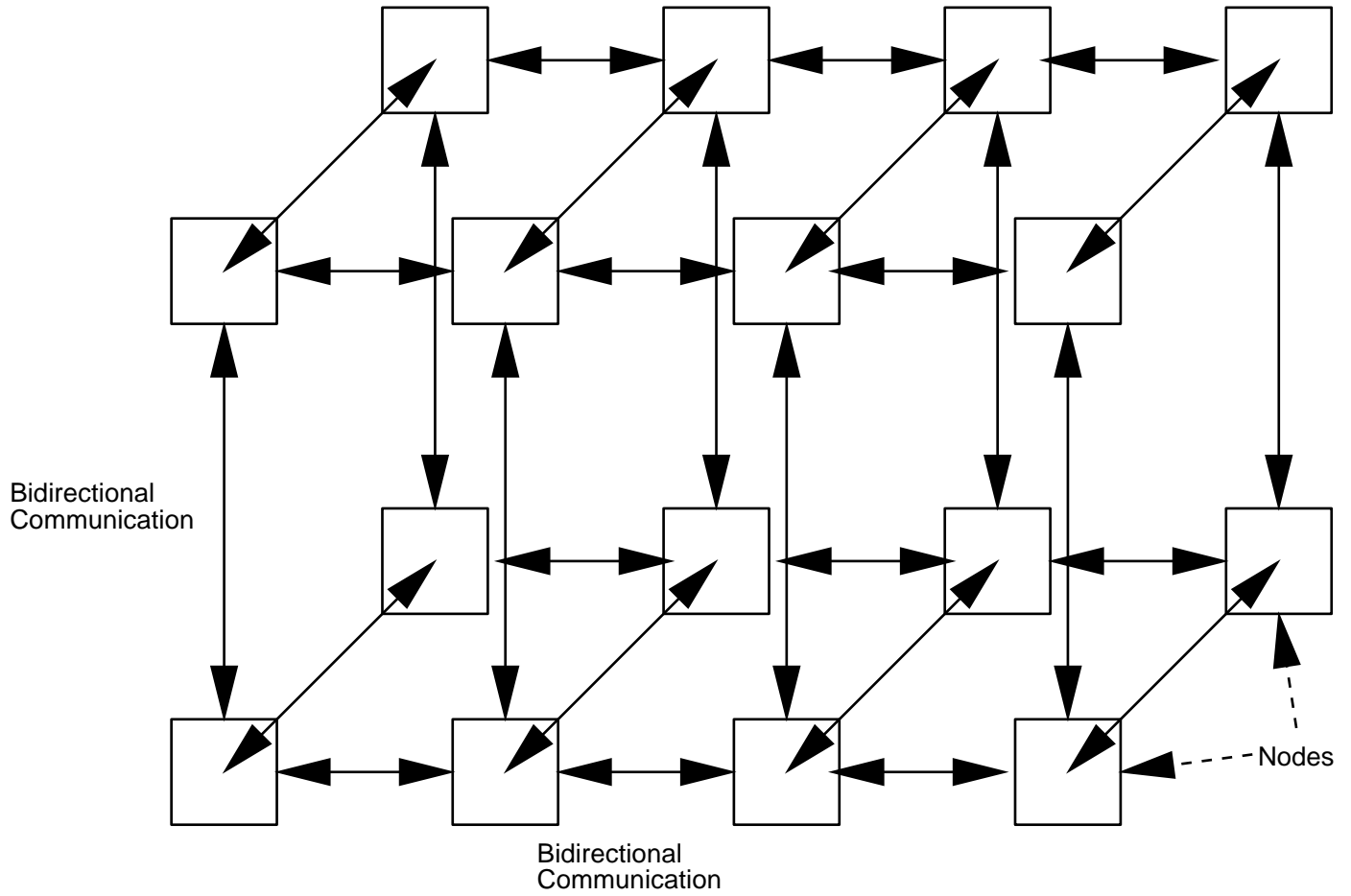


Figure 5.3: Uni-Partitioning Scheme of the Fluid Domain

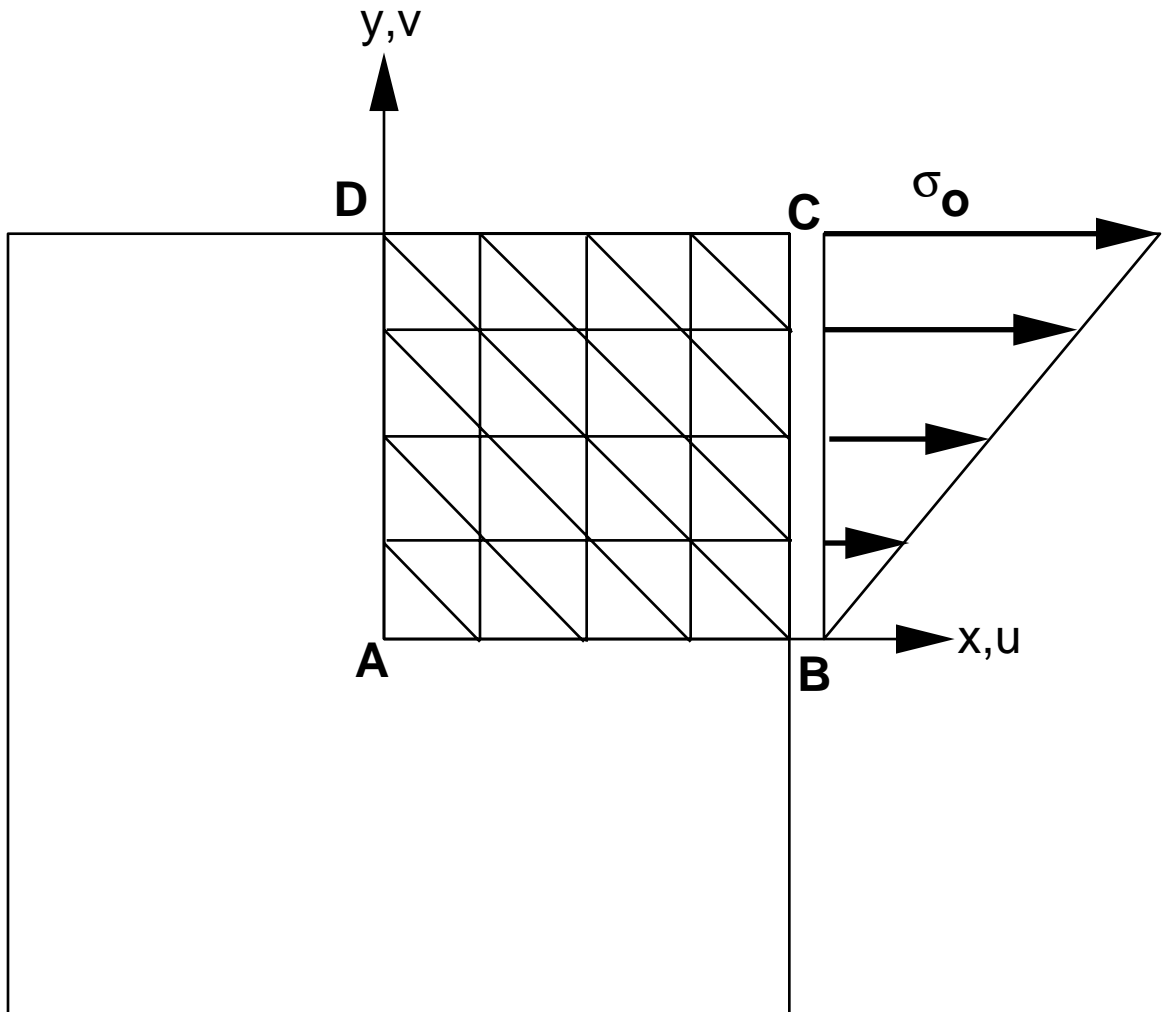


Figure 5.4: Discretization of a Square Panel with Linearly Varying Edge Normal Stress

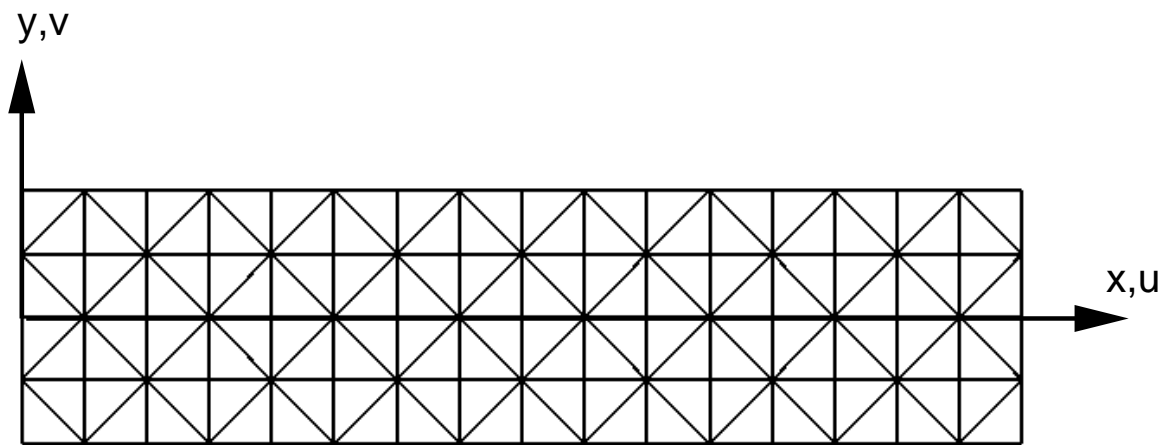


Figure 5.5: Discretization of a Cantilever Beam with Tip Load

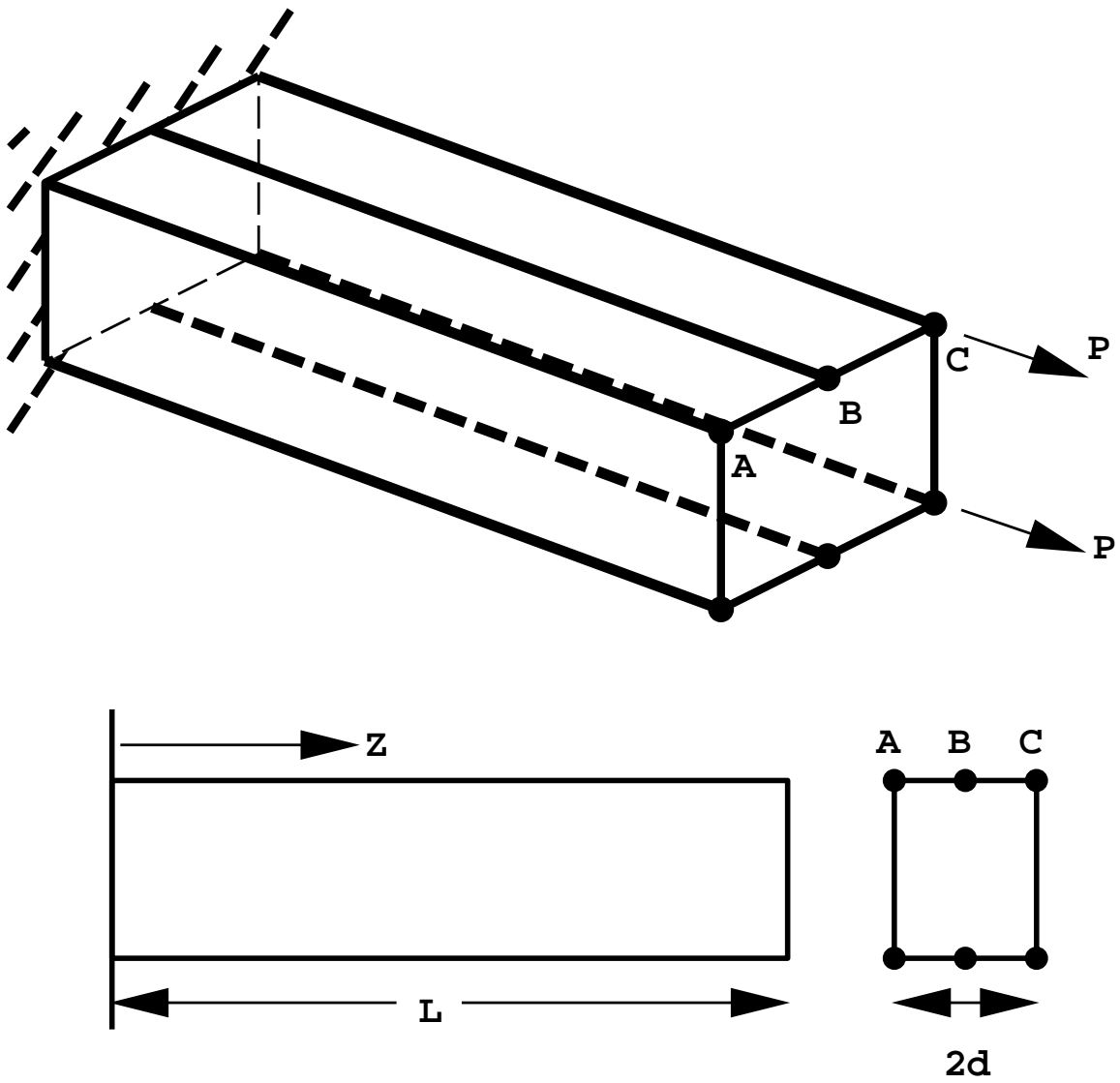


Figure 5.6: A Box Beam Subjected to Axial Loads

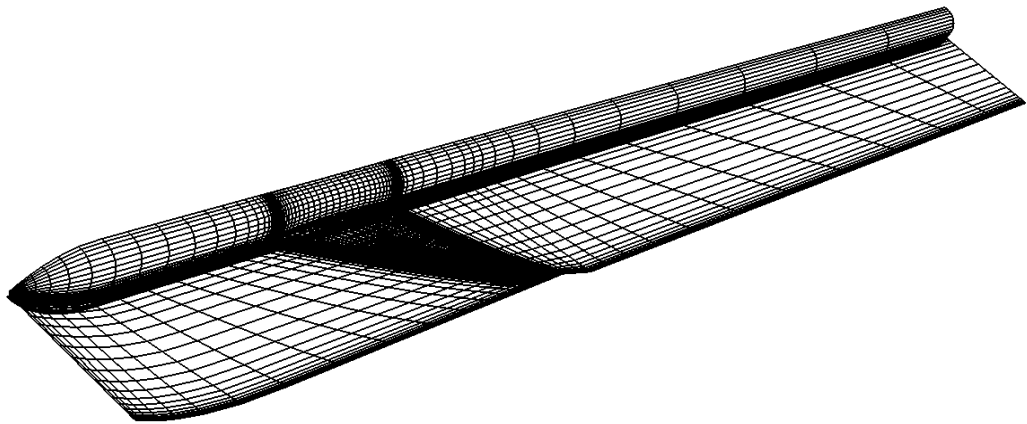


Figure 5.7: Aerodynamic Surface Grid for a Typical Wing-body Configuration

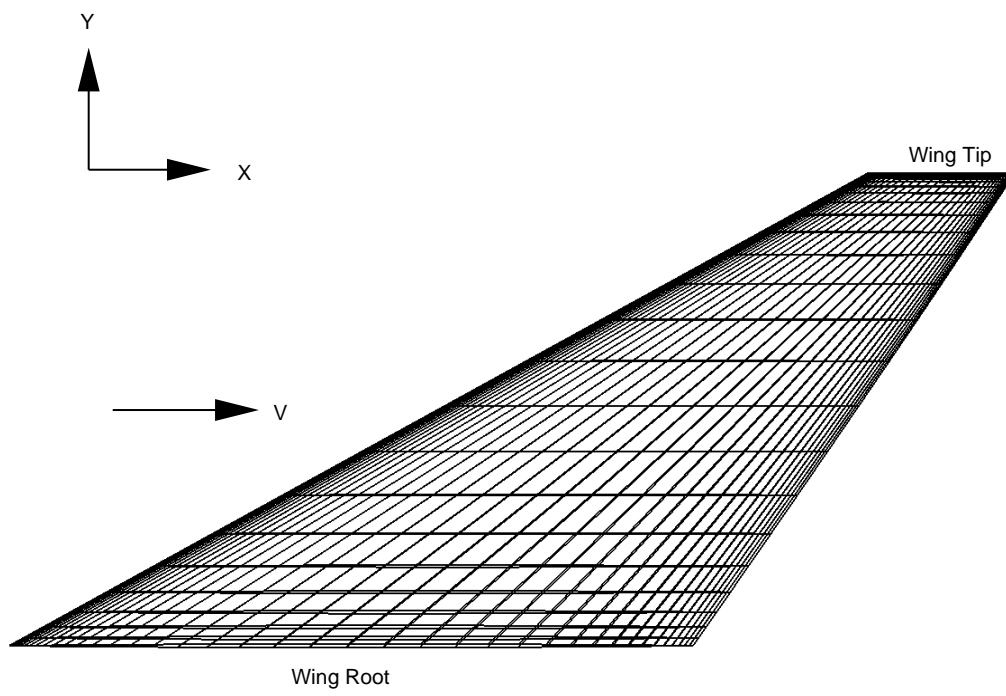


Figure 5.8: Aerodynamic Surface Grid of Wing

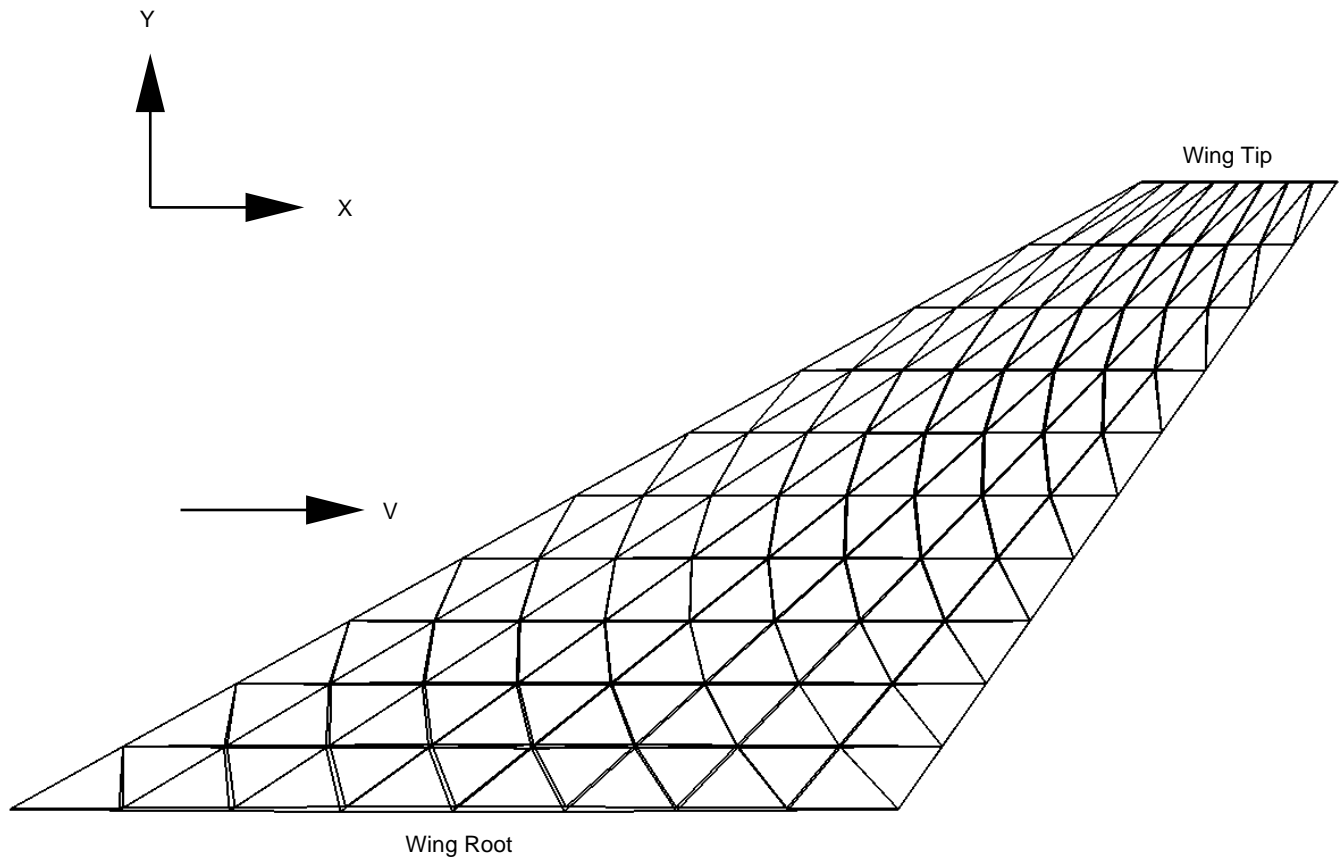


Figure 5.9: Top View of the Structural Discretization of the Wing

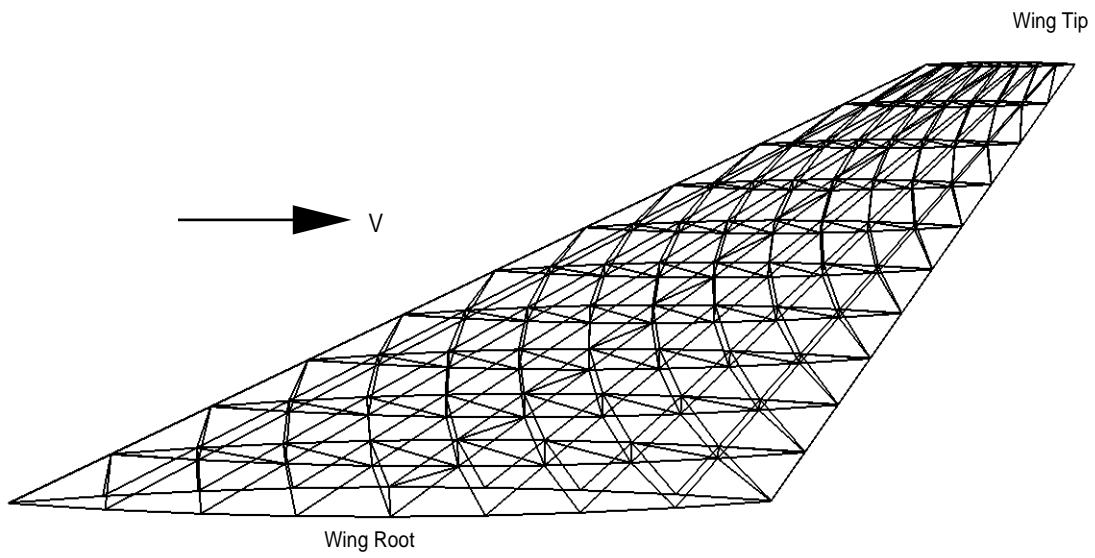


Figure 5.10: Structural Modeling of the Entire Wing

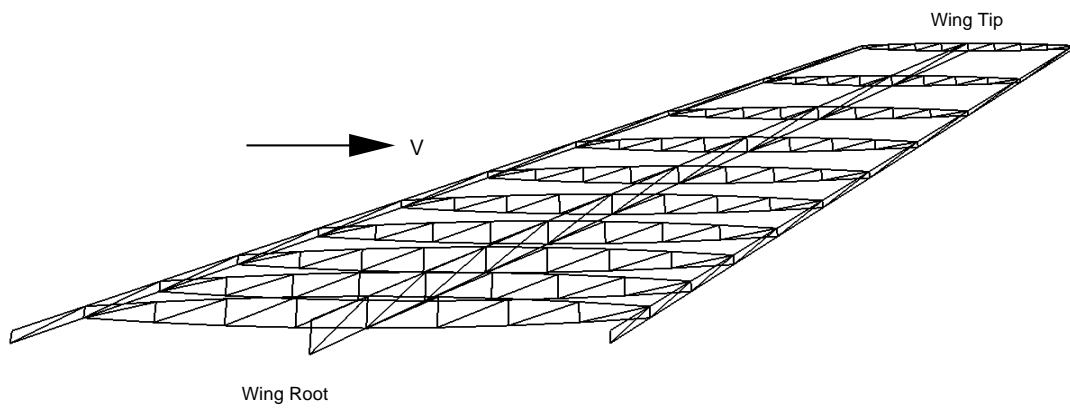


Figure 5.11: Structural Discretization of the Spars and Ribs of the Wing

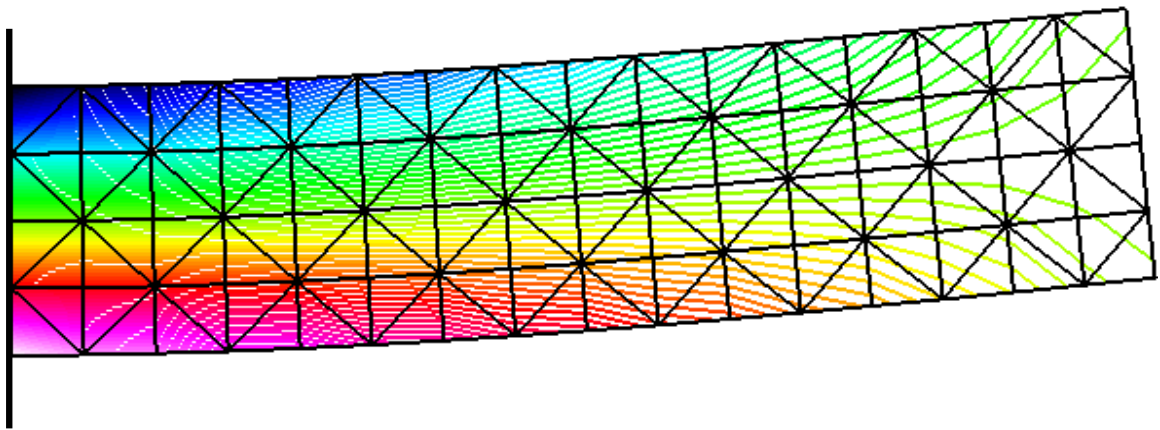


Figure 5.12: Stress Contours Due to a Tip Load on the Cantilevered Beam

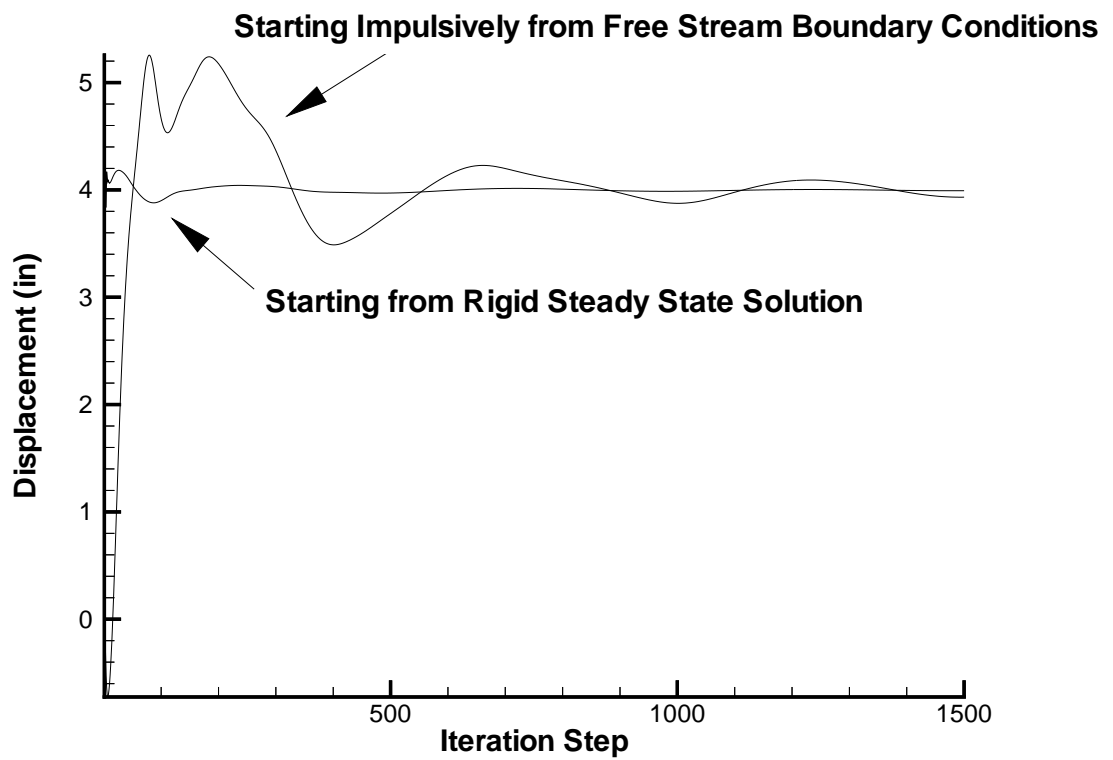


Figure 5.13: Wing Leading Edge Tip History Versus Iteration Step

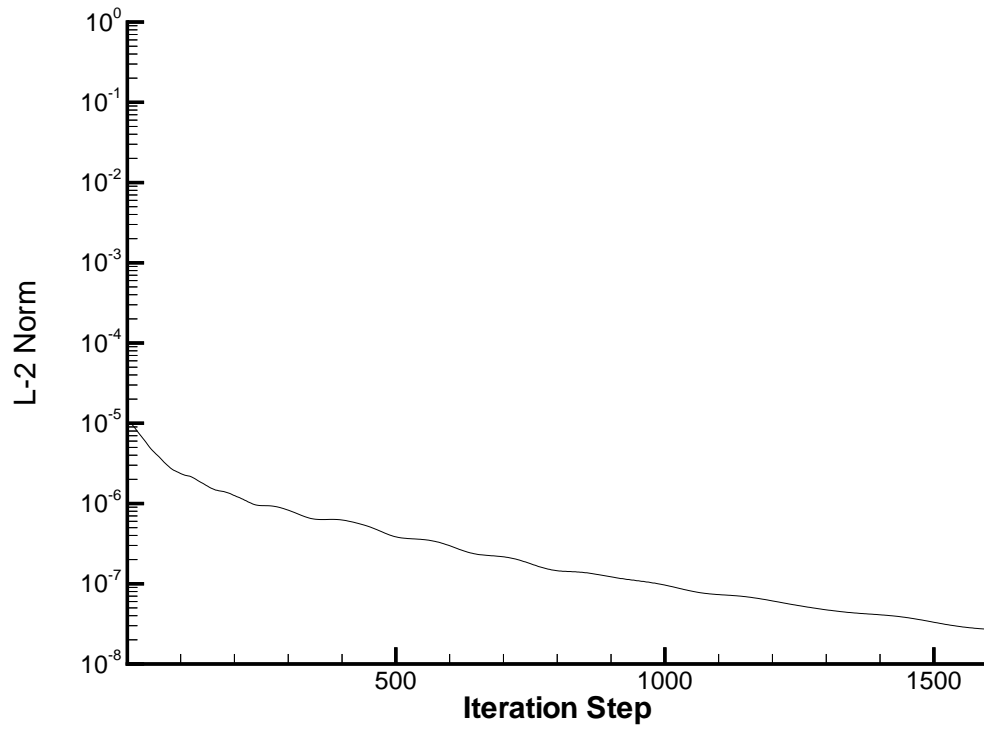


Figure 5.14:  $L_2$  Norm of the Residual of the Energy Equation

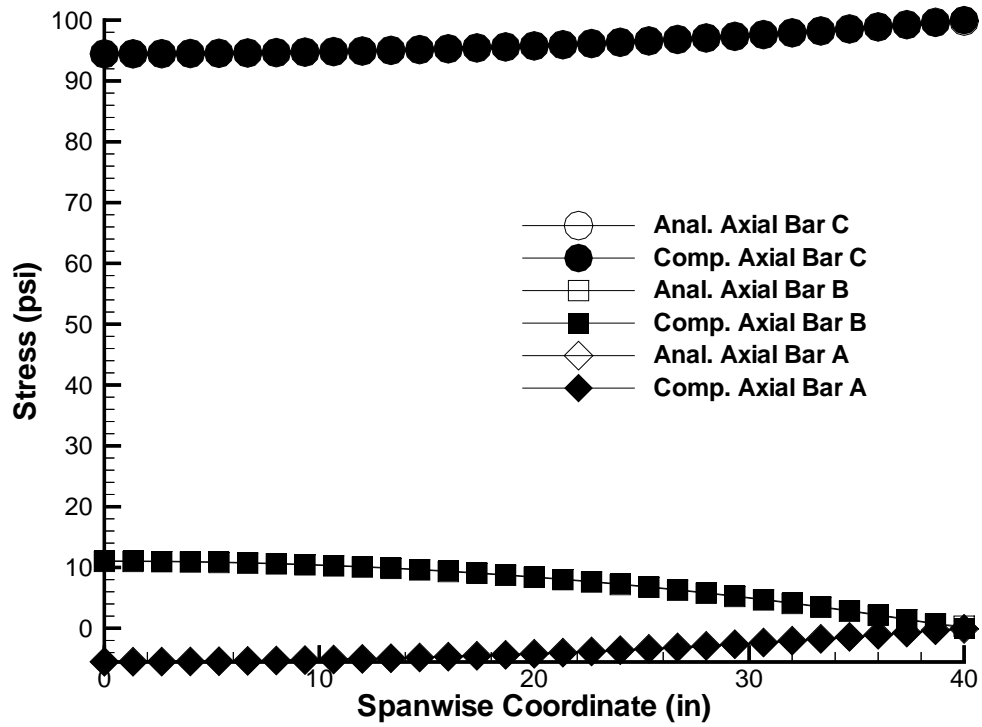


Figure 5.15: Comparison of the Stresses in the Axial Bars of Box Beam

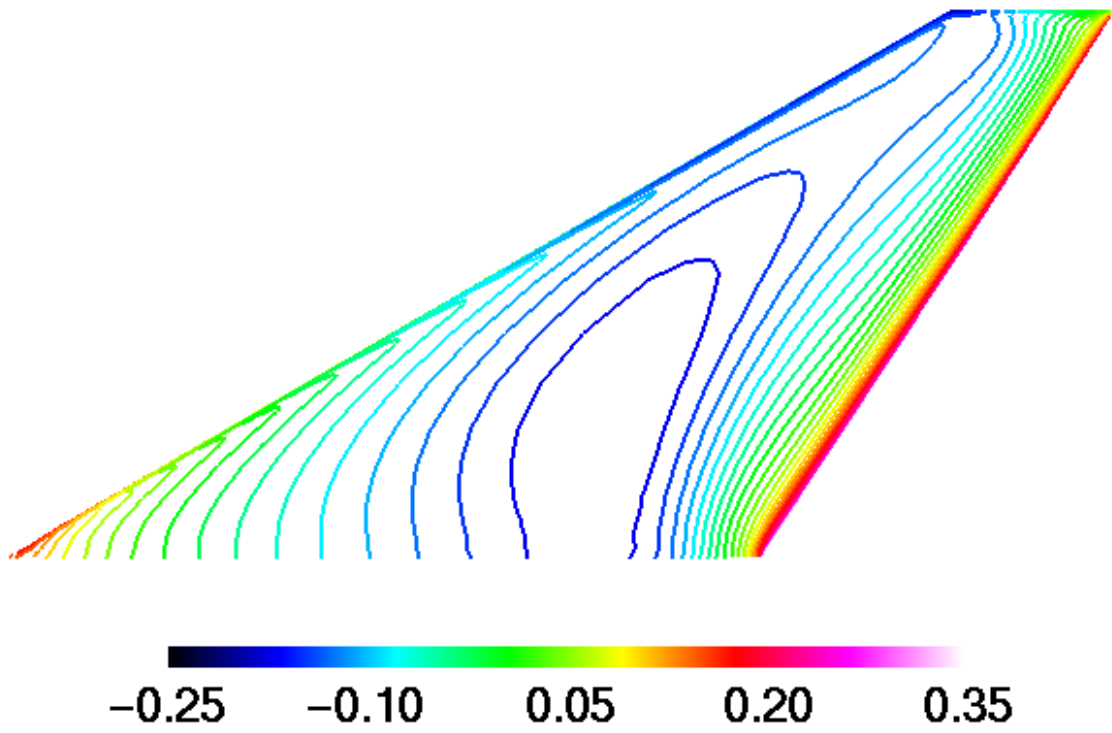


Figure 5.16:  $C_p$  Variation on the Upper Surface of a Rigid Wing

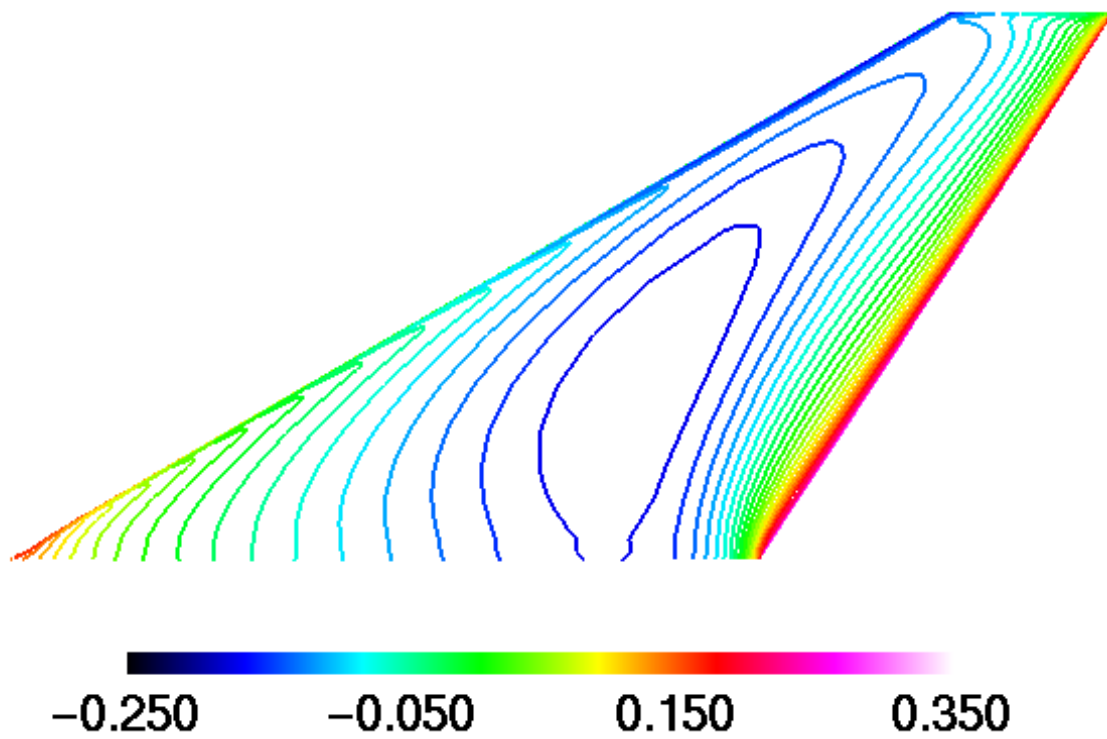


Figure 5.17:  $C_p$  Variation on the Upper Surface of a Flexible Wing

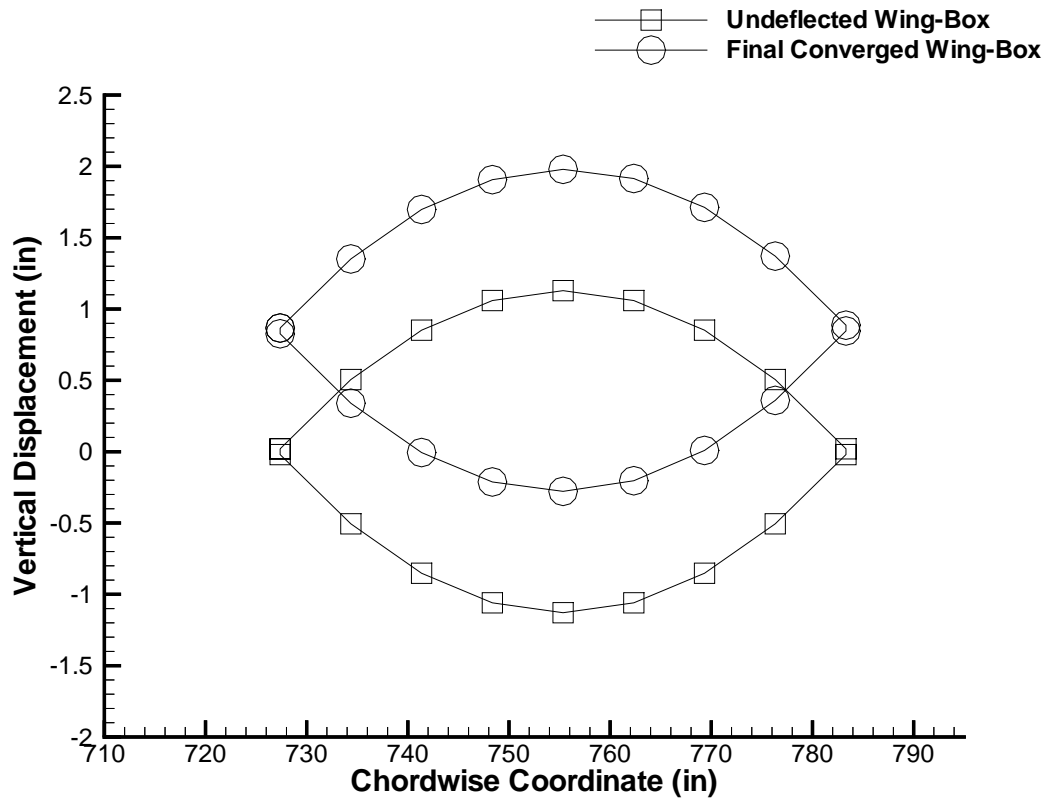


Figure 5.18: Initial Undeformed and Final Converged Wing-box Tip Section

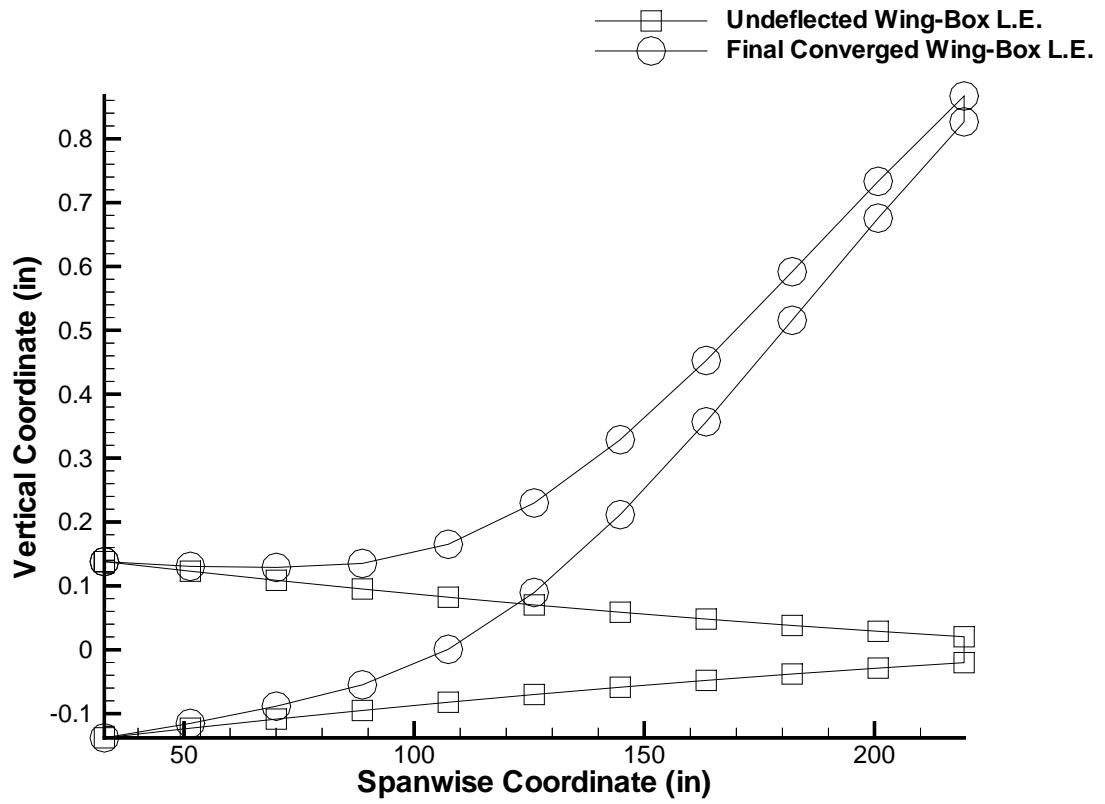


Figure 5.19: Initial Undeformed and Final Converged Wing-box Leading Edge (Front Spar)

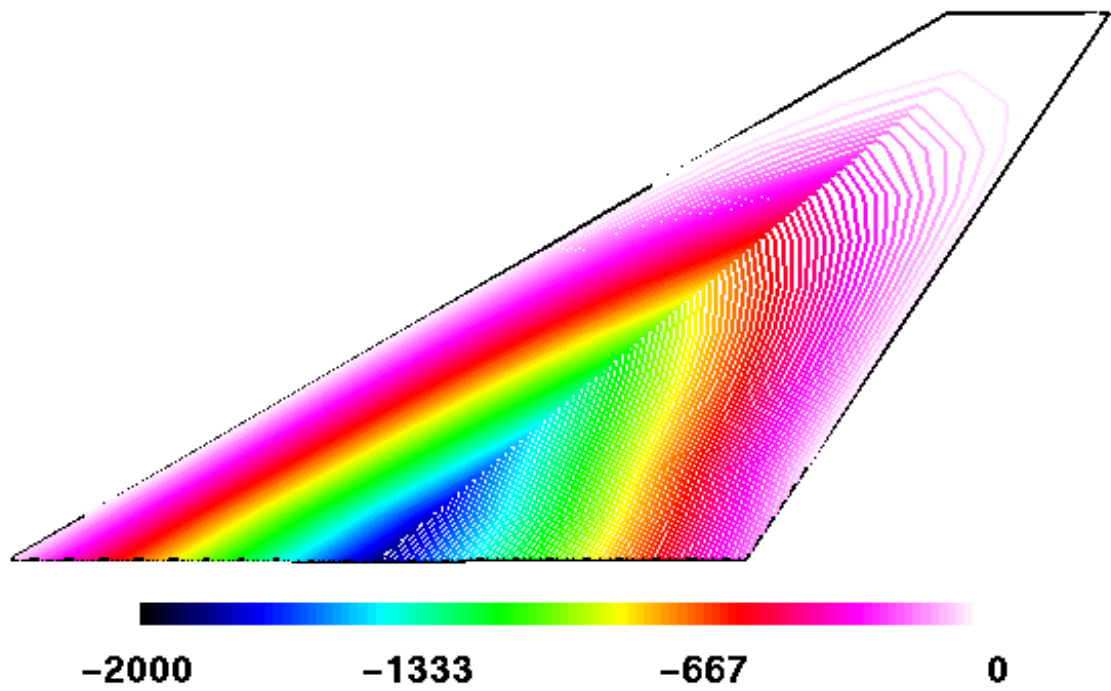


Figure 5.20: Bending Stress Variation on the Upper Surface of a Flexible Wing (psi)

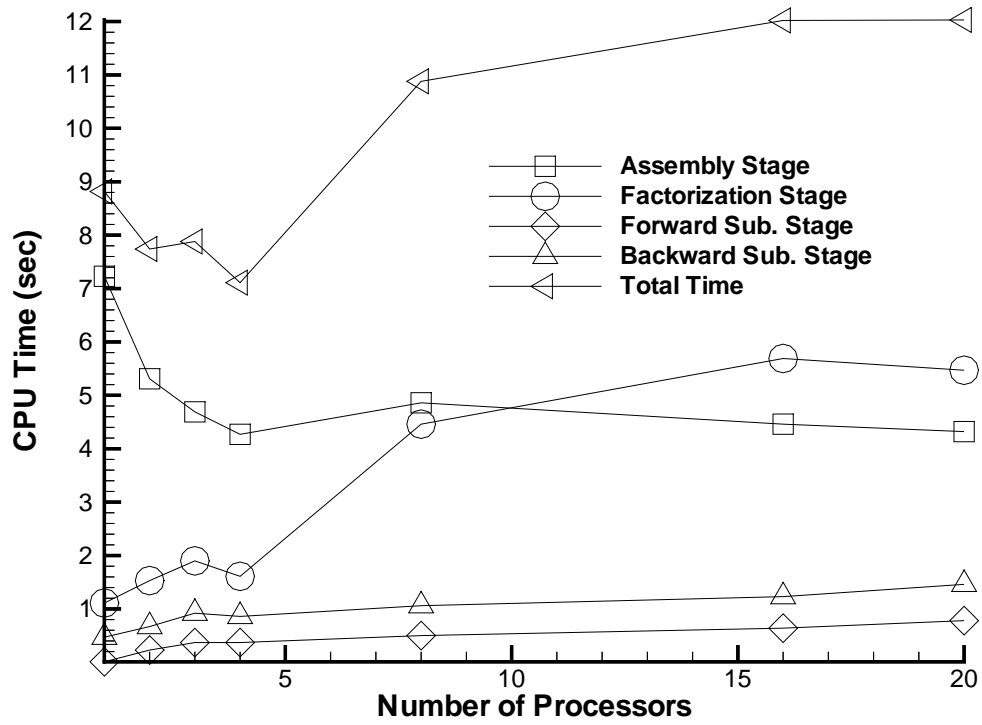


Figure 5.21: CPU Time for the Execution of the Various Parts of the Finite Element Wing-box Code

Element type	Stress at x=12 in and y=6 in	Vertical deflection at x=48 in and y=0 in
Allman's Triangular Element	59.6 ksi	0.3471 in
Constant Strain Triangle	53.5 ksi	0.3115 in
Linear Strain Triangle	60.0 ksi	0.3556 in
Olsen and Bearden [66]	60.0 ksi	0.3558 in

Table 5.1: Comparison of Allman's Triangular Element, Constant Strain Triangle, and Linear Strain Triangle for Analysis of Cantilevered Beam

# Chapter 6

## Conclusions

An aeroelastic coupling procedure is presented whereby static aeroelastic analysis is performed by coupling a wide variety of computational fluid dynamics (CFD) codes and computational structural dynamics (CSD) codes. The procedure is demonstrated by performing static aeroelastic analysis on an F/A-18 Stabilator by using finite element capability in NASTRAN coupled with Euler flow equations as available in NASTD (an in-house McDonnell Douglas CFD code). In addition, the Aeroelastic Research Wing (ARW-2) is used to validate the aeroelastic coupling procedure by using a finite element wing-box code (developed as part of this work) coupled with Navier-Stokes equations as available in ENSAERO (NASA Ames CFD code). Experimental data was used to compare the computational aeroelastic solution of the ARW-2 and good agreement was obtained. The increased accuracy of the use of direct finite element displacement data as opposed to modal analysis is also shown. The advantage of this aeroelastic coupling procedure is that it only requires the grid points of the CSD and CFD grids. Using only the grid point locations, necessary mappings are created to perform static aeroelastic analysis. This procedure is modular. Currently, only the vertical displacements are considered. Therefore, the interpolation scheme can be changed to account for the in-plane displacements.

The aeroelastic coupling procedure is not as efficient as a completely integrated scheme. This procedure is also limited in that large amounts of deformation will cause the problems with CFD grid deformation. This will occur at points near divergence

speeds. However, for swept back wings, divergence is not a problem.

Though the parallel aeroelastic analysis research is not at yet at the desired point of completion due to the rapid changes in technology, there is a need for a parallel finite element code which runs efficiently. This is required for the purpose of performing parallel aeroelastic analysis. In this author's opinion, there is no doubt that the future applications of aeroelastic analyses will be performed on parallel computers. With the costs of processors decreasing, more businesses and schools will be able to afford parallel machines, increasing the need for parallel software. The overall result will be the ability to run applications in a fraction of the time it takes now. Technology is improving rapidly, and considerable research efforts will be spent in the future creating efficient parallel codes.

# Bibliography

- [1] R. L. Bisplinghoff. *Principles of Aeroelasticity*, chapter 1. Dover Publications, Inc.
- [2] R. L. Bisplinghoff, H. Ashley, and R. L. Halfman. *Aeroelasticity*, chapter 1. Addison-Wesley Publishing Company.
- [3] R. Yurkovich. “Optimum Wing Shape for an Active Flexible Wing.” *AIAA Paper 95-1220*, April 1995.
- [4] B. P. III, S. R. Cole, and G. D. Miller. “A Summary of the Active Flexible Wing Program.” *AIAA Paper 92-2080*, 1992.
- [5] G. Andersen, E. Forster, R. Kolonay, and F. Eastep. “Multiple Control Surface Utilization in Active Aeroelastic Wing Technology.” *Journal of Aircraft*, **volume 34**, no. 4, July-August 1997.
- [6] G. D. Miller. “Active Flexible Wing (AFW) Technology.” *AFWAL-TR-87-3096*, 1987.
- [7] J. R. Hooker, A. W. Burner, and R. Valla. “Static Aeroelastic Analysis of Transonic Wind Tunnel Models Using Finite Element Methods.” *AIAA Paper 97-2243*, June 1997. Presented at the 15th Applied Aerodynamics Conference.
- [8] H. S. Murty and G. W. Johnson. “Nonlinear Aspects of Transonic Aeroelasticity.” *Canadian Aeronautics and Space Journal*, **volume 39**, no. 2, pages 78–84, June 1993.

- [9] G. P. Guruswamy. “Coupled Finite-Difference/Finite-Element Approach for Wing-Body Aeroelasticity.” *AIAA Paper 92-4680*, September 1992.
- [10] G. Tzong, H. H. Chen, K. C. Chang, T. Wu, and T. Cebeci. “A General Method for Calculating Aero-Structure Interaction on Aircraft Configurations.” *AIAA Paper 96-3982*, September 1996.
- [11] D. Macumurdy, R. Kapania, and G. P. Guruswamy. “Static Aeroelastic Analysis of Wings Using Euler/Navier-Stokes Equations Coupled with Wing-Box Finite Element Structures.” *AIAA Paper 94-1587*, April 1994.
- [12] M. K. Bhardwaj, R. K. Kapania, C. Byun, and G. P. Guruswamy. “Parallel Aeroelastic Computations by Using Coupled Euler Flow and Wing-Box Structural Models.” *AIAA Paper 95-1291*, April 1995.
- [13] G. P. Guruswamy and C. Byun. “Fluid-Structural Interactions Using Navier-Stokes Flow Equations Coupled with Shell Finite Element Structures.” *AIAA Paper 93-3087*, July 1993.
- [14] G. P. Guruswamy and C. Byun. “Direct Coupling of Euler Flow Equations with Plate Finite Element Structures.” *AIAA Journal*, **volume 33**, no. 2, pages 375–377.
- [15] J. C. Neuman III. “Efficient Nonlinear Static Aeroelastic Wing Analysis.” *Submitted to Computers and Fluids*, December 1996.
- [16] G. P. Guruswamy and T. Y. Yang. “Aeroelastic Time-Response Analysis of Thin Airfoils by Transonic Code LTRAN2.” *Computers and Fluids*, **volume 9**, no. 4, pages 409–425, December 1980.
- [17] O. A. Bauchau and J. U. Ahmad. “Advanced CFD and CSD Methods for Multidisciplinary Applications in Rotorcraft Problems.” *AIAA Paper 96-4151*, July 1996.
- [18] O. O. Bendiksen. “A New Approach to Computational Aeroelasticity.” *AIAA Paper 91-0939*, April 1991.

- [19] F. F. Felker. “A New Method for Transonic Static Aeroelastic Problems.” *AIAA Paper 92-2123*, April 1992.
- [20] C. J. Borland and D. Rizzetta. “XTRAN3S - Transonic Steady and Unsteady Aerodynamics for Aeroelastic Applications, Volume 1-Theoretical Manual.” *AFWAL-TR-80-3017*, December 1985.
- [21] G. P. Guruswamy, P. M. Goorjian, and F. J. Merritt. “ATRAN3S - An Unsteady Transonic Code for Clean Wings.” *NASA TM 86783*, December 1985.
- [22] J. T. Batina, R. M. Bennett, D. A. Seidal, S. R. Cunningham, and S. R. Bland. “Recent Advances in Transonic Computational Aeroelasticity.” *NASA TM 100663*, September 1988.
- [23] G. P. Guruswamy. “Unsteady Aerodynamic and Aeroelastic Calculations of Wings Using Euler Equations.” *AIAA Journal*, **volume 28**, no. 3, pages 461–469, March 1990.
- [24] G. P. Guruswamy. “Vortical Flow Computations on Swept Flexible Wings Using Navier-Stokes Equations.” *AIAA Journal*, **volume 28**, no. 12, pages 2077–2084, December 1990.
- [25] R. Chipman, C. Walters, and D. MacKenzie. “Numerical Computation of Aeroelastically Corrected Transonic Loads.” *AIAA Paper 79-0766*, 1979.
- [26] W. H. Mason, D. MacKenzie, M. Stern, W. F. Ballhaus, and J. Frick. “An Automated Procedure for Computing the Three Dimensional Transonic Flow over Wing Body Combinations, Including Viscous Effects. Volume 1 - Description of Analysis Methods and Applications.” *AFFDL TR-77-122*, **volume 1**, October 1977.
- [27] J. T. Batina. “Unsteady Euler Algorithm with Unstructured Dynamic Mesh for Complex-Aircraft Aeroelastic Analysis.” *AIAA Paper 89-1189*, April 1989.

- [28] R. D. Rausch, J. T. Batina, and H. T. Y. Yang. “Three-Dimensional Time-Marching Aeroelastic Analyses Using an Unstructured Euler Method.” *NASA TM 107567*, March 1992.
- [29] R. R. Craig Jr. *Structural Dynamics*, chapter 15. John Wiley & Sons, 1981.
- [30] M. D. Salas, editor, *Accuracy of Unstructured Grid Techniques Workshop*. NASA Conference Proceedings, NASA Langley Research Center, January 1990.
- [31] T. W. Purcell, C. J. Borland, and E. N. Tinoco. “Non-Linear Aeroelastic Predictions of Transport Aircraft.” *AIAA Paper 90-1852*, 1990.
- [32] “ELFINI Aeroelasticity - General Introduction Manual.” *Document No. ELF-AER-1*, Avions Marcel Dassault-Breguet, Aviation, Saint-Cloud. May 1989.
- [33] D. Schuster, J. Vadyak, and E. Atta. “Static Aeroelastic Analysis of Fighter Aircraft Using a Three-Dimensional Navier-Stokes Algorithm.” *AIAA Paper 90-0435*, January 1990.
- [34] D. T. Yeh. “Aeroelastic Analysis of a Hinged-Flap and Control Surface Effectiveness Using the Navier-Stokes Equations.” *AIAA Paper 95-2263*, June 1995.
- [35] J. K. Nathman and J. M. Barton. “Aeroelastic Calculations with an Euler Code.” *AIAA Paper 97-2271*, June 1997. Presented at the 15th Applied Aerodynamics Conference.
- [36] B. A. Robinson, J. T. Batina, and H. T. Y. Yang. “Aeroelastic Analysis of Wings Using the Euler Equations with a Deforming Mesh.” *AIAA Paper 90-1032*, April 1990.
- [37] V. B. Venkayya and V. A. Tischler. “ANALYZE - Analysis of Aerospace Structures with Membrane Elements.” *AFFDL-TR-78-170*, December 1978.
- [38] P. G. Buning, W. M. Chan, K. J. Renze, D. Sondak, I. T. Chiu, and J. P. Slotnick. “OVERFLOW User’s Manual, Version 1.6.” NASA Ames Research Center. 1991.

- [39] J. L. Hess, D. M. Friedman, and R. W. Clark. “Calculation of Compressible Flow About Three-Dimensional Inlets with Auxiliary Inlets, Slats, and Vanes by Means of a Panel Method.” *NASA CR 174975*, 1985.
- [40] M. J. Smith, D. H. Hodges, and C. E. S. Cesnik. “An Evaluation of Computational Algorithms to Interface between CFD and CSD Methodologies.” *Report WL-TR-96-3055*, November 1995.
- [41] K. Appa. “Finite-Surface Spline.” *Journal of Aircraft*, **volume 26**, no. 5, pages 495–496, 1989.
- [42] C. Byun and G. P. Guruswamy. “Wing-Body Aeroelasticity Using Finite-Difference Fluid/Finite-Element Structural Equations on Parallel Computers.” *AIAA Paper 94-1487*, April 1994.
- [43] R. M. V. Pidaparti. “Structural and Aerodynamic Data Transformation Using Inverse Isoparametric Mapping.” *Journal of Aircraft*, **volume 29**, no. 3, pages 507–509, 1992.
- [44] R. L. Harder and R. N. Desmarais. “Interpolation Using Surface Splines.” *Journal of aircraft*, **volume 9**, no. 2, pages 189–191, October 1971.
- [45] S. Obyashi and G. P. Guruswamy. “Convergence Acceleration of a Navier-Stokes Solver for Efficient Static Aeroelastic Computations.” *AIAA Journal*, **volume 33**, no. 6, pages 1134–1141, June 1995.
- [46] R. H. MacNeal. “The NASTRAN Theoretical Manual.” *NASA SP-221(01)*, April 1971.
- [47] G. P. Guruswamy. “ENSAERO - A Multidisciplinary Program for Fluid/Structural Interaction Studies of Aerospace Vehicles.” *Company System Engineering*, **volume 1**, no. 2-4, pages 237–257, 1990.
- [48] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*, chapter 1. The Benjamin/Cummings Publishing Company, Inc., 1994.

- [49] R. L. Burden and J. D. Faires. *Numerical Analysis*, chapter 2. PWS-KENT Publishing Company.
- [50] E. H. D. et. al. *A Modern Course in Aeroelasticity*, chapter 1. Sijthoff and Noordhoff.
- [51] W. P. Rodden, J. A. McGrew, and T. P. Kalman. “Comment on ‘Interpolation Using Surface Splines’.” *Journal of aircraft*, **volume 9**, pages 869–871, December 1972.
- [52] R. H. Bush. “A Three Dimensional Zonal Navier-Stokes Code for Subsonic Through Hypersonic Propulsion Fields.” *AIAA Paper 88-2830*, 1988.
- [53] M. C. Sanford, D. A. Seidel, C. V. Eckstrom, and C. V. Spain. “Geometrical and Structural Properties of an Aeroelastic Research Wing (ARW-2).” *NASA TM 4110*, April 1989.
- [54] M. Farhangnia, G. P. Guruswamy, and S. Biringen. “Transonic-Buffer Associated Aeroelasticity of a Supercritical Wing.” *AIAA Paper 96-0286*, January 1996.
- [55] T. A. Byrdsong, R. R. Adams, and M. C. Sanford. “Close-range Photogrammetric Measurement of Static Deflections of an Aeroelastic Supercritical Wing.” *NASA TM 4194*, 1990.
- [56] D. J. Allman. “A Compatible Triangular Element Including Vertex Rotations for Plane Elasticity Problems.” *Computers and Structures*, **volume 19**, no. 1-2, pages 1–8, 1996.
- [57] R. Peyret and H. Viviand. “Computation of Viscous Compressible Flows Based on Navier-Stokes Equations.” *AGARD AG-212*, 1975.
- [58] R. Beam and R. F. Warming. “An Implicit Finite-Difference Algorithm for Hyperbolic Systems in Conservation Law Form.” *Journal of Computational Physics*, **volume 22**, no. 9, pages 87–110, September 1976.

- [59] T. H. Pulliam and D. S. Chaussee. “A Diagonal Form of an Implicit Approximate Factorization Algorithm.” *Journal of Computational Physics*, **volume 39**, no. 2, pages 347–363, February 1981.
- [60] C. Farhat and E. Wilson. “A Parallel Active Column Equation Solver.” *Computers and Structures*, **volume 28**, no. 2, pages 289–304, 1988.
- [61] J. S. Ryan and S. K. Weeratunga. “Parallel Computation of 3-D Navier-Stokes Flowfields for Supersonic Vehicles.” *AIAA Paper 93-0064*, pages 87–110, January 1993.
- [62] T. L. Holst, J. Flores, U. Kaynak, and N. Chaderjian. “Navier-Stokes Computations, Including a Complete F-16 Aircraft.” In P. A. Henne, editor, *Applied Computational Aerodynamics*, volume 125 of *Progress in Astronautics and Aeronautics*, chapter 21. 1990.
- [63] M. J. Turner, R. W. Clough, H. C. Martin, and L. J. Topp. “Stiffness and Deflection Analysis of Complex Structures.” *Journal of Aeronautical Sciences*, **volume 23**, pages 805–823, 1956.
- [64] B. F. de Veubeke. “Displacement and Equilibrium Models in the Finite Element Method.” In O. C. Zeinkiewicz and G. S. Holister, editors, *Stress Analysis*, chapter 9, pages 145–197. Wiley, New York, 1965.
- [65] J. H. Argyris. “Triangular Elements with Linearly Varying Strain for the Matrix Displacement Method.” *Journal of Royal Aeronautical Sciences*, **volume 69**, pages 711–713, 1965.
- [66] M. D. Olsen and T. W. Bearden. “A Simple Flat Shell Element Revisited.” *International Journal of Numerical Methods in Engineering*, **volume 14**, pages 51–68, 1979.

# Appendix A

## Finite Element Wing-box Source Code

This is the finite element wing-box source code written in FORTRAN. It is used to do the parallel aeroelastic analysis as well as serial analysis on the aeroelastic research wing (ARW-2). The following is the serial version, but can be run using NX if the csend and crecv routines are deleted. The forsub, bacsub, and factor routines are not included, but can be found in Ref. [60].

```
c      This is the program for wingbox modeling.
c      include 'wboxbc.f'
c      include '../fcube.h'
c      maxsen      =      number of d.o.f. per processor (after b.c.)
c      nsizeg      =      total number of elements in skyline scheme
c                      (after b.c.)
c      tot         =      total number of elements per processor
c      nsizes      =      total number of elements per processor
c                      (after b.c.)
c
c      integer maxsen , nsizeg , nsizes , tot
c      parameter (maxsen = nogn / nodcube + 1 )
c      parameter (nsizeg = nogn * (nogn+1) / 2 )
c      parameter (tot   = (tdof * (tdof+1) / 2) / nodcube + 1 )
```



```

c      method                :          1..5 depending on which
c                          :          dynamic solver (see below)
c      The following integer delcarations based on include file.
      integer no1(nele) , no2(nele) , no3(nele) , nopel(nele) ,
1      mat(nele) , isurf(nele) ,
2      dd(tdof) , df(tdof) , dap(tdof) ,
3      mp(nogn) , maxa(nogn+1) ,
4      colpos1(maxsen) , coltip(maxsen)
c      The following are arrays for integers
      integer tids(0:32) , ibuff(20) , nfldom(64) , ele(50,50)
c      The following 4 lines are for common blocks
      integer con , colpos , mm ,
7      colposp , mk ,
8      jwngle , jwngtre , kroot1 , kroot2 , indtip ,
9      nfspc , fsprocnum , ibeg , iend , jbeg , jend
c      The remaining are all other integers.
      integer n1 , n2 , n3 , nodn , nofn , nproc , node ,
1      ile , ite , jroot , jroot1 , jtip , mpp ,
2      m , step , i , j , ch , iii , nrsrt , indx ,
3      jj , add , col , del , dbs , k , band , q1 , q2
      integer rowg , colg , tie , out , numt , info ,
5      nwksub , ncolpn , len , ii , ist , who ,
6      static , method , count , mess , icol ,
7      msgtype, myphynd , fphynd0 , msgid , msgid1 , msgid2
      integer msgid3 , ldx , ldy , msglen ,
9      nxt , nyt , nstart , nstop , irestart ,
1     nlep , io , ijk , dep , ent , rowgp , colgp
c     x(i), y(i), z(i)      :          coordinates of node i
c     e(i)                 :          Young's Modulus of
c                          :          material i, NOT element i
c     ke(i)                :          element stiffness matrix

```

```

c          (membrane element)
c      kz(i)          :      global stiffness matrix
c                      in skyline storage
c      u(i)          :      global displacement vector
c      fn(i)         :      global force vector
c      kzp(i)        :      stiffness matrix in skyline
c                      storage after
c                      applying B.C.'s
c      fp(i)         :      force vector used in solver
c      colval(i)     :      used in LDL solver, indicates
c                      which values
c                      :      of the stiffness vector
c                      are going to be
c                      worked on for this process
c      nu(i)         :      Poisson's Ratio for material i
c      diagon, subfor, work :      used in Active Column Solver
c      mz(i)         :      global mass matrix
c      mzp(i)        :      reduce mass matrix after
c                      applying B.C.'s
c      me           :      element mass matrix for
c                      membrane element
c      t1(i), t2(i), t3(i) :      thickness of nodes 1,2,3
c                      of element i or
c                      if axial bar, t1(i) is
c                      cross sectional area
c      rho(i)        :      density of material i
c      ma, ka        :      mass and stiffness matrices
c                      for axial bar
c      colvalm(i)    :      used in active column solver
c                      to indicate
c                      which part of mass matrix is

```

```

c                                     being used in this process
c   deltat, alpha, gamma,
c   a1,a2,a3,a4,a5,time,
c   limit, unew(i),uold(i),
c   velo(i),veln(i),acco(i),
c   accn(i),fo(i)           :           used in dynamic solver
real*8  x(tn) , y(tn) , z(tn) ,
1       e(nmat) , rho(nmat) , nu(nmat) ,
2       kz(tot) , mz(tot) ,
3       u(tdof) , fn(tdof) ,
4       unew(tdof) , velo(tdof) , veln(tdof) ,
5       acco(tdof) , accn(tdof) , fo(tdof) ,
6       uold(tdof) , fhat(tdof) , kzt(tdof,tdof+1) , err(tdof) ,
7       fp(nogn) , diagon(nogn) , sol(nogn) , sol1(nogn) ,
8       subfor(maxsen) ,
9       work(nsizeg) ,
1      t1(nele) , t2(nele) , t3(nele)
real*8  ke(18,18) , me(18,18) , rbuf(1000) , xg(3000) ,
1      yg(3000) , zg(3000) , area(50,50) , ug(100) ,
2      phig(100) , cp(50,50)
real*8  chord , x1 , y1 , x2 , y2 , x3 , y3 , t11 , t22 ,
1      t33 , z1 , z2 , z3 , lx , ly , lz , ln , deltat ,
2      alpha , gamma , a1 , a2 , a3 , a4 , a5 , time ,
3      pos , ti , limit
character *144 msgline
c   external initcubecomm , cubemap
common /soup/ con(nele,dof) , colpos(tdof+1) , mm(tdof)
common /cream/ colposp(nogn+1) , mk(nogn)
common /gridinfo/ jwngle , jwngtre , kroot1 , kroot2 , indtip
common /cpinfo/ nfspc , fsprocnm(32) , ibeg(32) , iend(32) ,
*           jbeg(32) , jend(32)

```

```
c      Find out number of processor and this process's node number
c      nproc = numnodes()
c      node = mynode()
      node = 0
      nproc = 1
      nstart = 1
      nstop = 1
      write(*,*) 'Hello'
      if (nproc.ne.nodcube) stop
      write(*,*) 'Hello'
c      if static = 0, do static analysis, i.e., Ku=F, and
c      if static not equal to zero, do dynamic analysis, i.e.,
c      Mu: + Ku = F. Also, if doing a dynamic analysis, set
c      time limit in seconds.
      static = 0
      limit = 0.1d0
      step = 1
      deltat = 0.01d0
      time = 0.0d0
c      If doing a dynamic analysis, need to choose which of Newmark's
c      methods to use.
c      method = 1: constant-average acceleration method (stable)
c      method = 2: linear acceleration method (conditionally stable)
c      method = 3: central difference method (conditionally stable)
c      method = 4: Galerkin method (stable)
c      method = 5: backward difference method (stable)
      method = 1
      count = 0
c      For dynamic analysis, set acceleration and
c      velocity to zero initially
      do i = 1 , tdof
```

```

        unew(i) = 0.0d0
        uold(i) = 0.0d0
        velo(i) = 0.0d0
        veln(i) = 0.0d0
        acco(i) = 0.0d0
        accn(i) = 0.0d0
        fhat(i) = 0.0d0
    end do
do i = 0 , 32
    tids(i) = i
end do
c Setup Intercube Communication Info
c call sficc_setup(node,nproc,nstart,nstop,irestart,nrsrt)
c Read Surface Aerodynamic Grid for Wing-Body and Extract Wing Only
c call rgrid(node,nproc,ile,ite,jroot,jroot1,jtip,phylen,xg,yg,zg)
c Calculate Areas for Pressures
c call areac(ile,ite,jroot,jtip,jroot1,phylen,xg,yg,zg,area)
c Need to get structural model here
c Call WINGP to partition the wing box and calculate initial
c configurations.
call wingp(nxt,nyt,nopel,no1,no2,no3,mat,x,y,z,t1,t2,t3,e,
*         nu,rho,isurf)
call mesh(tn,nax,nrib+nspar+nmem,nele,nopel,no1,no2,no3,isurf,
*         x,y,z)
c call beamp(nopel,no1,no2,no3,mat,x,y,z,t1,t2,t3,e,nu,rho)
c Map Aero. Grid Pts to Appropriate F.E.
c call ftsmap(nele,ile,ite,jroot,jtip,jroot1,nxt,nyt,isurf,
c *         xg,yg,zg,no1,no2,no3,x,y,z,ele)
c Call BOUND to get the boundary conditions for the wing box
call bound(nodn,nofn,dd,df,u,fn)
c Get loads

```

```
x1 = 1.0d0/3.0d0
x2 = 100.0d0*0.4d0*0.4d0*0.5d0*x1
c   do i = 1 , nele
c     if (isurf(i).eq.1) then
c       n1 = 6*no1(i)-3
c       n2 = 6*no2(i)-3
c       n3 = 6*no3(i)-3
c       fn(n1) = fn(n1) + x2
c       fn(n2) = fn(n2) + x2
c       fn(n3) = fn(n3) + x2
c     end if
c   end do
do i = 1 , tdof
  do j = 1 , tdof+1
    kzt(i,j) = 0.0d0
  end do
end do
write(*,*) 'tot ', tot
do i = 1, tot
  kz(i) = 0.0d0
  mz(i) = 0.0d0
end do
c Call CONNECT to calculate the connectivity matrix
call connect(nopel,no1,no2,no3)
j = 0
do i = node+1 , tdof , nproc
  j = j + 1
  if (i.eq.node+1) then
    colposp(j) = 1
    mk(j) = mm(i)
  else
```

```

        colposp(j) = colposp(j-1) + (io-mm(io)+1)
        mk(j) = mm(i)
    endif
    io = i
end do
colposp(j+1) = colposp(j) + (io-mm(io)+1)
ncolpn = j
do i = 1 , tdof
    dap(i) = 100
end do
c Calculate the stiffness matrix for each element
band = 0
do 60 i = 1 , nele
    n1 = no1(i)
    n2 = no2(i)
    n3 = no3(i)
c Calculate Bandwidth
    q1 = max0(n1,n2,n3)
    q2 = min0(n1,n2,n3)
    if (q2.eq.0) q2=min0(n1,n2)
    q1=6*q1
    q2=6*q2-5
    band=max0(band,q1-q2)
    t11 = t1(i)
    t22 = t2(i)
    t33 = t3(i)
    x1 = x(n1)
    x2 = x(n2)
    y1 = y(n1)
    y2 = y(n2)
    z1 = z(n1)

```

```

      z2 = z(n2)
      if (n3.ne.0) then
        x3=x(n3)
        y3=y(n3)
        z3=z(n3)
      else
        x3 = 0.0d0
        y3 = 0.0d0
        z3 = 0.0d0
      end if
      mpp = mat(i)
      m = abs(mpp)
c     if nodes per element is 3 then call membrane element routines
      if (nopel(i).eq.3) then
        call stiff(e(m),nu(m),x1,y1,z1,x2,y2,z2,x3,y3,z3,
*          t11,t22,t33,ke)
c          call mmat(rho(m),x1,x2,x3,y1,y2,y3,z1,z2,z3,t11,t22,t33,me)
          call mlump(rho(m),x1,x2,x3,y1,y2,y3,z1,z2,z3,t11,t22,t33,me)
      endif
c     If nodes per element is 2 then call axial bar routines
      if (nopel(i).eq.2) then
        lx = x2 - x1
        ly = y2 - y1
        lz = z2 - z1
        ln = (lx**2 + ly**2 + lz**2)**0.5
c          Area = t11
          call axstiff(e(m),t11,ln,lx,ly,lz,ke)
          call axmass(rho(m),t11,ln,lx,ly,lz,me)
      endif
c     Calculate Global Stiffness matrix using connectivity info.
      do 70 j = 1 , nopel(i)*dof1

```

```

do 80 k = 1 , j
  rowg = con(i,k)
  colg = con(i,j)
  if (rowg.gt.colg) then
    tie = rowg
    rowg = colg
    colg = tie
  endif
  if (mod(colg-1,nproc).eq.node) then
    do jj = 1 , nodn
      if (rowg.eq.dd(jj).or.colg.eq.dd(jj)) then
        me(k,j) = 0.0d0
        ke(k,j) = 0.0d0
        ke(j,k) = 0.0d0
        me(j,k) = 0.0d0
      endif
      if (rowg.eq.colg.and.rowg.eq.dd(jj).and.
*       dap(jj).ne.0) then
        me(k,j) = 1.0d0
        ke(k,j) = 1.0d0
        dap(jj) = 0
      endif
    end do
    add = 0
    if (mod(colg,nproc).ne.0) add = 1
    colgp = int(colg/nproc) + add
    rowgp = rowg
c   Membrane element assembly
    if (nopel(i).eq.3) then
      mz(colposp(colgp)+rowgp-mk(colgp)) = me(k,j) +
*      mz(colposp(colgp)+rowgp-mk(colgp))

```

```

        kz(colposp(colgp)+rowgp-mk(colgp)) = ke(k,j) +
*       kz(colposp(colgp)+rowgp-mk(colgp))
    endif
c     Axial Bar assembly
    if (nopel(i).eq.2) then
        mz(colposp(colgp)+rowgp-mk(colgp)) = me(k,j) +
*       mz(colposp(colgp)+rowgp-mk(colgp))
        kz(colposp(colgp)+rowgp-mk(colgp)) = ke(k,j) +
*       kz(colposp(colgp)+rowgp-mk(colgp))
    endif
    endif
80   continue
70   continue
    do j = 1 , nopel(i)*dof1
        do k = 1 , nopel(i)*dof1
            rowg = con(i,j)
            colg = con(i,k)
            kzt(rowg,colg) = kzt(rowg,colg) + ke(j,k)
        end do
    end do
60   continue
c     Eliminate any columns that are all zeroes
c     Use Known Displacements to eliminate columns and rows
c     call reduce(nodn,kz,mz,dd,kzp,mzp,mp,colposp,mk)
c     if (static.ne.0) then
c         len = colposp(ncolpn+1)-1
c         call dysetup(deltat,method,alpha,gamma,a1,a2,a3,a4,a5)
c         call khat(len,a3,kz,mz)
c     endif
    write(*,*) 'band =',band
c     Begin Active Column Solver

```

```

do i = 1 , nodn
  fn(dd(i)) = 0.0d0
end do
write(*,*) 'Here we go'
c   call gauss1(tdof,band,kzt,u,fn,err)
write(*,*) 'we are out'
write(*,*) (mk(i),i=1,4)
do i = 1 , tdof
  count = 0
  do j = colpos(i),colpos(i+1)-1
    k = mk(i)+(j-colpos(i))
    if (kzt(i,k).eq.0) count = count + 1
  end do
  if (count.eq.colpos(i+1)-colpos(i)) write(*,*) 'yep',i
end do
c   goto 1234
do i = 1 , nofn
  fp(i) = fn(df(i))
end do
c   Remember the RHS for dynamic purposes
do i = 1 , nofn
  fo(i) = fp(i)
end do
do i = 1 , ncolpn
  coltip(i) = mk(i)
end do
do i = 2 , ncolpn+1
  mk(i-1) = colposp(i) - colposp(i-1) - 1
end do
j = 0
do i = node+1 , nofn , nproc

```

```

        j = j + 1
        subfor(j) = fp(i)
    end do
c    call zerocol(nodn,nofn,dd,df,kz,mz,mm)
c    call zerocol(ncolpn,colposp,subfor,kz)
    if (node.eq.0) write(*,*) 'Made it'
c    Decompose the global stiffness matrix equally among the
c    different processes.
c    call prepare(node,nproc,nwksub,ncolpn,colval,colvalm,colpos1,
c    *          coltip,subfor,nofn,maxa,mk,kzp,mzp,fp)
c    Factor the portion assigned to this process using L D L-transpose.
    call factor(kz,colposp,coltip,diagon,work,nofn,nproc,
c    *          ncolpn,node,tids)
c    Load Restart Files if needed
    nlep = indtip - kroot1 + 1
c    if (irestart.eq.1.and.node.eq.0) call loares(nstart,nlep,ug,phig)
    if (node.eq.0) write(*,*) '3'
c    Perform Forward Substitution
    do iii = nstart , nstop
        write(*,*) 'ready'
        open(1,file='stiff.dat')
        write(1,*) nofn,nproc,ncolpn,node
        do i = 1 , ncolpn
            write(1,*) colposp(i),coltip(i)
        end do
        write(1,*) colposp(ncolpn+1)
        do i = 1 , colposp(nofn+1)
            write(1,*) kz(i)
        end do
        close(1)
    end do
13 continue

```

```

        write(*,*) 'go'
        call forsub(nofn,nproc,ncolpn,node,
*           kz,colposp,coltip,subfor,tids)
c       Perform Backward Substitution
        write(*,*) (subfor(i),i=50,55)
        call bacsub(nofn,nproc,ncolpn,node,kz,colposp,
*           subfor,work,tids)
        write(*,*) (subfor(i),i=50,55)
c       When ready, need to assemble global solution vector
        if (node.eq.0) then
            do i = 1 , nofn
                sol(i) = 0.0d0
            end do
c       Assemble the solution of node 0.
            do i = 1 , ncolpn
                k = 1+(nproc*(i-1))
                sol(k) = subfor(i)
            end do
c       Receive the rest of the solutions from the other processes
c       and assemble the displacement solution.
            if (nproc.gt.1) then
                do i = 1 , nproc-1
                    k = 1000
                    mess = 3*(i-1)
c       Receive solutions from other processes
c       and assemble global solution vector
                    call crecv(1000+mess,who,4)
                    call crecv(1001+mess,j,4)
                    call crecv(1002+mess,sol1,8*j)
                    do ii = 1 , j
                        k = who+1+(nproc*(ii-1))

```

```

        sol(k) = sol1(ii)
    end do
end do
endif
else
c   if not node 0 send the portion of the solutions this process
c   has calculated.
    mess = 3*(node-1)
    call csend(1000+mess,node,4,0,0)
    call csend(1001+mess,ncolpn,4,0,0)
    call csend(1002+mess,subfor,8*ncolpn,0,0)
endif
c   Arrange total solution vector for displacements.
if (node.eq.0) then
    do i = 1 , nofn
        u(df(i)) = sol(i)
    end do
c   First Calculate L.E. Pitch and Displacement then
c   send to Fluids Cube while receving Cp's
if (irestart.eq.1.and.iii.eq.nstart) then
    i = 1
else
c   call pad(phylen,ile,ite,jroot,jroot1,jtip,nxt,nyt,x,y,z,
c   *           xg,yg,zg,u,ug,phig)
endif
c   call fscp(node,nproc,jroot,jtip,phylen,cp,ug,phig)
c   Given CP's calculate loads on structural nodes
c   call nodload(iii,tdof,ile,ite,jroot,jtip,jroot1,dynfre,
c   *           no1,no2,no3,ele,cp,area,fn,x,y,z,xg,yg,zg)
do j = 1 , nofn
    fp(j) = fn(df(j))

```

```

        end do
        do j = 1 , nodn
            fp(dd(j)) = 0.0d0
        end do
    endif
c    call gsync()
c    If doing a dynamic analysis then calculate the new
c    force vector, Fhat, and do the backward sub. and forward
c    sub. again until time limit is met.
    count = count + 1
    time = time + deltat
c    if (time.gt.limit) goto 321
c    Calculate new velocities and accelerations
c    and calculate the augmented (?) force vector
c    for dynamic analysis.
c    if (static.ne.0.and.node.eq.0) then
c        call dynamic(nofn,a1,a2,a3,a4,a5,uold,sol,
c        *            velo,veln,acco,accn)
c        call fhats(nofn,a3,a4,a5,uold,velo,acco,mz,fo,fp,
c        *            mp,colposp)
c    endif
c    Pass global force vector to all if node 0, if
c    not node 0, receive global force vector.
    if (node.eq.0) then
        if (nproc.gt.1) then
            call csend(1500,fp,8*nofn,-1,0)
        endif
    else
        call crecv(1500,fp,8*nofn)
    endif
c    Calculate RHS needed for this process to be used

```

```

c      in LDL solver.
      icol = node + 1
      do i = 1 , ncolpn
        subfor(i) = fp(icol)
        icol = icol + nproc
      end do

c      Save Restart Files
c      if (mod(iii-1,nrsrt).eq.0.and.node.eq.0)
c      *          call savres(iii,nlep,ug,phig)
c      This end do goes with nstart , nstop loop
      end do

c      if (static.ne.0) goto 323
1234 continue
321 if (node.eq.0) then
      call postit(tn,nele,nxt,nyt,phylen,no1,no2,no3,
*          isurf,mat,x,y,z,u,nu,e)
      do i = 1 , tn
        x(i) = x(i) + u(6*i-5)
        y(i) = y(i) + u(6*i-4)
        z(i) = z(i) + u(6*i-3)
      end do
      call mesh1(tn,nax,nrib+nspar+nmem,nele,nopel,no1,no2,no3,isurf,
*          x,y,z)
      open(1,file='solution.dat')
      do i = 1 , tn
        write(1,*) u(6*i-3)
      end do
      close(1)
    endif
  stop
end

```

```
subroutine arw2wing(x,y,z)
c   This program reads in the data from data.file
c   which contains the joint numbers of the wing.
implicit none
integer i , j , index
real xx , yy , zz
real*8 x(1) , y(1) , z(1)
open(1,file='arw2.dat')
index = 0
do j = 1 , 17
  do i = 1 , 24
    read(1,*) xx,yy,zz
    if (j.ne.3.and.j.ne.5.and.j.ne.13.and.j.ne.15) then
      index = index + 1
      x(index) = dble(xx)
      y(index) = dble(yy)
      z(index) = dble(zz)
    end if
  end do
end do
close(1)
open(1,file='fsparinfo.dat')
do i = 8 , 296 , 24
  write(1,*) y(i),z(i)
end do
close(1)
open(1,file='rsparinfo.dat')
do i = 16 , 304 , 24
  write(1,*) y(i),z(i)
end do
close(1)
```

```

    return
end
c   Calculate Stiffness in Global System
    subroutine axmass(rho , a , l , lx , ly , lz , mat1)
    implicit none
    integer i , j
    real*8 a , l , mat(6,6) , k(3,3) , lx , ly , lz , thx ,
*     thy , thz , dir(3,3) , rho , mat1(18,18)
c   Angles of rotation about x,y,z
    thx = 0.0d0
    thy = atan2(lz,dsqrt(lx**2+ly**2))
    thz = atan2(ly,lx)
c   Corresponding transformation matrix
c   These are the direction cosines for a 3-d rotation.?
    dir(1,1) = lx/l
    dir(1,2) = ly/l
    dir(1,3) = lz/l
    dir(2,1) = sin(thz)*cos(thy)
    dir(2,2) = sin(thz)*sin(thy)*sin(thx)+cos(thz)*cos(thx)
    dir(2,3) = -sin(thz)*sin(thy)*cos(thx)+cos(thz)*sin(thx)
    dir(3,1) = sin(thy)
    dir(3,2) = -cos(thy)*sin(thx)
    dir(3,3) = cos(thy)*cos(thx)
c   Calculate stiffness matrix
c   specially for axial bar case
    k(1,1) = dir(1,1)**2
    k(1,2) = dir(1,1)*dir(1,2)
    k(1,3) = dir(1,1)*dir(1,3)
    k(2,1) = k(1,2)
    k(2,2) = dir(1,2)**2
    k(2,3) = dir(1,2)*dir(1,3)

```

```
k(3,1) = k(1,3)
k(3,2) = k(2,3)
k(3,3) = dir(1,3)**2
do 10 i = 1 , 3
  do 20 j = 1 , 3
    mat(i,j) = 2.0d0*k(i,j)*rho*a*1/6.0d0
    mat(i+3,j+3) = 2.0d0*k(i,j)*rho*a*1/6.0d0
    mat(i+3,j) = k(i,j)*rho*a*1/6.0d0
    mat(i,j+3) = k(i,j)*rho*a*1/6.0d0
  20 continue
10 continue
c This is to change to global matrix
do i = 1 , 3
  do j = 1 , 3
    mat1(i,j) = mat(i,j)
    mat1(i,j+3) = 0.0d0
    mat1(i+3,j) = 0.0d0
    mat1(i+3,j+3) = 0.0d0
    mat1(i,j+6) = mat(i,j+3)
    mat1(i,j+9) = 0.0d0
    mat1(i+3,j+6) = 0.0d0
    mat1(i+3,j+9) = 0.0d0
    mat1(i+6,j) = mat(i+3,j)
    mat1(i+6,j+3) = 0.0d0
    mat1(i+9,j) = 0.0d0
    mat1(i+9,j+3) = 0.0d0
    mat1(i+6,j+6) = mat(i+3,j+3)
    mat1(i+6,j+9) = 0.0d0
    mat1(i+9,j+6) = 0.0d0
    mat1(i+9,j+9) = 0.0d0
  end do
```

```

        end do
        return
    end
c    Calculate Stiffness in Global System
    subroutine axstiff(e , a , l , lx , ly , lz , mat1)
    implicit none
    integer i , j
    real*8 e , a , l , mat(6,6) , k(3,3) , lx , ly , lz ,
*    thx , thy , thz , dir(3,3) , mat1(18,18)
c    Angles of rotation about x,y,z
    thx = 0.0d0
    thy = atan2(lz,dsqrt(lx**2+ly**2))
    thz = atan2(ly,lx)
c    Corresponding transformation matrix
c    These are the direction cosines for a 3-d rotation.?
    dir(1,1) = lx/l
    dir(1,2) = ly/l
    dir(1,3) = lz/l
    dir(2,1) = sin(thz)*cos(thy)
    dir(2,2) = sin(thz)*sin(thy)*sin(thx)+cos(thz)*cos(thx)
    dir(2,3) = -sin(thz)*sin(thy)*cos(thx)+cos(thz)*sin(thx)
    dir(3,1) = sin(thy)
    dir(3,2) = -cos(thy)*sin(thx)
    dir(3,3) = cos(thy)*cos(thx)
c    Calculate stiffness matrix
c    specially for axial bar case
    k(1,1) = dir(1,1)**2
    k(1,2) = dir(1,1)*dir(1,2)
    k(1,3) = dir(1,1)*dir(1,3)
    k(2,1) = k(1,2)
    k(2,2) = dir(1,2)**2

```

```
k(2,3) = dir(1,2)*dir(1,3)
k(3,1) = k(1,3)
k(3,2) = k(2,3)
k(3,3) = dir(1,3)**2
do 10 i = 1 , 3
  do 20 j = 1 , 3
    mat(i,j) = k(i,j)*e*a/l
    mat(i+3,j+3) = k(i,j)*e*a/l
    mat(i+3,j) = -k(i,j)*e*a/l
    mat(i,j+3) = -k(i,j)*e*a/l
  20 continue
10 continue
c This is to change to global matrix
do i = 1 , 3
  do j = 1 , 3
    mat1(i,j) = mat(i,j)
    mat1(i,j+3) = 0.0d0
    mat1(i+3,j) = 0.0d0
    mat1(i+3,j+3) = 0.0d0
    mat1(i,j+6) = mat(i,j+3)
    mat1(i,j+9) = 0.0d0
    mat1(i+3,j+6) = 0.0d0
    mat1(i+3,j+9) = 0.0d0
    mat1(i+6,j) = mat(i+3,j)
    mat1(i+6,j+3) = 0.0d0
    mat1(i+9,j) = 0.0d0
    mat1(i+9,j+3) = 0.0d0
    mat1(i+6,j+6) = mat(i+3,j+3)
    mat1(i+6,j+9) = 0.0d0
    mat1(i+9,j+6) = 0.0d0
    mat1(i+9,j+9) = 0.0d0
```

```

        end do
    end do
    return
end
c    This routine calculates the integration of B matrix
    subroutine bmat(b,x1,x2,x3,y1,y2,y3,e,nu,t1,t12,t13)
    implicit none
    real*8 b(18,18) , x1 , x2 , x3 , y1 , y2 , y3 , x12 ,
*       x13 , x23 , y12 , y13 , y23 , area , ix ,
*       iy , ixy , ix2y , ixy2 , ix3 , iy3 , ix2 , iy2 ,
*       alpha , beta , gamma , aa , bb , cc , e , nu ,
*       t12 , t13 , t1 , fud
    x12 = x1 - x2
    x13 = x1 - x3
    x23 = x2 - x3
    y12 = y1 - y2
    y13 = y1 - y3
    y23 = y2 - y3
    aa = e/(1.0d0-(nu**2))
    bb = aa*nu
    cc = aa*(1.0d0-nu)*0.5d0
    fud = 1.0d0
    aa = aa*fud
    bb = bb*fud
    beta = (t12*y13-y12*t13)/(x12*y13-y12*x13)
    if (y12.eq.0.) then
        gamma = (-1.0d0*t13+beta*x13)/(-1.0d0*y13)
    else
        gamma = (-1.0d0*t12 + beta*x12)/(-1.0d0*y12)
    endif
    alpha = t1 - beta*x1 - gamma*y1

```

```

area = 0.5d0*(x2*y3 - x3*y2 + x3*y1 - x1*y3 + x1*y2 - x2*y1)
if (area.lt.0) then
  write(*,*) 'Messed up bmat'
  stop
endif
ix = (x1+x2+x3)*area/3.0d0
iy = (y1+y2+y3)*area/3.0d0
ixy = (x1*y1 + x2*y2 + x3*y3)*(area/6.0) + (x1*y2 + x1*y3 +
1      x2*y1 + x2*y3 + x3*y1 + x3*y2)*(area/12.0)
ix2 = (x1**2 + x2**2 + x3**2 + x1*x2 + x1*x3 + x2*x3)
1      *area/6.0
iy2 = (y1**2 + y2**2 + y3**2 + y1*y2 + y2*y3 + y1*y3)*area/6.0
ix2y = ((x1**2)*y1 + (x2**2)*y2 + (x3**2)*y3)*(area/10.0) +
1      ((x2**2)*y1 + (x3**2)*y1 + (x1**2)*y2 + (x3**2)*y2 +
2      (x1**2)*y3 + (x2**2)*y3)*(area/30.0) + (x1*x2*y1 +
3      x1*x3*y1 + x1*x2*y2 + x2*x3*y2 + x1*x3*y3 + x2*x3*y3)*
4      (area/15.0) + (x2*x3*y1 + x1*x3*y2 + x1*x2*y3)*(area/30.0)
ixy2 = ((y1**2)*x1 + (y2**2)*x2 + (y3**2)*x3)*(area/10.0) +
1      ((y2**2)*x1 + (y3**2)*x1 + (y1**2)*x2 + (y3**2)*x2 +
2      (y1**2)*x3 + (y2**2)*x3)*(area/30.0) + (y1*y2*x1 +
3      y1*y3*x1 + y1*y2*x2 + y2*y3*x2 + y1*y3*x3 + y2*y3*x3)*
4      (area/15.0) + (y2*y3*x1 + y1*y3*x2 + y1*y2*x3)*(area/30.0)
ix3 = ((x1**3) + (x2**3) + (x3**3) + x1*(x2**2) + x1*(x3**2) +
1      (x1**2)*x2 + x2*(x3**2) + (x1**2)*x3 + (x2**2)*x3 +
2      x1*x2*x3)*(area/10.0)
iy3 = ((y1**3) + (y2**3) + (y3**3) + y1*(y2**2) + y1*(y3**2) +
1      (y1**2)*y2 + y2*(y3**2) + (y1**2)*y3 + (y2**2)*y3 +
2      y1*y2*y3)*(area/10.0)
call clr(b)
b(1,1) = aa*alpha*area + aa*beta*ix + aa*gamma*iy
b(1,2) = aa*alpha*iy + aa*beta*ixy + aa*gamma*iy2

```

```

b(1,3) = bb*alpha*area + bb*beta*ix + bb*gamma*iy
b(1,4) = bb*alpha*ix + bb*beta*ix2 + bb*gamma*ixy
b(1,5) = 0.0d0
b(2,1) = b(1,2)
b(2,2) = aa*alpha*iy2 + aa*beta*ixy2 + aa*gamma*iy3 +
*      cc*alpha*ix2 + cc*beta*ix3 + cc*gamma*ix2y
b(2,3) = bb*alpha*iy + bb*beta*ixy + bb*gamma*iy2
b(2,4) = (bb+cc)*(alpha*ixy + beta*ix2y + gamma*ixy2)
b(2,5) = -cc*alpha*ix - cc*beta*ix2 - cc*gamma*ixy
b(3,1) = b(1,3)
b(3,2) = b(2,3)
b(3,3) = aa*alpha*area + aa*beta*ix + aa*gamma*iy
b(3,4) = aa*alpha*ix + aa*beta*ix2 + aa*gamma*ixy
b(3,5) = 0.0d0
b(4,1) = b(1,4)
b(4,2) = b(2,4)
b(4,3) = b(3,4)
b(4,4) = aa*alpha*ix2 + aa*beta*ix3 + aa*gamma*ix2y +
*      cc*alpha*iy2 + cc*beta*ixy2 + cc*gamma*iy3
b(4,5) = -cc*alpha*iy - cc*beta*ixy - cc*gamma*iy2
b(5,1) = b(1,5)
b(5,2) = b(2,5)
b(5,3) = b(3,5)
b(5,4) = b(4,5)
b(5,5) = cc*alpha*area + cc*beta*ix + cc*gamma*iy
return
end
subroutine bound(nodn,nofn,dd,df,u,fn)
include 'wboxbc.f'
c   This routine calculates the boundary conditions
c   on the wing box structure

```

```

integer i , j , nodn , nofn , dd(1) , df(1) , n1 ,
*      n2 , n3 , ma(18) , count
real*8 u(1) , fn(1) , fs(6) , fe(9) ,
*      t(18,18) , x1 , x2 , x3 , len , dx ,
*      y1 , y2 , y3 , z1 , z2 , z3 , x3p , x2p , y3p , y2p ,
*      dum(18) , tt(18,18) , x1p , y1p
do i = 1 , tdof
  u(i) = 0.0d0
  fn(i) = 0.0d0
end do
c      Calculate number of forces known and displacements known
nodn = dof1*2*(nxx+2)
j = 0
do i = 1 , 2*(nxx+2)
  do k = 6*i-5,6*i
    j = j + 1
    dd(j) = k
  end do
end do
nofn = tdof
do i = 1 , tdof
  df(i) = i
end do
c      Tip Load
c      do i = 2*(nxx+2)*(nyy+1)+1, tn-1, 2
c        fn(6*i-3) = 100.0d0
c      end do
i = 303
fn(6*i-3) = -100.0d0
c      i = (nxx+2)*2*(nyy+1) + 7
c      i = (nxx+2)*2*(nyy) + 7

```

```

c      Twist l.e. up
c      fn(6*i-3) = 1.0d0
c      fn(6*(i+9)-3) = -1.0d0
c      Twist l.e. down
c      fn(6*(i+1)-3) = -1.0d0
c      fn(6*(i+8)-3) = 1.0d0
      return
      end
c      This routine clears a matrix
      subroutine clr(a)
      implicit none
      integer i , j
      real*8 a(18,18)
      do 10 i = 1 , 18
         do 20 j = 1 , 18
            a(i,j) = 0.0d0
20      continue
10      continue
      return
      end
c      This routine calculates a c matrix
      subroutine cmat(c,x1,x2,x3,y1,y2,y3)
      implicit none
      integer i , j
      real*8 c(18,18) , x1 , x2 , x3 , y1 , y2 , y3 ,
*      x12 , x13 , x23 , y12 , y13 , y23 , area
      area = 0.5d0*(x2*y3 - x3*y2 + x3*y1 - x1*y3 + x1*y2 - x2*y1)
      if (area.lt.0) write(*,*) 'wo baby'
      x12 = x1 - x2
      x13 = x1 - x3
      x23 = x2 - x3

```

```
y12 = y1 - y2
y13 = y1 - y3
y23 = y2 - y3
c(1,1) = y23
c(1,2) = 0.0d0
c(1,3) = 0.5d0*y1*y23
c(1,4) = -y13
c(1,5) = 0.0d0
c(1,6) = -0.5d0*y2*y13
c(1,7) = y12
c(1,8) = 0.0d0
c(1,9) = 0.5d0*y3*y12
c(2,1) = 0.0d0
c(2,2) = 0.0d0
c(2,3) = -0.5d0*y23
c(2,4) = 0.0d0
c(2,5) = 0.0d0
c(2,6) = 0.5d0*y13
c(2,7) = 0.0d0
c(2,8) = 0.0d0
c(2,9) = -0.5d0*y12
c(3,1) = 0.0d0
c(3,2) = -x23
c(3,3) = 0.5d0*x1*x23
c(3,4) = 0.0d0
c(3,5) = x13
c(3,6) = -0.5d0*x2*x13
c(3,7) = 0.0d0
c(3,8) = -x12
c(3,9) = 0.5d0*x3*x12
c(4,1) = 0.0d0
```

```

c(4,2) = 0.0d0
c(4,3) = -0.5d0*x23
c(4,4) = 0.0d0
c(4,5) = 0.0d0
c(4,6) = 0.5d0*x13
c(4,7) = 0.0d0
c(4,8) = 0.0d0
c(4,9) = -0.5d0*x12
c(5,1) = -x23
c(5,2) = y23
c(5,3) = 0.5d0*(-x1*y23 - y1*x23)
c(5,4) = x13
c(5,5) = -y13
c(5,6) = 0.5d0*(x2*y13 + y2*x13)
c(5,7) = -x12
c(5,8) = y12
c(5,9) = 0.5d0*(-x3*y12 - y3*x12)
do 80 i = 1 , 5
  do 90 j = 1 , 9
    c(i,j) = 0.5d0*c(i,j)/(area)
80 continue
90 continue
return
end
c routine calculated a connectivity matrix used to
c determine colpos and m. Where colpos determines the
c column position in a one-dimensional array storage of
c a symmetric matrix and m is the column heights. This
c routine assume there are 6 d.o.f. per node.
subroutine connect(nopel,no1,no2,no3)
include 'wboxbc.f'

```

```

integer nopel(1) , no1(1) , no2(1) , no3(1) ,
*      n(5) , i , j , k , mk(tdof) ,
*      mini , mini1 , aa ,
*      con , colpos , m
common /soup/ con(nele,dof) , colpos(1) , m(1)
c      Need tdof , nele , dof1
c      nopel(i):  nodes per element i
c      no1(i), no2(i), no3(i), no4(i):  nodes 1,2,3,and 4 of element i
do i = 1 , tdof
  m(i) = tdof+1
  colpos(i) = 0
end do
do i = 1 , nele
  n(1) = no1(i)
  n(2) = no2(i)
  n(3) = no3(i)
  mini1 = tdof+1
  aa = 0
  do j = 1 , nopel(i)
    do k = dof1*n(j)-(dof1-1) , dof1*n(j)
      aa = aa + 1
      con(i,aa) = k
      mini = min0(con(i,aa),mini1)
      mini1 = mini
    end do
  end do
  do j = 1 , nopel(i)*dof1
    if (m(con(i,j)).gt.mini) m(con(i,j)) = mini
  end do
end do
colpos(1) = 1

```

```

do i = 2 , tdof+1
    mk(i-1) = (i-1) - m(i-1)
    colpos(i) = colpos(i-1) + mk(i-1) + 1
end do
return
end
subroutine crecv(a,b,c)
integer a , c
real*8 b(1)
return
end
subroutine cross(a,b,c)
c This routine calculates the cross product of a x b and puts the
c result in c.
implicit none
real*8 a(3) , b(3) , c(3)
c(1) = a(2)*b(3)-a(3)*b(2)
c(2) = a(3)*b(1)-a(1)*b(3)
c(3) = a(1)*b(2)-a(2)*b(1)
return
end
subroutine csend(a,b,c,d,e)
integer a , c , d , e
real*8 b(1)
return
end
subroutine extract(x,y,x1,y1,tot,k,l,nx,ny)
implicit none
integer i , k , tot , nx , ny , l , j , d1 , d2 , nyy , kk
real*8 x1(1) , y1(1) , x(1) , y(1)
c if l = 0 means it's the first partition

```

```

nyy = 0
if (l.eq.0) nyy = ny+2
j = 1
kk = tot
do i = tot+1 , tot+k
  d1 = (j-1)*(nx+2) + 1
  d2 = j*(nx+2)
  if (l.eq.0) goto 10
  if (i-tot.eq.d1.or.i-tot.eq.d2) goto 20
10  kk = kk + 1
    x1(kk) = x(i-tot)
    y1(kk) = y(i-tot)
20  if (real(i-tot)/real(nx+2).eq.int(real(i-tot)/real(nx+2)))
*    j = j + 1
  end do
  tot = tot + k - 2*(ny+2-nyy)
  return
end
subroutine mesh(tn,nax,ntrian,nele,nopel,no1,no2,no3,isurf,
*             x,y,z)
  implicit none
  integer tn , ntrian , nele , nopel(1) , no1(1) , index ,
*         no2(1) , no3(1) , isurf(1) , i , nax , nsp , j
  real*8 x(1) , y(1) , z(1) ,
* dx , dy ,dz , dx1 , dy1, dz1 , ddx , ddy , ddz
  ddx = 49.4684d0-x(119)
  ddy = 49.0002d0 - y(119)
  ddz = -0.0555252d0 - z(119)
  write(*,*) 'ddz ',ddz,ddy,ddx
c    dx1 = 74.0921d0 - x(311)
c    dy1 = 111.19d0 - y(311)

```

```

c      dz1 = 0.0870121d0 - (z(311)+ddz)
c      dy = y(311)-y(119)
      open(unit=1,file='mesh.rigid.data')
      write(*,*) ntrian , nele-nax
      write(1,*) tn , ntrian , 0
      do i = 1 , tn
        if (i.le.120) then
          x(i) = x(i)+ddx
          y(i) = y(i)+ddy
          z(i) = z(i)+ddz
        else
          x(i) = x(i)+ddx
          y(i) = y(i)+ddy
          z(i) = z(i)+ddz
c      z(i) = z(i)+ddz+(((y(int((i-1)/24)*24+23)-y(119))/dy)*dz1)
        end if
      end do
      write(1,*) (x(i),i=1,tn),
*      (y(i),i=1,tn),
*      (z(i),i=1,tn),
*      (no1(i),no2(i),no3(i),i=nax+1,nele),
*      (isurf(i)+2,i=nax+1,nele)
c      write(1,*) (x(i)+ddx,i=1,120),(x(i)+ddx,i=121,tn),
c      *      (y(i)+ddy,i=1,120),(y(i)+ddy,i=121,tn),
c      *      (z(i)+ddz,i=1,120),
c      *      (z(i)+ddz+(((y(int((i-1)/24)*24+23)-y(119))/dy)*dz1),i=121,tn),
c      *      (no1(i),no2(i),no3(i),i=nax+1,nele),
c      *      (isurf(i)+2,i=nax+1,nele)
      close(unit=1)
      open(1,file='spline.pts')
      open(2,file='splinepts.map')

```

```

nsp = 42
write(1,*) nsp
write(2,*) nsp
do j = 1 , 13 , 2
  do i = 1 , 23 , 2
    if (i.ne.3.and.i.ne.7.and.i.ne.11.and.i.ne.13.and.
*      i.ne.17.and.i.ne.21) then
      index = (j-1)*24 + i
      write(1,*) x(index) , y(index)
      write(2,*) 6*index-3
    end if
  end do
end do
close(2)
close(1)
open(1,file='struc.grid')
write(1,*) tn
do i = 1 , tn
  write(1,*) i,x(i),y(i),z(i)
end do
close(1)
return
end
subroutine mesh1(tn,nax,ntrian,nele,nopel,no1,no2,no3,isurf,
*              x,y,z)
implicit none
integer tn , ntrian , nele , nopel(1) , no1(1) ,
*      no2(1) , no3(1) , isurf(1) , i , nax
real*8 x(1) , y(1) , z(1)
open(unit=1,file='mesh.flex.data')
write(*,*) ntrian , nele-nax

```

```

write(1,*) tn , ntrian , 0
write(1,*) (x(i),i=1,tn),(y(i),i=1,tn),(z(i),i=1,tn),
*         (no1(i),no2(i),no3(i),i=nax+1,nele),
*         (isurf(i)+2,i=nax+1,nele)
close(unit=1)
return
end
subroutine mlump(rho,x1p,x2p,x3p,y1p,y2p,y3p,z1p,z2p,z3p,
*             t1,t2,t3,m)
implicit none
integer i , j , k
real*8 x1 , x2 , x3 , y1 , y2 , y3 , t1 ,t2 , t3 ,
*     m(18,18) , g12 , g23 , g31 , h12 , h23 , h31 ,
*     area , rho , a1 , a2 , a3 , a4 , a5 , a6 ,
*     a7 , a8 , a9 , x1p , x2p , x3p , y1p , y2p , y3p ,
*     z1p , z2p , z3p , tr(18,18) , trt(18,18) ,
*     tmat(18,18)
c   Change 3-d coordinates to 2-d coordinates
call trans(x1p,x2p,x3p,y1p,y2p,y3p,z1p,z2p,z3p,x1,x2,x3,
*         y1,y2,y3)
area = 0.5d0*(x2*y3-x3*y2+x3*y1-x1*y3+x1*y2-x2*y1)
call clr(m)
do i = 1 , 18
  if (mod(i,6).le.3) m(i,i) = rho*area*t1*0.333333333d0
end do
return
end
subroutine multi(a,b,c,l,m,n)
implicit none
integer i , j , k , l , m , n
real*8 a(18,18) , b(18,18) , c(18,18)

```

```

do i = 1 , l
  do j = 1 , n
    c(i,j) = 0.0d0
    do k = 1 , m
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
    end do
  end do
end do
return
end
subroutine part(nx,ny,xlr,xlt,xtr,xtt,ylr,ylt,ytr,ytt,x,y,
*kt)
c   This subroutine takes a planform wing and divides it
c   equally into several parts for finite element wing
c   box modeling
implicit none
integer i , j , k , kk , nx , ny , kt
real*8 xlr , xlt , xtr , xtt , ylr , ylt , ytr , ytt ,
*   x(1) , y(1) , xl , xt , yl , yt , d1 , d2 ,
*   d3 , d4 , d5 , d6 , r1 , r2 , r3 , r4 , r5 , r6 ,
*   sx , sy , s1 , s2 , s3 , s4 , s5 , s6 , alpha ,
*   beta , m , xlrp , xtrp , xttp , xltp , ylrp ,
*   yltp , ytrp , yttp
c   Use these coefficients to determine the reduction in
c   chord(alpha) and spanwise(beta) directions from the
c   original wing planform to wing box planform. Also
c   gamma is for tip .
alpha = 0.0d0
beta = 0.0d0
c   alpha = 0.20d0
c   beta = 0.10d0

```

```
c    Reduction based on beta
      m = (xlr-xlt)/(ytr-ylt)
      d1 = ylt - ytr
      r1 = (1.0d0-beta)*d1
      ylt = ytr + r1
      xlt = m*(ytr-ytr) + xlr
      m = (xtt-xtr)/(ytt-ytr)
      d2 = ytt - ytr
      r2 = (1.0d0-beta)*d2
      ytt = ytr + r2
      xtt = m*(ytt-ytr) + xtr
c    Reduction based on alpha
      m = (ytr-ytt)/(xtr-xtt)
      d3 = xtt - xtr
      r3 = alpha * d3 * 0.5d0
      xltp = xtr + r3
      xttp = xtt - r3
      yltp = m*(xltp-xtr) + ytr
      yttp = m*(xttp-xtr) + ytt
      m = (ytr-ytt)/(xtr-xtr)
      d4 = xtr - xlr
      r4 = alpha*d4*0.5d0
      xtrp = xtr - r4
      xlrp = xlr + r4
      ytrp = m*(xtrp-xlr) + ytr
      ylrp = m*(xlrp-xlr) + ytr
      xlt = xltp
      xtt = xttp
      ylt = yltp
      ytt = yttp
      xtr = xtrp
```

```
xlr = xlrp
ytr = ytrp
ylr = ylrp
c Calculate Step Size in x and y directions
sx = 1.0d0/dfloat(1+nx)
sy = 1.0d0/dfloat(1+ny)
d1 = ylt - ylr
d2 = xlt - xlr
d3 = ytt - ytr
d4 = xtt - xtr
s1 = sy * d1
s2 = sy * d2
s3 = sy * d3
s4 = sy * d4
c Start with leading edge root, progress toward the trailing
c edge and repeat from new leading edge root coordinate.
xl = xlr
xt = xtr
yl = ylr
yt = ytr
xl = xl - s2
xt = xt - s4
yl = yl - s1
yt = yt - s3
do i = 1 , ny+2
  k = (nx+2)*(i-1) + 1
  yl = yl + s1
  xl = xl + s2
  yt = yt + s3
  xt = xt + s4
  x(k) = xl
```

```

        y(k) = y1
        x(k+nx+1) = xt
        y(k+nx+1) = yt
        kk = k
        do j = 1 , nx
            kk = kk + 1
            d5 = yt - y1
            d6 = xt - x1
            s5 = sx * d5
            s6 = sx * d6
            x(kk) = x(kk-1) + s6
            y(kk) = y(kk-1) + s5
        end do
    end do
    kt = (nx+2)*(ny+2)
    return
end
subroutine postit(tn,nele,nx,ny,phylen,no1,no2,no3,isurf,mat,xx,
*             yy,zz,u,nu,e)
    implicit none
    integer i , j , k , l , no1(1) , no2(1) , tn ,
*         no3(1) , isurf(1) , nele , mat(1) ,
*         nx , ny , ii
    real*8 xx(1) , yy(1) , zz(1) , u(1) , phylen ,
*         nu(1) , e(1) , ddz , ddy , ddx , xl(400) ,
*         yl(400) , zl(400)
    character*132 msgline
    open(unit=1,file='strinfo.dat')
    write(1,90) nx , ny , tn
90 format(2x,i3,2x,i3,2x,i4)
c     call cwrite(1,msgline,17)

```

```
        write(1,100) nele
c      call cwrite(1,msgline,11)
100   format(2x,i8)
        write(1,110) phyllen
110   format(2x,f20.15)
c      call cwrite(1,msgline,23)
        do i = 1 , nele
            write(1,120) no1(i),no2(i),no3(i),isurf(i),mat(i)
120   format(2x,i3,2x,i3,2x,i3,2x,i2,2x,i2)
c      call cwrite(1,msgline,24)
        end do
        do i = 1 , tn
            write(1,130) xx(i) , yy(i) , zz(i)
130   format(3(f20.15,2x))
c      call cwrite(1,msgline,67)
        end do
        do i = 1 , 6*tn
            write(1,140) u(i)
140   format(f20.15)
c      call cwrite(1,msgline,21)
        end do
        do i = 1 , 4
            write(1,150) nu(i) , e(i)
150   format(f20.15,2x,f20.10)
c      call cwrite(1,msgline,43)
        end do
        close(unit=1)
        ddz = -58.65872420512695d0
        ddy = -1.420301708984375d0
        ddx = -233.2096090332031d0
c      Read Lloyd's files
```

```

open(1,file='eldred.data')
do i = 1 , 312
    read(1,*) ii,xl(i),yl(i),zl(i),j
end do
close(1)
open(1,file='def1.dat')
write(1,*) 'Zone T="Front Spar" I=13 F=Point'
do i = 7 , 295 , 24
c    write(1,*) yy(i)-ddy,u(6*i-3),zl(i)
    write(1,*) yy(i)-ddy,u(6*i-3)
end do
write(1,*) 'Zone T="Rear Spar" I=13 F=Point'
do i = 15 , 303 , 24
c    write(1,*) yy(i)-ddy,u(6*i-3),zl(i)
    write(1,*) yy(i)-ddy,u(6*i-3)
end do
write(1,*) 'Zone T="Auxi Spar" I=13 F=Point'
do i = 19 , 307 , 24
c    write(1,*) yy(i)-ddy,u(6*i-3),zl(i)
    write(1,*) yy(i)-ddy,u(6*i-3)
end do
close(1)
open(1,file='twist.dat')
write(1,*) 'Zone T="Twist" I=13 F=Point'
do i = 7 , 295 , 24
    j = i + 8
    write(*,*) i , xx(i)-xx(j)
    write(1,*) yy(i)-ddy,datan2(u(6*i-3)-u(6*j-3),
*        dabs(xx(i)-xx(j)))
end do
close(1)

```

```
    return
  end
c   This subroutine sorts an array of length k
c   where nx and ny represent TOTAL grid size
  subroutine sort(x,y,k,nx,ny)
  implicit none
  integer i , j , k , l , m , s , nx , ny
  real*8 x(1) , y(1) , temp , temp1
  do i = 1 , k
    do j = i , k
      if (y(i).gt.y(j)) then
        temp = x(i)
        temp1 = y(i)
        x(i) = x(j)
        y(i) = y(j)
        x(j) = temp
        y(j) = temp1
      endif
    end do
  end do
  do i = 1 , ny+2
    s = (i-1)*(nx+2) + 1
    do j = s , s+nx+1
      do l = j , s+nx+1
        if (x(j).gt.x(l)) then
          temp = x(j)
          temp1 = y(j)
          x(j) = x(l)
          y(j) = y(l)
          x(l) = temp
          y(l) = temp1
        endif
      end do
    end do
  end do
```

```

        endif
    end do
end do
end do
return
end
c   This subroutine calculates the stiffness matrix of
c   a membrane element with drilling degrees of freedom
c   subroutine stiff(e,nu,x1p,y1p,z1p,x2p,y2p,z2p,x3p,y3p,z3p,
*           t1,t2,t3,kr)
    implicit none
    integer i , j , k
    real*8 e , nu , x1 , y1 , x2 , y2 , x3 , y3 , t1 , t2 ,
*       t3 , kr(18,18) , c(18,18) , ct(18,18) ,
*       b(18,18) , tmat(18,18) , lx , ly , x1p , x2p , x3p ,
*       y1p , y2p , y3p , z1p , z2p , z3p , tr(18,18) ,
*       trt(18,18) , t12 , t13 , ee
    t12 = t1 - t2
    t13 = t1 - t3
    ee = e
    if (e.lt.0) e = -e
c   Change 3-d coordinates to 2-d coordinates
    call trans(x1p,x2p,x3p,y1p,y2p,y3p,z1p,z2p,z3p,x1,x2,x3,
*           y1,y2,y3)
    call clr(kr)
    call clr(b)
    call clr(c)
    call clr(ct)
    call clr(tmat)
    if (x1.lt.x2.and.x1.lt.x3) then
        lx = x1

```

```
else if (x2.lt.x3) then
  lx = x2
else
  lx = x3
endif
if (y1.lt.y2.and.y1.lt.y3) then
  ly = y1
else if (y2.lt.y3) then
  ly = y2
else
  ly = y3
endif
x1 = x1 - lx
x2 = x2 - lx
x3 = x3 - lx
y1 = y1 - ly
y2 = y2 - ly
y3 = y3 - ly
call bmat(b,x1,x2,x3,y1,y2,y3,e,nu,t1,t12,t13)
call cmat(c,x1,x2,x3,y1,y2,y3)
call transpose(c,ct,5,9)
call multi(b,c,tmat,5,5,9)
call multi(ct,tmat,kr,9,5,9)
if (ee.eq.0) then
  do i = 1 , 9
    do j = 1 , 9
      if (mod(i,3).ne.0.and.mod(j,3).ne.0) then
        kr(i,j) = 0.0d0
      end if
    end do
  end do
end do
```

```

    end if
c    Now calculate global stiffness matrix, and first
c    calculate the transformation matrix, tr
    call tmax(x1p,x2p,x3p,y1p,y2p,y3p,z1p,z2p,z3p,tr)
    call transpose(tr,trt,9,18)
    call multi(kr,tr,tmat,9,9,18)
    call multi(trt,tmat,kr,18,9,18)
    return
end
subroutine tmax(x1,x2,x3,y1,y2,y3,z1,z2,z3,tr)
implicit none
integer i , j , k
real*8 x1 , x2 , x3 , y1 , y2 , y3 , z1 , z2 , z3 ,
*      tr(18,18) , l(3) , m(3) , n(3) , xb(3) , rb(3) ,
*      yb(3) , zb(3) , d12 , d13 , sum , sum1
xb(1) = x2 - x1
xb(2) = y2 - y1
xb(3) = z2 - z1
rb(1) = x3 - x1
rb(2) = y3 - y1
rb(3) = z3 - z1
d12 = dsqrt(xb(1)**2 + xb(2)**2 + xb(3)**2)
d13 = dsqrt(rb(1)**2 + rb(2)**2 + rb(3)**2)
call cross(xb,rb,zb)
sum = dsqrt(zb(1)**2 + zb(2)**2 + zb(3)**2)
call cross(zb,xb,yb)
sum1 = dsqrt(yb(1)**2 + yb(2)**2 + yb(3)**2)
do i = 1 , 3
    xb(i) = xb(i)/d12
    rb(i) = rb(i)/d13
    zb(i) = zb(i)/sum

```

```
        yb(i) = yb(i)/sum1
    end do
c      The above has calculated unit vectors in local coordiantes
c      xb , yb , zb. Now calculate the direction cosines l, m, n
c      as discussed in Yang's F.E. Structural Analysis (pg. 81)
    l(1) = xb(1)
    m(1) = xb(2)
    n(1) = xb(3)
    l(2) = yb(1)
    m(2) = yb(2)
    n(2) = yb(3)
    l(3) = zb(1)
    m(3) = zb(2)
    n(3) = zb(3)
    do i = 1 , 9
        do j = 1 , 18
            tr(i,j) = 0.0d0
        end do
    end do
    tr(1,1) = l(1)
    tr(1,2) = m(1)
    tr(1,3) = n(1)
    tr(2,1) = l(2)
    tr(2,2) = m(2)
    tr(2,3) = n(2)
    tr(3,4) = l(3)
    tr(3,5) = m(3)
    tr(3,6) = n(3)
    do i = 1 , 3
        do j = 1 , 6
            tr(i+3,j+6) = tr(i,j)
        end do
    end do
```

```

        tr(i+6,j+12) = tr(i,j)
    end do
end do
return
end
subroutine trans(x1,x2,x3,y1,y2,y3,z1,z2,z3,x1p,x2p,x3p,
*           y1p,y2p,y3p)
c  this routine transforms a triangle in the 3-d plane to
c  an equivalent triangle in the x-y plane.
implicit none
integer i
real*8 x1 , x2 , x3 , y1 , y2 , y3 , z1 , z2 , z3 ,
1      d12 , d13 , theta , l12(3) , l13(3) , sum , sum1 ,
2      sum2 , x1p , x2p , x3p , y1p , y2p , y3p ,
3      tr(18,18) , trt(18,18) , res(9)
l12(1) = x2 - x1
l12(2) = y2 - y1
l12(3) = z2 - z1
l13(1) = x3 - x1
l13(2) = y3 - y1
l13(3) = z3 - z1
sum = 0.0d0
sum1 = 0.0d0
sum2 = 0.0d0
do i = 1 , 3
    sum = sum + l12(i)**2
    sum1 = sum1 + l13(i)**2
    sum2 = sum2 + l12(i)*l13(i)
end do
d12 = dsqrt(sum)
d13 = dsqrt(sum1)

```

```

theta = dacos(sum2/(d12*d13))
x1p = 0.0d0
x2p = d12
x3p = d13*dcos(theta)
y1p = 0.0d0
y2p = 0.0d0
y3p = d13*dsin(theta)
return
end
c This routine calculates the transpose, t12it,
c of a given matrix, t12i
SUBROUTINE transpose(t12i , t12it, m , n)
implicit none
integer i , j , m , n
real*8 t12i(18,18) , t12it(18,18)
do 20 i = 1 , n
  do 30 j = 1 , m
    t12it(i,j) = t12i(j,i)
30  continue
20  continue
return
end
c This is the include file wboxbc.f.
c FILE INCLUDED AT BEGINNING DESCRIBING WING PARTITIONING
c Set nxx and nyy, make sure it has same values as nxt and
c nyt in wing.f, i.e., nxx and nyy should be the total number
c of divisions in the x and y direction respectively.
integer nxx , nyy , tn , nax , nmem , nrrib , nspar ,
*       nele , tdof , nmat , dof , nodcube , dof1 , nogn ,
*       noddn , sloc , rloc , naxchord
real*8 phylen , dynfre

```

```
c    Partitions of Rectangular wing in x direction (nxx) and
c    y-direction (nyy), keep nxx odd due to middle spar.
      parameter ( nxx = 10 )
      parameter ( nyy = 11 )
c    Total Nodes
      parameter ( tn = 2*(nxx+2)*(nyy+2) )
c    Total number of axial bars (nax), spars (nspar),
c    ribs (nrrib), membrane elements (nmem) which make
c    up the upper and lower surface.
      parameter ( nax = 2*(2)*(nyy+1) + 2*4 )
      parameter ( nspar = (nyy+1)*2*(2) + 4*2 + 2*(nyy+1)*2 )
      parameter ( nrrib = (nxx+1)*2*12 )
      parameter ( naxchord = 0 )
      parameter ( nmem = (nxx+1)*(nyy+1)*2*2 )
c    Total number of elements (nele)
      parameter ( nele = nax + nspar + nrrib + nmem + naxchord )
c    Total d.o.f. (tdof), number of forces known (nogn),
c    total possible materials used (nmat) ,
c    total d.o.f. per Allman's element (dof),
c    total d.o.f. per axial bar (dof1).
      parameter ( tdof = 6*tn )
      parameter ( nogn = tdof )
      parameter ( nmat = 4 )
      parameter ( dof = 18 )
      parameter ( dof1 = 6 )
c    If using multiple processors, how many (nodcube),
c    and the total number of displacements to be solved for (noddn)
      parameter ( nodcube = 1 )
      parameter ( noddn = tdof - nogn )
c    The physical length (phylen) by which coordinates
c    have been scaled, and the dynamic free stream
```

```

c      dynamic pressure (dynfre).
c      parameter ( phyllen = 200.0d0 )
c      parameter ( phyllen = 1.0d0 )
c      parameter ( dynfre = 0.5d0 )
c      subroutine wingp(nxt,nyt,nnodes,n1,n2,n3,mat,xc,yc,zc,
*          t1,t2,t3,e,nu,rho,isurf)
c      when partitioning a wing more than once, either nx or
c      ny must remain constant during the entire process or
c      there will be discontinuities in the domain.
c      i,j,k      =      dummy counters
c      tot        =      1/2 the total number of nodes after
c                  partitioning
c      nx         =      number of partitions in x-direction
c      ny         =      number of partitions in y-direction
c      nxt        =      total partitions in x-direction
c      nyt        =      total partitions in y-direction
c      st         =      1 less than the total number of times a wing
c                  has been partitioned.
c      isurf      =      0 if not upper or lower surface
c                  1 if upper surface
c                  -1 if lower surface
c      Wing is assumed as follows:
c      Wing is viewed from above.
c      Leading edge is on top.
c      Trailing edge in on bottom.
c      Root is on left.
c      Tip is on right.
c      x1         =      x coordinate of top left point of wing
c      x2         =      x coordinate of top right point of wing
c      x3         =      x coordinate of bottom left point of wing
c      x4         =      x coordinate of bottom right point of wing

```

```

c   y1..y4    =      y coordinates (same as x)
c   These coordinate do not necessarily define the wing.  They define
c   the portions of which one wishes to partion the wing.  So a wing
c   can be divided in 9 parts by setting nx = ny = 3.  And if more
c   divisions are needed around quater chord of the wing, new
c   coordinates may be chosen as to divide the wing even further.
c   To avoid discontinuities in the structure, either nx or ny
c   must be held constant during the entire process.
c   x(500)    =      x coordinates of nodes of partitioned wing
c   y(500)    =      y coordinates of nodes of partitioned wing
c   xx(500)   =      x coordinates of nodes of global wing after part.
c   yy(500)   =      y coordinates of nodes of global wing after part.
c   fr        =      fraction of chord where more lines are needed
c                   in the spanwise direction
c   zu(500)   =      upper surface z coordinates after partitioning
c   zl(500)   =      lower surface z coordinates after partitioning
c   The numbering scheme of the nodes of the wing box is a follows.
c   Node 1 is the top left and all the nodes on the upper
c   surface of the wing box are odd numbered.  So the numbering
c   progresses down into neg. z-direction first which is into
c   the paper, than down the paper which is in + x-direction
c   and then in y-direction.
c           *-----> y-direction
c           |
c           |
c           V x-direction
c   xc(i), yc(i) , zc(i)    =      global nodal coordinates
c                               where the the numbering
c                               scheme goes from 1 to tn.
c   tn                      =      total nodes
c   nele                    =      total number of elements

```

```

c      nnodes(i)           =      nodes per element i
c      n1(i)...           =      node 1...3 of element i
c      t1(i)...           =      thickness of element at node i
c                          =      or cross-sectional
c                          =      area of axial bar
c      e                   =      Young's modulus
c      nu                  =      Poisson's Ratio
c      rho                  =      density of material
c      mat                  =      material of element i
c      To keep 3 spars in wing box model, try to keep nx = odd number so
c      that the spars may be equally divided. One is l.e. and one is the
c      t.e. and on in middle of t.e. and l.e.
      implicit none
      include 'wboxbc.f'
      integer i , j , k , tot , nx , ny , nxt , nyt , st , ei ,
*          krem , count , it , ii , no1 , no2 , no3 , isp ,
*          ribloc(12) , l , m , ifs , irsu , irsl , iasu , iasl ,
*          n1(1) , n2(1) , n3(1) , mat(1) , nnodes(1) , isurf(1)
      real*8 x1 , x2 , x3 , x4 , y1 , y2 , y3 , y4 , x(tn) , y(tn) ,
*          xx(tn) , yy(tn) , fr , zu(tn) , zl(tn) , mid , fu ,
*          t11,t11p,areas(100) ,rarea, lethick(100) ,
*          xc(1) , yc(1) , zc(1) , t1(1) , t2(1) , t3(1) ,
*          e(1) , nu(1) , rho(1) , fud , tx , tx1 , yr , yt
      data ribloc / 25 , 49 , 73 , 97 , 121 , 145 , 169 , 193 , 217 ,
*          241 , 265 , 289 /
      nxt = nxx
      nyt = nyy
      count = 0
c      Get rod areas
      open(1,file='rodareas.dat')
      do j = 1 , 100

```

```
        areas(j)=0.0d0
    end do
    i = 0
13 i = i + 1
    read(1,*,end=19) j , rarea
    areas(j) = rarea
    goto 13
19 close(1)
c    Get ARW-2 Wing Coordinates
    call arw2wing(xc,yc,zc)
c    Calculate Axial Bar Properties for Spars
    ifs = 44-1
    irsu = 1-1
    irsl = 17-1
    iasu = 71-1
    iasl = 76-1
    do m = 1 , tn - 2*(nxt+2) , 2*(nxt+2)
        i = m+6
        j = i+8
        k = i+12
        count = count + 4
        do ii = 0 , 3
            if (ii.eq.0) then
                ifs = ifs + 1
                if (ifs.gt.58) ifs=58
                isp = ifs
                it = i
            else if (ii.eq.1) then
                it = i+1
                ifs = ifs
                if (ifs.gt.58) ifs=58
```

```
    isp = ifs
else if (ii.eq.2) then
    it = j
    irsu = irsu + 1
    if (irsu.gt.15) irsu=15
    isp = irsu
else if (ii.eq.3) then
    it = j+1
    irsl = irsl + 1
    if (irsl.gt.31) irsl=31
    isp = irsl
end if
    nnodes(count-ii) = 2
if (areas(isp).eq.0) then
    rarea=areas(isp-1)
    if (rarea.eq.0) rarea=areas(isp+1)
else
    rarea=areas(isp)
end if
if (rarea.eq.0) then
    write(*,*) 'error in rarea',rarea,isp
    stop
end if
t1(count-ii) = rarea
t2(count-ii) = 0.0d0
t3(count-ii) = 0.0d0
n1(count-ii) = it
n2(count-ii) = it + 2*(nxt+2)
n3(count-ii) = 0
mat(count-ii) = 1
isurf(count-ii) = 0
```

```
end do
if (k.lt.115) then
  count = count + 2
  do ii = 0 , 1
    if (ii.eq.0) then
      it = k
      iasu = iasu + 1
      if (iasu.gt.75) iasu=75
      isp = iasu
    else if (ii.eq.1) then
      it = k+1
      iasl = iasl + 1
      if (iasl.gt.80) iasl=80
      isp = iasl
    end if
    nnodes(count-ii) = 2
    if (areas(isp).eq.0) then
      rarea=areas(isp-1)
      if (rarea.eq.0) rarea=areas(isp+1)
    else
      rarea=areas(isp)
    end if
    if (rarea.eq.0) then
      write(*,*) 'error in rarea',rarea,isp
      stop
    end if
    t1(count-ii) = rarea
    t2(count-ii) = 0.0d0
    t3(count-ii) = 0.0d0
    n1(count-ii) = it
    n2(count-ii) = it + 2*(nxt+2)
  end do
end if
```

```
        n3(count-ii) = 0
        mat(count-ii) = 1
        isurf(count-ii) = 0
    end do
end if
end do
write(*,*) count , nax
c Calculate global node numbers of ribs, number of nodes,
c material #, and thickness of element at each node
do i = 1 , 12
    do j = ribloc(i) , ribloc(i)+20 , 2
        count = count + 2
        nnodes(count) = 3
        nnodes(count-1) = 3
        if (j.ge.31.and.j.le.39) then
            t1(count) = 0.11d0
            t2(count) = 0.11d0
            t3(count) = 0.11d0
            t1(count-1) = 0.11d0
            t2(count-1) = 0.11d0
            t3(count-1) = 0.11d0
        else
            t1(count) = 0.04d0
            t2(count) = 0.04d0
            t3(count) = 0.04d0
            t1(count-1) = 0.04d0
            t2(count-1) = 0.04d0
            t3(count-1) = 0.04d0
        end if
        n1(count) = j
        n2(count) = j + 1
```

```

        n3(count) = j + 3
        n1(count-1) = j + 3
        n2(count-1) = j + 2
        n3(count-1) = j
        isurf(count) = 0
        isurf(count-1) = 0
        mat(count) = 2
        mat(count-1) = 2
    end do
end do
write(*,*) count-nax , nrrib
c Calculate global node numbers of triangular skins,
c number of nodes, thickness of element at each node,
c material #. (membrane elements)
open(1,file='thickness.dat')
read(1,*)
read(1,*) t11p
ei = 0
do i = 1 , tn-2*(nxt+2) , 2
    if (mod(i+1,2*(nxt+2)).ne.0) then
        count = count + 4
        read(1,*) k,t11
        if (t11.lt.0) t11=t11p
        if (mod(i-1,24).eq.0) then
            ei = ei + 1
            lethick(ei) = t11
        end if
    do j = 0 , 3
        nnodes(count-j) = 3
        t1(count-j) = t11
        t2(count-j) = t11
    end do
end do

```

```

        t3(count-j) = t11
        mat(count-j) = 3
    end do
c    Upper surface
    n1(count) = i
    n2(count) = i+2
    n3(count) = i+2*(nxt+2)
    n1(count-1) = i+2
    n2(count-1) = i+2+2*(nxt+2)
    n3(count-1) = i+2*(nxt+2)
    isurf(count) = 1
    isurf(count-1) = 1
c    Lower surface
    n1(count-2) = i+1
    n2(count-2) = i+3
c    n3(count-2) = i+3+2*(nxt+2)
    n3(count-2) = i+1+2*(nxt+2)
c    n1(count-3) = i+1
    n1(count-3) = i+3
    n2(count-3) = i+3+2*(nxt+2)
    n3(count-3) = i+1+2*(nxt+2)
    isurf(count-2) = -1
    isurf(count-3) = -1
endif
end do
close(1)
write(*,*) count-nrib-nax , nmem
c    Calculate global node numbers of spars, the thicknesses of the
c    element at node i,material #, and number of nodes.
c    i index is for leading edge while j is for trailing edge
c    while k is index for the middle spar. Remember we are

```

```
c      assuming 3 constant spars.
      open(1,file='spars.dat')
      do i = 1 , 100
        areas(i) = 0.0d0
      end do
10 read(1,*,end=20) i , rarea
      areas(i) = rarea
      goto 10
20 close(1)
      ifs = 61-1
      irsu = 37-1
      iasu = 69-1
      ei = 0
      do i = 1 , tn - 2*(nxt+2) , 2*(nxt+2)
        ei = ei + 1
        j = i+6
        k = i+14
        l = i+18
        m = i+22
        do ii = 0 , 3
          fud = 1.0d0
          if (ii.eq.0) then
            it=i
            isp = -37
          else if (ii.eq.1) then
            it=j
c      Added this fud
            fud = 0.80d0
            ifs = ifs + 1
            if (ifs.gt.63) ifs=63
            isp = ifs
```

```
else if (ii.eq.2) then
    it=k
    irsu = irsu + 1
    isp = irsu
else if (ii.eq.3) then
    it=m
    isp = -37
end if
count = count + 2
nnodes(count) = 3
nnodes(count-1) = 3
if (isp.ne.-37) then
    rarea=areas(isp)
else
    rarea=lethick(ei)
end if
if (rarea.eq.0) then
    write(*,*) 'error',isp,rarea
    stop
end if
t1(count) = rarea*fud
t2(count) = rarea*fud
t3(count) = rarea*fud
n1(count) = it
n2(count) = it+1
n3(count) = it+1+2*(nxt+2)
t1(count-1) = rarea
t2(count-1) = rarea
t3(count-1) = rarea
n1(count-1) = it+1+2*(nxt+2)
n2(count-1) = it+2*(nxt+2)
```

```
n3(count-1) = it
if (isp.ne.-37) then
  isurf(count) = 0
  mat(count) = 4
  isurf(count-1) = 0
  mat(count-1) = 4
else
  isurf(count) = 1
  mat(count) = 3
  isurf(count-1) = 1
  mat(count-1) = 3
end if
end do
if (1.lt.115) then
  it = 1
  iasu = iasu + 1
  isp = iasu
  count = count + 2
  nnodes(count) = 3
  nnodes(count-1) = 3
  rarea=areas(isp)
  if (rarea.eq.0) then
    write(*,*) 'error',isp,rarea
    stop
  end if
  t1(count) = rarea
  t2(count) = rarea
  t3(count) = rarea
  t1(count-1) = rarea
  t2(count-1) = rarea
  t3(count-1) = rarea
```

```
        n1(count) = it
        n2(count) = it+1
        n3(count) = it+1+2*(nxt+2)
        n1(count-1) = it+1+2*(nxt+2)
        n2(count-1) = it+2*(nxt+2)
        n3(count-1) = it
        isurf(count) = 0
        mat(count) = 4
        isurf(count-1) = 0
        mat(count-1) = 4
    end if
end do
write(*,*) count-nax-nrib-nmem , nspar
c Now define the various material properties
c Axial Bar:          mat# 1
c Ribs:              mat# 2
c Skins:             mat# 3
c Spars:             mat# 4
c The following are properties for Aluminum wing
c Young's Modulus of Elasticity (Redundant, ain't it)
e(1) = 10.30d6
e(2) = 10.30d6
e(3) = 10.68d6
e(4) = 10.30d6
c Poisson's Ratio
nu(1) = 0.3205d0
nu(2) = 0.3205d0
nu(3) = 0.3d0
nu(4) = 0.3205d0
c Density of Material
rho(1) = 0.1d0
```

```
        rho(2) = 0.1d0
        rho(3) = 0.101d0
        rho(4) = 0.1d0
        fu = 10.0d0*0.1d0
        do i = 1 , nele
c      1 axial, 2 rib, 3 skin, 4 spar
          if (mat(i).eq.1) then
c            changed from 0.5
              t1(i) = t1(i)*0.35d0
              continue
          else if (mat(i).eq.2) then
c            changed from 1.0 to
              t1(i) = t1(i)*1.0d0
              t2(i) = t2(i)*1.0d0
              t3(i) = t3(i)*1.0d0
              continue
          else if (mat(i).eq.3) then
c            changed from 0.7
              t1(i) = t1(i)*0.92d0
              t2(i) = t2(i)*0.92d0
              t3(i) = t3(i)*0.92d0
              continue
          else
c            changed from 0.7
              t1(i) = t1(i)*0.625d0
              t2(i) = t2(i)*0.625d0
              t3(i) = t3(i)*0.625d0
              continue
          end if
        end do
      do i = 1 , tn
```

```

        xc(i) = xc(i)*phylen
        yc(i) = yc(i)*phylen
        zc(i) = zc(i)*phylen
    end do
    if (count.ne.nele) then
        write(*,*) 'OOPS! Elements not equal!'
        write(*,*) count , nele
        stop
    endif
    write(*,*) count , nele
c    Write file for Lloyd
    open(1,file='input.deck')
    write(1,*) 'Total nodes: ',tn
    write(1,*) 'Total elements: ',nele
    do i = 1 , tn
        write(1,*) 'Grid',i,real(xc(i)),real(yc(i)),real(zc(i))
    end do
    write(1,*) 'element information:'
    write(1,*) 'crod    node 1    node 2    area    material#'
    write(1,*) 'mem     node 1    node 2    node 3    thickness
*    material#'
    do i = 1 , nele
        if (mat(i).eq.1) then
            write(1,*) 'crod',n1(i),n2(i),t1(i),mat(i)
        else
            write(1,*) 'mem',n1(i),n2(i),n3(i),t1(i),mat(i)
        end if
    end do
    write(1,*) 'Material #    Young s Modulus    Poisson s Ratio'
    do i = 1 , nmat
        write(1,*) i,e(i),nu(i)
    end do

```

```
    end do
    close(1)
    write(*,*) 'Done file for Lloyd'
c    Write output to be used by FAST
c    write(8,*) nzt+2,nyt+2,2
c    write(8,*) (xc(i),i=1,2*tot,2),(xc(i),i=2,tn,2)
c    write(8,*) (yc(i),i=1,2*tot,2),(yc(i),i=2,tn,2)
c    write(8,*) (zc(i),i=1,2*tot,2),(zc(i),i=2,tn,2)
    return
end
```

# Appendix B

## Aeroelastic Coupling Procedure Source Code

This is the source code used to create the interface mappings to perform static aeroelastic analysis. The source code below is the major part of the pressure to force mapping process. It takes data from a CFD and CSD grid, and creates the necessary mapping of CFD point to the CSD triangle. In order to use this, the forces on the CSD points need to be obtained, and that can be done in any manner one wishes. Also, the surface spline routines are not included due to their simplicity.

```
c-----c
c
c   Given a CFD grid and Structural Grid (upper, lower)   c
c   this routine tries to associate a structural triangle  c
c   with each CFD grid point.                             c
c
c   i1,itip,imax,j1,jtip,jmax define the aero. grid      c
c   xa,ya,za are the aerodynamic grid points coordintates c
c   ns is the number of structural points                 c
c   xs,ys,zs are the structural node coordinates         c
c   map maps structural point to overall position        c
c               in combined structural grid              c
```

APPENDIX B. AEROELASTIC COUPLING PROCEDURE SOURCE CODE 189

```

c      This routine assumes we are not concerned with          c
c      z coordinate, i.e., the object is in x-y plane          c
c                                                                c
c-----c
      subroutine trian(i1,itip,imax,j1,jtip,jmax,nus,nls,
'                xa,ya,za,xus,yus,zus,xls,yls,zls,
'                mapus,mapls,strid,weight)
c-----c

      implicit none
c-----c
c      Number of Closest Points to Acquire                      c
c-----c

      integer np
      parameter ( np=35 )
c-----c
c      These are declartions of variables being passed          c
c-----c

      integer i1 , itip , imax , j1 , jtip , jmax , nus , nls ,
'          mapls(nls) , mapus(nus) , strid(3,imax,jmax)
      real*8 xa(imax,jmax) , ya(imax,jmax) , za(imax,jmax) ,
'          xus(nus) , yus(nus) , zus(nus) , xls(nls) ,
'          yls(nls) , zls(nls) , weight(3,imax,jmax)
c-----c
c      These are declarations for variables being used locally  c
c-----c

      integer i , j , cp(np) , nc , pt , m , lou , n , k ,
'          p , tri(3,np*np*np) , che , k1 , m1 , p1 , ntria ,
'          count , pn(3) , ii , trio(3,np*np*np) , temp , iii
      real*8 x , y , z , dist , max , meas(np) , xp , yp , zp ,
'          dist1 , dx(4) , dy(4) , de(4) , max1
c-----c

```

APPENDIX B. AEROELASTIC COUPLING PROCEDURE SOURCE CODE 190

```

c      Begin Program                                     c
c-----c
      iii = 0
c-----c
c      Begin Main Loop                                 c
c-----c
      do i = i1 , imax
        do j = j1 , jmax
          lou = -1
          iii = iii + 1
          nc = 0
c      lou = 1 (upper surface) lou = -1 (lower surface)
          if (i.gt.itip) lou = 1
          if (j.gt.jtip) lou = 1
          x = xa(i,j)
          y = ya(i,j)
          z = za(i,j)
c      To be used later for acquiring weights
          dx(4) = x
          dy(4) = y
c-----c
c      Get np closest points for grid pt. i,j         c
c-----c
          if (lou.eq.1) then
            n = nus
          else
            n = nls
          end if
          do k = 1 , n
            if (lou.eq.1) then
              xp = xus(k)

```

```

        yp = yus(k)
        zp = zus(k)
    else
        xp = xls(k)
        yp = yls(k)
        zp = zls(k)
    end if
    dist1 = dist(x,y,z,xp,yp,zp)
    if (nc.lt.np) then
        nc = nc + 1
        meas(nc) = dist1
        cp(nc) = k
    else
        max = 0.0d0
        do m = 1 , np
            if (max.lt.meas(m)) then
                max = meas(m)
                pt = m
            end if
        end do
        if (dist1.le.max) then
            meas(pt) = dist1
            cp(pt) = k
        end if
    end if
end do

c-----c
c      Done np closest points, now get all triangle      c
c      regions possible to get proper triangle          c
c-----c

    if (i.eq.1.and.j.eq.1) then

```

APPENDIX B. AEROELASTIC COUPLING PROCEDURE SOURCE CODE 192

```

ntria = 0
count = 0
do k = 1 , np
  do m = 1 , np
    if (m.eq.k) goto 10
    do p = 1 , np
      if (p.eq.m.or.p.eq.k) goto 20
      k1 = k
      m1 = m
      p1 = p
      call check(che,np,k1,m1,p1,ntria,tri)
      if (che.ne.1) then
        ntria = ntria + 1
        trio(1,ntria) = k1
        trio(2,ntria) = m1
        trio(3,ntria) = p1
        tri(1,ntria) = k1
        tri(2,ntria) = m1
        tri(3,ntria) = p1
      end if
20      continue
    end do
10      continue
  end do
end do
c      write(*,*) 'Triangles ',ntria
end if
c-----c
c      Now Have All Possible Triangles Without Dupes      c
c      So Check to See If CFD Point Is An Int Point      c
c-----c

```

APPENDIX B. AEROELASTIC COUPLING PROCEDURE SOURCE CODE 193

```

do k = 1 , ntria
  do ii = 1 , 3
    tri(ii,k) = trio(ii,k)
  end do
end do
count = ntria
do k = 1 , ntria
  do ii = 1 , 3
    pn(ii) = cp(tri(ii,k))
  end do
  do m = 1 , 3
    if (lou.eq.1) then
      dx(m) = xus(pn(m))
      dy(m) = yus(pn(m))
    else
      dx(m) = xls(pn(m))
      dy(m) = yls(pn(m))
    end if
  end do
  call getco(dx,dy,de)
c   Get rid of triangle if it doesn't contain point i,j
  if (de(1).lt.0.or.de(2).lt.0.or.de(3).lt.0.or.de(4).gt.0) then
    count = count - 1
    tri(1,k) = 0
    tri(2,k) = 0
    tri(3,k) = 0
  else if (de(1)+de(2)+de(3).lt.0.98) then
    count = count - 1
    tri(1,k) = 0
    tri(2,k) = 0
    tri(3,k) = 0
  end if
end do

```

```

        else if (de(1)+de(2)+de(3).gt.1.02) then
            count = count - 1
            tri(1,k) = 0
            tri(2,k) = 0
            tri(3,k) = 0
        else if (de(4).lt.0) then
            temp = tri(1,k)
            tri(1,k) = tri(3,k)
            tri(3,k) = temp
        end if
    end do

c-----c
c          Now Check Which Triangle Has Smallest Largest      c
c          Vertex Distance                                     c
c-----c

    if (count.ne.0) then
        max1 = 100000.0d0
        do k = 1 , ntria
            if (tri(1,k).ne.0) then
                max = 0.0d0
                do ii = 1 , 3
                    pn(ii) = cp(tri(ii,k))
                end do
                do m = 1 , 3
                    if (lou.eq.1) then
                        dx(m) = xus(pn(m))
                        dy(m) = yus(pn(m))
                    else
                        dx(m) = xls(pn(m))
                        dy(m) = yls(pn(m))
                    end if
                end do
            end if
        end do
    end if

```

APPENDIX B. AEROELASTIC COUPLING PROCEDURE SOURCE CODE 195

```

        end do
        do m = 1 , 3
            dist1 = dist(dx(m),dy(m),0.0d0,x,y,0.0d0)
            if (max.lt.dist1) max=dist1
        end do
c      Using max variable to get minimum
        if (max1.gt.max) then
            max1 = max
            pt = k
        end if
        end if
    end do
do ii = 1 , 3
    pn(ii) = cp(tri(ii,pt))
end do
do m = 1 , 3
    if (lou.eq.1) then
        dx(m) = xus(pn(m))
        dy(m) = yus(pn(m))
    else
        dx(m) = xls(pn(m))
        dy(m) = yls(pn(m))
    end if
end do
call getco(dx,dy,de)
do k = 1 , 3
    weight(k,i,j) = de(k)
    if (lou.eq.1) then
        strid(k,i,j) = mapus(pn(k))
    else
        strid(k,i,j) = mapls(pn(k))
    end if
end do
```

APPENDIX B. AEROELASTIC COUPLING PROCEDURE SOURCE CODE 196

```

                end if
            end do
        else
c-----c
c          If there are no triangles which contain    c
c          grid pt. i,j then map it to closest point  c
c          NOT triangle                                c
c-----c
            max = 10000000.0d0
c          write(*,*) 'not interior point'
            do k = 1 , np
                if (max.gt.meas(k)) then
                    max = meas(k)
                    pt = cp(k)
                end if
            end do
            do k = 1 , 3
                weight(k,i,j) = 0.0d0
                if (lou.eq.1) then
                    strid(k,i,j) = mapus(pt)
                else
                    strid(k,i,j) = mapls(pt)
                end if
            end do
        end if
        write(*,100) iii,imax*jmax
100      format('Done',2x,i4,'/',i4)
c          write(*,*) i , j
c          write(*,*) (strid(k,i,j),k=1,3)
c          write(*,*) (weight(k,i,j),k=1,3)
c-----c

```

APPENDIX B. AEROELASTIC COUPLING PROCEDURE SOURCE CODE 197

```
c          Finished Doing Mapping for Grid Point i,j          c
c-----c
c          end do
c          end do
c-----c
c          End Main Loop          c
c-----c
c          return
c          end
c-----c
c          This function is used to get distance between 2 points  c
c-----c
c          function dist(x1,y1,z1,x2,y2,z2)
c          implicit none
c          real*8 x1 , x2 , y1 , y2 , dist , temp , z1 , z2
c          temp = (x2-x1)**2 + (y2-y1)**2 + (z2-z1)**2
c          dist = dsqrt(temp)
c          return
c          end
```

# Vita

Manoj Kumar Bhardwaj was born in India on November 26, 1969. He moved to the United States in March 1977. He graduated from James Wood High School in 1988 located in Winchester, Virginia. He then began his second life at Virginia Tech. He graduated with a B.S. in Aerospace Engineering in 1992. He also received an M.S. in Aerospace Engineering en route to his Ph.D. Manoj will work in Albuquerque, New Mexico at Sandia National Laboratories. His permanent address is:

142 Hackberry Drive Stephens City, VA 22655