APPLICATION OF SPRING ANALOGY MESH DEFORMATION TECHNIQUE IN AIRFOIL DESIGN OPTIMIZATION

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

BY

YOSHEPH YANG

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

JULY 2015

Approval of the thesis:

APPLICATION OF SPRING ANALOGY MESH DEFORMATION TECHNIQUE IN AIRFOIL DESIGN OPTIMIZATION

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

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

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

Prof. Dr. Serkan Özgen _____
Supervisor, **Aerospace Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Zafer Dursunkaya _____
Mechanical Engineering Dept., METU

Prof. Dr. Serkan Özgen _____
Aerospace Engineering Dept., METU

Assoc. Prof. Dr. Melin Şahin _____
Aerospace Engineering Dept., METU

Asst. Prof. Dr. Ercan Gürses _____
Aerospace Engineering Dept., METU

Assoc. Prof. Dr. Kürşad Melih Güleren _____
Aeronautical Engineering Dept., UTAA

Date: 30.07.2015

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

Name, Last Name: Yosheph Yang

Signature:

ABSTRACT

APPLICATION OF SPRING ANALOGY MESH DEFORMATION TECHNIQUE IN AIRFOIL DESIGN OPTIMIZATION

Yang, Yosheph
M.S., Department of Aerospace Engineering
Supervisor : Prof. Dr. Serkan Özgen

July 2015, 111 pages

In this thesis, an airfoil design optimization with Computational Fluid Dynamics (CFD) analysis combined with mesh deformation method is elaborated in detail. The mesh deformation technique is conducted based on spring analogy method. Several improvements and modifications are addressed during the implementation of this method. These enhancements are made so that good quality of the mesh can still be maintained and robustness of the solution can be achieved. The capability of mesh deformation is verified by considering rotating case of an airfoil for both inviscid and viscous meshes. The edge connectivity required in the spring analogy itself is computed by several simple algorithms. It is found that the presence of modified spring analogy technique leads to better solution in mesh deformation technique.

Regarding the aerodynamic design optimization, SU2 3.2.9 open source software is used as the CFD Solver. During the computation, the initial mesh used in the optimization is obtained from Pointwise® mesh generation software. OPTLIB Gradient Optimizer of Phoenix Model Center is implemented as the optimization solver. The optimization process is conducted for four different flight conditions. In each flight condition, minimizing airfoil drag becomes the objective function with

different angle of attack constraints imposed. Furthermore, several shape parameterizations are utilized. It is found that in each case, optimized airfoil can be found based on the designated design variables.

Keywords: Mesh Deformation, Spring Analogy, Airfoil Design Optimization, Computational Fluid Dynamics

ÖZ

YAY BENZETİMLİ ÇÖZÜM AĞI DEFORMASYON TEKNİĞİNİN KANAT KESİTİ TASARIMI EN İYİLEŞTİLEŞTİRİLMESİNDE UYGULANMASI

Yang, Yosheph
Yüksek Lisans, Havacılık ve Uzay Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Serkan Özgen

Temmuz 2015, 111 sayfa

Bu tezde, kanat kesiti en iyileştirilmesinde kullanılan Hesaplamalı Akışkanlar Dinamiği (HAD) analizleri ile birleştirilmiş çözüm ağı deformasyon tekniği detaylı bir biçimde anlatılmıştır. Kullanılan çözüm ağı deformasyon tekniğinde, yay benzetim metodu baz alınmıştır. Tez içerisinde, metodun uygulanışındaki geliştirme ve modifikasyonlara da yer verilmiştir. Bu geliştirmeler, deforme olmuş çözüm ağının kalitesinden ve çözümün gürbüzlüğünden emin olabilmek için yapılmıştır. Viskoz ve viskoz olmayan çözüm ağlarında, kanat kesitinin döndürülme durumu incelenerek çözüm ağı deformasyonunun yeteneği doğrulanmıştır. Yay benzetiminde gerekli olan çözüm ağı düğüm noktaları bağlantıları çeşitli basit algoritmalar kullanılarak hesaplanmıştır. Modifiye edilmiş metodun, klasik metoda göre daha iyi çözümler verdiği tespit edilmiştir.

Aerodinamik tasarım en iyileştirilmesinde, SU2 3.2.9 açık kaynak kodlu yazılımı HAD çözücüsü olarak kullanılmıştır. En iyileştirmede kullanılacak ilk çözüm ağı Pointwise® çözüm ağı oluşturma yazılımıyla elde edilmiştir. En iyileştirme çözücüsü olarak Phoenix Model Center yazılımın "OPTLIB Gradient Optimizer" modülü kullanılmıştır. En iyileştirme sürecinde dört farklı uçuş koşulu göz önüne

alınmıştır. Her bir uçuş koşulunda, farklı hücum açısı kısıtlamaları kullanılarak, kanat kesitinin oluşturduğu sürüklemeyi en aza indirmek amaç fonksiyonu olarak belirlenmiştir. Ayrıca, çeşitli şekil parametrelendirmesi de kullanılmıştır. Her bir durumda, en iyileştirilmiş kanat kesitinin belirtilen tasarım değişkenleri temel alınarak elde edilebileceği tespit edilmiştir.

Anahtar Kelimeler: Çözüm Ağı Deformasyonu, Yay Benzetimi, Kanat Kesiti Tasarımı En İyileştirmesi, Hesaplamalı Akışkanlar Dinamiği

To my family and closest friends
who inspired me to finish this thesis

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Prof. Dr. Serkan Özgen for his guidance, advice, criticism, insight, and encouragement throughout the research.

I would also like to thank my superiors within the CHANGE Project, Prof. Dr. Yavuz Yaman, Assoc. Prof. Dr. Melin Şahin, Assist. Prof. Dr. Ercan Gürses for their supports during the study.

I want to express my gratitude to my colleagues, İlhan Ozan Tunçöz, Nima Pedramasl, Ramin Rouzbar, Harun Tıraş, Uğur Kalkan, Pınar Arslan, Onur Akın, and my other friends who were there to give me support during my study.

I would like to thank my parents, my sister, and my brother who never ceased to give me encouragement and pray for me during my graduate study.

I would also like to show my thanks to my Indonesian friends in Ankara, especially my housemates. Thank you very much for being there to support me.

I would also like thank to The Scientific and Technological Research Council of Turkey (TÜBİTAK) "2215 Graduate Scholarship Programme for International Students" for supporting me financially during my graduate level education.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF SYMBOLS

A_{ijk}	Area of triangular cell whose node numbers are i, j, k
C_i^{ijk}	Torsional spring stiffness attached at node i in a triangular cell ijk
$\vec{F}_i$	Spring force exerted on node i
F_{ij_x}	Spring force exerted on node i by node j in the x-direction
F_{ij_y}	Spring force exerted on node i by node j in the y-direction
F^{ijk}	Force vectors generated in a triangular cell ijk
K	Global stiffness matrix corresponding to all degree of freedoms on the mesh
K_{aa}	Partitioned of global stiffness matrix corresponding to only active degree of freedoms
K_{bb}	Partitioned of global stiffness matrix corresponding to only prescribed degree of freedoms (boundary nodes)
K_{ab}	Partitioned matrix corresponding to active-prescribe degree of freedoms
K_{ba}	Partitioned matrix corresponding to prescribe-active degree of freedoms
k_{ij}	Linear spring stiffness for the segment spring analogy
k_{ip}	Spring stiffness used for the fictitious edge
$k_{ij}^{\text{semi}-\text{torsional}}$	Semi-torsional spring stiffness for the segment spring analogy
k_{ij}^{total}	Summation of all spring stiffness for the segment spring analogy
K_{ij}	Stiffness matrix defined for each edge $i - j$
$K_{torsion}^{ijk}$	Torsional stiffness matrix for each triangular cell ijk
l_{ij}	Length of edge whose node are i and j
M^{ijk}	Moments generated for each nodes in a triangular cell ijk
NE_{ij}	Number of cells whose one of edge contains both node i and j

$\vec{q}_i$	Displacement vector of node i
$\vec{q}_p$	Displacement vector of fictitious node p located in the center of triangular cell
R^{ijk}	Matrix represents kinematic relation between angular displacement and nodal displacement in a triangular cell ijk.
T^{ijk}	Transformation matrix in a triangular cell ijk for torsional spring analogy
u_i	Displacement of node i in x-direction
v_c	Number of neighbor cells surrounding node i
v_i	Number of neighbor nodes surrounding node i
v_i	Displacement of node i in y-direction
$\vec{x}_i$	Position vector of node i
$\vec{x}_p$	Position vector of fictitious node p located in the center of triangular cell
α_{ij}	Spring stiffness for the vertex spring analogy
$\Delta\theta_i^{ijk}$	Rotational displacement of node i in a triangular cell ijk
Δx_i	Displacement of node i in the x-direction
Δy_i	Displacement of node i in the y-direction
θ_{ij}	Angle made by each edge $i-j$
θ_i^{ijk}	Angle made between edge $i-j$ and edge $i-k$ in the triangular cell ijk
λ	Spring constant used for semi-torsional spring analogy
Φ	Multiplying factor used for stiffness value of the spring
Ψ	Exponential factor used for stiffness value of the spring
ω	Relaxation parameter used in the SOR solution method

LIST OF ABBREVIATIONS

CFD	Computational Fluid Dynamics
CHANGE	Combined morphing assessment software using flight envelope data and mission based morphing wing prototype development
NACA	National Advisory Committee for Aeronautics
OPTLIB	Optimization Library
PARSEC	Parameterized Section
PDE	Partial Differential Equations
RANS	Reynolds Averaged Navier Stokes
SOR	Successive Overrelaxation
SQP	Sequential Quadratic Programming
SU2	Stanford University Unstructured

CHAPTER 1

INTRODUCTION

1.1 Motivation of the Study

The status of Computational Fluid Dynamics (CFD) tools at the current level brings a great help to many aircraft designers during airfoil selection. By the aid of this tool, aerodynamic properties of the airfoil can be calculated easily. Combination of this tool with an optimization tool will help the designers to find the optimum design easily instead of doing many experimental analysis. In order to enhance the optimization in terms of updating mesh, instead of generating a new mesh for each iteration, mesh deformation technique is implemented.

In this thesis, the aforementioned methods are actualized in design optimization of an airfoil. The optimization process is performed by considering the objective function to be minimizing airfoil sectional drag by specifying airfoil sectional lift as the constraint. Additionally, the optimization procedure is conducted for some flight conditions in the mission profile. During the optimization process, the deformed mesh is attained by using the developed mesh deformation tool, based on the spring analogy mesh deformation method.

1.2 Limitation of the Study

In this thesis, the study is limited to aerodynamic point of view of the airfoil design optimization based on CFD Tools. It is assumed that the results obtained from CFD analysis is reliable. The turbulence modelling of the CFD solver is already verified[1]. As a result, no further experimental analysis for the airfoil is conducted.

In the mesh deformation and CFD analyses performed here, the meshes should be unstructured meshes in 2-D Airfoil. Different type of unstructured polygon meshes or structured meshes will not be taken into account in this thesis.

Apart from the above mentioned constraints, the flow regime is limited to incompressible flow only. Consequently, the optimization for subsonic or supersonic flow is not addressed in this thesis.

1.3 Layout of the Study

Chapter 2 encloses the literature study about airfoil design optimization. This includes the shape parameterization and optimization scheme utilized in the analysis. Additional, some past works concerning variation in mesh deformation technique is elaborated in here as well.

Chapter 3 gives brief information regarding spring analogy mesh deformation methods applied in this thesis. These methods are divided into some categories based on the procedure and numerical solution used to get the final deformed mesh. Moreover, two different solution procedures implemented in the study are explained here. The required mesh connectivity and data structure used in the code are also studied in this chapter.

Chapter 4 contains the explanation about how the CFD and optimization analyses are conducted in the thesis. In the CFD analyses, information regarding initial mesh and flow parameters are provided. For the optimization analyses, implemented optimization scheme and implemented shape parameterizations are elaborated.

In Chapter 5, the results of mesh deformation method for the various spring analogy approaches are shown. Furthermore, the 2-D airfoil design optimization with given constraints are also presented in this chapter. The results are given in a systematic way by considering each case in the optimization.

Chapter 6 contains the general conclusions of the study. Moreover, the recommendation for the future work is also provided in here.

CHAPTER 2

LITERATURE REVIEW

This chapter is devoted to some brief explanations regarding aerodynamic design optimization, and mesh deformation method. Mostly, the information regarding recent research is elaborated.

2.1 Aerodynamic Optimization

The concept of aerodynamic optimization in the design process is not unfamiliar in the current decades. In fact, the two-dimensional aerodynamic design was already introduced by Lighthill [2] back in 1945. The fact that Computational Fluid Dynamics (CFD) tools have developed so greatly during these past decades becomes one of main reasons that optimization tools have been greatly combined with CFD tools. However, optimization tools were not implemented due to several reasons [3]: the estimation of drag coefficient just became more accurate in the past decade, high number of design spaces and non-linear constraints necessary to find the optimum value, and huge demanding computational resources to perform the optimization.

Like any other optimization concept, the notion of aerodynamic optimization also has some design variables and objective functions with some required constraints. Shahroki and Jahangirian [4] have described that choosing appropriate design variables for shape parameterization plays a significant role in determining the optimum shape of the airfoil, especially in transonic flow. Excellent design variables should encompass extensive design spaces. The objective functions defined in the optimization mainly comprises aerodynamic coefficients which govern the performance, like: maximum lift to drag ratio, minimum pitching moment, and many others. Some constraints are also imposed on the aerodynamic optimization in order

to achieve feasible optimum design. Epstein et al. [5] classified the constraints imposed during the optimization process as either geometry constraints, which mainly deals with the geometrical properties of the design or aerodynamic constraints, which are concerned more about the performance of the aircraft.

In order to have optimum wing, which is the major issue encountered in the optimization problem, one should also consider the airfoil, the basic element of the wing that dictates large-scale flow phenomena occurring in the wing [6].

The airfoil optimization itself in general can be categorized into two different categories: inverse design optimization, which tries to find a geometry which has a prescribed distribution of pressure coefficient. On the other hand, direct numerical optimization aims to find the best feasible design for some given constraints [7].

2.1.1 Shape Parameterization

Shape parameterization, which is dominantly introduced in the airfoil optimization depends on whether the aim is to improve a current design or to introduce a completely new design. For the improvement of the current design, some local perturbations along the airfoil surface are sufficient. However, getting a completely new design can be achieved by using other design shape parameterizations, which allows the significant changes in the geometry.

There are some attempts made to parameterize the airfoil shape. One of the well-known methods to parameterize airfoil shapes is known as PARSEC (PARameterized SECtion), which was developed by Sobieczky in 1998 [8]. The idea of this method is to parameterize the airfoil into several design parameters. Figure 2.1 specifies some required parameters to define the airfoil shape, which are: leading edge radius (R_{le}), abscissa of maximum peak for lower airfoil (X_{lo}), abscissa of maximum peak for upper airfoil (X_{up}), ordinate of maximum peak for lower airfoil (Y_{lo}), ordinate of maximum peak for upper airfoil (Y_{up}), curvature of maximum peak for lower airfoil (YXX_{lo}), curvature of maximum peak for upper airfoil (YXX_{up}), trailing edge thickness (T_{TE}), trailing edge offset (T_{off}), trailing edge direction angle (α_{TE}),

and trailing edge wedge angle (β_{TE}). A mathematical equation with six terms is selectively considered to describe both the upper and the lower airfoil surfaces separately. These representations of upper and lower airfoil surfaces are shown in Equation (2.1).

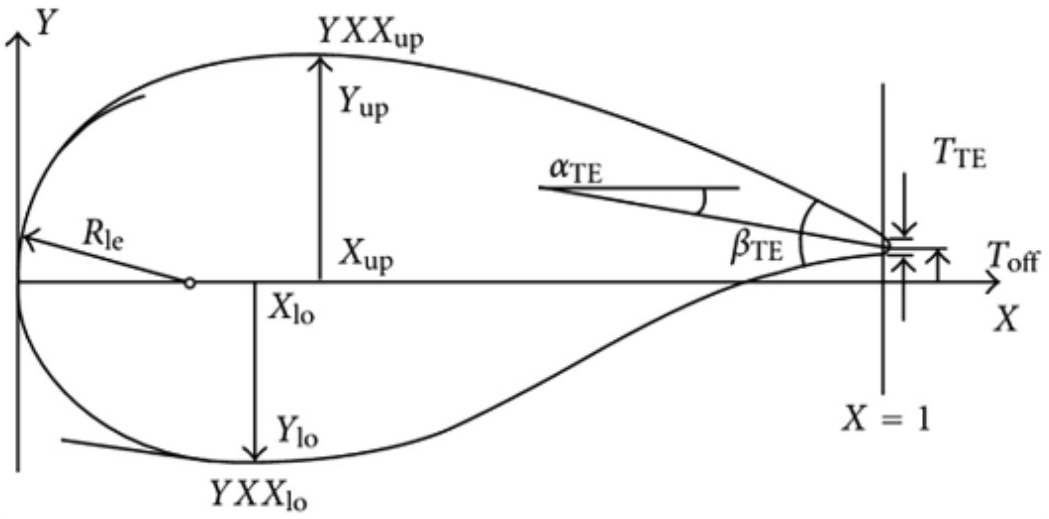


Figure 2.1 Airfoil shape parameterization using PARSEC method [8]

$$y_{upper} = \sum_{i=1}^{6} a_i x^{i-\frac{1}{2}}$$
$$y_{lower} = \sum_{i=1}^{6} b_i x^{i-\frac{1}{2}} \quad (2.1)$$

Coefficients of the mathematical equation representing the airfoil curve are found by satisfying the input parameters defined earlier in PARSEC method. Later, by using these coefficients, the curve for both upper and lower airfoil can be generated.

Another method that can be considered in the airfoil shape parameterization during the optimization is implementation of a bump function. Hicks and Henne introduced this notion while trying to apply some wing numerical optimization in 1978 [9]. The bump functions used by Hicks and Henne are shown in Equation (2.2).

$$y_{upper} = y_{upper_{basic}} + \sum_{i=1}^{5} a_i f_i$$
$$y_{lower} = y_{lower_{basic}} + \sum_{i=1}^{5} b_i f_i \tag{2.2}$$

The values of a_i and b_i are considered as the amplitude of the introduced bump function. Later, these values are taken as the design variables during the optimization process. Figure 2.2 depicts the shapes of the bump functions implemented by Hicks-Henne for numerical optimization.

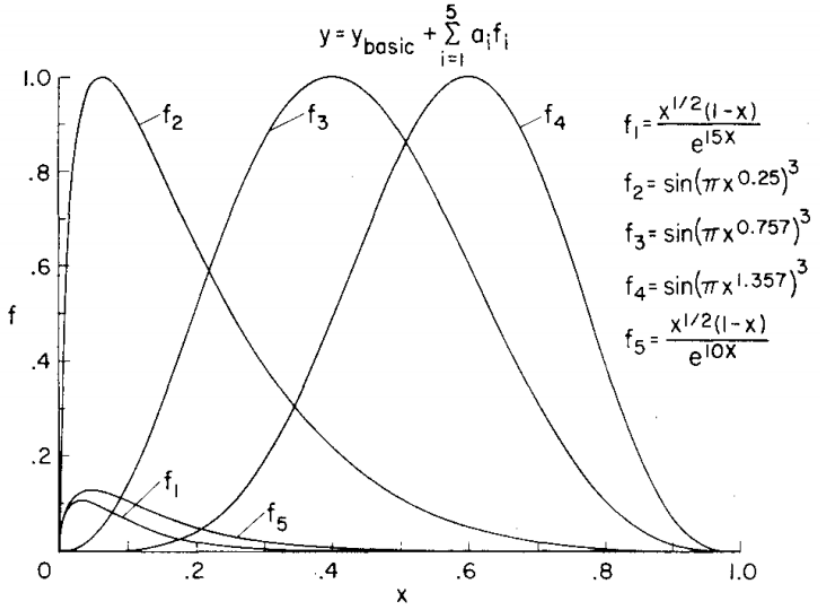


Figure 2.2 Hicks-Henne Bump Function [9]

The above defined bump functions also contain sinusoidal bump functions for f_2 to f_4. One can also generalize these bump functions by considering Equation (2.3), plotted in Figure 2.3.

$$f_i(x) = \left[\sin\left(\pi x^{\frac{\log 0.5}{\log t_1}}\right)\right]^{t_2} \tag{2.3}$$

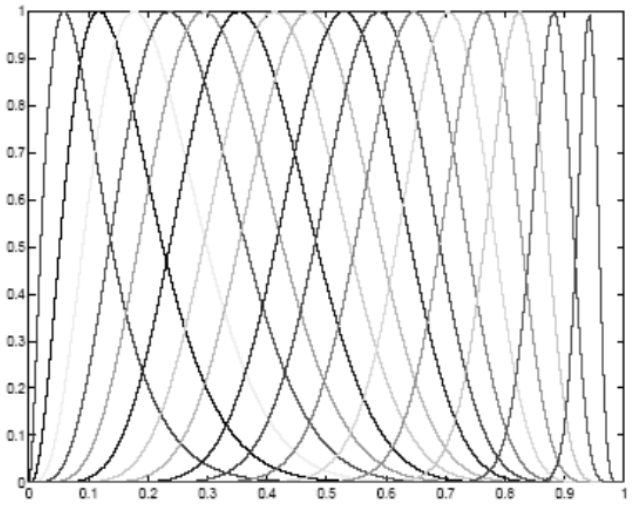


Figure 2.3 Set of Sinusoidal Bump Functions with Different Location of Maximum Bump [10]

The above function is utilized during the optimization procedure done by Tashinizi et al. in their work [10]. The variable t_1 defines the location of the maximum bump, whereas t_2 describes the width of the bump function.

Another traditional way to parameterize the airfoil shape is by using NACA 4-digit airfoil. In their work as well, Tashinizi et al. also considered NACA 4-digit airfoil as design parameters for the optimization [10]. They utilized the digit in NACA airfoil as design parameters used during the optimization.

Chen et al. [11] also consider modified Joukowsky transformation combined with smooth curvature technique for airfoil shape parameterization during the optimization. Detail of the transformation scheme is shown in Figure 2.4. They defined $\rho(\theta)$ as the shape function of the airfoil, which is shown in Equation (2.4).

$$\rho(\theta) = C_0 + C_1\theta + C_2\theta^2 + C_3\theta^3 + \cdots + C_k\theta^k + \cdots \quad (2.4)$$

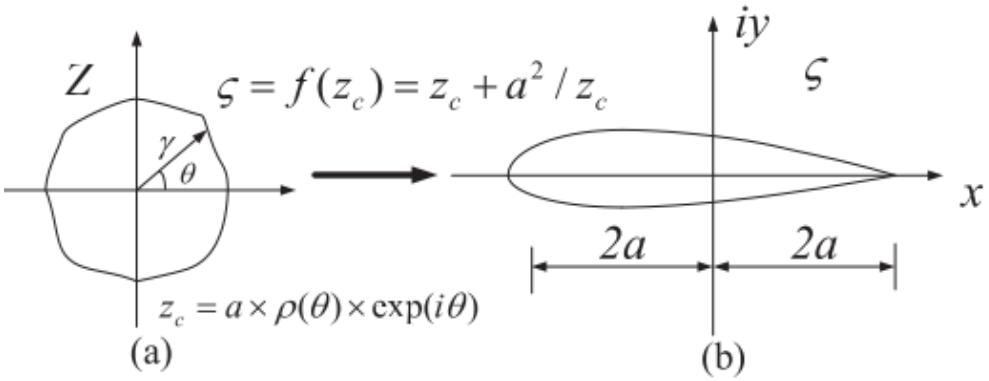


Figure 2.4 Conformal Transformation [11]

Later by defining smooth curvature technique, the corrected value of a can be estimated. In their work, the coefficients of the shape function up to order ten are considered as the design variables.

Concerning the wing shape parameterization, Sobieczky [8] introduced two different concepts in order to define the wing sections: blending support airfoils data and varying generating parameter along the wing span. In the former method, several support airfoils are chosen in some desired cross sections. The wing geometry between the consecutive cross sections are computed by using interpolation scheme. This similar approach is applied by Hicks and Henne [9] in their work in wing numerical optimization as shown in Figure 2.5. In the second method, parameters defining the airfoils are taken to be varying along the span direction. As a results, several sets of parameters are used in this approach.

Hicks and Henne [9] also stated that wing twist can also be considered as a design variable for wing shape parameterization. Later, they also emphasized that wing planform changing such as: aspect ratio, taper ratio, and sweep angle can be recognized as design variables during the optimization. However, during the implementation of these variables, high computational resources might be required as well.

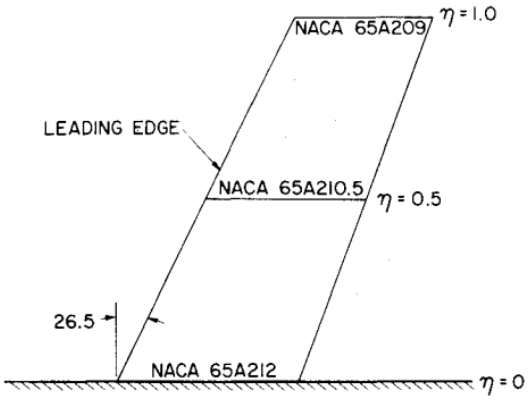


Figure 2.5 Hicks-Henne Wing Paramerization using Some Cross Sections [9]

2.1.2 Optimization Algorithm Scheme

In general, there are two families of optimization algorithms that are applied during aerodynamic optimization: gradient based optimization and genetic algorithm. The algorithms implemented mainly depends on the number of design variables and the availability of computational resources. Dulikravich [12] made comparison between these two algorithms as shown in Figure 2.6. He pointed that gradient based method algorithm is suitable for lower number of design variables since less number of gradient vectors are computed. The implementation of gradient based algorithm with higher number of design variables yields to high computation time caused by the calculation of gradient vectors. On the other hand, the genetic algorithm is more favorable for high number of variables due to the fact no gradient vector computation is required in this scheme.

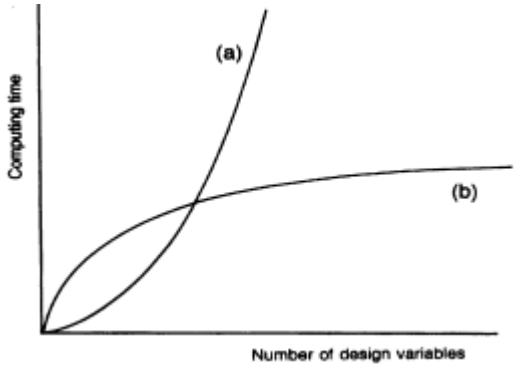


Figure 2.6 Comparison between (a) Gradient Based Algorithm and (b) Genetic Algorithm [12]

The aerodynamic optimization scheme in general is combined with a flow solver. The most common methods used for the flow solver are: Panel Methods [13], Euler Solver [10], or even RANS Solver [14]. This optimization in general is accompanied by a sensitivity analysis in order to enhance the process. Peter and Dwight [15] categorized the methods applied in the sensitivity analysis into several methods, which are: finite difference method, discrete direct method, discrete adjoint method, and continuous adjoint method.

2.2 Mesh Deformation Method

Unsteady flow simulation and numerical design optimization are two cases for which the mesh needs to be updated during the process. Lin et al. [16] categorized three general ways to update the mesh, which are remeshing, mesh deformation, and combination of remeshing and mesh deformation. In the remeshing approach, a new local or global mesh is generated by the aid of mesh generator according to the new geometry domain. On the other hand, the mesh deformation concept changes the nodal location while keeping the nodal connectivity intact.

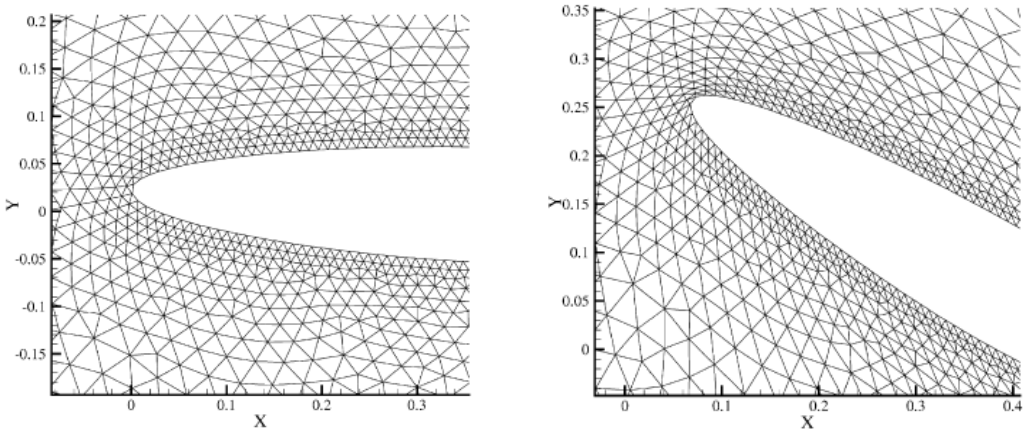
The mesh deformation methods have been developed greatly since Batina [17] who introduced the spring network in the mesh deformation method. There are many different new approaches that have been reported in the literature. These approaches

can be generally categorized as [18]: partial differential equation (PDE) methods, physical analogy methods, algebraic methods, and their combination.

In his work, Luke et al. [19] mentioned that the solver stability and accuracy should be the primary concern during mesh deformation algorithm. During the deformation process, the elements can become inverted or highly skewed which can advance to solver stability problems. Consequently, choosing the appropriate method for the problems should be done by considering the capability of each mesh deformation technique.

2.2.1 Partial Differential Equation Method

In this method, the mesh motion is solved through proposed differential equations using certain boundary conditions. Generally, Laplacian and biharmonic equations are chosen as the partial differential equations. This method mostly works for a problem which requires small deformation since it does not have a high mesh deformation capability [20]. Masud et al. [21] conducted a research using Arbitrary Lagrangian-Eulerian method, which is considered as one of the PDE methods. Figure 2.7 illustrates the result achieved by using this method.



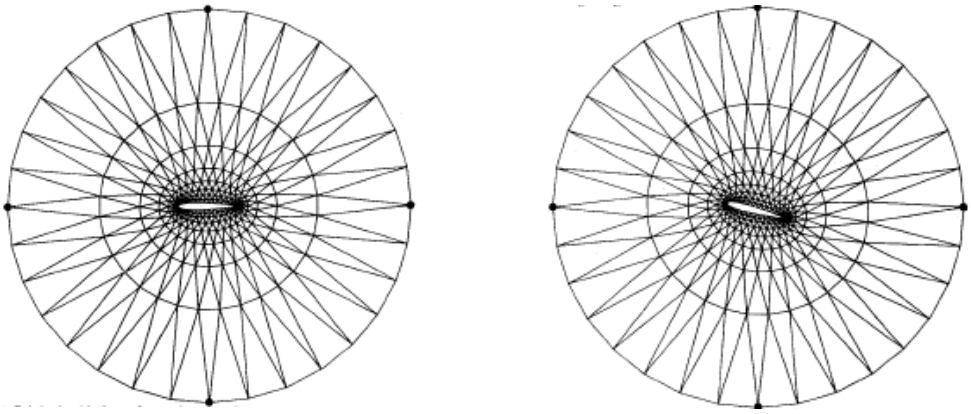
(a) (b)

Figure 2.7 (a) Initial Airfoil Mesh (b) Rotated Airfoil Mesh using ALE Method

2.2.2 Spring Analogy Method

This method is the most commonly used method in the mesh deformation scheme. This is mainly due to the fact that this method can be easily implemented to the problem. Since first introduced by Batina [17], several kinds of spring network concepts have been introduced. Figure 2.8 depicts the result of initial spring analogy method proposed by Batina [17].

The idea used in this method is basically considering each edge on the mesh to behave like a spring which has its own stiffness. Different stiffness definitions have been introduced for comparing one spring analogy method to the other one. Furthermore, many improvements regarding spring analogy methods have also been proposed by other researchers. Details of several spring methods are explained in the later chapter.



(a) (b)

Figure 2.8 Application of Spring Analogy by Batina for Pitching Airfoil (a) Initial (b) 15 Degree Rotation [17]

2.2.3 Algebraic Method

Zhou and Li [20] described the algebraic methods as methods on which the movement of grid nodes are defined as a function of the boundary nodes which has no physical meaning, like the one spring analogy has. They indicated that these methods are more effective compared to the aforementioned techniques. However, this method

is more difficult to be implemented. Several algebraic methods developed so far include: Inverse Distance Weighting Interpolation [22], Delaunay Interpolation [23], and Radial Basis Function Interpolation [24]. Figure 2.9 gives the result of mesh deformation method using radial basis function approach.

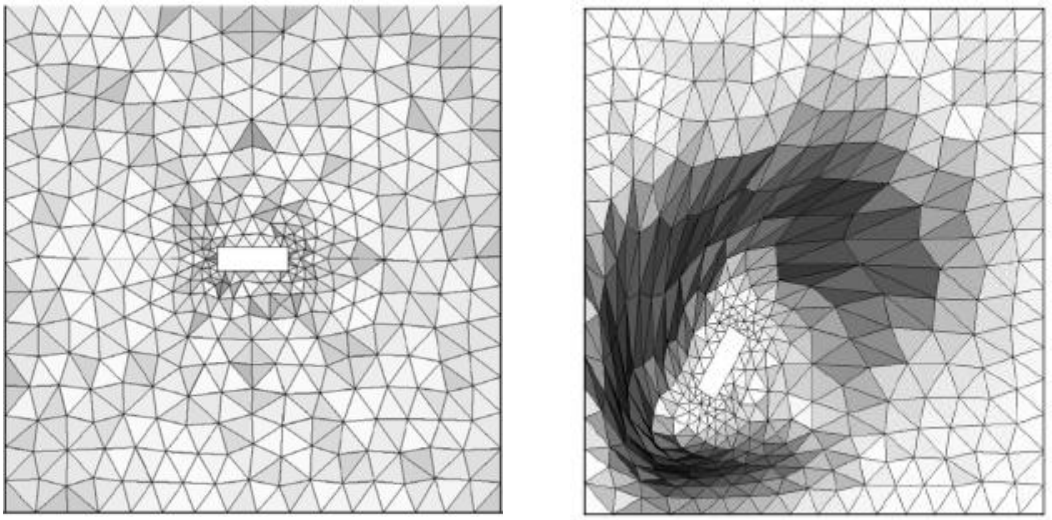


Figure 2.9 Sample of Algebraic Mesh Deformation Method using Radial Basis Function [24]

CHAPTER 3

SPRING ANALOGY MESH DEFORMATION METHOD

3.1 Basic Idea of Spring Analogy Method

Based on the implemented variables for the spring force computation, there are two major types of spring analogy mesh deformation methods: vertex spring analogy method and segment spring analogy method. In the vertex spring analogy method, nodal coordinates are considered as the variables. On the other hand, nodal displacements are used in the segment spring analogy method.

3.1.1 Vertex Spring Method

The idea used in the vertex spring method is by considering each edge as a spring which obeys the linear Hooke's Law. The equilibrium length of the spring is considered as zero in this method. The force exerted on node i by surrounding nodes j can be calculated as:

$$\vec{F}_i = \sum_{j=1}^{v_i} \alpha_{ij}\left(\vec{x}_j - \vec{x}_i\right) \tag{3.1}$$

where

α_{ij} : stiffness of the spring between node i and node j

v_i : number of neighbors of node i

$\vec{x}_i$: position vector of node i

The stiffness coefficient, α_{ij} is taken as constant $(\alpha_{ij} = 1)$.

The equilibrium can be achieved by considering the fact that force summation in each node should be equal to zero. Typical network spring around an arbitrary node i is shown in Figure 3.1. Based on Equation (3.1), the iterative solution for the new position vector of node i can be calculated as:

$$\vec{x}_i^{k+1} = \frac{\sum_{j=1}^{v_i} \alpha_{ij} \vec{x}_i^k}{\sum_{j=1}^{v_i} \alpha_{ij}} \tag{3.2}$$

The boundary conditions for this method is Dirichlet type, which means that the position of boundary nodes are fixed during the iteration procedure. For the interior points, Equation (3.2) needs to be solved iteratively.

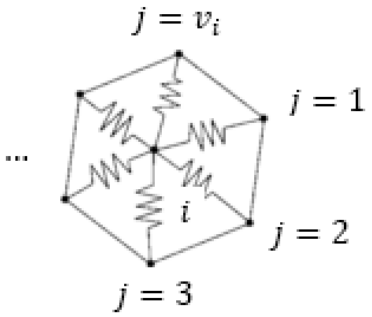


Figure 3.1 Physical Description of Spring Analogy Method [25]

3.1.2 Segment Spring Method

The segment spring method is developed by Batina [17] for deforming the mesh around pitching airfoil. Unlike the former method, this method has the equilibrium length equal to the original length. Moreover, the Hooke's Law for spring is applied to the node displacement instead node position. Mathematically, the force exerted on node i can be written in Equation (3.3).

$$\vec{F}_i = \sum_{j=1}^{v_i} k_{ij}(\vec{q}_j - \vec{q}_i) \tag{3.3}$$

where

k_{ij} : stiffness of the spring between node i and node j

v_i : number of neighbors of node i

$\vec{q}_i$: displacement vector of node i

The stiffness of the spring is proposed to be proportional to the inverse of the edge length. Mathematically, it can be written as:

$$k_{ij} = \frac{1}{\sqrt{(\vec{x}_j - \vec{x}_i) \cdot (\vec{x}_j - \vec{x}_i)}} \tag{3.4}$$

Similar criteria for the equilibrium condition is applied in this method as well. The iterative solution for new displacement vector of node i can be calculated as:

$$\vec{q}_i^{k+1} = \frac{\sum_{j=1}^{v_i} k_{ij}\vec{q}_j^k}{\sum_{j=1}^{v_i} k_{ij}} \tag{3.5}$$

Dirichlet type boundary condition is also applied as the known displacement vectors on the boundary nodes. The final location vector for the node i can be calculated as:

$$\vec{x}_i^{k+1} = \vec{x}_i^k + \vec{q}_i^k \tag{3.6}$$

Compared to the earlier Vertex Spring method, the Segment Spring method requires higher computational memory since displacement vector needs to be stored as well. However, Blom [25] noticed that the former method may lead to the contraction of the mesh near a convex boundary, where a line which connects between

two points inside the boundary still lies inside the boundary. Consequently, Segment Spring Method is more preferable compared to the Vertex Spring Method.

Unlike using iteration procedure shown in Equation (3.6), Botasso et al. [26] introduced the incremental displacement algorithm for updating the displacement vector field. In that algorithm, some scaling factor are used to compute the displacement increment for each iteration based on the final prescribed boundary condition.

3.2 Improvement Over Basic Spring Analogy Method

During the implementation of the spring analogy method, element inversion (node passes through the edge) might occur for a problem with high displacement vector. In order to remedy this issue, some improvements have been proposed so far. Several of the improvement methods are described below.

3.2.1 Angle Consideration in the Linear Spring Formulation

The linear spring formulation described in Section 3.1 lacks the coordinates interaction between x and y, which might not truly represents the spring behavior. Burg [27] proposed an angle made between two spring nodes during the formulation of the force exerted on node, as shown in Figure 3.2. The forces exerted on the nodes are computed based on the local stiffness matrix for each edge. The derivation of this stiffness matrix is very similar to the one used in Finite Element Methods for truss members [28]. The 2-D local stiffness matrix with angle considerations can be written as [27]:

$$K_{ij} = k_{ij}\begin{bmatrix} \cos^2\theta_{ij} & \cos\theta_{ij}\sin\theta_{ij} & -\cos^2\theta_{ij} & -\cos\theta_{ij}\sin\theta_{ij} \\ \cos\theta_{ij}\sin\theta_{ij} & \sin^2\theta_{ij} & -\cos\theta_{ij}\sin\theta_{ij} & -\sin^2\theta_{ij} \\ -\cos^2\theta_{ij} & -\cos\theta_{ij}\sin\theta_{ij} & \cos^2\theta_{ij} & \cos\theta_{ij}\sin\theta_{ij} \\ -\cos\theta_{ij}\sin\theta_{ij} & -\sin^2\theta_{ij} & \cos\theta_{ij}\sin\theta_{ij} & \sin^2\theta_{ij} \end{bmatrix} \quad (3.7)$$

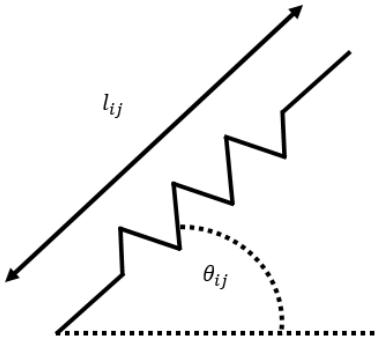


Figure 3.2 Schematic of Angular Consideration in the Linear Spring Analogy

The relation between the force exerted on nodes and displacement vectors for this updated formulation is shown in Equation (3.8).

$$\begin{Bmatrix} F_{ij_x} \\ F_{ij_y} \\ F_{ji_x} \\ F_{ji_y} \end{Bmatrix} = K_{ij} \begin{Bmatrix} \Delta x_i \\ \Delta y_i \\ \Delta x_j \\ \Delta y_j \end{Bmatrix} \tag{3.8}$$

Consequently, the force in x-direction at node i can be computed as:

$$F_{ij_x} = k_{ij}\left[\left(\Delta x_i - \Delta x_j\right)\cos^2\theta_{ij} + \left(\Delta y_i - \Delta y_j\right)\cos\theta_{ij}\sin\theta_{ij}\right] \tag{3.9}$$

The force in y-direction at node i can be computed as:

$$F_{ij_y} = k_{ij}\left[\left(\Delta x_i - \Delta x_j\right)\cos\theta_{ij}\sin\theta_{ij} + \left(\Delta y_i - \Delta y_j\right)\sin^2\theta_{ij}\right] \tag{3.10}$$

By summing the forces exerted on node i from its surrounding nodes separately for each x and y direction, the following equations are obtained:

$$\sum_{j=1}^{v_i} F_{ij_x} = 0$$

$$\left(\sum_{j=1}^{v_i} k_{ij}\cos^2\theta_{ij}\right)\Delta x_i - \left(\sum_{j=1}^{v_i} k_{ij}\cos^2\theta_{ij}\,\Delta x_j\right) + \left(\sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij}\right)\Delta y_i - \left(\sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij}\,\Delta y_j\right) = 0 \tag{3.11}$$

$$\sum_{j=1}^{v_i} F_{ij_y} = 0$$

$$\left(\sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij}\right)\Delta x_i - \left(\sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij}\,\Delta x_j\right) + \left(\sum_{j=1}^{v_i} k_{ij}\sin^2\theta_{ij}\right)\Delta y_i - \left(\sum_{j=1}^{v_i} k_{ij}\sin^2\theta_{ij}\,\Delta y_j\right) = 0 \tag{3.12}$$

Both equations (3.11) and (3.12) are coupled with the same unknown terms Δx_i and Δy_i. These unknown terms are computed by solving these two equations simultaneously. In matrix form, those two equations are shown in Equation (3.13). Solution for this equation can be computed by using any means to solve a 2 x 2 matrix. In this study, Cramer's rule is used to solve this system of equations.

$$\begin{bmatrix} \sum_{j=1}^{v_i} k_{ij}\cos^2\theta_{ij} & \sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij} \\ \sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij} & \sum_{j=1}^{v_i} k_{ij}\sin^2\theta_{ij} \end{bmatrix} \begin{Bmatrix} \Delta x_i \\ \Delta y_i \end{Bmatrix} = \begin{Bmatrix} \sum_{j=1}^{v_i} k_{ij}\cos^2\theta_{ij}\,\Delta x_j + \sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij}\,\Delta y_j \\ \sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij}\,\Delta x_j + \sum_{j=1}^{v_i} k_{ij}\sin^2\theta_{ij}\,\Delta y_j \end{Bmatrix} \tag{3.13}$$

Another solution for angle consideration can also be computed by indirect solution method similar to the procedure proposed by Burg [27] which is related to the solution method for the Finite Element Analysis in truss member solution.

3.2.2 Torsional Spring Analogy

Farhat et al. [28, 29] added additional torsional spring on top of the linear spring definition. This additional spring helps to prevent the cell inversion for large displacement case. The basic idea is to attach each node i, for each triangular cell Ω_{ijk} connected to node i, shown in Figure 3.3, a torsional spring whose stiffness is given by:

$$C_i^{ijk} = \frac{1}{\sin^2 \theta_i^{ijk}} \tag{3.14}$$

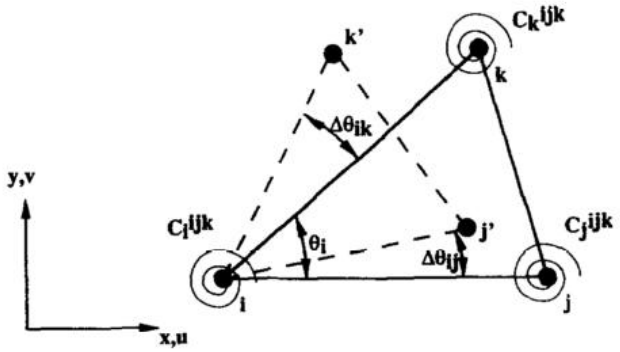


Figure 3.3 Motion and Deformation of a Triangle in the Torsional Spring [29]

The value of $\sin \theta_i^{ijk}$ is computed based on the area computation of triangular cell Ω_{ijk}. The formulation is shown in Equation (3.15).

$$\begin{aligned} A_{ijk} &= \frac{1}{2} l_{ij} l_{ik} \sin \theta_i^{ijk} \\ \sin \theta_i^{ijk} &= \frac{2A_{ijk}}{l_{ij} l_{ik}} \end{aligned} \tag{3.15}$$

In the torsional spring analogy, it is required to have a transformation from the angular displacements into nodal displacements. This is required since the torsional spring analogy only deals with angular displacement [29]. This transformation is achieved by considering both kinematics formulation and equilibrium condition of the torsional spring analogy. The final expression of kinematic formulation for the torsional spring analogy is shown in Equation (3.16) [29]. Detail of the derivation of this matrix is provided in the Appendix A.

$$\Delta\theta^{ijk} = \begin{Bmatrix} \Delta\theta_i^{ijk} \\ \Delta\theta_j^{ijk} \\ \Delta\theta_k^{ijk} \end{Bmatrix} = \underbrace{\begin{bmatrix} b_{ik}-b_{ij} & a_{ij}-a_{ik} & b_{ij} & -a_{ij} & -b_{ik} & a_{ik} \\ -b_{ji} & a_{ji} & b_{ji}-b_{jk} & a_{jk}-a_{ji} & b_{jk} & -a_{jk} \\ b_{ki} & -a_{ki} & -b_{kj} & a_{kj} & b_{kj}-b_{ki} & a_{ki}-a_{kj} \end{bmatrix}}_{R^{ijk}} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_k \\ v_k \end{Bmatrix} \tag{3.16}$$

Similar to the basic spring analogy, the final nodal coordinates are computed based on force equilibrium. In the torsional spring analogy, each node contributes moment forces [29]. These moment forces are defined as shown in Equation (3.17).

$$M^{ijk} = \begin{Bmatrix} M_i \\ M_j \\ M_k \end{Bmatrix} = \begin{bmatrix} C_i^{ijk} & 0 & 0 \\ 0 & C_j^{ijk} & 0 \\ 0 & 0 & C_k^{ijk} \end{bmatrix} \begin{Bmatrix} \Delta\theta_i^{ijk} \\ \Delta\theta_j^{ijk} \\ \Delta\theta_k^{ijk} \end{Bmatrix} \tag{3.17}$$

These moment forces are later transformed by a transformation matrix for each triangular cell Ω_{ijk}, T^{ijk}, into linear force which is defined in Equation (3.18).

$$F^{ijk} = \begin{bmatrix} F_{ix} \\ F_{iy} \\ F_{j_x} \\ F_{j_y} \\ F_{k_x} \\ F_{k_y} \end{bmatrix} = T^{ijk} M^{ijk} \tag{3.18}$$

Based on the fact that work done by force should be equal to work done by moment, the transformation matrix is later shown in Equation (3.19) [29].

$$
\begin{aligned}
F^{ijk^T} q^{ijk} &= M^{ijk^T} \Delta\theta^{ijk} \\
\text{where } F^{ijk} &= T^{ijk} M^{ijk} \text{ and } \Delta\theta^{ijk} = R^{ijk} q^{ijk} \\
M^{ijk^T} T^{ijk^T} q^{ijk} &= M^{ijk^T} R^{ijk} q^{ijk} \\
T^{ijk} &= R^{ijk^T}
\end{aligned}
\tag{3.19}
$$

Therefore, the expression for linear force due to the torsional spring analogy is shown in Equation (3.20).

$$
F^{ijk} = \underbrace{\left[R^{ijk^T} C^{ijk} R^{ijk}\right]}_{K^{ijk}_{torsion}} q^{ijk} \tag{3.20}
$$

The final force equilibrium is achieved by combining the forces arising from linear spring and torsional spring for each edge in the mesh. Instead of using the solution method proposed by Farhat et al. [29], where contribution from each triangular cell to each edge is considered, a different solution method is proposed.

In the proposed solution, a similar approach like done in Finite Element Analysis, each triangular cell is considered as an element which has a 6 x 6 local stiffness matrix $K^{ijk}_{torsion}$. Details regarding the implementation of this solution method are elaborated in Section 3.3.

3.2.3 Semi-Torsional Spring Analogy

The improvement method shown in the previous section requires a complicated formulation to be done. Zeng [31] introduced the notion of the semi-torsional spring method. This method behaves like the linear spring method with angle information incorporated into the spring stiffness.

The stiffness of the spring edge is defined as the superposition of linear spring defined earlier and the semi-torsional spring. The linear spring is exactly similar to the one defined in Equation (3.4). Mathematically, this can be written as:

$$k_{ij}^{\text{total}} = k_{ij} + k_{ij}^{\text{semi-torsional}}$$

where

$$k_{ij}^{\text{semi-torsional}} = \lambda \sum_{m=1}^{NE_{ij}} \frac{1}{\sin^2 \theta_m^{ij}} \tag{3.21}$$

For a triangular 2-D cell shown in Figure 3.4, the spring forces on the edge $i - j$ are calculated as[31]:

$$[F_{ij}] = \left(\frac{1}{l_{ij}} + \kappa \left(\frac{1}{\sin^2 \theta_1} + \frac{1}{\sin^2 \theta_2} \right) \right) [B^*][q_{ij}]$$

$$[F_{ij}] = \begin{bmatrix} F_{ix} \\ F_{iy} \\ F_{jx} \\ F_{jy} \end{bmatrix} \quad [B^*]_{4x4} = \delta_{pq} - \delta_{p,q+2} - \delta_{p+2,q} \tag{3.22}$$

$$[q_{ij}] = \begin{bmatrix} \Delta x_i \\ \Delta y_i \\ \Delta x_j \\ \Delta y_j \end{bmatrix}$$

δ_{pq} is a Kronecker's Delta

$$[B^*]_{4x4} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \tag{3.23}$$

In this method that proposed by Zeng [31], the matrix $[B^*]$ defined for the computation is similar to the idea of basic spring analogy. For the implementation in this study, this method is later improved by adding the edge angle into the consideration as well. As a result, the matrix $[B^*]$ defined earlier is changed into the matrix shown in the Equation (3.7).

Compared to the previous torsional spring method, this method only includes 2 angles (θ_1 and θ_2). Consequently, the data saving will be less compared to the previous method.

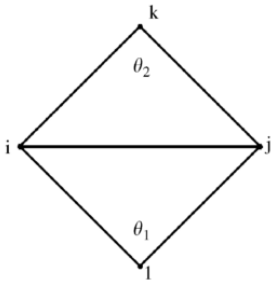


Figure 3.4 Angle Definition Used in 2-D Semi Torsional Spring

3.2.4 Ball-Center Spring Analogy

The idea of the Ball-Center spring analogy comes from the idea proposed by Bottasso et al. for Ball-Vertex spring analogy method [26]. In their approach, some additional linear springs are introduced to resist the motion of a mesh node towads its region-opposed faces. This Ball-Vertex spring analogy method is introduced by connecting node i to its projection p on the plane of the face F_i, opposite of node i. For more clarity, the location of projection point p can be seen on Figure 3.5.

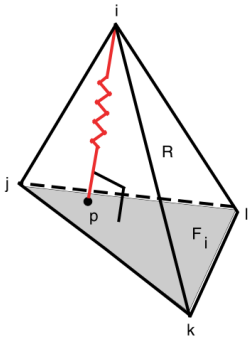


Figure 3.5 Location of Projection Point p on the face F_i

In the Ball-Center Spring Analogy itself, instead of creating a linear spring based on node i and its projection on the opposite plane, the additional spring will be

created from the node i and the center of the cell of a triangular cell in 2-D). The detail of this proposed method is shown in Figure 3.6 [32].

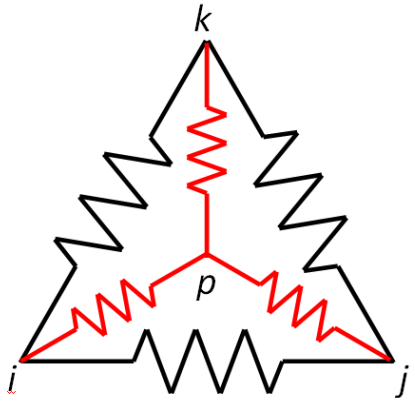


Figure 3.6 Schematic of Ball-Center Spring Analogy for 2-D Unstructured Mesh

In 2-D formulation, the center of the cell is assumed to be located at centroid of the triangular cell. The location and displacement of center node in a triangular cell Ω_{ijk} is formulated as in Equation (3.24).

$$\vec{x}_p = \frac{\vec{x}_i + \vec{x}_j + \vec{x}_k}{3}$$
$$\vec{q}_p = \frac{\vec{q}_i + \vec{q}_j + \vec{q}_k}{3} \tag{3.24}$$

The resulting force on node i by fictitious node p is defined in the same manner like spring force defined in the basic segment spring method. Mathematically, the spring force resulted from this fictitious spring is computed as in Equation (3.25).

$$\vec{F}_{ip} = k_{ip}\left(\vec{q}_p - \vec{q}_i\right) \tag{3.25}$$

In a similar manner like done in the angle consideration in the spring analogy, the force exerted due to the fictitious spring analogy is shown in Equation (3.26).

$$
\begin{aligned}
F_{ip_x} &= k_{ip}\left[\left(\Delta x_i - \Delta x_p\right)\cos^2\alpha_{ip} + \left(\Delta y_i - \Delta y_p\right)\cos\alpha_{ip}\sin\alpha_{ip}\right] \\
F_{ip_y} &= k_{ip}\left[\left(\Delta x_i - \Delta x_p\right)\cos\alpha_{ip}\sin\alpha_{ip} + \left(\Delta y_i - \Delta y_p\right)\sin^2\alpha_{ip}\right]
\end{aligned}
\tag{3.26}
$$

The final equilibrium equation is computed by considering the contribution from actual spring edge and fictitious edge shown previously. Details of mesh configuration used in the ball-center spring analogy is depicted in Figure 3.7. This means that one needs to solve all equations (3.9), (3.10), and (3.26) in *x*-direction and *y*-direction simultaneously for each node. The system of linear equation which governs the updated displacements based on this updated method is shown in Equation (3.27).

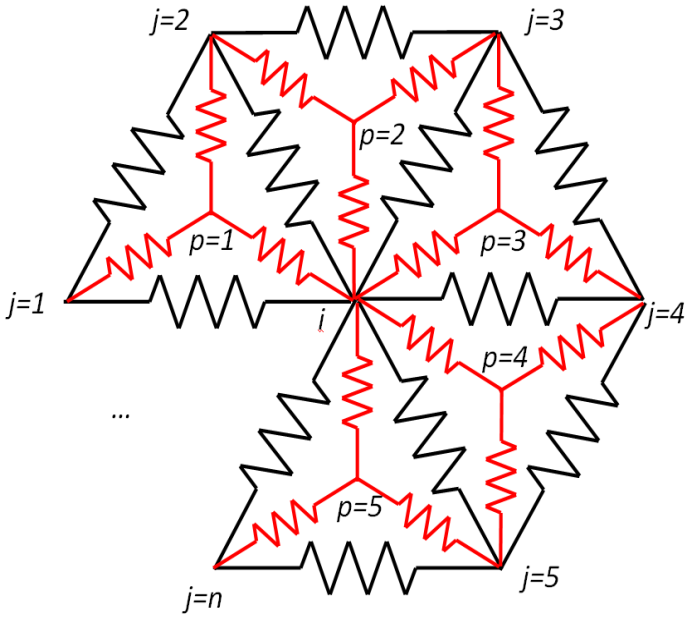


Figure 3.7 Schematic of Ball-Center an Arbitrary Node i

$$
\begin{aligned}
&\begin{bmatrix}
\sum_{j=1}^{v_i} k_{ij}\cos^2\theta_{ij} + \sum_{p=1}^{v_c} k_{ip}\cos^2\alpha_{ip} & \sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij} + \sum_{p=1}^{v_c} k_{ip}\cos\alpha_{ip}\sin\alpha_{ip} \\
\sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij} + \sum_{p=1}^{v_c} k_{ip}\cos\alpha_{ip}\sin\alpha_{ip} & \sum_{j=1}^{v_i} k_{ij}\sin^2\theta_{ij} + \sum_{p=1}^{v_c} k_{ip}\sin^2\alpha_{ip}
\end{bmatrix}
\begin{bmatrix}\Delta x_i \\ \Delta y_i\end{bmatrix} \\
&= \begin{bmatrix}
\sum_{j=1}^{v_i} k_{ij}\cos^2\theta_{ij}\Delta x_j + \sum_{p=1}^{v_c} k_{ip}\cos^2\alpha_{ip}\Delta x_p + \sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij}\Delta y_j + \sum_{p=1}^{v_c} k_{ip}\cos\alpha_{ip}\sin\alpha_{ip}\Delta y_p \\
\sum_{j=1}^{v_i} k_{ij}\cos\theta_{ij}\sin\theta_{ij}\Delta x_j + \sum_{p=1}^{v_c} k_{ip}\cos\alpha_{ip}\sin\alpha_{ip}\Delta x_p + \sum_{j=1}^{v_i} k_{ij}\sin^2\theta_{ij}\Delta y_j + \sum_{p=1}^{v_c} k_{ip}\sin^2\alpha_{ip}\Delta y_p
\end{bmatrix}
\end{aligned}
\tag{3.27}
$$

The above equation is later solved by the mean of Cramer's rule, similar to the solution method used in the case for angle consideration in spring analogy technique.

3.2.5 Boundary Improvement

The idea in this approach is application of the Saint-Venant principle for mesh deformation [25]. Consequently, the boundary displacements only have local impact and do not spread far into the mesh. Mathematically, this improvement is shown in Equation (3.28).

$$k_{ij} = \frac{\phi}{\left[(\vec{x}_j - \vec{x}_i) \cdot (\vec{x}_j - \vec{x}_i)\right]^{\Psi}} \tag{3.28}$$

The idea is to increase the stiffness of the springs around to the boundary by using $\phi = 5$ or decreasing the value of Ψ to 0.05. This may help to prevent the spreading displacement far into the mesh [25].

In this study, two different cases are considered during the implementation of the Saint-Venant principle:

- Adjacent Boundary Improvement

  The improvement for this method is only applied to the edge whose one of the node is located on the airfoil surfaces.

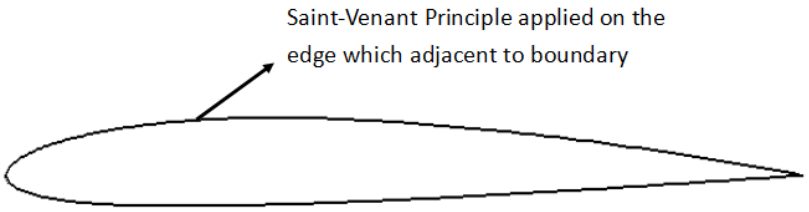


Figure 3.8 Adjacent Boundary Improvement in the Spring Analogy Method

- Surrounding Boundary Improvement

  In this case, the stiffness increasing is applied to some region around the airfoil boundary. The region is bounded by the airfoil boundary and the

designated box whose dimensions are shown in Figure 3.9. The chosen length and width of the designated box also encloses the viscous mesh region around the airfoil.

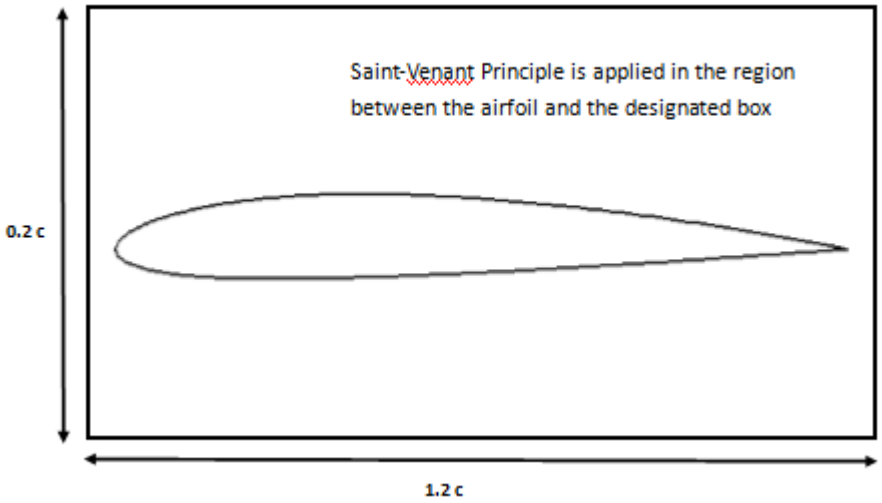


Figure 3.9 Surrounding Region Boundary Improvement in the Spring Analogy Method

In the case of ball-center spring analogy, the improvement in the spring constant is treated differently since the edge is shorter compared to the actual edge. Similar formulation like shown in Equation (3.28) is applied as well with different values for ϕ and Ψ. The values chosen for ϕ and Ψ are 10 and 0.01, respectively for the fictitious edges.

3.3 Solution Method

This section is mainly related to the numerical solution of the final formulation of the spring analogy method. The displacement of the movable nodes became the values that should be computed. These values can be computed by using two different approaches: direct solution and indirect solution. In the coding implementation, the solution methods are classified based on Figure 3.10.

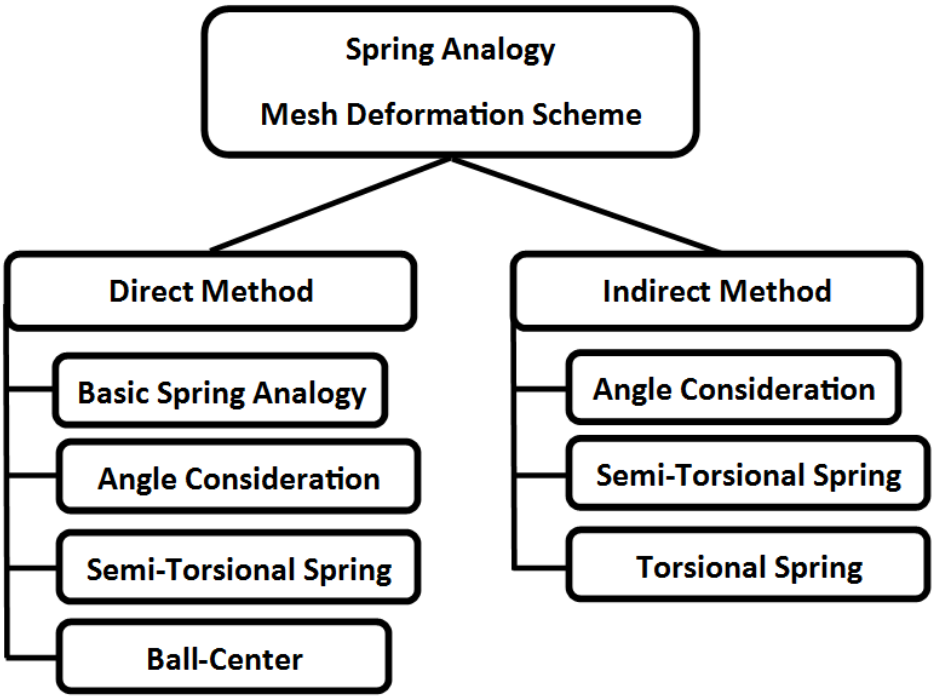


Figure 3.10 Implemented Numerical Methods in Spring Analogy

3.3.1 Direct Solution

In the direct solution approach of this method, each movable node is visited and the displacement corresponding to this node is computed. In other words, it solves each displacement value of the nodes directly in a vertex-by-vertex fashion using an iterative manner as shown in Equation (3.5). Some improvement could also be accomplished in this method by introducing some relaxation parameter similar to Successive Overrelaxation (SOR) method. The improvement using SOR method for the direct solution is shown in Equation (3.29). The convergence of this method is determined based on the residual value of the computed nodal displacement for each nodes.

$$\vec{q}_i^{k+1} = \vec{q}_i^{k} + \omega \left(\underbrace{\frac{\sum_{j=1}^{v_i} k_{ij}\vec{q}_j}{\sum_{j=1}^{v_i} k_{ij}}}_{\text{updated term}} - \vec{q}_i^{k} \right) \tag{3.29}$$

where ω = the relaxation parameter

In the case where the angle made by spring is considered during the computation, a slight modification is required for the computation. Direct solution for the angular consideration is performed by solving 2 x 2 matrix from Equation (3.13). By solving this equation for each node, one gets the nodal displacements in *x* and *y* directions. The solution found from the solution of 2 x 2 matrix is substituted into the updated term defined in Equation (3.29).

3.3.2 Indirect Solution

The indirect method here refers to solving the displacement value for each node by means of the local stiffness matrix. This method is very similar to Finite Element Method used in Structural Analysis [28]. The local stiffness matrices for each edge are combined together into a global stiffness matrix. This method has been applied by Burg [27] and Markou et al. [33] for their work in 3-D mesh deformation. The global stiffness matrix later is partitioned into several partitioned matrices corresponding to either prescribed degree of freedoms or active degree of freedoms.

In this method, one requires to assemble the global stiffness matrix based on the local stiffness matrix. The assemble process are based upon the method proposed by Cook [28]:

- Generation of ID Array Matrix

  This matrix is needed to determine whether a given degree of freedom in a node is prescribed or active degree of freedom (unknown displacement). Since 2-D mesh deformation is considered, each node has only 2 degree of freedoms; *x* and *y* displacements of the node. The

number of columns that this matrix has is corresponding to the number of nodes in the given mesh. On the other hand, the number of rows corresponding to the number of degree of freedoms that each node has, equals to two. Binary numbers are considered as the input for this ID array matrix; input equals to one for the nodes located on the prescribed boundary condition and equals to zero for the other condition.

- Generation of Destination Array

  The destination array is generated in order to number the degree of freedoms for all nodes that are used in the computation. There are two separate arrays used in here: one destination array is corresponding to active degree of freedoms and the other one is corresponding to the prescribed degree of freedoms.

- Global Stiffness Matrix Assemble

  The global stiffness matrix is assembled based on the information found from destination array for both active and prescribed degree of freedoms. In each local stiffness matrix, each entry corresponds to a specific degree of freedom in the global stiffness matrix. The global stiffness matrix can be partitioned in such a way that degree of freedoms corresponding to the active degree of freedoms are numbered first in the column. As a result, this matrix can be written as:

$$K = \begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}$$
$$\begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix} \begin{bmatrix} q_a \\ q_b \end{bmatrix} = \begin{bmatrix} 0 \\ R_b \end{bmatrix} \tag{3.30}$$

The subscript a corresponds to the active degree of freedoms, while the subscript b corresponds to the prescribed degree of freedoms. In the implementation, instead of dealing with a big stiffness matrix K, the partitioned

matrices K_{aa} and K_{ab} are used as the help to compute the active degree of freedoms. As a result, assemble of partitioned matrices are considered here. Assemble of active stiffness matrix, K_{aa}, is based on the Algorithm 1 shown below.

Algorithm 1. Assemble Process of Active Stiffness Matrix K_{aa}

Input: all edges with local stiffness matrix
Output: active stiffness matrix K_{aa}

```
1: for each iter_n in [1,edge_number]  do
2:     set node1 = 1st node of  edge(iter_n)
3:     set node2 = 2nd node of edge(iter_n)
4:
5:     dof_array(1) = dest_array_1(1, node_1)
6:     dof_array(2) = dest_array_1(2, node_1)
7:     dof_array(3) = dest_array_1(1, node_2)
8:     dof_array(4) = dest_array_1(2, node_2)
9:
10:    for each iter_i in [1,4] do
11:        if dof_array(iter_i) > 0 then
12:          set index_i = dof_array(iter_i)
13:          for each iter_j in [1,4] do
14:              if dof_array(iter_j) > 0 then
15:                 set index_j = dof_array(iter_j)
16:                     Kaa(index_i, index_j) += stiff_matrix (iter_i, iter_j)
                                                 of edge(iter_n)
17:             end if
18:          end for
19:       end if
20:    end for
21: end for
```

In a similar fashion, the assemble process of matrix K_{ab} is conducted as well. The unknown displacement is later computed based on the solution from the first row of Equation (3.28). Mathematically, the displacements corresponding to the active degree of freedom are computed as:

$$\begin{aligned} K_{aa}q_a + K_{ab}q_b &= 0 \\ q_a &= -K_{aa}^{-1}[K_{ab}q_b] \end{aligned} \tag{3.31}$$

The active stiffness matrix K_{aa} is a symmetric matrix since the assemble process is based on the symmetric matrix shown in Equation (3.7). Furthermore, the active stiffness matrix is a sparse matrix since not all nodes are connected to each other. This makes some of entries in the active stiffness matrix are equal to zero. In order to solve the unknown displacement, Conjugate-Gradient Method [34] is applied on the solution procedure. The Conjugate-Gradient algorithm is explained in Appendix B.1.

The above approach is also applied to the torsional spring analogy formulation. In the implementation, instead of dealing with the local stiffness matrix (4 x 4) for each edge, a local stiffness matrix (6 x 6) is considered. However, the idea of global matrix assemble similar to the one implemented above is considered in here as well.

For a better clarification, a sample case regarding global stiffness assemble process for both angle consideration and torsional spring analogy is elaborated in Appendix C.

3.4 Coding Implementation of Spring Analogy

After a brief explanation about the spring analogy mesh deformation method in the previous sections, this section describes how this method is implemented in the code. The flow chart used in the spring analogy mesh deformation technique is shown in Figure 3.11

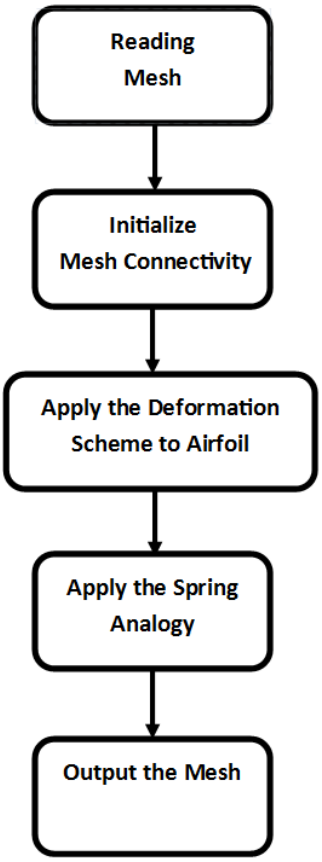


Figure 3.11 Flow Chart Implemented in the Code

3.4.1 Implemented Data Structure

In order to enhance the computational procedure, the capability of derived data type in FORTRAN 95 is implemented. The data type used in the computational procedure mainly divided into three different big data types: cells, edge, and nodes. The information contained in each data type is summarized in Table 3.1. Each component in the data type is accessed by using the "%" operator. It can be seen very clearly that one can assess the coordinate of a node in a given triangular cell based on the data structure used in here.

Table 3.1 Implemented Derived Data Type in Mesh Deformation Code

Cell Data	Edge Data	Node Data
Cell Number Cell Nodes Cell Neighbors Cell Edges Cell Center Coord. Cell Area	Edge Number Edge Nodes Edge Length Edge Angle Edge Opposite Nodes Edge Opposite Angles Edge Adjacent Cells Edge Spring Value Edge Stiffness Matrix	Node Number Node Coordinates Node Neighbors Node Adjacent Cells Node Adjacent Edges Node Adjacent Fictitious Edges

3.4.2 Mesh Connectivity

One of the main interest in the spring analogy mesh deformation technique is to know the neighbor nodes of a given node. This information can be perceived by mesh connectivity. The meshing connectivity is perceived based on the native mesh format of .su2 mesh file. Basically, the information contained in the native mesh file are comprised of three different groups:

- Element Connectivity
  This contain the information about each triangular cell used in the mesh and all nodes which define the triangular element.

- Node Coordinates
  This contains the information about the coordinates of all nodes defined in the mesh used during the computation

- Boundary Condition
  This part contains the information about the boundary condition defined for boundary region of the solution domain.

In order to simplify the nodal connectivity, CFD element mesh is numbered based on the standard numbering convention for its vertices and edges. Figure 3.12

illustrates the nodal and edge numbering convention for a triangular element. The underscored numbers correspond to the local edge numbering inside a triangular cell.

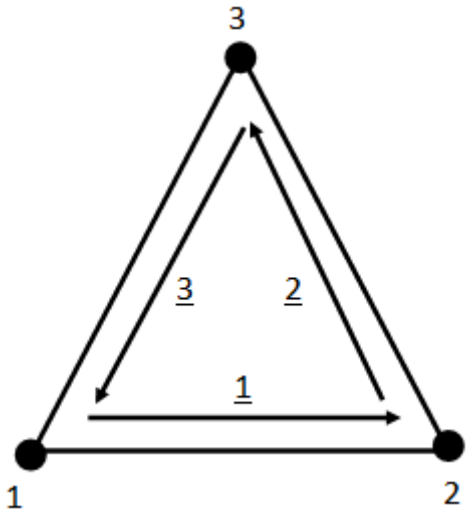


Figure 3.12 Standard Numbering Convention for a 2-D Triangular Element

This spring analogy mesh deformation method mainly deals with the edge as the main component. On the other hand, the mesh information is mainly based on the triangular element. To provide the edge information, one should equip the edge information data. The foundation of this information is based on the edge numbering process. The algorithm for numbering process is shown in Algorithm 2 and Figure 3.13. The *edge_temp* is a temporary edge data structure which has the similar contents to the edge data structure described in Table 3.1.

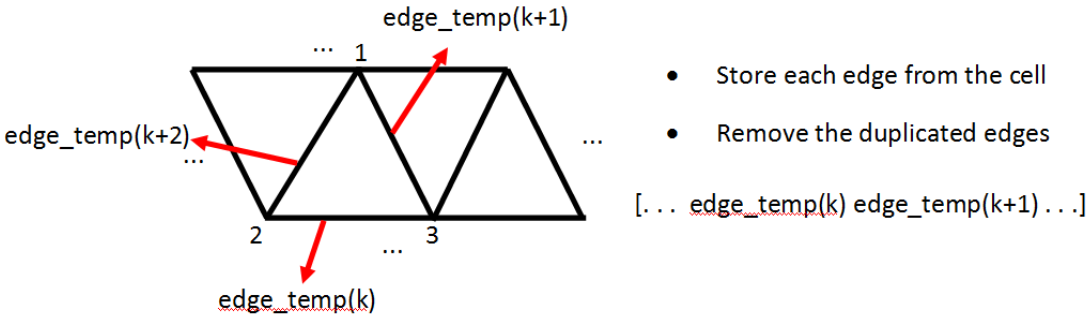


Figure 3.13 Actual Edges Numbering System

Algorithm 2. Edge Numbering based on the Triangular Element Data

Input: element connectivity (mesh)
Output: edge numbering

```
 1: set temp = 0
 2: for each cell in mesh do
 3:     for each node in cell do
 4:         temp = temp + 1
 5:         sort the remaining node number from small to big
 6:         append other nodes into edge_temp (temp) data
 7:     end for
 8: end for
 9: set edge_number = 1
10: set edge(1) = edge_temp(1)
11: for each iter_i in [2, temp] do
12:     for each iter_j in [1,edge_number]
13:         if edge(iter_j) == edge_temp(iter_i)  cycle for 11:
14:     end for
15:     edge_number = edge_number + 1
16:     edge(edge_number) = edge_temp(iter_i)
17: end for
18: set total_edge = edge_number
```

After numbering the edge, it is required to compute the list of adjacent edges for a given node. This computation is done based on Algorithm 3.

Algorithm 3. Adjacent Edges Computation for a Given Node

Input: updated element connectivity (mesh) from **Algorithm 2**
Output: list of adjacent edges for all nodes in the mesh

```
 1: initialize number of adjacent edges for each node to be zero
 2: for each edge in mesh do
 3:     increase number of adjacent edges of node in edge by one.
 4: end for
 5:
 6: for each node in mesh do
 7:     set index_adj_edge = 0
 8:     for each edge in mesh do
 9:         if one of the node number in edge equals to node number of node then
10:             index_adj_edge += 1
11:             set edge as the adjacent edge of node in position of index_adj_edge
12:         end if
13:         if index_adj_edge equals to number of adjacent edges then
14:             exit loop for 8:
```

```
15:          end if
16:      end for
17: end for
```

It is also required to know about the neighbor nodes of a given node. This can be easily found from the list of adjacent edges of a given node found in Algorithm 3. The neighbor nodes is the other node stored in the adjacent edge.

In the case where the Ball-Center Spring Analogy is concerned, it is required to know about the adjacent cells for a given node. This computation is done based on Algorithm 4.

Algorithm 4. Adjacent Cells Computation for a Given Node

```
Input: updated element connectivity (mesh) from Algorithm 2
Output: list of adjacent cells for all nodes in the mesh
 1: initialize number of adjacent cells for each node to be zero
 2: for each cell in mesh do
 3:      increase number of adjacent cells of each node in cell by one.
 4: end for
 5:
 6: for each node in mesh do
 7:      set index_adj_cell = 0
 8:      for each cell in mesh do
 9:          if one of the node number in cell equals to node number of node then
10:              index_adj_cell += 1
11:              set cell as the adjacent edge of node in position of index_adj_cell
12:          end if
13:          if index_adj_cell equals to number of adjacent cell then
14:              exit loop for 8:
15:          end if
16:      end for
14: end for
```

Another data type is also required to compute the fictitious edges in the ball-center spring analogy. These fictitious edges are stored in another data set, similar to the edge data shown in Table 3.1. The main difference in this data type is the node used. This fictitious edge connects the actual node to the fictitious center of triangular cell. Similar algorithm that was used in the actual edges numbering is also

implemented in these fictitious edge numbering as shown in Algorithm 5. This algorithm is conducted based on Figure 3.14.

Algorithm 5. Fictitious Edge Numbering

Input: Element Connectivity with Edge Numbering from **Algorithm 2**
Output: Fictitious Edge Numbering

```
set num_edge_fict = 3 times number of cells
for each iter_i in [1, num_edge_fict]:
    set index_cell = ⌈iter_i/3⌉ − 1
    if (mod (iter_i,3) == 0) then
        set index_node = mod (iter_i,3)  + 3
    else
        set index_node = mod (iter_i,3)
    end if
    set index_node and index_cell as edge node for edge_fict (iter_i)
end for
```

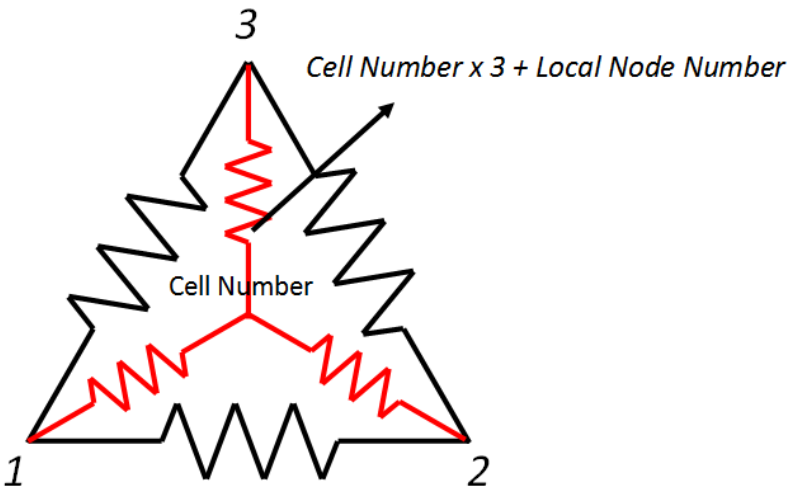


Figure 3.14 Fictitious Edge Numbering System Used in the Ball-Center Spring Analogy

Ball center spring analogy method also requires the information regarding the neighbor fictitious cell center nodes around an arbitrary node as depicted in Figure 3.7. This information is stored in the fictitious edge data and can be perceived based on Algorithm 6.

Algorithm 6. Finding Number of Fictitious Edges Surrounding Node *i*

Input: Fictitious Edge Numbering from **Algorithm 4.**
Output: list of adjacent fictitious edge for all nodes in the mesh

```
1: initialize number of adjacent edges for each node to be zero
2: for each node in mesh do
3:     set num_adj_fict_edge = num_adj_cell
4:     for each iter_j in [1, num_adj_fict_edge]:
5:         set index_cell = the adjacent cell in order of iter_j of node
6:         for each iter_k in [1,3]:
7:             if node number of node = cell node in in order of iter_k of
                   index_cell
8:                 set index_node = iter_k
9:                 exit iteration 6:
10:            end if
11:        end for
12:        set adjacent fictitious edge as  index_cell*3+index_node
13:    end for
14: end for
```

In order to make the user easily interacts the code, a basic input file is defined. The input file contains the information concerning about the mesh deformation parameters, and design variables used in the optimization scheme. Sample of the input file used in the mesh deformation code is attached in the Appendix D.1.

CHAPTER 4

CFD AND OPTIMIZATION ANALYSES

4.1 CFD Analyses

The Computational Fluid Dynamic (CFD) analysis is conducted in order to compute the aerodynamic coefficients of the airfoil that are required in the optimization scheme. This analysis is performed by using the aid of SU2 (Stanford University Unstructured) CFD Solver [1]. In order to get an accurate drag computation, instead of using inviscid flow solver, RANS solver combined with Spallart-Almaras turbulence modelling is implemented. The equations used in SU2 is shown briefly in Appendix E.

SU2 CFD solver requires two different input in order to be able to perform the analysis: configuration file (.cfg file) and native .su2 mesh file. The sample of configuration file used in the analysis is shown in Appendix D.2. The initial native .su2 mesh is directly attained from Pointwise mesh generation software by defining the appropriate boundary condition used in the solver.

The CFD analysis is conducted in parallel by using parallel computation capability of SU2 CFD solver. In each parallel computation, the aim is to find the final aerodynamic coefficients of the airfoil by satisfying the required lift coefficient. In other words, regardless the initial angle of attack entered by the user, the solver tries to find the corresponding final angle of attack in order to generate sufficient lift coefficient.

4.1.1 Mesh Generation

As mentioned earlier, the mesh is generated by using Pointwise® Mesh Generation Software [35]. Farfield domain is modelled as a circle whose radius is taken as 12 times of the chord length. Figure 4.1 depicts the farfield domain used in the computation procedure. There are two separate types of meshes consider during the computational procedure: inviscid and viscous mesh. The viscous mesh is used for RANS simulation. Figure 4.2 and Figure 4.3 show the inviscid mesh and viscous mesh used in the analysis performed in this study.

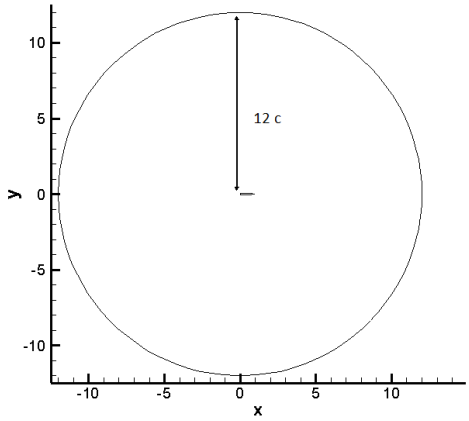
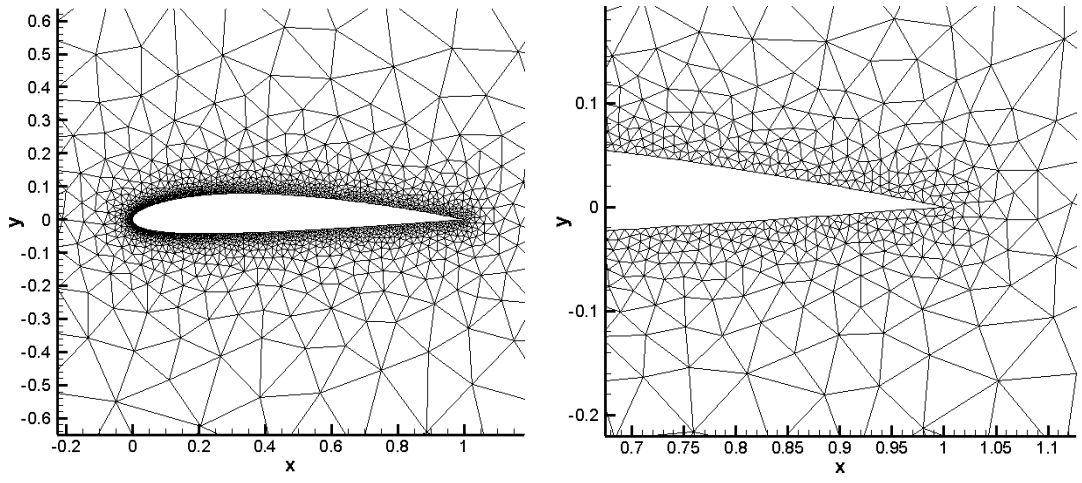


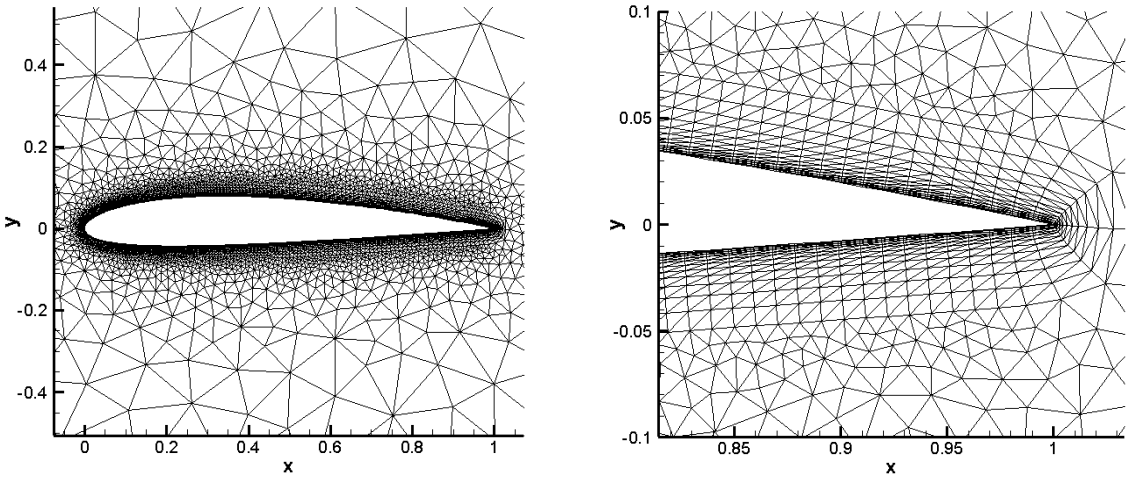
Figure 4.1 Farfield Domain Description Used in the Mesh Generation



(a) Outer View (b) Zoom View Near Trailing Edge

Figure 4.2 Inviscid Mesh around Baseline Airfoil

The terms inviscid and viscous meshes here are used to describe the corresponding required mesh to perform inviscid or viscous simulation in CFD, respectively. The inviscid mesh is used only to check the capability of mesh deformation method. This inviscid mesh is not going to be applied in the optimization analysis. Only the viscous mesh shown in Figure 4.3 is considered during the RANS simulation used in the airfoil design optimization. Numbers of cells and nodes used in the inviscid mesh are 6072 and 3234, respectively. On the other hand, the numbers of cells and nodes used for the viscous mesh are 12060 and 6228, respectively.



(a) Outer View (b) Zoom View Near Trailing Edge

Figure 4.3 Viscous Mesh around the Baseline Airfoil

4.1.2 Flow Parameters in CFD

The airfoil design optimization is applied for an airfoil whose flow parameters are computed based on the flight conditions defined in the CHANGE FP7 project [36], an European Union project which combines several morphing capabilities into one wing. Basically, there are 4 different flight regimes considered in this study: take-off, loiter, high speed, and landing. The summary of flow properties used in each flight regime is tabulated in Table 4.1.

Table 4.1 Flow Properties used in the Optimization Analysis

	Take-Off	Loiter	High Speed	Landing
Velocity [m/s]	21.164	15.278	30.556	13.244
Density [kg/m^3]	1.225	1.1895	1.1895	1.1895
Altitude [feet]	0	1000	1000	1000
Reynolds Number	858441	605075	1210135	524536
Mach Number	0.0622	0.0451	0.090	0.039

For all above flow properties used, it is assumed that the baseline aircraft has a span and chord whose lengths are 4 m and 0.6 m, respectively. Furthermore, the aircraft's mass is taken as 25 kg. Based on this information, the airfoil's target sectional lift is computed based on Equation (4.1). It is found that the target sectional lift for the airfoil is 61.3125 N/m.

$$\text{Target Sectional Lift} = \frac{\text{Aircraft Weight}}{\text{Aircraft Span}} \tag{4.1}$$

In the optimization procedure, the same sectional lift is applied for all different flow parameters. However, the target lift coefficient for each flight parameter is determined based on the corresponding velocity. The target lift coefficient is computed based on Equation (4.2).

$$\text{Sectional Lift Coefficient} = \frac{\text{Target Sectional Lift}}{0.5\rho V_\infty^2 c} \tag{4.2}$$

4.2 Optimization Analyses

The optimization procedure is achieved by utilizing Phoenix ModelCenter Optimization Software. The optimization is performed by making some modules which wrap each component of optimization procedure. In the optimization case, there are 3 different modules considered during the optimization process. Figure 4.4 depicts the order and relation between these modules during the optimization procedure. The input required in the optimization is entered manually from the Component Tree in the ModelCenter as shown in Figure 4.5.

- Input Module

  This module provides the information about the input parameters used for the CFD computation. These parameters comprised of: air density, velocity, viscosity, and the required sectional lift for the computation. For different flight conditions, different values of flight velocity is manually entered in the module.

- Mesh Deformation Module

  This module mainly wraps the mesh deformation code that is prepared earlier. The module contains the information about parameter used during the mesh deformation analysis. The parameters used in this module are the input parameters used in the code as shown in Appendix D.1.

- CFD Solver Module

  This module contains the information about the input parameters used in the SU2 CFD solver. This module mainly contains about the simulation parameters: number of processors, iteration counter.

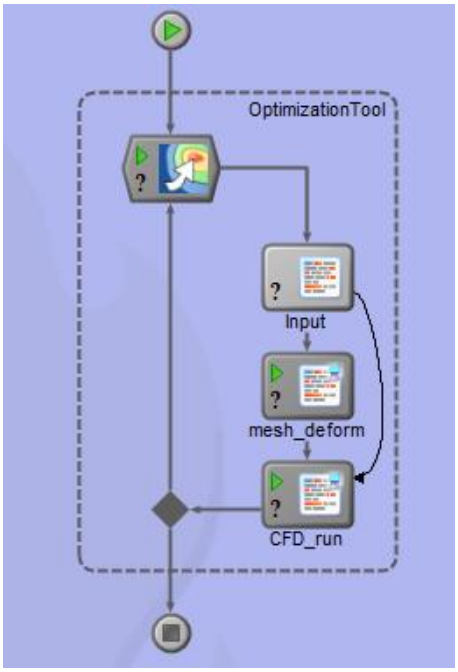


Figure 4.4 Optimization Scheme Implemented in Model Center

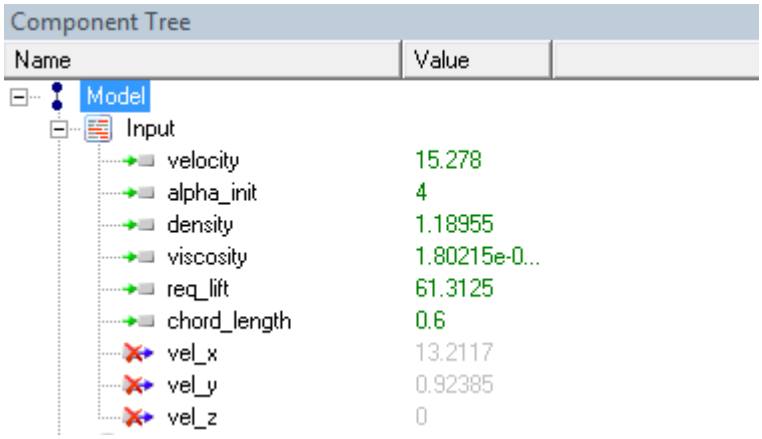


Figure 4.5 Component Description in Phoenix ModelCenter for Input Module

4.2.1 Optimization Scheme Explanation

The optimization was done by using Gradient Based Optimization Solver from Phoenix Model Center Optimization Module [37]. OPTLIB Gradient Optimizer which is considered as gradient based optimization is considered as the optimizer.

OPTLIB implements Sequential Quadratic Programming (SQP) in the optimizations scheme. Furthermore, the gradient value is computed based on the finite difference concept. The initial step size used in the gradient computation is approximated as 0.0001. However, OPTLIB optimizer later can handle the appropriate step size used for the gradient computation.

The main objective in the optimization is to minimize the sectional drag of an airfoil and satisfy the sectional lift requirement of the airfoil for different flow parameters. Furthermore, an additional angle of attack is also imposed for each case. The angle of attack constraints for each flow parameters are explained detail in section 5.2.

4.2.2 Shape Parameterization

As mentioned in the introduction, the shape parameterization implemented during the optimization analysis should encompass sufficient design spaces in order to guarantee that the optimum design can be found. In this analysis, 3 different shape parameterizations are implemented.

4.2.2.1 Variation of Camber and Thickness

The idea used in the optimization is to change the camber and thickness of the airfoil. The camber and thickness variation are computed based on the initial camber and thickness distribution. The initial camber line is computed based on the average of the ordinate of the upper and lower airfoil nodes which are located on the same abscissa. The fact that the mesh nodes might not be located on the same abscissa, spline interpolation concept is applied.

The cubic spline interpolation [38] is used to perform the spline interpolation. A third order polynomial defined in Equation (4.3) is used as a model equation. This equation is valid for an interval [a,b] which contains n defined points. The coefficients b_i, c_i, d_i are defined in $(n-1)$ intervals. As a result, $3n-3$ equations are required to in order to solve these unknown coefficients. These coefficients are computed based on the required continuity and compatibility of the spline interpolation.

$$s_i(x) = y_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{4.3}$$

At each interior points in the interval $[a, b]$ should satisfy Equation (4.4). Furthermore, the interpolated functions should have continuous both first and second order derivative as shown in Equation (4.5) and Equation (4.6), respectively.

$$s_i(x_{i+1}) = y_{i+1} \tag{4.4}$$

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \tag{4.5}$$

$$s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}) \tag{4.6}$$

Natural boundary conditions are imposed on the end interval $[a, b]$ by specifying the second order derivative of boundary points to be zero.

$$\begin{aligned} s_i''(a) &= 0 \\ s_i''(b) &= 0 \end{aligned} \tag{4.7}$$

Upon having the same abscissa for nodes on both upper and lower airfoil, the camber line is estimated as:

$$\mathrm{y_{camber}} = \frac{y_{\mathrm{upper}} + \mathrm{y_{lower}}}{2} \tag{4.8}$$

The initial thickness distribution for the upper and lower is estimated as the difference between the initial camber line and airfoil surface coordinates. Equation (4.9) shows the estimation for the upper and thickness distribution.

$$y_{thick_{upper}} = y_{upper} - y_{camber}$$
$$y_{thick_{lower}} = y_{lower} - y_{camber} \quad (4.9)$$

The camber line variation is estimated by specifying several control points on the initial camber line. In the non-dimensional form, the abscissa location of control points are as follows: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9. The new camber line is estimated by multiplying the ordinate of the initial camber line with a factor specified by the user. For the points located in between the control points, similar spline interpolation explained earlier is applied.

On the other hand, the updated thickness variation is computed by multiplying the initial distribution shown in Equation (4.9) by some factors defined by the user. Both upper and lower thickness distribution is multiplied by the same factor. As a result, there are at most 2 different parameters used during the shape parameterization using camber and thickness variation. The range of these design variables are shown in Table 4.2.

Table 4.2 Boundary Imposed on Camber and Thickness Factors

Design Variables	Lower Limit	Upper Limit
Camber Factor	0	3
Thickness Factor	0.6	3

In the optimization analyses conducted in here, three different combinations of the above shape parameterizations are implemented: camber variation only, thickness variation only, and the combination of camber and thickness variation.

4.2.2.2 PARSEC Shape Parameterization

Detail explanation regarding parameters used in the PARSEC shape parameterization is depicted in Figure 2.1. Based on equations shown in Equation (2.1), it is required to compute the coefficients of a_i and b_i.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ X_{up}^{1/2} & X_{up}^{3/2} & X_{up}^{5/2} & X_{up}^{7/2} & X_{up}^{9/2} & X_{up}^{11/2} \\ \frac{1}{2}X_{up}^{-1/2} & \frac{3}{2}X_{up}^{1/2} & \frac{5}{2}X_{up}^{3/2} & \frac{7}{2}X_{up}^{5/2} & \frac{9}{2}X_{up}^{7/2} & \frac{11}{2}X_{up}^{9/2} \\ -\frac{1}{4}X_{up}^{-3/2} & \frac{3}{4}X_{up}^{-1/2} & \frac{15}{4}X_{up}^{1/2} & \frac{35}{4}X_{up}^{3/2} & \frac{63}{4}X_{up}^{5/2} & \frac{99}{4}X_{up}^{7/2} \\ 1 & 1 & 1 & 1 & 1 & 1 \\ \frac{1}{2} & \frac{3}{2} & \frac{5}{2} & \frac{7}{2} & \frac{9}{2} & \frac{11}{2} \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{Bmatrix} \tag{4.10}$$

$$= \begin{Bmatrix} \sqrt{2R_{le}} \\ Y_{up} \\ 0 \\ YXX_{up} \\ T_{off} + T_{\frac{TE}{2}} \\ \tan\left(\alpha_{TE} - \beta_{\frac{TE}{2}}\right) \end{Bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ X_{low}^{1/2} & X_{low}^{3/2} & X_{low}^{5/2} & X_{low}^{7/2} & X_{low}^{9/2} & X_{low}^{11/2} \\ \frac{1}{2}X_{low}^{-1/2} & \frac{3}{2}X_{low}^{1/2} & \frac{5}{2}X_{low}^{3/2} & \frac{7}{2}X_{low}^{5/2} & \frac{9}{2}X_{low}^{7/2} & \frac{11}{2}X_{low}^{9/2} \\ -\frac{1}{4}X_{low}^{-3/2} & \frac{3}{4}X_{low}^{-1/2} & \frac{15}{4}X_{low}^{1/2} & \frac{35}{4}X_{low}^{3/2} & \frac{63}{4}X_{low}^{5/2} & \frac{99}{4}X_{low}^{7/2} \\ 1 & 1 & 1 & 1 & 1 & 1 \\ \frac{1}{2} & \frac{3}{2} & \frac{5}{2} & \frac{7}{2} & \frac{9}{2} & \frac{11}{2} \end{bmatrix} \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{Bmatrix} \tag{4.11}$$

$$= \begin{Bmatrix} -\sqrt{2R_{le}} \\ Y_{low} \\ 0 \\ YXX_{low} \\ T_{off} - T_{\frac{TE}{2}} \\ \tan\left(\alpha_{TE} + \beta_{\frac{TE}{2}}\right) \end{Bmatrix}$$

These coefficients are computed based on the airfoil geometry. Both equations (4.10) and (4.11) show the system of linear equation which govern the coefficients of a_i and b_i, respectively. These equations are later solved by using Gauss-Seidel Iteration. Detail of Gauss-Seidel method implemented in this study is shown in the Appendix B.2.

It is verified that the PARSEC design variables are very sensitive. As a result, specific range of design variables need to be determined before initializing the optimization scheme. The range of these parameters are determined based on the optimization results found by considering the effect of camber and thickness. Detail of the parameter range used in this optimization is explained in detail in Chapter 5.

CHAPTER 5

RESULTS AND DISCUSSIONS

This chapter contains the result of mesh deformation by using the aforementioned technique defined in earlier chapter. The best scheme among these methods is then applied in the airfoil optimization.

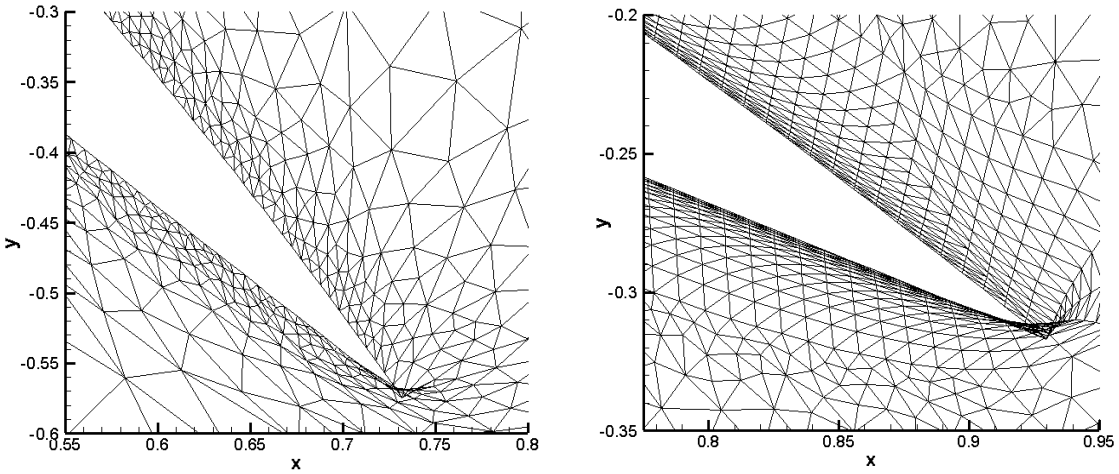
5.1 Mesh Deformation Results

The mesh deformation capability of the proposed methods is checked by using a simple test case. The test case used in here is to perform a rotating airfoil about quarter chord line by some degrees. Both inviscid and viscous meshes are considered in the verification case. For the inviscid mesh, the airfoil is rotated up to 50°. On the other hand, smaller rotation angle around 25° is introduced in the viscous mesh. The viscous mesh cannot be rotated by the same amount like in the inviscid mesh due to the presence of highly aspect ratio cell around the airfoil boundary. These cells somehow become a hindrance for spring analogy technique to perform the deformation scheme. Fortunately, the design spaces used in the airfoil optimization are encircled in this spring analogy technique.

5.1.1 Basic Spring Analogy Results

In the basic spring analogy results, no other improvements are considered during the application. The deformed meshes for both cases are shown in Figure 5.1. It can be seen clearly that this method fails to deform the mesh required in both cases. It is verified that inviscid mesh cannot be deformed with high degree of deformation. Some nodes near the trailing edge region (where huge displacement occurs) are

crossing over the opposite edges. In the viscous mesh case, the traditional spring analogy technique not only fails to prevent the cross-over nodes near the trailing edge regions, but also fails to maintain the right angle that cells around airfoil boundary have. This is caused by the fact that in the basic formulation, edge angle is not taking into account.

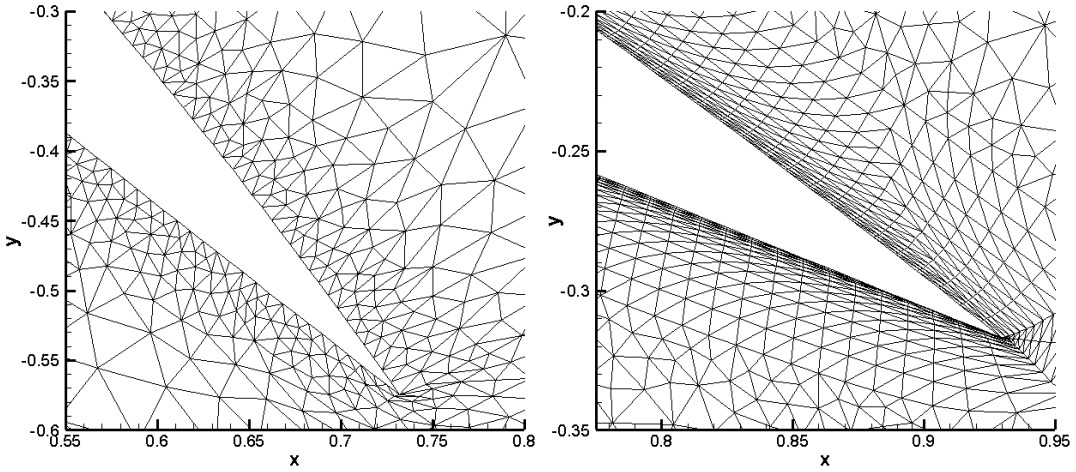


(a) 50° rotated inviscid mesh airfoil (b) 25° rotated viscous mesh airfoil

Figure 5.1 Deformed Meshes Resulted from Basic Spring Analogy

5.1.2 Angle Inclusion in Spring Analogy Results

In this case, the presence of angle in the spring is considered. This improvement somehow helps to generate the deformed mesh as it can be seen from Figure 5.2. The angle consideration helps the spring analogy to get a better deformed mesh on which there is no cross-over nodes occurring in trailing edge region and right angle near the surface boundary can still be maintained. The above computation is conducted with direct computation method. Similar deformed meshes are also achieved by using the indirect computation method. However, this computation required a lot of computation time compared to the direct method proposed earlier. Summary of the convergence analysis for the proposed spring analogy methods are shown in Figure 5.8.

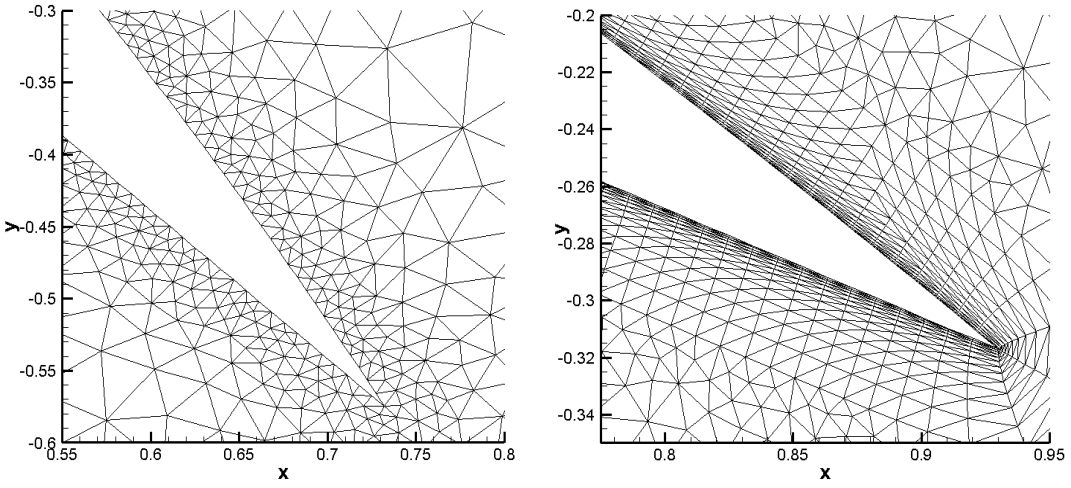


(a) 50° rotated inviscid mesh airfoil (b) 25° rotated viscous mesh airfoil

Figure 5.2 Deformed Meshes Resulted from Basic Spring Analogy with Angle Inclusion

5.1.3 Torsional Spring Analogy Results

The resulting mesh from this method can be seen from Figure 5.3. It can be seen clearly that concept of torsional spring analogy leads to a better quality in terms of no cross-over nodes and maintaining viscous angle near the surface boundary for deformed mesh. However, this method can only be solved using the indirect method which requires more computing time for a sequential execution. Summary of the convergence analysis for the proposed spring analogy methods are shown in Figure 5.8.

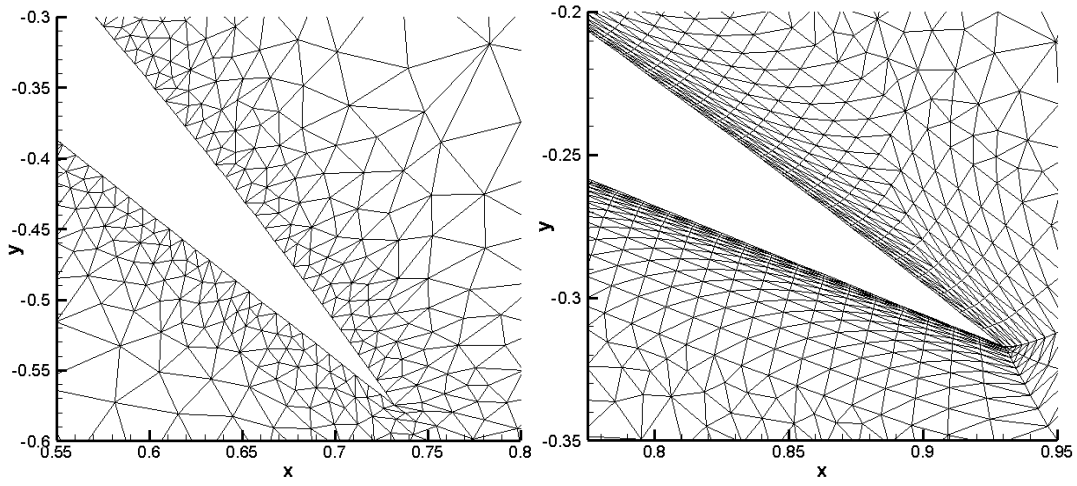


(a) 50° rotated inviscid mesh airfoil (b) 25° rotated viscous mesh airfoil

Figure 5.3 Deformed Meshes Resulted from Torsional Spring Analogy

5.1.4 Semi Torsional Spring Analogy Results

The deformed mesh from semi-torsional spring analogy is shown in Figure 5.4. In the computation, the angle formulation in edge is considered. It can be seen clearly that the results from this deformation scheme do not have any cross-over nodes and still maintain the angle of computation.

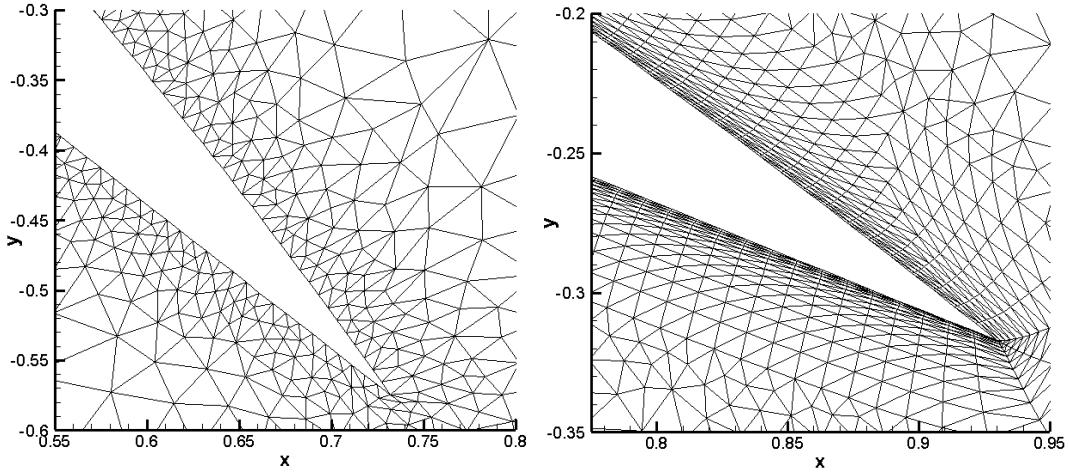


(a) 50° rotated inviscid mesh airfoil (b) 25° rotated viscous mesh airfoil

Figure 5.4 Deformed Meshes Resulted from Semi Torsional Spring Analogy

5.1.5 Ball-Center Spring Analogy

The ball-center spring analogy also yields to a quite similar results shown in the earlier schemes of mesh deformation techniques. This method yields to a better deformed mesh compared to the basic spring analogy method.

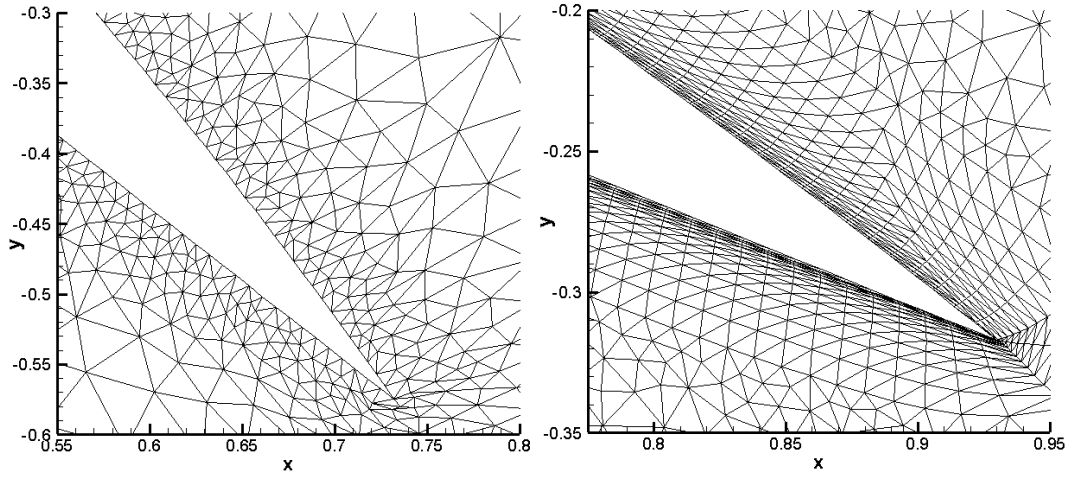


(a) 50° rotated inviscid mesh airfoil (b) 25° rotated viscous mesh airfoil

Figure 5.5 Deformed Meshes Resulted from Ball-Center Spring Analogy

5.1.6 Boundary Improvement

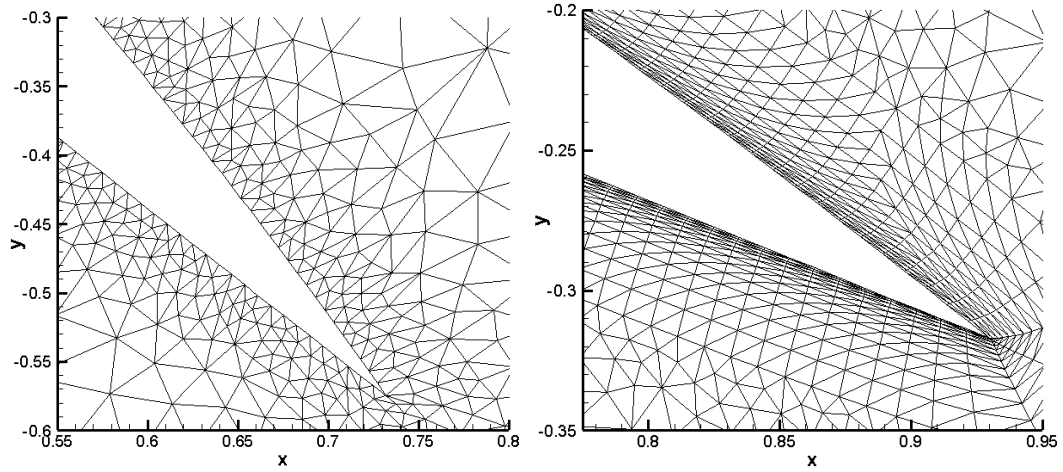
As elaborated in the earlier chapter, boundary improvement can be achieved by applying Saint-Venant principle during the implementation. In our case, this improvement is applied to the angle inclusion spring analogy method. Figure 5.2 shows the basic angle inclusion in spring analogy without any boundary improvements utilized. Two different concepts of Saint-Venant principle is applied in here: adjacent boundary improvement and surrounding region boundary improvement. The results corresponding to these two improvement are shown in Figure 5.6 and Figure 5.7.



(a) 50° rotated inviscid mesh airfoil

(b) 25° rotated viscous mesh airfoil

Figure 5.6 Deformed Meshes Resulted from Angle Inclusion Spring Analogy with Adjacent Boundary Improvement



(a) 50° rotated inviscid mesh airfoil

(b) 25° rotated viscous mesh airfoil

Figure 5.7 Deformed Meshes Resulted from Angle Inclusion Spring Analogy with Surrounding Region Boundary Improvement

It can be seen clearly that the deformed mesh by surrounding region boundary improvement leads to a better mesh in terms of the angle made by the cells around the trailing edge of the airfoil. The angle of cells around the trailing edge is higher in adjacent boundary improvement compared to surrounding region boundary improvement.

The proposed method is not only compared in terms of the deformed mesh results, but also in terms of the computation costs by means of number of iteration and computation time. In each method, the required number of iteration is assumed in such a way that the same residue value is achieved. The residual value is computed based on the nodal displacement for each node and shown in Equation (5.1).

$$Res = \sqrt{\frac{\sum_{i=1}^{\#\text{ of nodes}} \left(\frac{\Delta x_i}{c}\right) + \left(\frac{\Delta y_i}{c}\right)}{\#\text{ of nodes}}} \tag{5.1}$$

$$c = \text{chord length of the airfoil}$$

The residual plot is computed based on the logarithm with base 10 of the ratio of current residue value with the first initial residue value. Unlike CFD computation where a low residue value ($\log_{10}$ Res = -7) is required, the mesh deformation method can have the logarithmic of residual value around -3. However, for the deformation case in viscous meshes, the minimum tolerance value for the residue is -3.5.

The residual plot corresponding to inviscid mesh deformation for each proposed method is shown in Figure 5.8. It can be seen clearly among these methods, basic spring analogy requires less computational time compared to the other methods since no edge angle is considered in the computation. For a better illustration in regarding the direct solution utilized in this study, a residual plot up to 500 iteration is shown in Figure 5.9.

On the other hand, the torsional spring requires a quite huge number of iteration since it corresponding to solve a huge matrix by iterative manner. In the viscous mesh deformation case, similar plots shown in Figure 5.8 is achieved as well. However, the required computation time is difference since viscous mesh contains more number of nodes and elements compared to the inviscid mesh.

Table 5.1 Summary of Computation Time for Proposed Mesh Deformation Schemes

	Inviscid Mesh	Viscous Mesh
Basic Spring Analogy	5.256 Seconds	19.572 Seconds
Angle Inclusion	7.211 Seconds	34.940 Seconds
Semi-Torsional	7.431 Seconds	36.290 Seconds
Ball-Center	14.052 Seconds	59.292 Seconds
Torsional Spring	693.85 Seconds	1435.234 Seconds

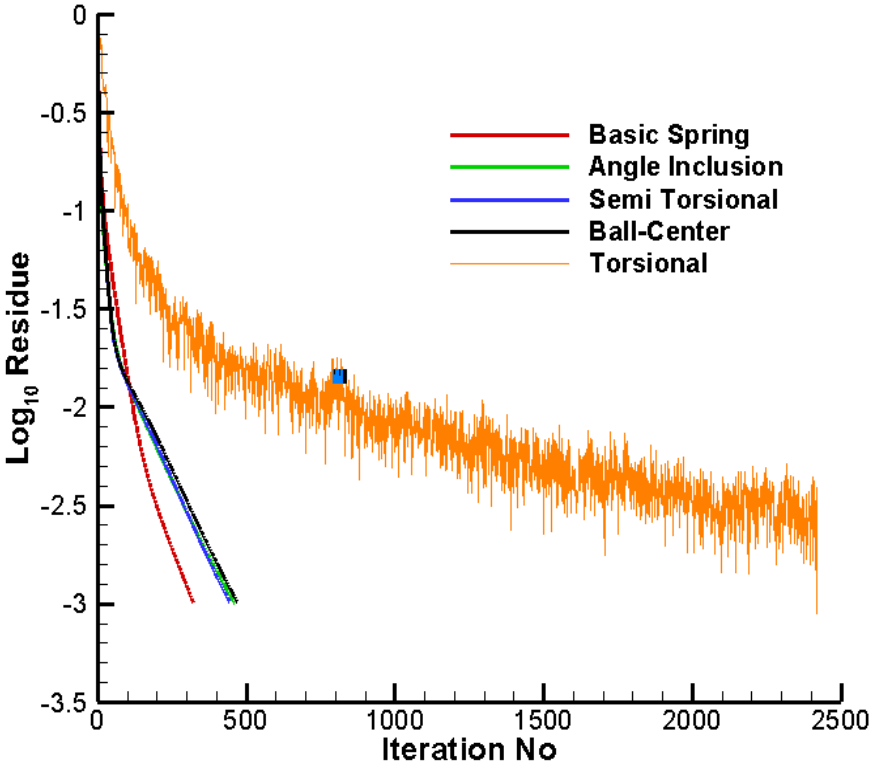


Figure 5.8 Residual Computation for Each Proposed Method in Spring Analogy Mesh Deformation Methods

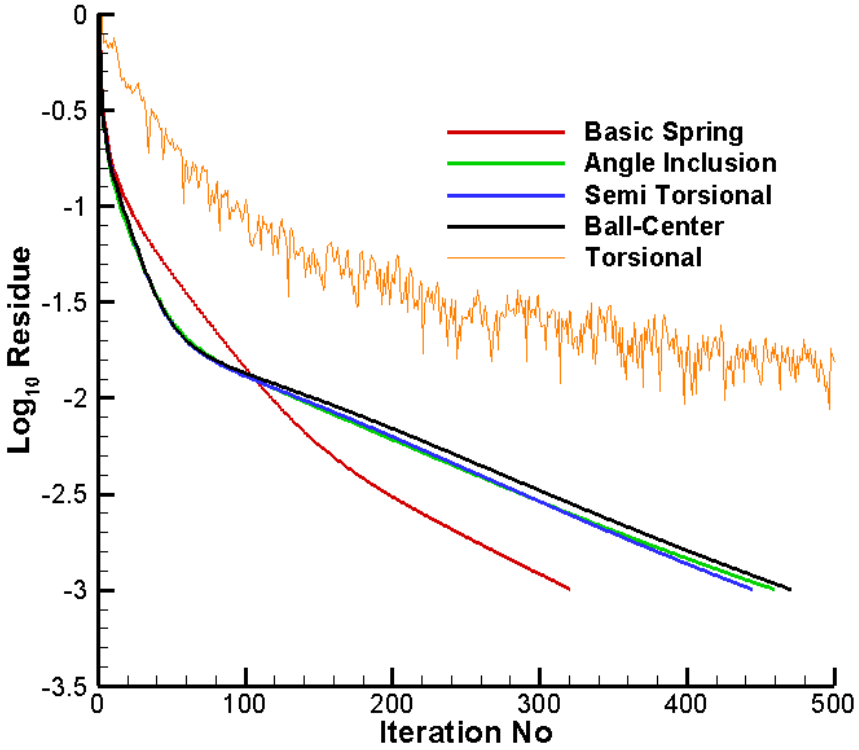


Figure 5.9 Residual Computation for Each Proposed Method in Spring Analogy Mesh Deformation Methods Up to 500 Iterations

From these analyses, it is concluded that direct computations has a better efficiency compared to indirect computation schemes. Furthermore, angle inclusion with surrounding boundary improvement, ball-center spring analogy, or semi-torsional spring analogy give almost similar results.

5.2 Optimization Results

The optimization is conducted based on the design variables defined in the previous chapter. Furthermore, the optimization is conducted in several different parameterization: camber only, thickness only, camber and thickness, and PARSEC shape optimization.

For camber only, thickness only, and camber and thickness optimization, a similar initial airfoil is used. On the other hand, the initial geometry used in the PARSEC optimization is defined based on the initial PARSEC parameters. As a result, slightly different initial drag values are achieved in the optimization.

5.2.1 Take-Off Configuration

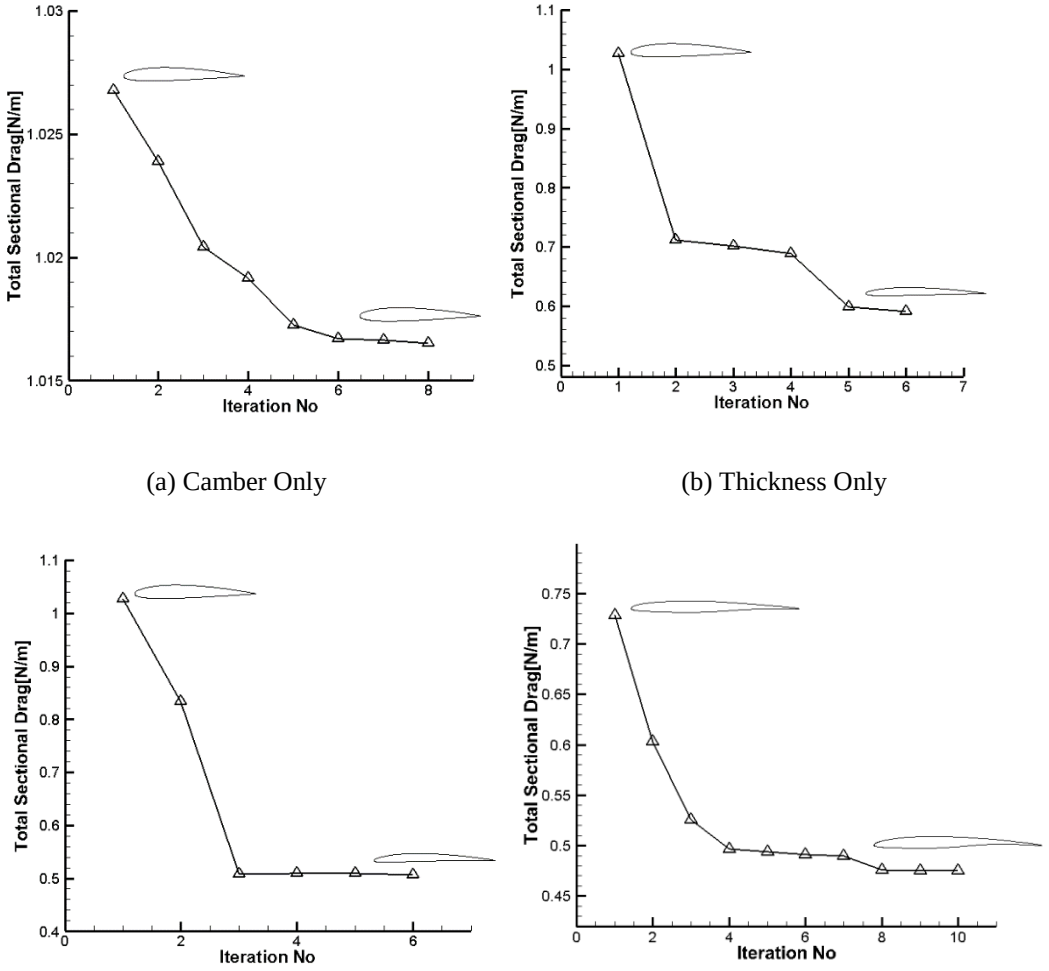
Based on the required sectional lift and Equation (4.2), the required lift coefficient for this configuration is 0.3725. In this optimization scheme, the angle of attack is constrained to be between -3° to 6°. The iteration history for several shape parameterizations in this parameter is shown in Figure 5.10.

In this take-off phase optimization, the optimum airfoil has a reduction in camber when only camber effect is considered. This is expected since the target lift coefficient is quite low and initial airfoil has quite relatively high camber and thickness. It is found that the optimum airfoil has a reduction in camber by a factor of 0.8333.

In the case where thickness is solely considered into account, the optimum airfoil has a reduction in thickness. The reduction in thickness helps the airfoil to keep the low drag coefficient. It is found that the reduction in thickness is by a factor of 0.6, which is the minimum allowable value.

For the case where both camber and thickness parameters are considered, instead of having reduction in camber as the case where the camber parameter is solely considered, the airfoil has an increased in camber by a factor of 1.452. The thickness is also reduced by a factor of 0.6 in this shape parameterization optimization. In the

case for PARSEC optimization, it is found that the optimum airfoil design leads to an optimum drag value. The range of PARSEC design variables and values for the optimum design are tabulated in Table 5.3.



(a) Camber Only

(b) Thickness Only

(c) Camber and Thickness

(d) PARSEC

Figure 5.10 Iteration History for Take-Off Optimization

Summary of the optimization results for this flight condition is shown in Table 5.2. It can be seen clearly that the optimization by considering both camber and thickness leads to a better optimum airfoil. The optimum airfoil shapes and their pressure distributions are shown in Figure 5.11 and Figure 5.12, respectively.

Table 5.2 Optimization Results for Take-Off Phase

Parameterization Scheme	Total Drag [N/m]		Angle of Attack [deg]	
	Initial	Optimum	Initial	Optimum
Camber	1.0267	1.0165	1.834	2.137
Thickness	1.0267	0.5907	1.834	1.768
Camber and Thickness	1.0267	0.5064	1.834	0.898
PARSEC	0.7284	0.475	1.982	1.123

Table 5.3 PARSEC Design Variables Range in the Take-Off Optimization

Parsec Airfoil Parameter	Lower Limit	Upper Limit	Initial Value	Optimum Value
Leading Edge Radius Upper $\left(R_{le_{upper}}\right)$ [m]	0.008	0.015	0.01	0.008
Leading Edge Radius Lower $\left(R_{le_{lower}}\right)$ [m]	0.001	0.005	0.003	0.00287
Peak Location for Lower Surface (X_{lo}) [m]	0.28	0.35	0.33	0.28
Peak Value for Lower Surface (Y_{lo}) [1/m]	-0.03	-0.02	-0.024	-0.02
Curvature for Lower Surface (YXX_{lo}) [1/m]	0.25	0.45	0.35	0.37037
Peak Location for Upper Surface $\left(X_{up}\right)$ [m]	0.28	0.35	0.33	0.35
Peak Value for Upper Surface $\left(Y_{up}\right)$ [m]	0.048	0.075	0.07	0.05709
Curvature for Upper Surface $\left(YXX_{up}\right)$[1/m]	-0.75	-0.4	-0.5	-0.4069
Trailing Edge Direction Angle (α_{TE})[deg]	-8	0	-3	-3.992
Trailing Edge Wedge Angle (β_{TE}) [deg]	12	20	12	12

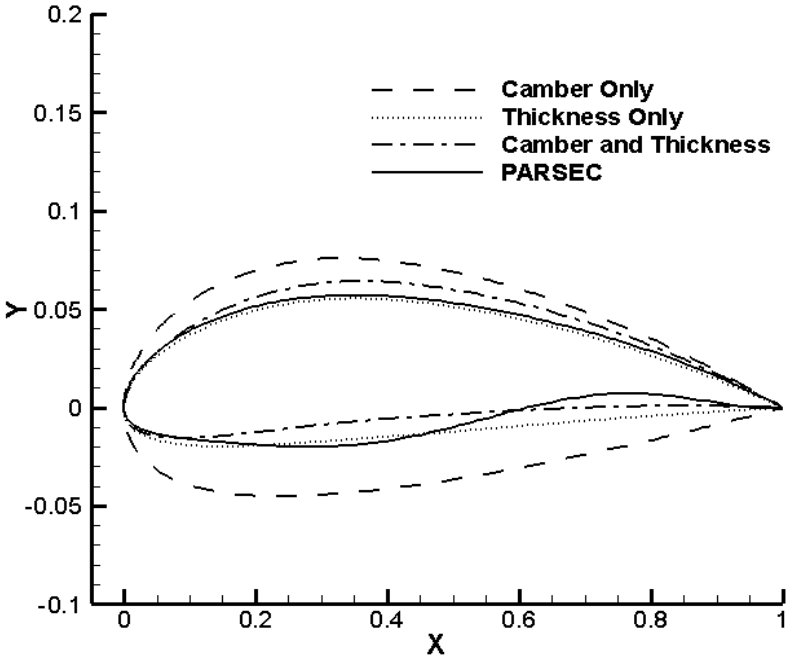


Figure 5.11 Optimum Airfoil Shapes for Take-Off Configuration

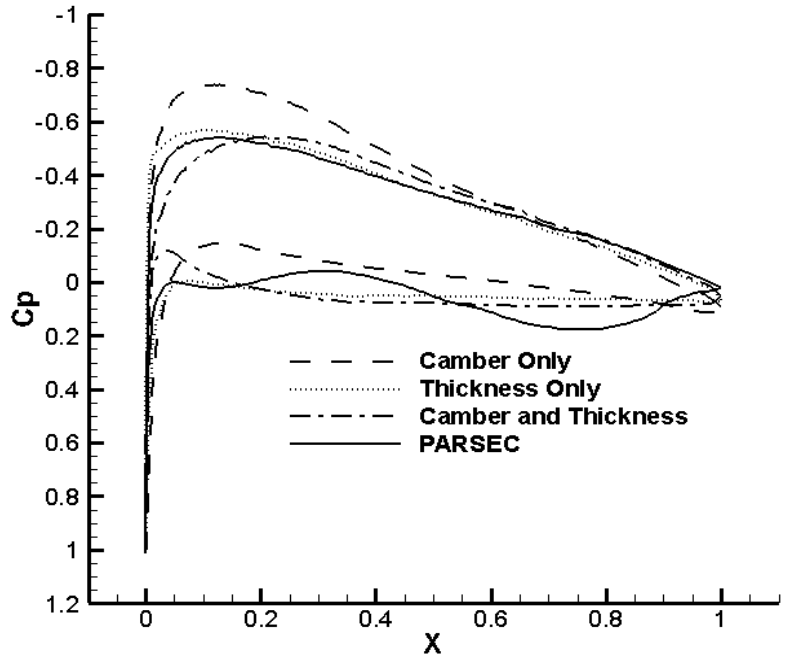
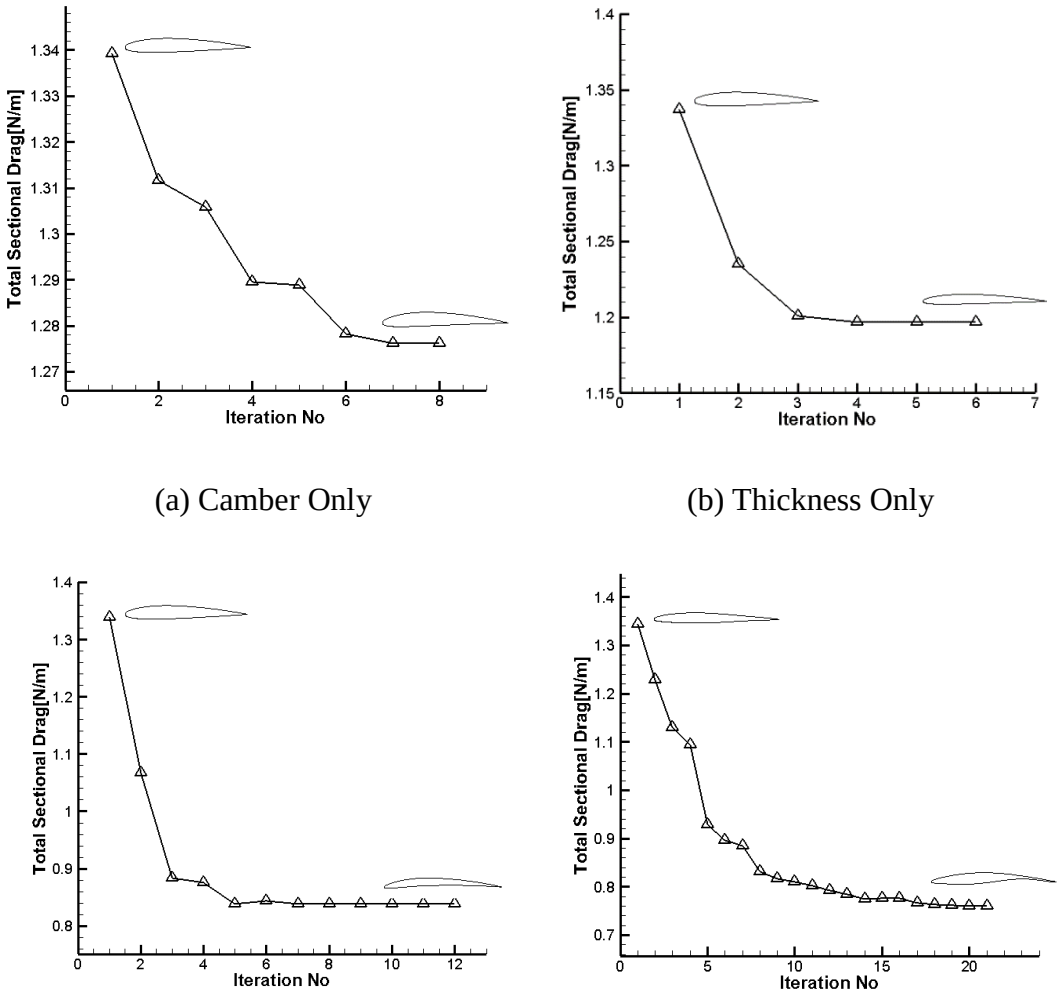


Figure 5.12 Pressure Distribution of Optimum Airfoil Shapes for Take-Off Configuration

5.2.2 Loiter Configuration

The required lift coefficient for this flight condition is computed as 0.7361. In this flight condition the angle of attack is constrained to be between -3° and 6°. The iteration history for 4 different shape parameterizations are shown in Figure 5.13.



(a) Camber Only (b) Thickness Only

(c) Camber and Thickness (d) PARSEC

Figure 5.13 Iteration History for Loiter Optimization

The fact that required lift coefficient for this flight condition is comparatively higher than take-off configuration, the optimization where only camber is considered tries to increase the camber of the airfoil by a factor of 1.747.

In the case where only thickness variation is merely considered, similar trend observed in take-off optimization also occurs in here. It is found that the optimum airfoil has a relatively decrease in thickness by a factor of 0.714.

Under the case where both camber and thickness is considered, it is observed that the optimum airfoil has an increase in camber by a factor of 2.485 and decrease in thickness by a factor of 0.6.

Summary of the optimization results for loiter optimization is shown in Table 5.4. In coherence with the solution from take-off optimization, it is perceived that the optimization by considering PARSEC shape parameterization leads to a better optimum value. PARSEC design variables used in the loiter optimization are tabulated in Table 5.5.

Table 5.4 Optimization Results for Loiter Phase

Parameterization Scheme	Total Drag [N/m]		Angle of Attack [deg]	
	Initial	Optimum	Initial	Optimum
Camber	1.3391	1.2761	5.417	4.030
Thickness	1.3391	1.1967	5.417	5.347
Camber and Thickness	1.3391	0.8376	5.417	2.395
PARSEC	1.3439	0.7591	5.701	2.941

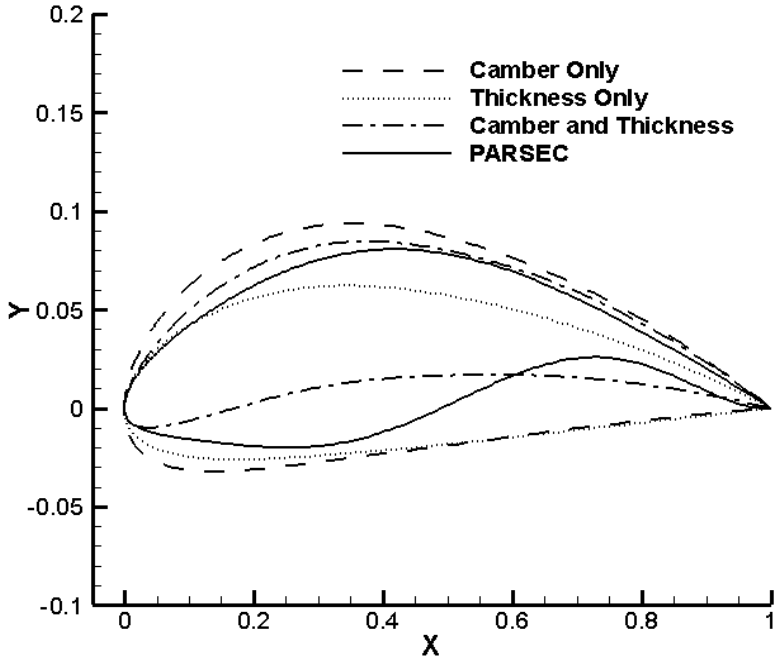


Figure 5.14 Optimum Airfoil Shapes for Loiter Configuration

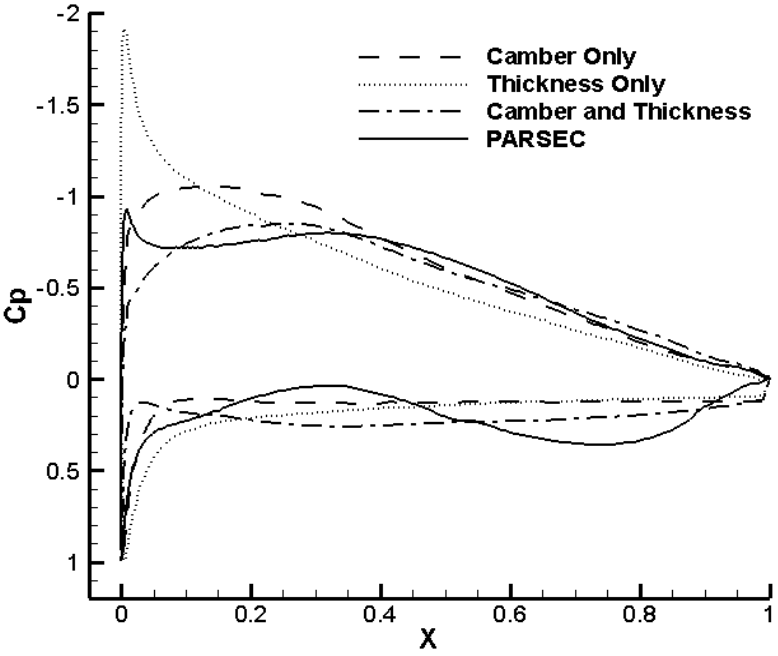


Figure 5.15 Pressure Distribution of Optimum Airfoil Shapes for Loiter Configuration

Table 5.5 PARSEC Design Variables Range in the Loiter Optimization

Parsec Airfoil Parameter	Lower Limit	Upper Limit	Initial Value	Optimum Value
Leading Edge Radius Upper $\left(R_{le_{upper}}\right)$ [m]	0.006	0.02	0.01	0.0069
Leading Edge Radius Lower $\left(R_{le_{lower}}\right)$ [m]	0.001	0.01	0.003	0.0027
Peak Location for Lower Surface (X_{lo}) [m]	0.25	0.38	0.33	0.25
Peak Value for Lower Surface (Y_{lo}) [m]	-0.05	-0.02	-0.03	-0.02
Curvature for Lower Surface (YXX_{lo}) [1/m]	0.25	0.6	0.35	0.5924
Peak Location for Upper Surface $\left(X_{up}\right)$ [m]	0.27	0.43	0.33	0.4178
Peak Value for Upper Surface $\left(Y_{up}\right)$ [m]	0.048	0.088	0.06	0.0807
Curvature for Upper Surface $\left(YXX_{up}\right)$ [1/m]	-0.85	-0.35	-0.6	-0.75
Trailing Edge Direction Angle (α_{TE}) [deg]	-10	0	-4	-5.165
Trailing Edge Wedge Angle (β_{TE}) [deg]	12	20	15	13.86

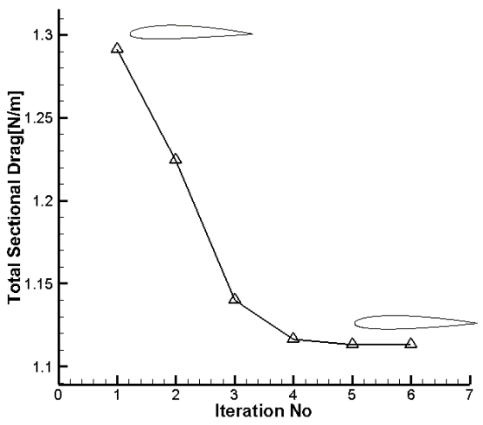
5.2.3 High-Speed Configuration

For this flight condition, it is calculated that the required lift coefficient is 0.18401. The angle of attack constraint is made to be between -3° and 3°. The angle of attack is relatively kept smaller since the required lift coefficient is also smaller. The iteration history for this configuration is shown in Figure 5.16.

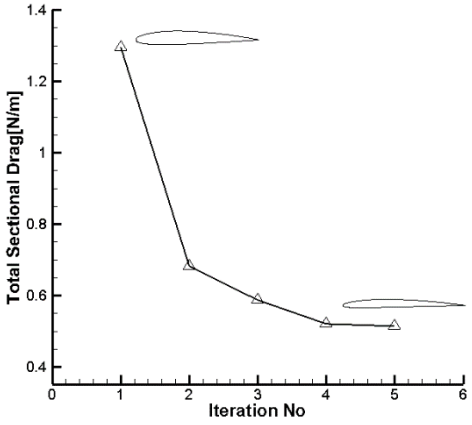
In the case where merely camber shape parameterization is considered, the optimum airfoil has a decrease in camber factor. The decreasing in camber is made in such a way that the optimum airfoil still produces the required lift coefficient and satisfies the angle of attack constraint. It is found that the optimum airfoil has a reduced in the camber by a factor of 0.4421.

For the thickness optimization, it is found that the thickness should be decreased in order to reduce the drag. In fact, the optimum airfoil has decreased in thickness by a factor of 0.6, which is the lower limit for the thickness parameter.

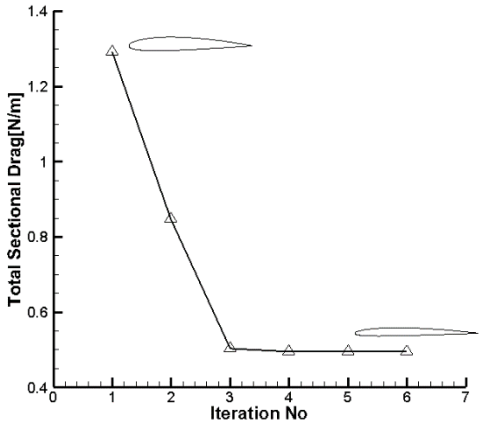
In the case where both camber and thickness is considered, the airfoil still has similar trends. The optimum airfoil has a reduced in both camber and thickness. The camber is reduced by a factor of 0.7059 and the thickness is reduced by a factor of 0.6.



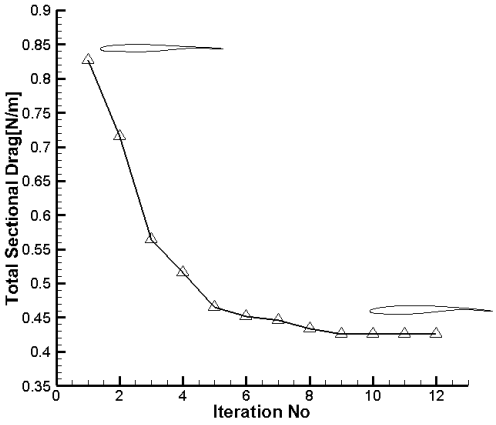
(a) Camber Only



(b) Thickness Only



(c) Camber and Thickness



(d) PARSEC

Figure 5.16 Iteration History for High-Speed Optimization

The summary concerning about the drag and angle of attack used in the high speed optimization is shown in Table 5.6. It is observed that the optimization conducted with PARSEC shape parameterization again leads to a better optimum results compared to other optimization cases. The optimum airfoil shapes and the

distribution of the pressure coefficient are shown in Figure 5.17 and Figure 5.18, respectively.

Table 5.6 Optimization Results for High-Speed Phase

Parameterization Scheme	Total Drag [N/m]		Angle of Attack [deg]	
	Initial	Optimum	Initial	Optimum
Camber	1.2955	1.1130	0.0126	1.033
Thickness	1.2955	0.5136	0.0126	-0.0252
Camber and Thickness	1.2955	0.4940	0.0126	0.5239
PARSEC	0.8268	0.4262	0.317	-0.465

Table 5.7 PARSEC Design Variables Range in the High-Speed Optimization

Parsec Airfoil Parameter	Lower Limit	Upper Limit	Initial Value	Optimum Value
Leading Edge Radius Upper $\left(R_{le_{upper}}\right)$[m]	0.012	0.025	0.02	0.01583
Leading Edge Radius Upper $\left(R_{le_{lower}}\right)$ [m]	0.005	0.012	0.009	0.00938
Peak Location for Lower Surface (X_{lo}) [m]	0.23	0.33	0.27	0.24125
Peak Value for Lower Surface (Y_{lo}) [m]	-0.05	-0.035	-0.045	-0.035
Curvature for Lower Surface (YXX_{lo}) [1/m]	0.4	0.3	0.5	0.49828
Peak Location for Upper Surface $\left(X_{up}\right)$ [m]	0.28	0.38	0.33	0.28465
Peak Value for Upper Surface $\left(Y_{up}\right)$ [m]	0.07	0.06	0.07	0.06717
Curvature for Upper Surface $\left(YXX_{up}\right)$ [1/m]	-0.7	-0.6	-0.5	-0.59982
Trailing Edge Direction Angle (α_{TE}) [deg]	-5	0	-3	-3.5347
Trailing Edge Wedge Angle (β_{TE}) [deg]	10	20	12	11.7674

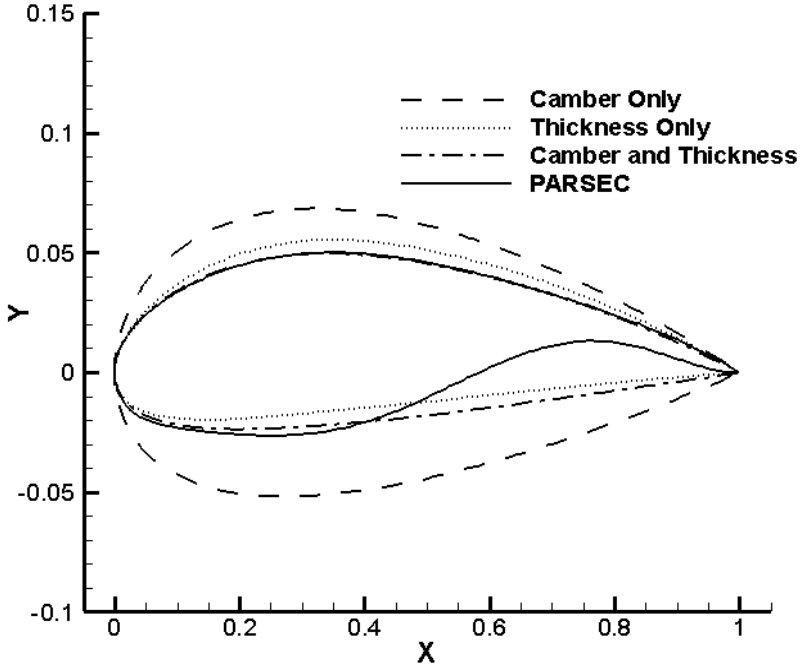


Figure 5.17 Optimum Airfoil Shapes for High Speed Configuration

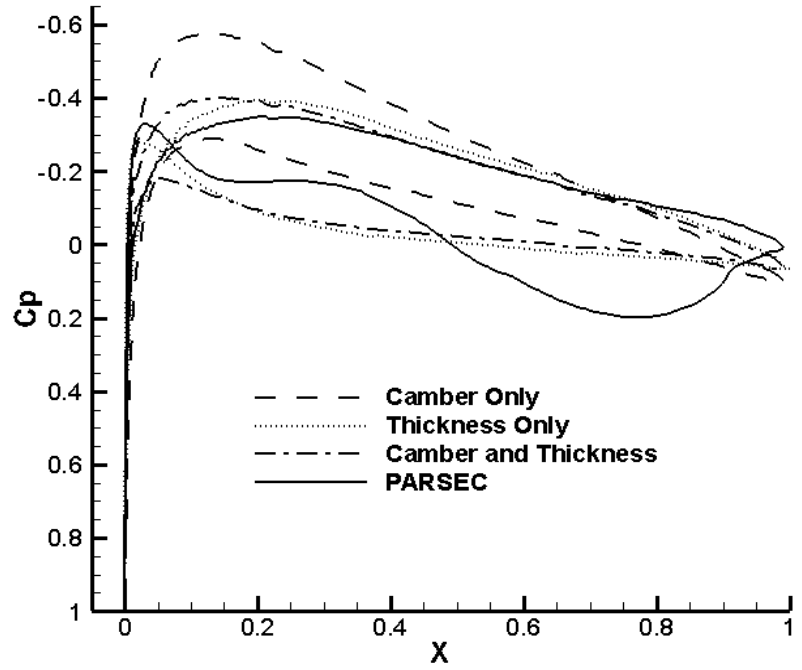
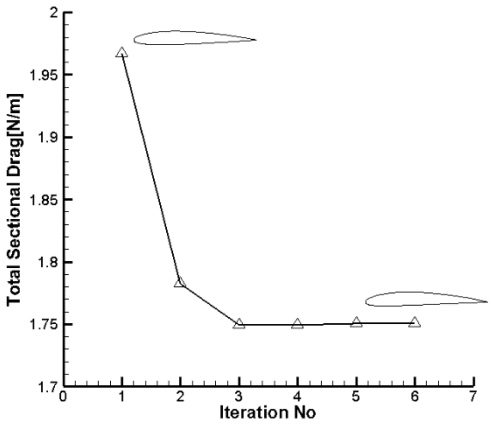


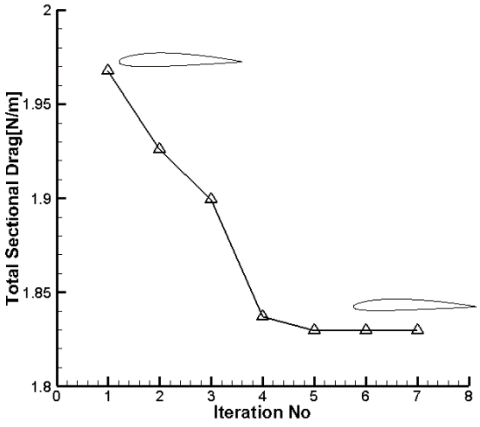
Figure 5.18 Pressure Distribution of Optimum Airfoil Shapes for High Speed Configuration

5.2.4 Landing Configuration

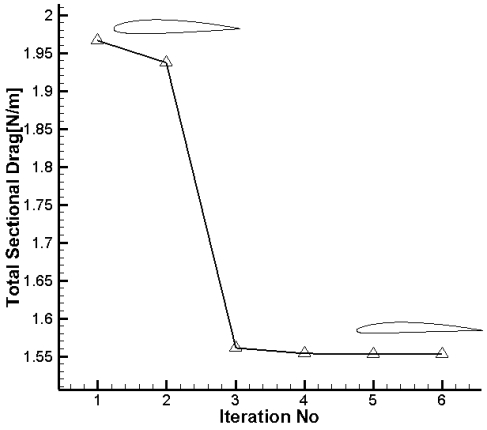
Based on the required sectional lift and Equation (4.2), the required lift coefficient for this flight condition is 0.98712. An additional angle of attack constraint is imposed on this configuration. The angle of attack for the constraint should be between 7 and 10 degrees. This high angle of attack is considered in order to ensure that sufficient drag can be achieved. The iteration history for the landing configuration is shown in Figure 5.19.



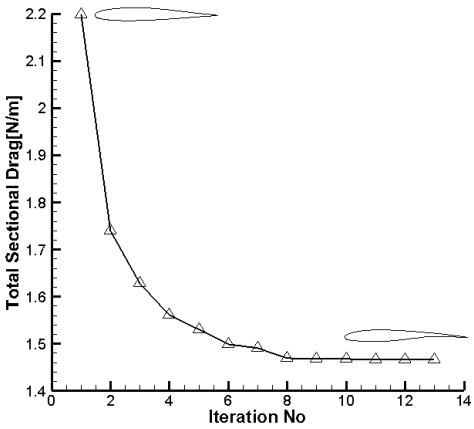
(a) Camber Only



(b) Thickness Only



(c) Camber and Thickness



(d) PARSEC

Figure 5.19 Iteration History for Landing Optimization

In the camber optimization, the optimization process tries to increase the camber factor of the airfoil to meet the high lift coefficient requirement. An increase in camber makes an airfoil to produce the required lift with less angle of attack, and hence less drag as well. However, one cannot increase the camber up to the maximum value since the airfoil is required to have a high angle of attack as well from the imposed constraint. The optimum airfoil has the camber increase by a factor of 1.574.

On the other hand, the thickness optimization tries to decrease the thickness distribution in order to decrease drag while still maintaining the required lift coefficient. It is found that the optimized airfoil has a decrease in thickness by a factor of 0.837.

In the optimization where both camber and thickness are considered, it is found that optimum airfoil has an increasing in camber and decreasing in thickness, similar to the trend observed in the earlier optimization. It is found that the optimum airfoil has camber increase by a factor of 1.434 and thickness decrease by a factor of 0.749.

The summary concerning about the drag and angle of attack used in the landing optimization is shown in Table 5.8. It is found that the PARSEC optimization leads to a better optimum value compared to the other shape parameterization schemes. PARSEC design variables used in this landing optimization is shown in Table 5.9. The optimum airfoil shapes and its pressure distribution are shown in Figure 5.20 and Figure 5.21, respectively.

Table 5.8 Optimization Summary for Landing Phase

Parameterization Scheme	Total Drag [N/m]		Angle of Attack [deg]	
	Initial	Optimum	Initial	Optimum
Camber	1.9677	1.7502	8.082	7.00
Thickness	1.9677	1.8294	8.082	7.95
Camber and Thickness	1.9677	1.5529	8.082	7.00
PARSEC	2.1981	1.4652	8.583	7.00

Table 5.9 PARSEC Design Variables Range in the Landing Optimization

Parsec Airfoil Parameter	Lower Limit	Upper Limit	Initial Value	Optimum Value
Leading Edge Radius Upper $\left(R_{le_{upper}}\right)$ [m]	0.012	0.025	0.02	0.01583
Leading Edge Radius Upper $\left(R_{le_{lower}}\right)$ [m]	0.005	0.012	0.009	0.00938
Peak Location for Lower Surface (X_{lo}) [m]	0.23	0.33	0.27	0.24125
Peak Value for Lower Surface (Y_{lo}) [m]	-0.05	-0.035	-0.045	-0.035
Curvature for Lower Surface (YXX_{lo}) [1/m]	0.4	0.3	0.5	0.49828
Peak Location for Upper Surface $\left(X_{up}\right)$ [m]	0.28	0.38	0.33	0.28465
Peak Value for Upper Surface $\left(Y_{up}\right)$ [m]	0.07	0.06	0.07	0.06717
Curvature for Upper Surface $\left(YXX_{up}\right)$ [1/m]	-0.7	-0.6	-0.5	-0.59982
Trailing Edge Direction Angle (α_{TE}) [deg]	-5	0	-3	-3.5347
Trailing Edge Wedge Angle (β_{TE}) [deg]	10	20	12	11.7674

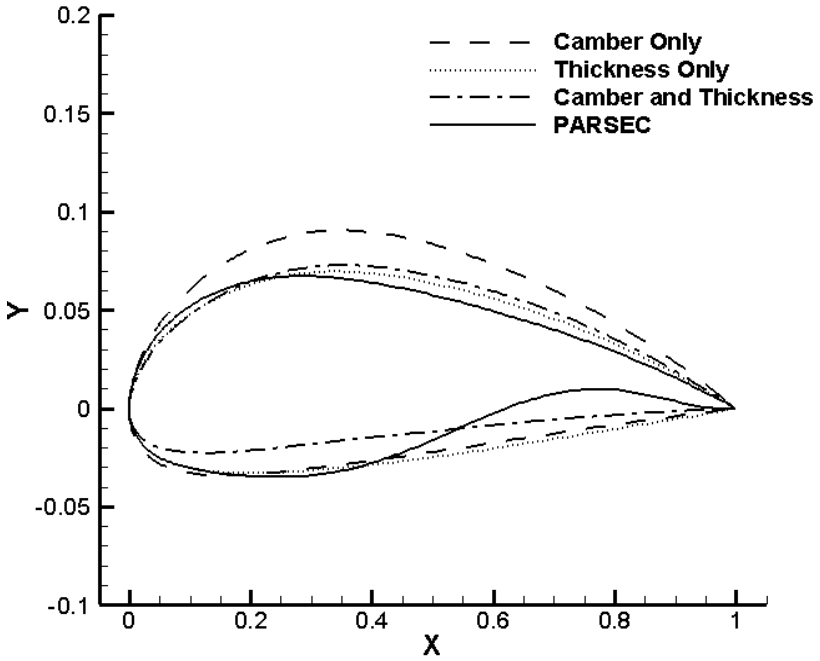


Figure 5.20 Optimum Airfoil Shapes for Different Parameterization in Landing Configuration

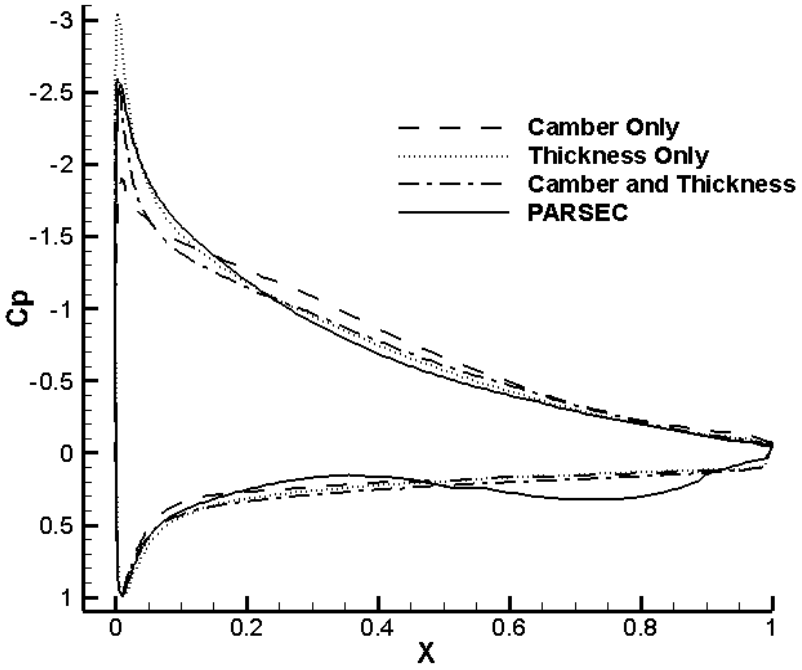


Figure 5.21 Cp Distribution for Optimum Airfoil in Landing Configuration with Several Shape Parameterizations

5.2.5 Miscellaneous Case

In order to guarantee that the optimization problem is a well-posed problem, a different initial geometry is chosen for one flight condition, loiter configuration. Instead of using NACA 2412 as the initial geometry, NACA 2415 is chosen as the initial geometry. This initial geometry is later used in the camber and thickness optimization.

In order to encompass similar geometry range, the range of thickness in NACA 2415 is changed as well. The new range used for NACA 2415 case is shown in Table 5.10.

Table 5.10 Range of Camber and Thickness Variables for NACA 2415 Case

Design Variables	Lower Limit	Upper Limit
Camber Factor	0	3
Thickness Factor	0.48	2.4

It is found that the optimum solution found in this case is very similar to the one found by using initial case to be NACA 2412. Summary of the parameters for the optimum design are tabulated in Table 5.11. The optimum airfoil shapes and its pressure distribution are shown in Figure 5.22 and Figure 5.23, respectively.

Table 5.11 Optimum Parameters for Two Different Cases in Loiter Configuration

	NACA 2412	NACA 2415
Optimum Camber Factor	0.6	2.484
Optimum Thickness Factor	0.48	2.453
Optimum Drag	0.8376	0.8537

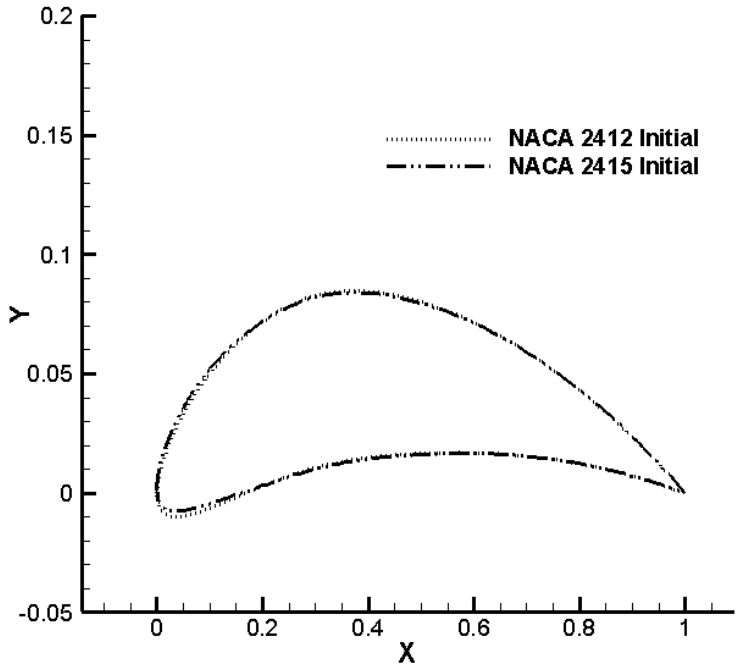


Figure 5.22 Optimum Airfoil Shapes for Loiter Optimization in Camber and Thickness Parameterization with Two Different Initial Airfoil Shapes

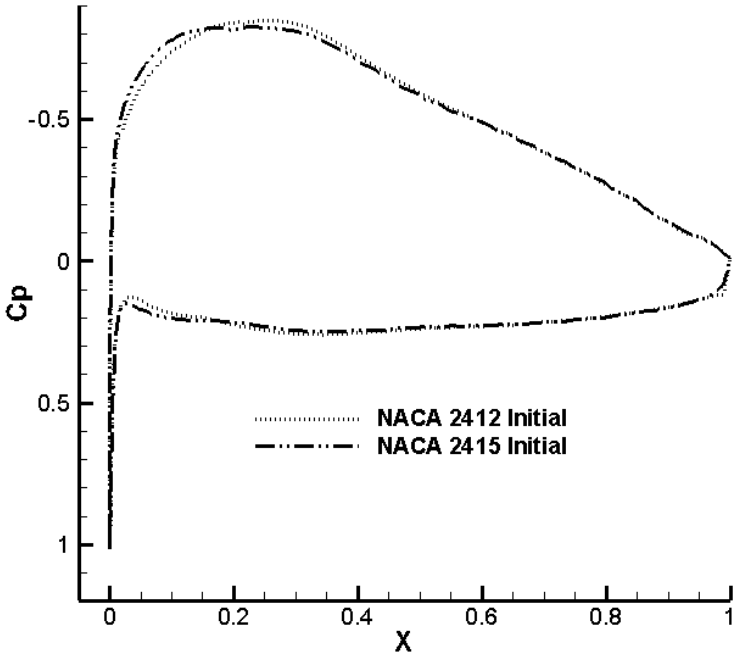


Figure 5.23 Pressure Distribution of Optimum Airfoil Shapes for Loiter Optimization in Camber and Thickness Parameterization with Two Different Initial Airfoil Shapes

Apart from the well-posed problem check for the optimization problem, one should also check the mesh convergence study for CFD computation. In order to perform the mesh convergence study, three different mesh sizes around NACA 2412 are considered. The difference between these meshes are based on the number of nodes and cells.

In order to check the convergence study, these meshes are utilized for CFD computation in loiter configuration. In each case, required lift coefficient is taken as 0.7358. Details of aerodynamic properties for these different cases are tabulated in Table 5.12. It can be seen clearly that the difference between Case 2 and Case 3 is not very much. As a result, the size of mesh similar to the one in Case 2 is considered in the CFD computation.

Table 5.12 Summary of Mesh Convergence Study for NACA 2412 Airfoil in Loiter Configuration

	Case 1	Case 2	Case 3
Number of Elements	6404	12060	24148
Number of Nodes	3300	6228	12397
c_l	0.7360	0.7354	0.7359
c_d	0.02	0.0163	0.0169
Required Angle of Attack [deg]	5.47	5.42	4.91

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis, a brief explanation about mesh deformation combined with CFD design optimization is elaborated. Several improvements in the spring analogy mesh deformations have been presented in the thesis. The improvements made in the spring analogy mesh deformation methods are as follows: angle consideration, semi-torsional, ball-center, and boundary improvement. For the case of angle considerations, two separate solution method have been proposed as well: direct and indirect methods. It is found that the proposed improvements in the spring analogy method remove the node crossing in the basic spring analogy method and maintain the cell angle of initial mesh. Furthermore, the indirect solution method proposed is more efficient in time compared to the direct solution method.

Based on the improved spring analogy method, an airfoil CFD design optimization is conducted. The optimization is conducted by aiming to reduce the sectional drag of an airfoil for several flight parameters. The conclusions made for each flight can be summarized as follows:

- For take-off configuration, a small thickness distribution accompanied with sufficient camber can lead to an optimum airfoil.
- For loiter configuration, on which velocity is low and no high angle of attack is required, a huge increase in camber accompanied by small thickness can lead to an optimum airfoil.

- For high-speed configuration, where the lowest lift coefficient is required, a small thickness accompanied by sufficient decrease in camber lead to an optimum airfoil.
- For landing configuration, which high angle of attack is required, a sufficient increase in camber accompanied by sufficient decrease in thickness yield to an optimum airfoil.

The summary for the above conclusion can be summarized in Table 6.1. The factors defined in here are the corresponding design variables used in the optimization analysis where both camber and thickness variation are considered.

Table 6.1 Summary of Camber and Thickness Factor Employed for Different Flight Parameters

Flight Parameter	Camber Factor	Thickness Factor
Take-Off	1.452	0.6
Loiter	2.485	0.6
Take-Off	0.7059	0.6
Landing	1.434	0.749

It is also found that another improvement in the optimization by utilizing PARSEC shape parameterization give a better optimum design compared to optimization by considering only camber and thickness distribution. Furthermore, the optimization problem is not very dependent to the starting point of camber and thickness distribution when both camber and thickness optimization are considered.

Furthermore, it is also found that in many case the optimization conducted reduce the required angle of attack value. This infact becomes another advantage since it is more beneficial to attain the same lift with low angle attack in a flight.

In summary, my contribution to this thesis are summarized as follows:

- Basic mesh connectivity principles applied to find the list of neighbor nodes surrounding an arbitrary node.

- Develop the idea of direct solution methods by considering 2 x 2 matrix for angle inclusion in spring analogy.
- Introducing the notion of ball-center spring analogy in the improvement method spring analogy.
- Applying the concept of boundary improvement via Saint-Venant principle in two different study cases.
- Applying the idea of global stiffness assemble process for indirect solution methods for torsional spring analogy approach.

6.2 Future Work

Mesh deformation technique can be enhanced by introducing parallel implementation in the computation. Apart from that, another improvement especially for the viscous mesh deformation scheme can be introduced in such a way that similar capability in inviscid analysis can be achieved. Edge connectivity might also be improved by using another advanced algorithm technique. Another challenging issue might be to perform mesh deformation technique for hybrid unstructured mesh instead of triangular unstructured mesh. The last meticulous work that can be considered is to implement this mesh deformation technique for 3-D mesh deformation scheme.

Regarding the optimization analysis, other improved shape parameterizations might be introduced in the analysis. Instead of using gradient based optimization, other optimization schemes like genetic algorithm or stochastic algorithm or even particle swarm optimization might be implemented as well. The optimization might be also applied in high Mach number (compressible flow) around the airfoil instead of low Mach number (incompressible flow). Last but not the least, another excellent work that can be considered is to combine the 3-D mesh deformation scheme with wing design optimization.

REFERENCES

[1] F. Palacios, T. D. Economon, A. C. Aranake, S. R. Copeland, A. K. Lonkar, T. W. Lukaczyk, D. E. Manosalvas, K. R. Naik, a S. Padr, B. Tracey, A. Variyar, and J. J. Alonso, "Stanford University Unstructured (SU2): Open-source Analysis and Design Technology for Turbulent Flows," in *52nd Aerospace Science Meeting*, 2014 *AIAA SciTech, 13-17 Jan. 2014, National Harbor, Maryland*, AIAA 2014-0243.

[2] M. J. Lighthill, *A New Method of Two-dimensional Aerodynamic Design*. Reports and Memoranda No 2112, 1945.

[3] B. Epstein and S. Peigin, "Accurate CFD driven optimization of lifting surfaces for wing-body configuration," *Comput. Fluids*, Vol. 36, pp. 1399–1414, 2007.

[4] A. Shahrokhi and A. Jahangirian, "Airfoil shape parameterization for optimum Navier-Stokes design with genetic algorithm," *Aerosp. Sci. Technol.*, Vol. 11, pp. 443–450, 2007.

[5] B. Epstein, S. Peigin, and S. Tsach, "A new efficient technology of aerodynamic design based on CFD driven optimization," *Aerosp. Sci. Technol.*, Vol. 10, pp. 100–110, 2006.

[6] A. Jahangirian and A. Shahrokhi, "Aerodynamic shape optimization using efficient evolutionary algorithms and unstructured CFD solver," *Comput. Fluids*, Vol. 46, No. 1, pp. 270–276, 2011.

[7] W. Song and A. J. Keane, "A study of shape parameterisation methods for airfoil optimisation," in *10th AIAA/ISSMO Multidiscplinary Analysis and Optimization Conference, 30 Aug. - 1 Sept. 2004, Albany, New York*, AIAA 2004-4482.

[8] H. Sobieczky, "Parametric Airfoils and Wings," *Notes Numer. Fluid Mech.*, Vol. 68, pp. 71–88, 1999.

[9] R. M. Hicks and P. A. Henne, "Wing design by numerical optimization," *J. Aircr.*, Vol. 15, No. 7, pp. 407–412, 1977.

[10] E. S. Tashnizi, A. A. Taheri, and M. H. Hekmat, "Investigation of the Adjoint Method in Aerodynamic Optimization Using Various Shape Parameterization Techniques," *J. Brazilian Soc. Mech. Sci. Eng.*, Vol. 32, No. 2, pp. 176–186, 2010.

[11] J. Chen, Q. Wang, X. Pang, S. Li, and X. Guo, “Improvement of airfoil design using smooth curvature technique,” *Renew. Energy*, Vol. 51, pp. 426–435, 2013.

[12] G. S. Dulikravich, “Aerodynamic Shape Optimization Methods,” in *New Design Concepts for High Speed Air Transport*, H. Sobieczky, Ed. Springer-Verlag, 1997, pp. 175–187.

[13] R. Mukesh, K. Lingadurai, and U. Selvakumar, “Airfoil shape optimization using non-traditional optimization technique and its validation,” *J. King Saud Univ. - Eng. Sci.*, Vol. 26, No. 2, pp. 191–197, 2014.

[14] M. de’ Michieli Vitturi and F. Beux, “A discrete gradient-based approach for aerodynamic shape optimisation in turbulent viscous flow,” *Finite Elem. Anal. Des.*, Vol. 43, pp. 68–80, 2006.

[15] J. E. V Peter and R. P. Dwight, “Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches,” *Comput. Fluids*, Vol. 39, No. 3, pp. 373–391, 2010.

[16] T. J. Lin, Z. Q. Guan, J. H. Chang, and S. H. Lo, “Vertex-Ball Spring Smoothing: An efficient method for unstructured dynamic hybrid meshes,” *Comput. Struct.*, Vol. 136, pp. 24–33, May 2014.

[17] J. T. Batina, “Using Unstructured Dynamic Meshes,” *AIAA J.*, Vol. 28, No. 8, pp.1381–1388, Aug. 1990.

[18] O. Estruch, O. Lehmkuhl, R. Borrell, C. D. P. Segarra, and a. Oliva, “A parallel radial basis function interpolation method for unstructured dynamic meshes,” *Comput. Fluids*, Vol. 80, No. 1, pp. 44–54, 2013.

[19] E. Luke, E. Collins, and E. Blades, “A fast mesh deformation method using explicit interpolation,” *J. Comput. Phys.*, Vol. 231, No. 2, pp. 586–601, Jan. 2012.

[20] X. Zhou and S. Li, “A new mesh deformation method based on disk relaxation algorithm with pre-displacement and post-smoothing,” *J. Comput. Phys.*, Vol. 235, pp. 199–215, Feb. 2013.

[21] A. Masud, M. Bhanabhagvanwala, and R. a. Khurram, “An adaptive mesh rezoning scheme for moving boundary flows and fluid-structure interaction,” *Comput. Fluids*, Vol. 36, No. 1, pp. 77–91, 2005.

[22] J. A. . Witteveen, “Explicit and Robust Inverse Distance Weighting Mesh Deformation for CFD,” in *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, 4-7 Jan. 2010, Orlando, Florida*, AIAA 2010-165.

[23] X. Liu, N. Qin, and H. Xia, “Fast dynamic grid deformation based on Delaunay graph mapping,” *J. Comput. Phys.*, Vol. 211, No. 2, pp. 405–423, 2006.

[24] A. de Boer, M. S. van der Schoot, and H. Bijl, “Mesh deformation based on radial basis function interpolation,” *Comput. Struct.*, Vol. 85, No. 11–14, pp. 784–795, 2007.

[25] F. J. Blom, “Considerations on the spring analogy,” *Int. J. Numer. Methods Fluids*, Vol. 32, No. 6, pp. 647–668, Mar. 2000.

[26] C. L. Bottasso, D. Detomi, and R. Serra, “The ball-vertex method : a new simple spring analogy method for unstructured dynamic meshes,” *Comput. Methods Appl. Mech. Eng.*, Vol. 194, pp. 4244–4264, 2005.

[27] C. Burg, “A Robust Unstructured Grid Movement Strategy Using Three-Dimensional Torsional Springs,” in *34th AIAA Fluid Dynamics Conference and Exhibit*, 28 *Jun. - 1 Jul. 2004, Portland, Oregon*, AIAA 2004-2529.

[28] R. D. Cook, D. S. Malkus, and M. Plesha, *Concepts and Applications of Finite Element Analysis*, 2nd ed. 1989.

[29] C. Farhat, “Torsional springs for two-dimensional dynamic unstructured fluid meshes,” *Comput. Methods Appl. Mech. Eng.*, Vol. 25, No. 98, pp. 231–245, 1998.

[30] C. Degand and C. Farhat, “A three-dimensional torsional spring analogy method for unstructured dynamic meshes,” *Comput. Struct.*, Vol. 80, No. 3–4, pp. 305–316, Feb. 2002.

[31] D. Zeng and C. R. Ethier, “A semi-torsional spring analogy model for updating unstructured meshes in 3D moving domains,” *Finite Elem. Anal. Des.*, Vol. 41, No. 11–12, pp. 1118–1139, Jun. 2005.

[32] Y. Yang and S. Ozgen, “Implementation of Ball-Center Spring Analogy Mesh Deformation Technique with CFD Design Optimization,” in *22nd AIAA CFD Conference, 2015 AIAA Aviation, 22-26 Jun. 2015, Dallas, Texas,* AIAA 2015-2612.

[33] G. A. Markou, Z. S. Mouroutis, and D. C. Charmpis, “The ortho-semi-torsional (OST) spring analogy method for 3D mesh moving boundary problems,” *Comput. Methods Appl. Mech. Eng.*, Vol. 196, pp. 747–765, 2007.

[34] Y. Saad, *Iterative Methods for Sparse Linear Systems*. International Thomson Publishing, 1996.

[35] “Pointwise.” [Online]. Available: http://www.pointwise.com/. [Accessed: 07-Jun-2015].

[36] “CHANGE Project.” [Online]. Available: http://change.tekever.com/. [Accessed: 07-Jun-2015].

[37] “Phoenix ModelCenter.” [Online]. Available: http://www.phoenix-int.com/modelcenter/integrate.php. [Accessed: 09-Jun-2015].

[38] A. Klein and A. Godunov, *Introductory Computational Physics*, 1st ed. Cambridge University Press, 2006.

APPENDIX A

DERIVATION OF KINEMATIC FORMULATION IN TORSIONAL SPRING ANALOGY METHOD

This section elaborates the kinematic formulation used in the torsional spring analogy by Farhat et al. [29]. The kinematic formulation is considered only for one of nodes in triangular cell Ω_{ijk}.

By assuming the angular displacement $\Delta\theta_{ik}$ (relative displacement of node k with respect to node i), shown in Figure 3.3 is small enough, the angular displacement can be computed as:

$$\Delta\theta_{ik} \cong \sin\Delta\theta_{ik} = \frac{\vec{r}_{ik} \times \vec{r}_{ik'}}{\|\vec{r}_{ik}\|\|\vec{r}_{ik'}\|} = \frac{\vec{r}_{ik} \times \vec{r}_{ik'}}{l_{ik} l_{ik'}}$$

$$\vec{r}_{ik} = \begin{Bmatrix} x_k - x_i \\ y_k - y_i \end{Bmatrix} \tag{A.1}$$

$$\vec{r}_{ik'} = \begin{Bmatrix} x_{k'} - x_i \\ y_{k'} - y_i \end{Bmatrix} = \begin{Bmatrix} x_k - x_i \\ y_k - y_i \end{Bmatrix} + \underbrace{\begin{Bmatrix} x_{k'} - x_k \\ y_{k'} - y_k \end{Bmatrix}}_{\vec{q}_k} = \vec{r}_{ik} + \begin{Bmatrix} u_k \\ v_k \end{Bmatrix}$$

Based on the above consideration, both length l_{ik} and $l_{ik'}$ are assumed to be equal to each other. As a result, the final expression for the angular displacement can be simplified as:

$$\Delta\theta_{ik} = \frac{(x_k - x_i)(y_k - y_i + v_k) - (y_k - y_i)(x_k - x_i + u_k)}{l_{ik}^2}$$

$$\Delta\theta_{ik} = \frac{(x_k - x_i)v_k - (y_k - y_i)u_k}{l_{ik}^2} \tag{A.2}$$

$$\Delta\theta_{ik} = \underbrace{\frac{(x_k - x_i)}{l_{ik}^2}}_{a_{ik}} v_k - \underbrace{\frac{(y_k - y_i)}{l_{ik}^2}}_{b_{ik}} u_k$$

$$\Delta\theta_{ik} = a_{ik}v_k - b_{ik}u_k$$

Another angular displacement attached to node i is coming from node j. Small rotation angle is also considered in the computation process. By doing similar procedure done previously, the terms for $\Delta\theta_{ij}$ is shown in Equation (A.3). Notice that there exists a sign different in the equation since positive angular displacement is defined as the increase in θ_i.

$$\Delta\theta_{ij} = -a_{ij}v_j + b_{ij}u_j \tag{A.3}$$

The last contribution for the angular increment $\Delta\theta_i$ comes from the node itself. Unfortunately, it is not plausible to derive this equation by similar procedures explained earlier. However, by inspection this angular increment is modeled as shown in Equation (A.4).

$$\Delta\theta_{ii} = \gamma v_i + \beta u_i \tag{A.4}$$

As a result, the total angular displacement for node i is shown in Equation (A.5).

$$\begin{aligned} \Delta\theta_i &= \Delta\theta_{ii} + \Delta\theta_{ij} + \Delta\theta_{ik} \\ \Delta\theta_i &= \gamma v_i + \beta u_i - a_{ij}v_j + b_{ij}u_j + a_{ik}v_k - b_{ik}u_k \end{aligned} \tag{A.5}$$

In order to complete the above expression, it is required to compute the coefficients of γ and β. These two coefficients are computed based on the rigid body motion condition for the triangular cell Ω_{ijk}. In the rigid body motion, each node in the triangular cell travels the same distance, that is $u_i = u_j = u_k = u$ and $v_i = v_j = v_k = v$. Another constraint that should be considered is in the rigid body motion, the total angular displacements should be equal to zero. Based on these assumptions, the coefficients of γ and β can be computed as shown in Equation (A.6).

$$\Delta\theta_i = (\gamma - a_{ij} + a_{ik})v + (\beta + b_{ij} - b_{ik})u$$
$$\Delta\theta_i = 0 \Rightarrow \begin{matrix} \gamma = a_{ij} - a_{ik} \\ \beta = -b_{ij} + b_{ik} \end{matrix} \quad \text{(A.6)}$$

Consequently, the final expression for the angle $\Delta\theta_i$ is shown in Equation (A.7).

$$\Delta\theta_i = (b_{ik} - b_{ij})u_i + (a_{ij} - a_{ik})v_i + b_{ij}u_j - a_{ij}v_j - b_{ik}u_k + a_{ik}v_k \quad \text{(A.7)}$$

By similar convention, the angle increment for $\Delta\theta_j$ and $\Delta\theta_k$ are computed as:

$$\begin{matrix} \Delta\theta_j = (b_{ji} - b_{jk})u_j + (a_{jk} - a_{ji})v_j + b_{jk}u_k - a_{jk}v_k - b_{ji}u_i + a_{ji}v_i \\ \Delta\theta_k = (b_{kj} - b_{ki})u_k + (a_{ki} - a_{kj})v_j + b_{ki}u_i - a_{ki}v_i - b_{kj}u_j + a_{kj}v_j \end{matrix} \quad \text{(A.8)}$$

By combining equations (A.7) and (A.8) together, the kinematic matrix which governs the relation between the angular displacement and the nodal displacement for each node is shown in Equation (A.9).

$$\theta^{ijk} = \begin{Bmatrix} \Delta\theta_i^{ijk} \\ \Delta\theta_j^{ijk} \\ \Delta\theta_k^{ijk} \end{Bmatrix} = \begin{bmatrix} b_{ik} - b_{ij} & a_{ij} - a_{ik} & b_{ij} & -a_{ij} & -b_{ik} & a_{ik} \\ -b_{ji} & a_{ji} & b_{ji} - b_{jk} & a_{jk} - a_{ji} & b_{jk} & -a_{jk} \\ b_{ki} & -a_{ki} & -b_{kj} & a_{kj} & b_{kj} - b_{ki} & a_{ki} - a_{kj} \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_k \\ v_k \end{Bmatrix} \quad \text{(A.9)}$$

APPENDIX B

ITERATIVE SOLVER

B.1 Conjugate Gradient Method

This conjugate gradient method is used to solver for the solution of a system of equation $Ax = b$. However, the matrix A should be a symmetric matrix. Algorithm B.1 explains about how this method is implemented [34].

Algorithm B.1 Conjugate Gradient Algorithm

Input: Symmetric matrix A and vector b
Output: solution vector of equation $Ax = b$

compute $r_0 = b - Ax_0$ and $p_0 = r_0$
for i = 0, 1, 2, … **do**
  $\alpha_i = (r_i, r_i)/(Ap_i, p_i)$
  $x_{i+1} = x_i + \alpha_i p_i$
  $r_{i+1} = r_i - \alpha_i Ap_i$
  $\beta_i = (r_{i+1}, r_{i+1})/(r_i, r_i)$
  $p_{i+1} = r_{i+1} + \beta_i p_i$
  if $|x_{i+1} - x_i| <$ tol **then** exit loop for **2:**
end for

B.2 Gauss-Seidel Iterative Solver

Given a system of equation $Ax = b$, the solution x is solved iteratively as:

$$x_i^k = \frac{1}{a_{ii}}\left[-\sum_{j=1}^{i-1}\left(a_{ij}x_j^k\right) - \sum_{j=i+1}^{n}\left(a_{ij}x_j^{k-1}\right) + b_i\right] \qquad \text{(B.1)}$$

The solution is said to be converged if the difference between the iterative solution is less than some designated tolerance defined by the user.

APPENDIX C

SAMPLE CASE OF GLOBAL STIFFNESS MATRIX ASSEMBLE

For a simple illustration, global stiffness matrix assemble for two different triangular cells shown in Figure C.1.

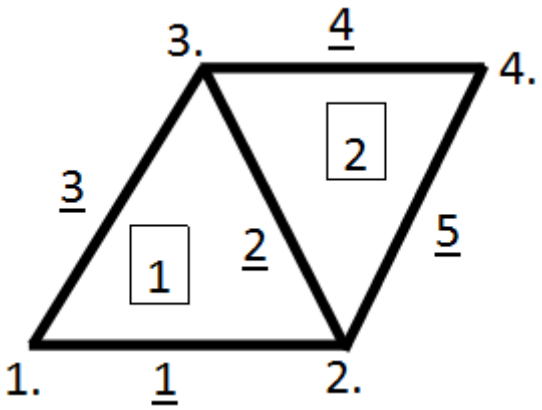


Figure C.1 Sample Case of Global Stiffness Matrix Assemble Process

The number with underline describes the edge number, number in a box describes the cell number, and the number with dot at the right describes the node number.

In this example both node 1 and node 2 are assumed to be prescribed with node 3, node 4, and node 5 are free to move. Based on this consideration, ID array matrix used in the computation can be written in Equation (C.1).

$$ID = \begin{matrix} \\ u \\ v \end{matrix}\begin{matrix} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \tag{C.1}$$

Based on the methods described earlier, the destination array required for computing the active stiffness matrix and prescribed stiffness matrix are shown below.

$$\text{Dest Array 1} = \begin{matrix} \\ u \\ v \end{matrix}\begin{matrix} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 3 & 5 \\ 0 & 0 & 2 & 4 & 6 \end{bmatrix} \end{matrix} \tag{C.2}$$

$$\text{Dest Array 2} = \begin{matrix} & 1 & 2 & 3 & 4 & 5 \\ u & 7 & 9 & 1 & 3 & 5 \\ v & 8 & 10 & 2 & 4 & 6 \end{matrix} \tag{C.3}$$

Global assemble procedure are conducted by the aids of these destination arrays. In the above example, there are 5 local stiffness matrices for edges and 2 local stiffness matrix for torsion.

$$\begin{bmatrix} k_{1_{11}} & k_{1_{12}} & k_{1_{13}} & k_{1_{14}} \\ k_{1_{21}} & k_{1_{22}} & k_{1_{23}} & k_{1_{24}} \\ k_{1_{31}} & k_{1_{32}} & k_{1_{33}} & k_{1_{34}} \\ k_{1_{41}} & k_{1_{42}} & k_{1_{43}} & k_{1_{44}} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \end{Bmatrix} \quad \begin{bmatrix} k_{2_{11}} & k_{2_{12}} & k_{2_{13}} & k_{2_{14}} \\ k_{2_{21}} & k_{2_{22}} & k_{2_{23}} & k_{2_{24}} \\ k_{2_{31}} & k_{2_{32}} & k_{2_{33}} & k_{2_{34}} \\ k_{2_{41}} & k_{2_{42}} & k_{2_{43}} & k_{2_{44}} \end{bmatrix} \begin{Bmatrix} u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix}$$

$$\begin{bmatrix} k_{3_{11}} & k_{3_{12}} & k_{3_{13}} & k_{3_{14}} \\ k_{3_{21}} & k_{3_{22}} & k_{3_{23}} & k_{3_{24}} \\ k_{3_{31}} & k_{3_{32}} & k_{3_{33}} & k_{3_{34}} \\ k_{3_{41}} & k_{3_{42}} & k_{3_{43}} & k_{3_{44}} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_3 \\ v_3 \end{Bmatrix} \quad \begin{bmatrix} k_{4_{11}} & k_{4_{12}} & k_{4_{13}} & k_{4_{14}} \\ k_{4_{21}} & k_{4_{22}} & k_{4_{23}} & k_{4_{24}} \\ k_{4_{31}} & k_{4_{32}} & k_{4_{33}} & k_{4_{34}} \\ k_{4_{41}} & k_{4_{42}} & k_{4_{43}} & k_{4_{44}} \end{bmatrix} \begin{Bmatrix} u_3 \\ v_3 \\ u_4 \\ v_4 \end{Bmatrix} \tag{C.4}$$

$$\begin{bmatrix} k_{5_{11}} & k_{5_{12}} & k_{5_{13}} & k_{5_{14}} \\ k_{5_{21}} & k_{5_{22}} & k_{5_{23}} & k_{5_{24}} \\ k_{5_{31}} & k_{5_{32}} & k_{5_{33}} & k_{5_{34}} \\ k_{5_{41}} & k_{5_{42}} & k_{5_{43}} & k_{5_{44}} \end{bmatrix} \begin{Bmatrix} u_2 \\ v_2 \\ u_4 \\ v_4 \end{Bmatrix}$$

The above local stiffness matrix are combined together in other to perform the solution for the angle consideration in spring analogy. Based on the problem, the size of the active global stiffness matrix will be 6 x 6 and prescribed global stiffness matrix will be 4 x 6. These partitioned matrices are achieved based on Algorithm 1 shown in Chapter 3.

For the case of torsional spring analogy, assemble of the matrices shown in Equation (C.5) is required. In fact, the final stiffness matrix is superposition of the contribution from both equations (C.4) and (C.5).

$$\begin{bmatrix} c_{1_{11}} & c_{1_{12}} & c_{1_{13}} & c_{1_{14}} & c_{1_{15}} & c_{1_{16}} \\ c_{1_{21}} & c_{1_{22}} & c_{1_{23}} & c_{1_{24}} & c_{1_{25}} & c_{1_{26}} \\ c_{1_{31}} & c_{1_{32}} & c_{1_{33}} & c_{1_{34}} & c_{1_{35}} & c_{1_{36}} \\ c_{1_{41}} & c_{1_{42}} & c_{1_{43}} & c_{1_{44}} & c_{1_{45}} & c_{1_{46}} \\ c_{1_{51}} & c_{1_{52}} & c_{1_{53}} & c_{1_{54}} & c_{1_{55}} & c_{1_{56}} \\ c_{1_{61}} & c_{1_{62}} & c_{1_{63}} & c_{1_{64}} & c_{1_{65}} & c_{1_{66}} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix}$$
$$\begin{bmatrix} c_{2_{11}} & c_{2_{12}} & c_{2_{13}} & c_{2_{14}} & c_{2_{15}} & c_{2_{16}} \\ c_{2_{21}} & c_{2_{22}} & c_{2_{23}} & c_{2_{24}} & c_{2_{25}} & c_{2_{26}} \\ c_{2_{31}} & c_{2_{32}} & c_{2_{33}} & c_{2_{34}} & c_{2_{35}} & c_{2_{36}} \\ c_{2_{41}} & c_{2_{42}} & c_{2_{43}} & c_{2_{44}} & c_{2_{45}} & c_{2_{46}} \\ c_{2_{51}} & c_{2_{52}} & c_{2_{53}} & c_{2_{54}} & c_{2_{55}} & c_{2_{56}} \\ c_{2_{61}} & c_{2_{62}} & c_{2_{63}} & c_{2_{64}} & c_{2_{65}} & c_{2_{66}} \end{bmatrix} \begin{Bmatrix} u_2 \\ v_2 \\ u_4 \\ v_4 \\ u_3 \\ v_3 \end{Bmatrix} \quad \text{(C.5)}$$

Similar to the approach implemented in the solution for spring analogy method with angle consideration, the torsional spring analogy also requires partitioning the global stiffness matrix based on whether the nodes are active degree of freedom or prescribed boundary condition. Similar technique elaborated in Algorithm 1 in Chapter 3 is also implemented in here.

APPENDIX D

INPUT FILES

D.1 Mesh Deformation Input File

In order to run the mesh deformation code developed in this study, a .dat input file is prepared. This input file describes the information required for choosing the method and explaining about the design variables used in the optimization.

```
#Deformation Parameter Description Accompanying the Mesh Deformation
Program

#Type of Deformation (1 = ROTATION, 2 = CAMBER, 3 = THICKNESS, 4 =
CAMBER and THICKNESS, 5 = HICKS-HENNE 6=PARSEC, 7= MOVEMENT )
NUM_DEFORM = 1                %number of deformation applied in the
program
DEFORM_TYPE = 1            %description of each type applied in the
program

#Desribe the Input for the ROTATION
ROT_CENTER = 0.25  0.0
ROT_ANGLE  = 50.0             %in degree
ROT_COND = 1                %Desribe whether the rotated is applied
to whole airfoil or not (0 = NO, 1 = YES)

#Desribe the position of design variable in camber
NUM_DES_CAM = 9
DES_LOC  = 0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9
FACT_CAM = 1 %increment factor in the camber distribution

#Desribe the factor used for the thickness distribution in upper and
lower
THICK_FACTOR = 0.6  %both upper and lower of the airfoil

#Desciribe the Hicks Henne Location
NUM_HICKS_HENNE_UPPER  = 11
WIDTH_UPPER = 5
HICKS_LOC_UPPER  = 0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,0.95
HICKS_FACT_UPPER =
0.0051,0.005,0.005,0.005,0.005,0.005,0.005,0.005,0.005,0.005,0.005

NUM_HICKS_HENNE_LOWER  = 11
```

```
WIDTH_LOWER = 5
HICKS_LOC_LOWER  = 0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,0.95
HICKS_FACT_LOWER =
0.005,0.005,0.005,0.005,0.005,0.005,0.005,0.005,0.005,0.005,0.005

#PARSEC AIRFOIL Shape Parameterization for Leading Edge Radius
rad_le_upper = 0.02        %the radius of the leading edge lower
rad_le_lower = 0.009      %the radius of the leading edge lower

#PARSEC AIRFOIL Shape Parameterization for maximum crest on lower
airfoil
x_low_max = 0.27
y_low_max = -0.045
yxx_low_max = 0.4

#PARSEC AIRFOIL Shape Parameterization for maximum crest on upper
airfoil
x_up_max = 0.33
y_up_max = 0.07
yxx_up_max = -0.6

#PARSEC AIRFOIL Shape Parameterization for defining the
te_off = 0
te_height = 0
alpha_te = -3               %in degree
beta_te = 12                %in degree

#Displacement Type of Deformation for the Airfoil
X_DIST = 2.0
Y_DIST = 2.0

#Method for Deforming the Mesh (1 = LINEAR, 2 = SEMITORSIONAL, 3 =
BALLCENTER, 4 = TORSIONAL)
DEF_MET = 1

#SOLUTION METHOD (1 = DIRECT,2 = INDIRECT)
SOL_METHOD = 1

#ITERATIVE METHOD FOR DIRECT SOLUTION (1=NORMAL, 2=SOR METHOD)
ITER_METHOD_DIR = 1

#ITERATIVE METHOD FOR INDIRECT SOLUTION (1= GAUSS, 2= JACOBI, 3 =
SOR)
ITER_METHOD_IND = 1

#ANGLE CONSIDERATION IN INDIRECT SOLUTION (0 = NO, 1= YES)
ANGLE_IND = 1

#IMPROVEMENT FOR THE BOUNDARY CONDITION SAINT VENANT PRINCIPLE(0 =
N0, 1 = YES)
BOUND_STAT = 1

#REGION WHERE SAINT VENANT PRINCIPLE SHOULD BE APPLIED (0= NONE, 1=
AIRFOIL SURFACE, 2 = SOME REGIONS)
```

```
BOUND_REG = 2

#Relaxation parameter used in the iteration procedure
OMEGA = 0.8

#Determine the output file name for the deformed mesh (for both .tec
and .su2 )
OUTPUT_MESH = output_mesh
```

D.2 SU2 Input File

In order to execute SU2 CFD solver, a configuration file (.cfg) is required. The configuration used in this study is shown below.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%                                                                       %
% SU2 configuration file                                                %
%        Case        description:        Incompressible        RANS
%
% File Version 3.2.9 "eagle"                                            %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%

% ------------- DIRECT, ADJOINT, AND LINEARIZED PROBLEM DEFINITION
------------%
%
% Physical governing equations (EULER, NAVIER_STOKES,
%                               TNE2_EULER, TNE2_NAVIER_STOKES,
%                                    WAVE_EQUATION, HEAT_EQUATION,
LINEAR_ELASTICITY,
%                               POISSON_EQUATION)
PHYSICAL_PROBLEM= NAVIER_STOKES
%
% Specify turbulent model (NONE, SA, SA_NEG, SST)
KIND_TURB_MODEL= SA
%
% Mathematical problem (DIRECT, ADJOINT, LINEARIZED)
MATH_PROBLEM= DIRECT
%
% Regime type (COMPRESSIBLE, INCOMPRESSIBLE, FREESURFACE)
REGIME_TYPE= INCOMPRESSIBLE
%
% Restart solution (NO, YES)
RESTART_SOL= NO
```

```
% ------------------------ UNSTEADY SIMULATION -------------------
------------%
%
%  Unsteady  simulation  (NO,  TIME_STEPPING,  DUAL_TIME_STEPPING-
1ST_ORDER,
%                       DUAL_TIME_STEPPING-2ND_ORDER, TIME_SPECTRAL)
UNSTEADY_SIMULATION= NO

% -------------------- INCOMPRESSIBLE FREE-STREAM DEFINITION ------
------------%
%
% Free-stream density (1.2886 Kg/m^3 (air), 998.2 Kg/m^3 (water))
FREESTREAM_DENSITY= 1.18955
%
% Free-stream velocity (m/s)
FREESTREAM_VELOCITY= ( 13.2117383361196, 0.923853959188664, 0.0 )
%
%  Free-stream  viscosity  (1.853E-5  Ns/m^2  (air),  0.798E-3  Ns/m^2
(water))
FREESTREAM_VISCOSITY= 1.853E-5
%
% ---------------------- REFERENCE VALUE DEFINITION ---------------
------------%
%
% Reference origin for moment computation
REF_ORIGIN_MOMENT_X = 0.25
REF_ORIGIN_MOMENT_Y = 0.00
REF_ORIGIN_MOMENT_Z = 0.00
%
% Reference length for pitching, rolling, and yawing non-dimensional
moment
REF_LENGTH_MOMENT= 1.0
%
%  Reference  area  for  force  coefficients  (0  implies  automatic
calculation)
REF_AREA= 1.0

% -------------------- BOUNDARY CONDITION DEFINITION --------------
------------%
%
% Navier-Stokes wall boundary marker(s) (NONE = no marker)
MARKER_HEATFLUX= ( airfoil, 0.0 )
%
% Farfield boundary marker(s) (NONE = no marker)
MARKER_FAR= ( farfield )
%
% Marker(s) of the surface to be plotted or designed
MARKER_PLOTTING= ( airfoil )
%
% Marker(s) of the surface where the functional (Cd, Cl, etc.) will
be evaluated
MARKER_MONITORING= ( airfoil )
```

```
% ------------- COMMON PARAMETERS DEFINING THE NUMERICAL METHOD ---
------------%
%
%   Numerical   method   for   spatial   gradients   (GREEN_GAUSS,
WEIGHTED_LEAST_SQUARES)
NUM_METHOD_GRAD= WEIGHTED_LEAST_SQUARES
%
% Courant-Friedrichs-Lewy condition of the finest grid
CFL_NUMBER= 10.0
%
% Adaptive CFL number (NO, YES)
CFL_ADAPT= NO
%
% Parameters of the adaptive CFL number (factor down, factor up, CFL
min value,
%                                          CFL max value )
CFL_ADAPT_PARAM= ( 1.5, 0.5, 1.0, 100.0 )
%
% Number of total iterations
EXT_ITER= 2000

% ----------------------- SLOPE LIMITER DEFINITION ----------------
------------%
%
% Reference element length for computing the slope and sharp edges
limiters.
REF_ELEM_LENGTH= 0.1
%
% Coefficient for the limiter
LIMITER_COEFF= 0.1
%
% Coefficient for the sharp edges limiter
SHARP_EDGES_COEFF= 3.0
%
% Reference coefficient (sensitivity) for detecting sharp edges.
REF_SHARP_EDGES= 3.0
%
% Remove sharp edges from the sensitivity evaluation (NO, YES)
SENS_REMOVE_SHARP= NO

% ------------------------ LINEAR SOLVER DEFINITION ---------------
------------%
%
% Linear solver for implicit formulations (BCGSTAB, FGMRES)
LINEAR_SOLVER= FGMRES
%
%  Preconditioner  of  the  Krylov  linear  solver  (JACOBI,  LINELET,
LU_SGS)
LINEAR_SOLVER_PREC= LU_SGS
%
% Minimum error of the linear solver for implicit formulations
LINEAR_SOLVER_ERROR= 1E-4
%
```

```
% Max number of iterations of the linear solver for the implicit
formulation
LINEAR_SOLVER_ITER= 5

% -------------------------- MULTIGRID PARAMETERS -----------------
------------%
%
% Multi-Grid Levels (0 = no multi-grid)
MGLEVEL= 0
%
% Multi-grid cycle (V_CYCLE, W_CYCLE, FULLMG_CYCLE)
MGCYCLE= V_CYCLE
%
% Multi-grid pre-smoothing level
MG_PRE_SMOOTH= ( 1, 2, 3, 3 )
%
% Multi-grid post-smoothing level
MG_POST_SMOOTH= ( 0, 0, 0, 0 )
%
% Jacobi implicit smoothing of the correction
MG_CORRECTION_SMOOTH= ( 0, 0, 0, 0 )
%
% Damping factor for the residual restriction
MG_DAMP_RESTRICTION= 0.75
%
% Damping factor for the correction prolongation
MG_DAMP_PROLONGATION= 0.75

% -------------------- FLOW NUMERICAL METHOD DEFINITION -----------
------------%
%
% Convective numerical method (JST, LAX-FRIEDRICH, CUSP, ROE, AUSM,
HLLC,
%                              TURKEL_PREC, MSW)
CONV_NUM_METHOD_FLOW= ROE
%
%  Spatial  numerical  order  integration  (1ST_ORDER,  2ND_ORDER,
2ND_ORDER_LIMITER)
%
SPATIAL_ORDER_FLOW= 2ND_ORDER_LIMITER
%
% Slope limiter (VENKATAKRISHNAN, MINMOD)
SLOPE_LIMITER_FLOW= VENKATAKRISHNAN
%
% 1st, 2nd and 4th order artificial dissipation coefficients
AD_COEFF_FLOW= ( 0.15, 0.5, 0.02 )
%
%   Time   discretization   (RUNGE-KUTTA_EXPLICIT,   EULER_IMPLICIT,
EULER_EXPLICIT)
TIME_DISCRE_FLOW= EULER_IMPLICIT

% -------------------- TURBULENT NUMERICAL METHOD DEFINITION ------
------------%
%
```

```
% Convective numerical method (SCALAR_UPWIND)
CONV_NUM_METHOD_TURB= SCALAR_UPWIND
%
%  Spatial  numerical  order  integration  (1ST_ORDER,  2ND_ORDER,
2ND_ORDER_LIMITER)
%
SPATIAL_ORDER_TURB= 1ST_ORDER
%
% Slope limiter (VENKATAKRISHNAN, MINMOD)
SLOPE_LIMITER_TURB= VENKATAKRISHNAN
%
% Time discretization (EULER_IMPLICIT)
TIME_DISCRE_TURB= EULER_IMPLICIT

% --------------------------- CONVERGENCE PARAMETERS --------------
------------%
%
% Convergence criteria (CAUCHY, RESIDUAL)
%
CONV_CRITERIA= RESIDUAL
%
% Residual reduction (order of magnitude with respect to the initial
value)
RESIDUAL_REDUCTION= 6
%
% Min value of the residual (log10 of the residual)
RESIDUAL_MINVAL= -10
%
% Start convergence criteria at iteration number
STARTCONV_ITER= 10
%
% Number of elements to apply the criteria
CAUCHY_ELEMS= 100
%
% Epsilon to control the series convergence
CAUCHY_EPS= 1E-6
%
%  Function  to  apply  the  criteria  (LIFT,  DRAG,  NEARFIELD_PRESS,
SENS_GEOMETRY,
%                        SENS_MACH, DELTA_LIFT, DELTA_DRAG)
CAUCHY_FUNC_FLOW= DRAG

% ------------------------- INPUT/OUTPUT INFORMATION --------------
------------%
%
% Mesh input file
MESH_FILENAME= final_mesh.su2
%
% Mesh input file format (SU2, CGNS, NETCDF_ASCII)
MESH_FORMAT= SU2
%
% Mesh output file
MESH_OUT_FILENAME= mesh_out.su2
%
```

```
% Restart flow input file
SOLUTION_FLOW_FILENAME= solution_flow.dat
%
% Restart adjoint input file
SOLUTION_ADJ_FILENAME= solution_adj.dat
%
% Output file format (PARAVIEW, TECPLOT, STL)
OUTPUT_FORMAT= TECPLOT
%
% Output file convergence history (w/o extension)
CONV_FILENAME= history
%
% Output file restart flow
RESTART_FLOW_FILENAME= restart_flow.dat
%
% Output file restart adjoint
RESTART_ADJ_FILENAME= restart_adj.dat
%
% Output file flow (w/o extension) variables
VOLUME_FLOW_FILENAME= flow
%
% Output file adjoint (w/o extension) variables
VOLUME_ADJ_FILENAME= adjoint
%
% Output objective function gradient (using continuous adjoint)
GRAD_OBJFUNC_FILENAME= of_grad.dat
%
% Output file surface flow coefficient (w/o extension)
SURFACE_FLOW_FILENAME= surface_flow
%
% Output file surface adjoint coefficient (w/o extension)
SURFACE_ADJ_FILENAME= surface_adjoint
%
% Writing solution file frequency
WRT_SOL_FREQ= 100
%
% Writing convergence history frequency
WRT_CON_FREQ= 1
```

APPENDIX E

RANS EQUATIONS USED IN SU2 CFD MODELLING

The governing equations used in SU2 CFD modelling based on RANS equations combined with turbulence modelling. Many turbulence modelling options are available in SU2 CFD. It has been verified that different turbulence modelling options in SU2 CFD lead to similar results [1]. The governing equations shown in here are taken from one of the papers from SU2 developers [1].

The complete system of equations Navier Stokes used in tensor are shown in Equation (E.1) [1].

$$\frac{\partial \vec{W}}{\partial t} + \nabla . \vec{F}_{conv} - \nabla . \vec{F}_{visc} - \vec{Q} = 0$$

$$\vec{W} = \begin{Bmatrix} \rho \\ \rho \vec{V} \\ \rho E \end{Bmatrix}, \vec{V} = \begin{Bmatrix} u \\ v \\ w \end{Bmatrix}, \vec{F}_{conv} = \begin{Bmatrix} \rho \vec{V} \\ \rho \vec{V}\vec{V} + \bar{\bar{I}}p \\ \rho E \vec{V} + p\vec{V} \end{Bmatrix} \tag{E.1}$$

$$\vec{F}_{visc} = \begin{Bmatrix} 0 \\ \bar{\bar{\tau}} \\ \bar{\bar{\tau}} . \vec{V} + \mu^*_{\text{total}} c_p \nabla T \end{Bmatrix}, \vec{Q} = \begin{Bmatrix} q_\rho \\ \vec{q}_{\rho \vec{V}} \\ q_{\rho E} \end{Bmatrix}$$

The term E expresses the total energy per unit mass. c_p is the specific heat at constant pressure, T is the temperature. The term $\bar{\bar{\tau}}$ expresses the viscous stress tensor defined in the RANS equations which is in tensor form can be defined in Equation (E.2).

$$\bar{\bar{\tau}} = \mu_{total} \left(\nabla \vec{V} + \nabla \vec{V}^T - \frac{2}{3} \bar{\bar{I}} (\nabla . \vec{V}) \right) \tag{E.2}$$

A perfect gas assumption is utilized with γ, a ratio of specific heats and R, gas constant. Based on this assumption, pressure, temperature, and specific heat are shown in Equation (E.3).

$$p = (\gamma - 1)\rho\left[E - \frac{1}{2}\vec{V}.\vec{V}\right], T = \frac{p}{\rho R}, c_p = \frac{\gamma R}{\gamma - 1} \tag{E.3}$$

The turbulence modelling is based on Boussinesq hypothesis that states the total viscosity is summation of laminar viscosity, μ_{dyn} and turbulence viscosity, μ_{turb}. The dynamic viscosity is computed as a function of Sutherland's formula (based on temperature only). On the other hand, the turbulent viscosity is computed based on turbulence modeling.

$$\begin{aligned} \mu_{tot} &= \mu_{dyn} + \mu_{turb} \\ \mu_{tot}^{*} &= \frac{\mu_{dyn}}{Pr_{dyn}} + \frac{\mu_{turb}}{Pr_{turb}} \end{aligned} \tag{E.4}$$

For the Spalart-Allmaras turbulence modelling, the turbulence viscosity is computed in Equation (E.5).

$$\mu_{turb} = \rho\hat{\nu}f_{v1}, f_{v1} = \frac{\chi^3}{\chi^3 + c_{\hat{\nu}1}^3}, \chi = \frac{\hat{\nu}}{\nu}, \nu = \frac{\mu_{dyn}}{\rho} \tag{E.5}$$

The term $\hat{\nu}$ is attained by solving a transport equation where the convective, viscous, and source terms are given as in Equation (E.6).

$$\begin{aligned} \vec{F}^c = \vec{V}\hat{\nu}, \vec{F}^v = -\frac{\nu + \hat{\nu}}{\sigma}\nabla\hat{\nu}, Q = c_{b1}\hat{S}\hat{\nu} - c_{w1}f_w\left(\frac{\hat{\nu}}{d_s}\right)^2 + \frac{c_{b2}}{\sigma}|\nabla\hat{\nu}|^2 \\ f_w = g\left[\frac{1 + c_{w_3}^6}{g^6 + c_{w_3}^6}\right]^{1/6}, g = r + c_{w2}(r^6 - r) \\ r = \frac{\hat{\nu}}{\hat{S}\kappa^2 d_s^2} \end{aligned} \tag{E.6}$$

The term d_s specifies the distance to the nearest wall. On the other hand, the term $\hat{S}$ defines the production term which is mathematically shown in Equation (E.7).

$$\hat{S} = |\vec{\omega}| + \frac{\hat{\nu}}{\kappa^2 d_s^2} f_{v_2}, \vec{\omega} = \nabla \times \vec{V}$$
$$f_{v_2} = 1 - \frac{\chi}{1 + \chi f_{v1}} \tag{E.7}$$

The constants used for this turbulence modelling are summarized in Equation (E.8).

$$\sigma = \frac{2}{3}, c_{b1} = 0.1355, c_{b2} = 0.622, \kappa = 0.41$$
$$c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma}, c_{w2} = 0.3, c_{w3} = 2, c_{v1} = 7.1 \tag{E.8}$$

In the computation, a no-slip condition is applied on the airfoil region. Furthermore, adiabatic condition is imposed on the airfoil boundary as well. The above equations are solved in SU2 by using Finite Volume Method with Upwind Scheme Moreover, the flux computation is done based on the Roe flux computation method.