The Pennsylvania State University
The Graduate School

INVISCID UNSTEADY AERODYNAMIC MODELS OF AIRFOILS MOVING NEAR BOUNDARIES

A Thesis in
Mechanical Engineering
by
Nilanjan Sen

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2008

The thesis of Nilanjan Sen was reviewed and approved* by the following:

Christopher D. Rahn
Professor of Mechanical Engineering
Thesis Advisor

Laura L. Pauley
Professor of Mechanical Engineering

Karen A. Thole
Professor of Mechanical Engineering
Department Head of Mechanical and Nuclear Engineering

*Signatures are on file in the Graduate School.

Abstract

There is considerable interest in the development of flapping wing Nano Air Vehicles (NAVs) for military applications. These NAVs operate in the Reynolds number regime $10^2 - 10^4$ which is similar to the Reynolds regime of small insects. It is important to understand the aerodynamics of flapping wing flight in this regime in order to build high performance NAVs. Aerodynamic models that can predict the flow and associated forces are a useful tool to understand flapping wing mechanisms. The Weis-Fogh clap-and-fling mechanism, is used by certain insects operating in the low Reynolds number range (10^2). The Weis-Fogh mechanism generates additional lift compared to conventional flapping wing mechanisms and is therefore advantageous to use in NAVs.

This thesis presents a two-dimensional, non-linear, unsteady aerodynamic model of an airfoil in a bounded domain. Key concepts of flapping wings such as leading edge vortices and wing-wake interaction are included in the model to provide a better prediction of the flow and aerodynamic forces. The model predicts the flow field and the lift and drag forces on the airfoil. The non-linear governing equations for the flow are solved using numerical methods. The Weis-Fogh clap-and-fling mechanism is studied and compared with previous CFD results.

Table of Contents

List of Figures

List of Tables

List of Symbols

A_0	stroke amplitude (m), p. 30
C_1	inner cylinder corresponding to airfoil, p. 8
C_2	outer cylinder corresponding to wall, p. 8
F	fluid force/length (kgs^{-2}) in inertial frame of reference (ζ), p. 11
$\hat{F}$	fluid force/length (kgs^{-2}) in translating frame of reference ($\hat{\zeta}$), p. 11
R	radius of cylinder (m), p. 6
Re	Reynolds number
S	outer control surface, p. 12
S_b	inner body surface, p. 12
V	control volume in which fluid forces are calculated, p. 12
Z	coordinates of plane where airfoil is mapped to a 2-D cylinder, p. 6
c	chord length (m), p. 6
f	frequency of stroke (s^{-1}), p. 30
h	heave of airfoil (m), p. 6
l	lunge of airfoil (m), p. 6
$\hat{n}$	unit normal, p. 12
q	square of ratio of radii of outer and inner cylinder, p. 9
r	radial coordinates in ω-plane
r_1	radius of inner cylinder in ω-plane, p. 8
r_2	radius of outer cylinder in ω-plane, p. 8
t	time(s)

u_b fluid velocity on inner body (ms^{-1}), p. 12

u_s fluid velocity on outer control surface (ms^{-1}), p. 12

v_{edge} velocity at the edge of airfoil (ms^{-1}), p. 18

v_{max} maximum translational velocity of airfoil (ms^{-1}), p. 22

α pitch of airfoil (rad), p. 6

δ_{ij} distance between two vortices i and j, p. 17

δ_t time step (s)

ϵ size factor used during vortex amalgamation, p. 17

γ vorticity (ms^{-1})

$\gamma_{0,af}$ quasi-steady component of bound vorticity on airfoil (ms^{-1})

$\gamma_{1,af}$ unsteady component of bound vorticity on airfoil (ms^{-1})

$\gamma_{1,wall}$ unsteady component of bound vorticity on wall (ms^{-1})

γ_{lv} vorticity of leading edge wake (ms^{-1})

γ_{wk} vorticity of trailing edge wake (ms^{-1})

ω coordinates of plane where airfoil and wall are mapped to an annulus, p. 8

ω_{max} rotational constant ($rads^{-1}$), p. 22

ϕ complex potential, p. 5

ϕ_t phase lag in wing rotation (rad), p. 30

ψ stream function, p. 5

ρ density of fluid (kgm^{-3})

θ angle (angular coordinates) in Z or ω plane (rad)

$\Delta\theta$ total angle of rotation of airfoil, p. 22

τ dimensionless time, p. 22

τ_{accel}, τ_{decel} dimensionless start of acceleration and deceleration of airfoil, p. 22

τ_{turn} dimensionless start of rotation of airfoil, p. 22

$\Delta\tau_{accel}$, $\Delta\tau_{decel}$ dimensionless duration of acceleration and deceleration of airfoil, p. 22

$\Delta\tau_{rot}$ dimensionless duration of rotation of airfoil, p. 22

ζ airfoil frame of reference, p. 6

$\hat{\zeta}$ translating frame of reference, p. 6

$\tilde{\zeta}$ inertial frame of reference, p. 6

Acknowledgments

I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

I want to thank my advisor Dr. Chris Rahn for giving me the freedom to work on this project and provide valuable input throughout my research. I would also like to thank my labmates Amir, Deepak and Harish for their support and suggestions. I am deeply indebted to Dr. Chenglie Hu for providing me the software DSCPACK.

Chapter 1

Introduction

There is considerable interest in the development of Nano Air Vehicles (NAVs) for military operations. Current Unmanned Air Vehicles are too large to penetrate buildings for situational awareness and reconnaissance, emplace important sensors, and sample/return of material. While a variety of Micro Air Vehicles have been built and flown that use propellers and flapping wings for lift generation, NAVs, defined as weighing less than 10 grams with wingspans less than 7.5 cm, have yet to be flown. Significant challenges remain in miniaturizing power, control, actuation, and communications for NAVs.

Understanding the aerodynamics of flapping wing flight is essential to the design and operation of flapping wing NAVs. NAVs operate in the same low Reynolds number $10^2 - 10^4$ regime as insects and experience similar flight conditions. In this regime, the aerodynamic performance of rigid fixed wings drops significantly while flexible flapping wings gain efficiency and are the preferred method for small natural fliers. Developing aerodynamic models to predict the performance of these flapping systems is a key towards building successful NAVs.

1.1 Aerodynamic Basis

Insect wings generate forces much higher than predicted by quasi-steady theory, primarily due to unsteady effects. The discovery of the leading-edge vortex (LEV) has aided in understanding the high lift coefficients observed in insects. Ellington et.al. [1] and Vandenberg and Ellington [2] discover the formation of a three dimensional leading edge vortex in tethered flying hawkmoth and dragonflies. LEV's are a phenomena of wings at high angles of attack, and effectively increase the camber of the wing, adding to the vorticity (and lift) of the wing. A detailed description of the LEV is found in [3].

Wing rotation also contributes to the generated lift by introducing a rotational circulation. Depending upon the direction and timing of the rotation, the circulation developed can either add or subtract to the net generated lift. Dickinson and Sane [4], using a dynamically scaled model of

the fruit fly, investigate the effect of rotational timing on the force generated. In addition to the lift due to the translational and rotational motion of the wing, the wing can also interact with the previously shed wake. Sane and Dickinson [4] discover differences between the forces generated by the scaled model of the fruit fly and those predicted using a revised quasi-steady model. These differences are attributed to the wing-wake interaction. Therefore wing-wake interaction must also be included in any model that attempts to describe the aerodynamics of flapping wings.

Weis Fogh [5], proposes a new aerodynamic mechanism that increases lift during flight, during his study of the hover of the *chalcid* wasp *Encarsia Formosa*. This mechanism has become known as the Weis-Fogh mechanism, and the corresponding motion has been termed the 'clap-and-fling'. The Weis-Fogh mechanism is used by some insects on the lower Reynolds number regime (10^2) to increase their lift. At the end of a flapping stroke the two wings of the insect 'clap' together. Then, at the start of the downstroke the wings open rapidly about the trailing edge (termed as the 'fling'). This forms large leading edge vortices, of opposite signs, on both wings. Therefore, although the net circulation about each individual wing is non-zero the circulation of the entire system remains zero. As a result, the formation of trailing edge vortices is not required to preserve the circulation, resulting in larger lift forces. Thus, incorporating clap-and-fling in a normal stroke is desirable.

1.2 Analytical Models

Early models of insect flight by Rayner [6] and Ellington [1] are derived using far field models of the wake that approximate flapping wings by propellers and calculate the net force generated by the downward air jet that keep the insect in hover. Although lift and power requirements for a flapping insect can be approximated from these models they cannot be used to provide a detailed description of the wing behavior with time.

The quasi-steady approach is the simplest analytical model to approximate the forces on an airfoil. The lift and drag coefficients are taken as time-invariant and empirically fit to wind-tunnel data. Dickinson and co-workers [7],[4],[8] use a revised quasi-steady model that includes the effect of wing rotation at the end of each stroke. The forces calculated using this quasi-steady model match quite well with experimental results. However, the effect of the wake is not included in the quasi-steady model and the forces at stroke reversal are therefore underestimated.

There are a number of analytical models that include the effect of airfoil-wake interaction. Lam [9] uses the classical unsteady airfoil theory of von Karman and Sears [10], without some linearizing assumptions to model the trailing edge wake evolution for an airfoil in a severe maneuver. Minotti [11] provides a non-linear model where the wake is divided into the LEV and the trailing edge wake. The position of the LEV on the airfoil in Minotti's model is determined empirically.

Following Lam [9], Ansari et.al. [12],[13], include the effect of the LEV to provide a comprehensive non-linear unsteady model for flapping wings. Using a blade-element approach, Ansari converts the general 3-D flapping wing problem to a quasi-3D problem. His results match well

with the experimental data published by Dickinson [8]. At present the method developed by Ansari is the most rigorous and inclusive non-CFD model to represent the flow around a flapping airfoil. Ansari's method, however is only applicable for a single airfoil in an unbounded domain.

Tomotika [14],[15] and Dappen [16] study the problem of wall interference on an airfoil at low angles of attack, using potential theory and conformal mapping. The particular Schwarz-Christoffel mapping used by Dappen is implemented in the current approach to model clap-and-fling motion. Lighthill [17], Edwards and Cheng [18], and Wu and Hu-Chen [19] develop analytical models for the 'fling'. They use a separation vortex model, where at each time step vorticity is added to a single leading edge vortex, to calculate the circulation and lift generated during fling. Their results are verified by the experimental results of Spedding and Maxworthy [20].

Miller and Peskin [21] use computational fluid dynamics, to calculate the lift and drag coefficients for an idealized complete 2-D 'clap-and-fling' stroke with Reynolds numbers (Re) varying from 8 to 128. They find that the lift coefficients are maximum for lower Re and decrease with increasing Re. Dickinson and Lehmann [22] experimentally investigate the effect of a 'clap-and-fling' motion on the generated lift and report a 15% increase in lift.

1.3 Numerical Schemes

Rosenhead [23] makes the first attempt at a discrete vortex model in his study of the Kelvin-Helmholts instability in vortex sheets. His algorithm uses point vortices to solve the 2-D inviscid Euler equations. Point vortices have infinite energy, however, and their use can lead to spurious results. Chorin [24] is the first to develop a computationally effective vortex method where he introduces key concepts like finite vortex cores and maintenance of no-slip boundary conditions by introducing new vortices at the boundary.

Vortex convection schemes are discussed in detail by Majda and Bertozzi [25] and Lewis [26]. Greengard and Rokhlin [27] present a novel algorithm to rapidly calculate the gravitational potential of a large number of particles. The algorithm is known as the 'Fast Multipole Method' (FMM). Strickland and Baty [28] implement the FMM algorithm for vortex problems.

1.4 Objective

This thesis extends Ansari's unsteady model to a single airfoil bounded by an outer wall and uses this modified theory to study 'clap-and-fling' aerodynamics. The present approach simplifies the two-airfoil clap-and-fling problem to a single airfoil with an artificial symmetry wall. Thus, the unbounded fluid domain in clap-and-fling is simplified to a bounded fluid domain where the symmetry wall acts as one side of the outer boundary. Essentially, the problem then reduces to modeling the flow in a doubly connected bounded fluid domain. The model is based on the circulation approach and includes the effect of wing-wake interaction. This non-linear unsteady

model represents the original contribution of this thesis. This approach has other applications including, for example, evaluating wind tunnel wall effects on an airfoil.

Chapter 2 describes the underlying theory for the proposed unsteady non-linear model. The current approach relies on conformally mapping the doubly-connected domain of interest along with the two boundaries to an annular domain. The flow is then solved for in the annular domain using potential theory and a circulation approach. The resulting flow is then mapped back to the domain and lift and drag forces are calculated.

An overview of the numerical implementation of the aerodynamic model described in Chapter 2, is presented in Chapter 3. Various methods are used to reduce the computation time and increase the stability and accuracy of the solution. In Chapter 4 the unsteady model and numerical approach are applied to clap-and-fling. The wings never touch each other during clap and fling, however, so a small gap remains. Thus, this can be considered a 'near-fling' motion of two airfoils. The results are compared with the CFD results of Miller and Peskin [21]. Chapter 5 provides the main conclusions and recommendations for future work.

Chapter 2

Airfoil in a Bounded Domain

2.1 Analysis

This section provides a brief description of the work of Lam [9] and Ansari et. al. [12] that is the basis for the present approach. In the following analysis, the solid boundaries (airfoil and wall) contain a continuous distribution of bound vortices that move with the boundaries. The wakes that emerge from the leading and trailing edge of the airfoil are also continuous distribution of vortices that are free to move in the fluid. The flow problem is solved using conformal mapping.

2.1.1 Potential Approach

Analysis of the flow around an airfoil can be described using a potential-flow model. Assuming an inviscid incompressible flow, the Navier-Stokes equations reduce to the Euler equations. Furthermore, if the flow is assumed to be irrotational everywhere except at the boundaries and discontinuities in the wake, then the flow is modeled by the Laplace equation,

$$\nabla^2(\phi + \iota\psi) = 0, \tag{2.1}$$

where ϕ and ψ are the complex potential and stream functions, respectively. Equation 2.1 is linear so the principle of linear superposition is used to combine the effects of the various components contributing to the fluid flow. In [12],[13] and the present approach, the flow is subdivided into wake-free (quasi-steady) and wake-induced (unsteady) components (see fig. 2.1). The wake-free part is further divided into the effect due to the free-stream velocity of the fluid and the effect due to the unsteady motion of the airfoil. These components exclude the wake so they are independent of the history of the fluid and are called the quasi-steady components. The free stream component is subsequently neglected by assuming that the free stream velocity of the fluid is zero. This is true, for example, during hover of a flapping wing NAV.

The wake induced components are divided into contributions of the LEV and the trailing-edge

wake. These components depend on the previous time history of the fluid and are termed as the unsteady part.

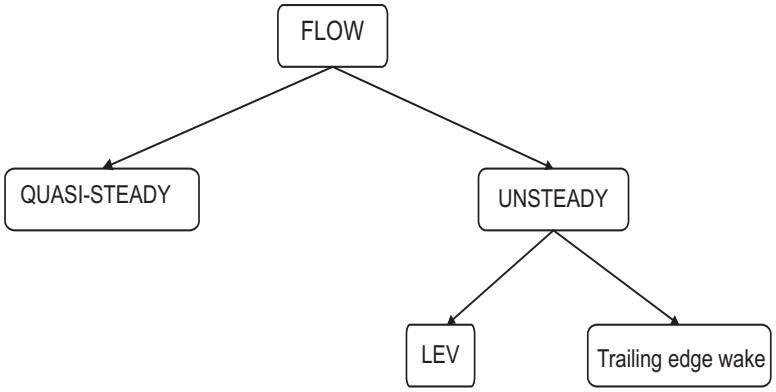

Figure 2.1. Flow components

2.2 Sign Conventions and Coordinate Systems

Circulation and vorticity are positive in the counterclockwise ($\curvearrowleft$) direction. The airfoil (length c) has three degrees of freedom: Lunge (horizontal movement, l), heave (vertical movement, h) and pitch (angular rotation, α). Positive lunge is defined to the right, positive heave is upward, and pitch is defined positive clockwise ($\curvearrowright$). Three different coordinate systems are defined: Airfoil (ξ, η), translating $(\hat{\xi}, \hat{\eta})$ and inertial $(\tilde{\xi}, \tilde{\eta})$. The airfoil coordinate frame is located at the centre of the airfoil with the abscissa coinciding with the airfoil. The translating coordinate frame is fixed to the airfoil at the pitching centre but aligned with the inertial frame. The inertial coordinate frame is fixed.

2.3 Conformal Mapping

Exact solutions of the Laplace equation for simple regions such as circles, rectangles and annuli are easily determined. It is therefore convenient to transform complex domains to such simple regions, solve the Laplace equation, and transform the solution back into the original complex domain. For example, the simple region outside of a two-dimensional cylinder (Z-plane) is mapped to the complex region outside of a flat plate (ζ-plane) using the Joukowski transformation,

$$\zeta = Z + \frac{R^2}{Z}, \tag{2.2}$$

where R is the radius of the cylinder and $c = 4R$ is the length of the airfoil (see fig. 2.3). The trailing (ζ_1) and leading (ζ_2) edges of the airfoil correspond to the angle $\theta = \pi$ and 0 on the cylinder respectively. The center of the airfoil lies at the center of the cylinder in the Z-plane.

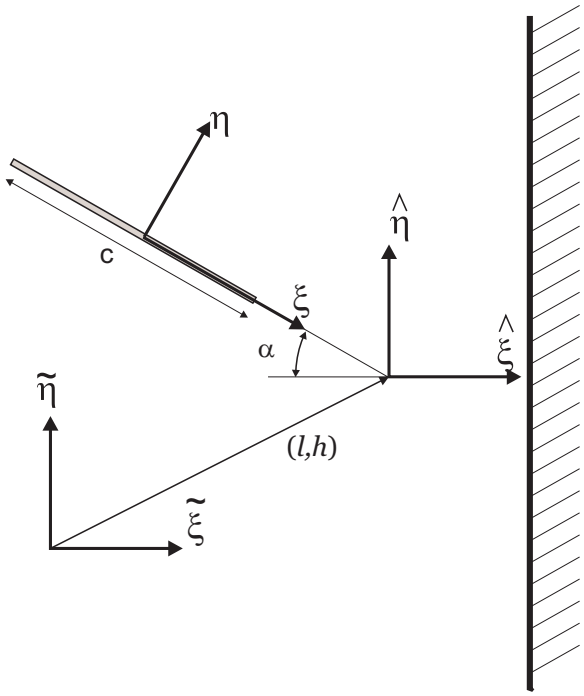

Figure 2.2. Coordinate system

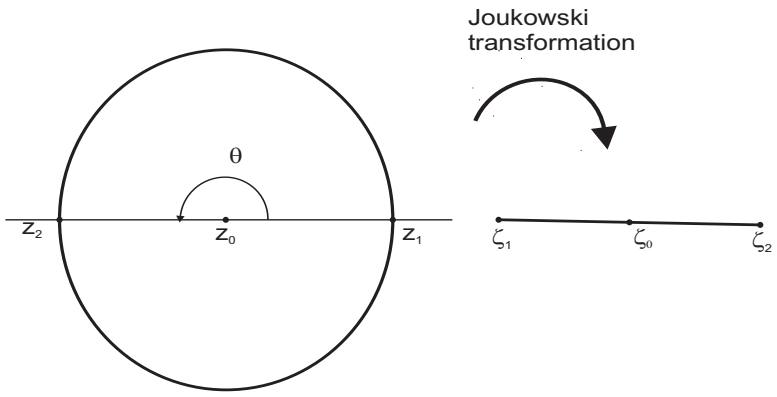

Figure 2.3. Joukowski mapping

The simple region between two concentric two-dimensional cylinders (C_1 and C_2 in ω-plane) can be mapped to a doubly connected polygonal region (in the inertial $\tilde{\zeta}$-plane) using a particular Schwarz-Christoffel (S-C) transformation. D is the doubly connected polygonal region whose boundary curves, P_2 and P_1 are closed polygons. The vertices of P_2 are $\tilde{\zeta}_{21}$,$\tilde{\zeta}_{22}$,...,$\tilde{\zeta}_{2m}$ and let the vertices of P_1 be $\tilde{\zeta}_{11}$,$\tilde{\zeta}_{12}$,...,$\tilde{\zeta}_{1n}$. The S-C transformation maps the annulus A, $0 < r_1 < |\omega| < r_2$, to the doubly connected polygonal region D. The mapping is not defined, however, on the set of points $\tilde{\zeta}_{21....k}$ and $\tilde{\zeta}_{11...k}$ ($\omega_{21...k}$ and $\omega_{11...k}$ in the w-plane). These points are termed as the singularities in the mapping. Chenglie-Hu [19], developed DSCPACK to numerically implement this S-C transformation. A detailed explanation of the numerical implementation of the S-C

mapping is found in [19].

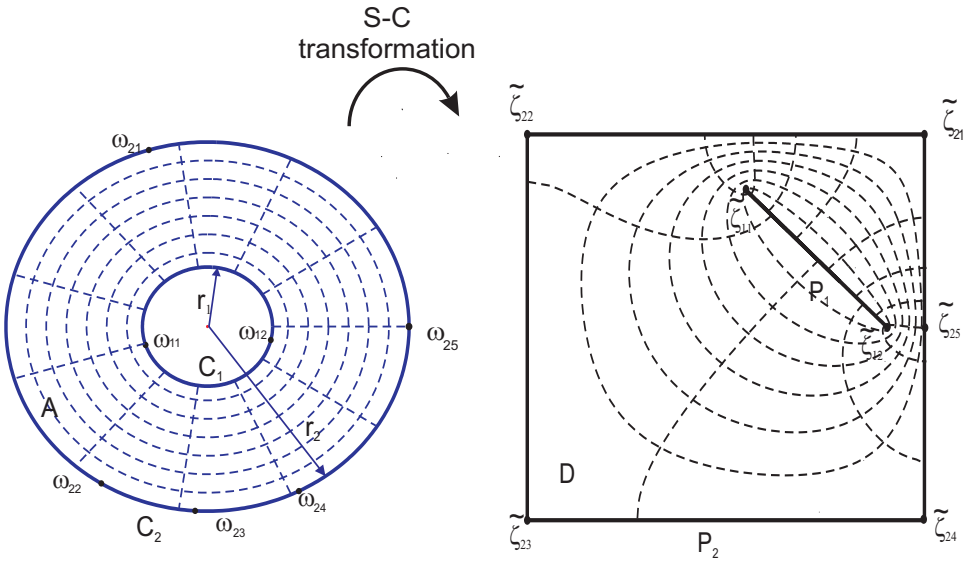

Figure 2.4. The S-C transformation used to map the airfoil and wall (D) into the annular region (A)

Fig. 2.4 shows the particular geometry used in the present investigation. The outer boundary is modeled as a rectangle. The airfoil is modeled as a flat plate with vertices at the leading and trailing edge. The location of the leading and trailing edge on the inner cylinder (C_1) in the annular domain depends upon the orientation and proximity of the airfoil to the wall. In the following analysis both mappings, Joukowski and Schwarz-Christoffel, are used to simplify calculations.

2.3.1 Identities

Velocity v and vorticity γ fields are transformed using,

$$\begin{aligned} v(\zeta) &= \frac{v(Z)}{d\zeta/dZ} = \frac{v(w)}{d\zeta/dw}, \\ \gamma(\zeta) &= \frac{\gamma(z)}{d\zeta/dZ} = \frac{\gamma(w)}{d\zeta/dw}. \end{aligned} \tag{2.3}$$

Circulation is the same in all regions ($\Gamma(\zeta) = \Gamma(Z) = \Gamma(w)$).

2.4 Quasi-Steady Component

The movement of the airfoil causes quasi-steady fluid motion in the boundary layer. To maintain zero flow through the airfoil, bound vortices are added at the airfoil surface. This bound unsteady vorticity at point $Z = Re^{\iota\theta}$ is given by Ansari et. al. [12] as,

$$\gamma_{0,af}(\theta) = \frac{1}{R}\left[-2R(\dot{l}\sin\alpha + \dot{h}\cos\alpha)\cos\theta - 2R^2\,\dot{\alpha}\cos 2\theta + \frac{\Gamma_0(t)}{2\pi}\right], \tag{2.4}$$

where

$$\Gamma_0(t) = 2\pi \left[2R(\dot{l}\sin\alpha + \dot{h}\cos\alpha) - 2\,\dot{\alpha}\,R(R+a) \right]. \tag{2.5}$$

The bound vorticity, $\gamma_{0,af}$, in the ω-plane is then calculated using a discretized approach (see section 3.1.2). The wall is stationary so there are no induced velocities on the wall.

2.5 Unsteady Model Development

Flow separation takes place at the leading edge (denoted by the subscript le) and trailing edge (denoted by the subscript te) of the airfoil. Two wakes are then shed from the two separation points, the LEV (denoted by the subscript lv) and the trailing edge wake (denoted by the subscript wk). The induced velocity on the airfoil due to these shed wakes is then used to calculate the unknown circulation of the wake.

In the annular domain the flow induced by a single vortex is determined by adding multiple image vortices at both cylinders (see Appendix A). The induced flow is then calculated by summing the contribution of the original vortex and the image vortices. For a vortex of strength $\mathrm{d}\Gamma$, situated between two coaxial cylinders of radii r_1 and r_2 ($r_1 < r_2$) at ω_0, the induced velocity $v(\omega)$ is

$$\overline{v}(\omega) = -\imath\frac{d\Gamma}{2\pi}\sum_{n=-\infty}^{\infty}\left[\frac{1}{\omega - \omega_0 q^n} - \frac{1}{\omega - \frac{r_1^2 q^n}{\overline{\omega_0}}}\right], \tag{2.6}$$

where $q = \frac{r_2^2}{r_1^2}$ [29].

Using the S-C mapping the fluid domain between the airfoil and the wall is transformed into an annular domain with inner cylinder radius r_1, and outer cylinder radius r_2. The inner cylinder corresponds to the airfoil and the outer cylinder corresponds to the wall. The circumferential velocity induced by the LEV and trailing edge wake on the cylinders can be derived from equation 2.6. Vorticity and velocity are interchangeable, so the vorticity (and velocity) induced on a point $\omega = re^{\imath\theta}$ on any of the cylinders is,

$$\begin{aligned}\gamma_1(r,\theta) =& -\frac{1}{2\pi}\oint_{wk}\sum_{n=-\infty}^{\infty}\left(\frac{re^{\imath\theta}}{re^{\imath\theta} - w_{wk}q^n} - \frac{re^{\imath\theta}}{re^{\imath\theta} - \frac{q^{n-1}}{\overline{w}_{wk}}}\right)\gamma_{wk}dw_{wk} \\ & -\frac{1}{2\pi}\oint_{lv}\sum_{n=-\infty}^{\infty}\left(\frac{re^{\imath\theta}}{re^{\imath\theta} - w_{lv}q^n} - \frac{r_1e^{\imath\theta}}{re^{\imath\theta} - \frac{q^{n-1}}{\overline{w}_{lv}}}\right)\gamma_{lv}dw_{lv}.\end{aligned} \tag{2.7}$$

The two unknown vorticities (γ_{wk} and γ_{lv}), are obtained by satisfying the Kutta-Joukowski condition at the trailing edge and the stagnation condition at the leading edge. To ensure smooth flow at the trailing edge of the airfoil, the Kutta-Joukowski condition ($v_\theta = 0$) is imposed. An additional circulation Γ_k is therefore added to the right hand side of eq. 2.7 so that at the trailing edge ($w_{te} = r_1e^{\imath\theta_{te}}$),

$$\begin{aligned}\Gamma_k =&\frac{1}{2\pi}\oint_{wk}\sum_{n=-\infty}^{\infty}\left(\frac{r_1e^{\imath\theta_{te}}}{w_{te}-w_{wk}q^n}-\frac{r_1e^{\imath\theta_{te}}}{r_1e^{\imath\theta_{te}}-\frac{q^{n-1}}{\overline{w}_{wk}}}\right)\gamma_{wk}dw_{wk}\\ &+\frac{1}{2\pi}\oint_{lv}\sum_{n=-\infty}^{\infty}\left(\frac{r_1e^{\imath\theta_{te}}}{r_1e^{\imath\theta_{te}}-w_{lv}q^n}-\frac{r_1e^{\imath\theta_{te}}}{r_1e^{\imath\theta_{te}}-\frac{q^{n-1}}{\overline{w}_{lv}}}\right)\gamma_{lv}dw_{lv}.\end{aligned} \tag{2.8}$$

.

Applying Kelvins circulation theorem over the closed outer wall (assuming that there is no flow through the wall in order to satisfy the material boundary condition at the wall) the circulation of the entire system must remain constant with time. Since the net circulation is zero at the beginning (assuming that the fluid and airfoil are at rest at t=0), therefore at any time t the net circulation must remain zero. The image vortices cancel out the contribution of the wake and the LEV vortices so that,

$$\Gamma_0(t)+\Gamma_k(t)=0. \tag{2.9}$$

When vortices get too close to the wall, however, they dissipate due to the surface viscosity of the wall. This represents a violation of the no flow through condition on the wall. A modification to eq. 2.9 is added to satisfy the circulation theorem. If $\Gamma_{out}(t)$ is the total circulation dissipating into the wall from t=0 to t, then

$$\Gamma_0(t)+\Gamma_k(t)+\Gamma_{out}(t)=0. \tag{2.10}$$

Therefore,

$$\Gamma_k(t)=-\Gamma_0(t)-\Gamma_{out}(t). \tag{2.11}$$

Substituting eq. 2.11 into eq. 2.8, gives the first constraint equation,

$$\begin{aligned}-\Gamma_0(t)-\Gamma_{out}(t)=&\frac{1}{2\pi}\oint_{wk}\sum_{n=-\infty}^{\infty}\left(\frac{r_1e^{\imath\theta_{te}}}{r_1e^{\imath\theta_{te}}-w_{wk}q^n}-\frac{r_1e^{\imath\theta_{te}}}{r_1e^{\imath\theta_{te}}-\frac{q^{n-1}}{\overline{w}_{wk}}}\right)\gamma_{wk}dw_{wk}\\ &+\frac{1}{2\pi}\oint_{lv}\sum_{n=-\infty}^{\infty}\left(\frac{r_1e^{\imath\theta_{te}}}{r_1e^{\imath\theta_{te}}-w_{lv}q^n}-\frac{r_1e^{\imath\theta_{te}}}{r_1e^{\imath\theta_{te}}-\frac{q^{n-1}}{\overline{w}_{lv}}}\right)\gamma_{lv}dw_{lv}.\end{aligned} \tag{2.12}$$

At the leading edge $w_{le}=r_1e^{\imath\theta_{le}}$, the velocity (or vorticity) is zero, thus

$$\begin{aligned}\gamma_1(r_1,\theta_{le}) = \gamma_{0,af}(\theta_{le}) + \left(\frac{\Gamma_k(t)}{2\pi r_1}\right) \\ - \frac{1}{2\pi}\oint_{wk}\sum_{n=-\infty}^{\infty}\left(\frac{r_1 e^{\imath\theta_{le}}}{r_1 e^{\imath\theta_{le}} - w_{wk}q^n} - \frac{r_1 e^{\imath\theta_{le}}}{r_1 e^{\imath\theta_{le}} - \frac{q^{n-1}}{\overline{w}_{wk}}}\right)\gamma_{wk}dw_{wk} \\ - \frac{1}{2\pi}\oint_{lv}\sum_{n=-\infty}^{\infty}\left(\frac{r_1 e^{\imath\theta_{le}}}{r_1 e^{\imath\theta_{le}} - w_{lv}q^n} - \frac{r_1 e^{\imath\theta_{le}}}{r_1 e^{\imath\theta_{le}} - \frac{q^{n-1}}{\overline{w}_{lv}}}\right)\gamma_{lv}dw_{lv} \\ = 0.\end{aligned} \tag{2.13}$$

Rearranging terms in eq. 2.13 yields the second constraint on the wake,

$$\begin{aligned}\gamma_{0,af}(\theta_{le}) - \frac{\Gamma_0(t) + \Gamma_{out}(t)}{2\pi r_1} = \\ \frac{1}{2\pi}\oint_{wk}\sum_{n=-\infty}^{\infty}\left(\frac{r_1 e^{\imath\theta_{le}}}{r_1 e^{\imath\theta_{le}} - w_{wk}q^n} - \frac{r_1 e^{\imath\theta_{le}}}{r_1 e^{\imath\theta_{le}} - \frac{q^{n-1}}{\overline{w}_{wk}}}\right)\gamma_{wk}dw_{wk} \\ + \frac{1}{2\pi}\oint_{lv}\sum_{n=-\infty}^{\infty}\left(\frac{r_1 e^{\imath\theta_{le}}}{r_1 e^{\imath\theta_{le}} - w_{lv}q^n} - \frac{r_1 e^{\imath\theta_{le}}}{r_1 e^{\imath\theta_{le}} - \frac{q^{n-1}}{\overline{w}_{lv}}}\right)\gamma_{lv}dw_{lv}.\end{aligned} \tag{2.14}$$

The wakes γ_{wk} and γ_{lv} are uniquely defined by equations 2.12 and 2.14. However, these equations are non-linear and do not have closed-form solutions. The constraint equations are therefore solved numerically using the time-marching algorithm described in Chapter 3.

2.6 Forces

Given an arbitrary, time-dependent control volume V(t) bounded externally by a surface S(t) and internally by the body surface $S_b(t)$, the fluid force F (for a fluid of density ρ) acting on the body is given by Noca et. al. [30]. Figure 2.5 shows the described control volume problem, where $\hat{n}$ is a unit normal vector, u is the flow velocity, u_b is the body wall velocity, u_s is the velocity of the outer surface $S(t)$, γ is the vorticity field and ζ is the position vector. Although the inner body corresponding to the airfoil has been modeled as an ellipse in fig. 2.5, it can be of any arbitrary shape in Noca's formulation.

For the case of a 2-D inviscid flow, the surfaces and control volume are assumed to extend to infinity in the third dimension. $S(t)$ is assumed to be infinitesimally close to the outer solid wall, such that $u_{S(t)} = \gamma_{1,wall}$. The outer boundary $S(t)$ is stationary, and the fluid flow is tangential to both boundaries ($u \cdot \hat{n} = 0$). A simplified expression for evaluating the fluid force in such a

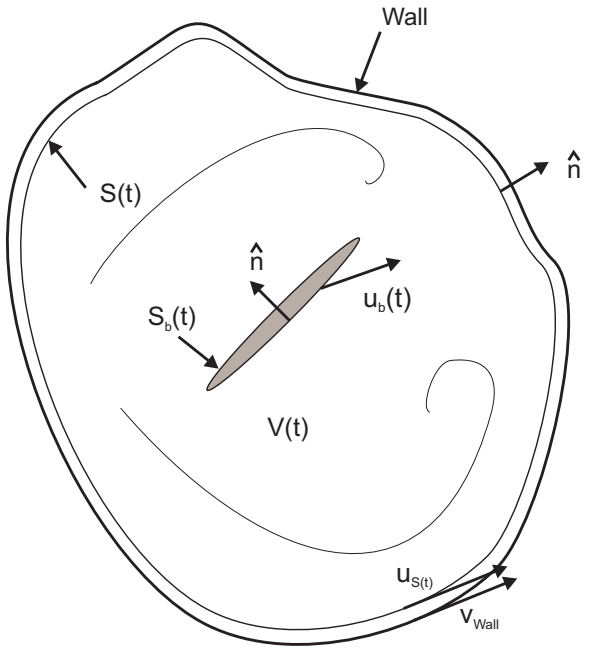

Figure 2.5. Finite domain for evaluation of fluid forces

system is given as,

$$\begin{aligned}\frac{F}{\rho} = &- \frac{d}{dt}\int_{V(t)} \zeta \times \gamma dV + \frac{d}{dt}\oint_{S_b(t)} \zeta \times (\hat{n} \times u) dS \\ &+ \frac{1}{2}\oint_{S(t)} |u|^2 \hat{n} dS.\end{aligned} \tag{2.15}$$

The first two terms in eq. 2.15 are equivalent to the force calculated on the body in an infinite fluid domain. For the case of a 2-D inviscid flow, where the control area is bounded by an outer wall, the fluid force acting on the inner body reduces to,

$$\begin{aligned}\hat{F}(t) = &\iota\rho\frac{d}{dt}\oint_{af} \hat{\zeta}\gamma_{0,af}(\hat{\zeta})d\hat{\zeta} + \iota\rho\frac{d}{dt}\oint_{af} \hat{\zeta}\gamma_{1,af}(\hat{\zeta})d\hat{\zeta} - \frac{\iota}{2}\rho\oint_{wall} |\gamma_{1,wall}|^2 d\hat{\zeta} \\ &+ \iota\rho\frac{d}{dt}\int_{wk} \hat{\zeta}\gamma_{wk}(\hat{\zeta})d\hat{\zeta} + \iota\rho\frac{d}{dt}\int_{lv} \hat{\zeta}\gamma_{lv}(\hat{\zeta})d\hat{\zeta},\end{aligned} \tag{2.16}$$

where subscript af means the integral is taken over the airfoil, subscript $wall$ means the integral is taken over the outer wall, the subscripts wk and lv mean the integral is taken over the respective shed wakes and the forces are calculated in the translating coordinate system ($\hat{\zeta}$-plane). The terms within the differentiation are called the force impulse.

Chapter 3

Numerical Implementation

3.1 Implementation

The airfoil problem is formulated as an initial value problem. The airfoil and fluid are assumed to be initially at rest and the flow for all subsequent time is solved using a time-marching procedure. During unsteady motion of the airfoil the kinematic boundary condition induces fluid velocities over the airfoil surface. To impose the stagnation condition at the edges, the induced velocity is modeled as bound circulation or vorticity on the airfoil. Subsequently at every time step a pair of vortices are shed from the leading and trailing edges of the airfoil in order to preserve the stagnation conditions and Kelvin's circulation law. During the time step, these shed vortices convect with the local Kirchoff velocity which in turn disturbs the stagnation conditions on the edges. Therefore, at the next time step a new pair of vortices is shed to restore the stagnation condition and the process is repeated.

3.1.1 Numerical Scheme Algorithm

Figure 3.1 describes the algorithm used for the numerical scheme. The basic procedure is as follows:

1. The problem is initialized by defining the airfoil parameters such as chord length, initial position and velocity.
2. A time marching algorithm is invoked to proceed with each time step.
    i. The airfoil position and velocity are calculated at the present time step.
    ii. The quasi steady circulation of the airfoil is calculated in the Z-plane.
    iii. A new pair of leading and trailing edge vortices are shed and positioned appropriately in the wake.

iv. The entire domain is mapped to the ω-plane, where the circulations of the newly shed vortices are calculated from the constraint equations.

v. The newly created vortices change the bound vorticity around the airfoil and wall. These modified bound circulations are then determined.

vi. The domain is mapped back to the $\tilde{\zeta}$-plane.

vii. The circulations and positions of all bound and shed vortices are used to compute the impulse in the translating coordinate frame.

viii. The entire wake system is convected.

3. After the final time step is reached, forces on the airfoil are computed from the impulse data series.

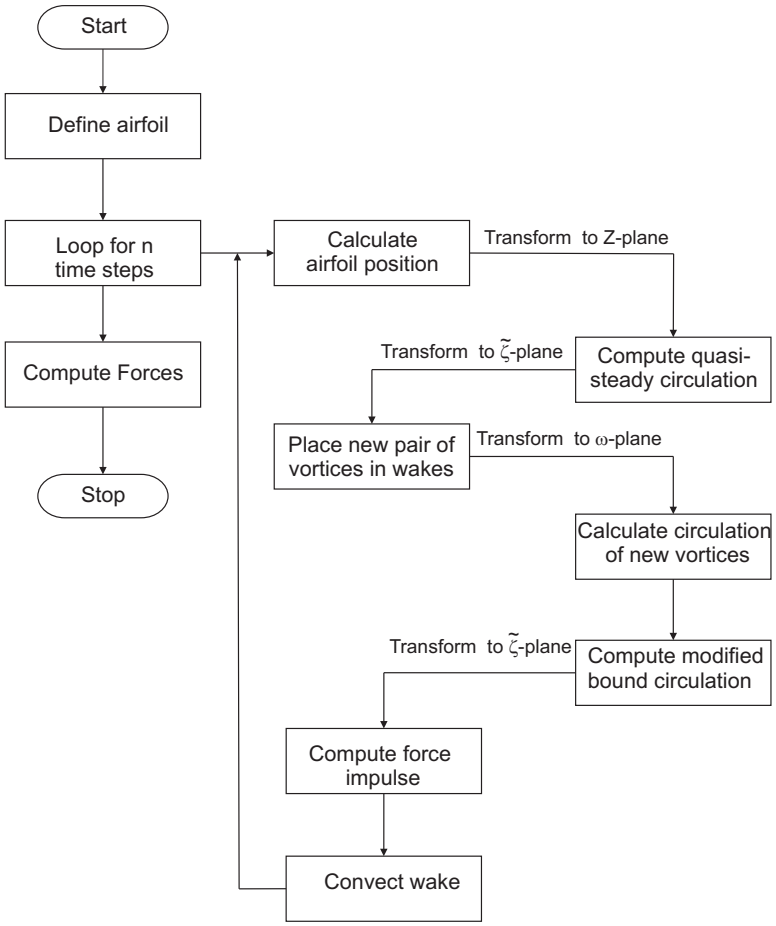

Figure 3.1. Flowchart depicting numerical scheme

3.1.2 Discretization of Equations

A number of equations derived in Chapter 2 do not have closed-form solutions and are therefore solved numerically. The expression for quasi-steady vorticity, eq. 2.5, has a simple form in the Z-plane (Joukowski mapping). However, it is difficult to compute the equivalent expression in the ω-plane, because of the complex nature of the SC-mapping. So a discrete element approach is used to obtain a numerical solution.

According to eq. 2.3 the circulation (Γ) of a bound vortex on a discrete element of the airfoil will remain constant irrespective of the plane in which it is being calculated (ζ, ω or Z plane). Therefore the airfoil is discretized and the bound vorticity associated with each discrete element is calculated. The airfoil (inner-cylinder) is first discretized in the ω-plane from which the corresponding discretized elements of the airfoil in the Z-plane ($dw \rightarrow dZ$) are obtained through successive conformal transformations from $w \rightarrow \zeta$ (with the help of DSCPACK) and then from $\zeta \rightarrow Z$ (using eq. 2.2). The bound circulation associated with each discrete element is known in the Z-plane ($\Gamma = \int \gamma dZ$) so the bound vorticity in the ζ and ω planes can be derived. To calculate the induced vorticity (velocity) on the airfoil and wall the discretized forms of eq. 2.7 are used. The quasi-steady vorticity in the ω-plane is then used to solve the constraint equations 2.12 and 2.14.

The continuous distribution of the vorticity over the wake is modeled by an array of discrete vortices. Therefore, the integrals of the constraint equations 2.12 and 2.14 become summations. Assuming that the circulation and position of all free vortices shed previously is known and the position of the newly shed pair vortices has been determined from eq. 3.5, then the discretized constraint eqs. 2.12 and 2.14 reduce to a pair of linear equations which can be solved simultaneously to obtain the unknown circulation of the latest shed vortices. Once the circulation and position of all the free vortices have been calculated, the wake-induced unsteady vorticity is determined from the discretized form of eq. 2.7.

3.1.3 Vortex Convection

The convection of the free vortices is calculated in the inertial coordinate frame ($\tilde{\zeta}$). Convection of the free vortices is due to the induction of the entire vortex system (free and bound vortices). The vortex system consists of the bound vortices on the airfoil ($\gamma_{0,a/f}$ and $\gamma_{1,a/f}$), the bound vortices on the wall ($\gamma_{1,wall}$), and the free vortices (γ_{wk} and γ_{lv}). Hence, by the Biot-Savart Law, the resultant convection velocity at a point ζ_0 is,

$$\begin{aligned}\overline{v}(\zeta_o) = &-\frac{\imath}{2\pi}\oint_{a/f}\frac{\left(\gamma_{0,a/f}+\gamma_{1,a/f}\right)d\zeta}{\zeta_o-\zeta} - \frac{\imath}{2\pi}\oint_{wall}\frac{(\gamma_{1,wall})\,d\zeta}{\zeta_o-\zeta} \\ &-\frac{\imath}{2\pi}\int_{wk}\frac{(\gamma_{wk})\,d\zeta}{\zeta_o-\zeta} - \frac{\imath}{2\pi}\int_{lv}\frac{(\gamma_{lv})\,d\zeta}{\zeta_o-\zeta}.\end{aligned} \tag{3.1}$$

The above eq. 3.1 is however given in terms of the distributed vorticity of the system. To

provide a numerical solution to the problem, the distributed vorticity on the system is modeled by discrete vortices and the integrals become summations.

A forward difference Euler scheme is used for vortex convection. Each time-step is further divided into sub-steps, to improve the numerical accuracy of the scheme,

$$\zeta(t+\delta t) = \zeta(t) + n\sum_{i=1}^{n} v_i(t)\frac{\delta t}{n}, \tag{3.2}$$

where δt represents each sub-step, n is the number of substeps per time-step and v_i is the convection velocity which is evaluated at each sub-step. There are limitations to the minimum time step possible during the simulation. Ordinarily in a simple time marching algorithm a smaller time step implies greater accuracy with an increase in the computational cost. This is not possible in the present algorithm since the program DSCPACK cannot distinguish between two distinct points beyond a certain minimum distance. A smaller time step would reduce the distance between any two discrete shed vortices and after a certain limit DSCPACK can no longer successfully map these discrete vortices to the annular domain. This limitation is mainly observed when the minimum distance between the wall and airfoil becomes large ($\geq 3c$) and the radius of the inner cylinder (corresponding to the inner airfoil) reduces to a small value. Care must, therefore, be taken while choosing an appropriate time step so that the simulation runs smoothly without compromising the accuracy of the numerical scheme.

To prevent infinite induced velocities when two vortices are very close to each other, finite vortex cores are introduced. The model proposed in Majda and Bertozzi [25], is used here,

$$v_\theta(r) = \frac{\Gamma}{2\pi}\frac{r^2}{\sqrt{r_c^4 + r^4}}, \tag{3.3}$$

where $v_\theta(r)$ is the circumferential velocity at radial distance r from the vortex of strength (Γ) and r_c is the vortex core radius. The vortex core radius must be chosen carefully. If a very small value is chosen, it would result in unrealistic induced velocities. On the other hand if the core radius is too high the validity of the discrete vortex approximation is questionable. The vortex core radii for all numerical calculations was taken to be between 2-10% of the chord length (c).

During convection, vortices may stray very close to the solid boundary. In a physical flow such vortices would dissipate when in close proximity to the boundary. Therefore, these vortices are artificially removed if they cross over to the other side of the boundary during convection. The strength of these removed vortices is adjusted in the bound circulation of the boundaries to satisfy the Kelvin circulation theorem.

3.1.3.1 Fast Multipole Method

Calculating the induced velocity of all the discrete vortices consumes the most computation time of the proposed approach. Individually calculating the induced velocity due to each vortex, the computation time varies as $O(N^2)$ with the number of vortices. In some problems the

number of vortices can be of the order 10^3 which drastically increases the computation time. The Fast Multipole Method reduces the computation cost to an $O(Nlog(N))$ and provides a significant reduction in computation time. The method takes advantage of multipole and local series expansions to calculate interactions between vortices which are in spatially well separated domains, see fig. 3.2. Most of the vortices are separated into domains A, B and D. Domain B is further subdivided into domains B1 and B2 to satisfy the limit on the maximum number of vortices allowed in one domain. Domains A, B1 (or B2) and D are considered as 'well separated' domains so the effect of each of these domains are calculated in one step. However domains B1 and B2 are adjacent and their interaction has to be evaluated at the level of each member vortex. The implementation and theory of the fast multipole method is derived from [28].

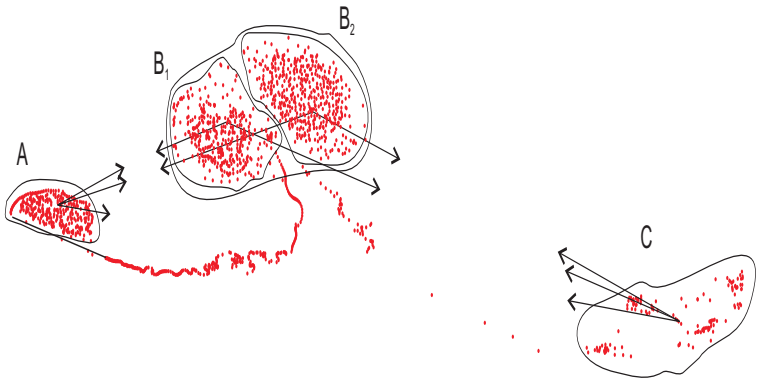

Figure 3.2. Vortices shed from an airfoil separated into domains A, B1, B2 and D.

Figure 3.3, compares the computation time to calculate interactions of vortices shed from a flat plate moving at an angle of attack $\pi/2$. The dots represent the direct $(O(N^2))$ method and the crosses are the results of the FMM method. The reduction in computation time obtained by implementing the FMM, is significant for large number of vortices.

3.1.4 Vortex Amalgamation

As described in Ansari [13], another way to reduce the computation time is to reduce the number of vortices. This is achieved by amalgamating two or more vortices, when they get very close. The criterion for vortex merging, proposed by Spallart [31], is used here,

$$\left| \frac{\Gamma_i \Gamma_j}{\Gamma_i + \Gamma_j} \right| \delta_{ij} < \epsilon, \tag{3.4}$$

where δ_{ij} is the distance between two vortices of circulation Γ_i and Γ_j. An ϵ-value of around $10^{-8}c$ is chosen for the numerical scheme.

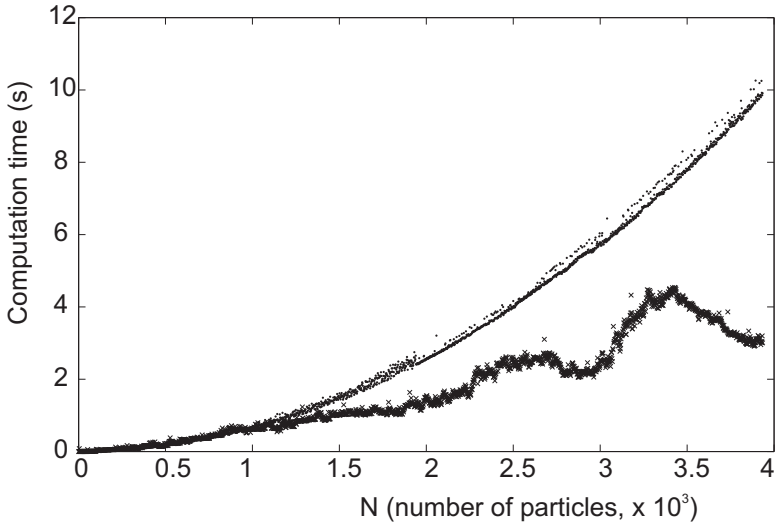

Figure 3.3. Computation time versus number of vortices without (dot) and with (cross) FMM.

3.1.5 Limiting the Vortex Strength

During extremely rapid maneuvers of the wing or while encountering the wake, the circulation of the newly shed vortices can be much higher (orders of 100) than the average circulation of the vortices. This could give rise to instability in the numerical scheme. So a saturation limit is fixed on the circulation of newly shed vortices (approximately 10 times the average vortex strength).

3.1.6 Placement of New Vortices

The scheme suggested by Ansari [13] is used to determine the position of newly shed vortices. According to Ansari, the new vortex is placed at one third of the distance of the previously shed vortex from the shedding edge:

$$\zeta_t = \zeta_{edge} + \frac{1}{3}(\zeta_{t-1} - \zeta_{edge}) \tag{3.5}$$

This has the advantage of taking into account the convection of the previously shed vortices.

During the initial time step, the position of the first shed vortices are determined from the local velocity at the shedding edges. If the complex velocity of the shedding edge is v_{edge}, then the position of the first vortex is given as,

$$\zeta_{\delta t} = \zeta_{edge} + \frac{1}{2}\overline{v}_{edge}\delta t \tag{3.6}$$

3.1.7 Forces

All forces are calculated in the translating coordinate frame ($\hat{\zeta}$). Equation 2.16 is discretized for numerical calculations. As explained in [13], because of the nature of the method used to

compute the forces, the unfiltered force results contain multiple spikes. These spikes are an effect of the wake interaction with the airfoil. Convection of the wake in one time step can sometimes switch the position of some of the wakes relative to the translating coordinate frame, which causes sudden changes in the impulse data series. On subsequent differentiation of the series to generate the forces, these changes are magnified resulting in large oscillations from the mean value. A simple butterworth filter is used to minimize these numerical errors.

Chapter 4

Clap and Fling

4.1 Model Development

Consider the doubly connected unbounded fluid domain $\tilde{\zeta}$ (see fig. 4.1) in which the two airfoils (representing the two wings) a_1 and a_3 are performing clap-and-fling. It is possible to obtain a conformal mapping to map the region ω^* between the two cylinders C_1 and C_3 (of radii r_1, r_3) to $\tilde{\zeta}$. The interior of cylinder C_1 corresponds to the interior of a_1 and the exterior of C_3 corresponds to the interior of a_3. The leading edges (C_{1le} and C_{3le}) correspond to the leading edges of a_1 and a_3 and lie on the same radial line. The trailing edges C_{1te} and C_{3te} have similar mappings. The strength of two vortices, shed simultaneously from the leading (or trailing) edges, would be equal and opposite due to symmetry. The corresponding location of the vortices in the ω^*-domain are w_1^* and w_3^*.

The line of symmetry L_2 bisects the $\tilde{\zeta}$-domain so that a_1 and all its shed vortices are to the left of L_2 and a_3 and all its vortices are to the right of L_2. Symmetry implies that L_2 is also a streamline, so that there is no flow across it. This assumption is not entirely valid during 'fling' since the trailing edge vortices, shed from each airfoil, merge and cancel. However except for that region L_2 can be treated as a streamline. Corresponding to L_2, the cylinder C_2 (with radius r_2) in ω^*, acts as a streamline. The only streamline in ω^* is the locus $|\omega| = \sqrt{r_1 r_3}$. Therefore,

$$r_2 = \sqrt{r_1 r_3}. \tag{4.1}$$

The two vortices w_1^* and w_3^* must also satisfy the condition $w_1^* \overline{w_3^*} = r_2^2$ to preserve the no flow through condition at C_2. Consider the domain ω which is bounded between C_1 and C_3. The conformal transformation maps this domain to the left half of the $\tilde{\zeta}$-domain (enclosing a_1 and its shed vortices). However this domain is unbounded. An outer boundary with one edge as L_2 and the rest of the vertices at $-\infty$ is defined such that the problem reduces to an airfoil in a bounded domain. The Schwarz-Christoffel mapping described in section 2.3 is then used for this domain.

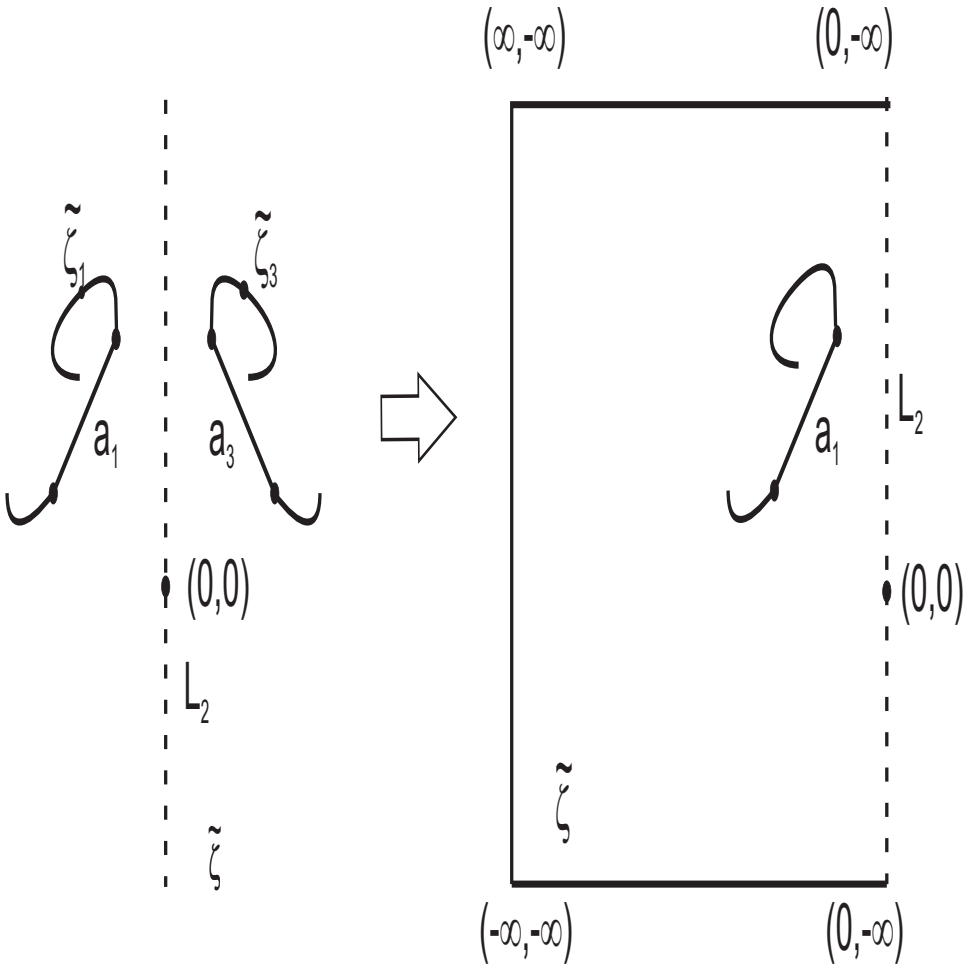

Figure 4.1. Simplification of symmetrical 2-airfoil problem to airfoil-wall problem

4.1.1 Implementation of S-C mapping

In figure 4.1 the wall is modeled as a rectangle with all vertices far away from the airfoil (approximately at a distance of 10c). This ensures that the artificially introduced outer boundaries do not affect the flow. The symmetry wall is chosen as the side of the rectangle to the right of the airfoil. The left airfoil is modeled as a flat plate (c = 0.05 m) with vertices at the leading and trailing edge. At the starting position, the airfoil is at a distance of 4c from the symmetry wall. This ensures that during fling the distance between the airfoil and the wall is ≈ 0.02c. An additional vertex is added to the wall at (0,0) (inertial $\tilde{\zeta}$-plane) to improve the numerical accuracy of the scheme.

The details of the motion for the left airfoil can be obtained from Miller and Peskin [21]. The kinematics of the left airfoil are shown in figure 4.2. The airfoil accelerates to a constant velocity during translation before clap and decelerates to a stop during clap. After fling, the airfoil again accelerates to a constant velocity. The velocity during acceleration and deceleration is given by,

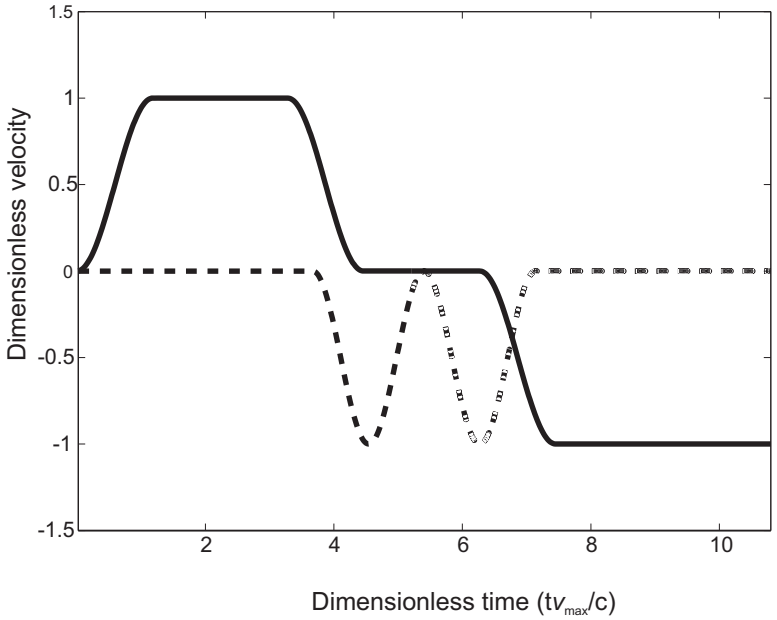

Figure 4.2. Dimensionless translational (v/v_{max}) (solid) and angular (ω/ω_{max}) (dashed) velocity of airfoil

$$v(\tau) = \frac{v_{max}}{2}\left\{1 + \cos\left\{\pi + \frac{\pi(\tau - \tau_{accel})}{\Delta\tau_{accel}}\right\}\right\}, \tag{4.2}$$

and,

$$\begin{aligned} v(\tau) &= v_{max} - \frac{v_{max}}{2}\left\{1 + \cos\left\{\pi + \frac{\pi(\tau - \tau_{decel})}{\Delta\tau_{decel}}\right\}\right\}, \\ \tau &= tv_{max}/c, \end{aligned} \tag{4.3}$$

where v_{max} is the maximum translational velocity, τ is the dimensionless time, τ_{accel} and τ_{decel} are the dimensionless time when acceleration and deceleration start, respectively, and $\Delta\tau_{accel}$, $\Delta\tau_{decel}$ are the dimensionless duration of acceleration and deceleration, respectively.

Similarly the angular velocity of the airfoil is given by,

$$\begin{aligned} \omega(\tau) &= \frac{w_{max}}{2}\left\{1 - \cos\left\{2\pi\frac{(\tau - \tau_{turn})}{\Delta\tau_{rot}}\right\}\right\}, \\ w_{max} &= \frac{2\Delta\theta v_{max}}{\Delta\tau_{rot}c}, \end{aligned} \tag{4.4}$$

where τ_{turn} is the dimensionless time when rotation starts, $\Delta\tau_{rot}$ is the dimensionless duration of rotation and $\Delta\theta$ is the total angle of rotation. During clap the airfoil rotates from an angle of 135 to 90 and during fling from 90 to 45.

The values of the variables are set such as to match the motion described in Miller and Peskin

[21] (see tab. 4.1.1).

Table 4.1. Table of 'clap-and-fling' motion variables

variable	clap	fling
τ_{accel}	0	6.27
τ_{decel}	3.28	-
$\Delta\tau_{accel}$	1.18	1.18
$\Delta\tau_{decel}$	1.18	-
τ_{turn}	3.67	5.4
$\Delta\tau_{rot}$	1.74	1.74
v_{max} (ms^{-1})	0.00375	0.00375
$\Delta\theta$ (rad)	$\pi/4$	$\pi/4$

4.2 Results

The streamline plots, at different time instances (i, ii, ..., viii), of the clap-and-fling results obtained by Miller and Peskin [21] (a) are compared with the simulation results (b) in Figs. 4.3 and 4.4. During the initial translation (i-iii) leading and trailing edge vortices form around the airfoil(s). The pair of vortices is shed during 'clap' (iv-v). New leading edge vortices are formed during 'fling' (vi-vii) while the strength of the new trailing edge vortices is very small. After fling, trailing edge vortices of higher strength start forming due to the translation of the airfoil (viii). These vortices are stronger in the numerical results (b-viii) as compared to the CFD results (a-viii).

4.2.1 Forces

Figures 4.5 and 4.6 compare the force coefficients obtained from the present approach (solid) with the CFD results of Miller and Peskin [21] at Re 8 (dash-dotted) and 128 (dashed). The overall trend of the results match the CFD results quite well. The initial peak in the lift coefficients correspond to the translational acceleration of the airfoil (i). During constant translation of the airfoil (i-iii), the lift coefficients remain steady and decrease during deceleration (iii). When the airfoils clap together (iv-v), there is a peak in the predicted lift coefficients caused by the jet of air pushed below by the closing airfoils. During fling (v-vi) lift coefficients increase due to the formation of large leading edge vortices. Another peak is observed in the lift coefficients when the airfoil translates after fling (vi-vii). The lift coefficients decrease subsequently due to the formation of trailing edge vortices. A similar behavior is seen for the predicted drag coefficients. Drag coefficients sharply increase during fling (v-vi) and a smaller peak is observed during subsequent translation. For $\tau > 0.9$, the distance between the airfoil and wall is greater than 3c and a coarser time step must be used (see Section 3.1.3). The force coefficients for this interval are therefore not reliable.

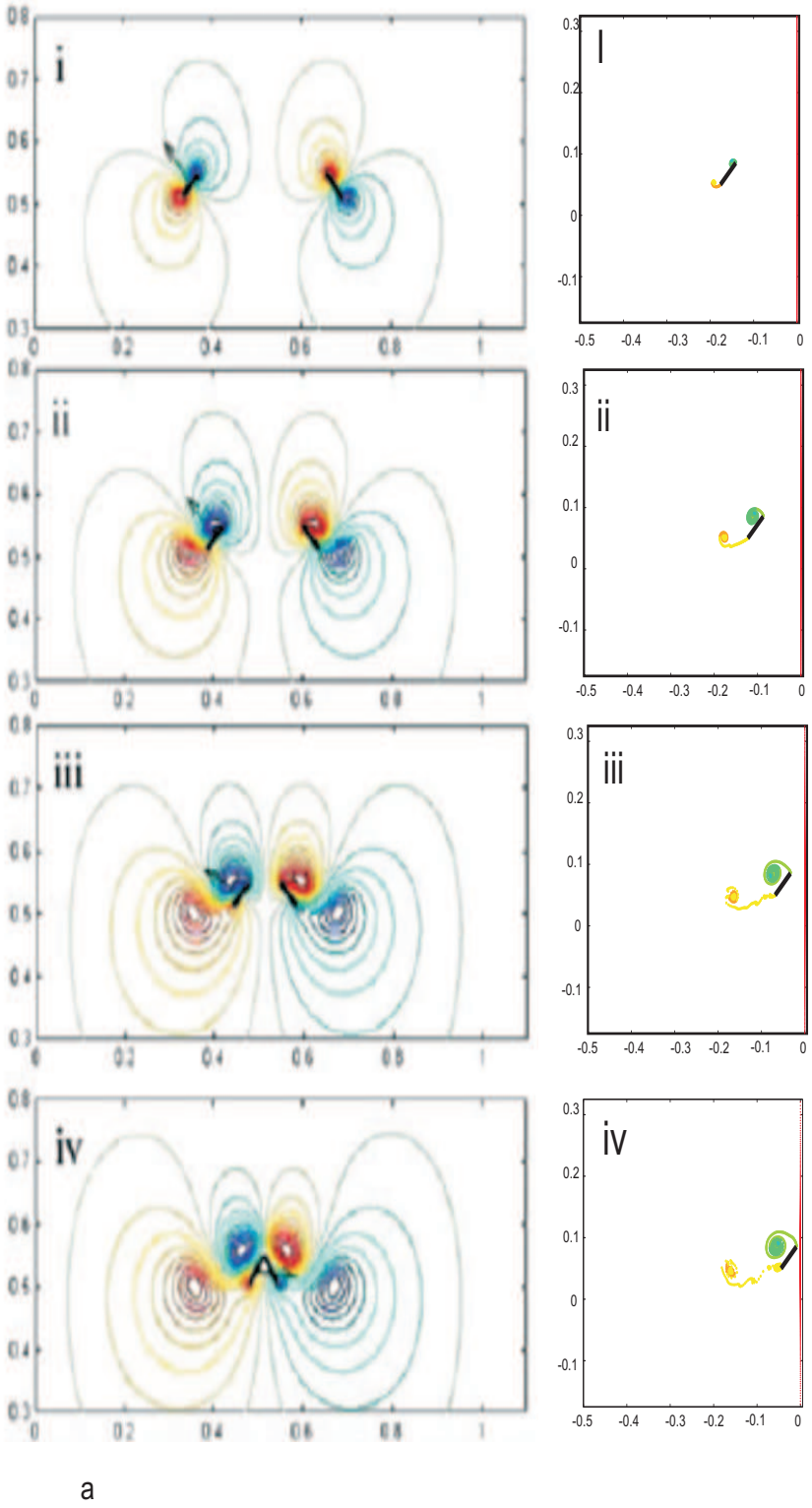

Figure 4.3. Clap-and-fling numerical results:
(a) Streamlines of fluid flow at Re 128 [21] and (b) vortex distribution for single airfoil against symmetry wall

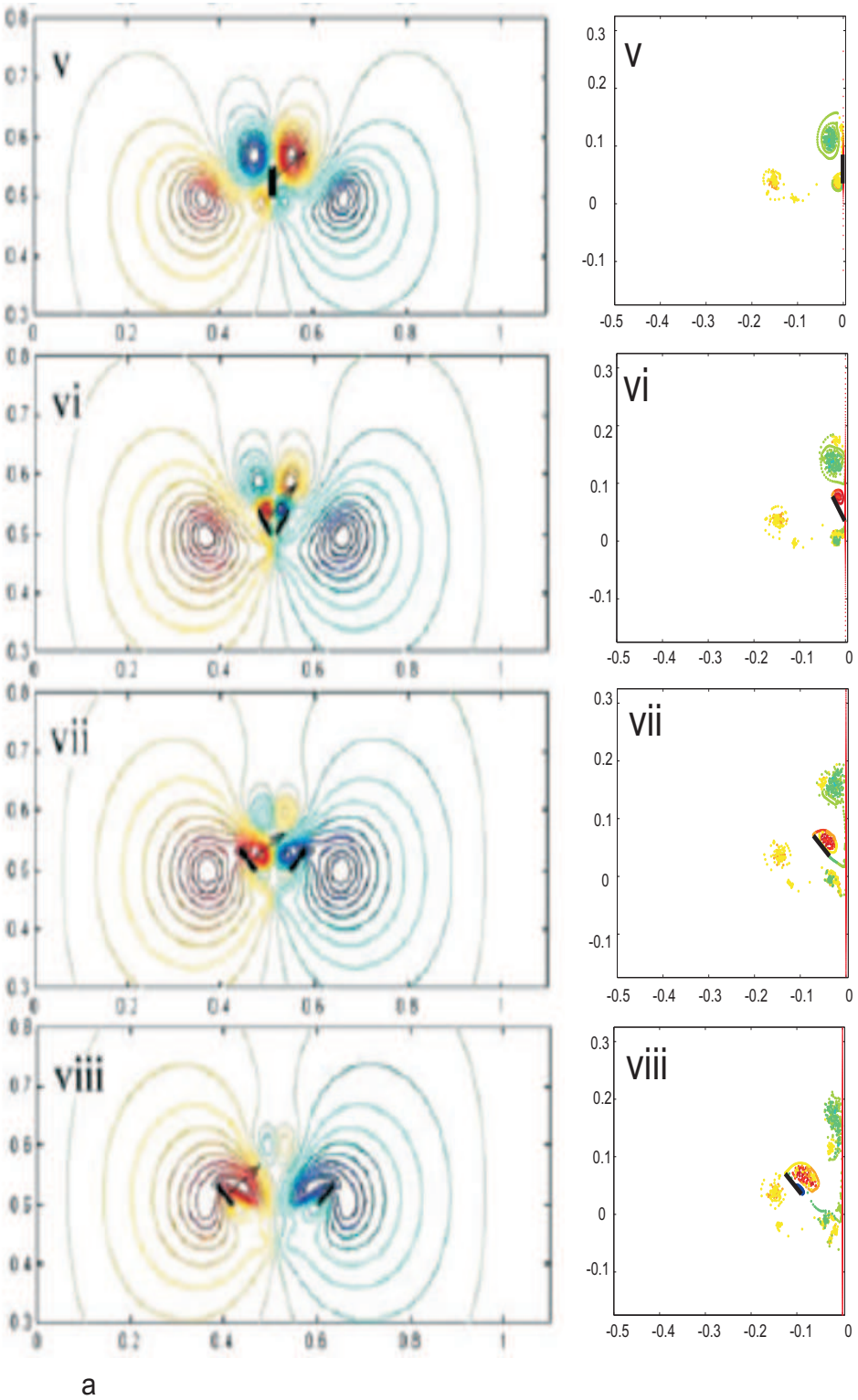

Figure 4.4. Clap-and-fling numerical results:
(a) Streamlines of fluid flow at Re 128 [21] and (b) vortex distribution for single airfoil against symmetry wall

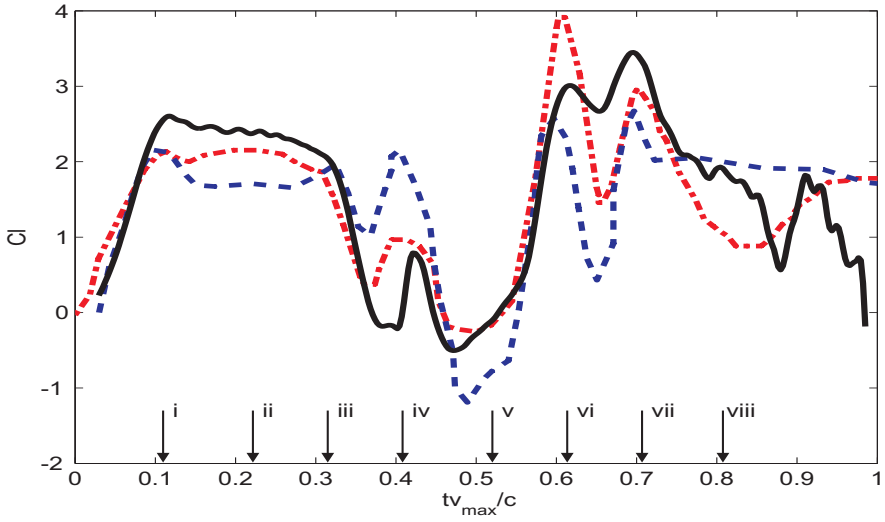

Figure 4.5. Lift coefficient versus stroke fraction for clap-and-fling
CFD results [21] at Re 8 (dash-dotted) and Re 128 (dashed) respectively and unsteady model (solid)

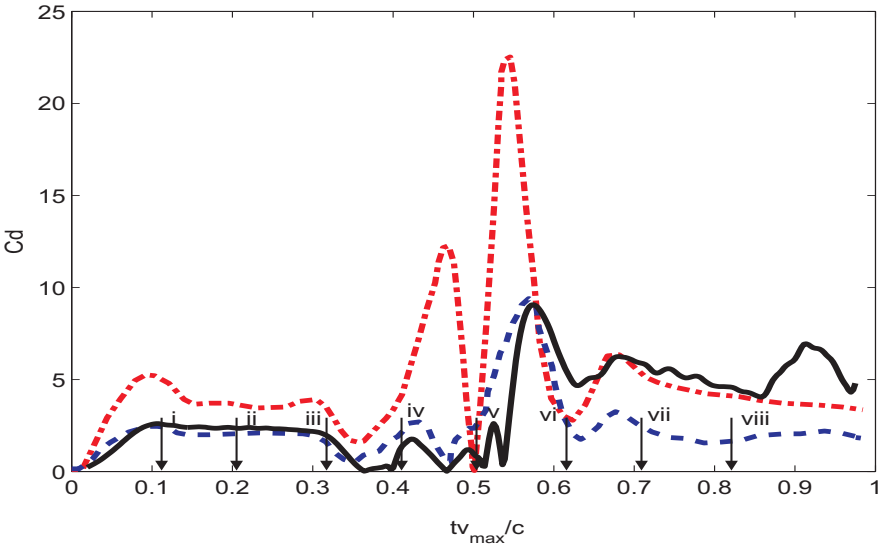

Figure 4.6. Drag coefficient versus stroke fraction for clap-and-fling
CFD results [21] at Re 8 (dash-dotted) and Re 128 (dashed) respectively and unsteady model (solid)

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The proposed 2-D, non-linear, unsteady aerodynamic model is validated both in terms of flow visualization and force results in comparison to the CFD results of Miller and Peskin [21] for the Weis-Fogh 'clap-and-fling' mechanism. The model predicts critical phenomena observed during 'clap-and-fling', such as the formation of large leading edge vortices and the absence of trailing edge vortices during fling. Overall, the vortex structures predicted by the model are very similar to those predicted by the CFD results. Miller and Peskin's CFD results give mean lift and drag coefficients over the entire stroke (for different Re) of 1.7 and 2 and 3 and 6 for Re 8 and 128, respectively. The mean lift and drag coefficients predicted by the present model are 1.57 and 3.46, independent of Re.

A limitation on the minimum permissible time step for numerical simulation arises mainly when the airfoil is at a distance greater than 3 chord lengths from the symmetry wall. At such distances, the radius of the inner cylinder (corresponding to the airfoil) shrinks to a very small value and the inherent error in DSCPACK is comparable to the distance between shed vortices. The accuracy of the numerical model, under such conditions is questionable. The numerical model can therefore be confidently used only for problems where the minimum distance between the airfoil and the boundary wall is less than 3c.

5.2 Future Work

The proposed model only estimates the forces on the airfoil and not the moments. Prediction of moments is important for understanding and better control of NAV's. A theoretical expression for moment estimates on the airfoil in the bounded domain needs to be derived to complement the force expressions given by Noca et. al. [30].

The current model is applicable only for an airfoil in a bounded fluid domain. If a conformal

mapping can be obtained which maps a two-airfoil unbounded fluid domain to an annular domain, then, following the outlines of the present work, a model to represent the flow over two airfoils can be developed without having to impose a symmetry condition on their motion. The model can then be used to study properties of other flapping mechanisms such as the phase relationship between the front and rear wings of dragonflies.

Using Ansari's [12] blade element approach, the present model can be extended to 3-D problem where the wing is divided into multiple radial chords and each chord is treated as an independent 2-D airfoil problem. The experimental 'clap-and-fling' results of Dickinson and Lehmann [22] can then be used to test the validity of the model.

Appendix A

Method of Images

The induced velocity due to a vortex (at ω_0) in an annular domain (of radii. $r_1, r_2,\ r_1 < r_2$) can be derived by adding infinite vortex images, to preserve the boundary conditions at the two cylinder surfaces. To preserve the no flow-through condition on the inner cylinder, using Milne's circle theorem, a vortex image denoted by w_I^1, of strength (-dΓ) is added at $\frac{r_1^2}{\overline{w_0}}$. Similarly for the outer cylinder, a vortex image, denoted by w_{II}^1 of strength (-dΓ) is added at $\frac{r_2^2}{\overline{w_0}}$. However w_I^1 disturbs the boundary condition at C_2, and a new vortex image w_{II}^2 of strength dΓ is added at $\frac{r_2^2}{\overline{w_I^1}}$ to restore the boundary condition. The same procedure applies for the inner cylinder C_1, and is carried out infinite times with each new vortex image added to preserve the boundary condition at one cylinder disturbing the boundary condition at the other cylinder. The final set of images we denote as ω_I^1, ω_I^2, ..., and $\omega_{II}^1, \omega_{II}^2$, ..., where

$$\begin{array}{ll} w_I^{1,-} = \frac{r_1^2}{\overline{w_0}} & w_{II}^{1,-} = \frac{r_2^2}{\overline{w_0}} \\ w_I^{2,+} = \frac{w_0}{q} & w_{II}^{2,+} = q w_0 \\ . & . \\ . & . \\ . & . \end{array} \tag{A.1}$$

where $q = \frac{r_2^2}{r_1^2}$.

Adding the contribution of each vortex to the flow, the resultant induced velocity is,

$$\overline{v}(\omega) = -\imath \frac{d\Gamma}{2\pi} \sum_{n=-\infty}^{\infty} \left[\frac{1}{\omega - \omega_0 q^n} - \frac{1}{\omega - \frac{r_1^2 q^n}{\overline{\omega_0}}} \right] \tag{A.2}$$

Appendix B

Dragonfly Flapping Simulation Using Ansari's Model

The one-winged flapping motion described in Wang [32], is modeled using the unsteady non-linear model of Ansari [12]. The equations of motion of the wing are as follows:

$$x(t) = \frac{A_0}{2} cos(2\pi ft) \tag{B.1}$$

and

$$\alpha(t) = \frac{\pi}{4} - \frac{\pi}{4} sin(2\pi ft + \phi_t) \tag{B.2}$$

where A_0 is the stroke amplitude, x(t) is the horizontal position of the wing center, $\alpha(t)$ is the angle of attack w.r.t the x-axis and ϕ_t sets the timing of rotation. In this case c = 1 cm, A_0=2.5 cm and $\phi_t = 0$.

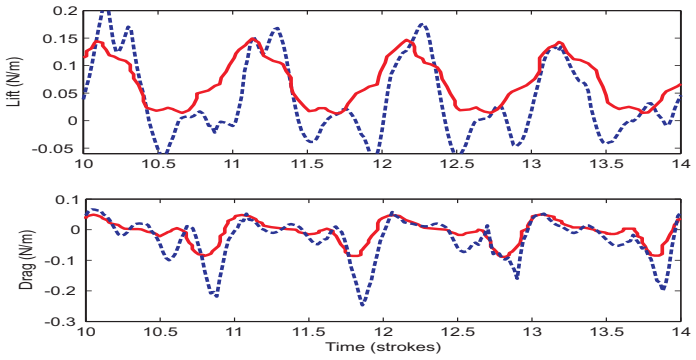

Figure B.1. Lift and drag per unit span for four strokes.
CFD results obtained for an elliptical airfoil by Wang [32] (solid) and results obtained with Ansari's model [12] (dashed) .

Lift and drag are compared to the CFD results of Wang in fig. B.1. There is good agreement between the force coefficients obtained from the unsteady non-linear model and the CFD simulation of Wang. On comparing the vorticity plots shown by Wang [32], see fig.B.2, the same phenomena is observed. During downstroke, the wing translation creates a pair of leading and trailing edge vortices with opposing circulation. At the end of downstroke the rotation of the wing converts them into a dipole, which then translates downwards,away from the wing. The momentum carried by this dipole generates lift on the wing and removes the previously shed vortices so that they do not interfere with the next cycle.

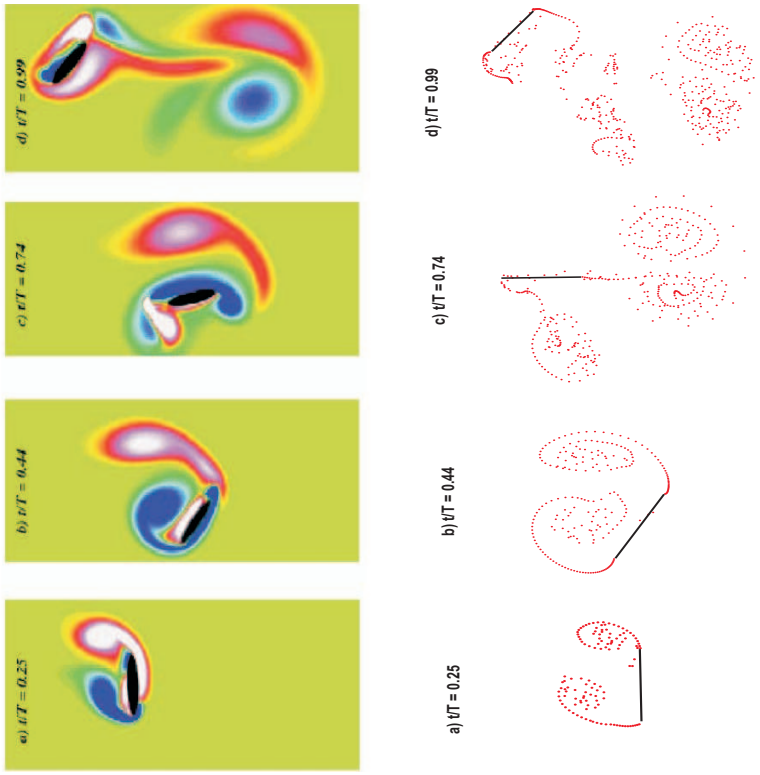

Figure B.2. Vorticity plot.
Comparison of flow visualization results from Wang [32] (left), with Ansari's model [12] .

Appendix C

Fling Results

Spedding and Maxworthy [20] have calculated the lift forces generated on a pair of rigid wings performing the 'fling' (opening of wings by rotation about a common trailing edge). The flow visualization results obtained confirm the development of large leading edge vortices (LEVs) about the two leading edges of the wings and the absence of any trailing edge wake.

A similar 'near fling' (a small gap exists between the two trailing edges) simulation is carried out using the unsteady model developed earlier. Initially the airfoil (modeled as a flat plate, c = 0.05 m) is very close to the boundary (symmetry) wall at a distance ≈ 0.014c. The airfoil then rotate opens at a constant angular velocity of 0.1 rad/s until it lies perpendicular to the wall. The lift coefficients obtained are compared with one of the experimental lift coefficients (exp. 3) obtained by Spedding and Maxworthy [20], and the theoretical results of Wu and Hu-Chen [33] in fig.C.1

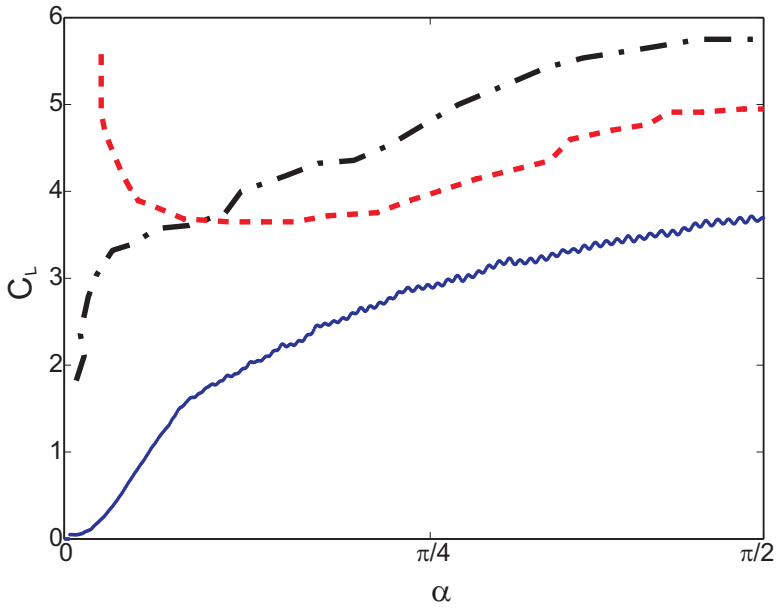

Figure C.1. Lift coefficient versus α during 'fling'
for experimental results (exp. 3) of Spedding and Maxworthy [20] (dash-dotted), theoretical results of Wu and Hu-Chen [33] (dashed), numerical results of unsteady model (solid).

Appendix D

Program Description

All programs are written in MATLAB. A brief description of individual routines is listed below:

Initialize Initializes the airfoil position and orientation at time t=0.

Start Carries out the first iteration.

trialalpha Carries out all further iterations and outputs force impulse series and flow results.

amal Merges two or more vortices depending upon their circulation and proximity.

euleritr Calculates convection velocity of vortices using either direct calculations or FMM.

fmm Calculates convection velocity due to free vortices using FMM. The **fmm** program uses several subroutines which are listed below:

- **crttree** Creates a tree mesh of all vortices using adaptive meshing
- **listtree** Calculates relationship between different cells of the tree based on their proximity.
- **isadj** Determines whether two cells in a three are adjacent.
- **veltree** Determines the convection velocities of all vortices.
- **Atree** Calulates a-coefficients of multipole-expansions of velocityfor each cell.
- **Btree** Calulates b-coefficients of multipole-expansions of velocity for each cell.
- **Wtree** Uses a and b coefficients to calulate velocities of member vortices in a cell.

genfile Writes flow and doundary coordinates to data files to be used later by **DSCPACK** for mapping.

g1 Calculates the velocity induced on the boundaries due to multiple shed vortices.

velcal Calulates the velocity of airfoil at any given time instant.

forcecal Calulates forces on airfoil from force impulse series.

sgdiff Differentiates force impulse data series.

The **DSCPACK** program runs in FORTRAN. It is compiled from two files **driver1.f** and **src.f**. A detailed description of the various sub-routines of **DSCPACK** is found in [19].

```
Initializealpha.m
%%%% Initializes the airfoil properties, position and
%%%% orientation at time t= 0

clear all;
clc;

syms zeta;
syms eta;
syms theta;

p = 1e3;
dt = 4e-1;
factor = 1;

%%%%%%%%%%%%%%%%%%%% aerofoil properties %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c     = 0.05;                                        % Chord length
sigma = 0;                                           % Thickness and camber
tau   = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%% defining starting orientation and velocities %%%%%%%%%%%%%%%%
Uinf = 0;
time = dt;

dln = -0.2e-8;
dhn = 0;
U0n = -dln' - i*dhn';
a = -c/2;
ln = -4.01*c;
hn = 0.5*c;
alphan = 3*pi/4;
dalphan = 0;
fid = fopen('eps.dat', 'wb');
eps=1e-8;
```

```
fprintf(fid,'%d \n',eps);
fclose(fid);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%% Define and write wall vertices %%%%%%%%%%%%%%%%%%%%%%%
xlimit = [-15.3*c 0*c]; %Artificial Boundaries placed far away from airfoil
ylimit = [-10*c 10*c];

Z0 = [ xlimit(2)        ylimit(2)   0.5
       xlimit(1)        ylimit(2)   0.5
       xlimit(1)        ylimit(1)   0.5
       xlimit(2)        ylimit(1)     0.5
       xlimit(2)        0             1  ];  % Additional vertex added
                              % at point nearest to trailing edge of
                              % airfoil for numerical accuracy of mapping


M = size(Z0,1);
fid = fopen('z0pts.dat', 'wb');
fprintf(fid,'%d \n',M);
fclose(fid);
save ('z0pts.dat','Z0','-ascii','-double', '-tabs','-append');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%% Derivatives of aer. prop. %%%%%%%%%%%%%%%%%%%%%%%%%
R = c/4;
e = (tau - i*sigma)/R;


A1n = -sigma*(dln.*cos(alphan)-dhn.*sin(alphan)) + (2*R-tau)*(dln.*sin(alphan)
      + dhn.*cos(alphan)) + a*(tau-2*R)*dalphan;

A2n = sigma*(dln.*cos(alphan)-dhn.*sin(alphan)) ...
     + tau*(dln.*sin(alphan)+dhn.*cos(alphan)) - a*tau*dalphan;

A3n = tau*(dln.*cos(alphan)-dhn.*sin(alphan)) ...
     - sigma*(dln.*sin(alphan)+dhn.*cos(alphan)) + a*sigma*dalphan;

A4n = -A3n;
```

```
A5n = -2*R*sigma*dalphan;

A6n = -A5n;

A7n = 4*R*(tau-R)*dalphan;

A8n = A6n/2;

A9n = -(tau^2 + sigma^2 + 2*R*tau)*dalphan/2;

A10n = (tau^2 + sigma^2 - 4*R*tau)*dalphan/2;

A11n = A5n;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Start.m

```
%%%%% Creates the first pair of shed vortices from  the
%%%%% airfoil

Initializealpha;   % Define airfoilproperties at start
k=1;
ratio = 0.00;      % Constant used during vortex merging
len = 200;         % Number of dicretized points used for airfoil
lenw = 1600;       % Higher number of dicretized points used for wall
                   % because of larger size

%%%%%%%%%%%%%%%%% Initialize first vortex pair posn %%%%%%%%%%%%%%%%%%%%%%%
qlvedge =
(dln-(a-c/2)*dalphan.*sin(alphan))-i*(dhn-(a-c/2)*dalphan.*cos(alphan));
qwkedge =
(dln-(a+c/2)*dalphan.*sin(alphan))-i*(dhn-(a+c/2)*dalphan.*cos(alphan));

psilvedge = (ln+i*hn) + (a-c/2)*exp(-i*alphan);
psiwkedge = (ln+i*hn) + (a+c/2)*exp(-i*alphan);

psilv = psilvedge - (1/2)*(qlvedge)*dt;
psiwk = psiwkedge - (1/2)*(qwkedge)*dt;
```

```
psihatlv = psilv - (ln(1)+i*hn(1));
psihatwk = psiwk - (ln(1)+i*hn(1));
psi = [psilv psiwk];

%%%%%%%%%%%%%%%%%%%%%% Calculating wall discretization in w-plane %%%%%%%%
thetaw = linspace(1e-3,2*pi+1e-3,lenw)';
Zwall = 1*exp(i*thetaw(1:end-1));
dthetaw = [diff(thetaw)]; thetaw = thetaw(1:end-1);
r2 = 1;
dZwall = r2*dthetaw;

%%%%%%%%%%%%%%%%%%%%%%Generate required files and run DSCPACK %%%%%%%%%%%%

genfile(len,thetaw,psiwkedge,psilvedge,psi,zeros(1,length(zetawall)));
dos('./driver.out');

%%%%%%%%%%%%%%%%%%%%%% Calculating airfoil discretization %%%%%%%%%%%%%%%%
load 'afedge.dat';
wlv = afedge(1,1)+i*afedge(1,2); thetalv = atan2(imag(wlv),real(wlv));
wwk = afedge(2,1)+i*afedge(2,2);
load theta1.dat; theta1 = theta1';
thetawk = theta1(1);
dtheta1 = 2*pi/len;
thetalv = thetalv+2*pi*(thetalv<0);
tlv = ceil(median(find(theta1-thetalv<0.5*dtheta1)));


%%%%%%%%%%%%%%%% Calculating wall discretization in inertial plane %%%%%%%%
load('z0wa.dat')
psiwall = (z0wa(:,1))+i*(z0wa(:,2)); psiwall = transpose(psiwall);
dpsiwall = [diff(psiwall) psiwall(1)-psiwall(end)];

for k=2:length(psiwall);
    dwall(k)=0.5*(dpsiwall(k-1)+dpsiwall(k));
end
dwall(1)=0.5*(dpsiwall(end)+dpsiwall(1));
dpsiwall = dwall;

load 'z1af.dat';
```

```
psibody = (z1af(:,1))+i*(z1af(:,2)); psibody = transpose(psibody);
load('wwpts.dat');
Z = (wwpts(:,1)+i*wwpts(:,2)); Z = transpose(Z);
Zlv = Z(1:length(psilv)); Zwk = Z(length(psilv)+1:end);

r1 = load('radius.dat');
q = r2^2/r1^2;

Zbody  = r1*exp(i*theta1);
dZbody = r1*dtheta1;
tlv = ceil(median(find(abs(psibody-psilvedge)<1e-3*c)));
twk = ceil(median(find(abs(psibody-psiwkedge)<1e-3*c)));
thetalv = theta1(tlv);
thetawk = theta1(twk);

%%%%%%%%%%%%%% Converting airfoil to Z-plane %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
psihatbody = psibody-(ln+i*hn);
psilocalbody = psihatbody*exp(i*alphan)-a;
thetab = (acos(real(psilocalbody)/2/R));
dthetab = abs(diff(thetab));
if (twk ==1)
     thetab = cumsum([0 dthetab]); thetab(len) = pi;
else thetab = cumsum([pi dthetab]);thetab(len) = 0;
end
% thetab = thetab(1:end-1);
in = find(thetab>2*pi); thetab(in)=thetab(in)-2*pi;
dthetab = [dthetab dthetab(end)]; dZb = R*dthetab;

%%%%%%%%%%%%%%%%% Calculating quasisteady airfoil vorticity %%%%%%%%%%%%%%%%
G0us = 2*pi*(2*R*(dln.*sin(alphan) + dhn.*cos(alphan))
+ dalphan*(tau^2/2 + sigma^2/2 - 2*R*(R+a)));

g0us = (1/R)*(-A1n*cos(theta)- (A2n+A7n/2)*cos(2*theta) + A3n*(sin(theta)) +
           (A4n - A5n/2 + A6n/2)*sin(2*theta) - A8n*sin(theta)*cos(2*theta) +
           A9n*sin(theta)*sin(2*theta) - A10n*cos(theta)*cos(2*theta) +
           A11n*cos(theta)*sin(2*theta) + G0us/2/pi)
  - 2*Uinf*(sin(theta-alphan));

g0nus = subs(g0us,thetab);
dG0us = g0nus.*abs(dZb);
```

```
g0n = dG0us./dZbody;   % Finding vorticity in annular frame

dG0 = dG0us;
G0n = G0us;
Gk = -G0n;
a11 = 0; a12 = 0; a21 = 0; a22 = 0; c1  =  -Gk; c2  = g0n(tlv)+Gk/2/pi/r1;
for n =-10:10
    Rwk = r1*exp(i*thetawk);
    a12 = a12 - (real((Zwk*q^n+Rwk)/(Zwk*q^n-Rwk)));
    a11 = a11 - (real((Zlv*q^n+Rwk)/(Zlv*q^n-Rwk)));
    Rlv = r1*exp(i*thetalv);
    a22 = a22 + (1/2/pi/r1)*(real((Zwk*q^n+Rlv)/(Zwk*q^n-Rlv)));
    a21 = a21 + (1/2/pi/r1)*(real((Zlv*q^n+Rlv)/(Zlv*q^n-Rlv)));
end
A = [a11 a12; a21 a22]; d = [c1;c2];
dG = A\d;

%%%%%%%%%%%%%%%%%%% Relaxation of Kutta Joukowski condition %%%%%%%%%%%%%
dG = min(abs(dG),limit).*sign(dG);
dGlv = dG(1);   dGwk = dG(2); dG = [dGlv dGwk];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

psi = [psilv psiwk];
psihat = [psihatlv psihatwk];

%%%%%%%%%%%%%%%%%% defining terms for loop %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
m=1;


%%%%%%%%%%%% Calculating bound vorticity on wall and a/f due to wake %%%%%%
g1af =  g1(0,G0n,r1,q,Zwk,Zlv,dGwk,dGlv,Zbody,psiwk,psilv,c); % for airfoil
dG1af = g1af.*dZbody;
g1wa =  g1(1,0*G0n,r2,q,Zwk,Zlv,dGwk,dGlv,Zwall,psiwk,psilv,c); % for wall
dG1wa = -g1wa.*dZwall';
v1wa = -dG1wa./abs(dpsiwall); v1o1 = 0;

%%%%%%%%%%%%%%%%% Vortex convection %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[qnew,qi,qd,qw] =
euleritr(0,4,psi,dt,dG,psibody,dG1af+dG0,psiwall,dG1wa,c,dln,dhn);
```

```
psitemp = psi + conj(mean(qnew,1))*dt;

psihattemp = psitemp-(ln(1)+i*hn(1));
psihatlvtemp = psihattemp(1:size(psihatlv,2));
psihatwktemp = psihattemp(1:size(psihatwk,2));

psihatwk = psihatwktemp;
psihatlv = psihatlvtemp;
psi      = psitemp;


%%%%%%%%%%%%%%%%%% Calculating force impulses %%%%%%%%%%%%%%%%%%%%%%%%%%
f1(:,m) = sum(psihatbody.*dG0,2);
f2(:,m) = sum(psihatbody.*dG1af,2);
f3(:,m) = sum(psihat.*dG,2);
m1(:,m) = sum(abs(psihatbody.^2).*dG0,2);
m2(:,m) = sum(abs(psihatbody.^2).*dG1af,2);
m3(:,m) = sum(abs(psihat.^2).*dG,2);
U0nm(:,m) = U0n;
G0nm(:,m) = G0n;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%% Time and m increment %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
time = time+dt;
m=2;
count =0;
psidata = single(psi(1,:));
psibodydata =  [psilvedge' psiwkedge'];
psiwalldata = single(psiwall);
dGdata = dG;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

trialalpha.m

```
ratio=0.05;                                          % Set amalgamation constant

V=0.00375;                                           %vmax
limit = 5e-5;                                        %Limit on circulation
Gout=0;
```

```
%%%%%%%%G%%%%%%%%%% start time loop %%%%%%%%%%%%%%%%%% %
while (time<10.8*c/V)

    tic
    %%%%%%%%%%%%%%% Calculate velocity and orientation of airfoil %%%%%%%%%
    [Vn,dalphan] = velcal(V,time,c,alphan);

    dhn = imag(Vn);
    dln = real(Vn)

    ln = ln + dln*dt/factor;
    hn = hn + dhn*dt/factor;
    alphan = alphan + dalphan*dt/factor;
    U0n = -dln - i*dhn;

    %%%%%%%%% Parameters used to calculate quasi-steady circulation %%%%%%%
    A1n = -sigma*(dln.*cos(alphan)-dhn.*sin(alphan)) + ...
          (2*R-tau)*(dln.*sin(alphan)+ dhn.*cos(alphan)) + a*(tau-2*R)*dalphan;
    A2n = sigma*(dln.*cos(alphan)-dhn.*sin(alphan)) ...
         + tau*(dln.*sin(alphan)+dhn.*cos(alphan)) - a*tau*dalphan;
    A3n = tau*(dln.*cos(alphan)-dhn.*sin(alphan)) ...
         - sigma*(dln.*sin(alphan)+dhn.*cos(alphan)) + a*sigma*dalphan;
    A4n = -A3n; A5n = -2*R*sigma*dalphan;
    A6n = -A5n; A7n = 4*R*(tau-R)*dalphan;
    A8n = A6n/2; A9n = -(tau^2 + sigma^2 + 2*R*tau)*dalphan/2;
    A10n = (tau^2 + sigma^2 - 4*R*tau)*dalphan/2; A11n = A5n;


    %%%%%%%%%%%%%%%% Amalgamate vortices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (mod(m,2) == 0)
        gc = 'lv';
        [dGlv,psilv] = amal(dGlv,psilv,ratio,c);
        gc = 'wk';
        [dGwk,psiwk] = amal(dGwk,psiwk,ratio,c);

    end
    dG = [dGlv dGwk];
    psi = [psilv psiwk];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%Position of new pair of vortices %%%%%%%%%%%%%%%%%%%%
qlvedge = (dln - (a-c/2)*dalphan.*sin(alphan)) ...
        - i*(dhn-(a-c/2)*dalphan.*cos(alphan));
qwkedge = (dln - (a+c/2)*dalphan.*sin(alphan)) ...
        - i*(dhn - (a+c/2)*dalphan.*cos(alphan));

psilvedge = (ln+i*hn) + (a-c/2)*exp(-i*alphan);
psiwkedge = (ln+i*hn) + (a+c/2)*exp(-i*alphan);

if (~isempty(psilv))
    psilvnew = psilvedge + (1/3)*(-psilvedge + psilv(end));
else psilvnew = psilvedge - (1/2)*(qlvedge)*dt/factor;
end
if(~isempty(psiwk))
    psiwknew = psiwkedge + (1/3)*(-psiwkedge + psiwk(end));
else psiwknew = psiwkedge - (1/2)*(qwkedge)*dt/factor;
end

psilv = [psilv psilvnew]; psiwk = [psiwk psiwknew];

%%%%%%%%%%%%%%%%%%%%Map to annular domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
psi = [psilv psiwk];
genfile(len,thetaw,psiwkedge,psilvedge,psi,Zwall);
dos('./driver.out');                          % Run the program DSCPACK

load('radius.dat');
r1 = radius;
r2 = 1*(r1/radius);
q = radius^(-4);

%%%%%%%%%%%%%% Calculating airfoil discretizations in w-plane %%%%%%%%%
load 'afedge.dat';
wlv = (r1/radius)*(afedge(1,1)+i*afedge(1,2));
wwk = (r1/radius)*(afedge(2,1)+i*afedge(2,2));
load theta1.dat; theta1 = theta1';
thetalv = theta1(1); tlv = 1;
dtheta1 = 2*pi/len;
thetalv = thetalv+2*pi*(thetalv<0);
```

```
load('z1af.dat');
psibody = (z1af(:,1))+i*(z1af(:,2)); psibody = transpose(psibody);
twk = floor(median(find(abs(psibody-psiwkedge)<1e-3*c)));
thetawk = theta1(twk);

%%%%%%%%%%%%%%%%%%%%%% Calculating wall discretization %%%%%%%%%%%%%%%
load('z0wa.dat');
psiwall = (z0wa(:,1))+i*(z0wa(:,2)); psiwall = transpose(psiwall);

dpsiwall = [diff(psiwall) psiwall(1)-psiwall(end)];
nwall = -i*dpsiwall./abs(dpsiwall);

%%%%%%%%%%%%%%%%%%%% Determine mapped vortices in w-plane, remove %%%%%%
%%%%%%%%%%%%%%%%%%%% unsuccesfull mappings %%%%%%%%%%%%%%%%%%%%%%%%%%%%

load('wwpts.dat');
Z = (r1/radius)*(wwpts(:,1)+i*wwpts(:,2)); Z = transpose(Z);
nlv = length(psilv);

Zlv = Z(1:nlv);
in = find(abs(Zlv(1:end-1))==0 | (abs(Zlv(1:end-1))-r1)/r1 <= 0.01);

Zlv(in) = []; psilv(in) = []; dGlv(in) =[];
Zwk = Z(nlv+1:end);
in = find(abs(Zwk(1:end-1))==0 | (abs(Zwk(1:end-1))-r1)/r1 <= 0.01);
Zwk(in) = []; psiwk(in) = []; dGwk(in) =[];


Zlvold = Zlv(1:end-1); Zlvnew = Zlv(end);
Zwkold = Zwk(1:end-1); Zwknew = Zwk(end);

Zbody  = r1*exp(i*theta1);
dZbody = r1*dtheta1;

%%%%%%%%%%%%% Converting airfoil discretization to Z-plane %%%%%%%%%%%
psihatbody = psibody-(ln+i*hn);
psilocalbody = psihatbody*exp(i*alphan)-a;
thetab = (acos(real(psilocalbody)/2/R));
dthetab = abs(diff(thetab)); thetab = cumsum([real(thetab(1)) dthetab]);
```

```
in = find(thetab>2*pi); thetab(in)=thetab(in)-2*pi;
dthetab = [dthetab dthetab(end)]; dZb = R*dthetab;
for k=2:length(dZb);
    dZbt(k)=0.5*(dZb(k-1)+dZb(k));
end
dZbt(1)=0.5*(dZb(end)+dZb(1));
dZb = dZbt;

%%%%%%%% Calculating quasi-steady airfoil vorticity and circulation %%%

G0us = 2*pi*(2*R*(dln.*sin(alphan) + dhn.*cos(alphan))
       + dalphan*(tau^2/2 + sigma^2/2 - 2*R*(R+a)));

g0us = (1/R)*(-A1n*cos(theta)- (A2n+A7n/2)*cos(2*theta)
              +A3n*(sin(theta)) + (A4n - A5n/2 + A6n/2)*sin(2*theta)
              -A8n*sin(theta)*cos(2*theta) + A9n*sin(theta)*sin(2*theta)
              -A10n*cos(theta)*cos(2*theta) + A11n*cos(theta)*sin(2*theta))
              - 2*Uinf*(sin(theta-alphan));

g0nus = subs(g0us,thetab);                  % vorticity in Z-plane
dG0us = g0nus.*abs(dZb);
g0n = dG0us./dZbody+G0us/2/pi/r1;           % vorticity in w-plane

dG0 = dG0us+G0us*dZbody/2/pi/r1;
G0n = G0us;

%%%%%%%%%%%%%%%%%%% Calculating circulation of new vortices %%%%%%%%%%
q = r2^4/r1^4;
Rwk = wwk; Rlv = wlv;
Gk = -G0n - (Gout);
a11 = 0; a12 = 0; a21 = 0; a22 = 0;
c1  =  -Gk/2/pi/r1; c2  = g0n(tlv)+Gk/2/pi/r1;

a12 = a12 + g1(0,0,r1,q,Zwknew,[],1,0,Rwk,psiwknew,[],c);
a11 = a11 + g1(0,0,r1,q,[],Zlvnew,0,1,Rwk,[],psilvnew,c);
c1  = c1  - g1(0,0,r1,q,Zwkold,Zlvold,dGwk,dGlv,
               Rwk,psiwk(1:end-1),psilv(1:end-1),c);

a22 = a22 - g1(0,0,r1,q,Zwknew,[],1,0,Rlv,psiwknew,[],c);
```

```
a21 = a21 - g1(0,0,r1,q,[],Zlvnew,0,1,Rlv,[],psilvnew,c);
c2  = c2  + g1(0,0,r1,q,Zwkold,Zlvold,dGwk,dGlv,
               Rlv,psiwk(1:end-1),psilv(1:end-1),c);

A = [a11 a12; a21 a22];
d = [c1;c2] ;

if (abs(Zwknew) == 0 || abs(Zlvnew)==0) %If S-C mapping is unsuccesfull
    dGlvnew = []
    dGwknew = []
    Zlvnew =[];
    Zwknew =[];
    psilv(end) = [];
    psiwk(end) = [];

else dGnew = A\d;
    %%%%%%%%%%%%%%%%%%% Relaxation of Kutta Joukowski condition %%%%%%%
    dGnew = min(abs(dGnew),limit).*sign(dGnew);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    dGlvnew = real(dGnew(1));
    dGwknew = real(dGnew(2));
    dGlvnew = (dGlvnew);
    dGwknew = (dGwknew);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

%%%%%%%%%%%%%%%%%  Adding new vortices to array %%%%%%%%%%%%%%%%%%%%%%%
Zwk = [Zwkold Zwknew];
Zlv = [Zlvold Zlvnew];
dGwk = [dGwk dGwknew];
dGlv = [dGlv dGlvnew];

%%%%%%%%% Calculating bound vorticity on wall and a/f due to wake %%%%%

g1af  =  g1(0,-Gk,r1,q,Zwk,Zlv,dGwk,dGlv,Zbody,psiwk,psilv,c); % for af
dG1af = g1af.*dZbody;
dG2af = g2af.*dZbody;

g1wa  =  g1(1,G0n+Gk,r2,q,Zwk,Zlv,dGwk,dGlv,r2*Zwall,psiwk,psilv,c); %wall
```

```
dG1wa = -g1wa.*dZwall'*r2;
v1wa  = -dG1wa./abs(dpsiwall);

%%%%%%%%%%%%%%%% Vortex Convection %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dG = [dGlv dGwk]; save('dG.mat','dG')
psi = [psilv psiwk];
if (length(dG)<100)
     toggle = 0;           %decides whether to use FMM or direct method
else toggle = 1;
end

[qnew,qi,qd,qw] = euleritr(toggle,10,psi,dt/factor,dG,psibody, ...
                           dG1af+dG0,psiwall,dG1wa,c,dln,dhn);

psitemp = psi + conj(mean(qnew,1))*dt/factor;            % Convect wake

psihattemp = psitemp-(ln(1)+i*hn(1));
psihatlvtemp = psihattemp(1:size(dGlv,2));
psihatwktemp = psihattemp(size(dGlv,2)+1:end);

psihatwk = psihatwktemp;
psihatlv = psihatlvtemp;
psihat = [psihatlv psihatwk];
psi      = psitemp;
psilv = psi(1:size(dGlv,2)); psiwk = psi(size(dGlv,2)+1:end);
psihatwall = psiwall -(ln+i*hn);
dhatwall = dpsiwall;  nhatwall = nwall;

indlv = find(real(psilv)>=-0.00*c); indwk = find(real(psiwk)>=-0.00*c;
Gout = Gout + sum(dGlv(indlv))+sum(dGwk(indwk))

psilv(indlv) = []; dGlv(indlv) = [];            % Remove vortices which
psiwk(indwk) = []; dGwk(indwk) = [];            % have crosed wall

psihatlv = psilv-(ln+i*hn);
psihatwk = psiwk-(ln+i*hn);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%Computing force impulses %%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    if (ceil(m)-m<1/factor)
        m = ceil(m)
        f1(m) = sum(psihatbody.*dG0);
        f2(m) = sum(psihatbody.*dG1af);
        f3(m) = sum(psihat.*dG);
        f4(m) = sum(psihatwall.*dG1wa);
        f5(m) = 0.5*sum((v1wa.^2).*nhatwall.*abs(dhatwall));

        U0nm(m) = U0n;
        G0nm(m) = G0n;
        titer(m-1) = toc;
    end

    %%%%%%%%%%%%%%% Collecting flow data periodically %%%%%%%%%%%%%%%%%%%%%

    if (mod(m,2) == 1)
        psidata = [psidata NaN single(psi)];
        psibodydata = [psibodydata;
                       psilvedge psiwkedge];
        psiwalldata = [psiwalldata NaN psiwall];
        dGdata = [dGdata NaN dG];
    end

    %%%%%%%%%%%%%%%%%%%% Change time step, advance time %%%%%%%%%%%%%%%%%%
    if (abs(ln)<c)
        factor = 2;
    end
    factor=3;                                  %  Changes size of time step
    m=m+1/factor;
    time =time+dt/factor;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end

save 'Results.mat';



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

amal.m

```
% This function merges two or more vortices if they come too close to each
% other

function [dGxx,psihatxx] = amal(dGxx,psihatxx,ratio,c)

sizev = size(dGxx,2);
k=1;
while k<=max(0,sizev-5)
    u=k+1;
    index = [];
    while u <= sizev
        rvortex = abs(psihatxx(k)-psihatxx(u));
        if rvortex<1*c && 0.1<dGxx(k)/dGxx(u)<10 &&
           (sign(dGxx(u))==sign(dGxx(k))) && dGxx(k)<2e-5
           ((abs(dGxx(k)*dGxx(u)/(dGxx(k)+dGxx(u)))*rvortex < ratio*1e-7*c) )
            index = [index u];
        end
        u=u+1;
    end
    psihatxx(k) =
    sum(psihatxx([k index]).*dGxx([k index]))/sum(dGxx([k index]));
    dGxx(k) = sum(dGxx([k index]));
    dGxx(index)=[];
    psihatxx(index)=[];
    sizev = sizev-length(index);
    k=k+1;
end

%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

eulerutr.m

```
% Calculates convection velocity
% toggle determines whether FMM or direct method is to be used
% k1 determines number of sub-time steps used
```

```
% psi, dG are the coordinates and circulation of the free vortices
% psibody, dGaf are the coordinates and circulation of the bound vortices
% on the airfoil
% psiwall, dG1wa are the coordinates and circulation of the bound vortices
% on the wall

function [qnew,q,qd,qw] = euleritr(toggle,k1,psi,dt,dG,psibody,
                                   dGaf,psiwall,dG1wa,c,dln,dhn)

q = zeros(k1,size(dG,2));
qd = zeros(k1,size(dG,2));
qw = qd;
psitemp = psi;
rc = 0.05*c;                          % Finite core radius for free vortices
rcore =0.05*c;                        %               airfoil bound vortices
rcwall= 0.05*c;                       %                  wall bound vortices
for count=1:k1
    %%%%%%%%%%%%%% For free vortices only %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (toggle==0)
        %%%%%%%%%%%%%%%%%%%% Use Direct method %%%%%%%%%%%%%%%%%%%%%%%%%%%%
        psitemph = [(ones(size(psitemp,2),1))*psitemp];
        psitempv = transpose(psitemph);
        rad = psitemph-psitempv + 1e8*eye(size(psitemph));
        dGv = [(ones(size(dG,2),1))*dG]';

        scalesign = 1;
        scalex = abs(rad).^2;
        scaley = sqrt(scalex.^2+(rc*scalesign).^4);
        scale = 1*scalex./scaley;

        x = -(i/2/pi)*dGv.*scale./(rad);
        q(count,:) = sum(x,1);
        %%%%%%%%%%%%%%%%%%%%%%%%%% Use FMM %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    elseif (toggle == 1)
        q(count,:) = fmm(psitemp,dG,rc);
    end

    %%%%%%%%%%%%%%%%%%%% For bound vortices on airfoil %%%%%%%%%%%%%%%%%%%%

    psitemph = [(ones(size(psibody,2),1))*psitemp];
```

```
    psibodyv = transpose(psibody)*(ones(1,size(psitemp,2)));
    radbody = [psitemph - psibodyv] + (dln+i*dhn)*dt*(count-1)/k1;;
    scalex1 = abs(radbody).^2;
    scaley1 = sqrt(scalex1.^2+rcore.^4);
    scale1 = scalex1./scaley1;
    dGbodyv = transpose(dGaf)*(ones(1,size(psitemp,2)));
    x1 = -(i/2/pi)*(dGbodyv).*scale1./(radbody);
    qd(count,:) = sum(x1,1);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%% For bound vortices on wall %%%%%%%%%%%%%%%%%%%%
    psitemph = [(ones(size(psiwall,2),1))*psitemp];
    psiwallv = transpose(psiwall)*(ones(1,size(psitemp,2)));
    radwall = [psitemph - psiwallv];
    scalex1 = abs(radwall).^2;
    scaley1 = sqrt(scalex1.^2+rcwall.^4);
    scale1 = scalex1./scaley1;
    dGwav = transpose(dG1wa)*(ones(1,size(psitemp,2)));
    x1 = -(i/2/pi)*(dGwav).*scale1./(radwall);
    qw(count,:) = sum(x1,1);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    psitemp = psitemp + conj(q(count,:)+ qd(count,:)+ qw(count,:))*dt/k1;
end

qnew = q+qd+qw;

%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

fmm.m

```
% The main program to implement the Fast Multipole method
% outputs q as the convection velocity
% psi, dG are position and circulation of free vortices
% c is chord length of airfoil

function [q] = fmm(psi,dG,c)

global index ;
```

```
rangex = max(real(psi))-min(real(psi));          % Determine size of domain
rangey = max(imag(psi))-min(imag(psi));
s = max(rangex,rangey);


%%%%%%% Initialize first level of tree which includes all vortices %%%%%%%%
tree(1).box.data=psi;
tree(1).box.dG = dG;
tree(1).box.size = s;
tree(1).box.centre = min(real(psi))+s/2 + i*(min(imag(psi)) + s/2);
tree(1).box.parent =[];
tree(1).box.child =[];
tree(1).box.list = [];

%%%%%%%%%%%%%%%% Recursively creates a tree of depth 50 %%%%%%%%%%%%%%%%%%%
index = [1 1 1];
[tree] = crttree(1,1,1,tree,50);

for lv = 1:size(index,1)
    level = index(lv,1);
    k = index(lv,2);
    l = index(lv,3);
    for clv = 1:size(index,1)
        clevel = index(clv,1);
        ck = index(clv,2);
        cl = index(clv,3);
        if (level>1)
            pk = tree(level).box(k,l).parent(2);
            pl = tree(level).box(k,l).parent(3);
            if(level-1 == clevel)
                if ~(pk+1<ck || pk-1>ck || pl+1<cl || pl-1>cl)
                    [tree] = listtree(level,k,l,clevel,ck,cl,tree);
                end
            elseif (level-1 < clevel)
                pl_right = 2^(clevel-level+1)*(pl+1);
                pl_left  = 2^(clevel-level+1)*(pl-2)+1;
                pk_top   = 2^(clevel-level+1)*(pk+1);
                pk_bott  = 2^(clevel-level+1)*(pk-2)+1;
                if ~(pk_top+1<ck || pk_bott-1>ck || pl_right+1<cl
                     || pl_left-1>cl)
```

```
                    [tree] = listtree(level,k,l,clevel,ck,cl,tree);
                end
            elseif (level-1 > clevel)
                cl_right = 2^(level-1-clevel)*(cl+1);
                cl_left  = 2^(level-1-clevel)*(cl-2)+1;
                ck_top   = 2^(level-1-clevel)*(ck+1);
                ck_bott  = 2^(level-1-clevel)*(ck-2)+1;
                if ~(ck_top+1<pk || ck_bott-1>pk || cl_right+1<pl
                   || cl_left-1>pl)
                    [tree] = listtree(level,k,l,clevel,ck,cl,tree);
                end
            end
        else
            [tree] = listtree(level,k,l,clevel,ck,cl,tree);
        end
    end
end

%%%%%%%%%%%%%%%%Calculate A B coeff. and velocities for tree %%%%%%%%%%%%%

[tree] = Atree(1,1,1,tree,5);
[tree] = Btree(1,1,1,tree,5);
[tree] = Wtree(1,1,1,tree,c);
q      = veltree(tree);               % Computes velocity

%%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

crttree.m

```
% Divides a parent cell into 4 child cells using an adaptive mesh
% Creates the final tree of vortex cells using recursive algorithm
% k,l are the coordinates of parent cell
% Nf is depth of parent cell in the tree

function [tree] = crttree(level,k,l,tree,Nf)
global index;
ibox = tree(level).box(k,l);
N = length(ibox.data);                % Defines  depth of tree
s = ibox.size;                        % Defines size of an indivudual cell
ichild11.data = [];                   % child cell of a parent cell
```

```
ichild11.dG = [];
ichild11.parent = [];
ichild11.child =[];
ichild11.list = [];
ichild12 = ichild11;
ichild21 = ichild11;
ichild22 = ichild11;

if N>Nf                          % Divides a parent cell into 4 child cells
    for l1 = 1:N
        dr = ibox.data(l1) - ibox.centre;
        if (real(dr)<=0 && imag(dr)<=0)
            ichild11.data = [ichild11.data ibox.data(l1)];
            ichild11.dG   = [ichild11.dG   ibox.dG(l1)  ];
        elseif (real(dr)>0 && imag(dr)<=0)
            ichild12.data = [ichild12.data ibox.data(l1)];
            ichild12.dG   = [ichild12.dG   ibox.dG(l1)  ];
        elseif (real(dr)>0 && imag(dr)>0)
            ichild22.data = [ichild22.data ibox.data(l1)];
            ichild22.dG   = [ichild22.dG   ibox.dG(l1)  ];
        elseif (real(dr)<=0 && imag(dr)>0)
            ichild21.data = [ichild21.data ibox.data(l1)];
            ichild21.dG   = [ichild21.dG   ibox.dG(l1)  ];
        end
    end
end

ichild11.size = s/2;
ichild12.size = s/2;
ichild22.size = s/2;
ichild21.size = s/2;

ichild11.centre = ibox.centre +(-1-i)*s/4;
ichild12.centre = ibox.centre + (1-i)*s/4;
ichild22.centre = ibox.centre + (1+i)*s/4;
ichild21.centre = ibox.centre +(-1+i)*s/4;

%%%%%%%% Assign parent cell vortices to each child cell %%%%%%%%%%%%%%%%%%%%
if(~isempty(ichild11.data))
    tree(level+1).box(2*k-1,2*l-1) = ichild11;
```

```
    tree(level+1).box(2*k-1,2*l-1).parent = [level k l];
    tree(level+1).box(2*k-1,2*l-1).child = [];
    tree(level+1).box(2*k-1,2*l-1).list = [];
    tree(level).box(k,l).child = [tree(level).box(k,l).child;
                                              level+1 2*k-1 2*l-1];
    [tree]= crttree(level+1,2*k-1,2*l-1,tree,Nf);
    index = [index; level+1 2*k-1 2*l-1];
end

if(~isempty(ichild12.data))
    tree(level+1).box(2*k-1,2*l) = ichild12;
    tree(level+1).box(2*k-1,2*l).parent = [level k l];
    tree(level+1).box(2*k-1,2*l).child = [];
    tree(level+1).box(2*k-1,2*l).list = [];
    tree(level).box(k,l).child = [tree(level).box(k,l).child;
                                              level+1 2*k-1 2*l];
    [tree]= crttree(level+1,2*k-1,2*l,tree,Nf);
    index = [index; level+1 2*k-1 2*l];
end

if(~isempty(ichild21.data))
    tree(level+1).box(2*k,2*l-1) = ichild21;
    tree(level+1).box(2*k,2*l-1).parent = [level k l];
    tree(level+1).box(2*k,2*l-1).child = [];
    tree(level+1).box(2*k,2*l-1).list = [];
    tree(level).box(k,l).child = [tree(level).box(k,l).child;
                                              level+1 2*k 2*l-1];
    [tree]= crttree(level+1,2*k,2*l-1,tree,Nf);
    index = [index; level+1 2*k 2*l-1];
end

if(~isempty(ichild22.data))
    tree(level+1).box(2*k,2*l) = ichild22;
    tree(level+1).box(2*k,2*l).parent = [level k l];
    tree(level+1).box(2*k,2*l).child = [];
    tree(level+1).box(2*k,2*l).list = [];
    tree(level).box(k,l).child = [tree(level).box(k,l).child;
                                              level+1 2*k 2*l];
    [tree]= crttree(level+1,2*k,2*l,tree,Nf);
    index = [index; level+1 2*k 2*l];
```

```
end

%%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

listtree.m

```
% Defines relationship between any two cells in the tree
% (k,l), (ck,cl) are coordinates of the two cells
% level, clevel is the depth of the two cells in the tree

function [tree] = listtree(level,k,l,clevel,ck,cl,tree)

err = 1e-5;                         % error defined to search cell adjacency

ibox = tree(level).box(k,l);
iclient = tree(clevel).box(ck,cl);

bc = ibox.centre;
bs = ibox.size;
cc = iclient.centre;
cs = iclient.size;
dcx = abs(real(bc-cc));
dcy = abs(imag(bc-cc));

if (~isempty(ibox.parent))
    iparent = tree(level-1).box(ibox.parent(2),ibox.parent(3));
    pc = iparent.centre;
end


    if (~isempty(iclient.parent))
        iparentc = tree(clevel-1).box(iclient.parent(2),iclient.parent(3));
    end

    if (dcx<=0.5*(bs+cs)+err && dcy<=0.5*(bs+cs)+err)
        if (isempty(ibox.child) && isempty(iclient.child))
            ibox.list = [ibox.list; clevel ck cl 1];
        end

    elseif (~isempty(ibox.parent) && ~isempty(iclient.parent)
```

```
             && level == clevel  && isadj(iparent,iparentc,err))

            ibox.list = [ibox.list; clevel ck cl 2];

    elseif (~isempty(iclient.parent) && level < clevel
             && isadj(iparentc,ibox,err))
            if(isempty(ibox.child))
                ibox.list = [ibox.list; clevel ck cl 3];
            end

    elseif (~isempty(ibox.parent) && level > clevel
             && isadj(iparent,iclient,err))
            if(isempty(iclient.child))
                ibox.list = [ibox.list; clevel ck cl 4];
            end
    end

tree(level).box(k,l).list = ibox.list;

%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

isadj.m

```
% Returns a true value if two cells (ibox and iclient) are adjacent
% to each other within a permissible error (err)

function [b] = isadj(ibox,iclient,err)

bc = ibox.centre;
bs = ibox.size;
cc = iclient.centre;
cs = iclient.size;

dcx = abs(real(bc-cc));
dcy = abs(imag(bc-cc));

if (dcx<=0.5*(bs+cs)+err && dcy<=0.5*(bs+cs)+err
    && (dcx>=0.5*(bs+cs)-err || dcy>=0.5*(bs+cs)-err))
    b=1;
else
```

```
    b=0;
end

%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

veltree.m

```
% Calculates velocity at each cell of tree

function [vel] = veltree(tree)
global index;
data = [];
vel =[];
in = zeros(size(data));
for ln=1:size(index,1)
    level = index(ln,1);
    k = index(ln,2);
    l = index(ln,3);
    temp = tree(level).box(k,l);
    if(isempty(temp.child))
        data = [data temp.data];
        vel = [vel temp.W];
    end
end
for ln=1:length(data)
    in(ln)   = find(data == tree(1).box.data(ln));
end

vel  = vel(in);

%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Atree.m

```
% Calculates the A-coefficients in cell-cell interaction for FMM using
% recursive logic
% k,l are coordinates of cell
% Nmax is limit to which multipole expansion is carried out
```

```
function [tree] = Atree(level,k,l,tree,Nmax)
fn = [1 1 2 6 24 120 720 5040 40320 362880 3628800 39916800 479001600];

if (~isempty(tree(level).box(k,l).data))

    A = zeros(1,Nmax+1);
    tree(level).box(k,l).A = A;
    ibox = tree(level).box(k,l);

    if(isempty(ibox.child))
        for m=0:Nmax
            ibox.A(m+1) =
            -sum((i/2/pi)*(ibox.dG).*((ibox.centre-ibox.data).^m)/fn(m+1));
        end

    else
        for k1=1:size(ibox.child,1)
            ck = ibox.child(k1,2);
            cl = ibox.child(k1,3);
            tree = Atree(level+1,ck,cl,tree,Nmax);
            ichild = tree(level+1).box(ck,cl);
            Ap = zeros(1,Nmax+1);
            for n = 0:Nmax
                for p=0:n
                    Ap(n+1) = Ap(n+1) +
                     (ichild.A(n-p+1))*(ibox.centre-ichild.centre)^p/fn(p+1);
                end
            end
            ibox.A = ibox.A+Ap;
        end
    end
    tree(level).box(k,l).A = ibox.A;
end

%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Btree.m

```
% Calculates the B-coefficients in cell-cell interaction for FMM using
```

```
% recursive logic
% k,l are coordinates of cell
% Nmax is limit to which multipole expansion is carried out

function [tree] = Btree(level,k,l,tree,Nmax)

%n-factorial
fn = [1 1 2 6 24 120 720 5040 40320 362880 3628800 39916800 479001600];

if (~isempty(tree(level).box(k,l).data))

    B = zeros(1,Nmax+1);
    tree(level).box(k,l).B = B;
    ibox = tree(level).box(k,l);
    zd = ibox.centre;
    for t=1:size(ibox.list,1)
        iclient = tree(ibox.list(t,1)).box(ibox.list(t,2), ibox.list(t,3));
        if(ibox.list(t,4)==2)
            zc = iclient.centre;
            for n=0:Nmax
                for p=0:Nmax-n
                    if (n+p) == 0
                        Knp = log(zd-zc);
                    else
                        Knp = -fn(n+p)/((-(zd-zc))^(n+p));
                    end
                    ibox.B(n+1) = ibox.B(n+1) + iclient.A(p+1)*Knp;
                end
            end
        elseif (ibox.list(t,4)==4)
           for n = 0:Nmax
               if (n==0)
                   Kn = log(zd- iclient.data);
               else
                   Kn = (-fn(n)./((-(zd-iclient.data)).^n));
               end
               ibox.B(n+1) = ibox.B(n+1) + sum((-i/2/pi)*(iclient.dG).*Kn);
           end
        end
    end
```

```
    if(~isempty(ibox.parent))
        iparent =tree(ibox.parent(1,1)).box(ibox.parent(1,2),ibox.parent(1,3));
        Bp = zeros(1,Nmax+1);
        for n=0:Nmax
            for  p = 0:Nmax-n
                Bp(n+1)= Bp(n+1) +
                iparent.B(n+p+1)*(ibox.centre-iparent.centre)^p/fn(p+1);
            end
        end
    ibox.B = ibox.B + Bp;
    end

    tree(level).box(k,l).B = ibox.B ;

    if(~isempty(ibox.child))
        for k1=1:size(ibox.child,1)
            ck = ibox.child(k1,2);
            cl = ibox.child(k1,3);
            tree = Btree(level+1,ck,cl,tree,Nmax);
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Wtree.m

```
% Calculate velocities in a particular cell due to all member vortices and
% outside cells

function [tree] = Wtree(level,k,l,tree,c)

fn = [1 1 2 6 24 120 720 5040 40320 362880 3628800 39916800 479001600];

if (~isempty(tree(level).box(k,l).data))


    Nmax  =length(tree(level).box(k,l).A)-1;
    W0 = zeros(1,length(tree(level).box(k,l).data));
```

```
tree(level).box(k,l).W = W0;
ibox = tree(level).box(k,l);
rcore = c;

for p=1:length(ibox.data)
    for n =1:Nmax
        W0(p) = W0(p) +
                (ibox.B(n+1)).*((ibox.data(p)-ibox.centre).^(n-1))/fn(n);
    end
end

W1 = zeros(1,length(ibox.data));
W3 = zeros(1,length(ibox.data));

for t=1:size(ibox.list,1)
    iclient = tree(ibox.list(t,1)).box(ibox.list(t,2), ibox.list(t,3));

    if(ibox.list(t,4)==1)
      psib = transpose(ibox.data)*ones(1,length(iclient.data));
      psic = ones(length(ibox.data),1)*iclient.data;
      dGc =  ones(length(ibox.data),1)*iclient.dG;
      dist = (psib-psic);

      if( length(iclient.data) == length(ibox.data) && dist(1,1) == 0 )
          dist = dist+1e8*eye(size(dist));
      end
      rho = abs(dist)/rcore;
      r2 = rho.^2;
      q = r2./sqrt(r2.^2+1);

      x = -(i/2/pi)*dGc.*q./(dist);
      W1 = W1 + transpose(sum(x,2));

     elseif(ibox.list(t,4)==3)
            for p=1:length(ibox.data)
                for m=0:Nmax
                    if m+1 == 0
                        Kmp1 = log(ibox.data(p)-iclient.centre);
                    else
                        Kmp1 =
```

```
                            -fn(m+1)/(-(ibox.data(p)-iclient.centre)).^(m+1);
                        end
                    W3(p) = W3(p) + (iclient.A(m+1))*Kmp1;
                    end
                end
         end
      end
   ibox.W = W0+W1+W3;
   tree(level).box(k,l).W = ibox.W;

   if(~isempty(ibox.child))
        for k1=1:size(ibox.child,1)
            ck = ibox.child(k1,2);
            cl = ibox.child(k1,3);
            tree = Wtree(level+1,ck,cl,tree,c);
        end
   end

end

%%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

genfile.m

```
% generates files storing wall and airfoil data for use by DSCPACK

function [] = genfile(len,thetaw,psiwkedge,psilvedge,psi,Zwall)

%%%%%%%%%%%%%% Stores airfoil edge coordinates %%%%%%%%%%%%%%%%%%%%%%%%%%%%
N = 2;
Z1 = [real(psilvedge) imag(psilvedge) 1
      real(psiwkedge) imag(psiwkedge) 1];

fid = fopen('z1pts.dat', 'wb');
fprintf(fid,'%d \n',N);
fclose(fid);
save ('z1pts.dat','Z1','-ascii','-double', '-tabs','-append');

%%%%%%%%%%%%%%% Stores wake coordinates %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
psi = transpose(psi);
```

```
ZZ = [real(psi) imag(psi)];
L = length(psi);
fid = fopen('zzpts.dat', 'wb');
fprintf(fid,'%d \n',L);
fclose(fid);
save ('zzpts.dat','ZZ','-ascii','-double', '-tabs','-append');

%%%%%%%%%%%%%%% Stores number of discretized points used in airfoil %%%%%%%
L = len;
fid = fopen('aflen.dat', 'wb');
fprintf(fid,'%d \n',L);
fclose(fid);

%%%%%%%%%%%%%%% Stores number of discretized points used in wall %%%%%%%%%%
L = length(thetaw);
fid = fopen('thetaw.dat', 'wb');
fprintf(fid,'%d \n',L);
fclose(fid);
save ('thetaw.dat','thetaw','-ascii','-double', '-tabs','-append');

%%%%%%%%%%%%%%% Stores Wall coordinates in w-plane %%%%%%%%%%%%%%%%%%%%%%%%
Zwall = transpose(Zwall);
Wold = [real(Zwall) imag(Zwall)];

save ('wold.dat','Wold','-ascii','-double', '-tabs');

%%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

g1.m

```
% Calculates induced vorticity, v1, on wall and airfoil by the wake
% toggle chooses between wall and airfoil
% r is radius of boundary in w-plane
% q is ratio of radii of cylinders in w-plane
% Zwk, Zlv, Zbody are coordinates of wake and boundary in w-plane
% dGwk, dGlv are the circulation of the wake
% psiwk, psilv are the coordinates of wake in inertial frame

function [v1] = g1(toggle,G0n,r,q,Zwk,Zlv,dGwk,dGlv,Zbody,psiwk,psilv,c)
```

```
dGwt = -dGwk;
dGlt = -dGlv;
limit = 10;
v2 = 0;

%%%%%%%%%%%%%% For airfoil (inner cylinder) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (toggle==0)
    r1 = r;
    r2 = r*(q^0.25);
    r3 = r*q^0.5;

    Zwt = r2^2./conj(Zwk);
    Zlt = r2^2./conj(Zlv);


    for u=1:length(Zbody)
        v1(u) = -G0n/2/pi/r1;
        V=0;
        for n=-limit:limit
            V = V-(1/2/pi)*(sum(dGwk.*(1./(Zbody(u)-Zwk*q^n)
                              -1./(Zbody(u) - r1^2/(q^n)./conj(Zwk))))
                            + sum(dGlv.*(1./(Zbody(u)-Zlv*q^n)
                                       -1./(Zbody(u)- r1^2/(q^n)./conj(Zlv))))
                            + sum(dGwt.*(1./(Zbody(u)-Zwt*q^n)
                                       -1./(Zbody(u)-r3^2/(q^(n))./conj(Zwt))))
                            + sum(dGlt.*(1./(Zbody(u)-Zlt*q^n)
                                    -1./(Zbody(u) - r3^2/(q^(n))./conj(Zlt)))));
        end

        v1(u) = v1(u) - real(Zbody(u)*V/r1);
    end

%%%%%%%%%%%%%%%%%%%%% For wall (outer cylinder) %%%%%%%%%%%%%%%%%%%%%%%%%%

elseif (toggle==1)
    r1 = r/(q^0.25);
    r2 = r;
    r3 = r*q^0.25;
```

```
    Zwt = r2^2./conj(Zwk);

    Zlt = r2^2./conj(Zlv);


    for u = 1:length(Zbody)
        v1(u) = G0n/2/pi/r2;
        V=0;
        for n=-limit:limit
            V = V-(1/2/pi)*(sum(dGwk.*(1./(Zbody(u)-Zwk*q^n)
                                    -1./(Zbody(u) - r1^2/(q^n)./conj(Zwk))))
                        + sum(dGwt.*(1./(Zbody(u)-Zwt*q^n)
                                    -1./(Zbody(u) - r3^2/(q^n)./conj(Zwt))))
                        + sum(dGlv.*(1./(Zbody(u)-Zlv*q^n)
                                    -1./(Zbody(u) - r1^2/(q^n)./conj(Zlv))))
                        + sum(dGlt.*(1./(Zbody(u)-Zlt*q^n)
                                   -1./(Zbody(u) - r3^2/(q^n)./conj(Zlt)))));

        end

        v1(u) = v1(u) - real(Zbody(u)*V/r2);
    end
end

%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

velcal.m

```
% Calculates the airfoil velocity during clap-and-fling

function [dln,dalphan] = velcal(V,time,c,alphan)

dtheta = pi/4;
wrot =2*dtheta*V/(1.74*c);
tau = V*time/c;

if tau<1.3
    dln = 0.5*V*(1+cos(pi+pi*(tau)/1.18));
elseif tau<3.28
    dln=V;
```

```
elseif tau<4.46
    dln = 0.5*V*(1-cos(pi-pi*(tau-3.28)/1.18));
elseif tau<6.27
    dln = 1e-11;
elseif tau<7.45
    dln = -0.5*V*(1+cos(pi+pi*(tau-6.27)/1.18));
else
    dln = -V;
end

% dln = -1e-11;
if tau<3.67
    dalphan=0;
elseif tau<5.4
    dalphan = -0.5*wrot*(1-cos(2*pi*(tau-3.67)/1.74));
    dln = dln - i*(c)*exp(-i*alphan).*dalphan;
elseif tau<7.14
    dalphan = -0.5*wrot*(1-cos(2*pi*(tau-5.4)/1.74));
else
    dalphan = 0;
end

%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

forcecal.m

```
% Calculates forces from force impulse data series
% G0nm is the quasi-steady circulation
% U0nm is the complex velocity of airfoil
% f1, f2, f3, f4, f5 are the force impulse data series
% p is the density of fluid

function [F] = forcecal(U0nm,G0nm,f1,f2,f3,f4,f5,dt,p)

%%%%%%%%%%%% Filter data using a butterworth filter %%%%%%%%%%%%%%%%%%%%%
[b,a]=butter(1, 0.05);

f1n = filter(b,a,f1);
f2n = filter(b,a,f2);
f3n = filter(b,a,f3);
```

```
f4n = filter(b,a,f4);
f5n = filter(b,a,f5);

%%%%%%%%%%%%%%%% Calculate Forces by differentiating filtered data %%%%%%%
F0 = -i*p*conj(U0nm).*G0nm;
F1 = i*p*sgdiff(f1n,dt);
F2 = i*p*sgdiff(f2n,dt);
F3 = -F0 + i*p*sgdiff(f3n,dt);
F4 = i*p*sgdiff(f4n,dt);

F = F0 + F1 + F2 + F3 + F4;

%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

sgdiff.m

```
%% Differentiates a time series data, y
function data = sgdiff(y,dt)

N = 4;                 % Order of polynomial fit
F = 11;                % Window length
[b,g] = sgolay(N,F);   % Calculate S-G coefficients

HalfWin  = ((F+1)/2) -1;
for n = (F+1)/2:length(y)-(F+1)/2,
  % Zero-th derivative (smoothing only)
  SG0(n) =   dot(g(:,1), y(n - HalfWin: n + HalfWin));

  % 1st differential
  SG1(n) =   dot(g(:,2), y(n - HalfWin: n + HalfWin));
end

data = [SG1/dt zeros(1,(F+1)/2)];

%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Bibliography

[1] Ellington, C. P. (1984) "The aerodynamics of hovering insect flight .5. A vortex theory," *Philosophical Transactions of the Royal Society of London Series B-Biological Sciences*, **305**(1122), pp. 115–144.

[2] VandenBerg, C. and C. P. Ellington (1997) "The three-dimensional leading-edge vortex of a 'hovering' model hawkmoth," *Philosophical Transactions of the Royal Society of London Series B-Biological Sciences*, **352**(1351), pp. 329–340.

[3] Lehmann, F. O. (2004) "The mechanisms of lift enhancement in insect flight," *Naturwissenschaften*, **91**(3), pp. 101–122.

[4] P., S. S. and D. M. H. (2002) "The aerodynamic effects of wing rotation and a revised quasi-steady model of flapping flight," *Journal of Experimental Biology*, **205**(8), pp. 1087–1096.

[5] Weisfogh, T. (1973) "Quick estimates of flight fitness in hovering animals, including novel mechanisms for lift production," *Journal of Experimental Biology*, **59**(1), pp. 169–230.

[6] Rayner, J. M. V. (1979) "Vortex theory of animal flight .1. Vortex wake of a hovering animal," *Journal of Fluid Mechanics*, **91**(APR), pp. 697–730.

[7] Dickinson, M. H., F. O. Lehmann, and S. P. Sane (1999) "Wing rotation and the aerodynamic basis of insect flight," *Science*, **284**(5422), pp. 1954–1960.

[8] Birch, J. M. and M. H. Dickinson (2003) "The influence of wing-wake interactions on the production of aerodynamic forces in flapping flight," *Journal of Experimental Biology*, **206**(13), pp. 2257–2272.

[9] Lam, C. M. G. (1989) *Nonlinear wake evolution of Joukowski aerofoils in severe maneuver*, Ph.D. thesis, Massachusetts Institute of Technology.

[10] von Karman, T. and W. R. Sears (1938) "Airfoil theory for non-uniform motion," *Journal of Aerodynamic Science*, **5**(10), pp. 379–390.

[11] Minotti, F. O. (2002) "Unsteady two-dimensional theory of a flapping wing," *Physical Review E*, **66**(5).

[12] Ansari, S. A., R. Zbikowski, and K. Knowles (2006) "Non-linear unsteady aerodynamic model for insect-like flapping wings in the hover. Part 1: methodology and analysis," *Proceedings of the Institution of Mechanical Engineers Part G-Journal of Aerospace Engineering*, **220**(G2), pp. 61–83.

[13] ——— (2006) "Non-linear unsteady aerodynamic model for insect-like flapping wings in the hover. Part 2: implementation and validation," *Proceedings of the Institution of Mechanical Engineers Part G-Journal of Aerospace Engineering*, **220**(G3), pp. 169–186.

[14] Tomotika, S., Z. Hasimoto, and K. Urano (1951) "The forces acting on an aerofoil of approximate Joukowski type in a stream bounded by a plane wall," *Quarterly Journal of Mechanics and Applied Mathematics*, **4**(3), pp. 289–307.

[15] Tomotika, S., K. Tamada, and H. Umemoto (1951) "The lift and moment acting on a circular-arc aerofoil in a stream bounded by a plane wall," *Quarterly Journal of Mechanics and Applied Mathematics*, **4**(1), pp. 1–22.

[16] Dappen, H. (1987) "Wind-tunnel wall corrections on a Two-dimensional plate by conformal mapping," *AIAA Journal*, **25**(11), pp. 1527–1530.

[17] Lighthill, M. J. (1973) "Weis-Fogh Mechanism of lift generation," *Journal of Fluid Mechanics*, **60**(21), pp. 1–17.

[18] Edwards, R. H. and H. K. Cheng (1982) "The separation vortex in the Weis-Fogh circulation-generation mechansim," *Journal of Fluid Mechanics*, **120**, pp. 463–473.

[19] Hu, C. (1998) "Algorithm 785: A software package for computing Schwarz-Christoffel conformal transformation for doubly connected polygonal regions," *ACM Transactions on Mathematical Software*, **24**(3), pp. 317–333.

[20] Spedding, G. R. and T. Maxworthy (1986) "The generation of circulation and lift in a rigid two-dimensional fling," *Journal of Fluid Mechanics*, **165**, pp. 247–272.

[21] Miller, L. A. and C. S. Peskin (2005) "A computational fluid dynamics of 'clap and fling' in the smallest insects," *Journal of Experimental Biology*, **208**(2), pp. 195–212.

[22] Lehmann, F. O., S. P. Sane, and M. Dickinson (2005) "The aerodynamic effects of wing-wing interaction in flapping insect wings," *Journal of Experimental Biology*, **208**(16), pp. 3075–3092.

[23] Rosenhead, L. (1932) "The point vortex approximation of a vortex sheet or the formation of vortices from a surface of discontinuity-check," *Proceedings of the Royal Society of London A*, **134**, pp. 170–192.

[24] Chorin, A. J. (1973) "Numerical study of slightly viscous flow," *Journal of Fluid Mechanics*, **57**, pp. 785–796.

[25] Majda, A. J. and A. L. Bertozzi (2002) *Vorticity and incompressible flow*, Cambridge Texts in Applied Mathematics, Cambridge.

[26] Lewis, R. I. (1991) *Vortex element methods for fluid dynamic analysis of engineering systems*, Cambride Engine Technology Series, Cambridge.

[27] Greengard, L. and V. Rokhlin (1987) "A fast algorithm for particle simulations," *Journal of Computational Physics*, **73**(2), pp. 325–348.

[28] Strickland, J. H. and R. S. Baty (1997) *A two-dimensional fast solver for arbitrary vortex distributions*, *Tech. rep.*, SCANDIA.

[29] Pashaev, O. K. and O. Yilmaz (2008) "Vortex images and q-elementary functions," *Journal of Physics a-Mathematical and Theoretical*, **41**(13).

[30] Noca, F., D. Shiels, and D. Jeon (1999) "A comparison of methods for evaluatng time-dependent fluid dynamic forces on bodies, using only velocity fields and their derivatives," *Journal of Fluids and Structures*, **13**, pp. 551–578.

[31] Spallart, P. R. (1988) *Vortex methods for separated flows*, *Tech. rep.*, NASA.

[32] Wang, J. Z. (2000) "Two dimensional mechanism for insect hovering," *Physical Review Letters*, **85**(10), pp. 2216–2220.

[33] WU, C., J. and H. HU-CHEN (1984) "Unsteady aerodynamics of articulate lifting bodies," *American Institute of Aeronautics and Astronautics, Applied Aerodynamics Conference, 2nd, 21-23 Aug.*