CRANFIELD UNIVERSITY

COLLEGE OF DEFENCE TECHNOLOGY

DEPARTMENT OF AEROSPACE, POWER & SENSORS

PhD Thesis

ACADEMIC YEAR 2003/04

Christopher B Pedersen

An Indicial-Polhamus Model of
Aerodynamics of Insect-like Flapping Wings in Hover

Supervisor: Dr R Żbikowski

17 June 2003

Abstract

As part of the ongoing development of Flapping-Wing Micro Air Vehicle (FMAV) prototypes at RMCS Shrivenham, a model of insect-like wing aerodynamics in hover has been developed, and implemented as **MATLAB** code. The model is intended to give better insight into the various aerodynamic effects on the wing, so is as close to purely analytical as possible. The model is modular, with the various effects treated separately. This modularity aids analysis and insight, and will allow future refinement of individual parts. However, it comes at the expense of considerable simplification, which requires empirical verification. The model starts from quasi-steady inviscid flow around a thin 2D rigid flat wing section, accounting for viscosity with the Kutta-Joukowski condition, and the leading edge suction analogy of Polhamus. Wake effects are modelled using the models of Küssner and Wagner on a prescribed wake shape, as initially used by Loewy. The model has been validated against experimental data of Dickinson's Robofly, and found to give acceptable accuracy. Some empirically inspired refinements of the Polhamus effect are outlined, but need further empirical validation.

This thesis comprises of six main parts: Part 1 is introductory material, and definitions, including an overview of what insect-like flapping flight actually entails, and detailed definitions of the variables and terms used later. Part 2 describes the new theoretical model, and a simple scaling analysis of the forces and moments predicted. Part 3 deals with the **MATLAB** implementation of the above theory, and the considerations required when adapting the theory for computational use. Part 4 shows and discusses the results of the above code, against experimental measurements on Dickinson's Robofly. Part 5 is the conclusions, including a comprehensive list of all assumptions made in the theory. Part 6, the appendices, contain useful mathematical identities, and a copy of the code that was developed.

Acknowledgements

This work has been supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through Grants GR/M22970 and GR/M78472. The latter has been co-funded by the UK Ministry of Defence.

The author gratefully acknowledges the assistance of those in the Engineering and Zoology community that have helped this work come to fruition. Special mention is due to Prof. Charlie Ellington of Cambridge University Zoology Department, Dr. Adrian Thomas and Graham Taylor of Oxford University Zoology Department, and Dr. Robin Wootton of School of Biological Sciences, Exeter University, for their invaluable insights into how insects manage to stay airborne. The data used in this report was kindly made available by Prof. Michael Dickinson and Dr. William Dickson of California institute of Technology, from their Robofly project.

On the personal side, I'm grateful to those that have supported me through this project - my friends and family for their encouragement and my mother, Jennifer, for teaching me at a tender age that you shouldn't pull the wing off flies.

Lastly, but by no means least, thanks go to my supervisor, Dr. Rafał Żbikowski, of Cranfield University, RMCS Shrivenham, UK, who went above and beyond the call of duty with inspiration, encouragement and fruitful discussions.

Contributions

- A modular approach to modelling the forces and moments on a thin, flat plate undergoing a flapping motion consisting of rotation about the root and rapid pitching up to 180°.
- Wakeless solutions for quasi-steady and added mass forces for the flapping motion above, without assuming small angle of attack, formulated to avoid the use of angle of attack as a parameter.
- An inviscid wake model for the effect of a highly curved wake filament, albeit with considerable simplifications.
- A generalisation of the Polhamus leading edge suction analogy, to include the effect of rapid pitching at large pitch angles.
- A method of calculating the force and moment of a wing, based on the kinematics of the tip, and a number of wing shape parameters.
- A scaling analysis of the forces and moments on the wing, and merit criteria such as induced power per mass.
- Adaptation of standard non-dimensional groups and parameters to the flapping motion above. Specifically, adaptation of C_L and advance ratio.
- An outline of how the above motion justifies the use of rotary chord, and a proposal for how this can be used in a "pseudo-chord analogy" (see Section 20).
- A code implementation of the above model. Due to the nature of the model, this code does *not* rely on successive approximation (as described in Section 13.2), so the runtime is dramatically lower than standard CFD codes. This comes at the expense of considerable simplification in forming the model above. The code, like the model, is modular so the individual effects can be examined independently, giving better insight into the results.

Contents

List of Figures

Glossary

Symbols

A_S	horizontal swept area (m^2)
A_W	wing area (m^2)
B	greatest semichord of wing (m)
b	local semichord of wing (m)
C_L	lift coefficient
C_D	drag coefficient
F	force (N), see page 25 for subscripts
M	moment (Nm), see page 29 for subscripts
Q	abbreviation for $\sqrt{1-\zeta^2}$
R	radius of wing (m)
r	radial distance, normalised w.r.t R
s	semichords distance travelled
t	time (s)
T	wingbeat period (s)
u	velocity (m/s), see page 25 for subscripts
u_i	downwash velocity (m/s)
x, y, z	rectangular global coordinates (m)
α	angle of attack (radians)
β	pitch angle (radians)
$\Gamma(x)$	total circulation of wing from leading edge to x
$\gamma(x)$	vorticity of wake filament per unit of distance x
θ,ψ	sweep and plunge angle (radians)
ν	kinematic viscosity (m^2/s)
ζ, η	normalised wing-fixed coordinates, see Section 7.1.3
ρ	fluid density (kg/m^3)
Φ	potential function
Λ	aspect ratio
$\angle$	sweepback angle of delta wing (degrees)
ψ'_W	Wagner function
ψ'_K	Küssner function
ψ_W	Wagner perturbation function
ψ_K	Küssner perturbation function

superscripts

$\dot{x}$	time differential of x
$\ddot{x}$	second time differential of x
$+, -$	on upper / lower surface

subscripts

T	linear / translational
R	radial / rotational
x	x-component
y	y-component
z	z-component
P	parallel direction (see section 7)
N	normal direction (see section 7)
V	vertical direction (see section 7)
H	horizontal direction (see section 7)
L	vertical, in rectangular coordinates (see section 7)
D	horizontal in rectangular coordinates (see section 7)
$\mathbf{T}$	total (vector) quantity (see section 7)
Q	quasi-steady effect
A	added mass effect
P	Polhamus effect (LEV)
W	Wagner wake effect
K	Küssner wake effect
E	at arbitrary chordwise position.
f	for fluid particle
l	at leading edge (see section 7)
m	at midpoint edge (see section 7)
r	at rear neutral point edge (see section 7)
t	at trailing edge (see section 7)
T	at wing tip (see section 7)
TD	translational part of Dirichlet component (see section 7)
RD	rotational part of Dirichlet component (see section 7)
TK	translational part of Kutta-Joukowski component (see section 7)
RK	rotational part of Kutta-Joukowski component (see section 7)

Mathematical Abbreviations and symbols

asin	inverse sin function
C	cos
S	sin
$\mathrm{S}^m_{n\alpha}$	$\sin(n\alpha)^m$ (general form)
∂	partial differential
Δ_n	correction due to n

Nondimensional Groups

Re	Reynolds Number
J_H	hovering advance number

Acronyms

MAV	micro air vehicle
$FMAV$	flapping winged micro air vehicle
LEV	leading edge vortex
$MEMS$	micro electro-mechanical systems

Part I
Introduction

This part provides an introduction to the concept of flapping wing flight, starting with the motivation for exploring flapping wing flight in Section 1. It is followed by a thesis overview - a section-by-section description of how this thesis is laid out, in Section 2. This is followed by an overview of the model that was developed in Section 3, along with a description of the context of the model, especially other work it was based on, in Section 4. Some of the main observations from the biological community on insect flight are summarised, along with how these may be useful for aerodynamic modelling in Section 5. Next, terminology is defined, along with some terms that have to be altered slightly to be of use in the model in Section 6. Finally, in Section 7 definitions are provided of the variables and coordinate systems that are used through the remainder of the thesis.

1 Motivation

As recent field experience has proven, there is considerable potential for unmanned air vehicles (UAVs) in military applications. The ability to get real-time battlefield information from "over the next hill" without risking the lives of your troops is the dream of every commander. However, in tight environments such as urban or cave fighting, the size and lack of mobility of current UAVs makes them unsuitable. There thus exists a niche for small, highly maneuverable reconnaissance drones discretely to penetrate confined spaces, and manoeuvre in them without the assistance of a human tele-pilot. Generally, these vehicles will be useful in civilian applications involving dull, dangerous or dirty (D3) environments, where direct or remote human assistance is not feasible (See [1]).

The inspiration for the FMAV is the closest natural analogy - insects. Here is a whole school of vehicles at the appropriate scale that we *know* work. The greatest advantage of flapping-winged flight is that it can use thin, flexible wings that are a) extremely silent compared to rigid wings, and b) capable of withstanding accidental impact with walls and other obstructions, something that would be disastrous for a standard rotorcraft. Visual mimicry of insects for covert use is considered an appealing, but ultimately impractical idea.

In a study undertaken at Cranfield University (RMCS Shrivenham), it was concluded that the advances needed for such vehicles in compact actuators, high-energy density batteries, smart wing materials and onboard logic will not be available for the next five to ten years. Therefore, this work is part of an ongoing concurrent design exercise - rather than waiting for the technology to be available, the design process is started early by creating functionally similar, but less compact physical models. As part of this design process, it is obviously desirable to be able to predict aerodynamic forces, both for flight performance, and for designing the airframe to cope with the loads experienced. This is the starting point for this project.

2 Thesis overview

This thesis consists of six main parts: Part 1 (this part) is introductory material, and definitions.

Section 3 is an overview of the theoretical model developed, and Section 4 is the theoretical background for the model. Section 5 contains an overview of what insect-like flapping flight actually entails, and how this is envisaged to be implemented in an FMAV. Because this is an unusual flight regime some additional terminology is needed. This is taken mainly from insect biology, and is outlined in Section 6. Also in this section are some considerations of how standard aerodynamic quantities and non-dimensional parameters need to be adapted for this application. Note especially that the angle of attack α has been abandoned as a usable parameter. This section leads directly into the detailed definitions of variables in Section 7, which deals with definitions of symbols and axis systems used.

Part 2 is the description of the proposed theoretical model. Sections 8 to 11 deal with the development of the theoretical model; the model is outlined below, in Section 3. A simple scaling analysis is presented in Section 12, along with conclusions on how this scaling is expected to affect performance.

Part 3 deals with the **MATLAB** implementation of the above theory, and the considerations required when adapting the theory for computational use. Note here that the code is *not* computational, in the sense of being a Computational Fluid Dynamics (CFD) model, merely a computer implementation of the analytical theory. This will be discussed further in Section 3. Note the focus on proofing the code against legacy, and modularity to allow improved modelling of individual effects.

Part 4 shows the results of the above code, on two sample datasets, in Sections 16 to 15. One dataset is from experimental measurements on Dickinson's Robofly [2], the other a predicted possible kinematics and wing geometry for an FMAV. These results, especially the comparison between the predicted forces and those actually measured, are discussed in Section 18.

Part 5 is the conclusion. The main conclusions are outlined in Section 19, including a comprehensive list of all assumptions made in the theory, and finally, suggestions for further refinement of the model are outlined in Section 20.

Part 6, the appendices, contain useful mathematical identities, which are utilised throughout the thesis, and a copy of the code used, with annotated explanation of how it functions.

3 Model overview

The aim of this model is to gain insight into the factors affecting aerodynamic performance of an $FMAV$ wing. Although this could be done by computational fluid dynamic (CFD) methods, it was desired to have a model that was predominantly analytical, because this gives greater insight into the aerodynamic effects. Also, an analytical model can eventually be reduced to a state-space form, for simple implementation of flight control. For more on the state-space expression, see [3]. This critical difference between standard CFD methods

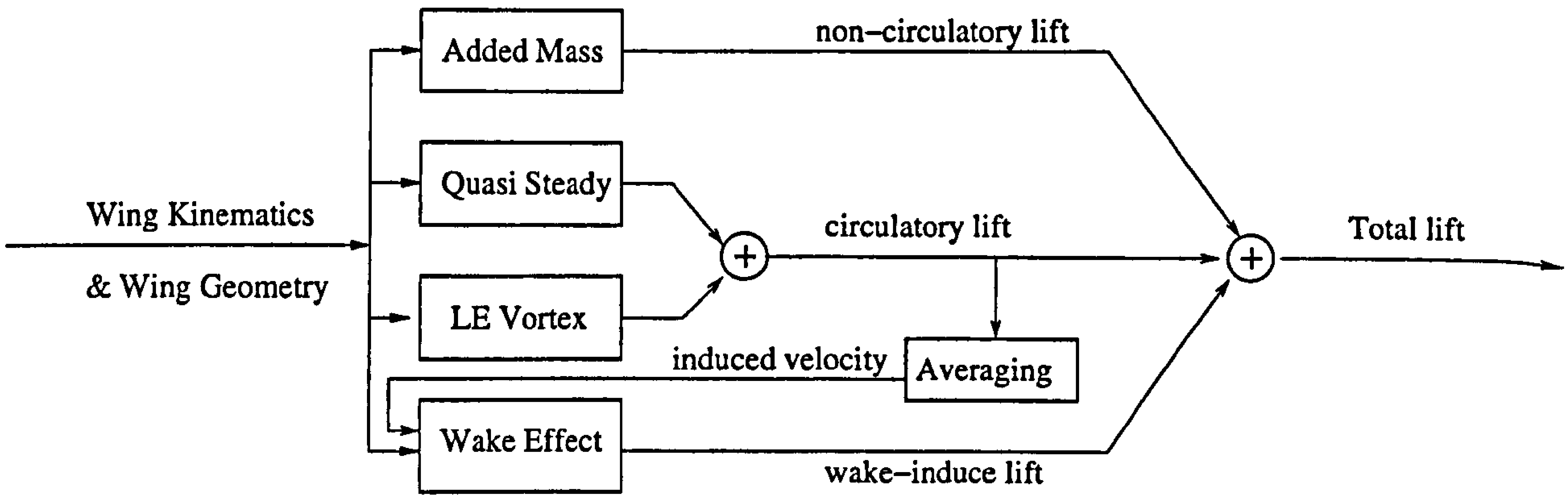


Figure 1: Overview of the modular method for modelling the aerodynamic effects. Note the lack of feedback loops in the above - the method is not iterative.

and our computer-based implementation of analytical theory will be revisited in Section 13.2.

Therefore, the main motivation for this model has been the need for an analytically tractable representation of the complex aerodynamics involved (see [4]). Such a model is useful for the following reasons:

1. Insight into the contributing factors to the overall lift effect.
2. Speed and ease of computation.
3. Possibility of inclusion in a flight dynamic model of the whole vehicle.
4. The possibility of modular refinement of the constituent parts of the model.
5. The possibility of performing simple scaling analysis on the results of the model.

All these considerations informed the choice of the approach adopted here. Wherever possible, existing analytical formulae for similar aerodynamic problems were exploited, modified and combined. This creative synthesis often involved compromise between physical fidelity and analytical tractability, with the balance usually tipped in favour of the latter. Because of this, the result is a first-order model.

This scheme is is shown diagrammatically in Figure 1.

The model starts with the inviscid flow around a thin, flat wing section in 2D, using the thin aerofoil theory (see [5, Chapter 4]). For this, a velocity potential approach can be adopted, using complex numbers to represent vector quantities such as position and velocity. The velocity potential is used to derive the quasi-steady forces in Section 8, and again for the added mass forces in Section 9. This uses the standard approach found in any good textbook on unsteady aerodynamics, but includes extra terms for the velocity due to rotation that most textbooks omit, since they can usually assume fast forward motion.

The separated flow at the sharp leading edge is modelled using the leading edge suction analogy of Polhamus, in Section 10. Briefly, this assumes separated flow, and models the effect of the attached vortex that is expected to occur on the upper side near the leading edge. The model assumes that any leading edge suction is rotated through 90°, to become an additional normal force. Although this is not a theoretical model, it has received plenty of empirical validation, for example in the development of sharp-edged delta wings. Additionally, it has the advantage of being extremely simple.

Simple modelling of wake effects are the main thrust of this work, dealt with in Section 11. Briefly, wake effects usually attenuate changes in forces, as the shed vorticity will oppose the creation of vorticity bound to the wing. The wake is treated as a thin filament of vorticity shed from the trailing edge, and the effect cannot be solved analytically for the general case. Instead, simplified models for cases that *can* be solved are used. First amongst these is the Wagner function, which deals with the effect of a $2D$ straight-line wake behind an arbitrarily pitching and accelerating airfoil. The Wagner function reduces to a function of the distance travelled since a given change in lift coefficient, which can be summed for all changes of lift coefficient since impulsive start. The Küssner function deals with a similar case, but for a change in lift coefficient that is stationary in space (such as a gust), that the wing gradually enters as it moves. It, too, reduces to a function of the distance travelled and the change of lift coefficient. Since the analysis pertains to hovering flight, the wake will tend to move downwards from the vehicle over time. Loewy modelled the wake of a hovering rotorcraft by splitting the wake into straight-line elements: a primary wake behind the wing, and a series of straight-line secondary wakes below the wing. This model is adapted for use with flapping flight.

The main simplifying assumptions made were:

1. The wing is thin and flat.
2. The flow is stationary for purposes of force calculations.
3. The flow is entirely inviscid.
4. The effect of the LEV is to rotate the leading edge suction force by 90°.
5. The LEV dissipates immediately when shed.
6. The flow leaves the trailing edge smoothly, satisfying the Kutta-Joukowski condition.
7. The wake is treated as a thin, globally stationary filament of vorticity, which has no self-induced velocity effects.
8. The wake is split into single-stroke elements, each of which is assumed to be a straight line.
9. The wake moves under constant downwash velocity u_i, without deforming under its own induced velocity.

Also, only the Polhamus model of the LEV accounts for flow separation. The rest of the model assumes it to be attached, despite high angle of attack, fast rotation, and so on.

4 Context of model

In this section, the context of the proposed model is outlined, especially the earlier work it was based on. This will be described in more detail in Part II, but will be summarised here to give a better overview of the methods involved. The reader may wish to refer back to this section after having read Part II.

There is a commonly held misconception that according to engineers, bumblebees cannot fly. This is true, up to a point. According to the classical steady-state theory of aerodynamics, insect wings simply do not generate enough lift. For example, Pringle [6] modelled the lift of insect wings, using the steady-state aerodynamic theory, which is based solely on the velocity of the wing and the angle of attack, and found a significant shortfall in the force predicted versus the force observed. The reason for this is obvious - insect flight is not even remotely steady. If the effect of pitching rate is included in the theory, the predicted lift is considerably higher. This is the quasi-steady theory of aerodynamics. For a good general overview of quasi-steady and unsteady thin-wing aerodynamics, and the potential model associated with them, the author recommends Katz & Plotkin [7], this book underlines the fundamental dependence of the aerodynamic forces on normal velocity, not angle of attack, and it explains the connection between potential and bound vorticity well. Helicopter aerodynamics, and the modelling of the pitching and plunging rotors, are explained in Leishman [3]. This work is recommended for a good explanation of the effect of the unsteady wake on the rotor, and the Theodorsen and Loewy models of same.

A good collection of experimental observations on wings at low Reynolds numbers, specifically aimed at MAV and FMAV applications can be found in Mueller [8], with particular reference to the work of Ellington & Usherwood [9] and Hall & Hall [10]. The concept of added mass, and how this relates to the unsteady potential form of the Bernoulli equation, is explained in Newman [11]. The mathematics of this problem are covered more rigorously in Milne-Thomson [12] and Sedov [13].

The LEV was first proposed as a vortex lift mechanism on delta wings by Polhamus [14], and later refined by Bradley et al. [15] and Purvis [16], amongst others. A good review of the refinements to Polhamus's method can be found in Lamar [17]. The idea that LEVs could be a high-lift mechanism in insect flight was first suggested by Ellington, in [18] [19] [20] [21] [22] [23], and later observed experimentally on a scaled model of a Hawkmoth wing by van den Berg and Ellington [24] and Ellington et al. [25]. The same LEV was observed on a model of the much smaller fruit fly by Birch & Dickinson [26].

This is not the first model to attempt to separate the contributions of various aerodynamic effects, to create a a modular model. Ellington [22] proposed the pulsed actuator disc model of the wake, which simply models the wake as a series of vortex rings, shed once per stroke, and convected downwards by a constant downwash velocity. He then applied this effect as a correction to the average lift during a cycle. Although this is a good first-order model, in that it correctly identifies the general shape of the wake vorticity, it does not model the unsteady lift profile during the individual strokes. Therefore, it is necessary to use the Loewy model above to model the instantaneous effect of the secondary wakes. Note, however, that the secondary wake shape model of 2-D horizontal filaments of vorticity is structurally similar to the pulsed actuator disc model. Walker [27] modelled the lift of the wing using the basic formula $F = \rho\, U\, \Gamma$, where U is the forwards velocity of the wing. He then treated the bound vorticity of the wing Γ as a superposition of four circulation components, similarly to this method. However, he introduced empirical corrections to the vorticity, for example for the observed effect of the wake on the wing. The aim of this model was to similarly split the wing lift model into separate contributions, but to keep then fully theoretical. The flight regimes that are likely to be investigated differ very much from standard aerodynamics, and even other insect aerodynamic models, so the use of empirical correction cannot be justified. This semiempirical approach of Walker's has the disadvantage of lumping together, in an unknown way, disparate aerodynamic effects by representing them through an amalgamated measurement. This is avoided in the model developed here, where each lift component is clearly identified and precisely derived.

Our wake modelling was initially based on the method of Theodorsen [28], and the generalization of this by van der Wall and Leishman [29]. The method of modelling the quasi-steady wing lift used in this thesis is based on Theodorsen's, as mentioned in Section 8. He modelled the potential function of the wing as a sum of contributions due to pitching, forward translation and plunging. He assumed the wing to be thin and flat, and that the angle of attack is low. He then assumed constant forward velocity, and sinusoidal variation of the plunging and pitching. For this case, the integral of the effect of the wake vorticity reduced to an analytical function (based on Bessel functions), which could be expressed as a function of the reduced frequency parameter k. van der Wall and Leishman further generalized this to include the variation of forwards velocity, for the application to helicopter rotors. Although this model was initially extremely promising for modelling the wake effects, in that Theodorsen's expressions for potential could readily have been generalised to the FMAV case of high pitching angle, there was a major problem: the wing reverses. The generalisation provided by [29] did not extend to cases where the wing horizontal velocity is 0 or reverses direction. This is unsurprising, as the model was derived for helicopters, where such a case will never occur. For this reason, indicial methods were deemed necessary and the Wagner [30] and Küssner [31] theories were introduced, both of which have been described in Section 11.

The effect of the secondary wakes, as modelled by Loewy, and how it was adapted for the FMAV application will be explained in detail in Section 11. Briefly, it uses Loewy's model of splitting the wake into a series of flat, horizontal lines that are convected downwards by a constant downwash velocity. As mentioned earlier in this section, this is a similar geometry to the pulsed actuator disc model of Ellington.

The added mass effect is described in most textbooks on the potential theory of unsteady aerodynamics. The author recommends Katz & Plotkin [7] or Newman [11] for a good overview of this. Briefly, using potential flow theory allows the added mass effects to be split into two parts: added mass due to the minimum-energy solution to flow around the wing (the irrotational Dirichlet solution), and a correction that satisfies the Kutta-Joukowski condition of the flow leaving the trailing edge smoothly (the Kutta-Joukowski correction). Most authors tend to express only the Dirichlet component, ignoring the effect of the Kutta-Joukowski correction. This is understandable, in that the Kutta-Joukowski component of added mass is closely associated with wake vorticity, and the cross-coupled effect of the wake and the Kutta-Joukowski component of added mass is not readily modelled. This is discussed further in Section 9. In general, most research on added mass has been in the offshore industry, where the blunter bodies and denser fluid makes added mass a far more important effect than in aerodynamics. Works of interest in this field are Keugelan & Carpenter [32], who proposed a characteristic non-dimensional parameter for oscillating bodies, since called the Keugelan-Carpenter number, and the refinement of Huse [33], which described the viscous and added mass effects on oscillating plates by a simplified correction to the drag coefficients, as a function of the Keugelan-Carpenter number. The early models of added mass effect were based on the work of Huse, but since his model relies on empirical force coefficients, it was considered to be of limited scope for this application, since empirical coefficients are not desirable. For further reading on this subject, and for future refinements that will occur after the time of publishing this work, the author recommends examining the proceedings of the Offshore Technology Conference (OTC), which is held in Houston, Texas, USA.

5 Insect-like flight

5.1 Biomimetic extraction

It is generally accepted that nature produces good designs. For any ecological niche, the combination of ruthless selection with billions of design iterations (generations), has left modern-day insects optimised for survival. However, although flying efficiency is important to survival, it is not the only merit criteria. Insects have to be able to find (possibly catch) food, mates and shelter all the while avoiding detection and capture by predators. All of these may influence wing design away from aerodynamically optimal. For example, butterflies rely on very erratic flight paths to avoid predators - less efficient, but better for survival. Similarly, the process of natural selection means that an insect need not only be the best adapted, but have the best-adapted ancestors. This may have caused some species to go down evolutionary dead ends, where they are finding local, rather than global, optima. Since the insect is an integrated organism, the wing design is also limited by other parts of the body, and naturally available materials. For example, during the late paleozoic (about 250 million years ago) there was a period of high oxygen content in the air which lead to gigantism in most insects. This shows that breathing apparatus is a limiting factor on the scale of insects (see [34]).

The conclusion of this is that insects are not highly-tuned machines, designed to operate at one condition, but have robustness to changing conditions and often have non-flying evolutionary pressures on their wing design. Thus, it is dangerous to use insects as a "blueprint" without understanding which features of their design are contributing to flight performance, and which are there for other reasons.

This process of identifying the salient features of a natural design, and how they can be applied to the FMAV has been dubbed a *Biomimetic Extraction.* Biomimetic because we are mimicking a biological system, and extraction because we are extracting only the usable portions of the design.

5.2 Wing geometry and structure

Insect wings are not streamlined aerofoils - in fact they are angular with rough surface textures, and seem decidedly non-aerodynamic at first glance (see Figure 2).

Insect wings are thin and flexible, tending to have most of the mechanical strength towards the leading edge. In structure, they somewhat resemble a sail [36], with a thin, flexible membrane kept in shape by thick cuticle at the front of the wing (the equivalent of a mast), and veins in the wing that acts like the spars of a sail, retaining its form and camber despite aerodynamic and inertial loading. Insect wing shape is actively controlled by the wing base articulation, and passively deformed by internal, elastic and aerodynamic loads. All actuation happens at the root, which is a considerable simplification of the wing construction. The pattern of venation and stiffness in nonetheless very complex.

The structural components of the wing, such as the veins, taper towards the tip, where the structural loads are lowest. This makes the insect wing flexible to tip impact.

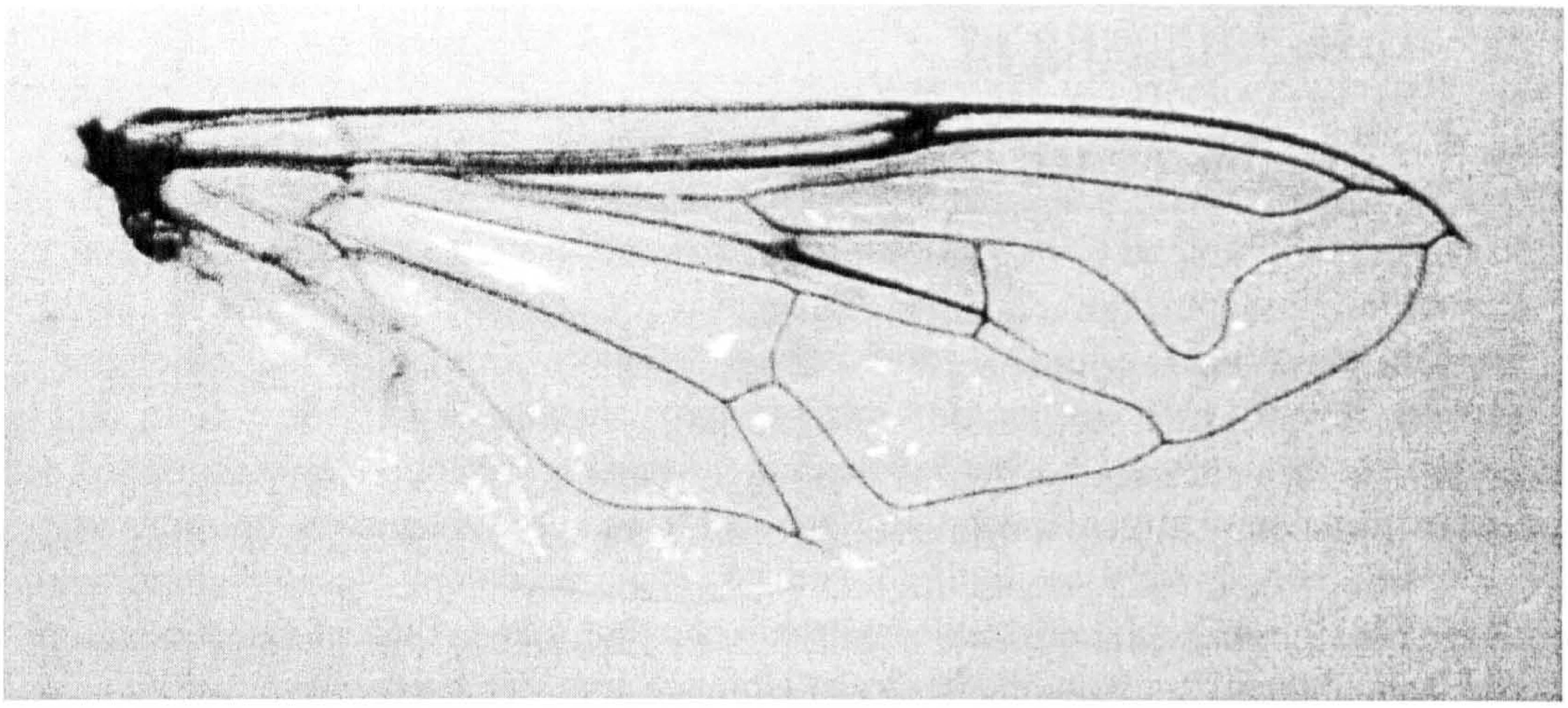


Figure 2: An Eristalis wing, showing the thick cuticle of the leading edge and surface spars, and the thin wing membrane. Picture from [35].

The deformability of the wing causes it to twist along its length - this is in effect similar to the washout of conventional airfoils, without which the angle of attack would increase towards the tip.

Many insect wings bear the signs of deliberate lines of weakness across the wing, for the purpose of transverse bending. This is useful for avoiding damage in collisions, but has also been observed by Wootton [36], [37] as a likely mechanism of camber control.

5.3 Wing kinematics

Insect wing kinematics are fundamentally similar to helicopter rotor kinematics: they rotate about a fixed point (hinge), so airspeeds on the wing increase with distance from the hinge. The majority of the motion is in the horizontal plane, and is called the *sweeping* motion, where the wing moves cyclically forwards and backwards. Additionally, the wing undergoes vertical *plunging* and pitching (called *rotation*). Each wing cycle consists of a forwards/downwards stroke, called the *downstroke*, and an upwards/backwards stroke, called the *upstroke*. These motions are defined more rigorously in Section 7.

A hovering insect typically flies with a near horizontal stroke plane (the mean line between up- and downstroke), and antisymmetrical strokes in a figure-of-eight motion[1], as shown on Figure 3. Either end of each stroke has fast, localised rotation to keep the leading edge forwards in the direction of travel. As the insect moves to forward flight, the stroke plane will incline, and the strokes become asymmetric, with most of the lift generation being caused on the downstroke.

[1]The figure of eight has been chosen as an idealised form of the tip trace. Wing tip traces vary from insect to insect, and between flying regimes. See Ennos [38].

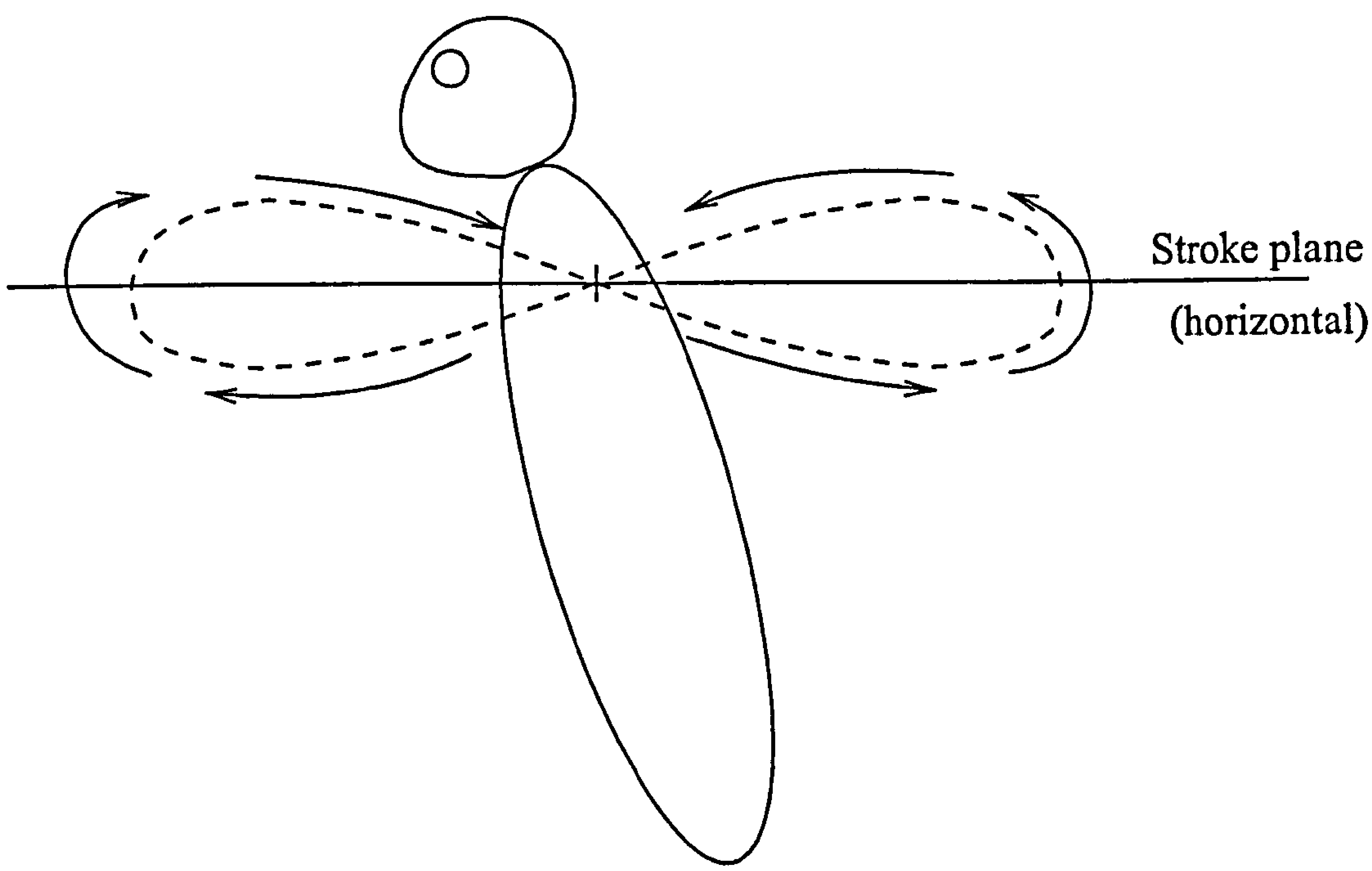


Illustration of flapping flight

Figure 3: A typical wing tip trace for the flapping motion of a wing

5.4 Lift generation

As stated in Section 5.2, insect wings tend to be very unlike conventional aerofoils, having very angular shapes and rough surface textures. This is because they operate at low Reynolds number, where viscous effects are far more dominant than for typical aerofoils. In such a viscous regime, the reduced friction drag due to a smooth surface is small, so smooth surface structure is far less important than for conventional-scale aerofoils. The major obstacle insects have to overcome is due to the reversing stroke: the Wagner effect.

5.4.1 The Wagner effect

The lift of an aerofoil is linked to its bound vorticity - the net rotation of the flow around the chord is the source of the lift. From inviscid theory, every time the bound circulation increases, an equal and opposite vortex will be shed. This also holds for viscous flow, although viscosity will cause the shed vortex to decay. In insect flight, the effect of these shed vortices is considerable, because they remain close to the wing for some considerable time. This is partly because the wings are flapping back and forth (thus returning to the space where the vorticity was shed), and because of the low velocity of the wings. The effect of these shed vortices is to oppose the increase in lift, and is known as the Wagner effect. The Wagner effect will cause attenuation of changes in the lift over time, and therefore a net loss of lift just after an impulsive start.

The above explanation of the effect will be examined in more detail during the wake modelling; for now, we simply observe that it happens, and consider some ways insects overcome it.

5.4.2 Overcoming the Wagner effect

The first method of overcoming the Wagner effect observed, was the so-called Weiss-Fogh mechanism [39], also known as the *clap and fling*, see Figure 4. With this mechanism, the insect avoids the Wagner effect by clapping the wings together, so the shed vortices from the two wings (which are of opposite sign) are brought together, and dissipate. Additionally, the wings are separated at the leading edge first, causing air to rush into the gap which causes the instant creation of a bound vortex. This effect is further enhanced by the bound vortex of the opposite wing. Some insects also rely on the variants the *Near clap*, and the *Clap and peel* (see [21]).

In the near clap, although the wings do not actually touch (or possibly only touch at their trailing edges), the shed vortex cancellation and bound vortex enhancement is still in effect, although not quite as pronounced as the full clap. This is often employed on the ventral (belly) side of the stroke, when the thorax obstructs clapping.

The clap and peel is a high aspect ratio version of the clap and fling - when the wings have clapped together, instead of rotating apart, the leading edges are pulled apart, and the surfaces peel apart from the leading edge. The rush of air into the gap between the wings acts similarly to the clap and fling.

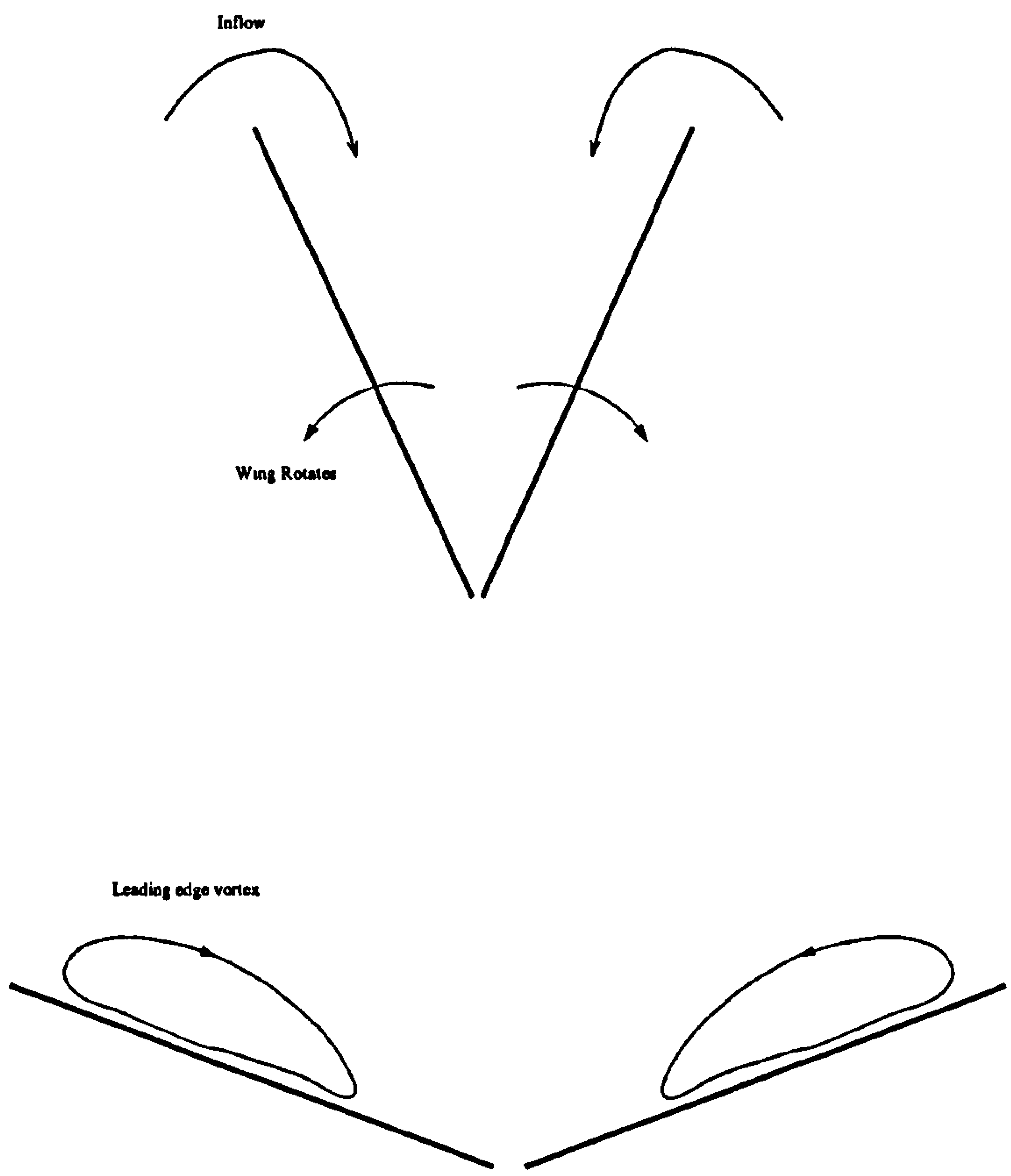


Figure 4: The Weiss-Fogh mechanism [39] for overcoming the Wagner effect. The wings are clapped together, then rotated apart leading edges first, so the flow of air into the gap between them "kickstarts" the LEV.

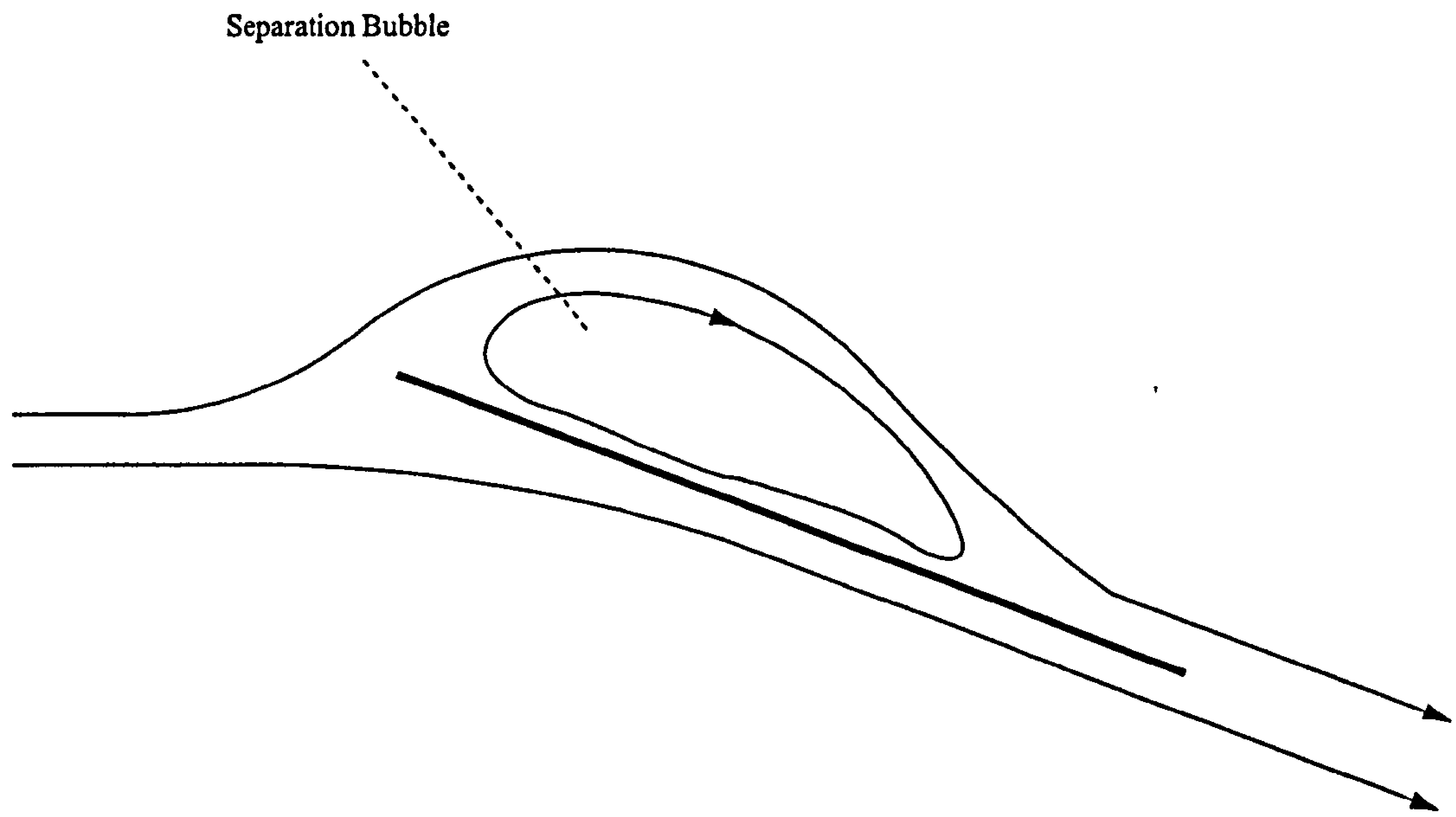


Figure 5: A flat wing with a separation bubble.

5.4.3 The leading edge vortex

This effect is worthy of special attention. Separation at the leading edge of a translating aerofoil will cause an attached separation bubble above the wing (see Figure 5). If instead of translating, the wing is rotating about a point, similar to a helicopter rotor, there will be a spanwise crossflow towards the tip, causing an outwardly spiralling helical flow. This will also occur if the leading edge is swept back, for example on a delta wing. This has been dubbed a Leading Edge Vortex (LEV) by Ellington, see [25], and can be seen in Figure 6. The effect of the LEV is to reduce the pressure above the wing, or alternatively can be considered as an increase in the bound vorticity of the wing through vortical lift [40]. The LEV was observed experimentally by Ellington et al on a rotor rig [9] where it was noted that the rotor could maintain a stronger steady LEV than the separation bubble of a translating aerofoil, due to secondary radial flow that forces the growing LEV off the tip of the aerofoil before it bursts, and becomes a deep stall. They have deduced four likely mechanisms for this radial flow - induced velocity due to the conically shaped LEV, low pressure due to higher speeds at the tip, centrifugal force of the mass of air trapped in the LEV and the sweep of the leading edge.

5.5 Types of insect flyer

Insects with low aspect ratio wings, such as butterflies, rely heavily on unsteady mechanisms, and the various clap mechanisms, while high aspect ratio wings, such as those of dragonflies rely more on conventional aerodynamics and the LEV. As discussed earlier, survival affects flight a great deal - thus, migratory insects such as locusts are built to move at high speed over long distances - they rely partly on forward airspeed for their lift, and cannot

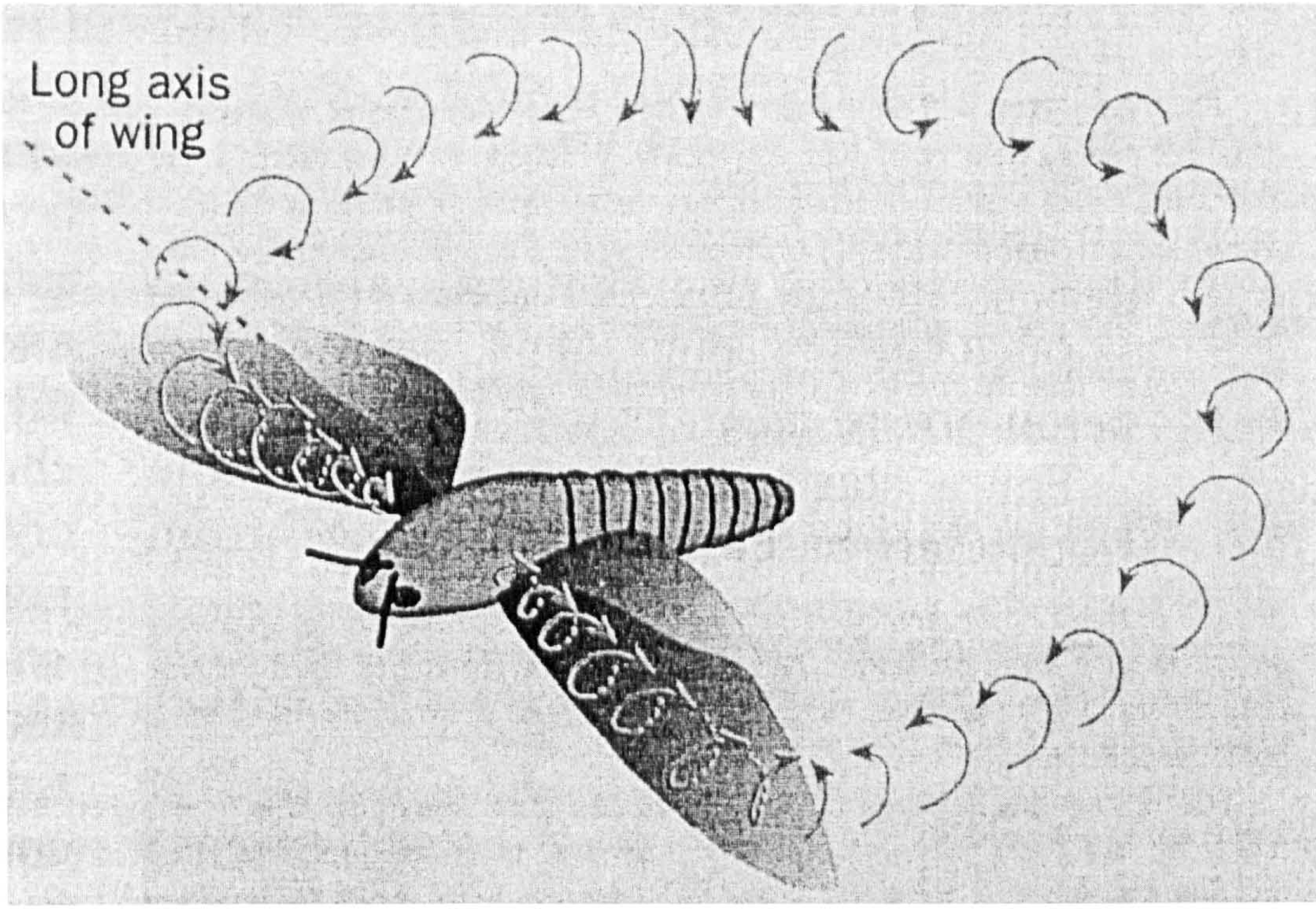


Figure 6: An illustration of the LEV on a wing, from [41]

hover. Hovering insects tend to use horizontal stroke planes. They incline their stroke plane in forward flight, analogous to helicopters.

5.6 Biomimetics

Firstly, a disclaimer is due to the biological community: the above remarks on insect flight are generalisations. For every statement above, a sharp-eyed biologist will be able to think of at least one species that is an exception. These generalisations have been made in order to arrive at a reference idealisation, representative of the problem.

The lower limit of the FMAV size may well be set not by performance merit, but by what can realistically be built and tested, and the scale of secondary components like visual sensors. Although MEMS technology has pushed the lower limit of scale downwards, Wootton has recently commented that work performed at Exeter University on fly wings has shown the wing membrane varies its properties continuously across the surface, by varying the structure on a molecular level. This is not a feat we can match.

The FMAV builder will, however, have access to rotary bearings, stiffer and stronger materials, and the major advantage of spare parts. The ability to design for a limited lifetime of certain high-stressed components that are simply replaced in not available in nature.

Nature has relatively power-dense, but not particularly efficient motors, yielding up to 200 W/kg at 10% efficiency. The effective energy density of their fuel is high. Note again, that this is a non-flying evolution - the "wasted" energy goes towards heating the insect, which is critical to its survival.

FMAV builders will have access to the same power density of motors, but not the same effective energy density of fuel, especially if battery power is used. Since sensors and on-board logic will require electrical energy, battery power is seen at the most feasible option, as opposed to a split fuel/battery combination for a combustion engine and the onboard electronics. Happily, the FMAV flight time is less of an issue - it will only need fuel for a single mission, plus a safety margin, and will never have to forage for its own supplies. Nonetheless, overcoming the energy storage problem of a half to full hour flight is considered one of the main challenges in FMAV design.

5.7 Which mechanisms to use

For the purpose of the present work, all of the above observations on insect wing shape and structure have to be set aside. The wing *has* to be treated as rigid for any sort of sensible first-order model to emerge. Restricting the analysis to hovering flight has been done for the same reason.

All the clap mechanisms are considered unusable, due to the high mechanical wear. The near-clap is a possibility, but considered dangerous as poorly controlled stroke amplitude or wing rotation will bring the wings into contact. Also, these are highly unsteady effects, and therefore difficult to model. The analysis has been restricted to more readily solvable aerodynamics, with a view to improving the modelling later.

The LEV is the most immediately promising mechanism, in that it has been show to work on full-scale aerofoils.

5.8 Conclusions on insect flight

- Insects use a number of high-lift mechanisms not available to conventional aerofoils, because insect wings reverse direction and pitch quickly.

- The high-lift mechanisms used above are necessary because a flapping wing would otherwise spend a lot of time at aerodynamically ineffective states, such as rotating at low velocity, or trying to overcome the Wagner effect at the start of each stroke.

- Many of the performance criteria scale unfavourably with size. This is revisited in section 12. The lower limit of the FMAV size is expected to be set by what can realistically be built, rather than by aerodynamic merit.

- Insect wings are not streamlined aerofoils: they more closely resemble sails.

- Insect wings are thin and deformable, with most of the mechanical strength towards the leading edge.

- Insect wings are not highly tuned to optimal operation at a single flight condition, but have robustness to changing conditions and often have evolutionary pressures not related to flying efficiency.
- The LEV is seen as an exploitable mechanism, partly because it has been shown to work up to conventional aerodynamic scale and Reynolds number, albeit with some variation due to Re effects.

6 Terminology

6.1 Flight regime terms

6.1.1 Flapping flight

Standard aerodynamics refer to a *flapped aerofoil* and an aerofoil with a flap: a part of the wing which is at an angle to the remainder, such as the aileron control surfaces on the trailing edge of the wing. This should not be confused with the term *flapping aerofoil*, which is simply an aerofoil undergoing a "flapping" motion. Flapping flight is where the wing translates in one direction, then comes to a halt while rotating (pitching), and translates in the opposite direction. The rotation at either end of the stroke means the leading edge is always ahead of the trailing edge in the direction of travel.

6.1.2 Reversal

Reversal is the term for when the direction of the wing's translation reverses—i.e. it stops translating briefly, and returns the way it came. Note that the wing may still be pitching during this time.

Specifically, reversal is defined for a point on a flapping aerofoil as the state where the horizontal velocity reverses sign. Note that since the wing may still be pitching, reversal does not occur simultaneously for all points on the wing.

More generally, the concept of "reversal" is referred to as the fact that the wing is flapping back and forth in the same space, as opposed to constantly translating forwards.

6.1.3 Cycles and strokes

When the vehicle is not manoeuvering, the flapping motion of the wing is a repeated cycle of motion. Each *cycle* is defined as a single, closed path of motion. The cycle is made up of two *strokes*, that start and end at the extremes of motion (the reversal). The kinematics during the two strokes need not be similar.

6.1.4 Stroke plane

Since the wing motion during the two strokes of a cycle is not necessarily similar, the stroke plane is defined as the mean line between the tip motion during the two strokes. For both the cases considered here, the stroke plane is horizontal.

6.1.5 Rotating and translating regime

Because of the reversal above, the wing will be in either one of two identifiable regimes: The translating regime, in the middle of every stroke, where the translational velocities are high, and the rotational velocities are low, and the rotational regime at either end of the stroke, where the translational velocities will be low, and the rotational velocities high. These are not hard-delimited regimes, but lead gradually into each other.

6.1.6 Upper and lower surface

Since the wing is capable of flipping entirely upside down, care has to be taken about which surface is being referred to. The *upper* surface is the surface that is upwards when the wing has its leading edge pointing forwards. This surface retains the name whatever the orientation of the wing. The other surface will be referred to throughout as the lower surface.

6.2 Aerodynamic terms

6.2.1 Angle of attack

The angle of attack is defined as the angle between the mean chordline (the line from leading to trailing edge) and the free stream flow. It is not used in this investigation, because 1) Wing rotation causes α to vary along the chord 2) α is mainly used when set ≈ 0, to express the normal velocity.

Instead, the following calculations are performed as a function of the normal velocity directly, with no reference to α. The wing attitude is obtained from the pitching angle β, which is defined from geometry, and therefore independent of the free stream. For two examples of the relationship between α and the pitching and plunging motions, see Figures 7 and 8.

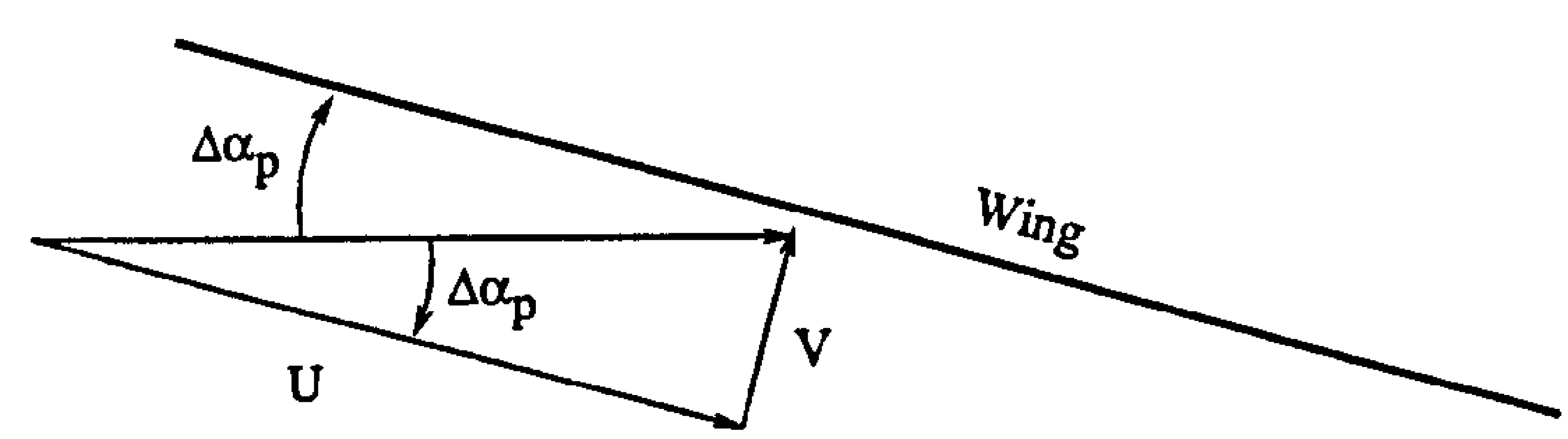
6.2.2 Advance ratio J

This comes from helicopter aerodynamics [3], where it is typically denoted by μ. It is the ratio of forward airspeed to the tip velocity of the rotor. This parameter can be used, unchanged. However, care must be taken since J values for helicopters and FMAVs are not directly comparable. Specifically, since the two wings of an FMAV are moving in phase (both moving forwards at the same time), they are not restricted by an upper limit of J. Helicopters are typically restricted to $J < 0.4$, because their wings are in antiphase: the forward velocity of the helicopter adds to the airspeed of the rotor on one side, but is subtracted from the airspeed on the other side, causing large rolling moments. Of more use to this application is the *Hovering velocity ratio*, which is the ratio between the average induced velocity and the r.m.s. velocity of the wing. A low hovering advance ratio means that the effect of the wake can be treated as stationary in time, since it changes slowly compared to the velocity of the wing.

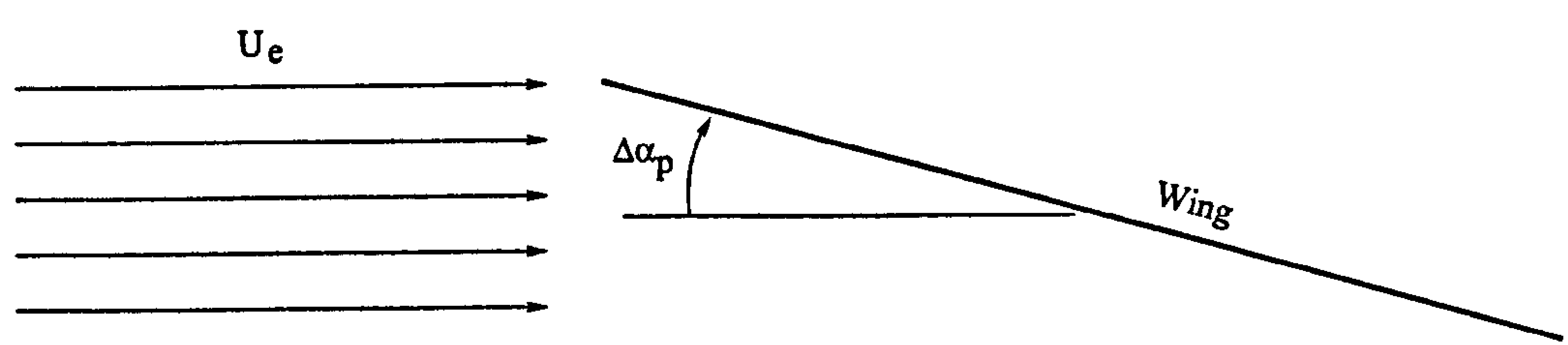
6.2.3 Reduced frequency k

The reduced frequency is defined as $frequency * characteristic\ length\ /\ velocity$. It is typically used to relate the spatial variation of a property to the temporal variation, or express the degree of unsteadiness in a flow. For most typical applications, this uses the chord of the wing as the length parameter and the forward velocity to relate the frequency of a variation at the wing (e.g. in bound circulation) to the wavelength of that variation. In the context of insect-like flapping, this is not an applicable parameter, because it is not constant. Most uses of k are to map a sinusoidal variation in time to a sinusoidal variation in space. Since k is

(a) Actual Situation

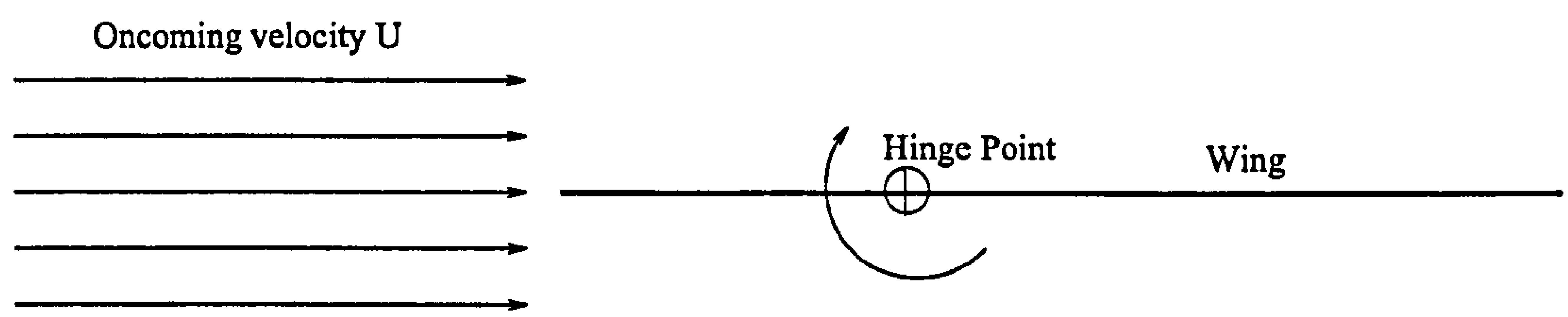


(b) modelled geometry

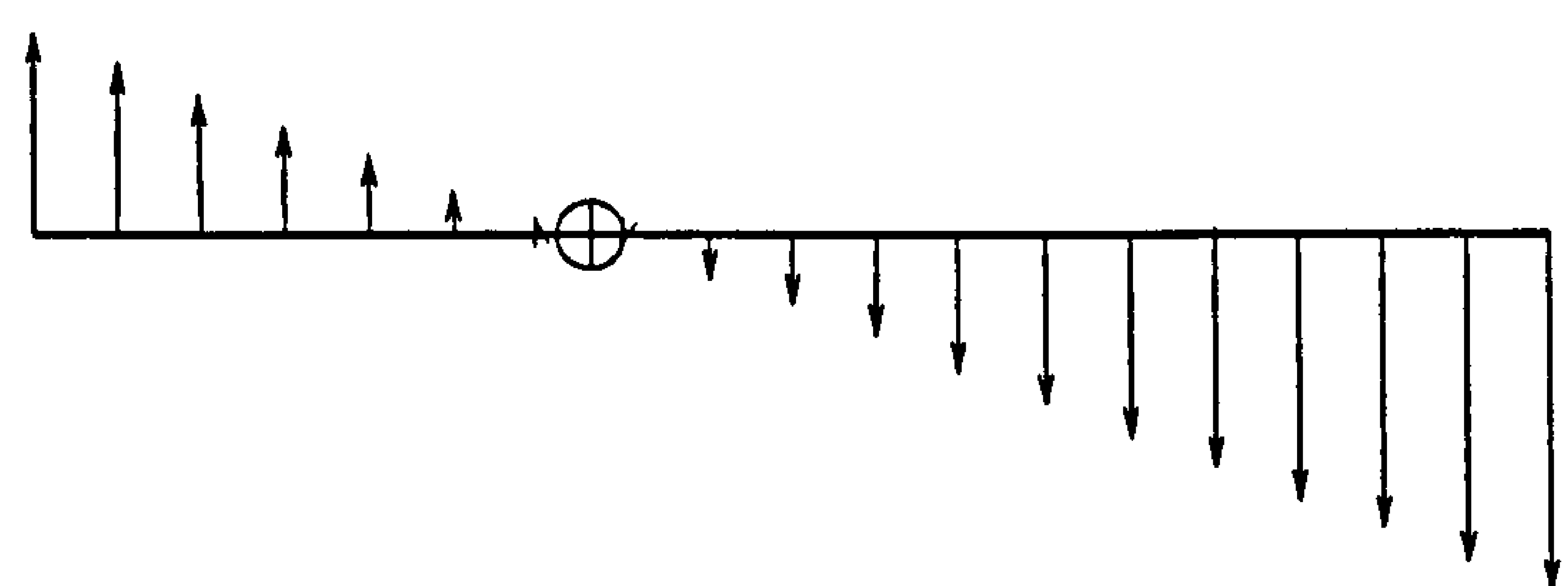


(c) equivalent airflow

Figure 7: Illustration of the effect of plunging velocity correction on α



(a) Actual Situation



(b) pitching velocity due to rotation

Figure 8: Pitching effect on normal velocity, and hence α

not constant, this mapping will not be to a sinusoid. Also, all points of a flapping wing will, at some point, have zero horizontal velocity—at these points k goes to infinity, so it is not even possible to place appropriate bounds on the values of k.

7 Definitions

7.1 Position

7.1.1 Rectangular coordinates

A right-hand body-aligned coordinate system x, y, z in metres is defined, as shown in Figure 9. The vertical z axis is always downwards, and the origin is fixed to the root of the right wing.

7.1.2 Spherical coordinate system

A spherical coordinate system is defined, based on the wing position angles θ, ψ, and the radius r, which is normalised with respect to the tip radius. The radial position r is defined along the *hinge line*—the line from the root to the point on the wing furthest from the root. r is normalised with respect to the wing tip radius R. The chord line is defined normal to the hinge line. This is shown in Figure 9. Horizontal motion (increasing θ) is called *sweeping motion*, vertical motion (increasing ψ) is called *plunging* motion, and wing pitching (increasing β) is called *pitching* or *rotating* motion.

7.1.3 Wing-fixed coordinate system

A wing-fixed coordinate system is defined, at a section of wing with constant r. Here, we define the chordwise ζ coordinate and normal η coordinates, both normalised with respect to the local semichord b of the wing.
The origin of ζ is at midchord, so it is -1 at the leading edge, and $+1$ at the trailing edge. The origin of η is the midchord of the wing, positive towards the "upwards"side of the wing, as shown in Figure 10. Note that η is 0 everywhere on the wing, and therefore rarely used.

7.1.4 Wing sections

The following analysis deals with the wing using $2D$ analysis on individual sections of the wing, and integrating across sections in a spanwise direction. A sample section is shown in Figure 11.

7.1.5 Hinge line

The wing rotates about a single point, at the root of the wing, and the origin of the rectangular x,y,z coordinate system. This point is called the *hinge*. The *hinge line* is the straight line connecting the root and the tip of the wing. This is the assumed pitching axis of the wing at all times. For any given spanwise section, the hinge location a is defined as the chordwise position of the hinge line at that section. The hinge location is normalised with respect to the local semichord b, so has values from -1 at the leading edge to 1 at the trailing edge, as illustrated in Figure 10.

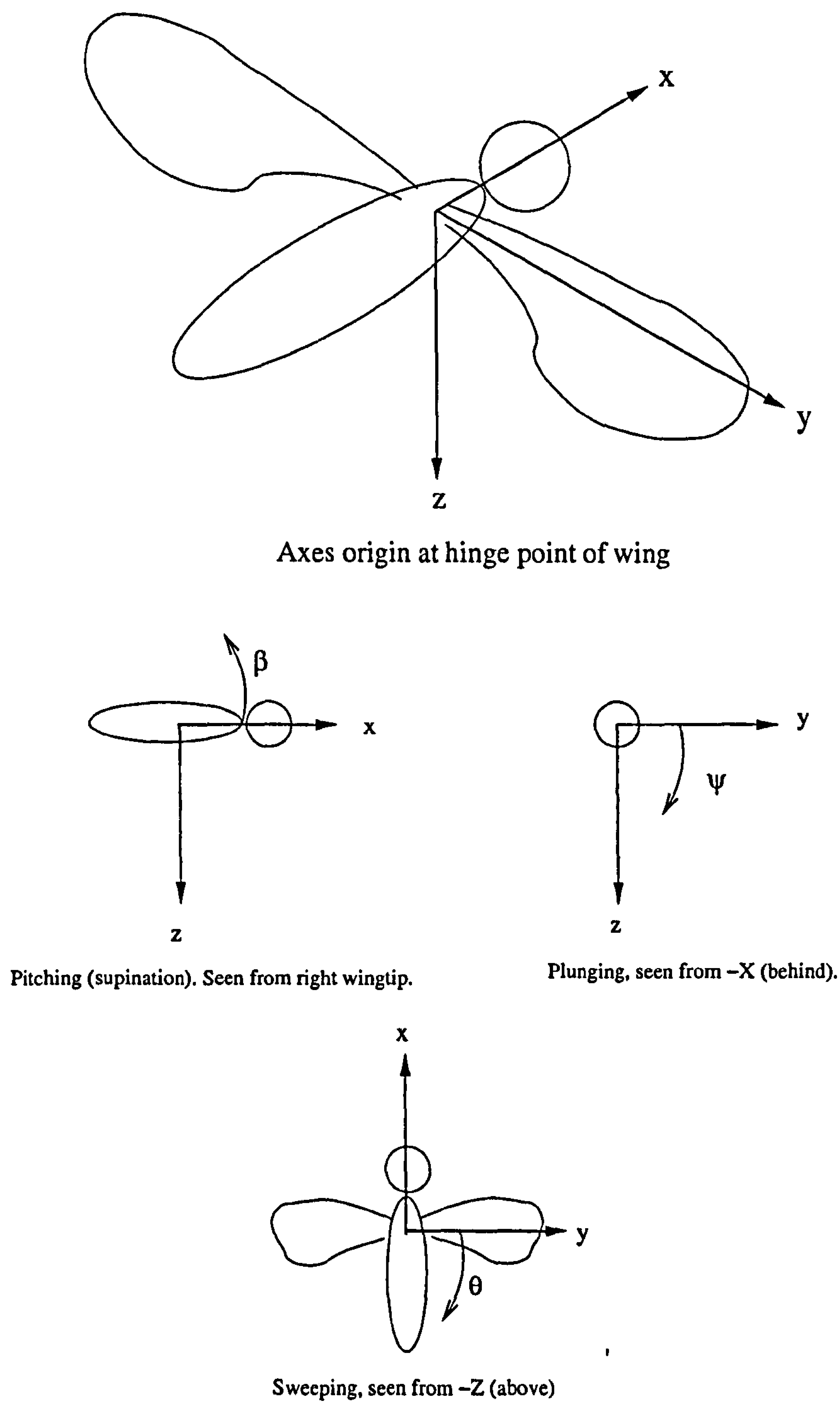


Figure 9: Coordinate system.

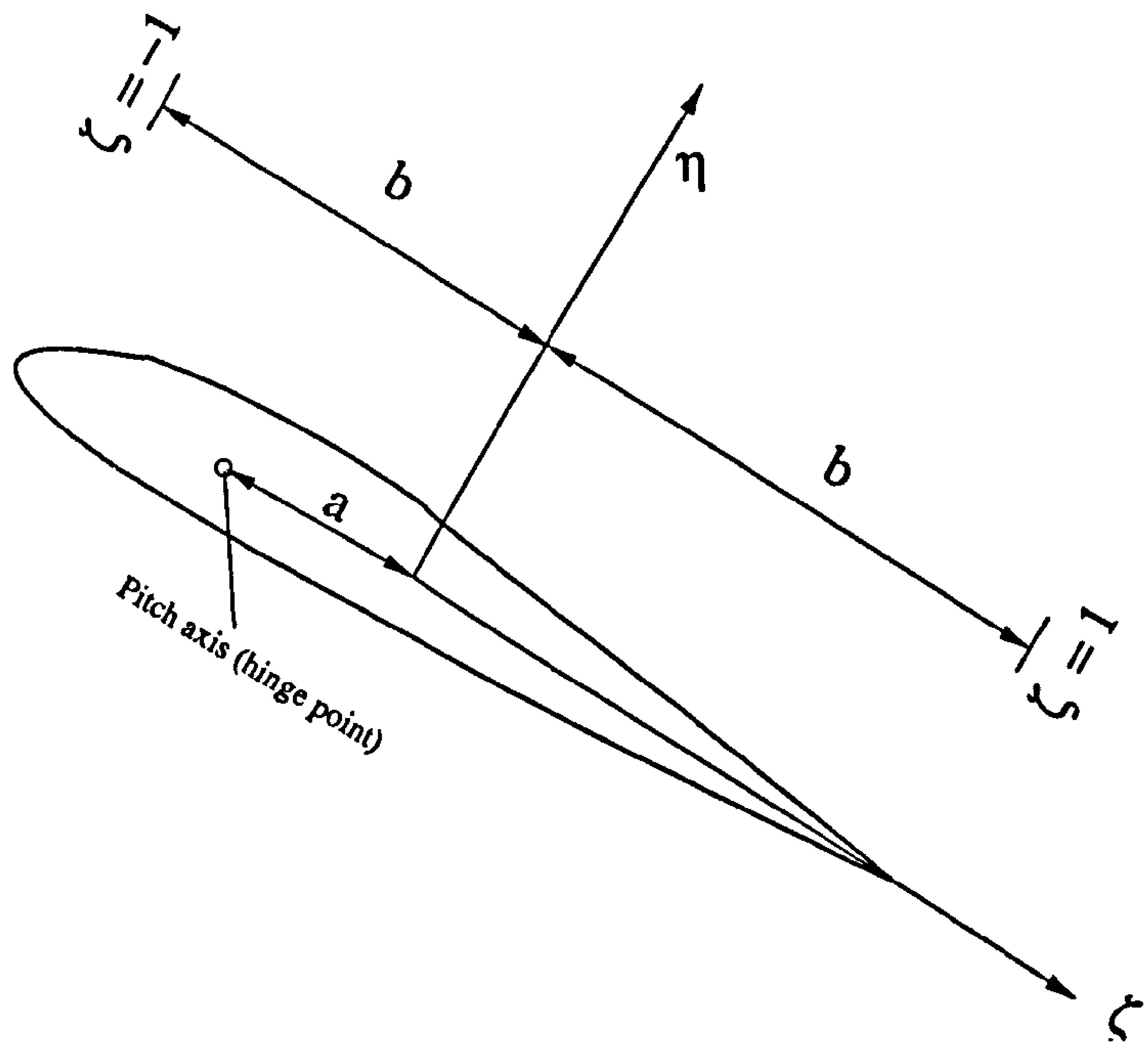


Figure 10: The wing-fixed coordinate system.

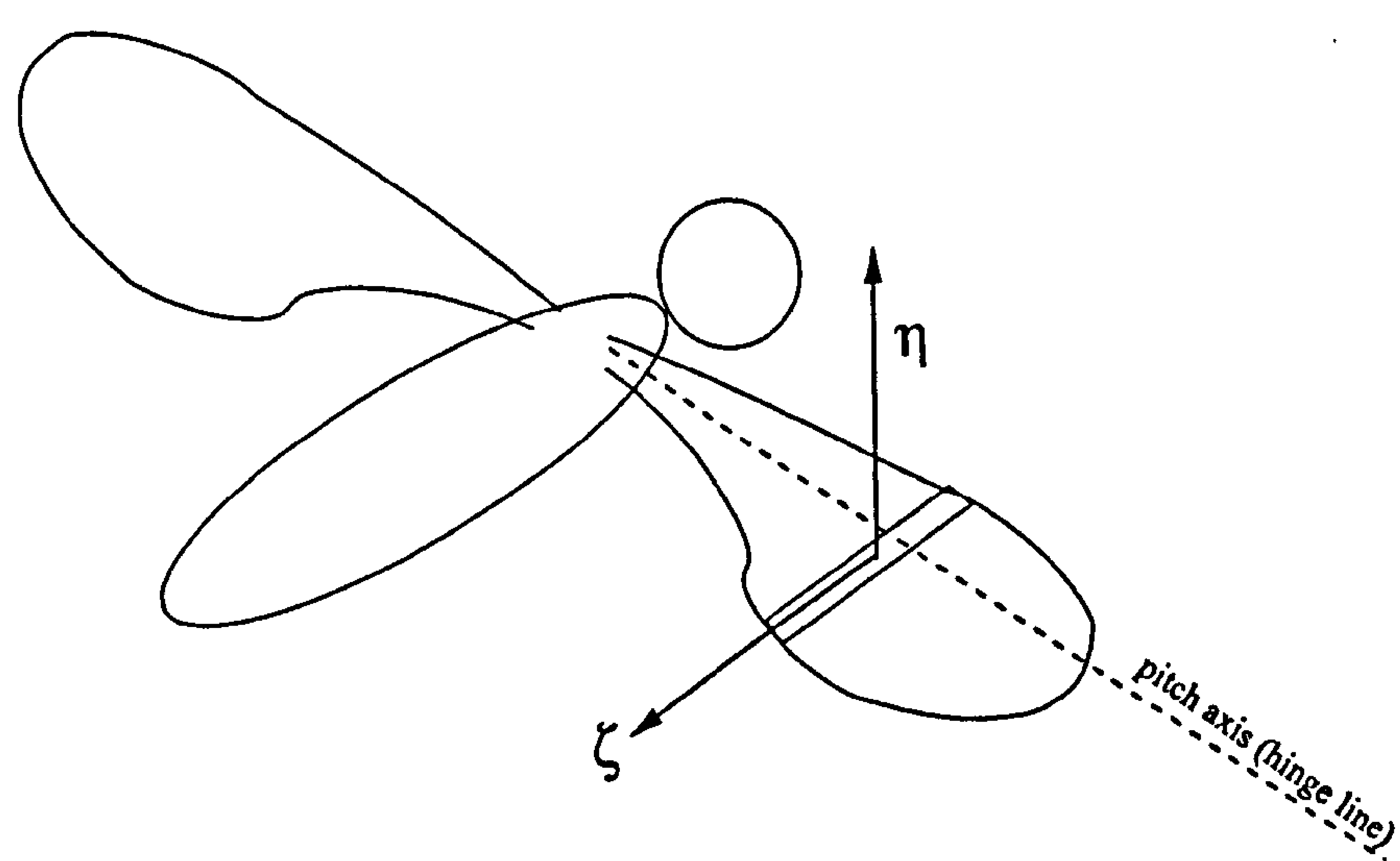


Figure 11: A sample wing section.

7.2 Velocities

Velocities are written in metres per second, as the velocity of the fluid relative to the wing, in the general form: u_{NlT}

The first subscript is the direction of the velocity:
P is in the wing fixed system, parallel to the wing, towards the trailing edge.
N is in the wing fixed system, normal to the wing, towards the "upwards" side.
V is in the spherical coordinate system, upwards (i.e. the $-\psi$ direction.)
H is in the spherical coordinate system, backwards (i.e. the $+\theta$ direction.)
T is the total velocity, in either coordinate system.
See Figure 12 and 13 for illustrations of these directions.

The second subscript is the chordwise location of the velocity:
l is at the leading edge.
t is as the trailing edge.
m is at the midpoint.
r is at $3/4$ chord, called the *rear neutral point.*
If this subscript is omitted, the velocity is assumed to be at the hinge point.

The third subscript is the spanwise position of the velocity:
T is at the tip of the wing, assumed on the hinge line.
If this subscript is omitted, the velocity is assumed to be at a radial position r.

Special case:
The velocity u_i is the average downward velocity induced by the lift of the wing, it is positive downwards, i.e. the $+z$ direction.

7.3 Forces

Forces are written in Newtons, the general form: $F_{NA\ TD\ W}$

The first subscript is the direction of the force:
P is in the wing fixed system, parallel to the wing, towards the leading edge.
N is in the wing fixed system, normal to the wing, towards the "upwards" side.
V is in the spherical coordinate system, upwards (i.e. the $-\psi$ direction.)
H is in the spherical coordinate system, forwards (i.e. the $-\theta$ direction.)
L is in the rectangular coordinate system, upwards (i.e. the $-z$ direction.)
D is in the rectangular coordinate system, forwards (i.e. the $+x$ direction.)
These direction are shown in Figures 14 and 15.

The second subscript is the cause of the force:
Q is quasi-steady.
A is added mass.
P is Polhamus effect.
W is Wagner (primary wake).
K is Küssner (secondary wakes).

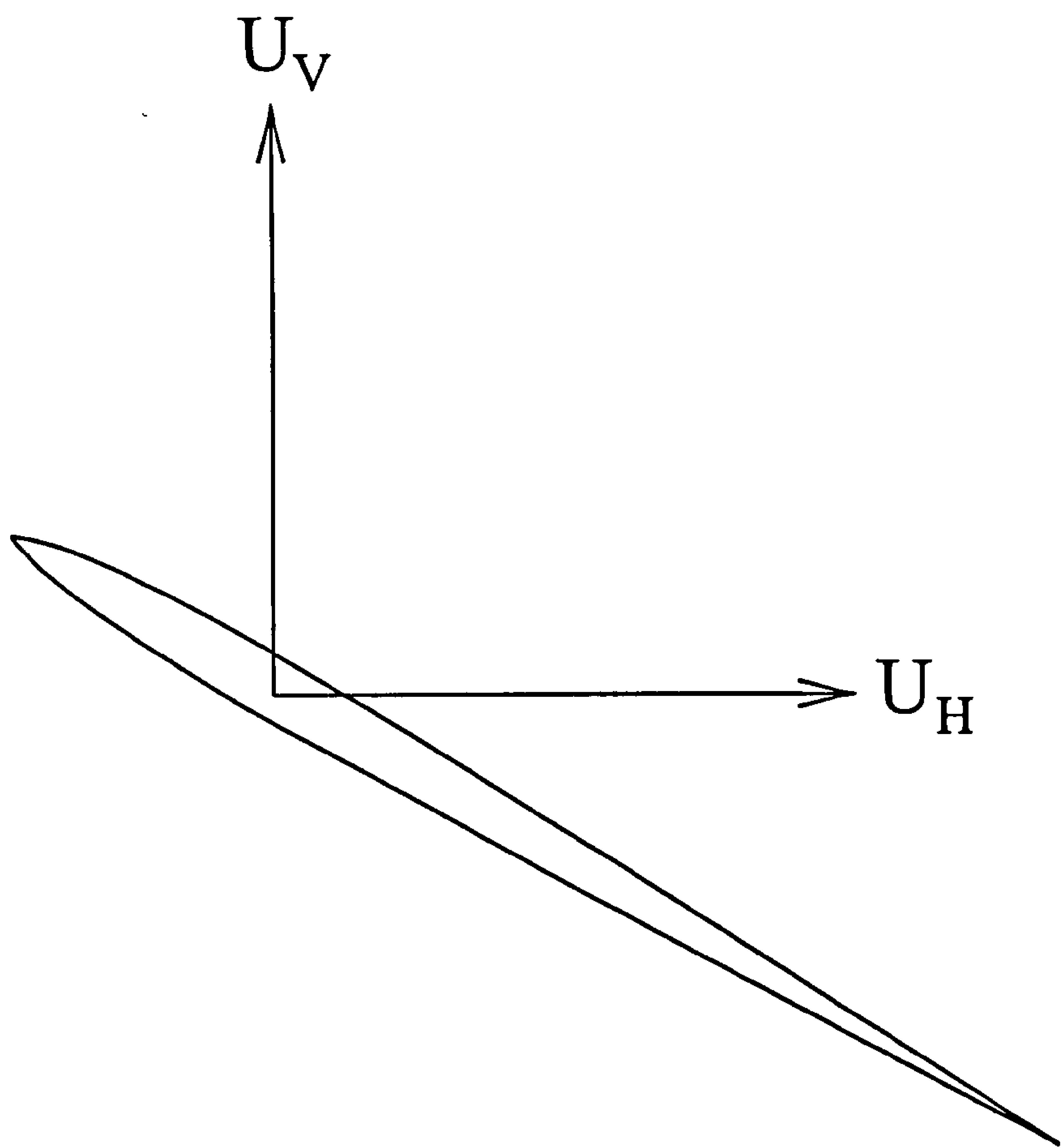


Figure 12: Direction of the horizontal velocity u_H and the vertical velocity u_V, as seen from the root of the wing. These are defined as velocity of the fluid relative to the wing, in the spherical coordinate system.

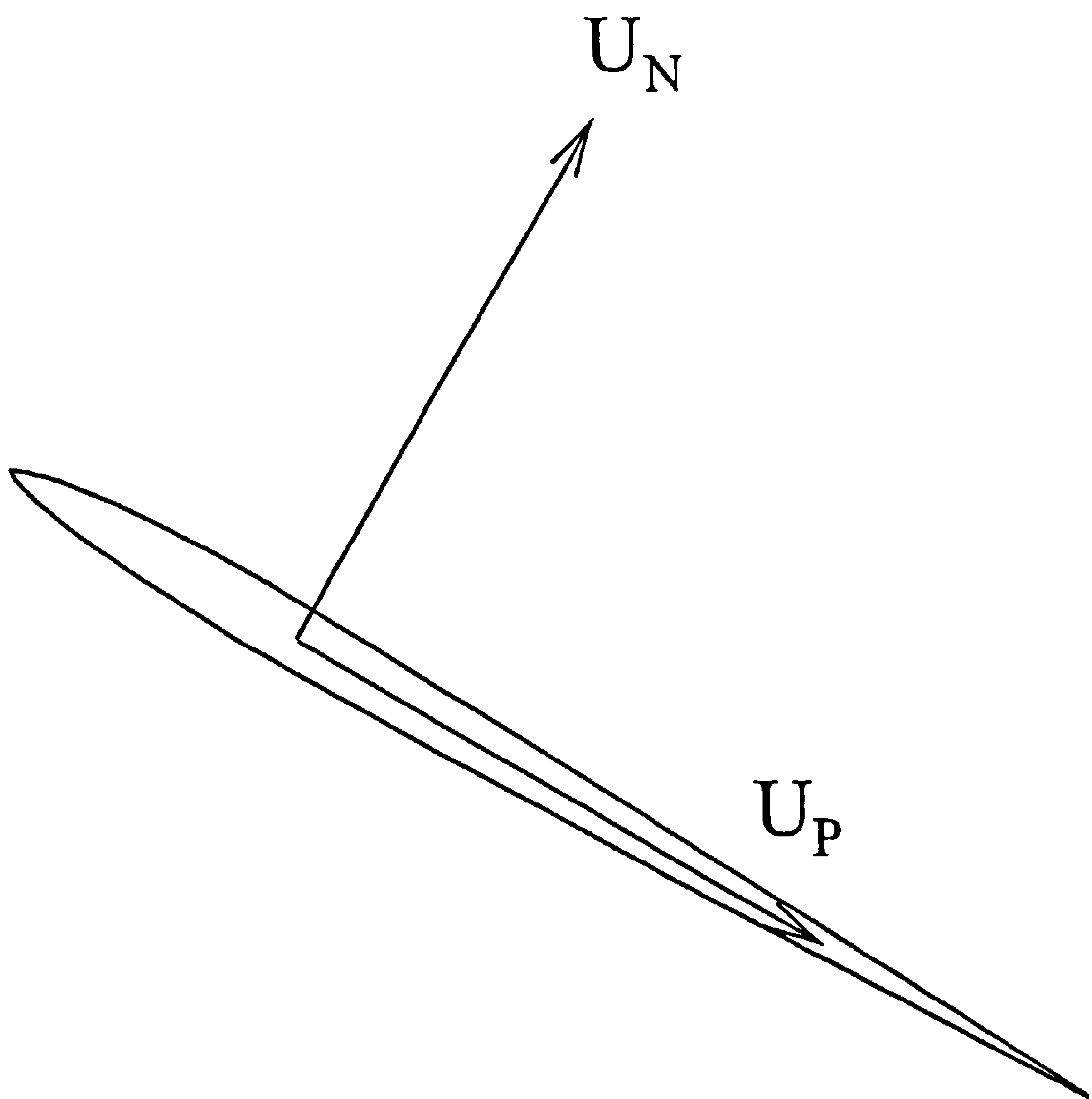


Figure 13: Direction of the normal velocity u_N and the parallel velocity u_P, as seen from the root of the wing. These are defined as velocity of the fluid relative to the wing in the wing-fixed coordinate system of Figure 10.

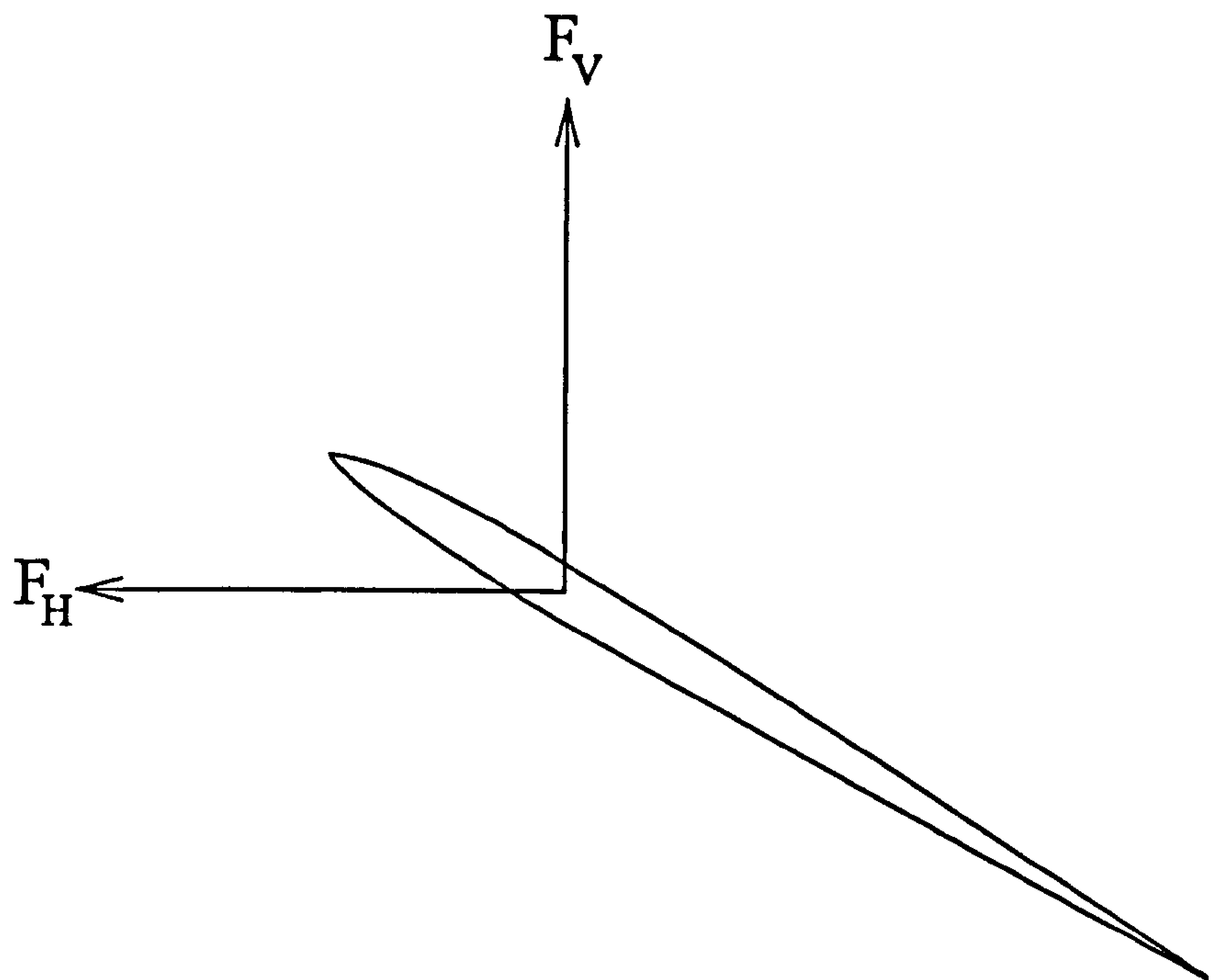


Figure 14: Direction of the horizontal force F_H and the vertical force F_V, as seen from the root of the wing, in the spherical coordinate system.

If this subscript is omitted, the force is assumed to be the sum of all the above contributions.

The third subscript (one or two letters) is the component of the contribution, for the quasi-steady and added mass terms only.
TD is the translational component of the Dirichlet solution.
RD is the rotational component of the Dirichlet solution.
TK is the translational component of the Kutta-Joukowski correction.
RK is the rotational component of the Kutta-Joukowski correction.
D is the total Dirichlet solution.
K is the total Kutta-Joukowski correction.
Typically, this subscript is omitted, in which case the force is assumed to be for the total contribution of all components.

The fourth, optional, subscript denotes the area of integration:
W means the force is integrated over the entire wing. If this subscript is omitted, the force is assumed to be per metre span.
Note that in the following the shorthand "lift" is used for the vertical force F_V and "drag" for the horizontal force F_H.

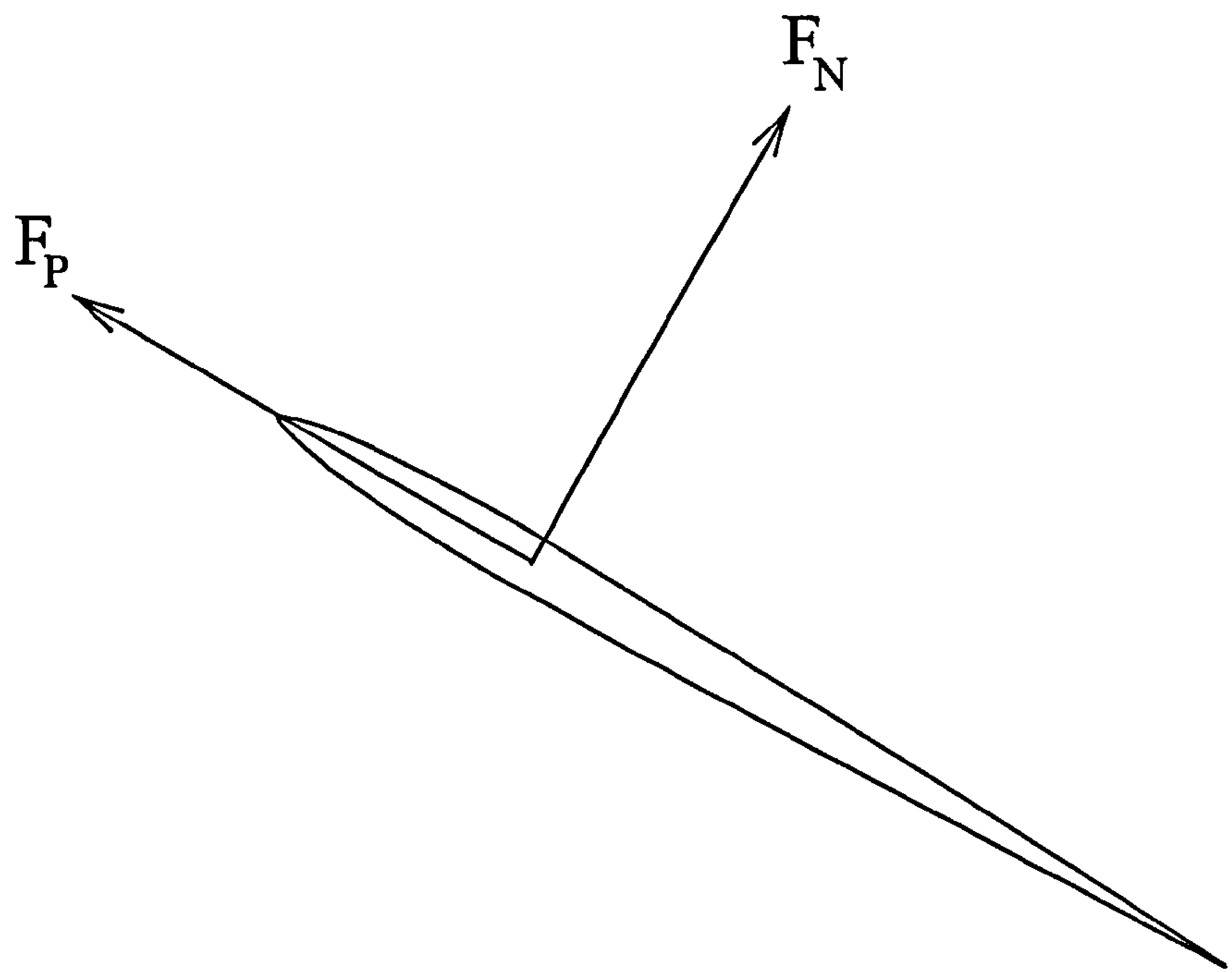


Figure 15: Direction of the normal force F_N and the parallel force F_P, as seen from the root of the wing, in the wing-fixed coordinate system of Figure 10.

7.4 Moments

Moments are written in Newton-metres, in the general form: $M_{VA\ TD\ W}$
The subscripts are similar to those for forces above, except the first, where the only cases are:
V Moment about the x axis, positive in the $-\psi$ direction (upwards).
H Moment about the z axis, positive in the $-\theta$ direction (forwards).
P Pitching moment about the hinge line, positive in the $+\beta$ (pitching up).

Note that the descriptions of direction ("upwards", and so on) are local to the right-hand wing, which is the only one considered.

7.5 Other definitions

7.5.1 Downwash velocity u_i

Any lift generation causes a downwash velocity, as is readily apparent from simple momentum considerations. The Rankine-Froude theory for an actuator disc assumes a constant downwash velocity u_i across the swept disc of a propeller. This is available in any good

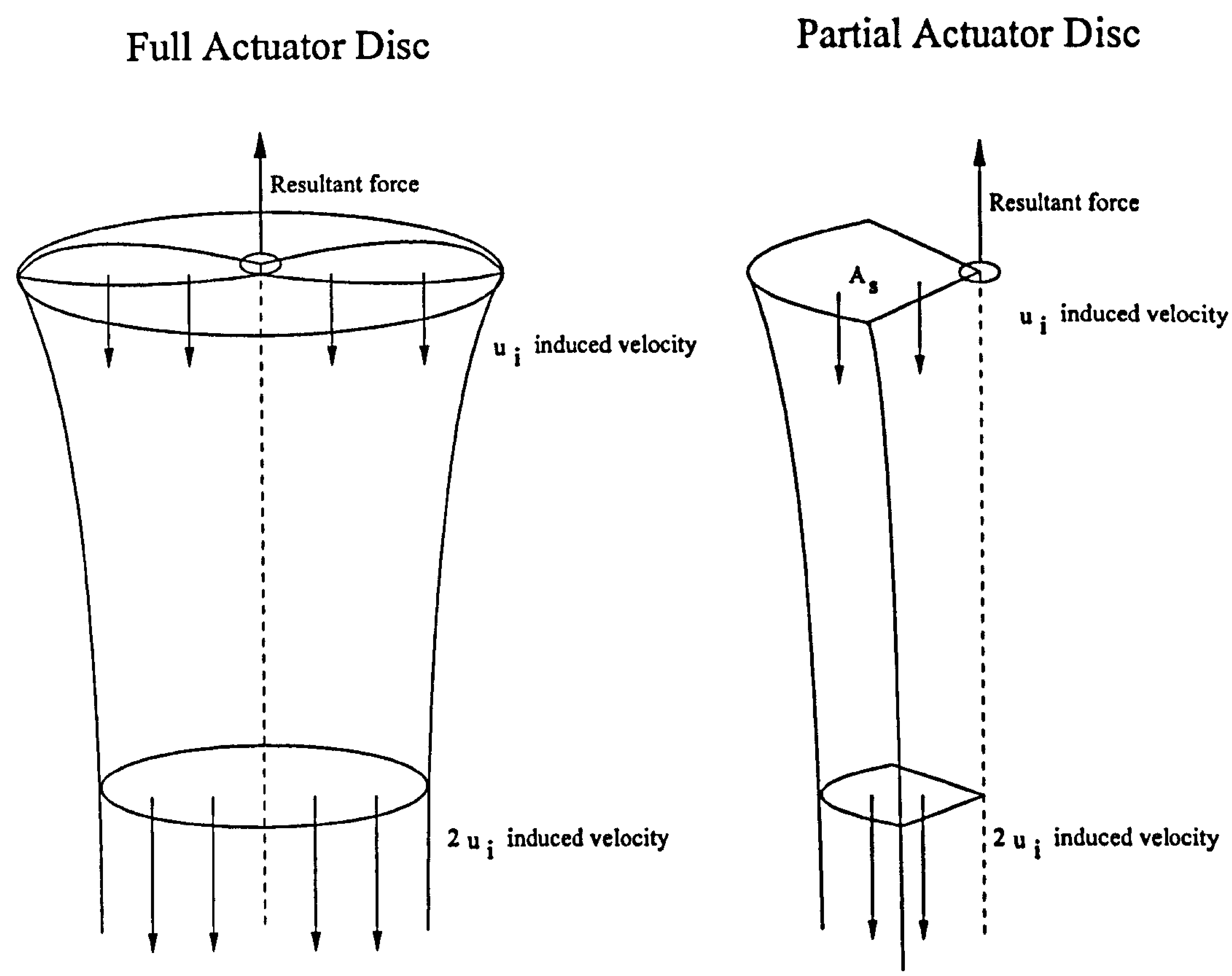


Figure 16: Swept area used to determine induced velocity, based on momentum theory, after Ellington [22]

textbook on aerodynamics, and gives:

$$u_i = \sqrt{\frac{\bar{F}}{2\rho\, A_S}}, \tag{1}$$

in hover, where $\bar{F}$ is the average lift force, and A_S is the swept area of the propeller - the area of the actuator disc. For the flapping case, where the wing does not perform full revolutions, it is more appropriate to calculate u_i based on the area that is actually swept by the wing, rather than the full circle. See Figure 16 for an illustration of this.

7.6 Basic identities

On the basis of the above definitions, some basic identities are available:

$$u_H = -R\, r\, \dot{\theta} \tag{2}$$

$$u_V = R\, r\, \dot{\psi} \tag{3}$$

$$u_N = u_H\, \mathrm{S}_\beta + u_V\, \mathrm{C}_\beta \tag{4}$$

$$u_P = u_H \, \mathrm{C}_\beta - u_V \, \mathrm{S}_\beta, \tag{5}$$

see Figure 9 on page 23. Note the minus sign in the first equation—remember that velocity is defined in terms of the free stream velocity relative to the wing, so the velocities above are the opposite to those of the wing in still air. The last two equations are simply from resolving velocities in the spherical coordinate system to the wing-local system.

The above can be used to find the velocities at local points of the wing. Subscript E is used for an arbitrary point on the wing. The local wing semichord is b, and the hinge location in wing-fixed coordinates is a:

$$u_{PE} = u_P \tag{6}$$
$$u_{NE} = u_N + b\,\dot{\beta}\,(\zeta - a) \tag{7}$$
$$u_{Nl} = u_N + b\,\dot{\beta}(-1 - a) \tag{8}$$
$$u_{Nm} = u_N + b\,\dot{\beta}\,(-a) \tag{9}$$
$$u_{Nr} = u_N + b\,\dot{\beta}\,(\frac{1}{2} - a) \tag{10}$$
$$u_{Nt} = u_N + b\,\dot{\beta}\,(1 - a) \tag{11}$$

Note the first equation: when using the wing-fixed coordinate system, the rotational velocity will manifest itself purely as a normal component. The last four equations are just special cases of Equation 7. Also, note that all velocities at the hinge will scale linearly with the radius, so:

$$u = r\,u_T \tag{12}$$

for all velocities at the hinge.

Some basic identities can be formed for forces by resolving between coordinate system.

$$F_V = F_N \, \mathrm{C}_\beta + F_P \, \mathrm{S}_\beta \tag{13}$$
$$F_H = -F_N \, \mathrm{S}_\beta + F_P \mathrm{C}_\beta \tag{14}$$
$$F_N = F_V \, \mathrm{C}_\beta - F_H \, \mathrm{S}_\beta \tag{15}$$
$$F_P = F_V \, \mathrm{S}_\beta + F_H \, \mathrm{C}_\beta \tag{16}$$
$$F_L = F_V \, \mathrm{C}_\psi \tag{17}$$
$$F_D = F_H \, \mathrm{C}_\theta, \tag{18}$$

where all the above are found by resolving between coordinate systems. Note that the last two equations are the forces experienced by the body, in the rectangular coordinate system. Also the definition of "drag" makes it always positive forwards, and spanwise forces have been ignored completely. In the spherical coordinate system, the model used predicts no spanwise force (see the Polhamus model in Section 10). In the rectangular coordinate system, any spanwise force caused by one wing is assumed to be cancelled by the opposite wing.

Part II
Aerodynamic model

In this part, the core of the thesis, a theoretical aerodynamical model for the forces and moments on the wing is derived, using $2D$ thin aerofoil potential theory. The derivation of the quasi-steady forces in Section 8, and the added mass forces in Section 9, is similar to the standard form of unsteady aerodynamics, but without the assumption of fast forward motion. This relaxation introduces extra terms into the expressions for aerodynamic forces.

The flow around the sharp leading edge is modelled using the leading edge suction analogy of Polhamus, in Section 10. Briefly, this models the effect of the separation, and the attached vortex that is expected to occur on the upper side near the leading edge. The model assumes that any leading edge suction force is rotated through 90° to become an additional normal force.

The wake model of Section 11 treats the wake as a thin filament of vorticity shed from the trailing edge. In order to make this analytically tractable, some considerable simplifying assumptions are made, and a combination of simplified models for cases which *can* be solved is used. These simplified models are the Wagner and Küssner models of an arbitrarily accelerating and pitching aerofoil at low angle of attack, and the Loewy model for the downwashed wake under the wing. A refinement of the above method, based on the Polhamus correction from Section 10 is also described.

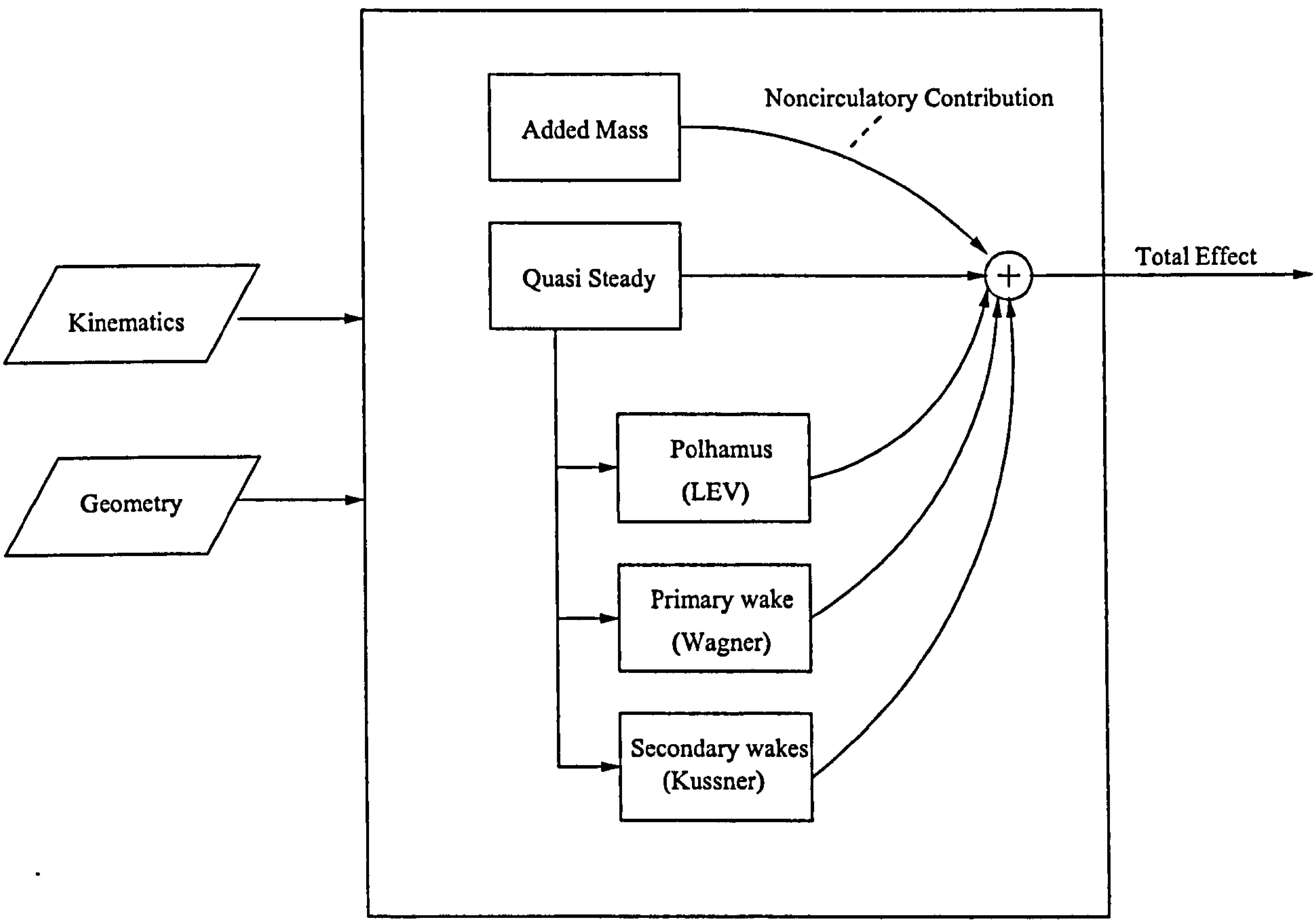


Figure 17: Model overview. Note that there is no iteration in the model above - the flow of information never forms a feedback loop. Effectively, it models a rigidly forced response - the kinematics of the wing are unaffected by the loading.

8 Quasi-steady effects

8.1 Potential theory

A 2-D potential model of the inviscid flow around a thin, flat aerofoil is used to form a complex velocity potential $\bar{\Phi}$, which has the property of differentiating to the velocity of the flowfield: $d\bar{\Phi}/d\bar{z} = u_P - iu_N$. For the case of a thin, flat plate, the potential is purely real, as a function of the wing coordinate ζ only: $\Phi(\zeta)$.

Some standard results of potential theory, from e.g. Katz & Plotkin [7] are:

1. The potential due to a bound vorticity γ is such that $\partial\Phi/\partial x = \gamma$.
2. The datum of Φ can be set arbitrarily.
3. Individual Φ for several flowfields can be superimposed, to give their combined effect.

Note also the following useful identity

$$\frac{\partial\Phi}{\partial\zeta} = b\gamma \tag{19}$$

Throughout the rest of the thesis, the quantity $Q = \sqrt{1-\zeta^2}$ is used extensively. Identities for integrals and differentials of Q can be found in Appendix A.

8.2 Dirichlet solution

The Dirichlet solution is the potential function needed to cancel out the component of the local free stream velocity normal to the surface of the wing, making the wing surface a streamline. It does this without contributing a net circulation to the flow. This is also the minimum energy solution to the problem, i.e. it is the solution that causes the least amount of kinetic energy to the fluid. This has been done in a variety of ways. von Kármán and Sears [42] directly wrote the bound γ needed. Theodorsen [28] formed the potential function from a set of source-sink pairs on the upper and lower surface of a unit circle, then used Joukowski mapping to map the circle to a line, where the source-sink pairs become doublets aligned normally to the wing. Finally Katz & Plotkin [7] wrote the expression for the doublet strength needed directly, then showed that this could be differentiated to give the bound vorticity.

Whichever method is used, the end result is a potential function split into two superposable parts: one for the translational motion, and one for the rotation about the hinge line (pitch axis). The potential on the upper surface is:

$$\Phi^+_{TD} = u_N bQ \tag{20}$$

$$\Phi^+_{RD} = \dot{\beta} b^2 \left(\frac{\zeta Q}{2} - aQ\right) \tag{21}$$

For this the ability to define the datum of Φ arbitrarily was used, so the potential on the upper and lower surface are exactly equal and opposite.

Φ^+ can be differentiated to give the bound vorticity:

$$\gamma^+_{TD} = u_N b \frac{-\zeta}{Q} \tag{22}$$

$$\gamma^+_{RD} = \dot{\beta} b^2 (\frac{1}{2} - \zeta^2 + a\zeta)/Q \tag{23}$$

Note here that the vorticity of the upper and lower surface are identical, not of opposite sign.

There are two singularities, at the leading edge and trailing edge. At these points γ and velocity becomes infinite, and Φ is discontinuous unless zero. This is dealt with in the next section.

8.3 Kutta-Joukowski condition

Kutta and Joukowski independently observed that the discontinuity at the trailing edge is equivalent to the flow passing around the trailing edge, experiencing infinite acceleration as it does. In a real fluid, the flow will be unable to do this, and will instead separate at the trailing edge. Satisfying the Kutta-Joukowski condition involves superposing a net bound vorticity onto the Dirichlet solution, so the flow leaves smoothly at the trailing edge. The correction required to satisfy this condition is referred to in this work as the Kutta-Joukowski correction. It is an empirically-inspired correction to the potential flow model, to make the flow behave like a real, viscous fluid. This additional vorticity should not cause any net normal flow anywhere on the wing, so it remains a streamline.

Again, this can be approached from the potential or vorticity perspective. von Kármán and Sears [42] write the expression for the vorticity needed to cancel the velocity at the trailing edge directly. Note, however, that their solution includes the vorticity of the shed wake, which will be dealt with as a separate effect in the model. Theodorsen [28] uses a uniform distribution of vorticity about a unit circle, of sufficient strength to cancel the Dirichlet potential at the trailing edge, then maps this to a line. Katz & Plotkin [7] write the vorticity needed directly.

For the wake-free case, the latter two methods give expressions for potential and vorticity:

$$\Phi^+_{TK} = u_N b\,(\mathrm{asin}(\zeta) - \pi/2) \tag{24}$$

$$\Phi^+_{RK} = \dot{\beta} b^2 \left(\frac{1}{2} - a\right) (\mathrm{asin}(\zeta) - \pi/2) \tag{25}$$

$$\gamma^+_{TK} = u_N b/Q \tag{26}$$

$$\gamma^+_{RK} = \dot{\beta} b^2 \left(\frac{1}{2} - a\right) /Q, \tag{27}$$

where the expressions have been split into a translational and rotational part, as above.

The discontinuity at the leading edge still exists - this is dealt with later using leading edge suction, and the Polhamus leading edge suction analogy.

8.4 Unsteady form of Bernoulli equation

The well known unsteady Bernoulli equation (see for example Katz & Plotkin [7]) is:

$$p = \rho\left(p_0(t) + \frac{\partial\Phi}{\partial t} - \frac{1}{2}u_{\mathrm{T}Ef}^2\right) \tag{28}$$

The $\frac{\partial\Phi}{\partial t}$ term is the added mass, which will be dealt with later (Section 9). The last term becomes the quasi-steady pressure. The velocity of a fluid particle on the upper surface $u_{\mathrm{T}Ef}^+$, is written as:

$$u_{\mathrm{T}Ef}^+ = (u_P + u_{P\gamma} + iu_{NE}), \tag{29}$$

where u_P is the velocity of the undisturbed freestream relative to the wing, $u_{P\gamma}$ is the additional velocity relative to the wing, caused by the bound vorticity. As described earlier, this velocity is purely parallel to the wing surface, and equals $\partial\Phi/\partial\zeta$ and γ. The square of this velocity $\bar{u}_{\mathrm{T}}^{2+}$ is obtained by substituting γ for $u_{P\gamma}$:

$$u_{\mathrm{T}Ef}^{2+} = (u_P + \gamma + iu_{NE})(u_P + \gamma - iu_{NE}) \tag{30}$$

$$= u_P^2 + \gamma^2 + u_{NE}^2 + 2\gamma\, u_P \tag{31}$$

Consider the pressure difference across the wing Δp. The stagnation pressure p_0 is the same above and below. The first three terms of the above are the velocity of the wing, which is the same above and below, so they cancel, leaving:

$$\begin{aligned}\Delta p_Q &= -\rho\,\frac{1}{2}2u_P\gamma^+ - \rho\,\frac{1}{2}2u_P\gamma^- \\ &= -2\rho\, u_P\gamma^+\end{aligned} \tag{32}$$

This gives the normal force for a unit spanwise element of the wing as:

$$\begin{aligned}dF_{NQ} &= 2\rho\, u_P\gamma^+\, d\zeta \\ &= \rho\, u_P\gamma\, d\zeta\end{aligned} \tag{33}$$

This is, again, a standard result - that a uniform free stream flowing past a vortex will cause a force normal to the flow, of a magnitude proportional to the product of the velocity and the circulation.

8.5 Leading edge suction correction

The result of equation (33) is used to incorporate the effect of leading edge suction, by substituting the total velocity for the parallel velocity, so:

$$dF_{NQ} + idF_{PQ} = \rho\,\bar{u}_{\mathrm{T}E}\,\gamma\, d\zeta, \tag{34}$$

where dF is the increment of force corresponding to the increment of chord length $d\zeta$. The total force is normal to the total velocity.

8.6 Quasi-steady forces

The quasi-steady forces on the wing are found for each of the γ components calculated above, using standard integrals of the parameter Q, which can be found in Appendix A. The values for γ employed here are twice those for γ^+ given earlier, as explained above. This calculation differs from the standard textbook case, in that the rotational component of normal velocity $i\dot{\beta}b(\zeta - a)$, is not neglected here, since it is not small compared to the translational velocity. This is because this application has low translational velocity and high angle of attack. For clarity, the solution is split into four components: The cases of isolated translation (T) and rotation (R),for the Dirichlet (D) and Kutta-Joukowski (K) potentials:

TD part:

$$\begin{aligned}
F_{NQ} + iF_{PQ} &= \rho \int_{-1}^{1} \bar{u}_{\mathrm{TE}}\, \gamma_{TD}\, d\zeta \\
&= \rho \int_{-1}^{1} 2\, \bar{u}_{TE}\, u_N\, b \frac{-\zeta}{Q}\, d\zeta \\
&= 2\rho b u_N \int_{-1}^{1} \bar{u}_{\mathrm{TE}} \frac{-\zeta}{Q}\, d\zeta \\
&= 2\rho\, b\, u_N \int_{-1}^{1} (u_P + iu_N + i\dot{\beta}\, b\, (\zeta - a)) \frac{-\zeta}{Q} d\zeta \\
&= 2\, \rho\, b\, u_N\, (0 + 0 - i\dot{\beta}\, b\, \pi/2) \\
&= 2\, \pi\, \rho\, b^2\, u_N\, \dot{\beta}\, (-\frac{1}{2})
\end{aligned} \tag{35}$$

RD part:

$$\begin{aligned}
F_{NQ} + iF_{PQ} &= \rho \int_{-1}^{1} \bar{u}_{TE}\, \gamma_{RD}\, d\zeta \\
&= \rho \int_{-1}^{1} \bar{u}_{TE}\, 2\, \dot{\beta}\, b^2\, (\frac{1}{2} - \zeta^2 + a\, \zeta)/Q\, d\zeta \\
&= 2\, \rho\, b^2\, \dot{\beta} \int_{-1}^{1} \bar{u}_{\mathrm{TE}}\, (\frac{1}{2} - \zeta^2 + a\zeta)/Q\, d\zeta \\
&= 2\, \rho\, b^2\, \dot{\beta} \int_{-1}^{1} \left(u_P + iu_N + i\dot{\beta}b(\zeta - a)\right) \left(\frac{1}{2} - \zeta^2 + a\zeta\right) /Q\, d\zeta \\
&= 2\rho b^2 \dot{\beta} \left[u_P(\pi/2 - \pi/2 + 0) + iu_N(\pi/2 - \pi/2 + 0) + i\dot{\beta}b(\pi/2)\right] \\
&= 2\pi\rho b^2 u_N \dot{\beta}(\frac{1}{2})
\end{aligned} \tag{36}$$

The total quasi-steady contribution of the Dirichlet part is zero, which is as expected - since there is no net vorticity, there can be no net force.

TK part:

$$\begin{aligned}
F_{NQ} + iF_{PQ} &= \rho \int_{-1}^{1} \bar{u}_{\mathrm{TE}}\, \gamma_{TK}\, d\zeta \\
&= \rho \int_{-1}^{1} \bar{u}_{TE}\, 2\, u_N\, b \frac{1}{Q}\, d\zeta
\end{aligned}$$

$$
\begin{aligned}
&= 2\,\rho\, b\, u_N \int_{-1}^{1} \bar{u}_{\mathrm{TE}} \frac{1}{Q}\, d\zeta \\
&= 2\,\rho\, b\, u_N \int_{-1}^{1} \left(u_P + i u_N + i\dot{\beta} b(\zeta - a)\right) \frac{1}{Q}\, d\zeta \\
&= 2\,\rho\, b\, u_N \left[\pi u_P + i\pi u_N + i\pi\dot{\beta} b(0 - a)\right] \\
&= 2\,\pi\, \rho b\, u_N \left[u_P + i u_N - i\pi\dot{\beta} b a\right] \\
&= 2\,\pi\, \rho b\, u_N\, \bar{u}_{\mathrm{T}m} \qquad (37)
\end{aligned}
$$

RK part:

$$
\begin{aligned}
F_{NQ} + iF_{PQ} &= \rho \int_{-1}^{1} \bar{u}_{\mathrm{TE}}\, \gamma_{RK}\, d\zeta \\
&= \rho \int_{-1}^{1} \bar{u}_{\mathrm{TE}}\, 2\,\dot{\beta}\, b^2 \left(\frac{1}{2} - a\right)/Q\, d\zeta \\
&= 2\,\rho\, b^2 \left(\frac{1}{2} - a\right)\dot{\beta} \int_{-1}^{1} \bar{u}_{\mathrm{TE}}\, /Q\, d\zeta \\
&= 2\,\rho\, b^2\, \dot{\beta} \left(\frac{1}{2} - a\right) \int_{-1}^{1} (u_P\; + i u_N\; + i\dot{\beta}\, b\, (\zeta - a))/Q\, d\zeta \\
&= 2\,\rho\, b^2\, \dot{\beta} \left(\frac{1}{2} - a\right) \left[\pi u_P + i\pi u_N + i\pi\dot{\beta} b(0 - a)\right] \\
&= 2\,\pi\, \rho\, b^2\, \dot{\beta} \left(\frac{1}{2} - a\right) \left[u_P\; + i u_N\; - i\dot{\beta}\, b\, a\,\right] \\
&= 2\,\pi\, \rho\, b^2\, \dot{\beta} \left(\frac{1}{2} - a\right) \bar{u}_{\mathrm{T}m} \qquad (38)
\end{aligned}
$$

Note how the Kutta-Joukowski components produce net forces, because they have a net vorticity.

8.7 Total quasi-steady force

The total quasi-steady force is written as the sum of the four components given in equation 35 to 38:

$$
\begin{aligned}
F_{NQ} + iF_{PQ} &= 2\,\pi\, \rho\, b\, (u_N\, \bar{u}_{\mathrm{T}m}\; + b\, \dot{\beta} \left(\frac{1}{2} - a\right)\bar{u}_{\mathrm{T}m}) \qquad (39) \\
&= 2\,\pi\, \rho\, b\, u_{Nr}\, \bar{u}_{\mathrm{T}m}, \qquad (40)
\end{aligned}
$$

where the normal and parallel components are:

$$
\begin{aligned}
F_{NQ} &= 2\pi\, \rho\, b\, u_{Nr}\, u_P \qquad (41) \\
F_{PQ} &= 2\pi\, \rho\, b\, u_{Nr}\, u_{Nm}. \qquad (42)
\end{aligned}
$$

The horizontal and vertical components of these forces are:

$$
F_{HQ} = -F_{NQ}\, \mathrm{S}_\beta + F_{PQ}\, \mathrm{C}_\beta
$$

$$\begin{aligned}
&= 2\pi\rho\, b\, u_{Nr}\, (-u_P\, \mathrm{S}_\beta + u_{Nm}\, \mathrm{C}_\beta) \\
&= 2\pi\rho\, b\, u_{Nr}\, u_{Vm} \qquad (43)\\
F_{VQ} &= F_{NQ}\, \mathrm{C}_\beta + F_{PQ}\, \mathrm{S}_\beta \\
&= 2\pi\, \rho\, b\, u_{Nr}\, (u_P\, \mathrm{C}_\beta + u_{Nm}\, \mathrm{S}_\beta) \\
&= 2\pi\, \rho\, b\, u_{Nr}\, u_{Hm}. \qquad (44)
\end{aligned}$$

Mapping from spherical to rectangular coordinates, the force on the wing is:

$$\begin{aligned}
L &= F_{VQ}\mathrm{C}_\psi \qquad (45)\\
D &= F_{HQ}\mathrm{S}_\theta \qquad (46)
\end{aligned}$$

Recall that drag is defined as force in the $+x$ direction, not the direction opposing motion.

Note that spanwise force is ignored. This is because of the assumption that the wing comes to a point at the tip, combined with the Polhamus correction for tip suction will make the spanwise force zero. This is explained more fully in Section 10.

The standard results are recovered readily by making the same assumptions about fast forward motion at low angle of attack, i.e. that β is small, and $\theta, \psi = 0$. In this case, $u_P \approx u_H$, and the lift force will be the normal force:

$$L = 2\pi\rho b u_{Nr} u_H \qquad (47)$$

This is indeed the standard result for a pitching aerofoil at low β, and can be found in any good textbook on aerodynamics.

8.8 Wing integrals

The above calculations are forces per unit span. This is now extended to the force for the entire wing by integrating along the span, using $2D$ strip theory, extended to arbitrary $3D$ geometry:

$$\begin{aligned}
F_{NQW} &= R\int_0^1 F_{NQ}\, dr \\
&= R\int_0^1 \rho\, b\, u_{Nr}\, u_P\, dr \\
&= \rho R\int_0^1 b\, u_{Nr}\, u_P\, dr \\
&= \rho R\int_0^1 b\, u_P\, (u_N \;+\dot\beta\, b\, (\frac{1}{2} - a))\, dr \\
&= \rho R\int_0^1 b\, u_P\, u_N\, dr + \rho R\int_0^1 b^2\, u_P\, \dot\beta\, (\frac{1}{2} - a))\, dr \qquad (48)
\end{aligned}$$

Considering the first term, note that the velocities at the pitch axis scale directly with r, so can be written in terms of the tip velocities and r:

$$\begin{aligned}
\int_0^1 b\, u_P\, u_N\, dr &= \int_0^1 b\, r\, u_{PT}\, r\, u_{NT}\, dr \\
&= u_{PT}\, u_{NT} \int_0^1 b\, r^2 dr \qquad (49)
\end{aligned}$$

Also, the semichord b can be expressed as a fraction of the maximum semichord B:

$$\begin{aligned} u_{PT}\, u_{NT} \int_0^1 b\, r^2\, dr &= u_{PT}\, u_{NT}\, B \int_0^1 \frac{b}{B}\, r^2\, dr \\ &= u_{PT}\, u_{NT}\, B\, b_1 r_2 \end{aligned} \tag{50}$$

The term $b_1 r_2$ is defined as the integral $\int_0^1 \frac{b}{B} r^2 dr$; the subscripts are the powers of b/B and r, respectively. Thus the integral is purely a function of the wing *shape*, not the wing scale. These so-called *wing shape factors* are convenient in that they speed up calculation considerably, and give additional insight into how the wing shape affects the forces.

Now consider the second term:

$$\begin{aligned} \int_0^1 b^2\, u_P\, \dot\beta\, (\frac{1}{2} - a)\, dr &= \dot\beta\, u_{PT}\, B^2 \int_0^1 (\frac{1}{2} - a)(\frac{b}{B})^2\, r\, dr \\ &= \dot\beta\, u_{PT}\, B^2 \left[\frac{1}{2}\, b_2 r_1 - b_2 r_{1a}\right], \end{aligned} \tag{51}$$

where $b_2 r_1 = \int_0^1 (\frac{b}{B})^2 r\, dr$, and $b_2 r_{1a} = \int_0^1 (\frac{b}{B})^2 r a\, dr$.

Assuming a is constant along the span, equation 51 can be simplified to:

$$\begin{aligned} \int_0^1 b^2\, u_P\, \dot\beta (\frac{1}{2} - a)\, dr &= \dot\beta\, u_{PT}\, B^2\, (\frac{1}{2} - a) \int_0^1 (\frac{b}{B})^2\, r\, dr \\ &= \dot\beta\, u_{PT}\, B^2\, (\frac{1}{2} - a)\, b_2 r_1 \end{aligned} \tag{52}$$

The total force on the wing, assuming that a is constant along the span, is

$$\begin{aligned} F_{NQW} &= R \int_0^1 F_{NQ}\, dr \\ &= R \int_0^1 2\pi\, \rho\, b\, u_{N_r}\, u_P\, dr \\ &= 2\pi\, \rho\, R\, B\, u_{PT} \left[u_{NT}\, b_1 r_2 + \dot\beta\, B\, (\frac{1}{2} - a)\, b_2 r_1\right] \end{aligned} \tag{53}$$

$$\begin{aligned} F_{PQW} &= R \int_0^1 F_{PQ}\, dr \\ &= R \int_0^1 2\pi\, \rho\, b\, u_{N_r}\, u_{Nm}\, dr \\ &= 2\pi\rho R \int_0^1 b \left(u_N + b\, \dot\beta (\frac{1}{2} - a)\right) \left(u_N - b\, \dot\beta\, a\right)\, dr \\ &= 2\pi\rho R \int_0^1 b \left(u_N^2 + u_N\, b\, \dot\beta\, (\frac{1}{2} - 2a) + b^2\, \dot\beta^2\, a^2\right)\, dr \\ &= 2\pi\rho R B \left[u_{NT}^2\, b_1 r_2 + u_{NT}\, B\, \dot\beta\, (\frac{1}{2} - 2a) b_2 r_1 + b^2\, \dot\beta^2\, a^2\, b_3 r_0\right] \end{aligned} \tag{54}$$

8.9 Moments

The vertical (M_{VQ}) and horizontal moments (M_{HQ}) are straightforward:

$$M_{VQ} = R\, F_{VQ}\, r \tag{55}$$

$$M_{HQ} = R\,F_{HQ}\,r \tag{56}$$

The pitching moment about the hinge is formed by going back to the original integral of normal force along the chord, and multiplying by the offset from the hinge $b(a-\zeta)$.

$$M_{\beta Q} = \rho \int_{-1}^{1} \gamma\, u_P\, b\, (a-\zeta)\, d\zeta \tag{57}$$

Equation 57 is now applied to the components of γ in Equations 22, 23, 26 and 27:
TD part:

$$\begin{aligned} M_{\beta QTD} &= \rho \int_{-1}^{1} \gamma_{TD}\, u_P\, b\, (a-\zeta)\, d\zeta \\ &= \rho \int_{-1}^{1} 2u_N\, b\, \frac{-\zeta}{Q}\, u_P\, b\, (a-\zeta)\, d\zeta \\ &= 2\rho\, u_N\, u_P\, b^2 \int_{-1}^{1} \frac{-\zeta}{Q}\, (a-\zeta)\, d\zeta \\ &= 2\rho\, u_N\, u_P\, b^2 \int_{-1}^{1} \frac{\zeta^2-\zeta a}{Q}\, d\zeta \\ &= 2\pi\rho\, u_N\, u_P\, b^2 \left(\frac{1}{2}\right) \end{aligned} \tag{58}$$

RD part:

$$\begin{aligned} M_{\beta QRD} &= \rho \int_{-1}^{1} \gamma_{RD}\, u_P\, b\, (a-\zeta)\, d\zeta \\ &= \rho \int_{-1}^{1} 2\dot{\beta}\, b^2\, \frac{\frac{1}{2}-\zeta^2+a\zeta}{Q}\, u_P\, b\, (a-\zeta)\, d\zeta \\ &= 2\rho\, \dot{\beta}\, u_P\, b^3 \int_{-1}^{1} \frac{\left(\frac{1}{2}-\zeta^2+a\zeta\right)(a-\zeta)}{Q}\, d\zeta \\ &= 2\rho\, \dot{\beta}\, u_P\, b^3 \int_{-1}^{1} \frac{\left(\frac{1}{2}a - a\zeta^2 + a^2\zeta - \frac{\zeta}{2} + \zeta^3 - a\zeta^2\right)}{Q}\, d\zeta \\ &= 2\pi\rho\, \dot{\beta}\, u_P\, b^3 \left(\frac{1}{2}a - \frac{1}{2}a + 0 - 0 + 0 - \frac{1}{2}a\right) \\ &= 2\pi\rho\, \dot{\beta}\, u_P\, b^3 \left(-\frac{1}{2}a\right) \end{aligned} \tag{59}$$

TK part:

$$M_{\beta QTK} = \rho \int_{-1}^{1} \gamma_{TK}\, u_P\, b\, (a-\zeta)\, d\zeta$$

$$\begin{aligned}
&= \rho \int_{-1}^{1} 2u_N \, b \, \frac{1}{Q} \, u_P \, b \, (a-\zeta) \, d\zeta \\
&= 2\rho \, u_N \, u_P \, b^2 \int_{-1}^{1} \frac{a-\zeta}{Q} \, d\zeta \\
&= 2\pi\rho \, u_N \, u_P \, b^2 \, (a)
\end{aligned} \tag{60}$$

RK part:

$$\begin{aligned}
M_{\beta QRK} &= \rho \int_{-1}^{1} \gamma_{RK} \, u_P \, b \, (a-\zeta) \, d\zeta \\
&= \rho \int_{-1}^{1} 2\dot{\beta} \, b^2 \frac{\frac{1}{2}-a}{Q} \, u_P \, b \, (a-\zeta) \, d\zeta \\
&= 2\rho \, \dot{\beta} \, u_P \, b^3 \int_{-1}^{1} \frac{\frac{1}{2}a - a^2 - \frac{1}{2}\zeta + a\zeta}{Q} \, d\zeta \\
&= 2\pi\rho \, \dot{\beta} \, u_P \, b^3 \left(\frac{1}{2}a - a^2\right)
\end{aligned} \tag{61}$$

$$\begin{aligned}
M_{\beta Q} &= M_{\beta QTD} + M_{\beta QRD} + M_{\beta QTK} + M_{\beta QRK} \\
&= \pi\rho \, b^2 u_P \left(u_N - b\dot{\beta}a + 2u_N a + 2\dot{\beta}b\left(\frac{1}{2}a - a^2\right)\right) \\
&= \pi\rho \, b^2 u_P \left(u_N (1+2a) + \dot{\beta}b\left(-a + a - a^2\right)\right) \\
&= \pi\rho \, b^2 u_P \left(u_N (1+2a) - \dot{\beta}ba^2\right)
\end{aligned} \tag{62}$$

Note that while the expressions for M_{VQ} and M_{HQ} bear considerable similarity to the force expressions from before, the pitching moment expression does not, due to the extra factor of ζ it introduces.

Note also that these need not be mapped in any way for them to be the actual moments at the hinge, unlike the forces. This is because the forces F_V, F_H in the spherical system are everywhere normal to the hinge.

8.10 Wing moment integrals

This proceeds exactly as Section 8.8 above: the wing integrals are written in terms of shape parameters, assuming the hinge location to be constant.

$$\begin{aligned}
M_{VQ} &= R \, F_{VQ} r \\
M_{VQW} &= R^2 \int_0^1 F_{VQ} \, r \, dr \\
&= 2\pi\rho \, R^2 \int_0^1 b \, u_{Nr} \, u_{Hm} \, r \, dr
\end{aligned}$$

$$
\begin{aligned}
&= 2\pi\rho R^2 \int_0^1 b\, r\, \left(u_N + b\dot{\beta}\left(\frac{1}{2} - a\right)\right)\left(u_H + b\dot{\beta}\, a\, \mathrm{S}_\beta\right) dr \\
&= 2\pi\rho R^2 \int_0^1 br\left(u_N u_H\ + u_N\, b\dot{\beta}\, a\, \mathrm{S}_\beta\ + u_H b\,\dot{\beta}\, \left(\frac{1}{2} - a\right) + b^2\dot{\beta}^2\left(\frac{1}{2}a - a^2\right)\mathrm{S}_\beta\right) dr \\
&= 2\pi\rho\, R^2\, B\, u_{NT}\, u_{HT}\, b_1 r_3 \\
&\quad +2\pi\rho\, R^2\, B^2 u_{NT}\dot{\beta}\, a\, \mathrm{S}_\beta\, b_2 r_2 \\
&\quad +2\pi\rho\, R^2\, B^2 u_{HT}\dot{\beta}\, \left(\frac{1}{2} - a\right) b_2 r_2 \\
&\quad +2\pi\rho\, R^2\, B^3\dot{\beta}^2\left(\frac{1}{2}a - a^2\right)\mathrm{S}_\beta\, b_3 r_1
\end{aligned} \tag{63}
$$

$$
\begin{aligned}
M_{HQ} &= R\, F_{HQ}\, r \\
M_{HQW} &= R^2 \int_0^1 F_{VQ}\, r\, dr \\
&= 2\pi\rho\, R^2 \int_0^1 b\, u_{N_T}\, u_{V_m} r\, dr \\
&= 2\pi\rho\, R^2 \int_0^1 b\, r\, \left(u_N + b\dot{\beta}\left(\frac{1}{2} - a\right)\right)\left(u_V + b\dot{\beta}\, a\, \mathrm{C}_\beta\right) dr \\
&= 2\pi\rho R^2 \int_0^1 br\left(u_N u_V + u_N\, b\dot{\beta}\, a\, \mathrm{C}_\beta + u_V b\,\dot{\beta}\, \left(\frac{1}{2} - a\right) + b^2\dot{\beta}^2\left(\frac{1}{2}a - a^2\right)\mathrm{C}_\beta\right) dr \\
&= 2\pi\rho\, R^2\, B\, u_{NT}\, u_{VT}\, b_1 r_3 \\
&\quad +2\pi\rho\, R^2\, B^2 u_{NT}\,\dot{\beta}\, a\, \mathrm{C}_\beta\, b_2 r_2 \\
&\quad +2\pi\rho\, R^2\, B^2 u_{VT}\,\dot{\beta}\, \left(\frac{1}{2} - a\right) b_2 r_2 \\
&\quad +2\pi\rho\, R^2\, B^3\dot{\beta}^2\, \left(\frac{1}{2}a - a^2\right)\mathrm{C}_\beta\, b_3 r_1
\end{aligned} \tag{64}
$$

$$
\begin{aligned}
M_{\beta Q} &= \pi\rho\, b^2 u_P\left(u_N(1+2a) - \dot{\beta}ba^2\right) \\
M_{\beta QW} &= \pi\rho\, R \int_0^1 u_P b^2\, \left(u_N(1+2a) - \dot{\beta}ba^2\right)\, dr \\
&= \pi\rho\, R\, B^2\, u_{PT}\left(u_{NT}(1+2a)\, b_2 r_2 - \dot{\beta}Ba^2\, b_3 r_1\right)
\end{aligned} \tag{65}
$$

8.11 Summary of assumptions and results

Standard potential theory has been used to derive the quasi-steady forces on a thin, flat wing. The calculations are as standard cases, except with the following two generalisations:

- β, α are not ≈ 0. This means the expressions had to be derived in terms of the parallel and normal velocities. Note especially that the bound vorticity is a function of the normal velocity *only*.

- The wing is not in fast forward motion. This means that the rotational component of the velocity can be considerable compared to the translational component, and cannot be discounted.

It is assumed that:

1. The flow is entirely inviscid, and globally irrotational.
2. The flow leaves the trailing edge smoothly, satisfying the Kutta-Joukowski condition.
3. The flow stays attached at the leading edge, despite it being sharp.
4. There is no shed wake.
5. The hinge point is constant (where wing shape factors are used).

The fourth assumption is a direct violation of the Kelvin-Helmholtz theorem, which states that circulation must be preserved. However, this is only an interim stage, as the effect of wake circulation will be dealt with later, see Section 11.

The third assumption is unrealistic, in that the flow passing round the leading edge will experience infinite acceleration (similarly to the basis for the Kutta-Joukowski condition). This will be corrected by the Polhamus leading edge suction analogy, see Section 10.

The main results for this section are the vertical and horizontal quasi-steady forces on the wing:

$$F_{HQ} = 2\pi\rho\, b\, u_{Nr}\, u_{Vm} \tag{66}$$
$$F_{VQ} = 2\pi\, \rho\, b\, u_{Nr}\, u_{Hm} \tag{67}$$

in the spherical coordinate system, and

$$L = F_{VQ}\mathrm{C}_{\psi} \tag{68}$$
$$D = F_{HQ}\mathrm{S}_{\theta} \tag{69}$$

in the rectangular coordinate system, noting that drag D is defined as force in the $+x$ direction, not the direction opposing motion.

9 Added mass effects

9.1 What is added mass?

In 1851, Stokes [43] showed experimentally that the force on a pendulum in a fluid depended not only on the speed of the pendulum, but also the acceleration. When a body is accelerated in a fluid, it will experience a retarding force, apart from the viscous drag. This is completely independent of the inertia of the body itself, and can be shown to occur even in a completely inviscid fluid for a massless object. This is called an *irrotational* or *non-circulatory* effect, because it does not rely on a net circulation in order to generate force. It is a purely potential effect. However, net circulatory components will also have an added mass effect, since adding them will modify the potential, see e.g. [28].

The concept of forces arising from an inviscid fluid is unusual, so there now follows an explanation of that effect, loosely based on that found in Massey [44]. Imagine an undisturbed inviscid fluid with a body moving through it at a constant velocity U. In order to allow the body passage, the fluid has to move aside ahead of the body, and close up after it, thus the fluid acquires kinetic energy due to the passage of the body, even when the free stream is at rest. When U is constant, this kinetic energy is also constant, and there is no net force on the body, as expected. However, increasing the velocity of the body will also increase the kinetic energy of the flow, so the body has to do work on the fluid.

If the body is not deforming or rotating, but accelerating in a single direction, the velocity field will be self-similar, in that it will scale linearly with the velocity of the body U, but the streamlines will have the same shape. Therefore, the kinetic energy of a point or of the fluid as a whole is kU^2, where k is a constant based on the shape and alignment of the body. The rate at which the body is doing work on the fluid is dT/dt, and is equal to the force F the body is exerting in the direction of motion, times the velocity U, so that[2]

$$\begin{aligned} FU &= \frac{dT}{dt} \\ &= \frac{dkU^2}{dt} \\ &= k2U\frac{dU}{dt} \\ F &= 2k\frac{dU}{dt} \end{aligned} \tag{70}$$

Note the variables T, U and k are local to this section, and will not be used elsewhere.

This shows the body will experience a force proportional to the acceleration - since this can be modelled as if the body had slightly more inertia, it is called an *added mass* effect. By definition, an added mass force is the dynamic force opposing acceleration of a body relative to a fluid.

If the body is changing shape or alignment, the k term in the above will not be constant.

[2] Newton's second law yields $\frac{d}{dt}(mU) = F$, so that $\frac{d}{dt}(mU)U = FU$, or $\frac{d}{dt}(\frac{1}{2}mU^2) = FU$, for m constant.

Although it is often used as a simple explanation, added mass does not represent fluid that is rigidly bound to the wing by viscosity. It is an artefact of the fluid being given kinetic energy by the body. For that reason, since viscosity does affect the velocity of the fluid, it will affect the added mass, but is not necessary for the definition of added mass, see for example [11].

9.2 Potential form of added mass

The informal example of Section 9.1 is now revisited rigorously, by referring to Milne-Thomson [12, pp. 94–95]. Since the potential function Φ completely describes the inviscid flow, the kinetic energy and pressure can be expressed in terms of Φ:

$$T = \frac{1}{2}\int_V \rho \bar{u}_{\mathrm{T}}^2 \, dV \tag{71}$$

$$\tag{72}$$

This simply states that the kinetic energy of a volume of fluid is the volumetric integral of the kinetic energy at every point, but $\bar{u}_{\mathrm{T}} = \nabla\Phi$, so that:

$$\begin{aligned} T &= \frac{1}{2}\rho\int_V (\nabla\Phi)^2 \, dV &(73)\\ &= \frac{1}{2}\rho\int_S \Phi\frac{\partial\Phi}{\partial n}\, dS &(74) \end{aligned}$$

where Green's theorem was used to relate the volume integral over V to the integral over the surface S of volume V, and n is a unit outward normal vector.

For the case of a thin, flat 2D plate, the above reduces to the familiar unsteady Bernoulli equation, as already shown on page 36. This reduction can be found in, e.g. Sedov [13, pp. 15–27] or Milne-Thomson [12, pp. 82–89].

$$p = \rho\left(p_0(t) + \frac{\partial\Phi}{\partial t} - \frac{1}{2}u_{\mathrm{T}Ef}^2\right) \tag{75}$$

The third term is the quasi-steady pressure, as used in Section 8. The first two terms relate to the added mass. However, since p_0 is constant, it can be ignored, so that the pressure due to added mass is:

$$p_a = \rho\frac{\partial\Phi}{\partial t} \tag{76}$$

The normal force for a single surface is obtained by integrating this along the chord:

$$\begin{aligned} F_{NA} &= -b\int_{-1}^{1} p_a \, d\zeta \\ &= -\rho\, b\int_{-1}^{1}\frac{\partial\Phi}{\partial t}d\zeta \\ &= -\rho\, b\frac{\partial}{\partial t}\int_{-1}^{1}\Phi\, d\zeta, \end{aligned}$$

where the last step can be taken because the variable of the differentiation, t, is independent of the variable of integration ζ, and Φ is continuously differentiable.

For the force normal to the wing, the difference $\Delta\Phi = \Phi^+ - \Phi^-$ in potential across the wing is considered. This gives the force:

$$F_{NA} = -\rho\, b\frac{\partial}{\partial t}\int_{-1}^{1} \Delta\Phi\, d\zeta \tag{77}$$

WARNING

The following step is *important*: because Φ has been defined in terms of the velocity of the fluid relative to the wing, but added mass is based on the velocity of the body relative to the fluid, the sign of Φ in Equation 77 has to be *reversed*:

$$F_{NA} = +\rho\, b\frac{\partial}{\partial t}\int_{-1}^{1} \Delta\Phi\, d\zeta. \tag{78}$$

9.3 Total circulation

Integration of the potential in Equation 78 is not straightforward, because Φ is discontinuous at the leading edge. Instead, the standard method of thin aerofoil theory is employed, to form the total circulation along the upper surface of the wing from the leading edge to a point ζ:

$$\Gamma^+(\zeta) = \int_{-1}^{\zeta} \gamma^+\, d\zeta, \tag{79}$$

remembering that $\Gamma(-1) = 0$.

Now consider the integral of Φ from the trailing edge:

$$\begin{aligned}\Phi^+(\zeta) &= \int_{1}^{\zeta} \gamma^+\, d\zeta \\ &= \Gamma^+(\zeta) - \Gamma^+(1),\end{aligned} \tag{80}$$

noting that $\Phi(1) = 0$.

This means that the total circulation $\Gamma = \Gamma^+ + \Gamma^- = 2\Gamma^+$ can be substituted for $\Delta\Phi$ in Equation 78 . This allows integration from the leading edge, since Γ is 0 there, and therefore continuous. This method is similar to that of Katz & Plotkin [7, page 73]. This could also have been done by using $\Delta\Phi$ and integrating from the trailing edge: this was the method adapted by Theodorsen [28], but is more cumbersome.

In either case, it is important to remember that this is still a calculation based on potential. The potential is simply being expressed in terms of the bound vorticity of the wing, in accordance with the thin aerofoil theory.

For the four given components of the potential, the equivalent total vorticity Γ is found by integrating the vorticity γ, of Equations (22), (23) (26) and (27):

$$\Gamma_{TD} = 2u_N bQ \tag{81}$$
$$\Gamma_{RD} = 2\dot{\beta}b^2\left(\frac{\zeta Q}{2} - aQ\right) \tag{82}$$
$$\Gamma_{TK} = 2u_N b\,(\mathrm{asin}(\zeta) + \pi/2) \tag{83}$$
$$\Gamma_{RK} = 2\dot{\beta}b^2\left(\frac{1}{2} - a\right)(\mathrm{asin}(\zeta) + \pi/2) \tag{84}$$

Note the similarity of these expressions to those of the potential of Equations 20, 21, 24 and 25. The first two terms are identical, while the second two only differ by a constant π, to ensure that Γ is zero at the leading edge, while Φ is zero at the trailing edge.

9.4 Normal added mass forces

Equation (78) is evaluated for the four components:

TD part:

$$\begin{aligned} F_{NA} &= \rho\, b\frac{\partial}{\partial t}\int_{-1}^{1}\Gamma_{TD}\, d\zeta \\ &= \rho\, b\frac{\partial}{\partial t}\int_{-1}^{1} 2u_N\, b\, Qd\zeta \\ &= 2\rho\, b^2\frac{\partial}{\partial t}\left(u_N \int_{-1}^{1} Qd\zeta\right) \\ &= \pi\rho\, b^2\frac{\partial}{\partial t}(u_N) \\ &= \pi\rho\, b^2\dot{u}_N \end{aligned} \tag{85}$$

RD part:

$$\begin{aligned} F_{NA} &= \rho\, b\frac{\partial}{\partial t}\int_{-1}^{1}\Gamma_{RD}\, d\zeta \\ &= \rho\, b\frac{\partial}{\partial t}\int_{-1}^{1} 2\dot{\beta}b^2\left(\frac{\zeta Q}{2} - aQ\right) d\zeta \\ &= 2\rho\, b^3\,\frac{\partial}{\partial t}\dot{\beta}\int_{-1}^{1}\frac{\zeta Q}{2} - aQ\, d\zeta \\ &= \pi\rho\, b^3(-a)\frac{\partial}{\partial t}\dot{\beta} \\ &= \pi\rho\, b^3(-a)\ddot{\beta} \end{aligned} \tag{86}$$

The two Dirichlet components combine to give

$$F_{NAD} = \pi\rho b^2\dot{u}_{Nm}, \tag{87}$$

where $\dot{u}_{Nm}$ is the normal acceleration of the midpoint of the wing.

TK part:

$$\begin{aligned}
F_{NA} &= \rho\, b\frac{\partial}{\partial t}\int_{-1}^{1}\Gamma_{TK}\, d\zeta \\
&= \rho\, b\frac{\partial}{\partial t}\int_{-1}^{1} 2u_N b\,(\mathrm{asin}(\zeta)+pi/2)\ d\zeta \\
&= 2\rho\, b^2\frac{\partial}{\partial t}\left(u_N\int_{-1}^{1}\mathrm{asin}(\zeta)+pi/2 d\zeta\right) \\
&= 2\pi\rho\, b^2\frac{\partial}{\partial t}\,(u_N) \\
&= 2\pi\rho\, b^2\dot{u}_N \qquad (88)
\end{aligned}$$

RK part:

$$\begin{aligned}
F_{NA} &= \rho\, b\frac{\partial}{\partial t}\int_{-1}^{1}\Gamma_{RK}\, d\zeta \\
&= \rho\, b\frac{\partial}{\partial t}\int_{-1}^{1} 2\dot{\beta}b^2\left(\frac{1}{2}-a\right)(\mathrm{asin}(\zeta)+\pi/2)\ d\zeta \\
&= 2\rho\, b^3\left(\frac{1}{2}-a\right)\frac{\partial}{\partial t}\left(\dot{\beta}\int_{-1}^{1}\mathrm{asin}(\zeta)+\pi/2\, d\zeta\right) \\
&= 2\pi\rho\, b^3\left(\frac{1}{2}-a\right)\frac{\partial}{\partial t}\dot{\beta} \\
&= 2\pi\rho\, b^3\left(\frac{1}{2}-a\right)\ddot{\beta} \qquad (89)
\end{aligned}$$

The two Kutta-Joukowski components combine to give:

$$F_{NAK} = 2\pi\rho b^2\dot{u}_{Nr}, \qquad (90)$$

where $\dot{u}_{Nr}$ is the normal acceleration of the 3/4-chord of the wing, also called the *rear neutral point.*

9.5 Accelerations

In order to find the acceleration, u_N is written in terms of the global velocities:

$$\begin{aligned}
u_N &= u_H\mathrm{S}+u_V\mathrm{C} && (91)\\
u_{Nm} &= u_H\mathrm{S}+u_V\mathrm{C}-\dot{\beta}\, b\, a && (92)\\
u_{Nm} &= u_H\mathrm{S}+u_V\mathrm{C}+\dot{\beta}\, b\left(\frac{1}{2}-a\right) && (93)\\
& && (94)
\end{aligned}$$

This gives the accelerations:

$$\begin{aligned}
\dot{u}_N &= \dot{u}_H\mathrm{S}+\dot{u}_V\mathrm{C}+2\,\dot{\beta}\,u_P && (95)\\
\dot{u}_{Nm} &= \dot{u}_H\mathrm{S}+\dot{u}_V\mathrm{C}+2\,\dot{\beta}\,u_P-\ddot{\beta}\, b\, a && (96)\\
\dot{u}_{Nr} &= \dot{u}_H\mathrm{S}+\dot{u}_V\mathrm{C}+2\,\dot{\beta}\,u_P+\ddot{\beta}\, b\left(\frac{1}{2}-a\right) && (97)
\end{aligned}$$

The reason for this substitution will be explained in Section 9.7.

9.5.1 Parallel added mass forces

F_{PA}, the parallel added mass force, is formed by substituting normal acceleration components for parallel ones, similar to the way velocities were substituted to find F_{PQ} in Section 8.5. However, this cannot be done by simply substituting the parallel acceleration. Some of the terms above are the result of an increase in Γ with the normal velocity. Intuitively, it is obvious that the plate will have smaller added mass when accelerating along its length than when it is accelerating normal to the chord, simply because in the second case it is blocking the flow.

The actual values are taken from [13], page 27, equation 4.17. Sedov writes the added mass forces on the wing in the absence of wake circulation as:

$$X = \lambda_y \Omega V + \lambda_{y\omega}\Omega^2 \tag{98}$$
$$Y = -\lambda_y \frac{dV}{dt} - \lambda_{y\omega}\frac{d\Omega}{dt}, \tag{99}$$

where X, Y are the parallel and normal forces at the leading edge the velocities are U, V, and the rotational velocity is Ω. λ_y and $\lambda_{\omega y}$ are added mass coefficients, tabulated on page 29 of the same reference: $\lambda_y = \rho\pi b^2$, $\lambda_{y\omega} = \rho\pi b^3$.

The expression for X uses the same values of λ as Y, so the expression for X can be formed by making the following substitutions into the expression for Y:

$$\frac{dV}{dt} \to -\Omega V \tag{100}$$
$$\frac{d\Omega}{dt} \to -\Omega^2 \tag{101}$$

From equation (99), it can be seen that the expression for Y is similar to the expression for normal force. By analogy with the above substitution, the substitution:

$$\dot{u}_N \to -\dot\beta\, u_N \tag{102}$$
$$\ddot\beta \to -\dot\beta^2 \tag{103}$$

is used in equations (87) and (90) to yield:

$$\begin{aligned} F_{NAD} &= \pi\rho\, b^2\, \dot{u}_{Nm} \\ &= \pi\rho\, b^2 \left(\dot{u}_N - \ddot\beta\, b\, a\right) \\ F_{PAD} &= \pi\rho\, b^2 \left(-\dot\beta\, u_N + \dot\beta^2\, b\, a\right) \end{aligned} \tag{104}$$

$$\begin{aligned} F_{NAK} &= 2\pi\rho\, b^2\, \dot{u}_{Nr} \\ &= 2\pi\rho\, b^2 \left(\dot{u}_N + \ddot\beta\, b\, \left(\frac{1}{2} - a\right)\right) \\ F_{PAK} &= 2\pi\rho\, b^2 \left(-\dot\beta\, u_N - \dot\beta^2\, b\, \left(\frac{1}{2} - a\right)\right) \end{aligned} \tag{105}$$

9.6 Vertical and horizontal added mass forces

The normal and parallel forces are resolved into horizontal and vertical components:

$$\begin{aligned}
F_{VAD} &= F_{NAD}\mathrm{C}_\beta + F_{PAD}\mathrm{S}_\beta & (106)\\
&= \pi\rho b^2\left[\dot{u}_H\mathrm{S}_\beta\mathrm{C}_\beta + \dot{u}_V\mathrm{C}_\beta^2 + 2\dot{\beta}u_P\mathrm{C}_\beta - \ddot{\beta}ab\mathrm{C}_\beta - \dot{\beta}u_N\mathrm{S}_\beta + \dot{\beta}^2ab\mathrm{S}_\beta\right]\\
&= \pi\rho b^2\left[\dot{u}_H\mathrm{S}_\beta\mathrm{C}_\beta + \dot{u}_V\mathrm{C}_\beta^2 + \dot{\beta}(2u_P\mathrm{C}_\beta - u_N\mathrm{S}_\beta) - \ddot{\beta}ab\mathrm{C}_\beta + \dot{\beta}^2ab\mathrm{S}_\beta\right]\\
F_{HAD} &= -F_{NAD}\mathrm{S}_\beta + F_{PAD}\mathrm{C}_\beta & (107)\\
&= \pi\rho b^2\left[-\dot{u}_H\mathrm{S}_\beta^2 - \dot{u}_V\mathrm{S}_\beta\mathrm{C}_\beta - 2\dot{\beta}u_P\mathrm{S}_\beta + \ddot{\beta}ab\mathrm{S}_\beta - \dot{\beta}u_N\mathrm{C}_\beta + \dot{\beta}^2ab\mathrm{C}_\beta\right]\\
F_{VAK} &= F_{NAK}\mathrm{C}_\beta + F_{PAK}\mathrm{S}_\beta & (108)\\
&= 2\pi\rho b^2\left[\dot{u}_H\mathrm{S}_\beta\mathrm{C}_\beta + \dot{u}_V\mathrm{C}_\beta^2 + 2\dot{\beta}u_P\mathrm{C}_\beta - \ddot{\beta}b(a-\frac{1}{2})\mathrm{C}_\beta - \dot{\beta}u_N\mathrm{S}_\beta + \dot{\beta}^2(a-\frac{1}{2})b\mathrm{S}_\beta\right]\\
F_{HAK} &= -F_{NAK}\mathrm{S}_\beta + F_{PAK}\mathrm{C}_\beta & (109)\\
&= 2\pi\rho b^2\left[-\dot{u}_H\mathrm{S}_\beta^2 - \dot{u}_V\mathrm{S}_\beta\mathrm{C}_\beta - 2\dot{\beta}u_P\mathrm{S}_\beta + \ddot{\beta}b(a-\frac{1}{2})\mathrm{S}_\beta - \dot{\beta}u_N\mathrm{C}_\beta + \dot{\beta}^2b(a-\frac{1}{2})\mathrm{C}_\beta\right]
\end{aligned}$$

9.7 Frames of reference

Equations 106 to 109 have several acceleration terms that come from a standard result of classical mechanics: That of an acceleration in a moving reference frame being mapped to an inertial reference frame. The local coordinate system on the wing is performing translational acceleration, rotational acceleration and is rotating - each of these will disqualify it as an inertial frame. Considering a vector $\bar{r}$ from the origin of a local coordinate system, designated R, to a point in space, the absolute acceleration of the point in a Newtonian reference frame is, from e.g. Marion & Thornton [45]:

$$a_0 + \frac{d^2\bar{r}}{dt^2} + \frac{d\bar{\omega}}{dt}\times\bar{r} + 2\bar{\omega}\times\frac{d\bar{r}}{dt} + \bar{\omega}\times(\bar{\omega}\times\bar{r}), \qquad (110)$$

where the axes in both frames form right-handed sets, a_0 is the translational acceleration of the origin of R and $\bar{\omega}$ is the rotational velocity of R. Note: these uses of $\bar{r}$ and R are employed only in this description, and will not be used in other sections.

The first term and second terms are obvious enough: the acceleration of R (the first term) is added to the translational acceleration in R (the second term).

The third term is the *Euler* effect. Imagine that $\bar{r}$ is fixed, but the frame has rotational acceleration - this obviously causes an acceleration, but it is invisible in R, because it it rotating - effectively "tracking" the point.

The fourth term is the *Coriolis* effect. If an observer moves in R, the movement is compounded by the rotation of R. For example, imagine standing on a constantly rotating disc (such as an LP record), near the centre. Points further from the centre are moving faster, tangentially to the radius. Therefore, for every step the observer takes towards the rim of the disc, he gains some of this tangential velocity - effectively, being accelerated sideways.

The fifth term is the *centripetal* effect, which should be familiar. If $\bar{r}$ is constant, and the observer is rotating about a point, he will be undergoing an acceleration towards the centre of rotation, in order to maintain the rotational motion. To the observer, this is manifest as a sensation of outwards centrifugal force, but it is purely a kinematic effect.

These terms manifest themselves in Equations (106) to (109) as follows: in F_{NAD}, the term $-\ddot{\beta}\ b\ a$ is an Euler term for the normal acceleration of the midpoint of the wing: the midpoint has this extra acceleration relative to the hinge. Note that this term is purely normal, it does not appear in the parallel force expressions. Similarly, the term $\dot{\beta}^2\ b\ a$ in the expression for F_{PAD} appears only in the parallel force expressions, because it is a centripetal term, and therefore points along the wing. The term $2\ \dot{\beta}\ u_P$ in the expression for $\dot{u}_N$ is a Coriolis term. This is best explained with another example. Imagine an observer travelling at constant velocity, and the wing is oriented into the flow so the velocity is purely parallel. If the observer maintains the same direction of travel, but starts pitching the wing up, the parallel velocity gradually becomes a normal velocity - thus experiencing a positive normal and negative parallel acceleration of the flow relative to the wing.

9.8 Moments

The root moment of the wing is formed similarly to the quasi-steady case in Section 8:

$$M_{VA} = RF_{VA}\, r \tag{111}$$
$$M_{HA} = RF_{HA}\, r \tag{112}$$

Similarly, the pitching moment about the hinge is formed by revisiting the normal force expressions in Equations 87 and 90, and multiplying by the backwards offset from the hinge $b(a-\zeta)$. Thus, the pitching moment becomes:

$$M_{\beta A} = \rho\, b \frac{\partial}{\partial t} \int_{-1}^{1} \Gamma\, b(a-\zeta) \tag{113}$$

The contributions for the four components, as for the forces are:
TD part:

$$\Gamma_{TD} = 2u_N\, b\, Q \tag{114}$$

$$\begin{aligned}
M_{\beta ATD} &= \rho\, b \frac{\partial}{\partial t} \int_{-1}^{1} \Gamma_{TD}\, b(a-\zeta) \\
&= \rho\, b \frac{\partial}{\partial t} \int_{-1}^{1} 2u_N\, b\, Q\, b(a-\zeta) \\
&= 2\rho\, b^3 \frac{\partial}{\partial t} u_N \int_{-1}^{1} Q\ (a-\zeta) \\
&= 2\rho\, b^3\, \dot{u}_N\, \pi \left(\frac{1}{2}a - 0\right) \\
&= \pi\rho\, b^3\, \dot{u}_N\, a
\end{aligned} \tag{115}$$

RD part:

$$\Gamma_{RD} = 2\dot{\beta}\, b^2\, Q\left(\frac{1}{2}\zeta - a\right) \tag{116}$$

$$
\begin{aligned}
M_{\beta ARD} &= \rho\, b\, \frac{\partial}{\partial t} \int_{-1}^{1} \Gamma_{RD}\, b\,(a-\zeta) \\
&= \rho\, b\, \frac{\partial}{\partial t} \int_{-1}^{1} 2\dot{\beta}\, b^2\, Q\left(\frac{1}{2}\zeta - a\right) b\,(a-\zeta) \\
&= 2\rho\, b^4\, \frac{\partial}{\partial t}\, \dot{\beta} \int_{-1}^{1} Q\, \left(\frac{1}{2}\zeta - a\right)(a-\zeta) \\
&= 2\rho\, b^4\, \ddot{\beta} \int_{-1}^{1} Q\, \left(\frac{1}{2}a\zeta - a^2 - \frac{1}{2}\zeta^2 + a\zeta\right) \\
&= 2\rho\, b^4\, \ddot{\beta}\pi\, \left(0 - \frac{1}{2}a^2 - \frac{1}{16} + 0\right) \\
&= \pi\rho\, b^4\, \ddot{\beta}\left(-a^2 - \frac{1}{8}\right)
\end{aligned} \tag{117}
$$

TK part:

$$\Gamma_{TK} = 2u_N\, b\, (\mathrm{asin}(\zeta) + \pi/2) \tag{118}$$

$$
\begin{aligned}
M_{\beta ATK} &= \rho\, b\, \frac{\partial}{\partial t} \int_{-1}^{1} \Gamma_{TK}\, b\,(a-\zeta) \\
&= \rho\, b\, \frac{\partial}{\partial t} \int_{-1}^{1} 2u_N\, b\, (\mathrm{asin}(\zeta) + \pi/2)\; b\,(a-\zeta) \\
&= 2\rho\, b^3\, \frac{\partial}{\partial t}\, u_N \int_{-1}^{1} (\mathrm{asin}(\zeta) + \pi/2)\; (a-\zeta) \\
&= 2\rho\, b^3\, \frac{\partial}{\partial t}\, u_N \int_{-1}^{1} (a\,\mathrm{asin}(\zeta) + a\pi/2 - \zeta\mathrm{asin}(\zeta) - \zeta\pi/2) \\
&= 2\rho\, b^3\, \dot{u}_N\, \pi\left(0 + a - \frac{1}{4} - 0\right) \\
&= \pi\rho\, b^3\, \dot{u}_N\, \left(2a - \frac{1}{2}\right)
\end{aligned} \tag{119}
$$

RK part:

$$\Gamma_{RK} = 2\dot{\beta}\, b^2\, \left(\frac{1}{2} - a\right)(\mathrm{asin}(\zeta) + \pi/2) \tag{120}$$

$$
\begin{aligned}
M_{\beta ATK} &= \rho\, b\, \frac{\partial}{\partial t} \int_{-1}^{1} \Gamma_{RK}\, b\,(a-\zeta) \\
&= \rho\, b\, \frac{\partial}{\partial t} \int_{-1}^{1} 2\dot{\beta}\, b^2\, \left(\frac{1}{2} - a\right)(\mathrm{asin}(\zeta) + \pi/2)\, b\,(a-\zeta) \\
&= 2\rho\, b^4\, \left(\frac{1}{2} - a\right)\ddot{\beta} \int_{-1}^{1} (a\,\mathrm{asin}(\zeta) + a\pi/2 - \zeta\mathrm{asin}(\zeta) - \zeta\pi/2)
\end{aligned}
$$

$$
\begin{aligned}
&= 2\rho\, b^4 \left(\frac{1}{2}-a\right)\ddot{\beta}\,\pi\,\left(0+a-\frac{1}{4}-0\right) \\
&= \pi\,\rho\, b^4\,\ddot{\beta}\left(\frac{1}{2}-a\right)\left(a-\frac{1}{4}\right) \qquad (121)
\end{aligned}
$$

9.9 Comparison with standard results

Although the added mass force and moment expressions contain acceleration terms, they do not reduce to the acceleration at a single point. This is because the calculation of added mass entails all of the normal acceleration, but only some of the parallel acceleration.

If the Kutta-Joukowski term is removed, and $\beta = \dot{\beta} = \ddot{\beta} = 0$, then:

$$
\begin{aligned}
F_{NAD} = F_{VAD} &= \pi\rho\, b^2\left[\dot{u}_V + 2\dot{\beta}u_P - \ddot{\beta}\, b\, a\right] \qquad (122) \\
&= \pi\rho\, b^2\dot{u}_{Vm} \qquad (123)
\end{aligned}
$$

This is the standard result of Jones, as outlined on pages 192–194 of Katz & Plotkin [7].

Another check is that for a closed cycle, F_{VA} and F_{HA} must sum to zero - they are *closed* functions. This does not apply to the forces F_{NA} and F_{PA} - they are defined in non-Newtonian axes, so closure is not guaranteed.

9.10 Wing integrals

The results above (which are forces per metre span) are converted to wing integrals, using the wing shape parameter method of Section 8.8. Again, it is assumed that the hinge is constant along the wing, which allows the use of a smaller set of wing shape parameters.

$$
\begin{aligned}
F_{VAD} &= \pi\rho b^2\left[\dot{u}_H \mathrm{S}_\beta \mathrm{C}_\beta + \dot{u}_V \mathrm{C}_\beta^2 + 2\dot{\beta}u_P\mathrm{C}_\beta - \ddot{\beta}ab\mathrm{C}_\beta - \dot{\beta}u_N\mathrm{S}_\beta + \dot{\beta}^2 ab\mathrm{S}_\beta\right] \\
F_{VADW} &= \pi\rho RB^2\left(\dot{u}_{HT}\mathrm{S}_\beta\mathrm{C}_\beta + \dot{u}_{VT}\mathrm{C}_\beta^2\right) b_2 r_1 \\
&\quad + \pi\rho RB^2\left(2\dot{\beta}u_P\mathrm{C}_\beta - \dot{\beta}u_N\mathrm{S}_\beta\right) b_2 r_1 \\
&\quad + \pi\rho RB^3\left(-\ddot{\beta}a\mathrm{C}_\beta + \dot{\beta}^2 a\mathrm{S}_\beta\right) b_3 r_0 \qquad (124)
\end{aligned}
$$

$$
\begin{aligned}
F_{HAD} &= \pi\rho b^2\left[-\dot{u}_H \mathrm{S}_\beta^2 - \dot{u}_V \mathrm{S}_\beta\mathrm{C}_\beta - 2\dot{\beta}u_P\mathrm{S}_\beta + \ddot{\beta}ab\mathrm{S}_\beta - \dot{\beta}u_N\mathrm{C}_\beta + \dot{\beta}^2 ab\mathrm{C}_\beta\right] \\
F_{HADW} &= \pi\rho B^2\left(-\dot{u}_H\mathrm{S}_\beta^2 - \dot{u}_V\mathrm{S}_\beta\mathrm{C}_\beta\right) b_2 r_1 \\
&\quad + \pi\rho B^2\left(-2\dot{\beta}u_P\mathrm{S}_\beta - \dot{\beta}u_N\mathrm{C}_\beta\right) b_2 r_1 \\
&\quad + \pi\rho B^3\left(+\ddot{\beta}ab\mathrm{S}_\beta + \dot{\beta}^2 ab\mathrm{C}_\beta\right) b_3 r_0 \qquad (125)
\end{aligned}
$$

$$
F_{VAK} = 2\pi\rho b^2\left[\dot{u}_H\mathrm{S}_\beta\mathrm{C}_\beta + \dot{u}_V\mathrm{C}_\beta^2 + 2\dot{\beta}u_P\mathrm{C}_\beta - \ddot{\beta}b(a-\frac{1}{2})\mathrm{C}_\beta - \dot{\beta}u_N\mathrm{S}_\beta + \dot{\beta}^2(a-\frac{1}{2})b\mathrm{S}_\beta\right]
$$

$$\begin{aligned} F_{VAKW} &= 2\pi\rho B^2\left(\dot{u}_H S_\beta C_\beta + \dot{u}_V C_\beta^2\right) b_2 r_1 \\ &+2\pi\rho B^2\left(+2\dot{\beta} u_P C_\beta - \dot{\beta} u_N S_\beta\right) b_2 r_1 \\ &+2\pi\rho B^3 (a-\frac{1}{2})\left(-\ddot{\beta} C_\beta + \dot{\beta}^2 S_\beta\right) b_3 r_0 \end{aligned} \quad (126)$$

$$\begin{aligned} F_{HAK} &= 2\pi\rho b^2\left[-\dot{u}_H S_\beta^2 - \dot{u}_V S_\beta C_\beta - 2\dot{\beta} u_P S_\beta + \ddot{\beta} b(a-\frac{1}{2}) S_\beta - \dot{\beta} u_N C_\beta + \dot{\beta}^2 b(a-\frac{1}{2}) C_\beta\right] \\ F_{HAKW} &= 2\pi\rho B^2\left(-\dot{u}_H S_\beta^2 - \dot{u}_V S_\beta C_\beta\right) b_2 r_1 \\ &= 2\pi\rho B^2\left(-2\dot{\beta} u_P S_\beta - \dot{\beta} u_N C_\beta\right) b_2 r_1 \\ &= 2\pi\rho B^3 (a-\frac{1}{2})\left(+\ddot{\beta} b S_\beta + \dot{\beta}^2 b C_\beta\right) b_3 r_0 \end{aligned} \quad (127)$$

9.11 Summary of assumptions and results

The added-mass forces on a thin, flat wing have been derived using thin aerofoil potential theory. The calculations are as standard cases, except with the following generalisation:

- β, α are not ≈ 0. This means the expressions had to be derived in terms of the parallel and normal velocities. This means there is a parallel component of the velocities

It is assumed that:

1. The flow is entirely inviscid.
2. The flow leaves the trailing edge smoothly, satisfying the Kutta-Joukowski condition.
3. There is no shed wake.
4. The hinge point is constant (where wing shape factors are used).

The third assumption is a direct violation of the Kelvin-Helmholtz theorem for the Kutta-Joukowski terms, as they rely on a net change in total bound vorticity. This assumption also means that the effect of the wake on the added mass forces on the wing is discounted. Unlike the same assumption for the quasi-steady case, an appropriate model for the wake influence on added mass is not available. This is discussed more in Section 11.8.

The main results of this section are the forces in the vertical V and horizontal H directions, at an arbitrary wing section:

$$\begin{aligned} F_{VA} &= \pi\rho b^2\left[\dot{u}_H S_\beta C_\beta + \dot{u}_V C_\beta^2 + 2\dot{\beta} u_P C_\beta - \ddot{\beta} ab C_\beta - \dot{\beta} u_N S_\beta + \dot{\beta}^2 ab S_\beta\right] \\ &+ 2\pi\rho b^2\left[\dot{u}_H S_\beta C_\beta + \dot{u}_V C_\beta^2 + 2\dot{\beta} u_P C_\beta - \ddot{\beta} b(a-\frac{1}{2}) C_\beta - \dot{\beta} u_N S_\beta + \dot{\beta}^2 (a-\frac{1}{2}) b S_\beta\right] \\ F_{HA} &= \pi\rho b^2\left[-\dot{u}_H S_\beta^2 - \dot{u}_V S_\beta C_\beta - 2\dot{\beta} u_P S_\beta + \ddot{\beta} ab S_\beta - \dot{\beta} u_N C_\beta + \dot{\beta}^2 ab C_\beta\right] \\ &+ 2\pi\rho b^2\left[-\dot{u}_H S_\beta^2 - \dot{u}_V S_\beta C_\beta - 2\dot{\beta} u_P S_\beta + \ddot{\beta} b(a-\frac{1}{2}) S_\beta - \dot{\beta} u_N C_\beta + \dot{\beta}^2 b(a-\frac{1}{2}) C_\beta\right] \end{aligned}$$

Assuming the hinge is at the same position for all cross sections, the total forces on the wing are:

$$
\begin{aligned}
F_{VAW} &= \pi\rho RB^2\left(\dot{u}_{HT}\mathrm{S}_\beta\mathrm{C}_\beta + \dot{u}_{VT}\mathrm{C}_\beta^2\right)b_2r_1 \\
&+\pi\rho RB^2\left(2\dot{\beta}u_P\mathrm{C}_\beta - \dot{\beta}u_N\mathrm{S}_\beta\right)b_2r_1 \\
&+\pi\rho RB^3\left(-\ddot{\beta}a\mathrm{C}_\beta + \dot{\beta}^2a\mathrm{S}_\beta\right)b_3r_0 \\
&+2\pi\rho B^2\left(\dot{u}_H\mathrm{S}_\beta\mathrm{C}_\beta + \dot{u}_V\mathrm{C}_\beta^2\right)b_2r_1 \\
&+2\pi\rho B^2\left(+2\dot{\beta}u_P\mathrm{C}_\beta - \dot{\beta}u_N\mathrm{S}_\beta\right)b_2r_1 \\
&+2\pi\rho B^3(a-\frac{1}{2})\left(-\ddot{\beta}\mathrm{C}_\beta + \dot{\beta}^2\mathrm{S}_\beta\right)b_3r_0
\end{aligned}
$$

$$
\begin{aligned}
F_{HAW} &= \pi\rho B^2\left(-\dot{u}_H\mathrm{S}_\beta^2 - \dot{u}_V\mathrm{S}_\beta\mathrm{C}_\beta\right)b_2r_1 \\
&+\pi\rho B^2\left(-2\dot{\beta}u_P\mathrm{S}_\beta - \dot{\beta}u_N\mathrm{C}_\beta\right)b_2r_1 \\
&+\pi\rho B^3\left(+\ddot{\beta}ab\mathrm{S}_\beta + \dot{\beta}^2ab\mathrm{C}_\beta\right)b_3r_0 \\
&+2\pi\rho B^2\left(-\dot{u}_H\mathrm{S}_\beta^2 - \dot{u}_V\mathrm{S}_\beta\mathrm{C}_\beta\right)b_2r_1 \\
&+2\pi\rho B^2\left(-2\dot{\beta}u_P\mathrm{S}_\beta - \dot{\beta}u_N\mathrm{C}_\beta\right)b_2r_1 \\
&+2\pi\rho B^3(a-\frac{1}{2})\left(+\ddot{\beta}b\mathrm{S}_\beta + \dot{\beta}^2b\mathrm{C}_\beta\right)b_3r_0 \qquad (128)
\end{aligned}
$$

10 Polhamus leading edge and tip suction correction

10.1 The leading edge vortex (LEV)

As mentioned in Section 8, the Dirichlet solution for the flow past a flat plate at incidence causes infinite acceleration of the flow at the leading and trailing edges. The Kutta-Joukowski condition avoids the problem at the trailing edge, but the leading edge has not been dealt with yet. At the leading edge, the flow is expected to separate, leading either to a deep stall, or a stable attached vortex above the leading edge, similar to that observed by Ellington et al. [25] (see Section 5). According to subsequent work by Ellington et al. [9], this LEV can be sustained for indefinite periods when the wing is in constant rotation, and up to high angles of attack, so it is reasonable to expect this phenomenon to occur on an FMAV wing. To model the effect of the LEV, Polhamus's analogy is used as a correction to the quasi-steady force found earlier.

10.2 Polhamus's analogy

Polhamus [14] modelled the LEV by assuming that the separation at the leading edge is a "hard" separation, causing total loss of leading edge suction, while the LEV causes a normal force component of equal magnitude. Effectively, the leading edge suction force is rotated by 90° onto the low-pressure side of the wing, as illustrated in Figure 18. This is called the Polhamus Leading Edge Suction Analogy. Although this is a very simple model, it has been shown to give remarkably good results, for example in predicting the attached vortex lift of delta wings—see for example [17].

The Polhamus analogy is desirable for three reasons:

1. It is simple to implement.

2. It is compatible with inviscid potential flow theory.

3. It is easy to extend to complex wing geometries (see next section).

10.3 Correction for leading edge sweep

The leading edge suction of a $2D$ wing section is called the leading edge thrust, since it is in the chordwise direction. For a swept wing, the leading edge suction force will actually be normal to the leading edge, but will still have the same forward thrust component. This means for a swept wing, the leading edge suction will be higher. It is the leading edge suction, not thrust that is rotated in the Polhamus approach. This is described in Bradley et al. [15], who outline a correction to the Polhamus analogy for leading edges that are swept. It relies on the original Polhamus analogy for swept, sharp-tipped wings, and the extension of this theory to rectangular wings by Lamar, which is outlined in [17]. The latter theory uses the Polhamus analogy on the tip suction force, causing an additional normal force component. The scheme of [15] uses these two theories to calculate the vortex lift for an arbitrary wing shape, as the summation of a series of trapezoidal wing sections.

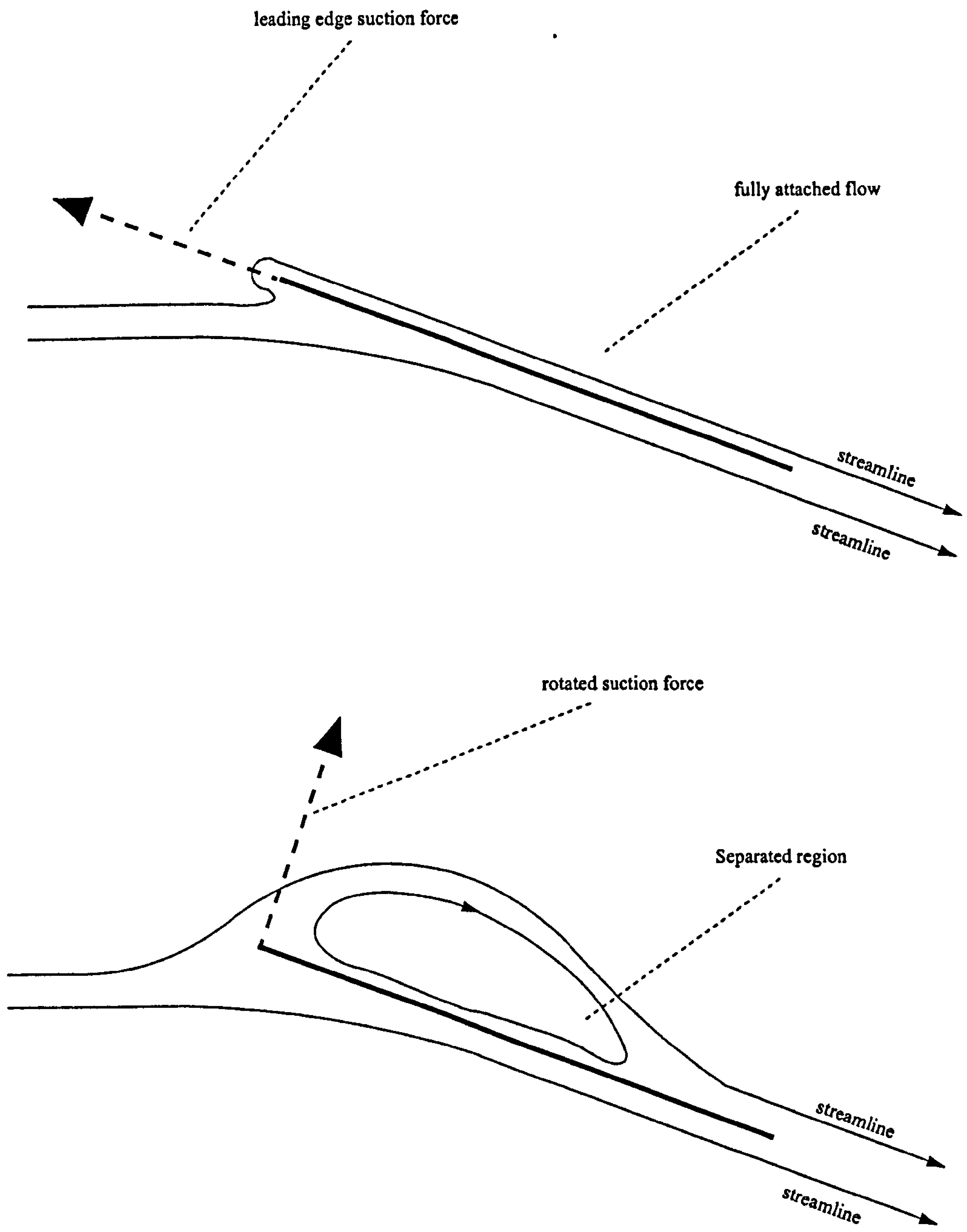


Figure 18: Illustration of the Polhamus leading edge suction analogy.

For the FMAV case considered here, it is assumed the wing comes smoothly to a point at the tip. Because of this, there is no side edge to the wing, and therefore no tip suction, and no need for the Lamar extension described above.

10.4 Implementation

The method outlined in the previous section is used: at any given spanwise position, the leading edge thrust is expressed in terms of the quasi-steady force F_{PQ}, given per unit span. Then the leading edge suction force F_S, is found using the sweepback angle of the leading edge $\angle$:

$$F_S = F_{PQ}/\cos(\angle) \tag{129}$$

Note there is a numerical issue here, if very high resolution is used close to the leading edge, so that $\cos(\angle)$ may become ≈ 0. For this reason, and because it is easier to implement from x, y coordinates of the wing geometry, the rate of change of x_l (the non-normalised chordwise coordinate of the leading edge, with respect to the non-normalised radius), φ is used:

$$F_S = F_{PQ}\sqrt{1+\varphi^2} \tag{130}$$

This can be verified from simple trigonometry.

Unlike the cases considered by other authors, there is a potential source of ambiguity: when the wing is translating slowly, and rotating fast, the normal flow is not necessarily to the same side along the entire chord, see for example Figure 8 on page 21. This is overcome by assuming that the direction the suction force rotates is governed by the normal velocity at the leading edge, since it is here the LEV is initially formed. This, however, gives the strange situation that the LEV magnitude can be influenced by a normal flow at the trailing edge, which is of the wrong sign. In order to alleviate this problem an empirically-inspired correction to this, called Polhamus effect scaling, has been devised.

10.5 Refinement: Polhamus effect scaling

An initial comparison of model predictions with the results of Sane and Dickinson [46] led to an empirically-inspired correction to the Polhamus analogy. It was observed that although the lift correction predicted by the Polhamus model seemed accurate enough, the loss of lift was being over-predicted considerably during the rotation phase at the end of either stroke. It is theorised that this is due to the problem outlined above, that part of the calculated Polhamus force is from normal velocity of the wrong sign. Therefore the Polhamus effect was scaled by the fraction of the suction force that is being generated "correctly", i.e. on the fraction of the chord where the normal velocity is of the correct sign. Effectively, during rotation, only part of the tip suction is being manifest as a normal force, while the remainder is simply lost.

The Polhamus normal suction force is scaled by the fraction of chord where the normal velocity is of the same sign as at the leading edge, by finding ζ_0, the point where the normal velocity is 0:

$$\begin{aligned} u_{NE} &= u_N + b\,\dot{\beta}\,(\zeta - a) \\ 0 &= u_N + b\,\dot{\beta}\,(\zeta_0 - a) \qquad (131) \\ \zeta_0 &= a - \frac{u_N}{b\,\dot{\beta}} \qquad (132) \end{aligned}$$

If ζ_0 falls outside the region $-1, 1$, the entirety of the wing has the same sign of normal velocity; in these cases, $\zeta_0 = 1$. Note this will happen if $\dot{\beta}$ is 0, when the second term goes to infinity. The fraction of the chord represented by this is then the Polhamus scaling:

$$\mathrm{S_P} = \frac{\zeta_0 + 1}{2} \qquad (133)$$

A more accurate result (but considerably more long-winded) is to integrate the actual suction force from -1 to ζ_0, rather than just taking a linear approximation.

For the test cases considered (see Sections 14 to 17), fast rotation occurred while the wing was nearly vertical - at these points, the suction force is almost vertical, so wether it is rotated to become a normal Polhamus force, or simply lost, it won't manifest itself as a lift force. The effect on the horizontal force was small, and very brief, localised at the reversal points, giving downwards "spikes" at these points. This is because the velocities due to the rotation were small compared to the velocities due to translation. For this reason, Polhamus effect scaling was not used in the final implementation, but has been mentioned here for possible further refinement.

10.6 Forces

The forces that result from the Polhamus effect are as follows:

$$\begin{aligned} F_{PP} &= -2\pi\,\rho\,b\,u_{Nr}\,u_{Nm} \qquad (134) \\ F_{NP} &= 2\pi\,\rho\,b\,u_{Nr}\,u_{Nm}\,\sqrt{1+\varphi^2}\,\mathrm{S_P T_P} \qquad (135) \end{aligned}$$

The parallel component F_{PP} is simply the opposite of the leading edge thrust, calculated in Section 8. The normal force is this thrust force, scaled by $\sqrt{1+\varphi^2}$, to become the leading edge suction, as explained in Section 10.3. The last two parameters $\mathrm{S_P}$ and $\mathrm{T_P}$ are the scaling and turn direction mentioned in Section 10.5 and 10.4. $\mathrm{T_P}$ is the sign of the normal velocity at the leading edge.

10.7 Wing integral

Similar to previous sections, the Polhamus forces on the entire wing are calculated by integrating along the wing. There is, however, one complication: since the semichord varies along the wing, the normal velocity at the leading edge will vary as well, and may reverse sign. This is dealt with by assuming that the turn direction is governed by the normal velocity at the leading edge at the point where the chord is maximum. Thus, the force on the entire wing due to the Polhamus effect is:

$$\begin{aligned} F_{PP} &= -2\pi\,\rho\, b\, u_{Nr}\, u_{Nm} \\ &= -2\pi\,\rho\, b\left(u_N^2 + u_N\dot{\beta}b\left(\frac{1}{2}-2a\right)+\dot{\beta}^2 b^2\left(a^2-a/2\right)\right) \end{aligned} \tag{136}$$

$$\begin{aligned} F_{PPW} &= -2\pi\,\rho\, B\, R\, u_{NT}^2\, b_1 r_2 \\ &\quad -2\pi\,\rho\, B^2\, R\,\dot{\beta} u_{NT}\left(\frac{1}{2}-2a\right) b_2 r_1 \\ &\quad -2\pi\,\rho\, B^3\, R\,\dot{\beta}^2\left(a^2-a/2\right) b_3 r_0 \end{aligned} \tag{137}$$

Ignoring the turn direction and scaling above, the normal force is formed, assuming that the entire suction force is an upward normal force.

$$\begin{aligned} F_{NP} &= 2\pi\,\rho\, b\, u_{Nr}\, u_{Nm}\sqrt{1+\varphi^2} \\ &= 2\pi\,\rho\, b\sqrt{1+\varphi^2}\left(u_N^2 + u_N\dot{\beta}b\left(\frac{1}{2}-2a\right)+\dot{\beta}^2 b^2\left(a^2-a/2\right)\right) \end{aligned} \tag{138}$$

$$\begin{aligned} F_{NPW} &= -2\pi\,\rho\, B\, R\, u_{NT}^2\, b_1 r_{2P} \\ &\quad -2\pi\,\rho\, B^2\, R\,\dot{\beta} u_{NT}\left(\frac{1}{2}-2a\right) b_2 r_{1P} \\ &\quad -2\pi\,\rho\, B^3\, R\,\dot{\beta}^2\left(a^2-a/2\right) b_3 r_{0P}, \end{aligned} \tag{139}$$

where the wing shape parameters $b_2 r_{1P}$ are similar to the standard wing shape parameters, except they include the effect of leading edge sweep. For example:

$$b_2 r_1 = \int_{-1}^{1}\left(\frac{b}{B}\right)^2 r^1\, dr \tag{140}$$

$$b_2 r_{1P} = \int_{-1}^{1}\left(\frac{b}{B}\right)^2 r^1\sqrt{1+\varphi^2}\, dr \tag{141}$$

10.8 Summary of assumptions and results

The Polhamus analogy has been used to derive the force corrections to the quasi-steady forces, due to the flow at the leading edge not being attached, but forming a leading edge vortex. The calculations are as standard cases, except with the following two generalisations:

- β, α are not ≈ 0. This means the expressions had to be derived in terms of the parallel and normal velocities.
- The wing is not in fast forward motion. This means that the rotational component of the velocity can be considerable compared to the translational component.

The first generalisation means that the Polhamus correction employed becomes an additional normal component, not necessarily an additional lift component as described by standard cases. The second generalisation means there is some ambiguity in how the Polhamus force should be scaled.

It is assumed that:

1. The flow is entirely inviscid.
2. The flow separates sharply from the leading edge, causing total loss of leading edge suction.
3. The flow always reattaches, and forms a leading edge vortex.
4. The effect of the LEV is to rotate the leading edge suction force by 90°.
5. The direction of the above rotation is in the direction of the normal velocity at the leading edge.
6. There is no effect of the leading edge separation on the wake.
7. There is no effect of the leading edge separation on the added mass.

The sixth item violates the Kelvin-Helmholtz theorem. There is no modelling of the effect of the Polhamus correction on the wake vorticity, because it cannot be modelled as a simple correction in bound vorticity. Remember that a change in bound vorticity will cause a change in the force normal to the incoming flow, which is not necessarily the direction of the Polhamus correction force. The third assumption means that the current model can only be used for kinematic regimes where stalling does not occur, as no prediction or simulation of stall in included. The final assumption means the flow at the leading edge is modelled differently in the added mass and Polhamus models: for added mass it is assumed that the flow stays attached, while for the Polhamus it is not. Also note that the contribution of Polhamus to pitching moment is ignored. This can be done by expressing the pressure difference due to Polhamus as a chordwise distribution, for example using the expression of Purvis [16].

The following assumptions apply to the integral over the span:

- The hinge point is constant (where wing shape factors are used)
- The direction of force rotation for the entire wing is based on conditions at the radial position where the semichord is maximum.
- The rotated suction force is scaled with the fraction of the chord that is experiencing normal velocity of the same sign as the leading edge - the remainder is lost.
- The wing tapers to a point at the tip, so no tip suction correction is needed.

The main results of this section are the correction to the quasi-steady forces, due to the LEV.

$$F_{PP} = -2\pi \rho \, b \, u_{Nr} \, u_{Nm} \tag{142}$$

$$F_{NP} = 2\pi \rho \, b \, u_{Nr} \, u_{Nm} \sqrt{1+\varphi^2} \, \mathrm{S_P T_P}, \tag{143}$$

where the sign of F_{NP} depends on the scheme employed to decide the direction of rotation, $\mathrm{T_P}$ - i.e. which side of the wing the LEV is expected to occur.

11 Wake effects

11.1 Potential form of wake model

The inviscid potential model of the wake is to treat it as a thin, continuous filament of vorticity being shed from the trailing edge of the wing, where any change in the bound circulation of the wing will cause an equal and opposite circulation to appear in the wake, to satisfy the Kelvin-Helmholtz theorem. In a real flow, the induced velocity due to the vorticity of the wing and wake will combine to cause motion and deformation of the wake filament. The presence of viscosity will introduce decaying effects into this process.

Assuming that the only motion of the wake is due to the uniform induced velocity u_i (the downwash), a wake shape similar to that of Figure 19 is obtained. Note, however, that in this model it is entirely possible for the wake to intersect with the wing, and for the wake to intersect itself.

The wake is by far the most complex part of the flow - in order to simplify it enough to be able to isolate its effect in an analytically tractable, considerable simplifications had to be introduced. Thus, experimental verification is seen as absolutely vital, but has not been done rigorously in this thesis.

This simplification process is set in context by considering first exact solutions to simplified $2D$ cases: the Wagner, Küssner and Loewy models.

11.2 Wagner's model

Wagner [30] assumed the wing to be moving at a changeable forward velocity and angle of attack. The angle of attack was assumed small, and the velocity horizontal, so the wake filament becomes a straight horizontal line behind the wing. He assumed this filament did not deform or move. Then he applied the quasi-steady force equations, similar to those of Section 8. However, the difference was that he integrated the effect of the entirety of the bound vorticity *and* the shed vorticity, under the same assumption that there is no flow penetration on the wing. This reduced to a different expression for the bound vorticity, and hence lift, which was the original quasi-steady (wakeless) result plus a correction based on the effect of the wake. This was expressed as a function of the distance travelled in semichords measured since a given change in either angle of attack or forward velocity. From superposition, the change due to a time series of such changes can be expressed by simply summing the effect of every single change (using Duhamel's theorem, see e.g. Leishman [3]).

Although the change was expressed in terms of changes of either angle of attack or forward velocity, both of these are actually expressions of the product of the bound vorticity and the forward velocity. Therefore, they are expressed here in terms of changes of the lift coefficient C_L. Remember from Section 7 that the lift coefficient is normalised by using the r.m.s. total velocity of the free stream.

The Wagner function can be approximated by:

$$\psi'_W(s) = 1 - 0.165e^{-0.041s} - 0.335e^{-0.3s}, \tag{144}$$

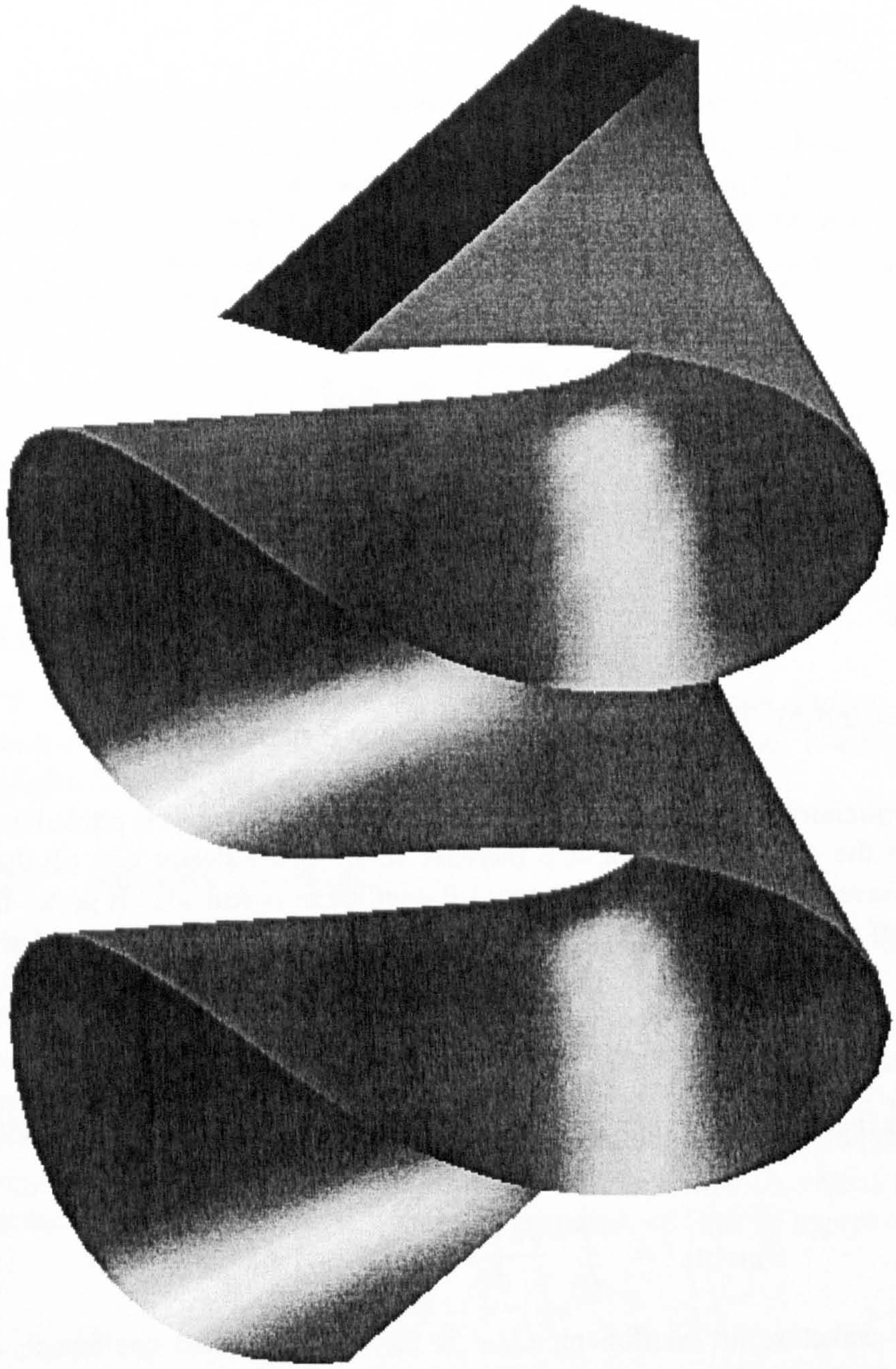


Figure 19: An example of the wake shape below a flapping wing, assuming uniform downwash velocity.

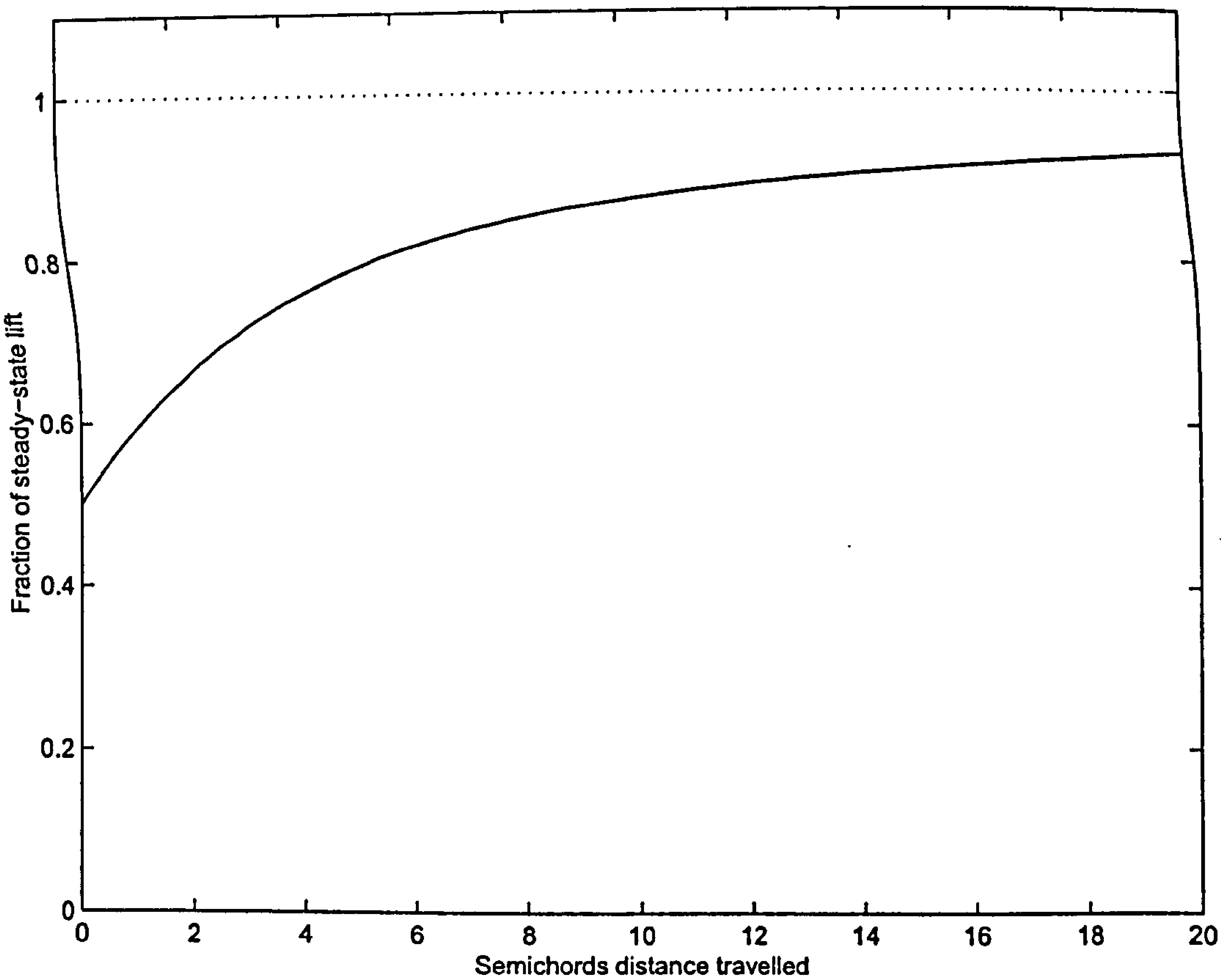


Figure 20: The Wagner function. Note that it starts from 1/2.

where s is the semichord distance travelled by the aerofoil. This function is plotted in Figure 20. It expresses the delay between a step increase in the quasi-steady C_L, till the wake induced effects have decayed, and the full new lift coefficient is realised. It grows from $\frac{1}{2}$, meaning only half the lift from a step change is realised at once, and goes asymptotically to 1 as $s \rightarrow \infty$.

11.2.1 Duhamel expression

Using Duhamel's theorem (see Appendix A.2.2 or [3]), the effect of a series of step changes in C_L is:

$$C_{LW}(s) = \int_{s_0}^{s_1} \frac{dC_L(\sigma)}{d\sigma} \psi'_W(s-\sigma)\, d\sigma, \tag{145}$$

where C_L is the wakeless lift coefficient, C_{LW} is the wake-modified coefficient, σ is a dummy variable for integration, and the motion goes from position s_0 to s_1. It is assumed that no changes in C_L occurred before position s_0.

11.2.2 Perturbation expression

The perturbation Wagner function (ψ_W) is defined as a perturbation from the quasi-steady lift *after* the step change. This is simply the expression of Equation 144 minus 1.

$$\psi_W(s) = \psi'_W(s) - 1 \quad (146)$$
$$= -0.165e^{-0.041s} - 0.335e^{-0.3s} \quad (147)$$

The perturbation form of the Wagner function in Equation (147) can serve as a correction to the wakeless quasi-steady result from Section 8. If the original Wagner function in Equation (144) were used, the quasi-steady component would be included twice - once in the quasi-steady calculation, and once in the non-perturbation form of the Wagner function.

The Duhamel sum of a series of changes in quasi-steady C_L, using the perturbation Wagner function is:

$$C_{LW} = \int_{s_0}^{s_1} \frac{dC_L(\sigma)}{d\sigma} \psi_W(s-\sigma)\, d\sigma \quad (148)$$

This is the change in C_L due to the effect of the wake *only* ignoring the step change in C_L itself, which is already part of the quasi-steady solution. Again, it is assumed that no changes in C_L occurred before s_0.

11.2.3 Assumptions for Wagner model

The Wagner function assumes the effects of individual step changes to be linearly superposable, and that they are stationary in absolute space. Also, it assumes the wake to be a straight, stationary filament behind the wing.

11.3 Küssner's model

The Küssner wake model was introduced by Küssner [31], but note that this reference contained a sign error, which was corrected in [42]. The Küssner model is very similar to the Wagner model, making the same assumptions about the wake being straight, horizontal and immobile in absolute space. However, instead of a change that applies to the entire wing at once, it considers a step increase in C_L that is stationary in space. This could, for example be a vertical gust region. The increase in C_L does not apply everywhere along the wing, but propagates along it as the wing moves into the increased C_L region (see Figure 21). The Küssner function is also an expression based on s, and can be approximated by

$$\psi'_K(s) = 1 - \frac{1}{2}e^{-0.13s} - \frac{1}{2}e^{-s} \quad (149)$$

(see Figure 22). As expected, it grows from 0, where the increased C_L region is first encountered at the leading edge, but has not yet affected any of the wing, and goes asymptotically to 1 as $s \to \infty$ where the gust-disturbed flow is the new steady condition. Also note that the Küssner model includes the added mass effect, which the Wagner model does not.

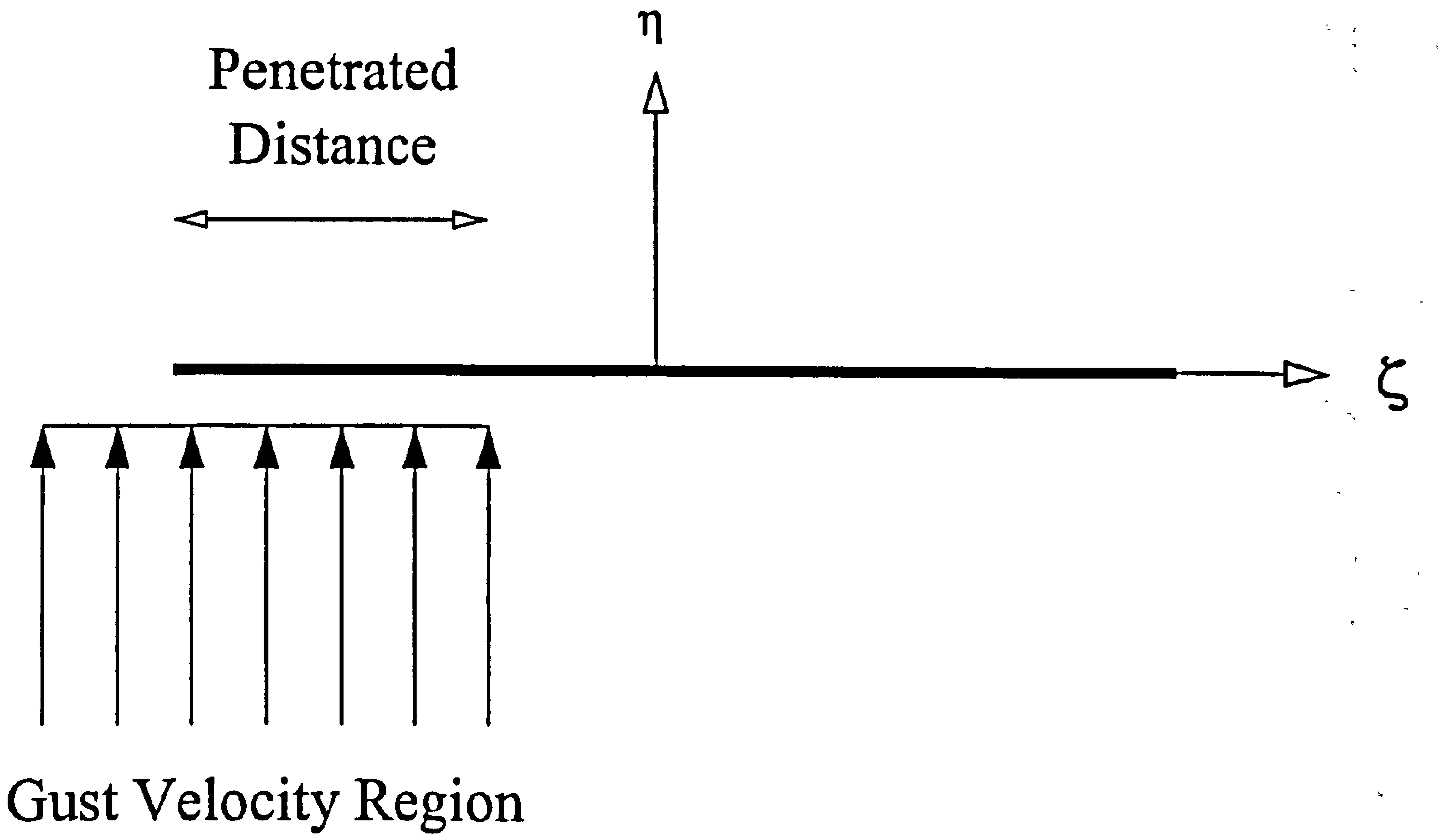


Figure 21: Sharp-edged gust penetration and the Küssner effect

11.3.1 Duhamel expression

Using Duhamel's theorem (see Appendix A.2.2 or [3]) the effect of a series of C_L regions is:

$$C_{LK} = \int_{s_0}^{s_1} \frac{dC_L(\sigma)}{d\sigma} \psi'_K(s-\sigma)\, d\sigma \tag{150}$$

where C_L is the wakeless lift coefficient, C_{LK} is the wake-modified coefficient, σ is a dummy variable for integration, and the motion goes from s_0 to s_1. It is assumed that no changes in C_L occurred before s_0.

11.3.2 Perturbation expression

The perturbation Küssner function (ψ_K)is defined as a perturbation from the quasi-steady lift *after* the step change. This is simply the above expression minus 1.

$$\begin{aligned} \psi_K(s) &= \psi'_K(s) - 1 \\ &= -\frac{1}{2}e^{-0.13s} - \frac{1}{2}e^{-s} \end{aligned} \tag{151}$$

The perturbation expression of Equation (151) can serve as a correction to the wakeless quasi-steady result. If the original expression of Equation (149) were used, the quasi-steady component would be included twice.

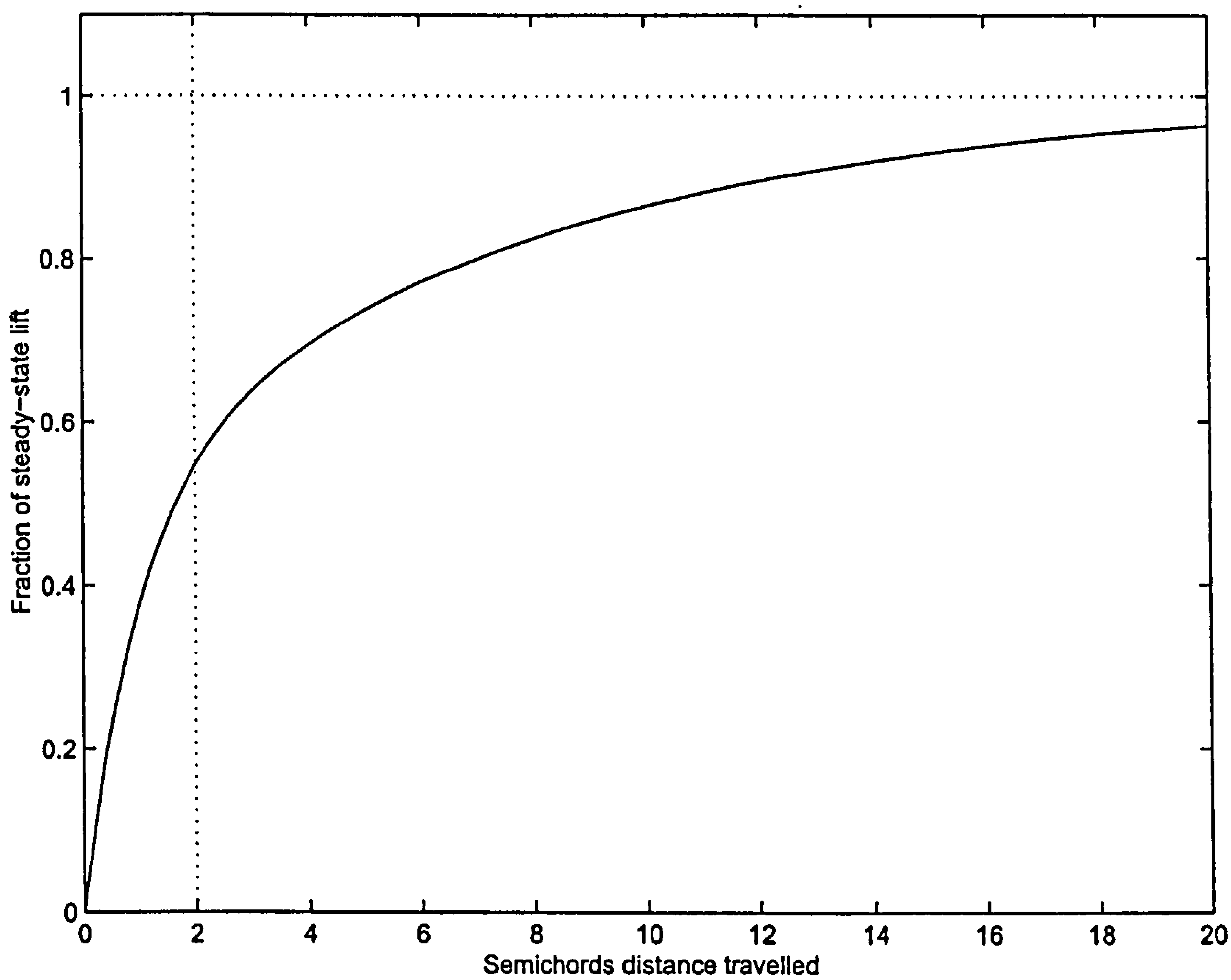


Figure 22: The Küssner function. The vertical line represents the point where the gust has fully propagated along the wing. Note that the Küssner function starts from 0.

The Duhamel sum of a series of changes in C_L, using the perturbation Küssner function, is:

$$C_{LK} = \int_{s_0}^{s_1} \frac{dC_L(\sigma)}{d\sigma} \psi_K(s-\sigma)\, d\sigma \qquad (152)$$

This is the change in C_L due to the effect of the wake *only* ignoring the step change in C_L itself, which is already part of the quasi-steady solution. Again, it is assumed that no changes in C_L occured before s_0.

11.3.3 Assumptions for Küssner model

The Küssner function assumes the effects of individual step changes to be linearly superposable, and that they are stationary in absolute space. Also, it assumes the wake to be a straight, stationary filament behind the wing. The regions of increased C_L are assumed to be stationary in absolute space, and propagate along the wing at the wing forward velocity.

11.4 Comparison of Wagner and Küssner functions

Figure 23 shows the comparison between the Wagner and Küssner functions after full gust penetration. In the Wagner case, this is instantaneous at $s = 0$, while in the Küssner case it occurs at $s = 2$. The Küssner function tends faster to 1 because of the more gradual introduction of vorticity into the wake, which has been happening for an entire chord length before $s = 0$ on the graph. Note again that the Küssner function includes added mass effects, which the Wagner function does not.

11.5 Loewy's model

When a helicopter hovers, the wake trailed behind the rotor is convected downwards ("downwashed"). Since the rotor is rotating, when it returns to the same position in the rotation, it will pass over the wake it has shed earlier (see Figure 24). Loewy [47] modelled this inviscidly by treating the wake as a straight horizontal vortex filament, as for the Wagner and Küssner models (see Figure 25, parts a,b and c). He assumed that the vorticity of the wing was varying sinusoidally, with spatial wavelength λ. He furthermore assumed that the helicopter had been in a steady hover for a long time, so the wake behind the rotor extended to infinity. This is the *primary* wake. The novelty of the Loewy approach was that he then modelled the encounter of previous wakes by reproducing the primary wake below the rotor, saying that during the cycle the wake would have moved downwards due to the uniform induced downwash u_i. He therefore modelled the wake passage as an infinite series of copies of the primary wakes, each offset a constant distance down and advanced in phase by a constant amount.

It may be initially counter-intuitive that the same point of the wake is treated as being in more than one place—occurring not just behind the wing, but also in successive wakes, each time further down and further advanced in phase. This was done because it makes the

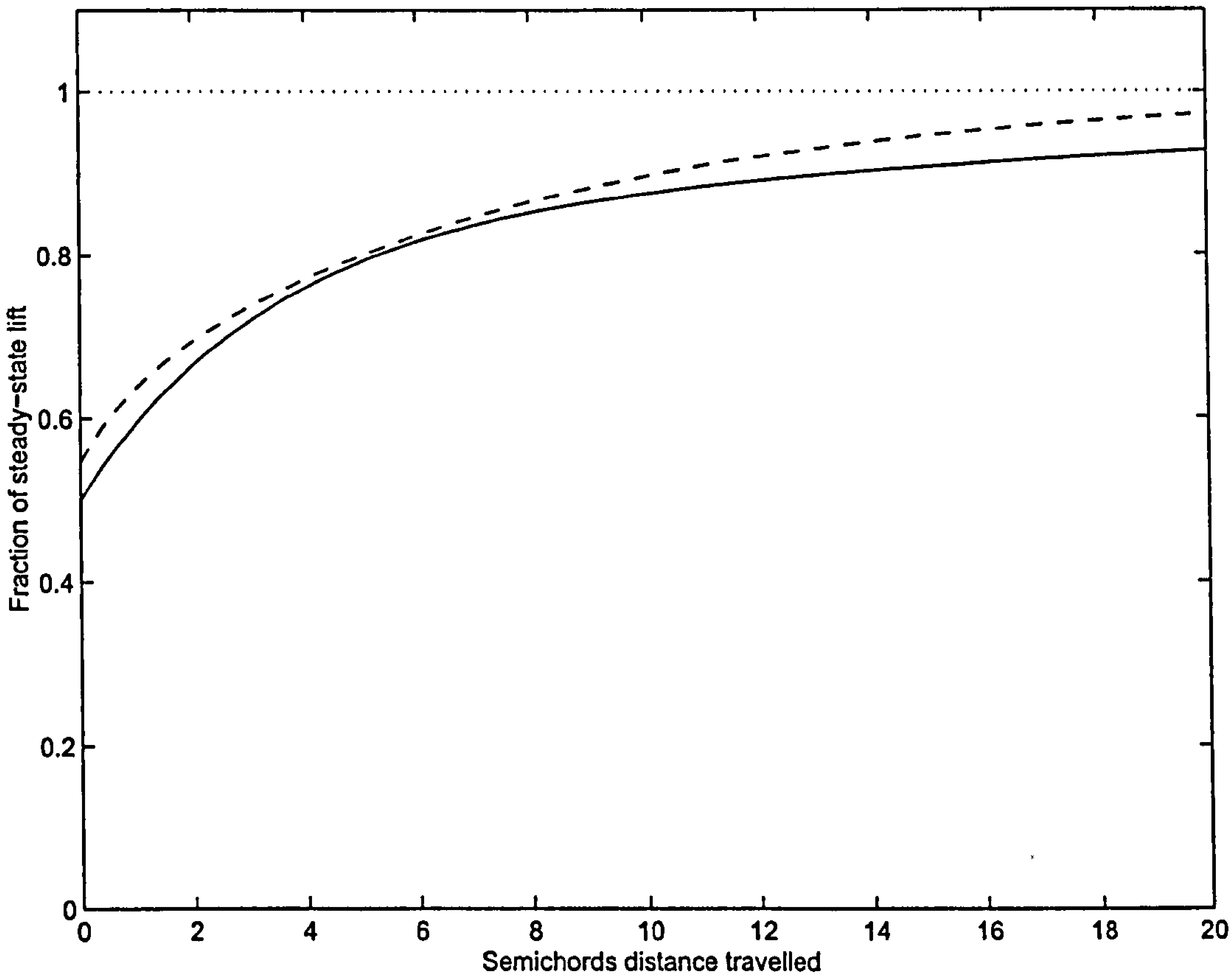


Figure 23: Comparison of the Wagner function (solid) versus the fully penetrated Küssner function (dashed). The Küssner function is offset to $s = 2$, which is where full penetration occurs.

summation limits infinity in both horizontal and vertical extent, meaning they will reduce to an algebraic form. The justification for this is:

1. The sinusoidal wake element will tend to cancel out far from the wing, as the distance between them becomes small compared to the distance to the wing.

2. The most pronounced effect of a wake element is the point where it is closest to the wing—so the wake element a full revolution of travel behind the wing has little effect compared to the wake element immediately below the wing. This comes from the Biot-Savart law, see e.g. Leishman [3].

Loewy furthermore extended this theory to an arbitrary number of rotor blades, by dividing the offset distance and phase difference between wakes by the number of rotor blades, assuming the phase angle of all blades was identical at the same point in the rotation.

11.6 Modified Loewy model

An attempt was made to form an equivalent of the Loewy expression for the case of a flapping wing, by splitting the wake into single-stroke parts, and offsetting each downwards by the amount determined by the downwash, as seen in Figure 25. However, the Loewy expressions assume the wake extends to infinity in the up- and downstream directions, in order to reduce the result to algebraic form. This assumption gives inaccurate results in the FMAV case, because the filaments are of finite length, which is comparable with the wing chord length. The details of this derivation are shown below, for the purpose of future refinement. It has not, however, been used in this work.

Although the algebraic expressions of Loewy have not been employed, his principle of secondary wakes has been utilised—treating the previous wakes as constantly offset straight vortex filaments below the wing. The difference is that computation is performed as a direct sum over the secondary wakes, rather than as closed form expressions. The output of this model is the induced velocity at a point on the wing. Note that while Loewy's assumption that the distance between wakes is constant was justified in that the time between wakes is constant, in the case of flapping flight, the extreme ends of the stroke should actually meet to form a continuous filament. This is another signifcant simplification that was deemed necessary.

This Section outlines an adaptation of the Loewy approximation of helicopter wake effects [47], to flapping flight with stroke reversal.

Firstly, the nomenclature of Loewy is collected:
n is the number of whole revolutions completed.
Q is the total number of blades (an integer).
q is the number of the current blade, starting from 0.
γ_a is the bound (attached) vorticity —this is called γ_b in the following.
γ_{00} is the wake vorticity in the primary wake (behind the wing).
γ_{nq} is the wake vorticity in the secondary wakes (below the wing).

h is the vertical separation between full revolution blocks (from one n to the next.)
Γ_b is the total bound vorticity.

For the FMAV case, only one blade is considered, so $Q = 1$ and $q = 0$ everywhere. Although there is another wing, it never passes under the first, since the wings are not performing full revolutions. This simplifies Loewy's expression for induced velocity u_w to:

$$u_{nw} = \frac{-1}{2\pi}\left[\int_{LE}^{TE} \frac{\gamma_a}{x-\zeta}\, d\zeta + \int_{TE}^{\infty} \frac{\gamma_{00}}{x-\zeta} d\zeta + \sum_{n=1}^{\infty} \int_{-\infty}^{\infty} \frac{\gamma_{n0}(x-\zeta}{(x-\zeta)^2 + n^2h^2}\, d\zeta\right] \tag{153}$$

This is simply an expression of the 2-D Biot-Savart law for all the vortical elements. The first term is the bound vorticity, the second term is the primary wake vorticity, and the summation of the final term is all of the secondary wakes.

Next, Loewy used the reduced frequency to express the above spatial integrals in terms of time. As discussed in Section 7, this reduction cannot be used here, because the forward velocity is not constant. However, the spatial distribution of the wake vorticity, in terms of the length travelled s, can be found and represented as a sum of sinusoidal elements using the fast Fourier transform. The induced velocity, caused by each sinusoidal element, can be calculated, and the vector sum of velocities formed to produce the total effect. In this way, the finite extent of the secondary wakes is taken into account directly, while the essence of Loewy's approach is preserved through the Fourier decomposition of the wake's vorticity.

11.7 Combined wake model

The models of Sections 11.2, 11.3 and 11.5 are combined to form a wake model of the actual, complex wake shape behind and below a flapping wing. Firstly, the wake is split into single-stroke segments, similarly to Loewy's model. The *primary* wake extends backwards in a horizontal line, to the start of the current stroke, and a number of *secondary* wakes, due to previous strokes, that are horizontal lines, each one offset by the distance $u_i T/2$ below the later stroke, where u_i is the average downwash velocity, and T is the period of a complete cycle, so $T/2$ is the period of a single stroke (see Figure 25).

The analysis is restricted to $2D$, assuming that the wing can be treated as a series of $2D$ spanwise segments, that do not affect each other. Also, this assumes that there is no spanwise flow.

The start and end of the stroke are governed by the position of the trailing edge, which is where the wake is being shed from.

The primary wake is assumed to be a line, so the effect of the primary wake can be treated as a Wagner-type effect, by applying the Wagner function to the changes in quasi-steady lift coefficient since the start of the current stroke. It is assumed that the compounded effect of the Wagner contributions instantly disappear at the start of a new stroke. Special care needs to be taken at the start of the stroke. If it were treated as a purely impulsive start in a wake-free fluid, then whatever bound vorticity the wing has would result in an impulsive (and large) shed vortex. This is unrealistic, as it would be an artefact of arbitrarily dividing

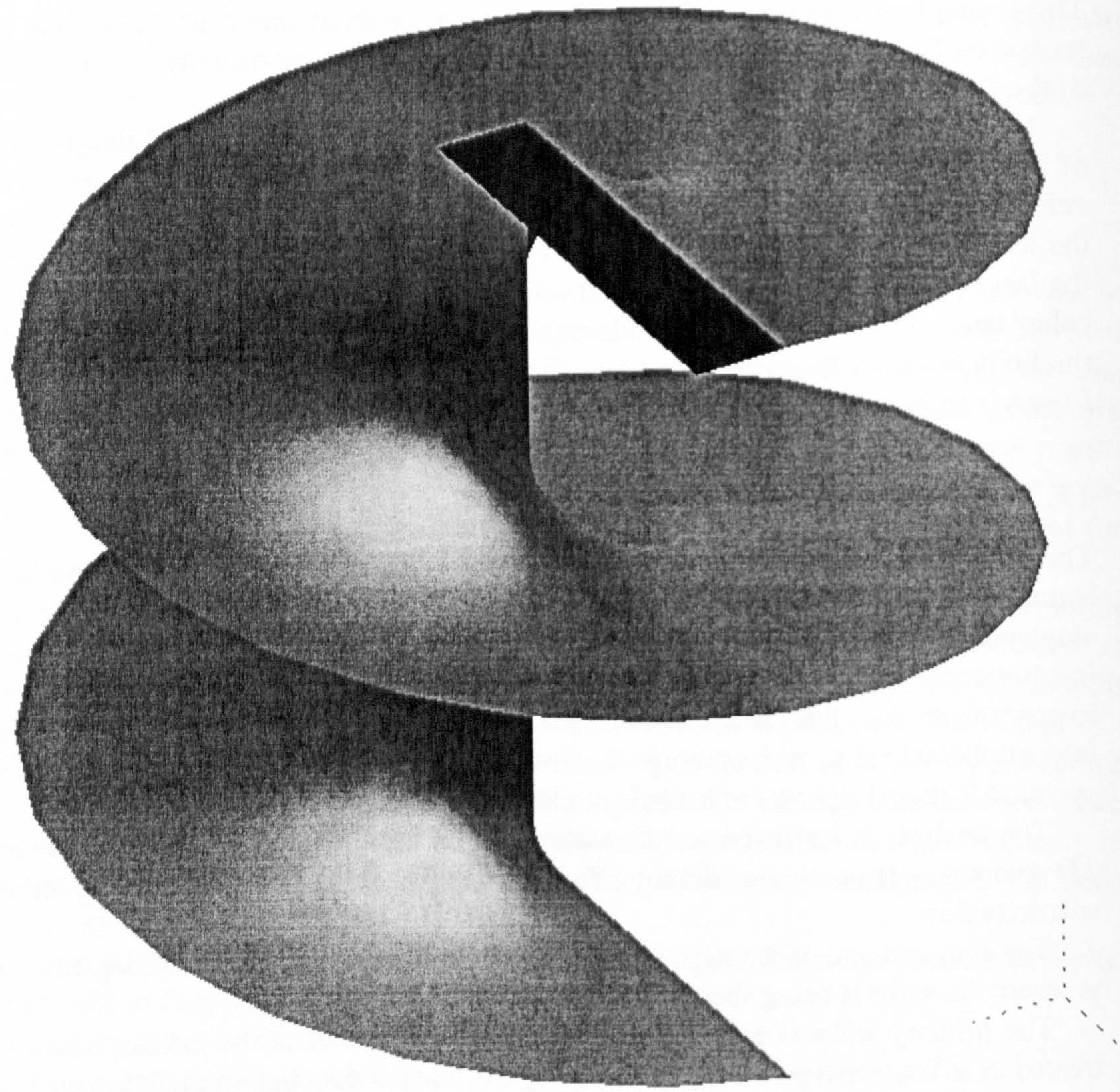


Figure 24: Sample 3D wake under a constantly rotating wing, similar to Loewy's model.

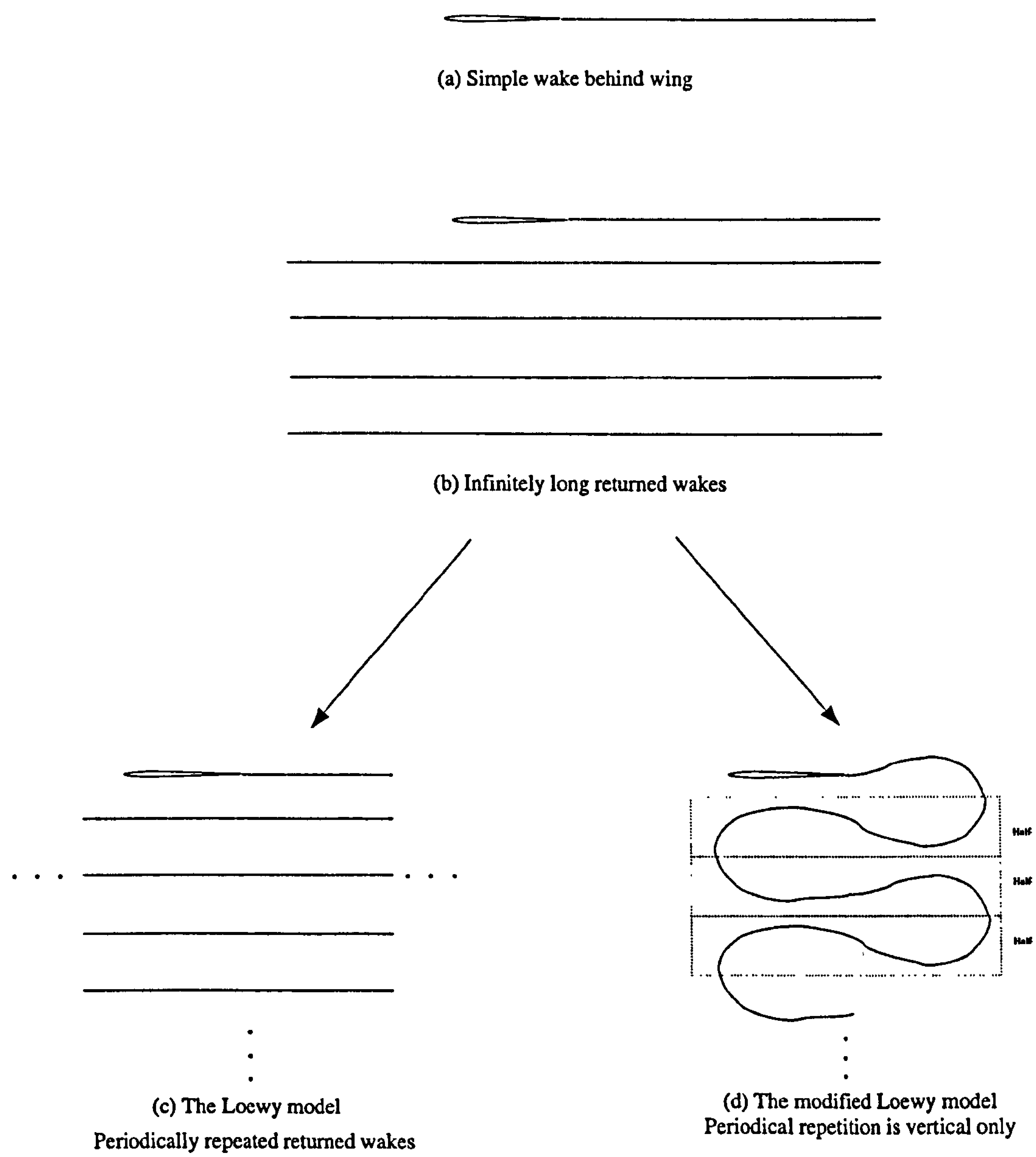


Figure 25: Illustration of the original and modified Loewy returning wake model

the wake into single-stroke segments. Instead, only the change in C_L between strokes is used - this is the step change in C_L from the end of one stroke to the start of the next.

The effect of the secondary wakes is incorporated by calculating the induced velocity at the leading edge, due to the vorticity of all the secondary wakes, in a Loewy type sum. These secondary wakes are assumed to be straight lines, and globally stationary, so the flowfield they cause is also globally stationary. The velocities induced by the secondary flowfield are treated as stationary gusts, and their effect on the wing is modelled as a Küssner-type effect. This is done by calculating C_L without and with the secondary wake-induced velocities, and treating the difference in C_L as a series of Küssner perturbations. Again, the wake has been split into single-stroke segments, so this is not treated as an impulsive start in a wake-free fluid, but the starting value of C_L is ignored.

The effects of the primary and secondary wakes are treated as entirely separate, and superposable.

11.8 Added mass and the wake

Added mass does not affect the wake - it is irrotational. However, as the wake affects the velocity of the fluid around the wing, it will obviously have an effect on the added mass. This has not been modelled accurately, because no Wagner type function exists for this effect. The Küssner function already includes the effect of added mass.

11.9 Polhamus correction and the wake

Unlike the added mass, it is expected that the LEV will cause a change in the bound vorticity, and therefore wake vorticity. For usual cases, where the incoming velocity is approximately parallel to the chord, the increased normal force due to the rotation of the leading edge suction can be modelled by an increase in bound circulation of the wing. However, for flapping kinematics, the incoming velocity is not approximately parallel to the chord. Therefore, for our case we cannot use a vorticity model of the Polhamus correction, as mentioned in Section 10.

For this reason, the effect of the LEV on the wake cannot be modelled accurately. A first-order correction for the effect of the *LEV* has been applied, by using the Polhamus-modified C_L in the wake calculations. However, note that this is a first-order model at best.

Similarly, because the Wagner and Küssner functions treat the wing and wake as horizontal, they predict the force that results to be purely normal to the wing, so no parallel component exists. This means they do not have an effect on the leading edge suction. It is technically possible to derive expressions similar to Wagner and Küssner's function, but for the leading edge suction. However, if this is applied to the Polhamus effect, it leads to an iterative model, where the results of an earlier function are affected by the results of a later one.

11.10 Summary of assumptions and results

A new model has been proposed to account for the effect of an inviscid wake filament on the wing forces, using the Wagner, Küssner and Loewy wake models. In order for the model to be transparent, several simplifying assumptions have been made:

- The wake is treated as a thin, globally stationary filament of vorticity.
- The wake is split into single-stroke elements, each of which is assumed to be a straight line.
- Each wake segment is assumed to be behind the wing until reversal, where all previous wakes jump downwards by a distance based on the average predicted downwash velocity.
- In using the Wagner function for the primary wake, it is assumed implicitly that the wake is flat and horizontal, and that the wing pitch angle is low. Therefore, the model predicts no parallel or horizontal velocities or forces due to the primary wake.
- In using the Loewy-like shape for the wake, it is assumed that the wake can be modelled by breaking it into single-stroke segments, and that the wake moves downwards in discrete jumps at the end of every stroke. The error due to this will be greatest at the start of every stroke.
- In using Küssner for the effect of the secondary wake induced velocities, it is assumed that the velocity field is globally stationary, and the wing pitch angle is low. The error due to this is expected to be greatest at either end of the stroke, where the pitch angle is large.

The main results of this Section are the perturbation forms of the Wagner and Küssner function, and the expression for the wake-induced velocity at a point:

$$C_{LW} = \int_{s_0}^{s_1} \frac{dC_L(\sigma)}{d\sigma} \psi_W(s-\sigma)\, d\sigma \tag{154}$$

$$C_{LK} = \int_{s_0}^{s_1} \frac{dC_L(\sigma)}{d\sigma} \psi_K(s-\sigma)\, d\sigma \tag{155}$$

$$u_{nw} = \frac{-1}{2\pi}\left[\int_{LE}^{TE} \frac{\gamma_a}{x-\zeta}\, d\zeta + \int_{TE}^{\infty} \frac{\gamma_{00}}{x-\zeta} d\zeta + \sum_{n=1}^{\infty} \int_{-\infty}^{\infty} \frac{\gamma_{n0}(x-\zeta}{(x-\zeta)^2+n^2h^2}\, d\zeta\right] \tag{156}$$

12 Scaling

As mentioned in the introduction, the planned FMAV will be considerably larger and heavier than most insects. It would not be prudent geometrically to scale an insect's wing geometry and kinematics, and expect to get the same aerodynamic performance. One benefit of having derived the formulae of the preceding sections is that a simple scaling analysis of the forces and moments can be performed, and some merit criteria investigated.

This scaling analysis is similar in nature to that undertaken by Ellington in [20], and indeed many of the results here match those found in that paper. The scaling parameters are chosen as:

R	is the wingtip radius
R_m	is the mass scale $m^{1/3}$
f	is the frequency
Λ	is the aspect ratio
θ_T	is the sweep amplitude

It is assumed that the wing tip trace retains its shape in the spherical coordinate system, so the plunge amplitude scales with θ_T as well.Note the mass scale R_m - this is simply a way of comparing the mass of the $FMAV$ to the scale R. Some authors perform this comparison by comparing R^3 and mass m.

The scaling of some basic parameters is:

b	$\propto$	$R\Lambda$	wing semichord
β	$\propto$	1	pitch angle
$\dot\beta$	$\propto$	f	pitch rate
$\ddot\beta$	$\propto$	f^2	pitch acceleration
$\dot\beta b$	$\propto$	$R\, f\, \Lambda$	
u	$\propto$	$R\, f\, \theta_T$	wing velocity
$\dot u$	$\propto$	$R\, f^2\, \theta_T$	wing acceleration

Note that $\dot u$ is the complete derivative of u, i.e. including the Euler term $\dot\beta u$ (see Section 9.7), from the above, it is clear that both the translational part of the acceleration (which is proportional to uf) and the Euler acceleration (which is proportional to $\dot\beta u$) scale similarly.

The last two lines are for all components of velocity u on the hinge line, for a given radial position, even the tip.

Consider the Reynolds number, Re, which is based on a length scale l, typically the wing semichord:

$$\begin{aligned} Re &= \frac{ul}{\nu} \\ &\propto ub \\ &\propto R\, f\theta_T\, b \\ &\propto R^2\, f\, \theta_T\, \Lambda \end{aligned} \tag{157}$$

This is identical to that derived by Ellington [20]. If instead of the wing semichord, l is based on the wing tip radius, or even the square root of the wing surface or normal area, the scaling will be as above, but with different factors of $\wedge$, depending on which length scale is chosen.

Examining the quasi-steady force equations for parallel force F_{PQW}, yields:

$$\begin{aligned}
F_{Q1} &= 2\pi\rho\, R\, B\, u_{NT}^2\, b_1 r_2 \\
&\propto R\, B\, (R f \theta_T)^2 \\
&\propto R^3\, B\, f^2\, \theta_T^2 \\
&\propto R^3\, R\, \wedge\, f^2\, \theta_T^2 \\
&\propto R^4\, f^2\, \theta_T^2\, \wedge \qquad (158) \\
F_{Q2} &= 2\pi\rho\, R\, B^2\, u_{NT}\, \dot{\beta}\, (\frac{1}{2} - 2a) b_2 r_1 \\
&\propto R\, B^2\, (R\, f\, \theta_T) f \\
&\propto R^2\, B^2\, f^2\, \theta_T \\
&\propto R^2\, (R\, \wedge)^2\, f^2\, \theta_T \\
&\propto R^4\, f^2\, \theta_T\, \wedge^2 \qquad (159) \\
F_{Q3} &= 2\pi\rho R B^3 \dot{\beta}^2 a^2 b_3 r_0 \\
&\propto R\, B^3\, f^2 \\
&\propto R\, (R\wedge)^3\, f^2 \\
&\propto R^4\, f^2\, \wedge^3 \qquad (160)
\end{aligned}$$

Note how all the above components scale with $R^4\, f^2$, with varying factors of θ_T and $\wedge$. The quasi-steady force resolved in all other directions will have terms similar to the above, and will therefore scale similarly. Also, since Polhamus is based on the leading edge suction, it will scale similarly to the above.

Examining the added mass forces, the components are split as:

$$\begin{aligned}
F_{VADW1} &= \pi\rho R B^2 \left(\dot{u}_{HT} \mathrm{S}_\beta \mathrm{C}_\beta + \dot{u}_{VT} \mathrm{C}_\beta^2\right) b_2 r_1 \\
&\propto R\, B^2\, \dot{u} \\
&\propto R\, B^2\, R f^2\, \theta_T \\
&\propto R^2\, B^2\, f^2\, \theta_T \\
&\propto R^2\, (R\wedge)^2\, f^2\, \theta_T \\
&\propto R^4\, f^2\, \theta_T\, \wedge^2 \qquad (161) \\
F_{VADW2} &= \pi\rho R B^2 \left(2\dot{\beta} u_P \mathrm{C}_\beta - \beta u_N \mathrm{S}_\beta\right) b_2 r_1 \\
&\propto R\, B^2\, \dot{\beta}\, (u) \\
&\propto R\, B^2\, f\, (f R \theta_T) \\
&\propto R^2\, B^2\, f^2\, \theta_T \\
&\propto R^2\, (R\wedge)^2\, f^2\, \theta_T
\end{aligned}$$

$$
\begin{aligned}
&\propto R^4\, f^2\, \theta_T\, \Lambda^2 && (162)\\
F_{VADW3} &= \pi\rho R B^3 \left(-\ddot{\beta} a \mathrm{C}_\beta + \dot{\beta}^2 a \mathrm{S}_\beta\right) b_3 r_0 \\
&\propto R\, B^3\, \ddot{\beta} \\
&\propto R\, B^3\, f^2 \\
&\propto R\, (R\Lambda)^3\, f^2 \\
&\propto R^4\, f^2\, \Lambda^3 && (163)
\end{aligned}
$$

Again it is seen that all the above components scale with $R^4 f^2$, with varying factors of θ_T and Λ. The added-mass force resolved in all other directions will have terms similar to the above, and therefore scale similarly. Thus, added-mass and quasi-steady forces will scale similarly.

Since the focus here is only on the scaling, the translational moments (vertical, horizontal, parallel and normal moments) can be found quickly from the above, by noting that they are simply a radius-dependent offset times the forces above:

$$
\begin{aligned}
M_{TQ1} &\propto R^5\, f^2 \theta_T^2\, \Lambda && (164)\\
M_{TQ2} &\propto R^5\, f^2 \theta_T\, \Lambda^2 && (165)\\
M_{TQ3} &\propto R^5\, f^2\, \Lambda^3 && (166)\\
M_{TA1} &\propto R^5\, f^2 \theta_T\, \Lambda^2 && (167)\\
M_{TA2} &\propto R^5\, f^2 \theta_T\, \Lambda^2 && (168)\\
M_{TA3} &\propto R^5\, f^2\, \Lambda^3 && (169)\\
& && (170)
\end{aligned}
$$

The quasi-steady pitching moment scales as:

$$
\begin{aligned}
M_{\beta QW1} &= \pi\rho\, R\, B^2\, u_{PT}\, u_{NT}\, (1+2a) b_2 r_2 \\
&\propto R\, B^2\, u^2 \\
&\propto R\, B^2\, (R\, f\, \theta_t)^2 \\
&\propto R^3\, B^2\, f^2\, \theta_t^2 \\
&\propto R^5\, f^2\, \theta_t^2\, \Lambda^2 && (171)\\
M_{\beta QW2} &= \pi\rho\, R\, B^2\, u_{PT}\, (-\dot{\beta} B a^2 b_3 r_1) \\
&\propto R\, B^2\, (u) \dot{\beta}\, B \\
&\propto R\, B^2\, (R\, f\, \theta_T) f B \\
&\propto R^2\, B^3\, f^2\, \theta_T \\
&\propto R^5\, f^2\, \theta_T\, \Lambda^3 && (172)
\end{aligned}
$$

The pitching moments for added mass are R times the added mass pitching moments per metre span found in Section 9:

$$
M_{\beta AW1} \propto \pi\rho\, R\, b^3\, \dot{u}
$$

$$
\begin{aligned}
&\propto R\,B^3\,\dot{u} \\
&\propto R\,B^3\,Rf^2\theta_T \\
&\propto R^5\,f^2\theta_T\,\wedge^3 \qquad (173) \\
M_{\beta AW2} &\propto \pi\rho\,R\,b^4\,\ddot{\beta} \\
&\propto R\,B^4\,\ddot{\beta} \\
&\propto R^5\,f^2\,\wedge^4
\end{aligned}
$$

(174)

It can be seen that all the quasi-steady and added mass moments scale with $R^5 f^2$, with varying factors of θ_T and $\wedge$.

Some other parameters are now considered: the lift coefficient needed C_L, average induced downwash velocity u_i, the induced mass flow $\dot{m}$ and the induced power P_i:

$$
\begin{aligned}
C_L &= \frac{mg}{\frac{1}{2}\rho\,u^2\,A_W} \\
&\propto \frac{R_m^3}{u^2\,R^2\,\wedge} \\
&\propto \frac{R_m^3}{(R\,f\,\theta_T)^2\,R^2\,\wedge} \\
&\propto \frac{R_m^3}{R^4\,f^2\,\theta_T^2\,\wedge} \qquad (175) \\
u_i &\propto \sqrt{\frac{m}{\rho\,A_S}} \\
&\propto \sqrt{\frac{R_m^3}{A_S}} \\
&\propto \sqrt{\frac{R_m^3}{R^2\,\theta_T}} \\
&\propto \sqrt{\frac{R_m^3}{R^2\,\theta_T}} \qquad (176) \\
&\propto R_m^{1.5}\,R^{-1}\,\theta_T^{-\frac{1}{2}} \qquad (177) \\
\dot{m} &= u_i\rho\,(A_S) \\
&\propto R_m^{1.5}\,R^{-1}\,\theta_T^{-\frac{1}{2}}(R^2\theta_T) \\
&\propto R_m^{1.5}\,R\,\theta_T^{1.5} \qquad (178) \\
P_i &= \frac{1}{2}\rho\,\dot{m}\,(u_i^2) \\
&\propto R_m^{1.5}\,R\,\theta_T^{1.5}(R_m^{1.5}\,R^{-1}\,\theta_T^{-\frac{1}{2}})^2 \\
&\propto R_m^{4.5}\,R^{-1}\,\sqrt{\theta_T} \qquad (179) \\
\frac{P_i}{m} &= \frac{P_i}{R_m^3} \qquad \text{Specific induced power}
\end{aligned}
$$

$$\propto \; R_m^{1.5}\, R^{-1}\, \sqrt{\theta_T} \tag{180}$$

$$\frac{L}{A_W} = \frac{m}{A_W} \qquad \text{Wing loading}$$

$$\propto \; \frac{R_m^3}{R^2 \Lambda} \tag{181}$$

The merit criterion of force per unit root moment is now examined. This is a merit criterion because the lift force is what is needed to be able to stay airborne, while bending and twisting root moments are the hinge loadings to be designed against in order to obtain the force. Using the moment and force results, and ignoring the θ_T and AR terms yields:

$$M_{TQW} = R^5 f^2 \tag{182}$$
$$M_{\beta QW} = R^5 f^2 \tag{183}$$
$$F_{QW} = R^4 f^2 \tag{184}$$
$$M_{TAW} = R^5 f^2 \tag{185}$$
$$M_{\beta AW} = R^5 f^2 \tag{186}$$
$$F_{AW} = R^4 f^2 \tag{187}$$

It can be seen that the force per moment scales with $1/R$.

12.1 Summary of scaling results

The main merit criterion for this section is the power per unit lift:

$$\frac{P_i}{L} \propto \frac{R_m^{4.5}\, R^{-1}\, \sqrt{\theta_T}}{R_m^3}$$

$$\propto \; R_m^{1.5}\, R^{-1} \sqrt{\theta_T}, \tag{188}$$

where the scaling of L with respect to R, f was used, ignoring the variable factors of Λ, θ_T. If simple geometric scaling is used (where $R_m = R$), it is seen that the induced power per unit lift increases with size.

Also, the force per moment goes as $1/R$.

There are a number of other practical considerations that affect the scaling. Mostly, these will favour larger scale. For example the difficulty of manufacturing very small components and the efficiency of electric motors. Since electric motors rely on generating an electrical field, their efficiency (power output per unit input) and effectiveness (power output per unit mass) decreases with smaller size.

From this it is concluded that the lower limit of the FMAV size will be set not by the merit criteria found above (which tend to favour smaller size), but by the practical difficulties of physical implementation.

Part III
Code implementation

This part describes the code implementation of the theory of Part II, and the considerations required when adapting the theory for computational use. Note here that the code is a numerical computer implementation of an analytical theory, not a CFD model. See Section 13.2 for more on this difference.

Briefly, the functions in the code are in a hierarchy, as seen in Figure 26, and split modularly, to match the modules of the theory developed in Part II.

This part starts with a general overview of the code. Section 13.1 has a description of legacy, a commonly encountered problem in code development, and the steps taken to overcome it. Section 13.2 contrasts our code with the CFD approach, while Section 13.3 describes the hierarchy of functions in greater detail. A number of runtime parameters were also defined, to allow the working of the code to be changed without editing the code. These are described in Section 13.9.

13 Code implementation

The code was implemented in **MATLAB** because it has a great deal of inbuilt functionality for handling vectors and matrices, which made code development easier, but more importantly makes the code far more compact and legible.

The code was split into a number of functions. Because the theory devised is non-iterative, it was possible to arrange them hierarchically by type, as shown in Figure 26. Briefly, the top level, run functions, are the command used to execute the entire code. These in turn call the master functions, which calculate the results for a given part of the model, e.g. the quasi-steady forces. They do this by calling calculation functions, that deal with a specific aspect of the calculation. At the lowest level, the data functions provide all the data needed by the other functions. The flow of information in the figure is almost entirely upwards. The exception is the quasi-steady results from the quasi-steady master functions, which are used by other master functions. At no point does information flow down the hierarchy. As already stressed, the main thrust of the model was that is was non-iterative, so the flow of information is unidirectional.

13.1 Legacy

When calculating forces in the code, the fluid density ρ is needed. Imagine if every force equation had this coded in as the value for air 1.225. If at any point forces for another fluid are wanted, every single value would have to be replaced. This is obviously time-consuming, but worse yet is that a single instance might be missed, meaning the results for part of the code are based on old values. This is *Data legacy*, and can be very time-consuming to track down and repair. Similarly, there is the concept of *Method legacy*: for example if a quasi-steady force is calculated in two places within the code, and the method of calculation is changed in one place but not in the other.

Both of the above need not be the result of deliberate changes, either. For every extra time a given equation or variable has to be typed, there is a risk of mis-typing.

The way to overcome these two problems is to centralise all the data and methods in specific functions, and make sure all calls to that value or method happen through the designated function.

Hence, the functions **kine** and **geom** provide all data on the kinematics and geometry of the wing. Also, the functions **am**, **pol** and **qs** handle all calculations related to added mass, Polhamus and quasi-steady forces, respectively.

13.2 Iterative models (CFD)

Most computational fluid dynamics (CFD) codes rely on successive approximation in some way, because an analytical expression cannot be found for the answer.

These *iterative methods* have the advantage of being the only possible way of modelling most viscous effects, but the disadvantages are that they tend to take a long time to run and adapting numerical analysis to such problems has become an entire field in itself. More

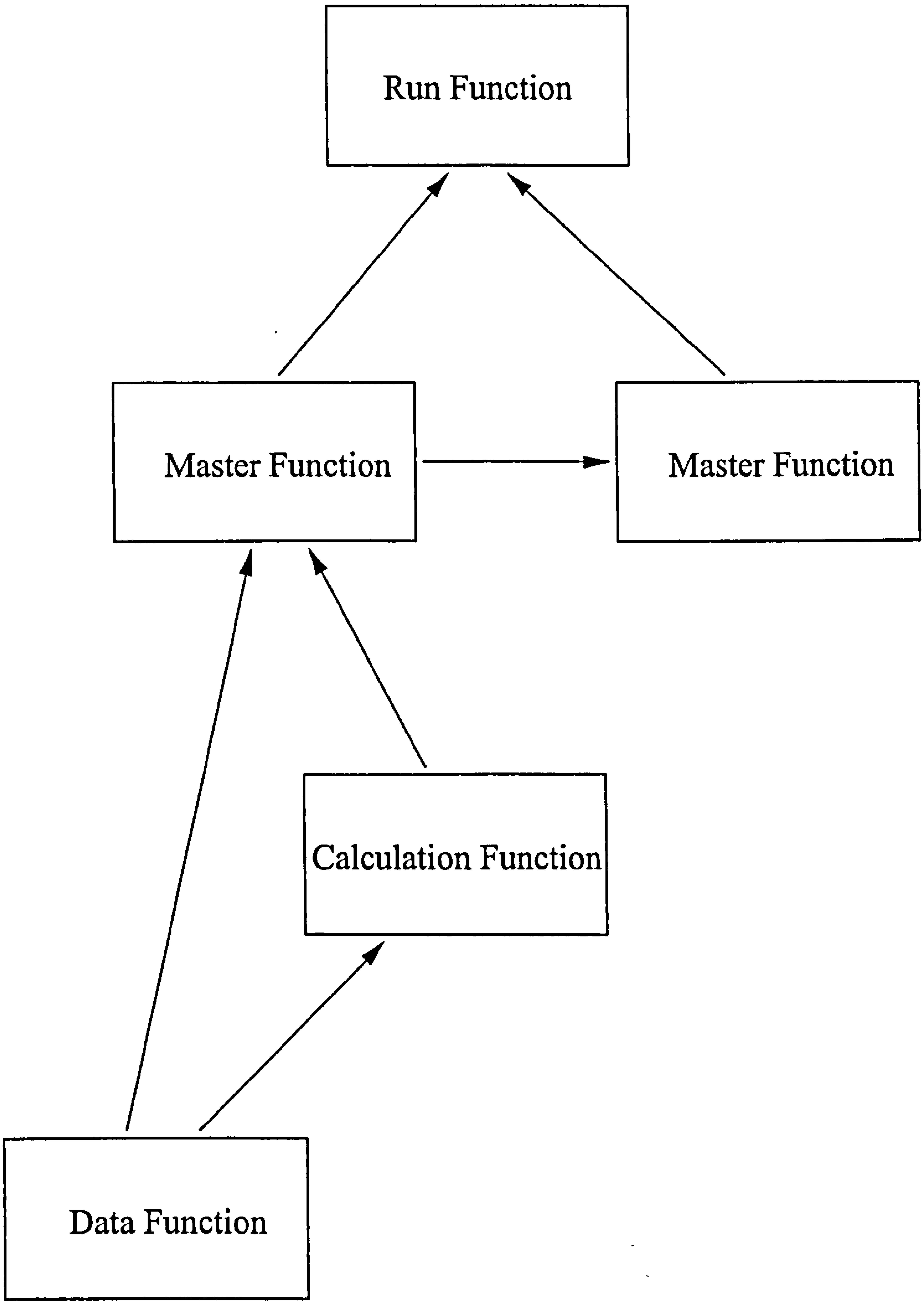


Figure 26: Code overview: The hierarchy of functions. Note that the flow of information is always upwards, or horizontal, never down.

importantly, they do not provide good insight into *why* the answer is what it is—they do not provide an overview.

For these reasons, the model developed and implemented in code was desired to be analytical. Although a closed form expression for the forces due to the wake was not obtained, at least the solution has been reduced to a non-iterative case: force components are calculated in order, and at no time need to refer to a later result to refiner an earlier one. All the simplifications made about the nature of the wake were introduced so that it would be possible to do this. Although the resulting model isn't as accurate as a fully iterative model, it gives acceptable results at a fraction of the runtime, and—above all—insight into what the contribution of the various force components is.

13.3 Types of functions

There are four levels of function:

1. *Data* functions, that are called to return any aspect of the data. For example **geom** is called to return the wing tip radius.

2. *Calculation* functions, that are called and return a single result. For example **qs** is called to return the bound vorticity of the wing.

3. *Master* functions, that perform all the calculations relevant to a single aspect of the model, by calling the relevant calculation functions. For example **master_qs** calculates all the quasi-steady forces.

4. *Run* functions, or *Global Masterfiles*. These are the functions that are called once, they then call all the master functions in order.

Note that the master functions do not store data in memory after completion: their output is saved to a path set in the run functions. This is done to reduce the memory footprint of the code.

There now follows an overview of what the various functions do. The MATLAB command *help foo* will display the correct form and order for inputs for function **foo**, and calling any of the calculation or data functions with *verb=1* will display which parameter is being returned.

13.4 Run functions

The calculation method is split into two cases: analytical and numerical data. *Analytical* data can be written as expressions, whereas *numerical* data exists purely as a set of datapoints. Obviously more accurate results are obtained using analytical data, because numerical integration is avoided. However, actual experimental data is almost always numerical.

The run functions for these two methods are **master** and **master_num**. They do no calculations of their own, but call the relevant calculation functions in the right order. The run functions create four runtime parameters, which are forwarded to the master files:

- *path* is the path that the master files store their results in. No results are saved in memory between master files.
- *verb* is short for verbosity, a numerical value that tells the called function to display extra information during runtime. *verb=1* will return information on the method being used. Higher values are used mainly for debugging, and display increasing amounts of runtime data, such as the current timestep or radial position.
- *show* is the amount of data to plot. *show=1* will plot the most important results, while higher values will cause the called function to provide more and more information. Like verb, *show>1* is used mainly for debugging.
- *skip* is the number of subroutines to skip in the master function. This is used when editing or debugging, to avoid re-running the time consuming parts of the code, but only run the parts changed.

13.5 Master functions

These calculate all results related to a single aspect of the code.

- **master_qsam** calculates all results for the quasi-steady and added mass models. The numerical equivalent is **numerical_qsam**
- **master_pol** calculates all results for the Polhamus correction. The numerical function is **numerical_polhamus**.
- **master_wag** calculates the effect of the primary wake, using the Wagner function. There is only a numerical form.
- **master_kus** calculates the effect of the secondary wakes, using the Küssner function. There is only a numerical form.

13.6 Calculation functions

The calculation functions calculate the forces and associated parameters of the forces from Sections 8 to 11. The specific mechanics of each function are detailed in Appendix 13

- **qs** calculates the properties related to quasi-steady theory, from Section 8.
- **am** calculates the properties related to added mass effect, from Section 9.
- **pol** calculates the properties related to the Polhamus leading edge suction analogy, from Section 10.
- **wagner** calculates the Wagner perturbation effect of a time series of changes of C_L, from Section 11.
- **kussner** calculates the Küssner perturbation effect of a time series of changes of C_L, from Section 11.

13.7 Data function

The data functions provide the basic data on the kinematics and geometry of the wing. These two functions are hard-coded to a given dataset. Note that the **kine** function also returns certain runtime parameters, telling the code how to deal with a given method, for example the variable *wakemethod*, which determines how secondary wakes are created. Again, this is to avoid method legacy by ensuring all functions are using the same value for *wakemethod*.

geom also returns the wing shape parameters discussed in sections 8 and 10.

13.8 Other functions

- **der(x,t)**. calculates the numerical differential of x wrt t, assuming x closes so we can form the first value of dx from the difference between the last and the first value.
- **find_crossings(x)** Finds the points where x crosses zero (i.e. changes sign), low-pass filtering the data to avoid multiple crossings in close succession for noisy data.
- **message(toc,string)** Displays a string to the run window, along with the time elapsed, *toc*
- **rotator** finds the actual location of the hinge line of the wing, based on the rotation vectors θ, ψ, and the normalised radius r.

13.9 Runtime parameters

As mentioned above, the **kine** function returns some runtime parameters, which determine the method the code uses. These are:

- *nwak* (integer, value 1 to ∞). The number of full cycles that will be used to create the secondary wakes.
- *firststep* (single letter, w,s or i). Deals with the wake effect at the first timestep, As mentioned in Section 11.
- *datalength* (single letter, f or o) tells the code whether the data represents a full cycle (that closes) or not.
- *wakemethod* (single letter, f or g) determines number of secondary wakes: whether they are fully formed at the outset, or grow over time.
- *usepolhamus* (single letter, y or n) tells the code whether to correct for the forces due to Polhamus when calculating wake vorticity
- *tailflag* (single integer, 1 or 0). Whether to calculate stroke reversal and wake location on the basis of the trailing edge location (1) or the hinge location (0).

13.10 Runtime and resource usage

Each execution of the code in the following section was completed in less than five minutes on a 1.8 GHz P4 with 512MB of memory. At this speed of execution, it was considered that any further reduction in runtime, at the expense of code legibility and ease of development was simply not worthwhile. Note that there is considerable wastage of processing time, for example in the data functions which calculate *all* the parameters they may be expected to return, then return the appropriate one. The advantage of this is that the datafunctions are clear and legible.

For the sake of being able to use the parameter *skip* in the master functions, the entire results of each master function is saved at the end of every subroutine. This obviously increases the hard drive space needed to store the results by a factor of 4-5 (the number of subroutines per master function). The total size of the datafunctions for a typical run was 40MB, so this is also not considered an issue. The data functions are overwritten with each code execution, so they do not grow in size with each run.

The memory footprint of the code is not an issue on most modern systems. For this reason, all the variables calculated within a master function were stored until the end of that master function, even after they were no longer needed. Considerable reduction in memory footprint can be obtained by clearing unneeded variables at the end of every subroutine, for systems where memory becomes an issue. The drawback to this (and the reason it wasn't done) is that having all workings available is very useful for debugging and detailed examination of the results.

No data is stored in memory between master functions. Instead, all the results of a master function are stored to the hard drive, then removed from memory. Calls to an earlier masterfile are actually performed by reading these results from the hard drive.

The code was developed in **MATLAB** 6.0, and has been tested on **MATLAB** 5.2 and 5.1. Earlier versions of **MATLAB** have not been tested.

The code is intended for legibility and further development, not for minimal runtime or resource usage, since these are acceptably low.

Part IV
Results

In this part, the results of running the developed code are examined. This is done on two datasets. The first was kindly provided by Dickinson and Dickson, from their Robofly project. This was for a scaled-up version of a fruit fly wing, flapping slowly in mineral oil. The geometry and kinematics of this wing are described in Section 14, and their measured results are compared with our analytical prediction in Section 15.

The second dataset, in Section 16, is for a theoretical FMAV design, the FMAV-50/2. This design has an overall bodyweight of $50g$, and a wingspan in the order of $350mm$, flapping at $20Hz$. Since this is a theoretical design, there are no experimental data with which to compare the results of Section 17. It has been included for the sake of highlighting some effects of our model that are not apparent from the Robofly dataset. Only the results that are of special interest will be shown for this dataset.

The discussion of results in Section 18 is in two parts: Section 18.1 discusses the physical implications of the results, while Section 18.2 focusses mainly on the comparison between our predicted results for the Robofly, and the actual measured values.

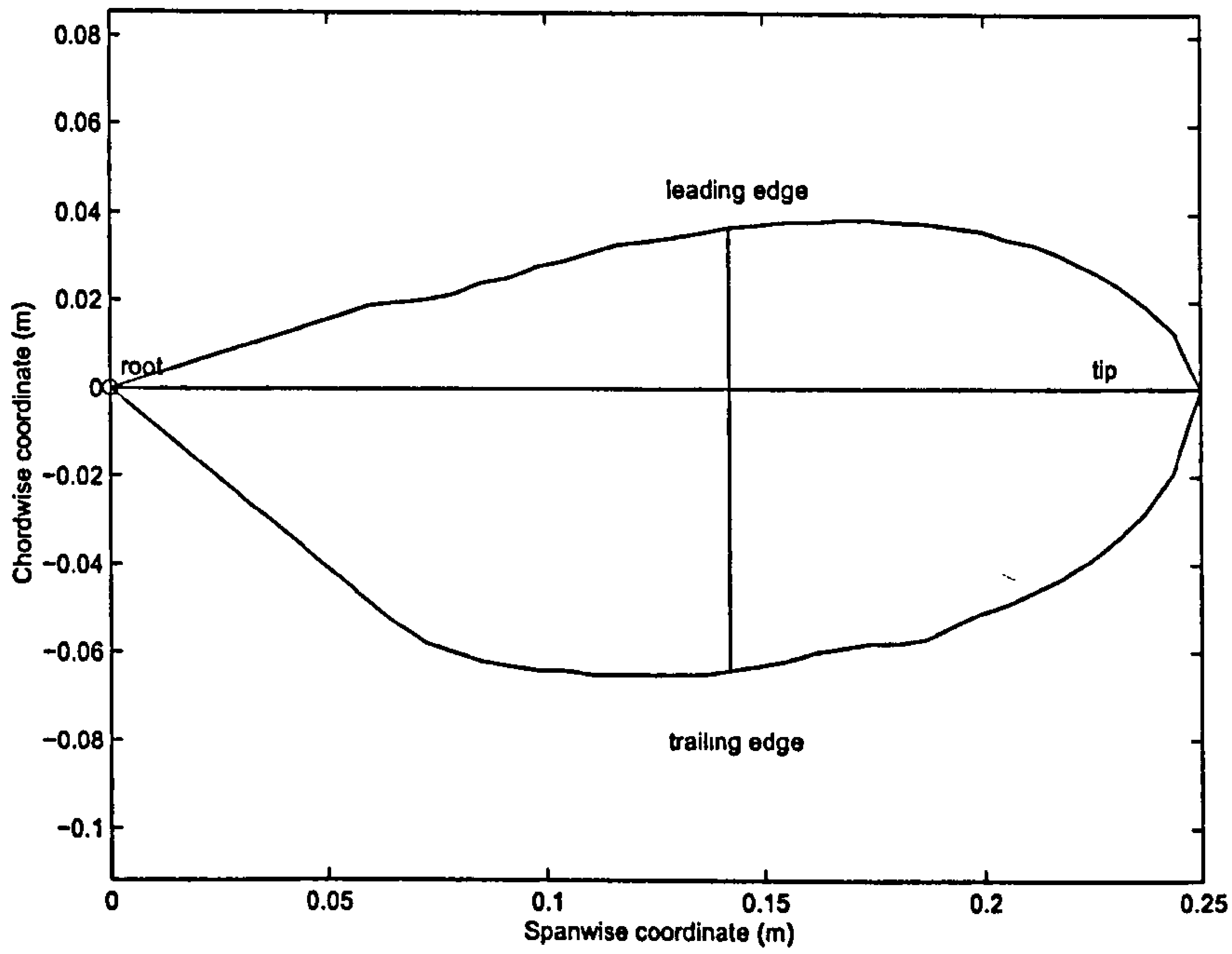


Figure 27: Robofly wing geometry. The horizontal and vertical lines are the hinge line (from the root to the point furthest from the root), and the maximum chord line (the longest line normal to the hinge line), respectively.

14 Dataset: Robofly

The Dickinson Robofly is a mechanical device that mimics the kinematics of a hovering insect, by controlling the movement of a wing at the root with electric motors. The wing is a scaled-up version of a fruit fly wing, which is flapped in mineral oil at low frequency, to preserve dynamic similarity with the original fruit fly. The frequency was $0.168Hz$, giving a period of about $6s$ for a full cycle. The equipment and procedures are explained in Sane & Dickinson [46], [2] and Dickinson [48]. The data provided were for the "advanced" case of [46], where the wing rotation leads the wing reversal.

14.1 Geometry

The wing geometry of Dickinson's Robofly is a scaled version of a *Drosophila Melanogaster* fruit fly wing. The tip radius is 250mm, but the inner 60mm of the wing is taken up with sensors, and is assumed not to contribute to the force. The shape is shown in Figure 27, where for the purpose of the plot the inner 60mm of the wing has been shown with straight trailing and leading edges.

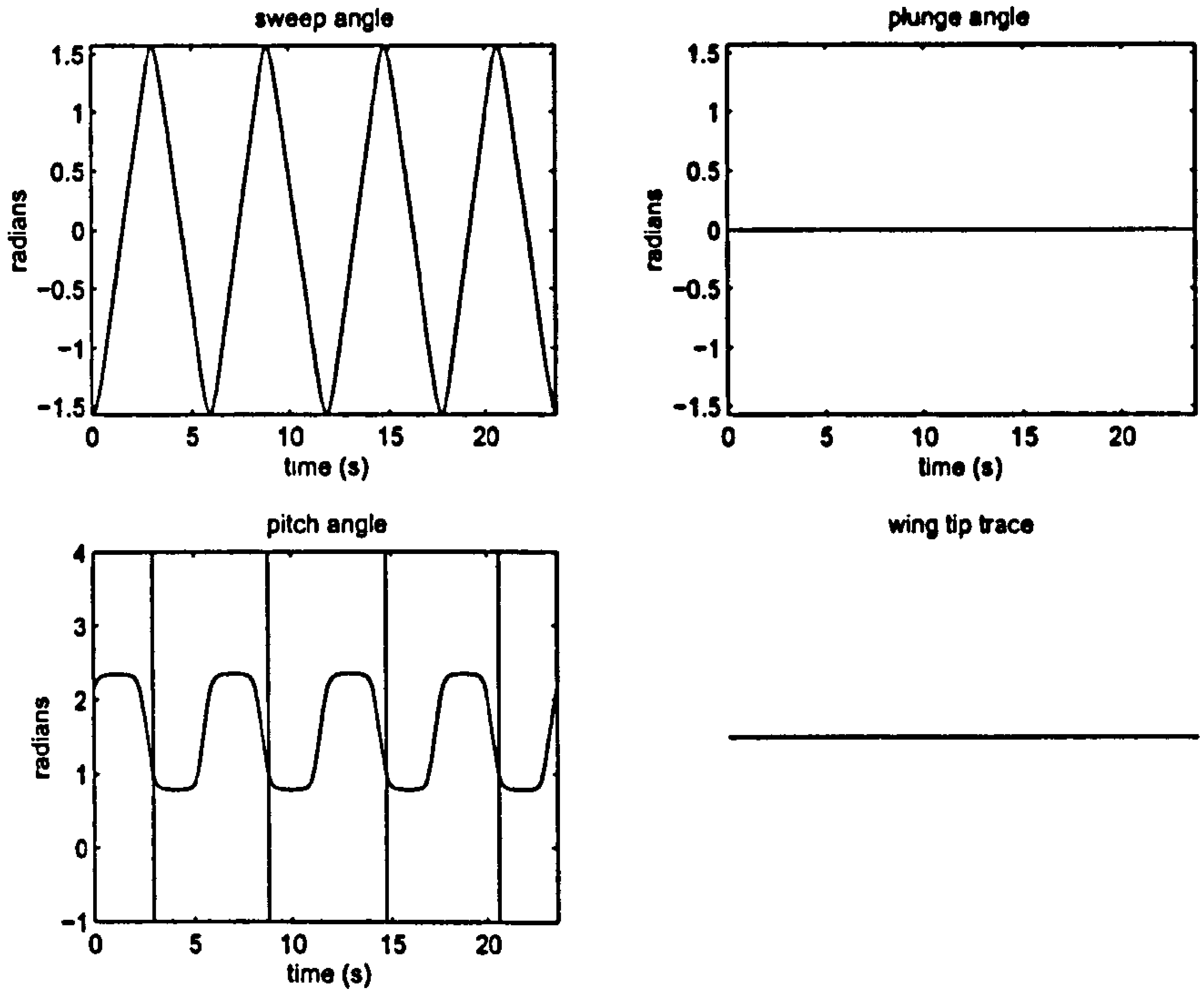


Figure 28: Robofly wing kinematics: the first stroke starts forwards (negative sweep angle θ) with the wing past the vertical. Angles are given in radians.

14.2 Kinematics

The Dickinson kinematics follow a simplified pattern, and do not mimic those of any particular insect. The sweeping motion (change of θ) is approximately a triangular wave, with near-constant sweeping velocity during midstroke. The sweeping amplitude is 80°, so the wing completes almost half a revolution each stroke. The pitching is approximately a square wave, with very sharp rotation - note that this is Dickinson's data for advanced rotation, so the rotation occurs before the hinge point comes to a stop at the end of a stroke. The plunge (change of ψ) is everywhere 0 - the tip trace and stroke plane are therefore the same horizontal line. See Figure 28, 29 and 30. The frequency is $\approx 1/6\ Hz$. There is no reduced frequency number, because this parameter is meaningless for flapping flight - see Section 6.2.3 for a discussion of this. An example of the shape of wake we can expect from these data is shown in Figure 31.

14.3 Code parameters

- RADIAL POINTS: 32, inner point is at $r = 0$, second is at $60mm$ (start of the wing proper), outer point is at $r = 1$, with remaining points evenly distributed between the start of the wing proper and the tip.
- TIME POINTS: 2356 evenly distributed across four cycles.

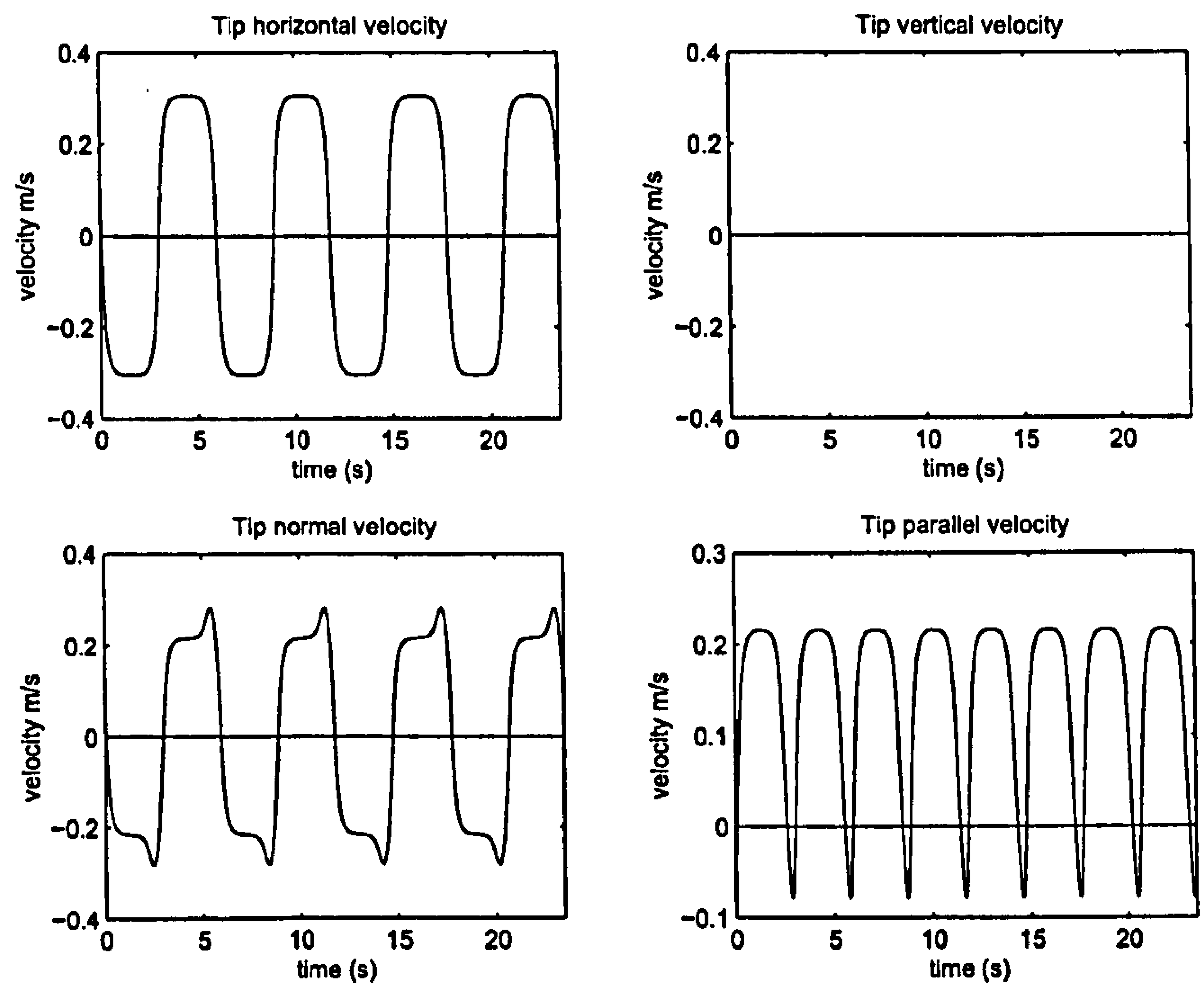


Figure 29: Robofly wingtip velocities: the velocities are of the fluid relative to the wing.

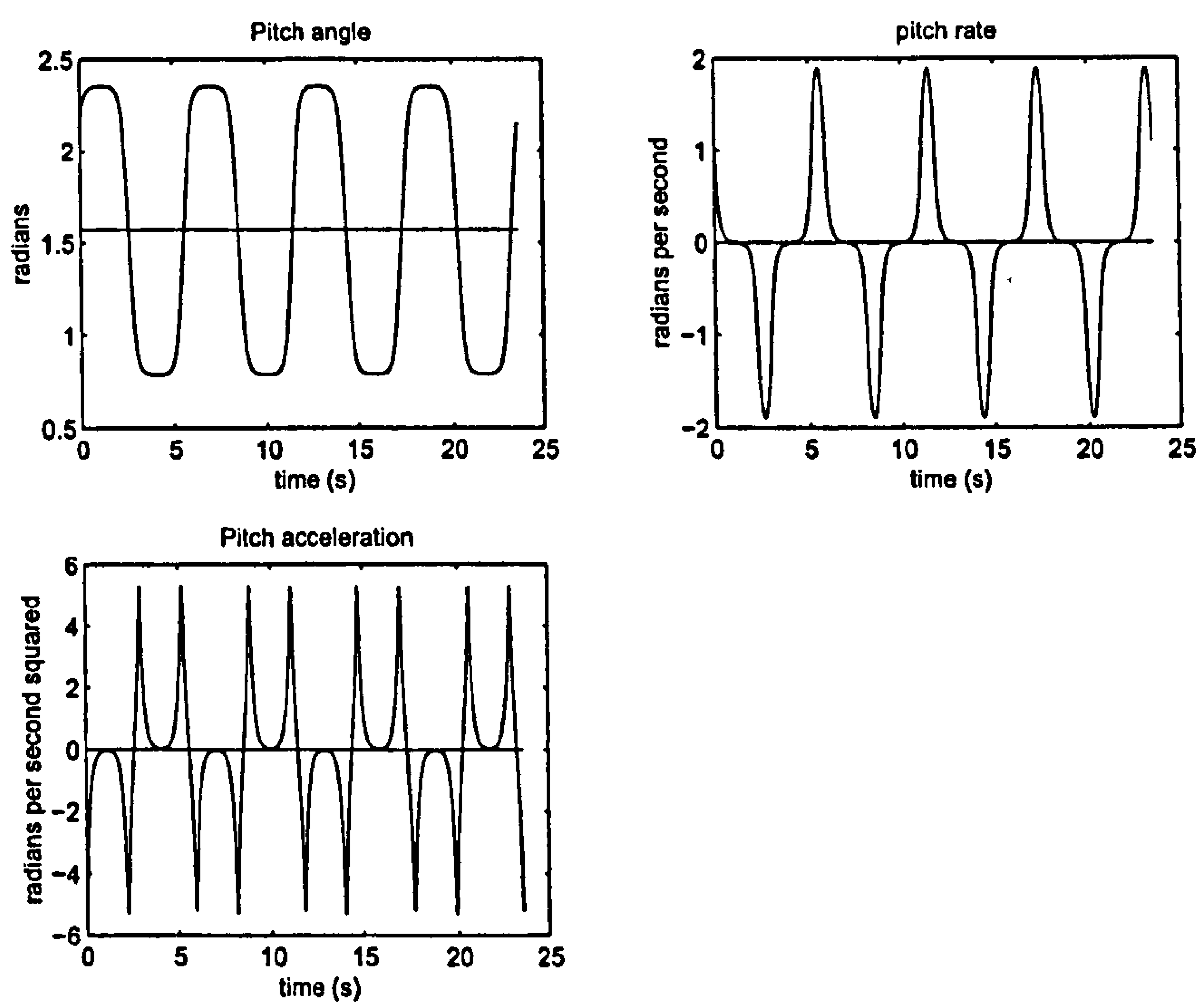


Figure 30: Robofly wing pitching: note that noise in the data was filtered to remove "spikes".

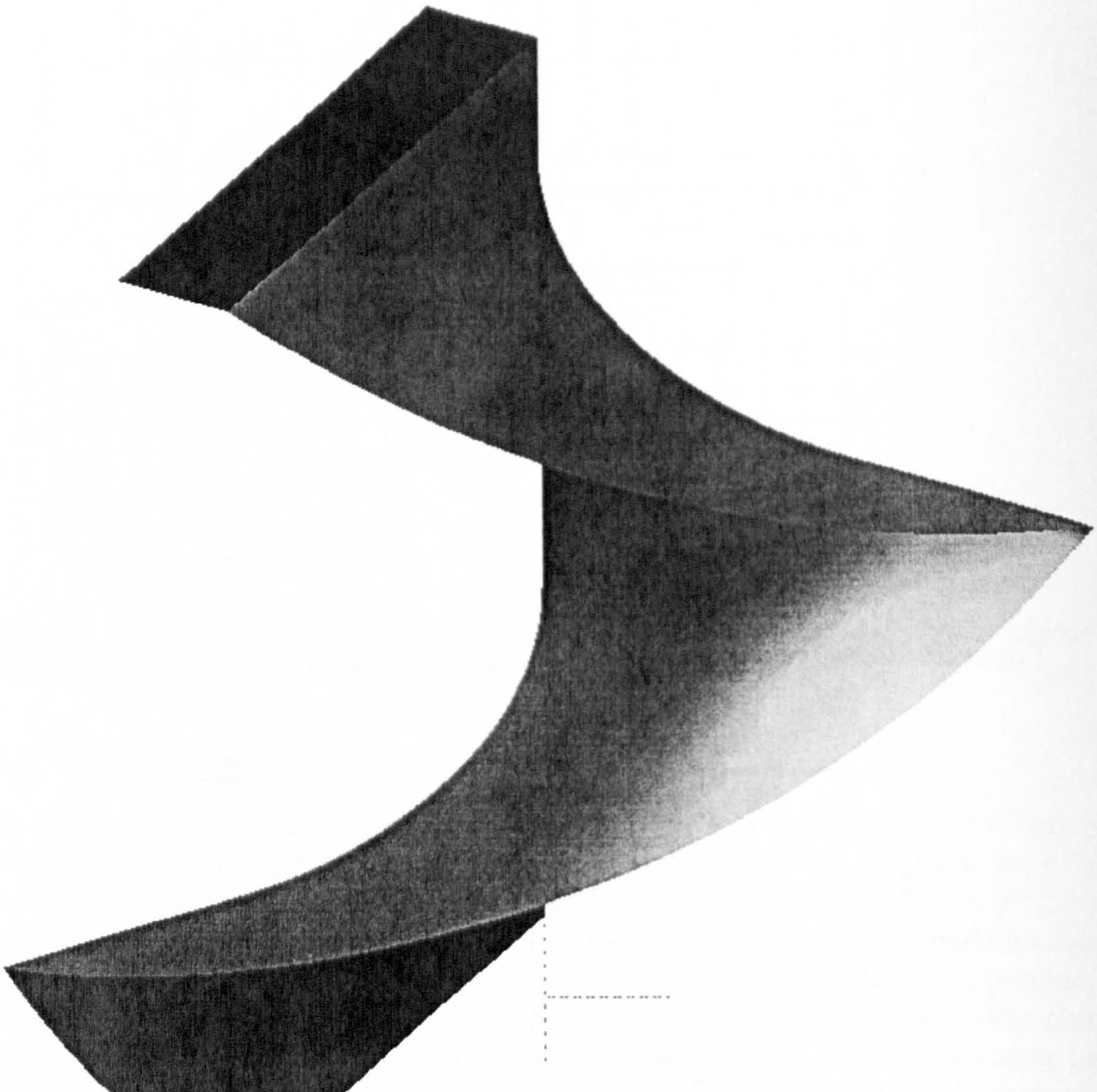


Figure 31: Sample $3D$ wake surface. The dark rectangle at the top is a thin, flat rectangular wing, undergoing horizontal sweeping with sharp reversal. The hinge of the wing is at the upper right corner in the figure (in line with the vertical dotted line at the bottom of the figure.) The $3D$ wake surface in the figure is the surface swept by the trailing edge, convected downwards at constant velocity.

- ROTATION: 90°. Wing is at 45° pitch during translation.
- PERIOD $\approx 6s$
- RHO = 870 kg/m^3. This is the fluid density (mineral oil).
- DATALENGTH = other. The data do not represent a single closed cycle.
- FIRSTSTEP = impulse. The first step is an impulsive start, with a strong starting vortex.
- NWAK = 1. The wake is using just the cycles of the data.
- WAKEMETHOD = grow. The secondary wake is grown, meaning there is no secondary wake during the first stroke, one secondary wake during the second stroke, and so on.
- USEPOLHAMUS = yes. C_L for the Wagner and Küssner effects are the effective C_L, as modified by the Polhamus correction.
- TAILFLAG = 1. Reversal times are based on the trailing edge position.

15 Results for Robofly

15.1 The "lift" force, F_V

Referring to Figure 32 it can be seen that the data comprises of eight strokes, in four full cycles. The results are not exactly equal from one stroke to the next, most noticeably the first stroke has a very suppressed initial peak compared with the rest. This fits with the Wagner effect. The average lift (the chain line) is 0.40 N. These experimental data will be reproduced on the following plots as a dotted line, for comparison.

Figures 48 and 49 show the predicted quasi-steady lift per metre span, for the first cycle only. Since translational velocities increase towards the tip, the force increases further from the root, up to the point where the wing semichord b starts to taper to a point. The radial position with the greatest forces is slightly outboard of the radius where the semichord is greatest. This holds generally for the forces due to other effects. The surface and contour plots for the other effects have been included, but will not be discussed here.

Referring to Figure 33, which shows the predicted quasi-steady lift on the entire wing versus the measured lift, it can be seen that lift is almost constant during the translation at the middle of each stroke, followed by a very sharp peak and trough at the rotation. These peaks are almost entirely due to suction forces, and are much lower in the measured data. Also, the quasi-steady component alone over-predicts lift by approximately a factor of 2. Referring to Figures 34 and 35, it is seen that the suction peak is effectively cancelled by the Polhamus effect. This is because the suction peak coincides with the wing being almost vertical, at which point rotating the suction force by 90° turns it into a horizontal force. This fits the measured data better. Polhamus has very little effect on the lift during translation. The wing is at 45°, so when the leading edge suction force is rotated by 90°, its vertical component is almost the same. The fact that it changes at all is because of the scaling due to leading edge sweep. The overall shape of the lift based on quasi-steady and Polhamus only is similar to that measured, but over-predicts lift by almost a factor of 2, and is missing some salient features of the shape.

Adding the primary wake correction for lift, in Figures 36 and 37 reduces the lift considerably, and illustrates the Wagner effect as opposing an increase in lift. Note that the lift is now increasing during the translation, matching the observation. The secondary wake lift correction in Figures 38 and 39 acts mainly to reduce the lift at midstroke. It is 0 during the first stroke, because there is no secondary wake until the first reversal. After that, it starts out strongly asymmetric due to the strong starting vortex, and unbalanced secondary wakes, but tends to symmetric between strokes as time progresses, as the starting vortex is further from the wing, and the secondary wake tends to a long series of asymmetric wakes. Considering the plot of wake vorticity for a sample radial position in Figures 46 and 47, it can be seen that the wake vorticity tends to cluster at the rotation, and the end of every stroke. The exception is the bump which is due to the rotation leading the reversal. The net effect of this is similar to the first-order model of the wake, as a pair of strong vortices inducing a downward velocity, or as a vortex ring, similar to the pulsed actuator disc model of Ellington. These have already been discussed in Section 5. The effect of this vortex pair will be most noticeable at midstroke, because this is where the offset vector is both short and horizontal, and therefore causes the greatest vertical velocity component.

However, one salient feature of the lift trace is not picked up - the upward "bump" at the start and end of each translation phase. This is a compound effect of a) ignoring added mass and b) the simplified secondary wake. Because the secondary wake has discrete "jumps"at the end of every stroke, the effect is to under-predict the secondary wake effect at the start of every stroke.

More importantly, including the added mass results into the above results (see Figures 40 and 41) highlights a major limitation of the model. With no wake correction of added mass, the Kutta-Joukowski term of the added mass equation becomes unrealistically large. Although the added mass does have the missing bumps, they are both far too large, and occur too soon to match those of the measured data. This is a direct effect of omitting the attenuating effect of the wake on the added mass, the effect of which would be to reduce, delay and smooth the added mass force. Remember that the added mass effect is the sum of two components: the irrotational Dirichlet solution, and the Kutta-Joukowski condition, see Equations (87) and (90). If only the Dirichlet component of the added mass is included (Figures 42 and 43) it is considerably better behaved, because it is not associated with vortex shedding, being irrotational. However, this, too, should be attenuated by the wake effect, even if it does not contribute to the vorticity of the wake. Compare the results of the total lift with full added mass effect in Figure 41 with the total lift where only half of the Kutta-Joukowski added mass is used, in Figure 45. This is a considerable improvement. There is no particular theoretical justification for choosing the factor $\frac{1}{2}$, except that the Wagner effect predicts the loss of half the circulatory quasi-steady lift, so it seemed a valid guess for the loss of circulatory added mass lift, too. A similar approach has been suggested by DeLaurier [49]. Note how well the scaled added mass figure matches the measured result, picking up all the critical features of the force trace, although they manifest a little too soon, and with too much magnitude. This is especially true for the loss of lift at rotation, which is being heavily overpredicted. Again, this is because the rotation is associated with strong vortex shedding, the effect of which on the added mass are not modelled. This underlines the conclusion that the model captures the unsteady aerodynamics rather well, but lacks a critical component in the modelling of the wake effect on the added mass.

Figure 44 shows the result of Figure 43, without correcting the force coefficients for the effect of Polhamus i.e. setting the variable *usepolhamus='no'* in the code. It shows that without the Polhamus correction to lift coefficients, the model heavily under-predicts F_V during rotation, because it is compensating for suction lift that is not being realised.

Added mass does not contribute a net force over a closed cycle—so although the model without added mass misses the "bumps" mentioned above, it does at least model the general shape of the lift trace, and will not affect the lift force. The average measured lift is $0.40N$, and the average predicted lift from the model is $0.37N$, only a 9% error.

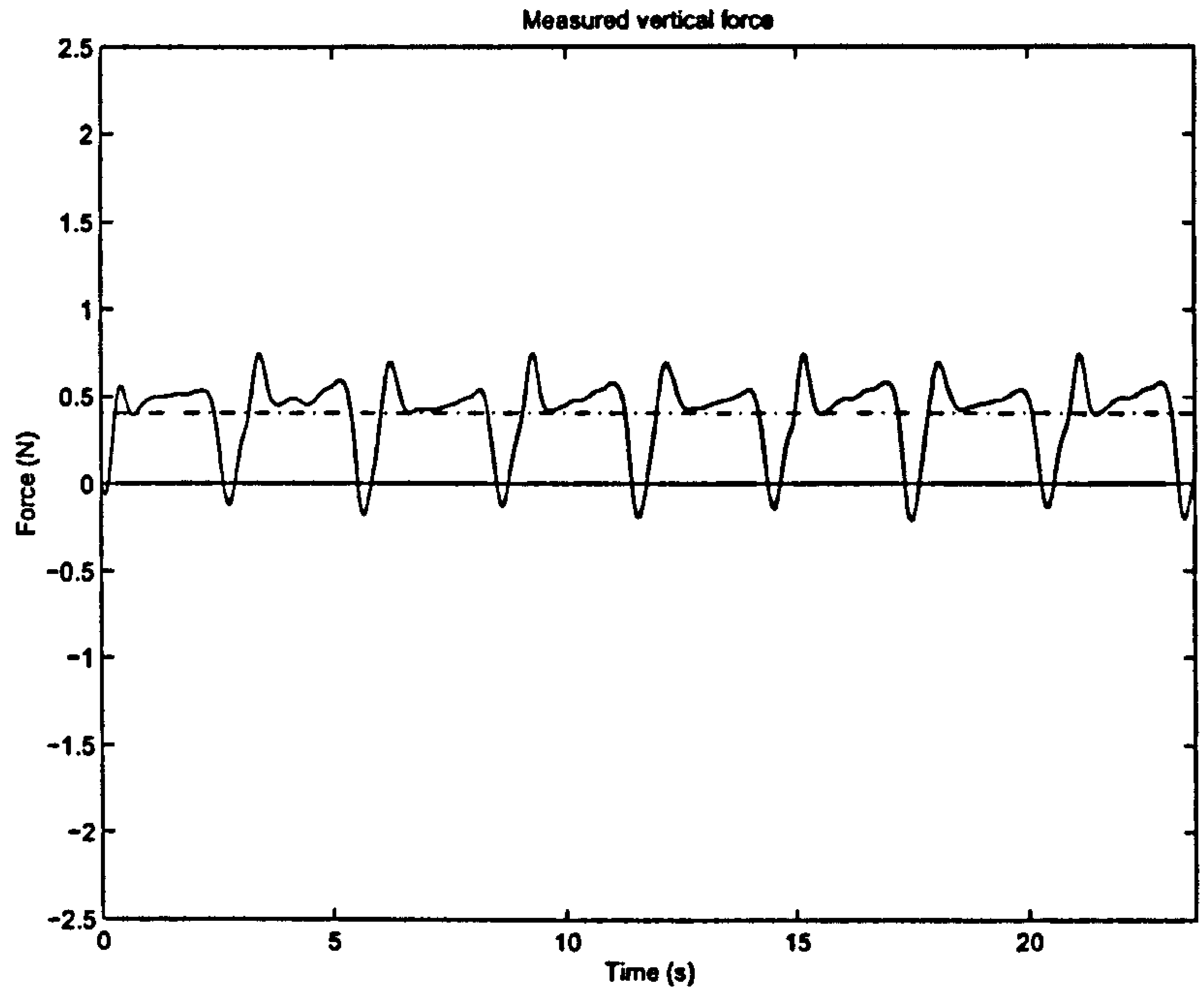


Figure 32: Measured lift force from Dickinson Robofly experiment (solid line). Chain line represents average predicted lift force.

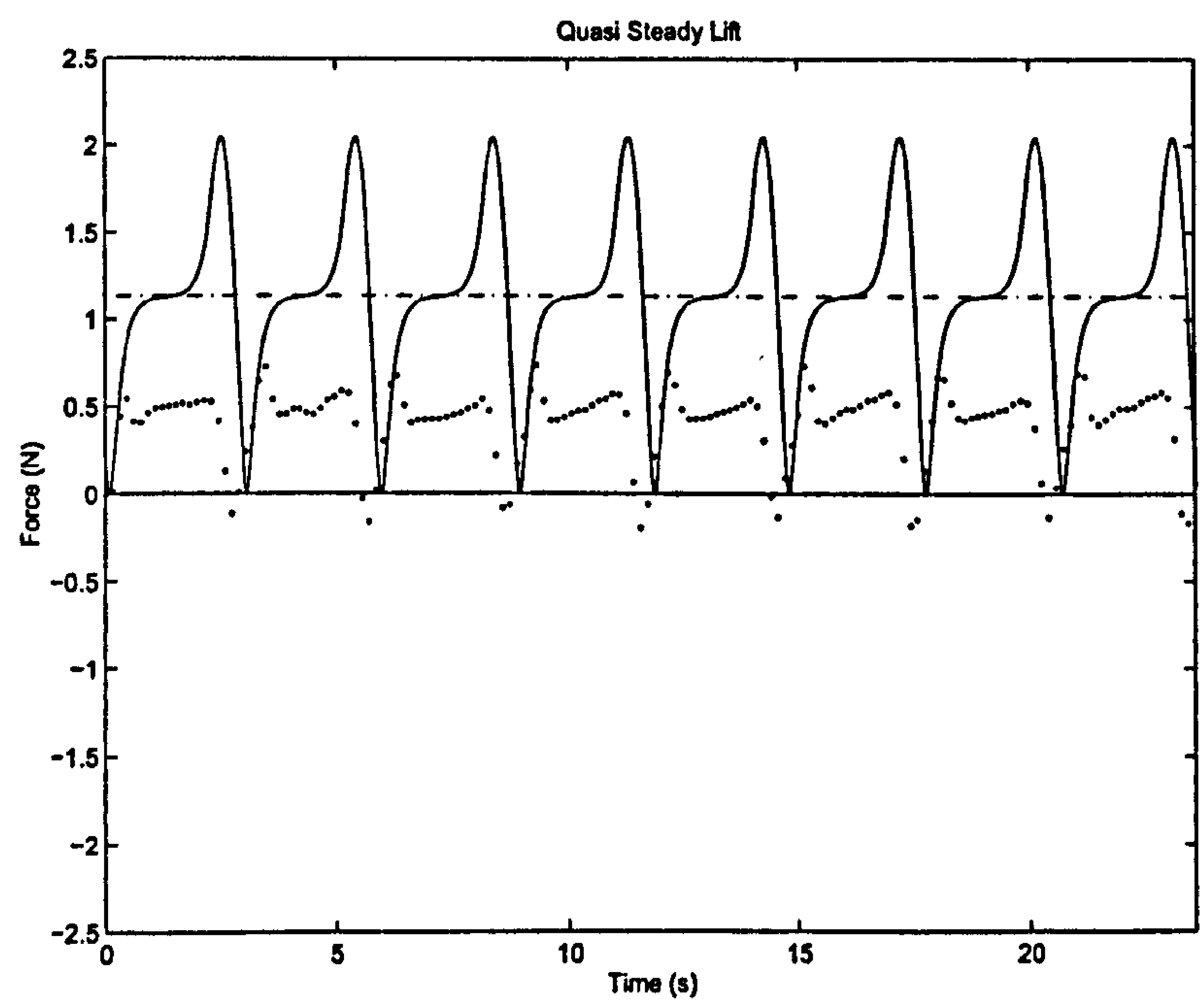


Figure 33: Predicted quasi-steady lift for the Robofly data (solid line) versus measured lift (dotted line). Chain line is average predicted force.

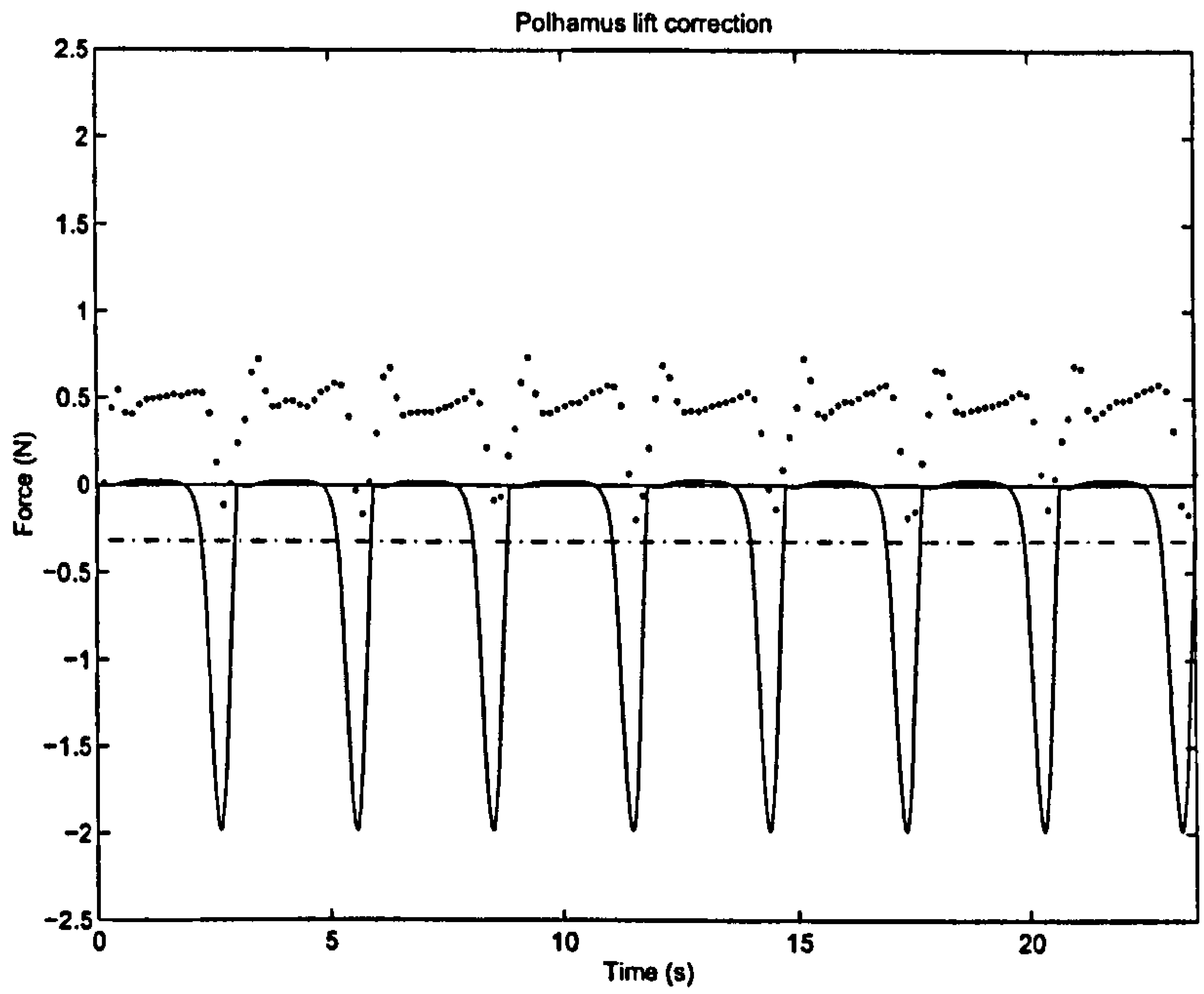


Figure 34: Polhamus correction to lift for the Robofly dataset (solid) versus measured lift (dotted). Chain line represents average predicted force.

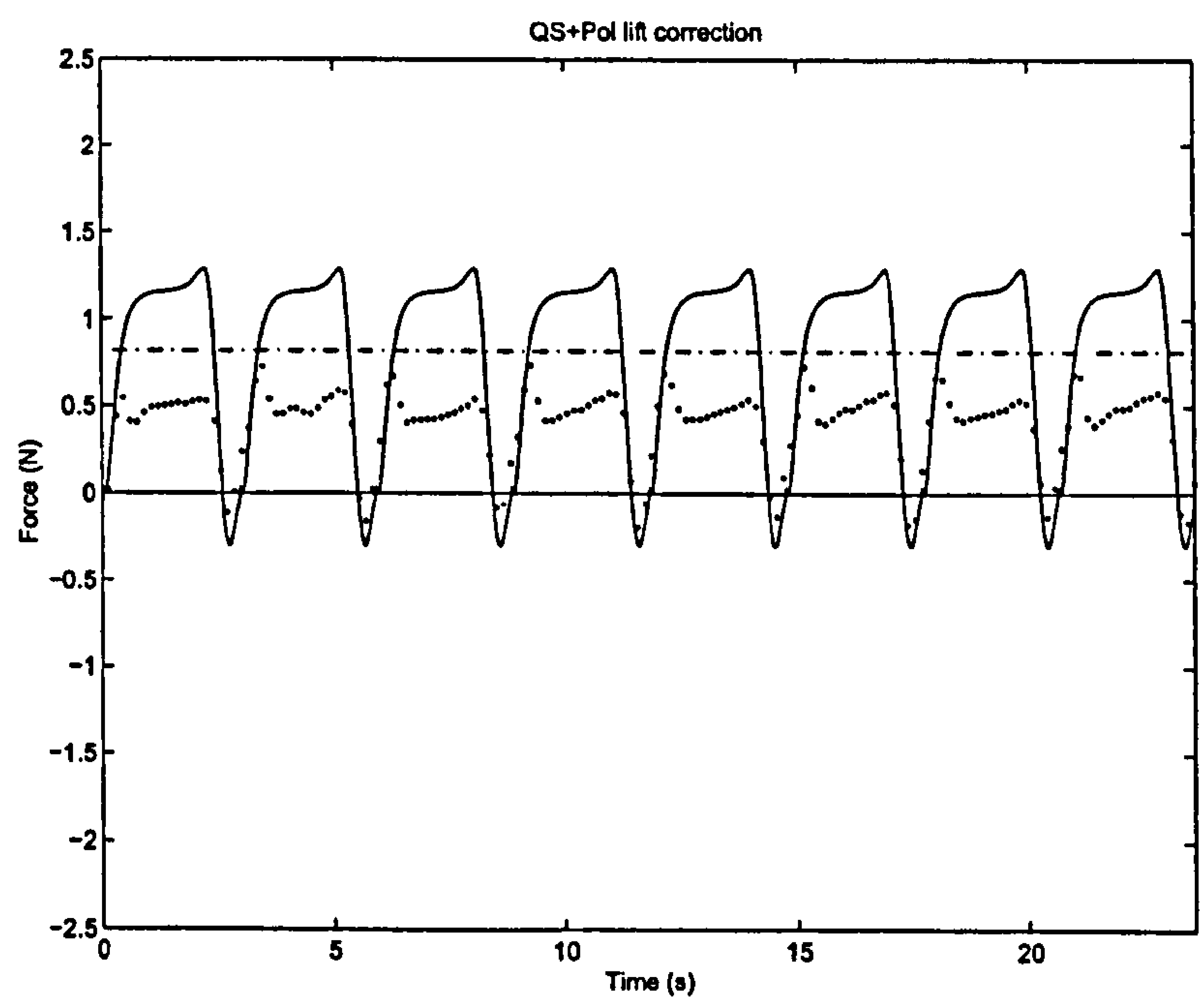


Figure 35: Quasi-steady lift, modified by Polhamus correction, for Robofly data (solid), versus measured lift (dotted). Chain line represents average predicted force.

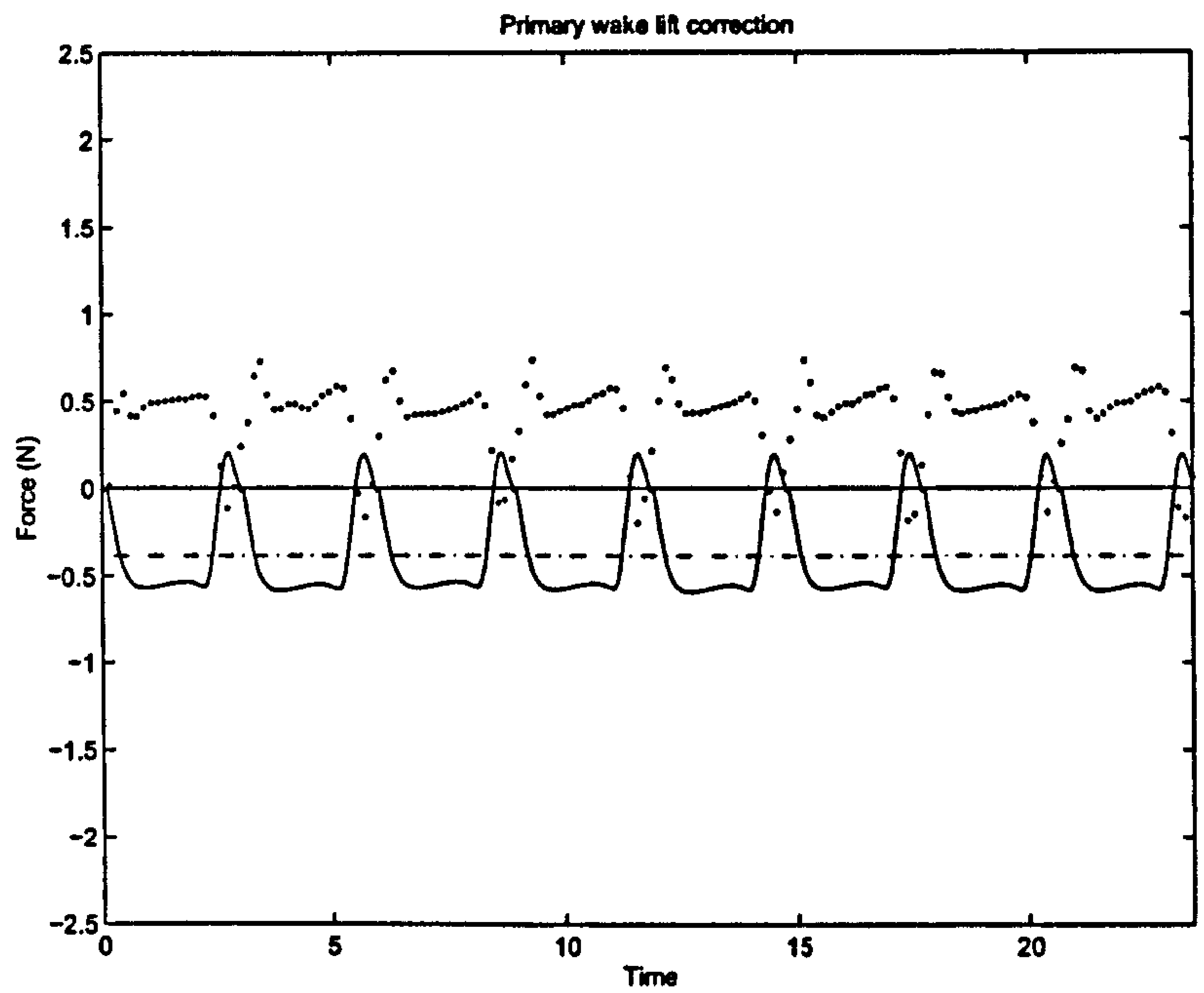


Figure 36: Wagner correction to lift for the Robofly dataset (solid line) versus measured lift (dotted line). Chain line represents average predicted force.

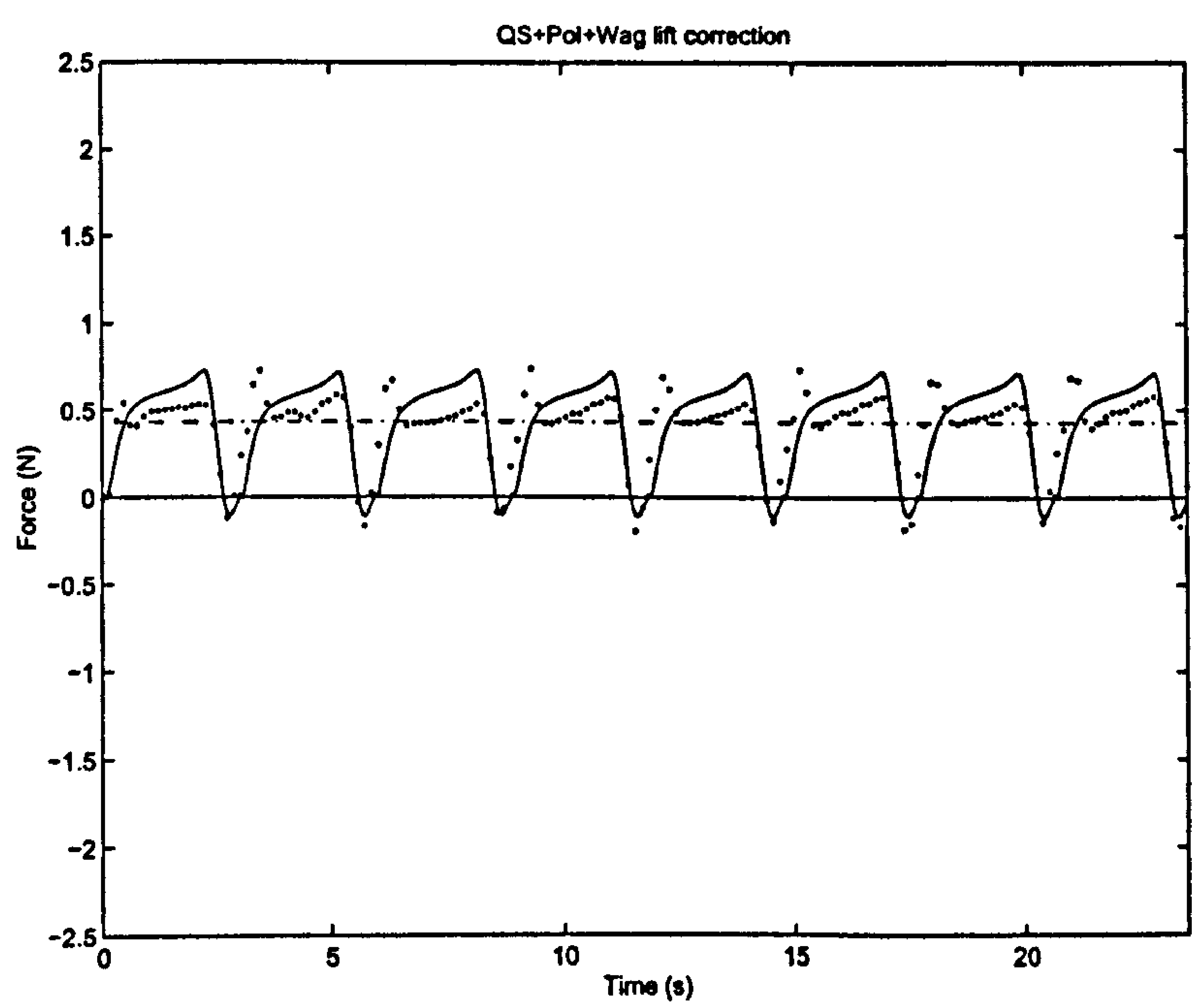


Figure 37: Quasi-steady lift, modified by Polhamus and primary wake (Wagner) correction, for Robofly data (solid line) versus measured lift (dotted line). Chain line represents average predicted force.

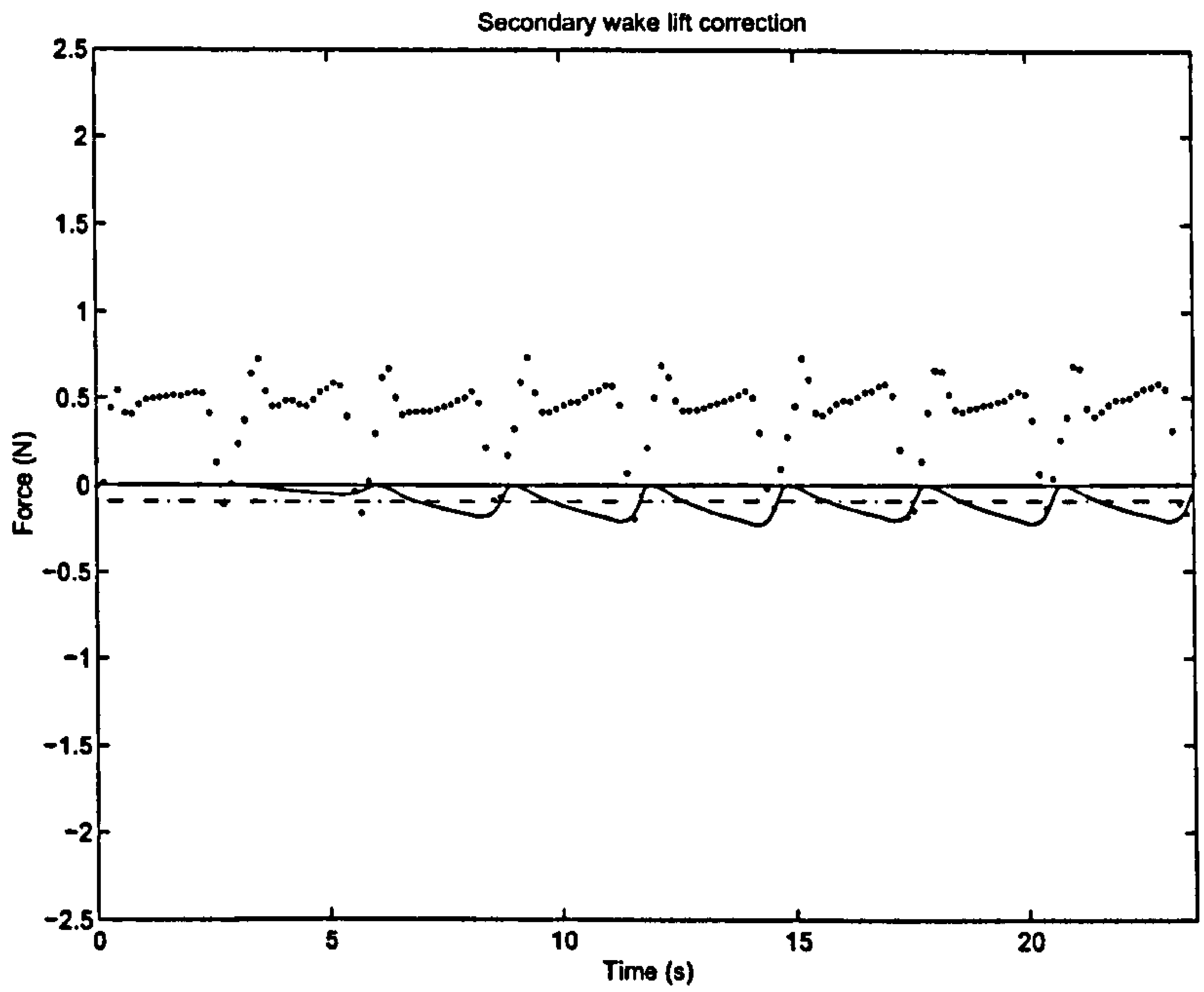


Figure 38: Secondary wake (Küssner) correction to lift for Robofly (solid line) versus measured lift (dotted line). Chain line represents average predicted force.

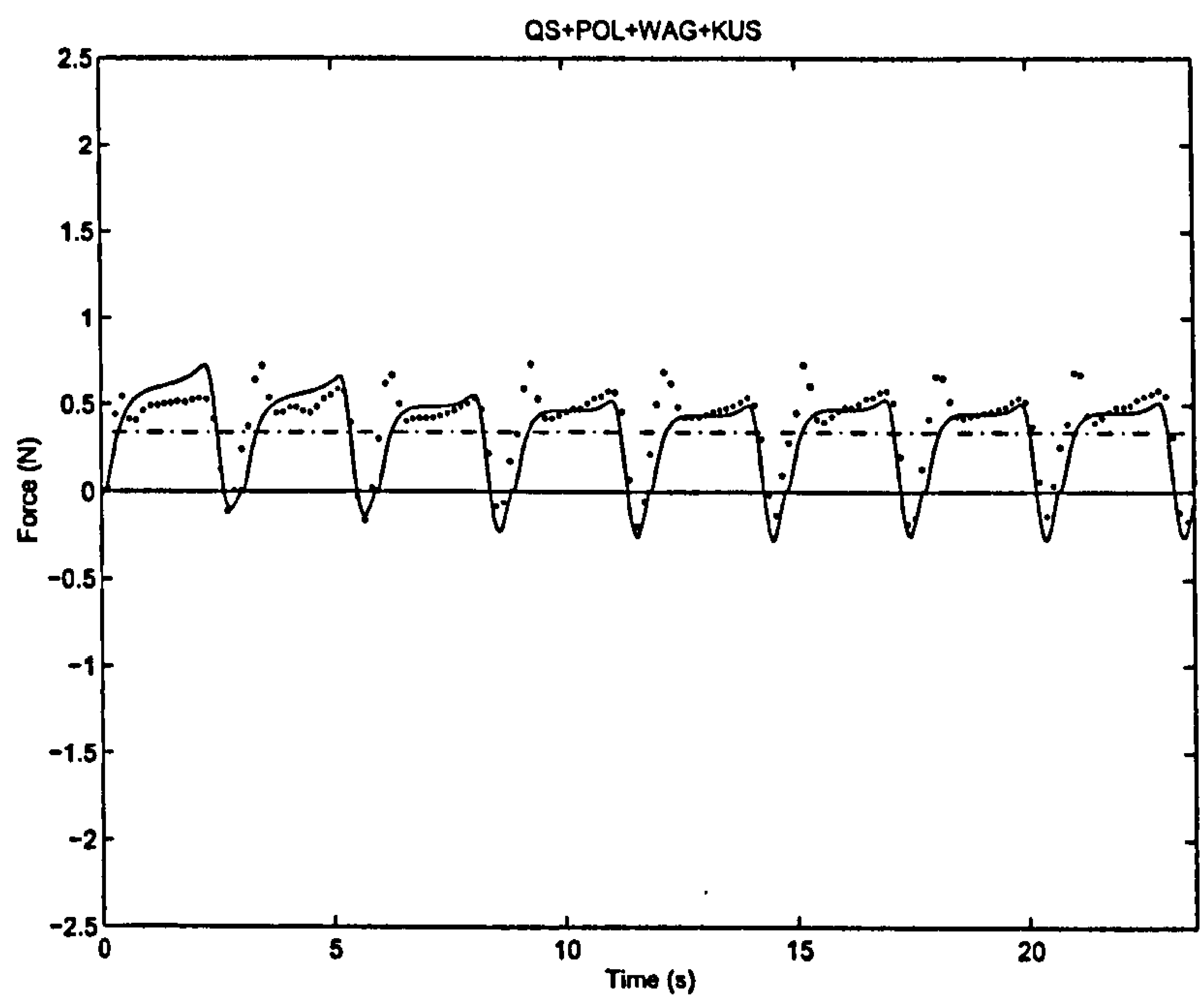


Figure 39: Quasi-steady + Polhamus + primary and secondary wake corrections to lift, for Robofly data (solid line) versus measured lift (dotted line). Chain line represents average predicted force.

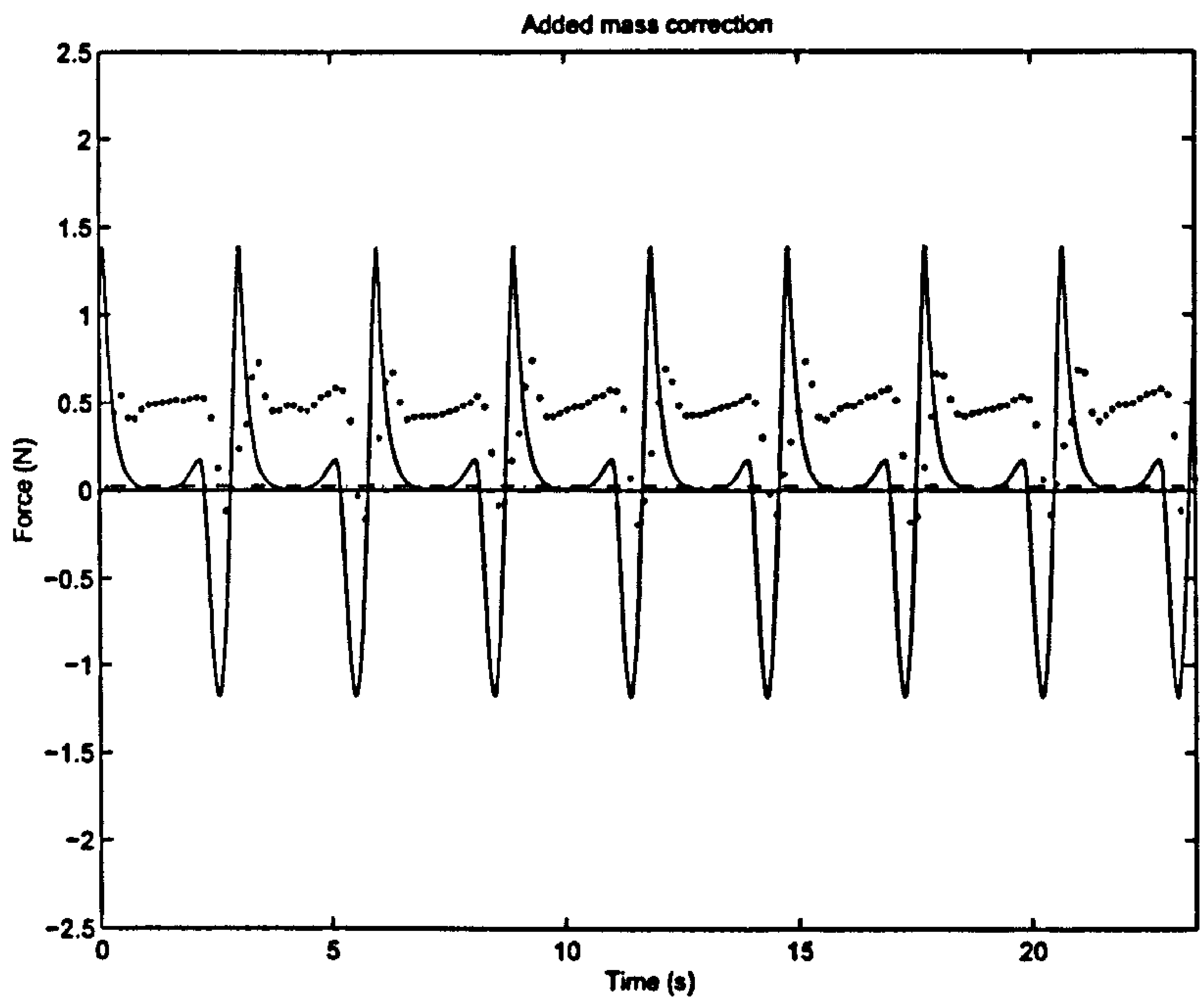


Figure 40: Added mass correction to lift for the Robofly dataset (solid line) versus measured lift (dotted line). Chain line represents average predicted force.

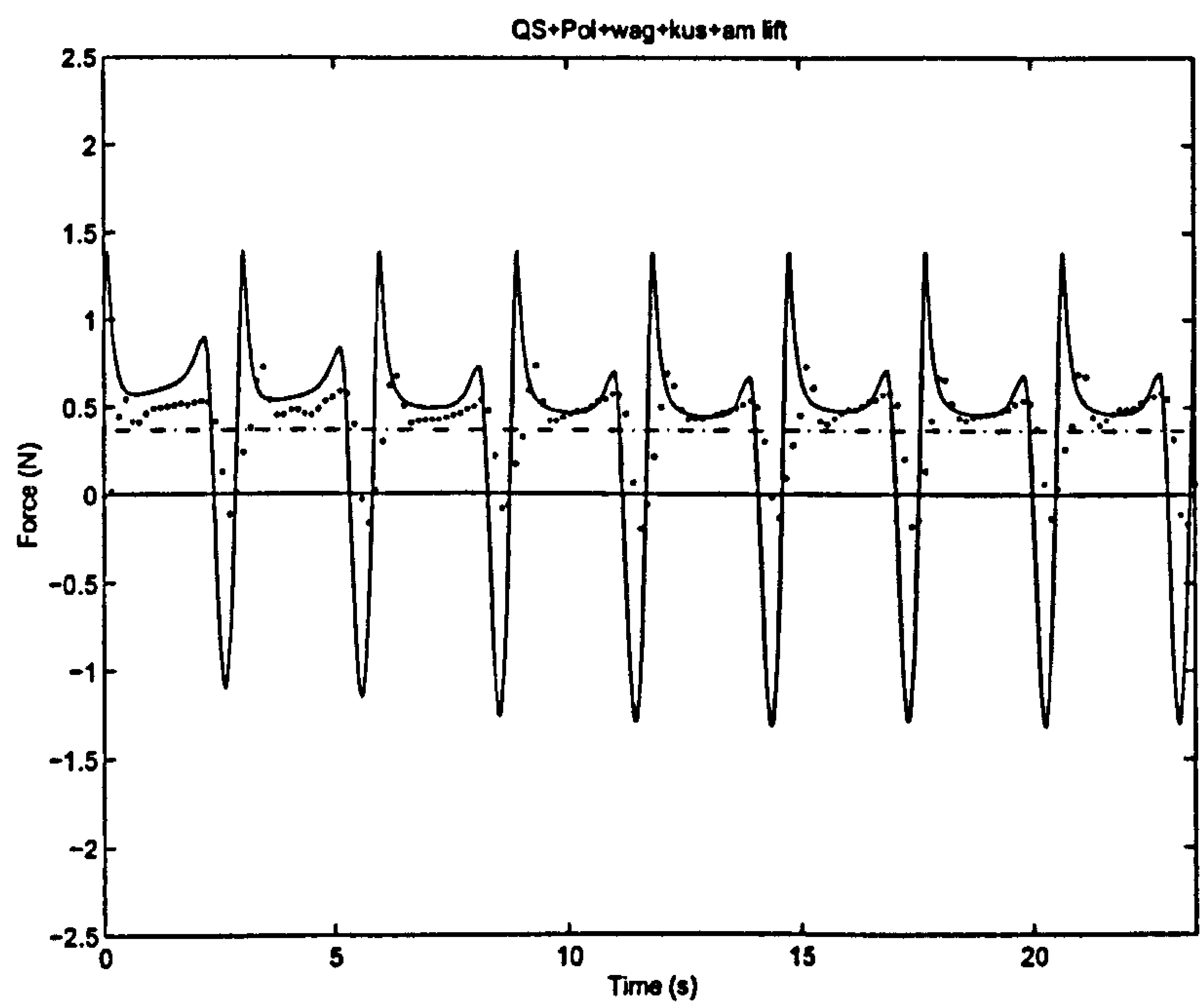


Figure 41: Total lift, including added mass forces, for Robofly dataset (solid line) versus measured lift (dotted line). Chain line represents average predicted force.

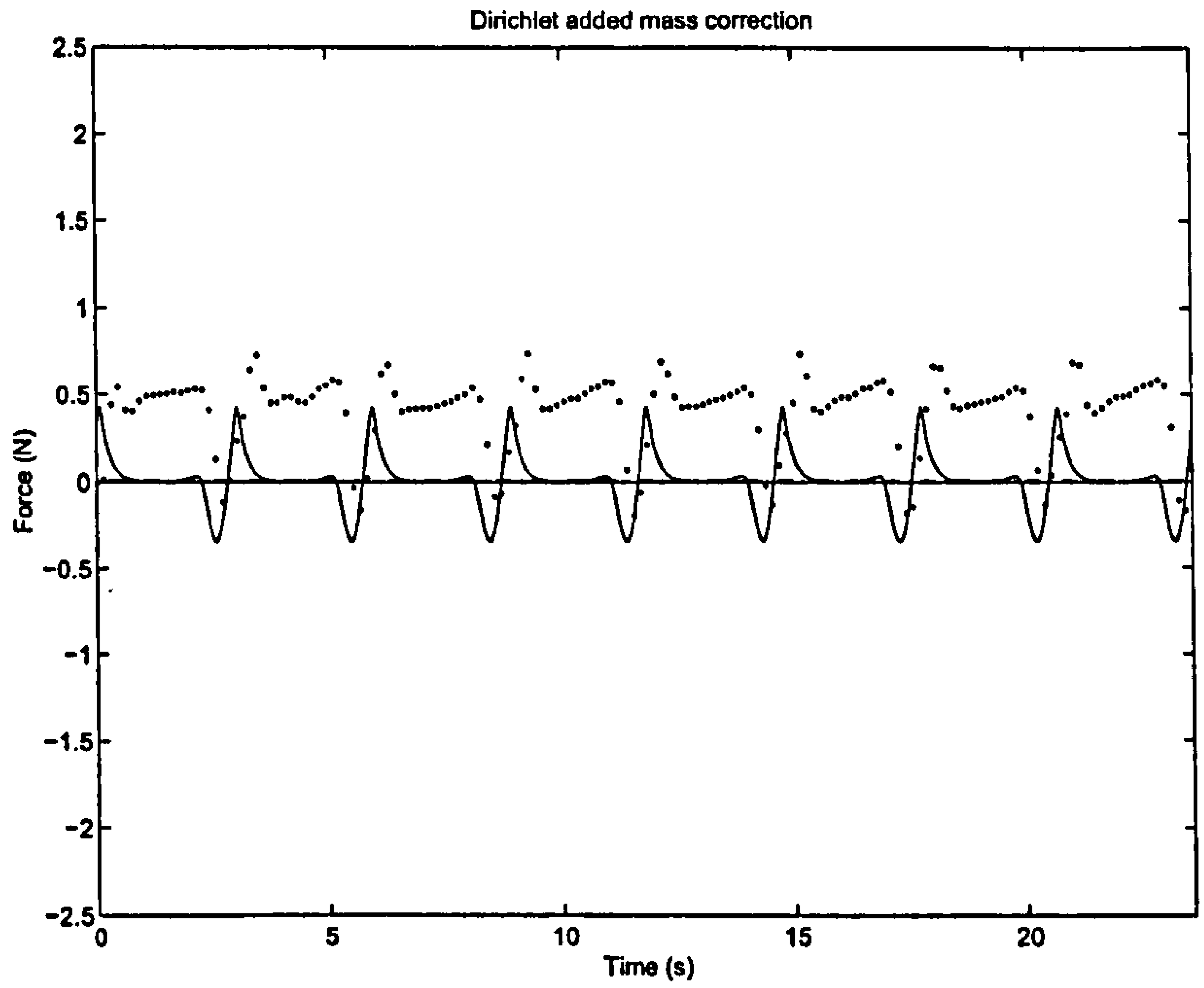


Figure 42: Dirichlet component of the added mass correction to lift for Robofly dataset (solid line) versus measured lift (dotted line). Chain line represents average predicted force.

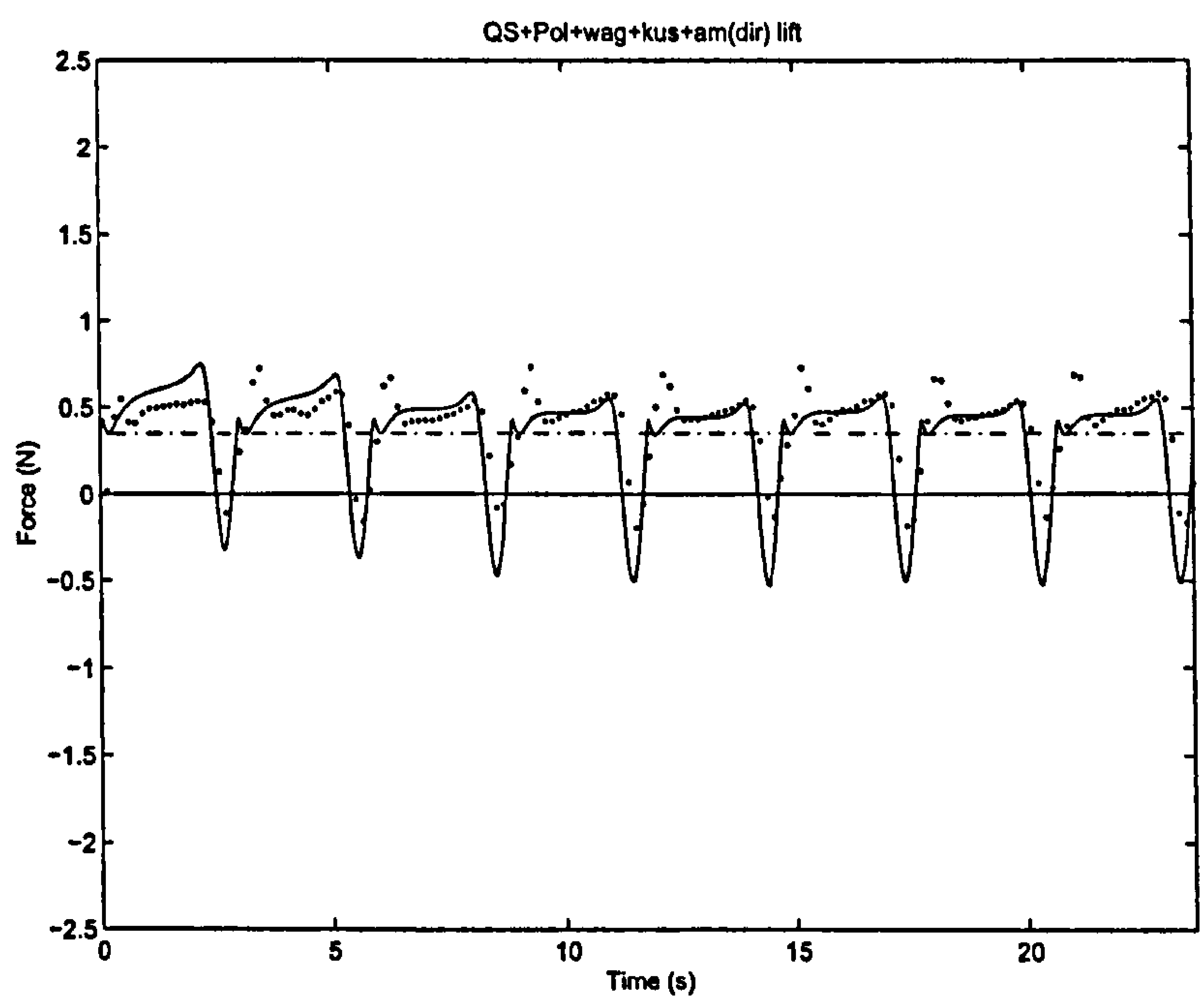


Figure 43: Total lift, including the Dirichlet component of added mass forces, for Robofly dataset (solid line) versus measured lift (dotted line). Chain line represents average predicted force.

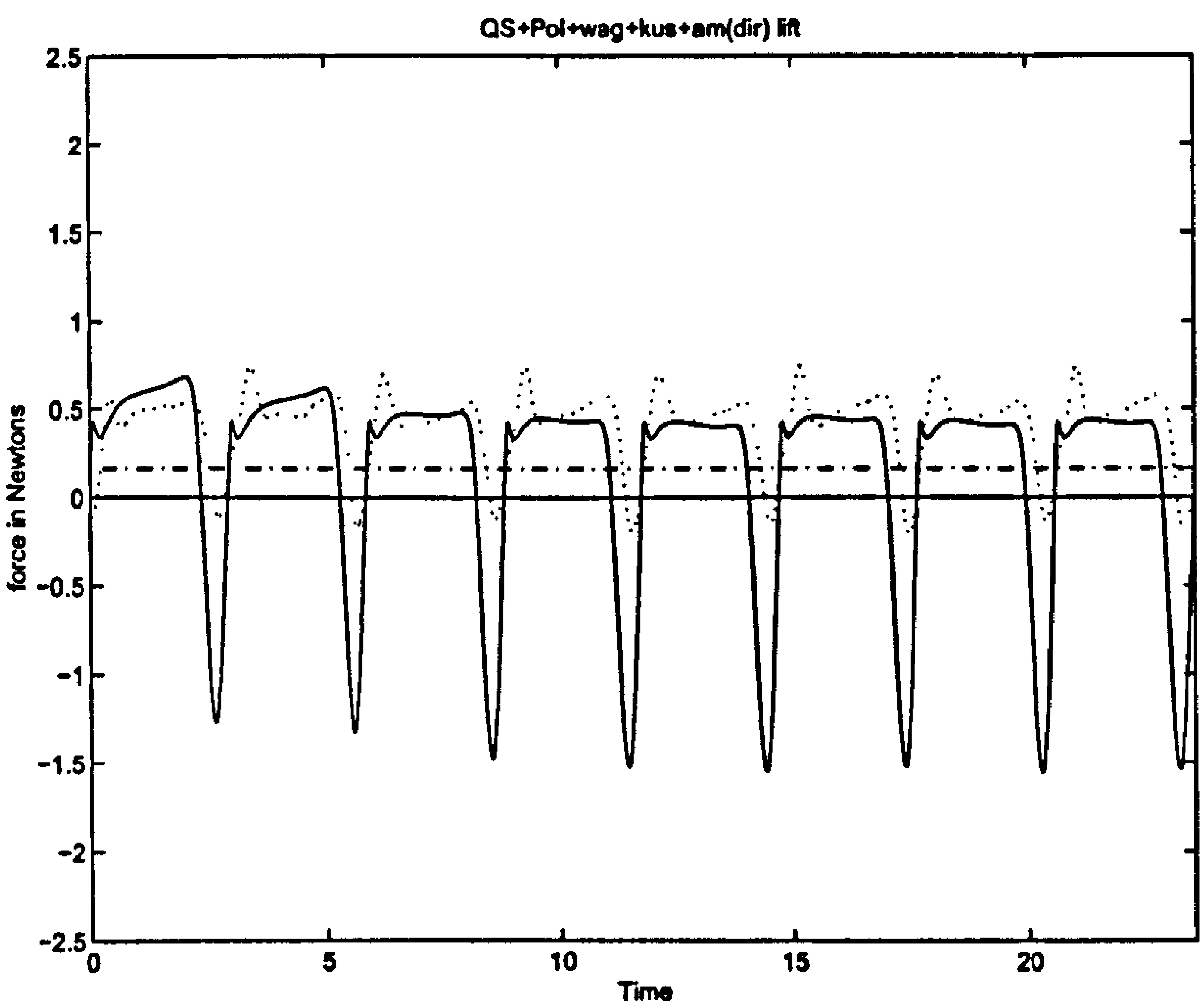


Figure 44: Robofly total lift with Dirichlet added mass correction, as in Figure 43, but with *usepolhamus='no'* (solid line) versus measured lift (dotted line). Note the loss of lift at reversal is much greater than in Figure 43, and the "bump" just before the reversal is lost. Chain line represents average predicted force.

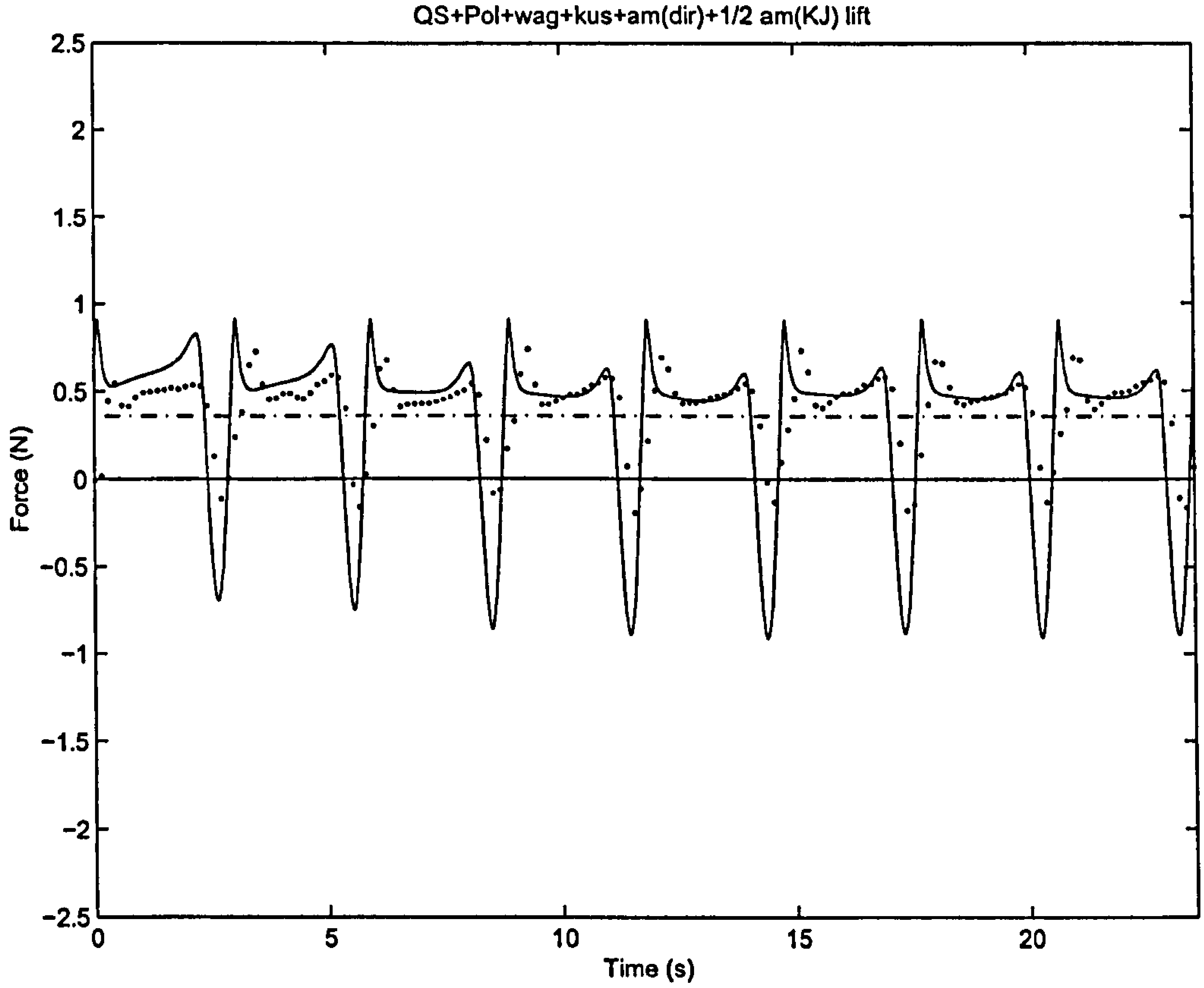


Figure 45: Total Robofly lift, including Dirichlet added mass forces, and half of the Kutta-Joukowski added mass force (solid line) versus measured lift (dotted line). Chain line represents average predicted force. Compare with Figure 41.

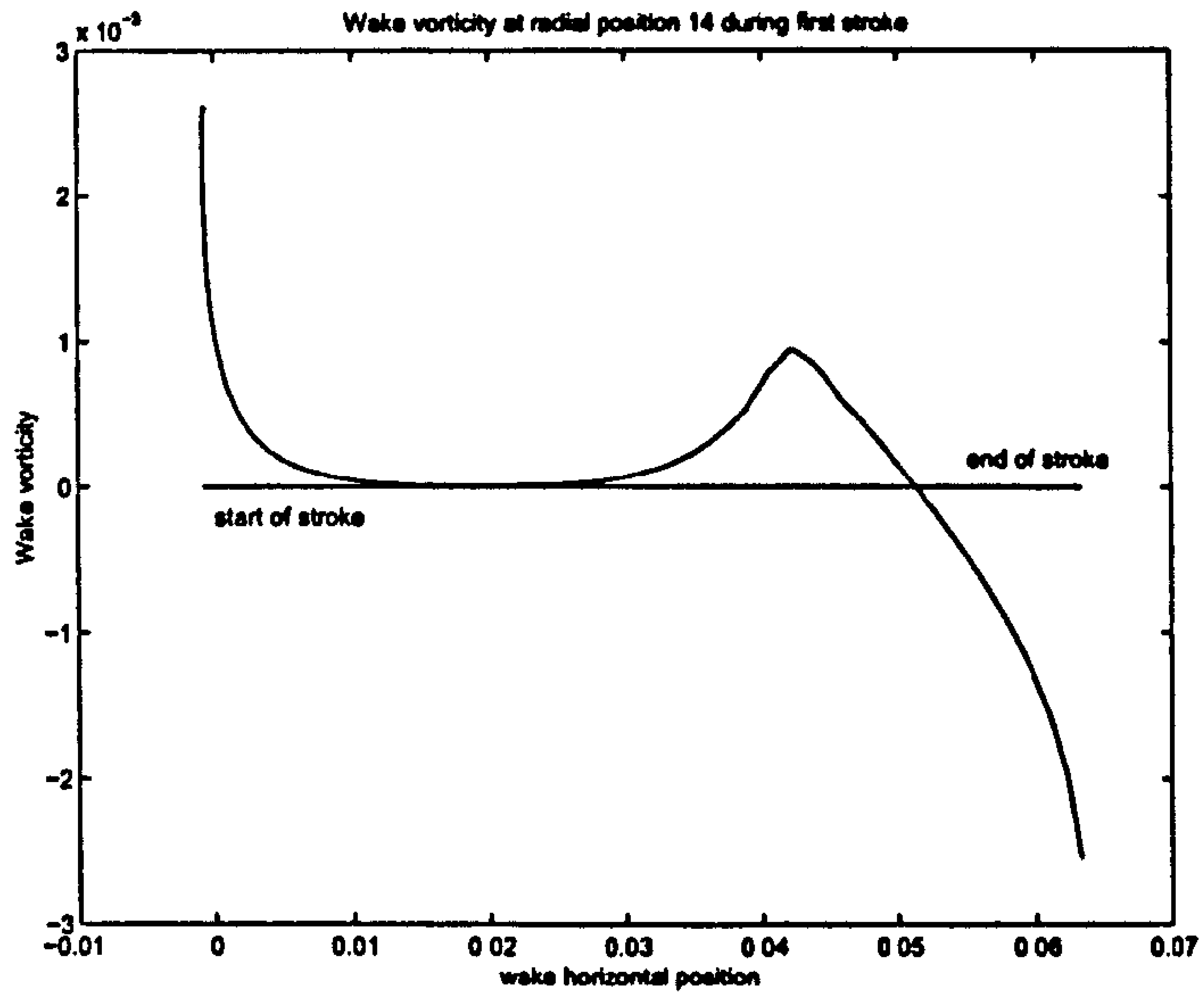


Figure 46: Calculated wake vorticity during first stroke for the Robofly dataset. The vertical scale depends on the size of the timesteps, and is unimportant. This figure is used only to illustrate the distribution of the vorticity. Note that the horizontal scale is now the horizontal position in the spherical coordinate system, not the time. Again, the units are unimportant.

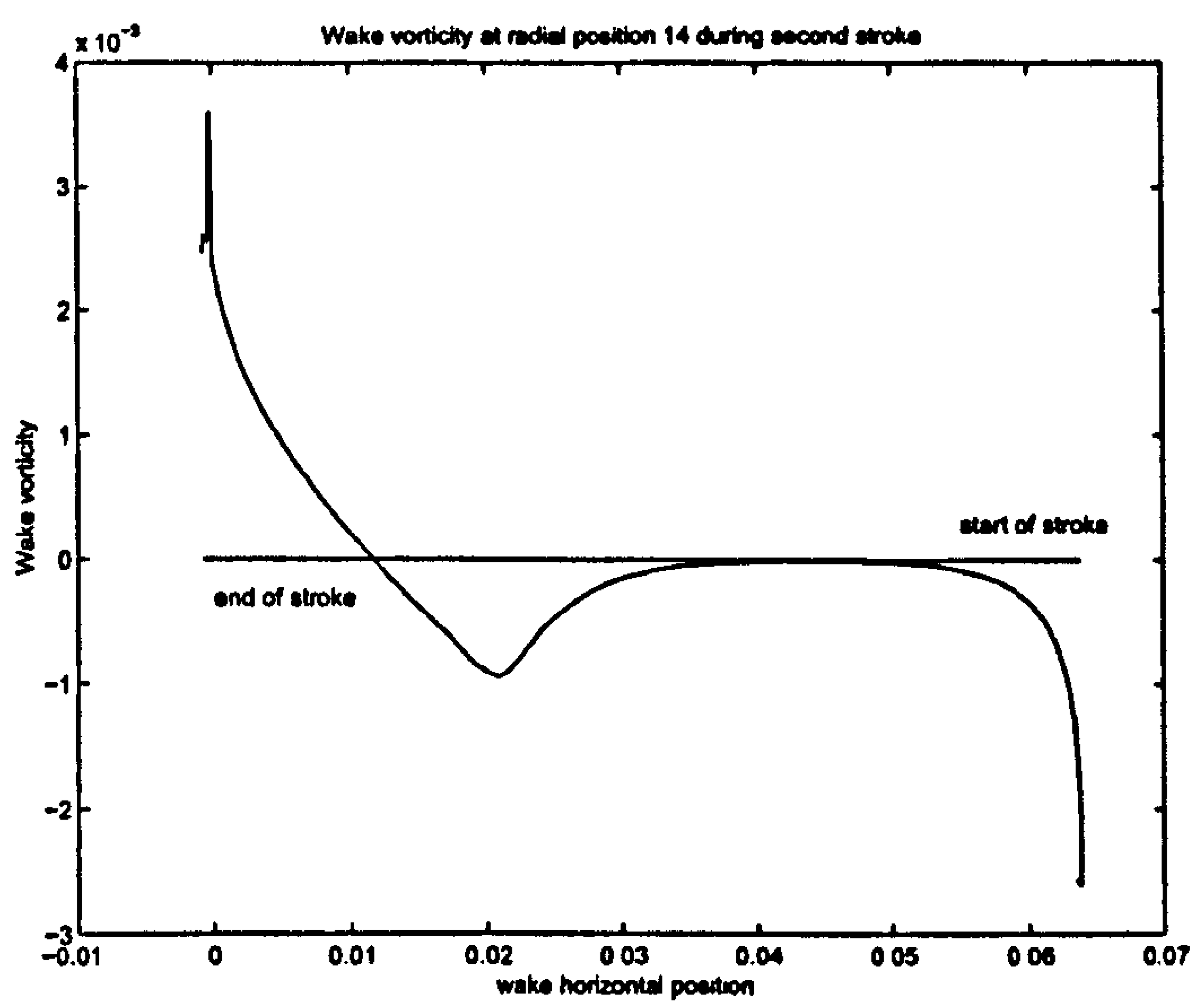


Figure 47: Calculated wake vorticity during second stroke for the Robofly dataset. The vertical scale depends on the size of the timesteps, and is unimportant. This figure is used only to illustrate the distribution of the vorticity. Note that the horizontal scale is now the horizontal position in the spherical coordinate system, not the time. Again, the units are unimportant.

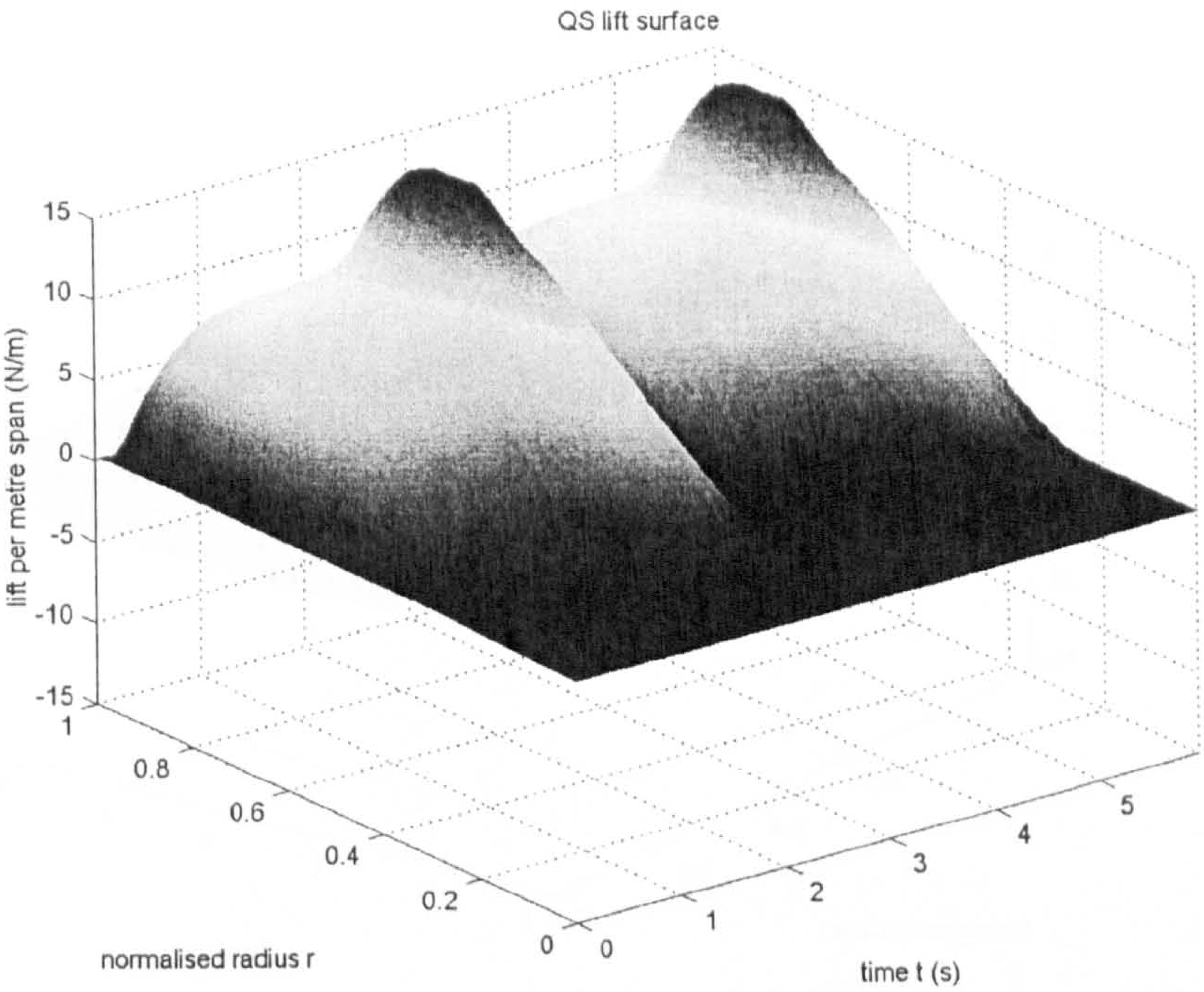


Figure 48: Surface of predicted local quasi-steady lift (F_V) per m span for Robofly data.

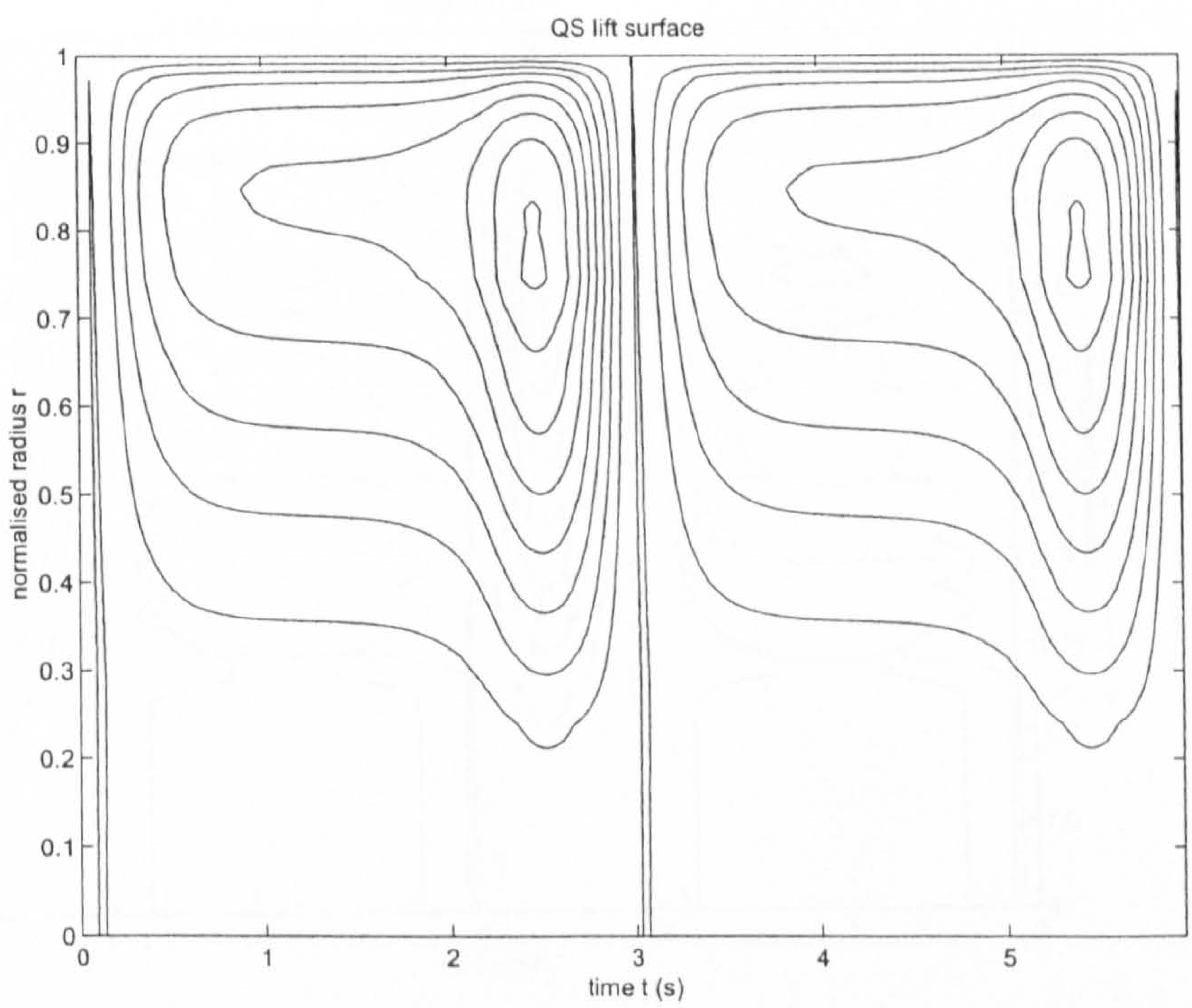


Figure 49: Contours of the predicted local quasi-steady F_V per m span for Robofly data.

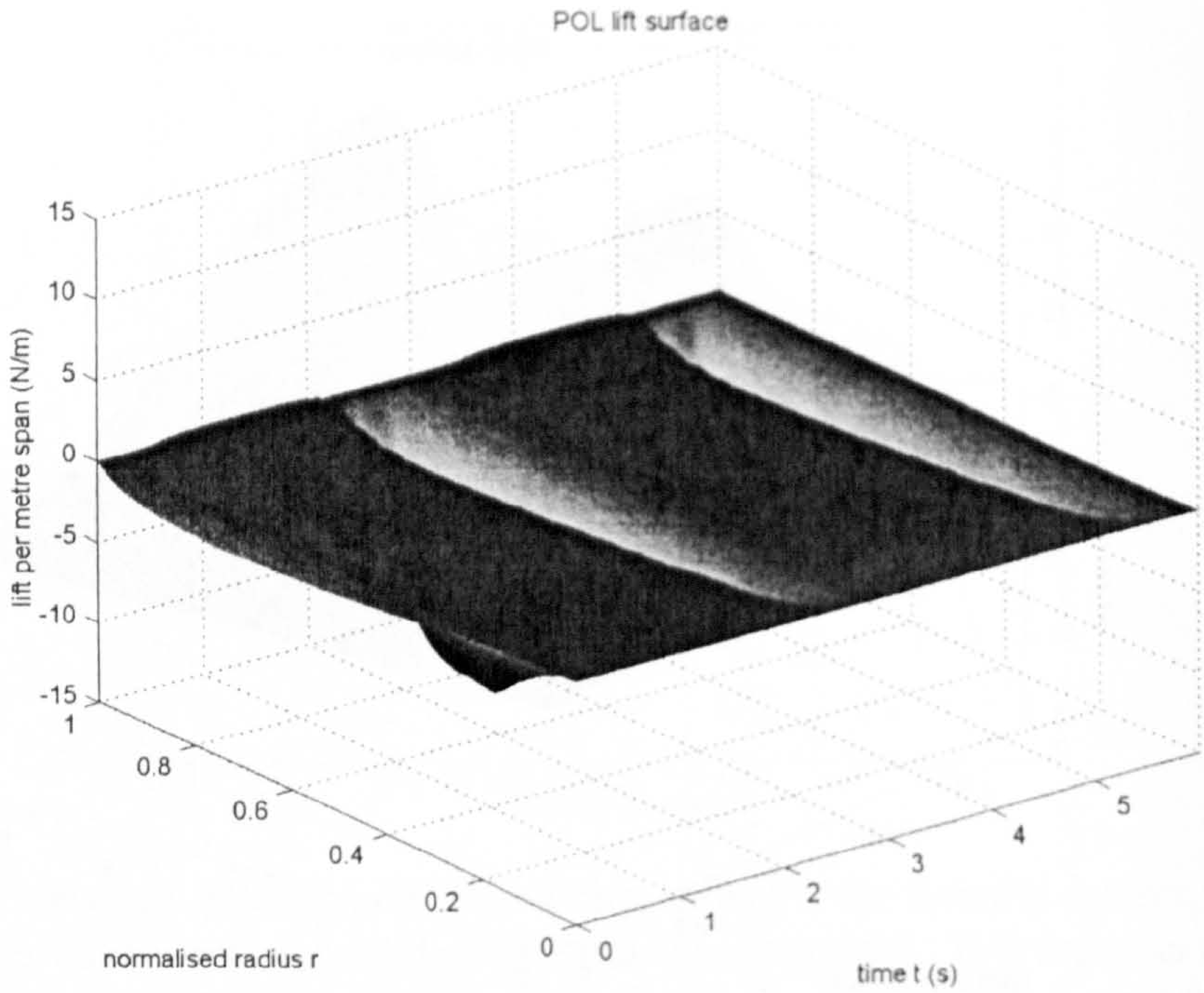


Figure 50: Surface of Polhamus correction to F_V for the Robofly dataset.

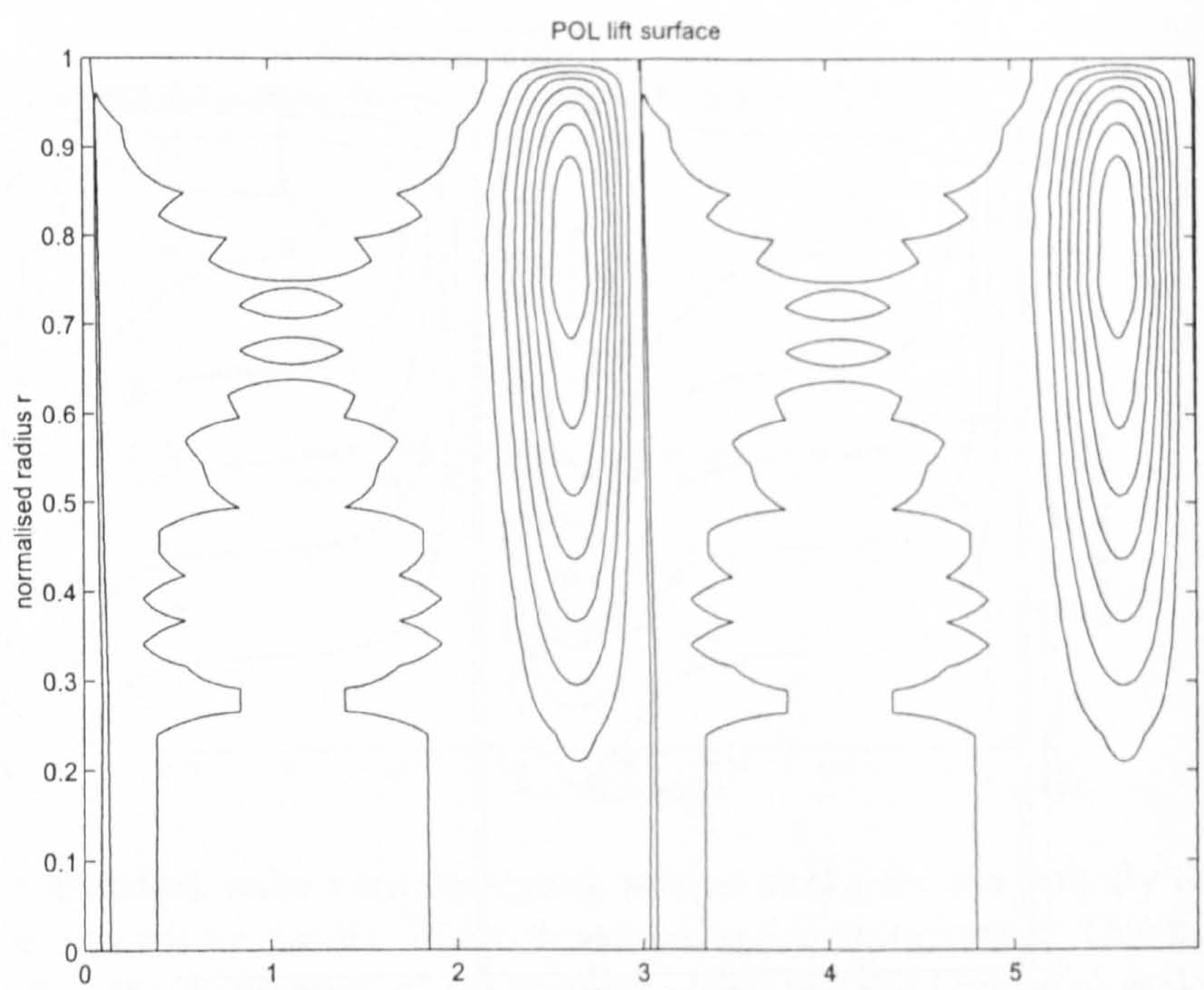


Figure 51: Contours of Polhamus correction to F_V for the Robofly dataset.

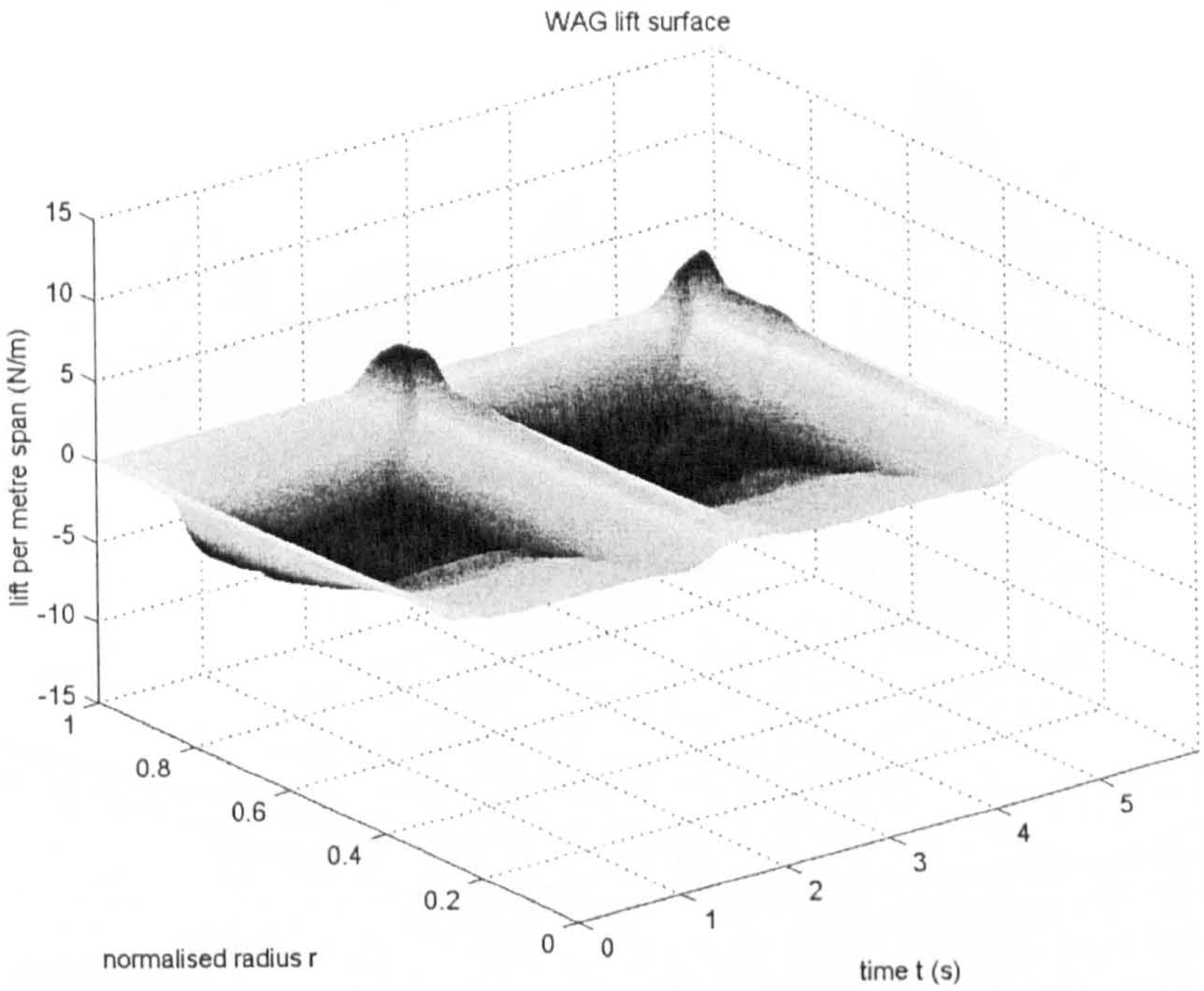


Figure 52: Surface plot of Wagner correction to F_V for the Robofly dataset.

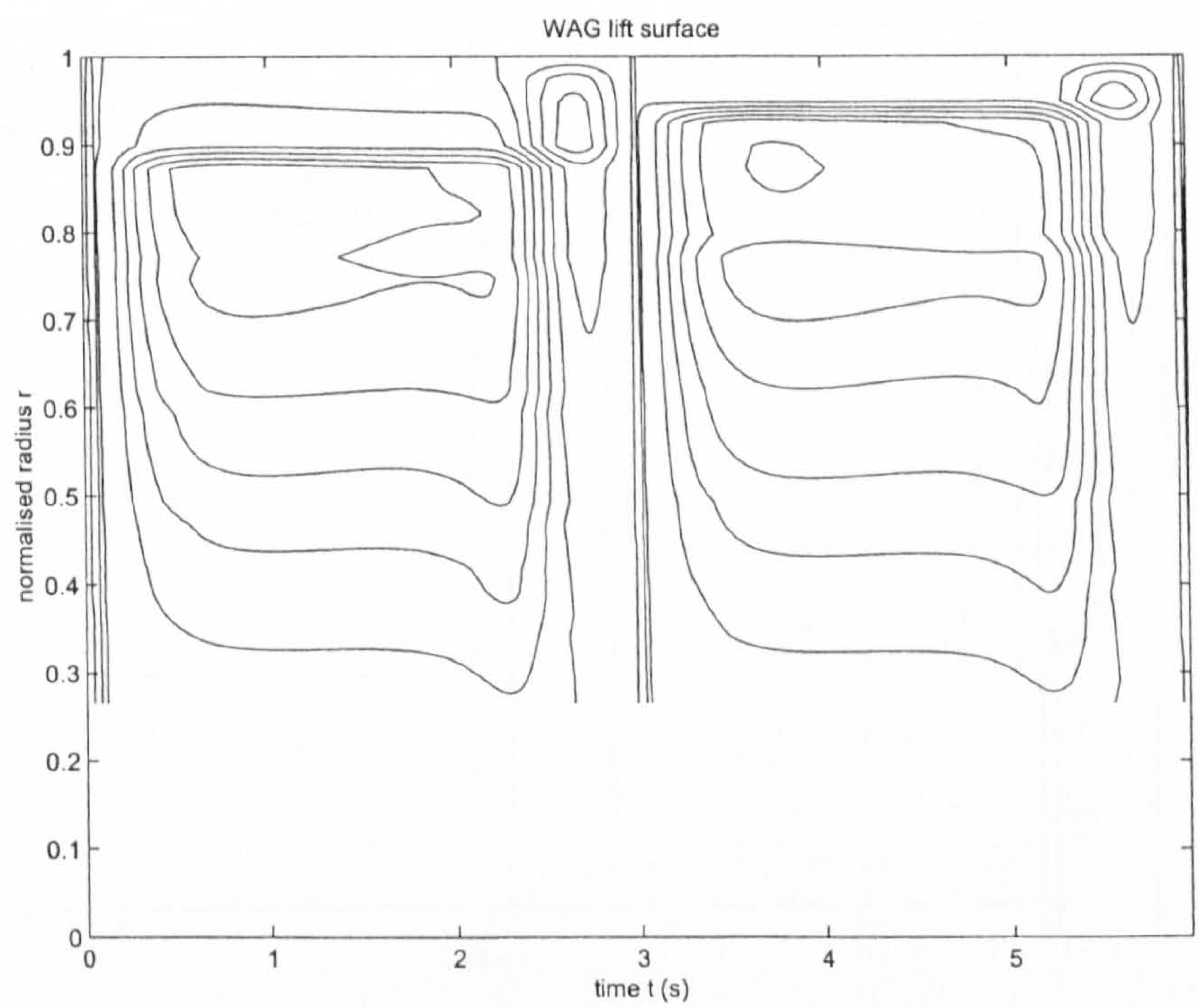


Figure 53: Contour plot of Wagner correction to F_V for the Robofly dataset.

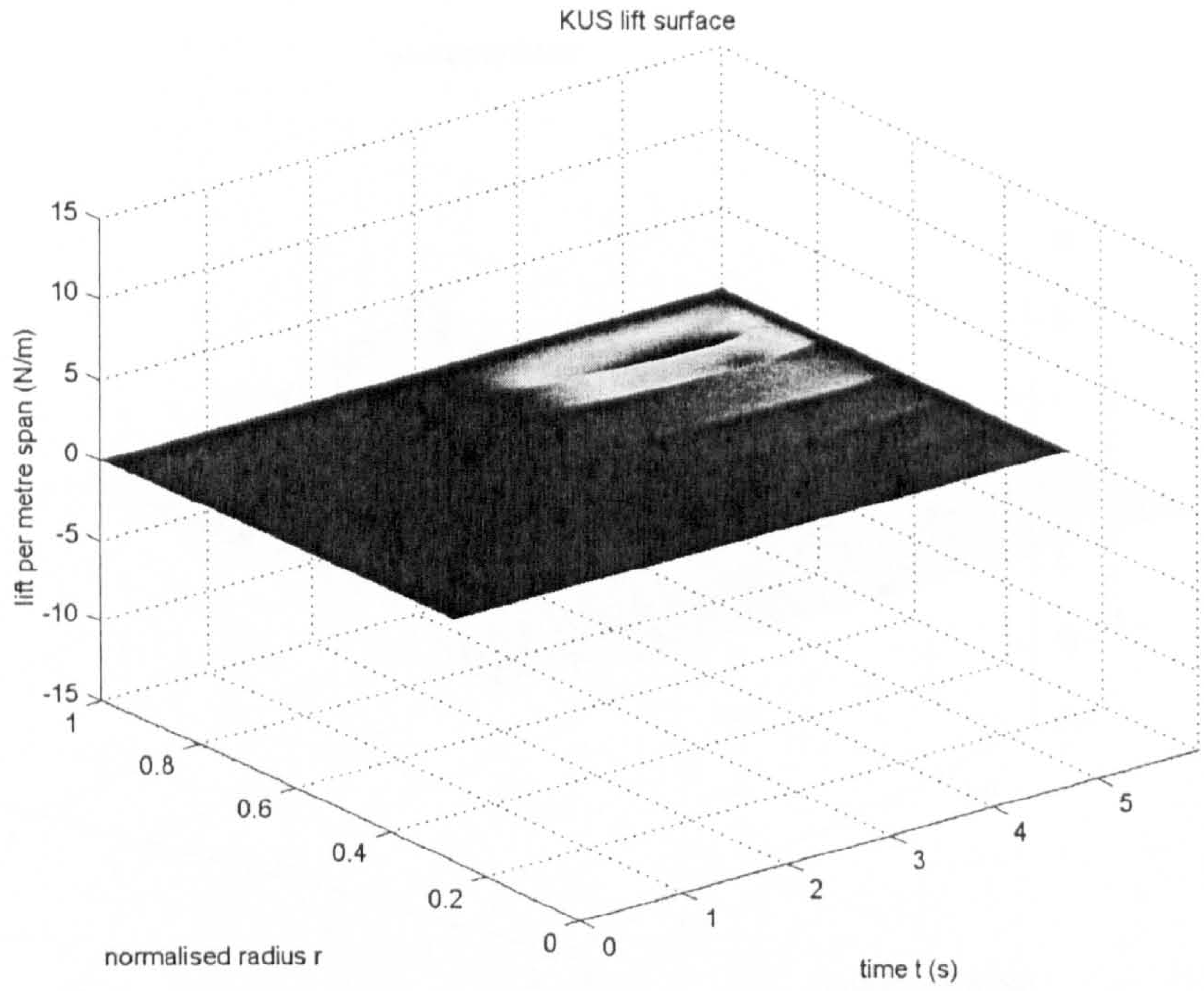


Figure 54: Surface plot of secondary wake (Küssner) correction to F_V for Robofly.

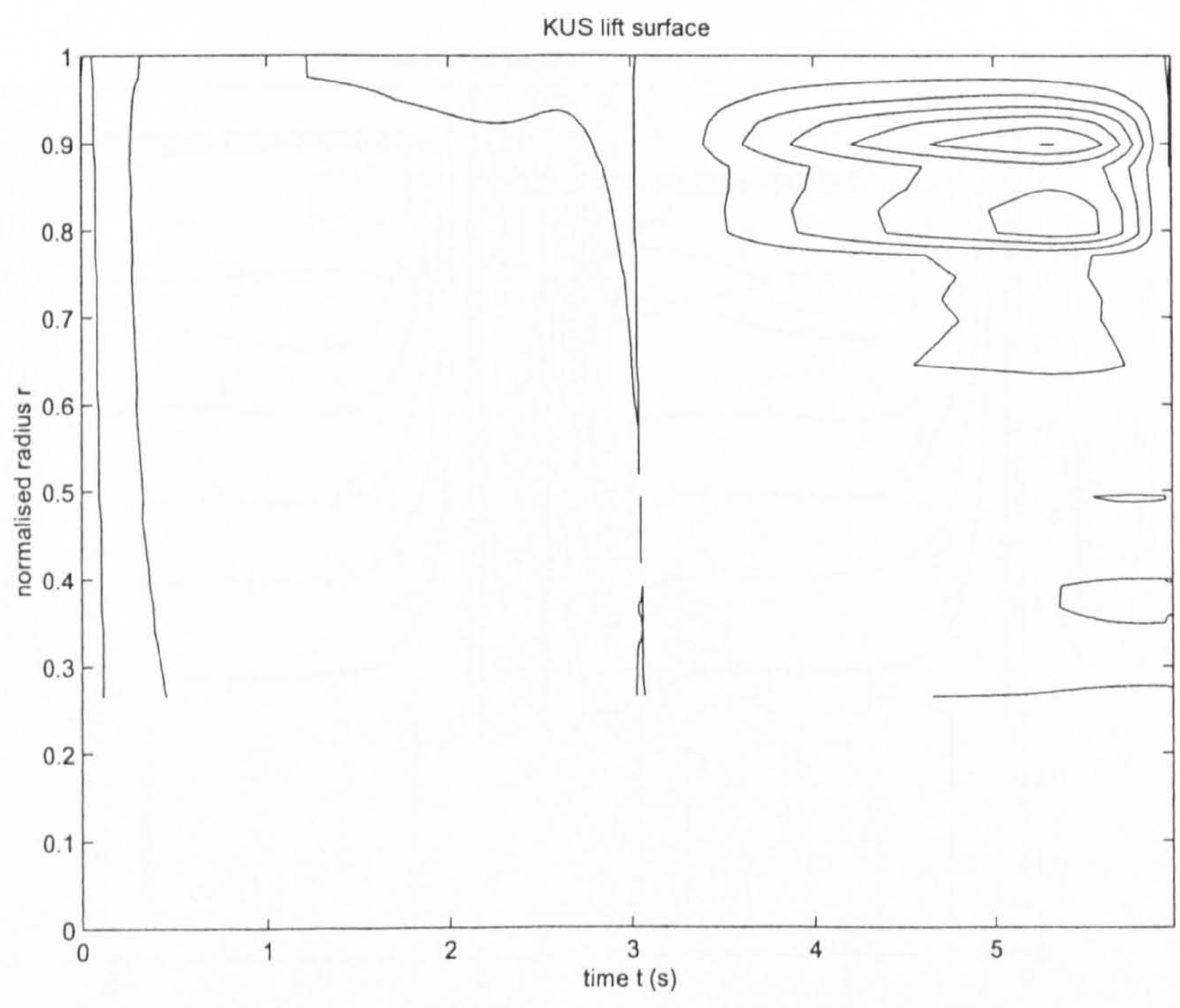


Figure 55: Contour plot of secondary wake (Küssner) correction to F_V for Robofly.

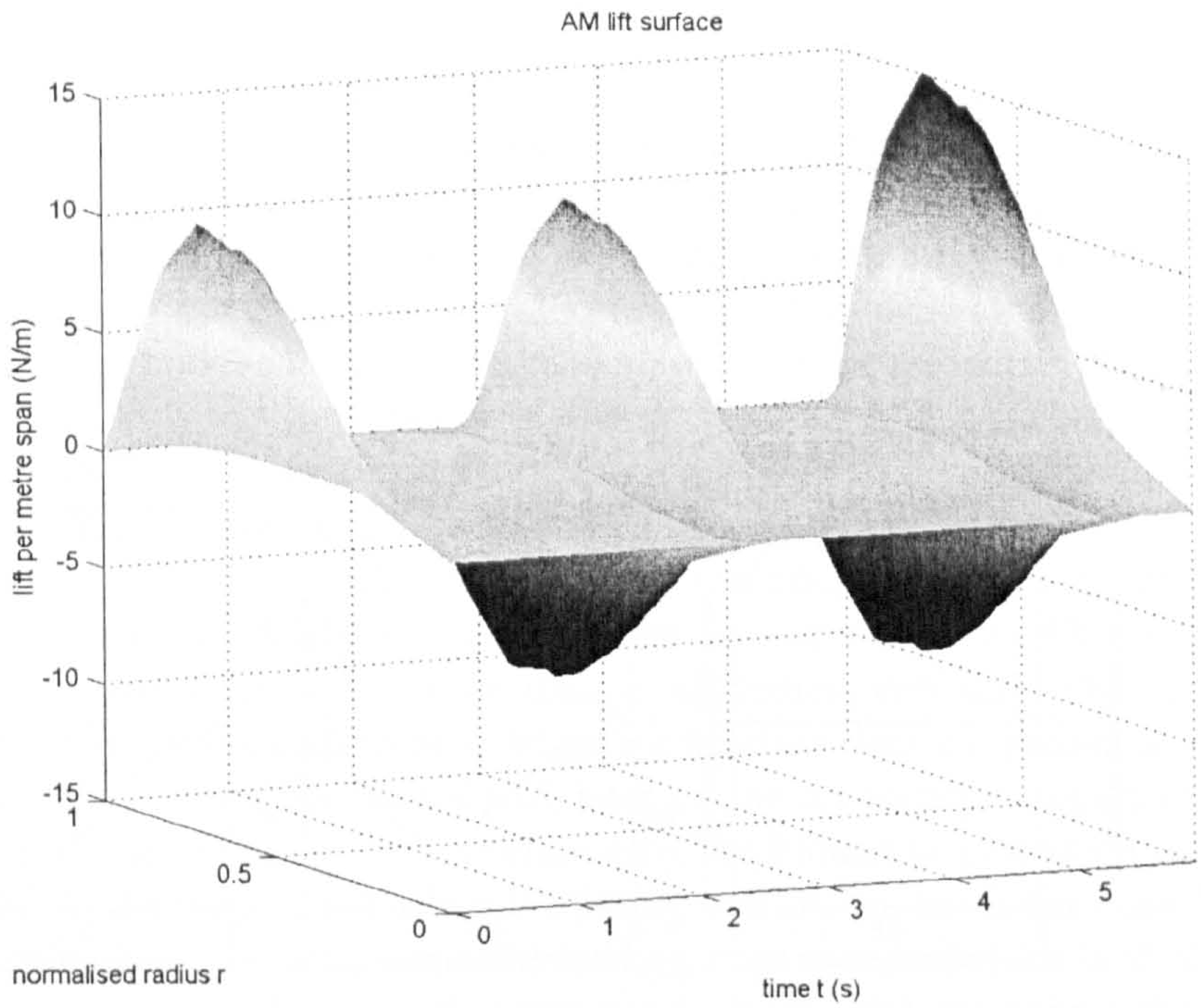


Figure 56: Surface plot of added mass correction to F_V for the Robofly dataset.

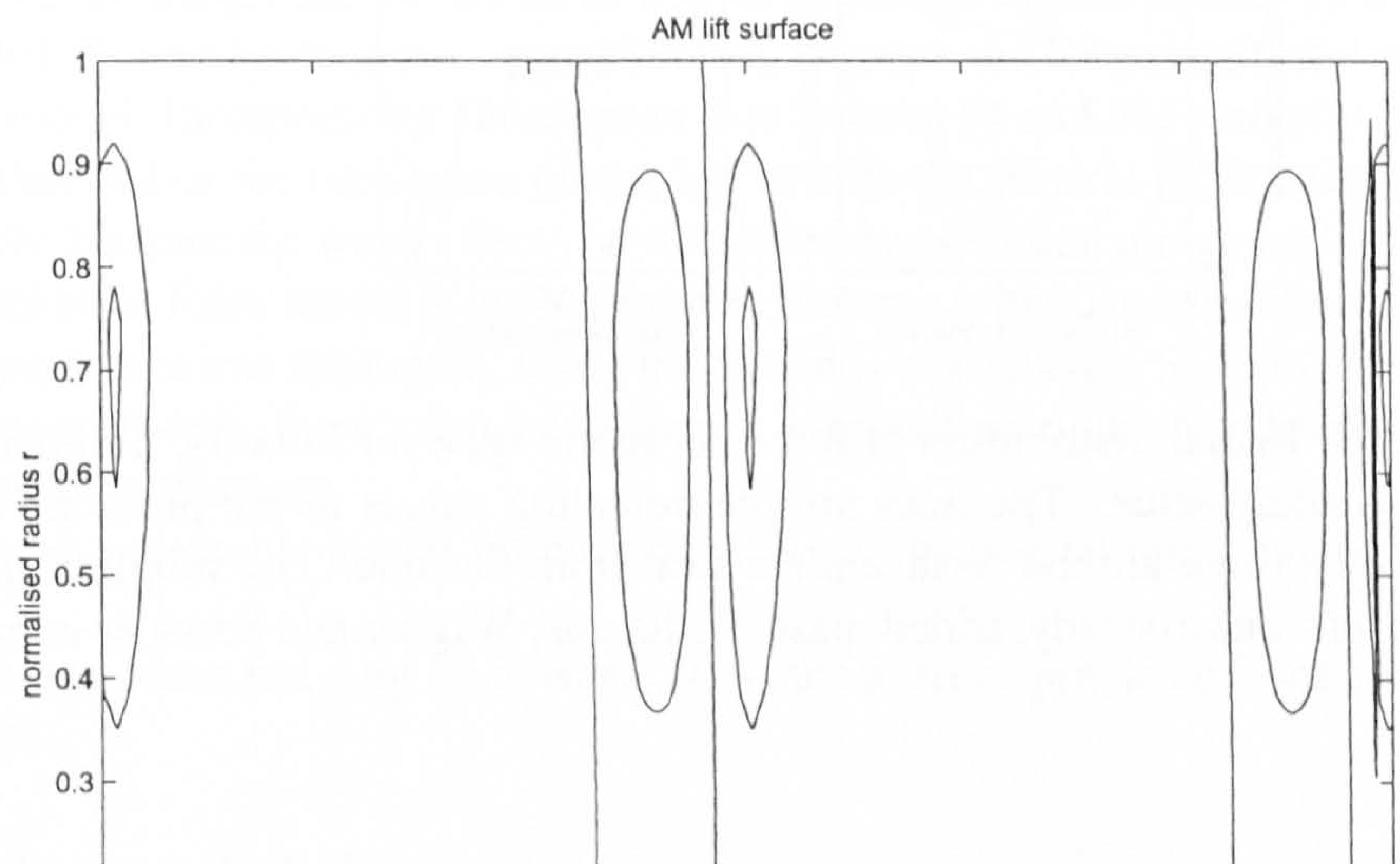


Figure 57: Contour plot of added mass correction to F_V for the Robofly dataset.

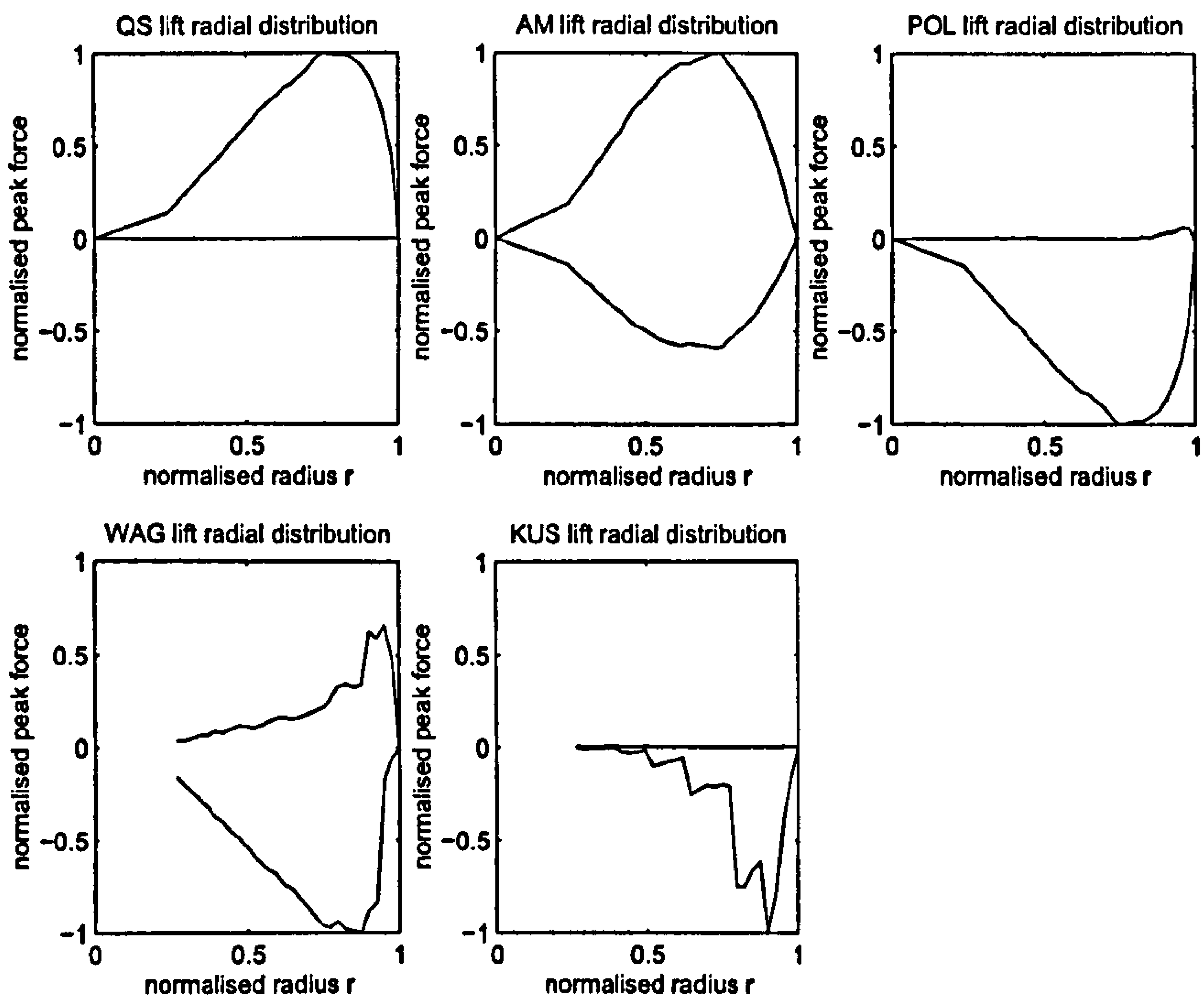


Figure 58: Radial distribution of force per metre span for Robofly, normalised with respect to the greatest value. The lines are the bounding values of the plots in Figures 48 to 57. Effectively, these are the surface plots seen from the side. The subplots are, in order from the top left: quasi-steady, added mass, Polhamus, Wagner and Küssner components.

15.2 The "drag" force

Dickinson defined drag in terms of the horizontal force in the direction opposing motion - therefore it was always positive. Our model defines horizontal force (F_H) in the $+\theta$ direction. Therefore, to compare with Dickinson's data, value of the predicted drag was multiplied by the sign of the horizontal tip velocity u_{HT}. The measured drag is shown in Figure 59. Considering Figure 60 it can be seen that there is almost no quasi-steady drag. This fits well with the theory, as there is very little vertical velocity of the midpoint. (There is no plunge anywhere, so the only vertical velocity is due to the rotation and the hinge offset from the midpoint, and the hinge is almost at the midpoint).

The Polhamus effect is considerable, as seen in Figure 61. Much like the quasi-steady lift force did, the Polhamus effect over-predicts the drag force by a factor of 2. This is because the Polhamus effect is a rotation of the quasi-steady suction force, and therefore scales with the suction force. However, the model predicts very little effect of the wake on drag - for the first component, because Wagner's function does not predict any effect due to the primary wake, and for the second part, because the secondary wake effect is small, just as it was for lift (see Figure 62). Therefore only the Polhamus correction is left, which is considerable. Polhamus did not affect lift during translation, but it has considerable effect on drag. Again, this is because when the leading edge suction (which is at 45°) is rotated, its vertical component is almost unchanged, but the horizontal component changes sign.

Considering the total drag for all components apart from added mass in Figure 63, it can be seen that the fit is poor: although the Polhamus effect correctly identifies the peak during each rotation, the magnitude of the force is over-predicted approximately by a factor of 2, just as for lift. The wake does not correct for this, because the Wagner effect does not affect drag in the model. Incorporating added mass (see Figures 64 and 65) correctly identifies the peaks at either end of the translation phase, but as with the vertical force, they are too large and too early, because the wake effects on the added mass forces are ignored.

The horizontal force model is not acceptably accurate without some refinement. Scaling the Polhamus effect was attempted, using the methods described in Section 10.5, but due to the low rotation speeds, the scaling was 1 almost everywhere, with a few localised spikes of lower value at the rotations.

Without an acceptable model for horizontal force, the overall force cannot be predicted accurately.

The average measured drag force was $0.60N$, the average predicted force was $1.00N$, an error of 66%.

15.2.1 Primary wake influence on drag

As can be seen from Figures 36, 38 and 62, the main effect of the wake is to reduce the predicted lift and drag forces at midstroke.

It is postulated that the majority of the error between the measured and predicted drag in Figure 65 is due to the omission of primary wake (Wagner) effects on the drag. From Figure 67 it can be seen that the Wagner effect on lift is almost exactly opposite half of the combined quasi-steady and Polhamus lift. Effectively, the Wagner effect is halving the quasi-steady and Polhamus contributions to lift. Assuming a similar effect of the primary wake on drag gives the result of Figure 68, which can be seen to be very close to the measured value. When using this correction, the average drag force was $0.4066N$, an under-prediction of 32%. This method of primary wake correction for drag is too tenuous to be relied on - note especially how it completely eliminates the first peak after reversal, because the primary wake effect should be delayed relative to the change in quasi-steady force. However, it does support the postulation that the majority of the error in predicted drag force is due to the omission of primary wake effects.

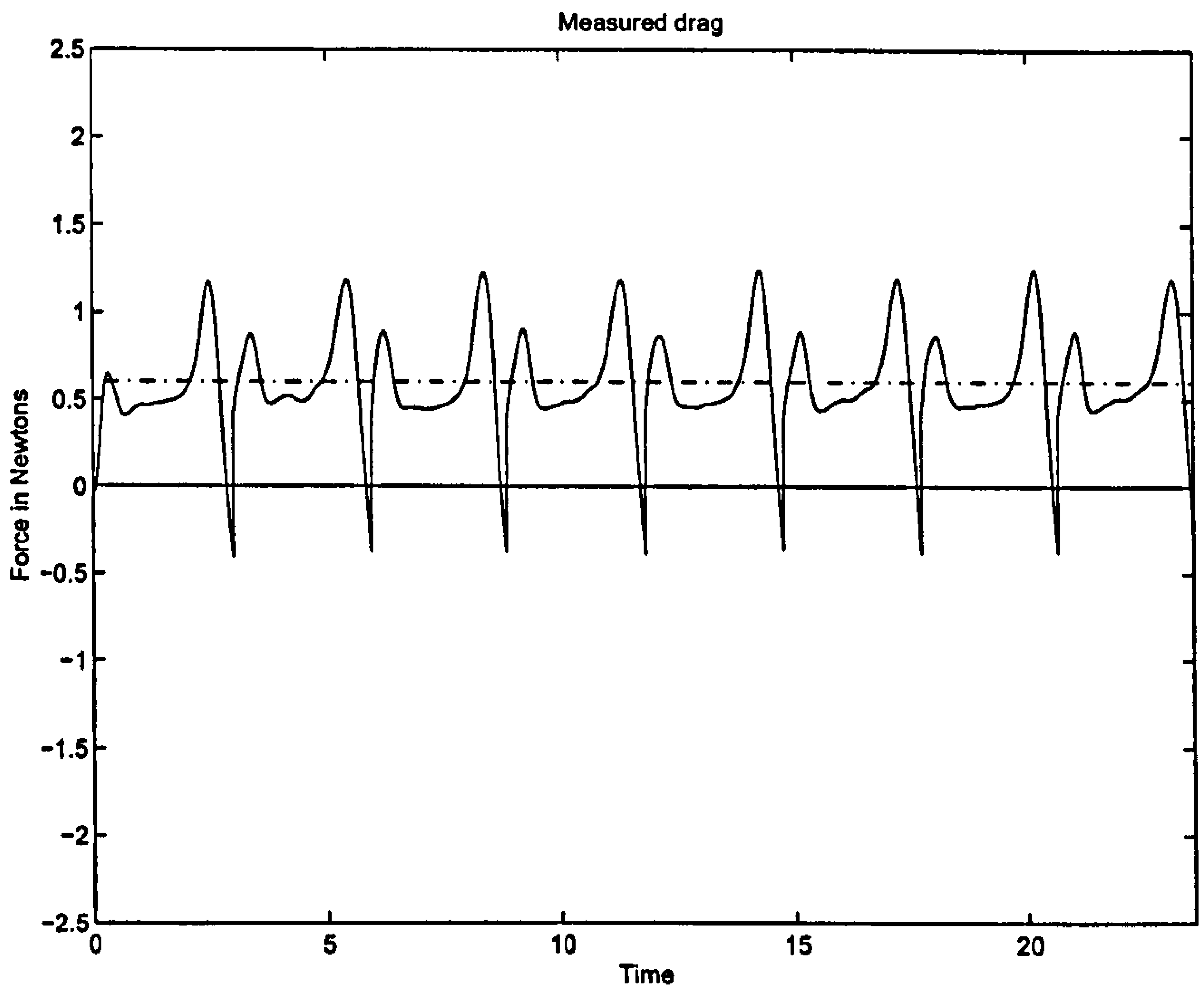


Figure 59: Measured drag for Robofly data. Chain line represents average measured force.

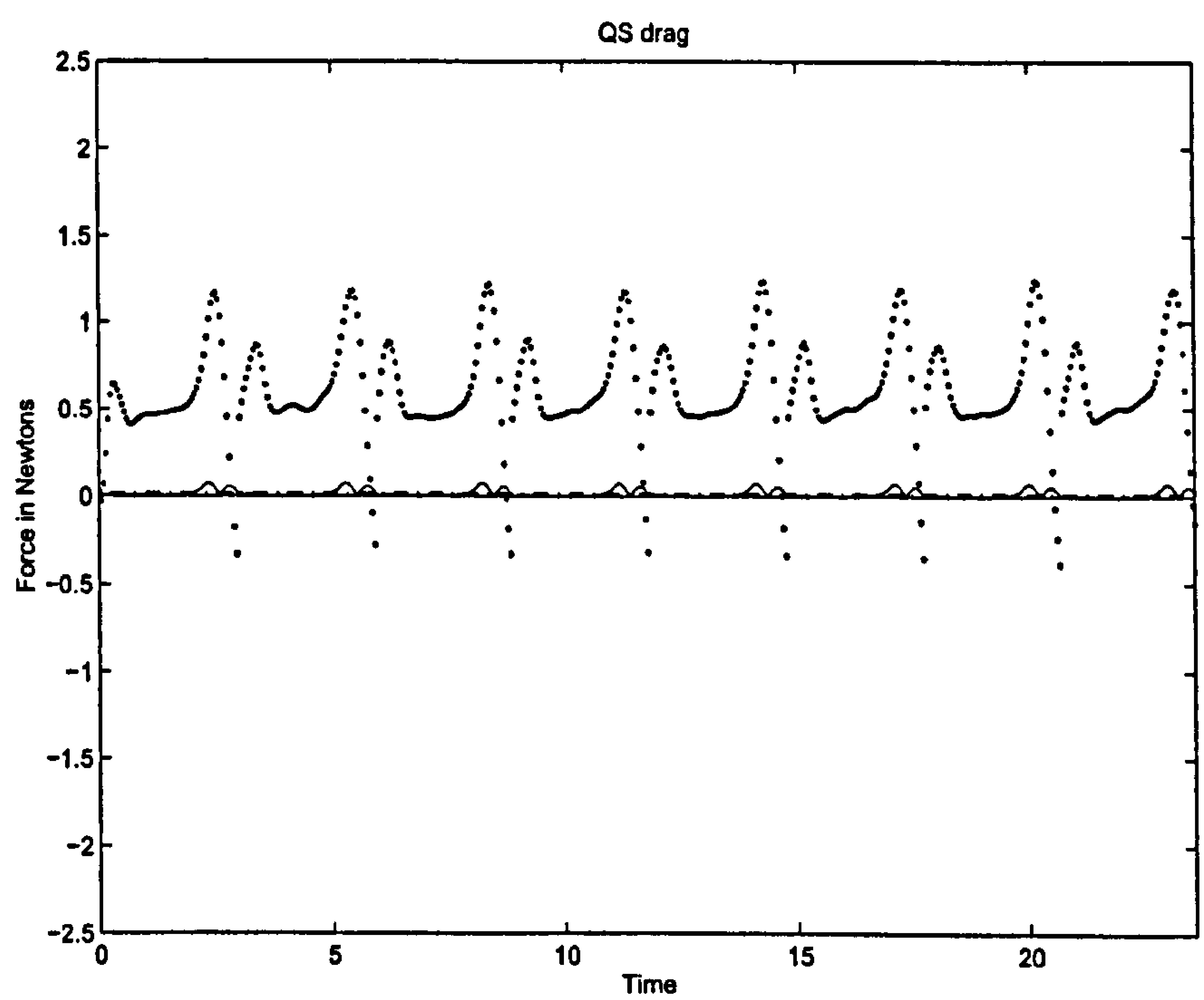


Figure 60: Quasi-steady drag (absolute value of F_{HQW}) for Robofly data (solid line) versus measured drag (dotted line). Chain line is average predicted force.

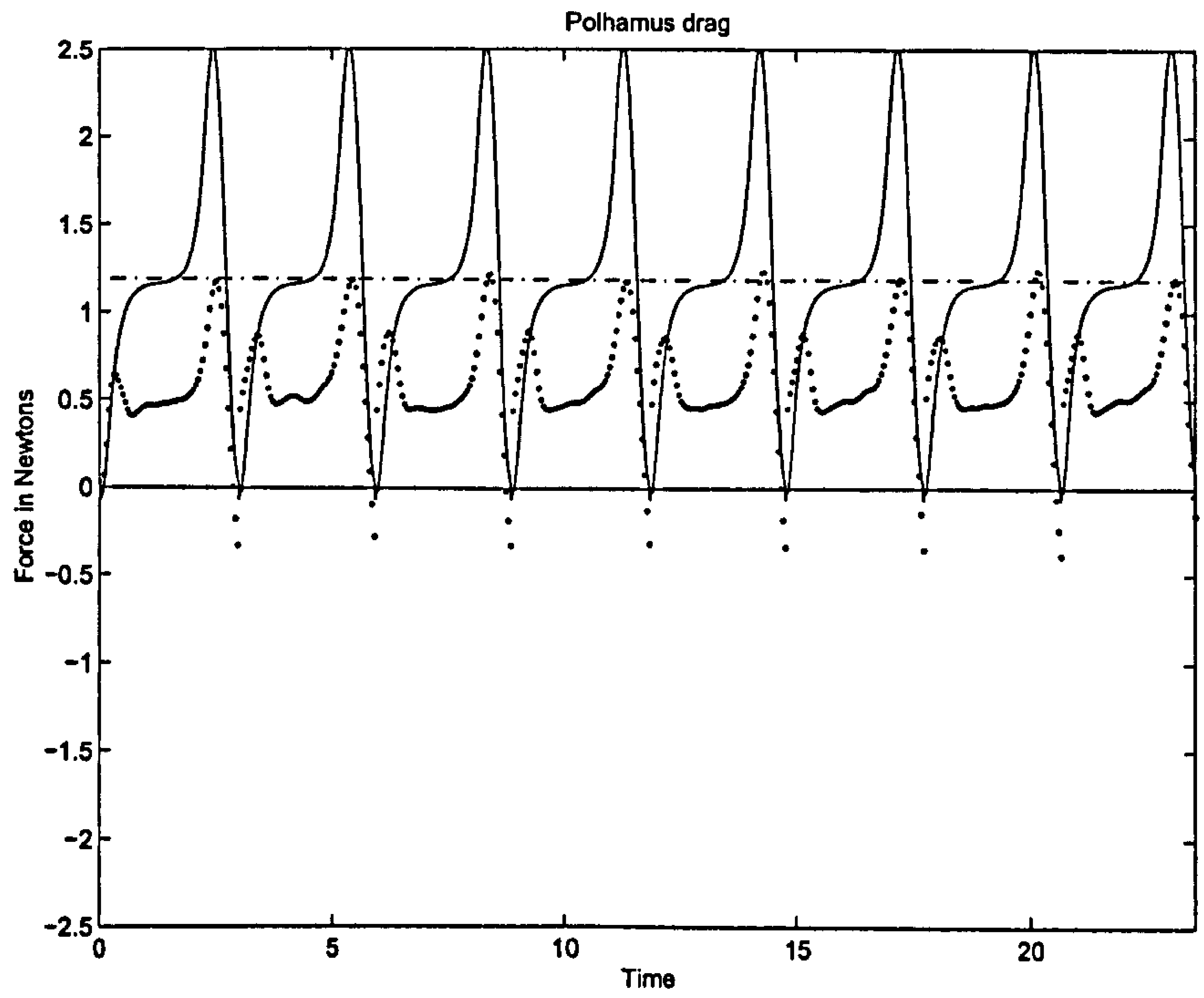


Figure 61: Polhamus correction to drag, for Robofly data (solid line) versus measured drag (dotted line). Chain line is average predicted force.

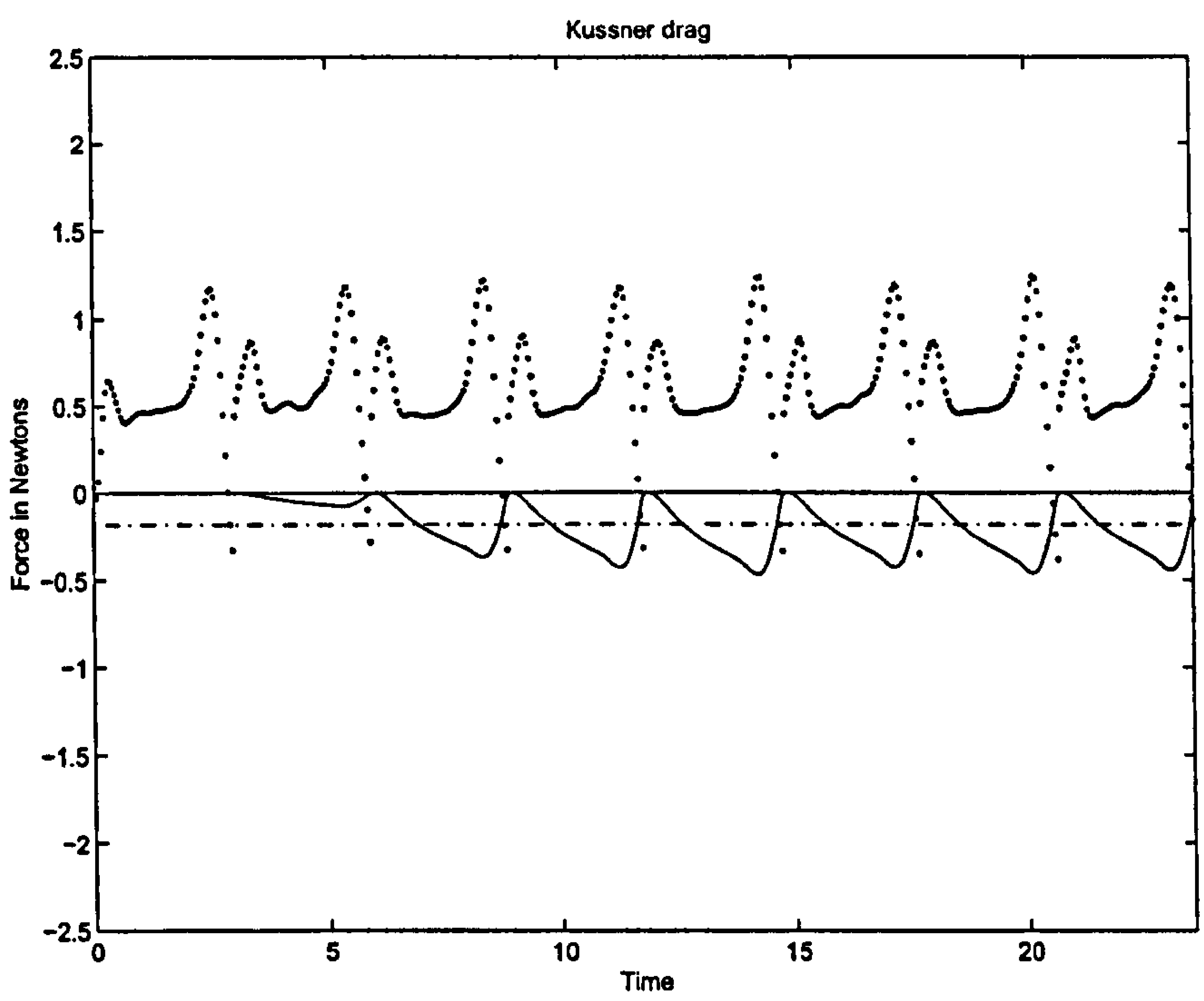


Figure 62: Secondary wake (Küssner) correction to drag for Robofly (solid line) versus measured drag (dotted line). Chain line is average predicted force.

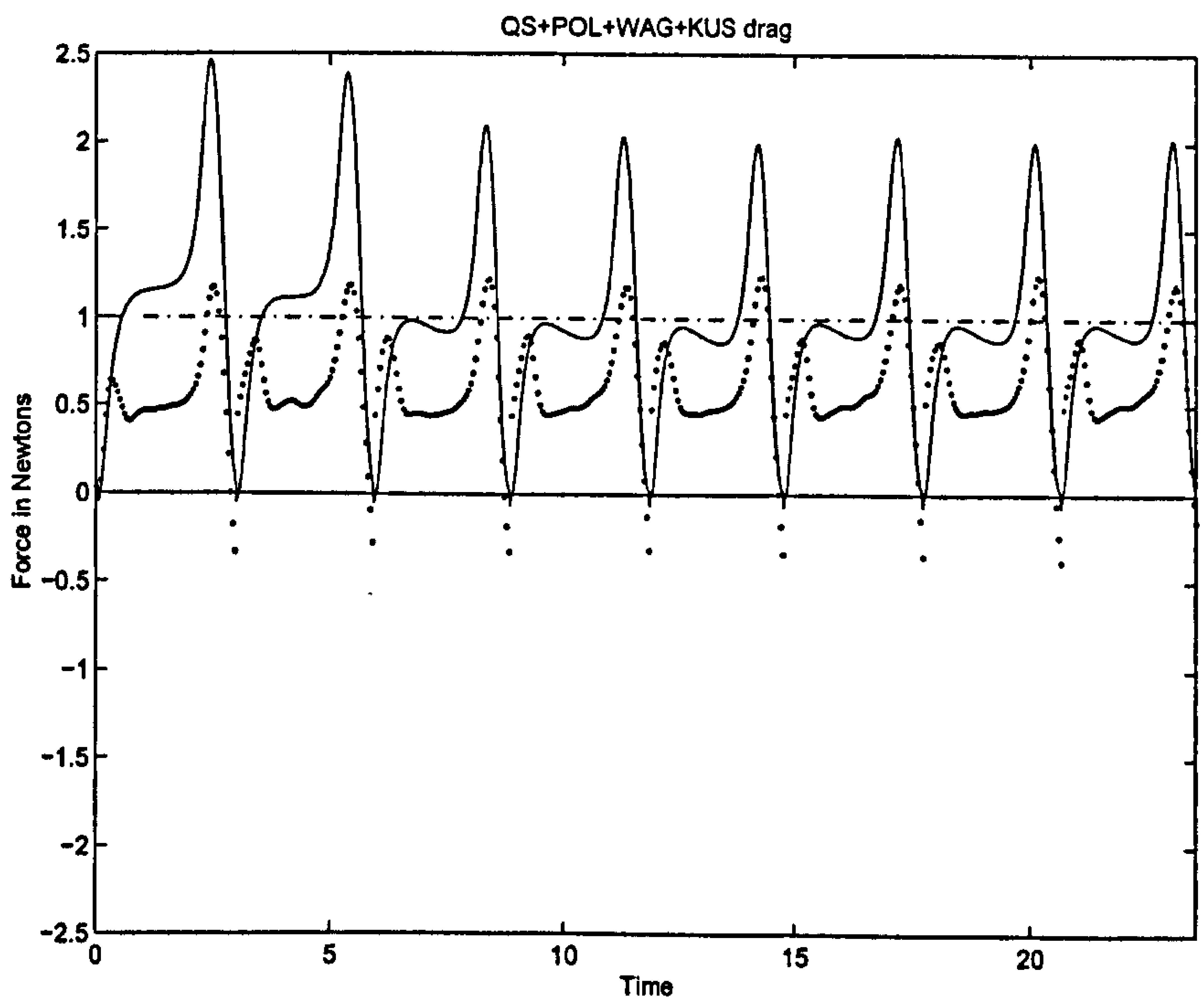


Figure 63: Total drag without added mass for Robofly dataset (solid line) versus measured drag (dotted line). Chain line is average predicted force.

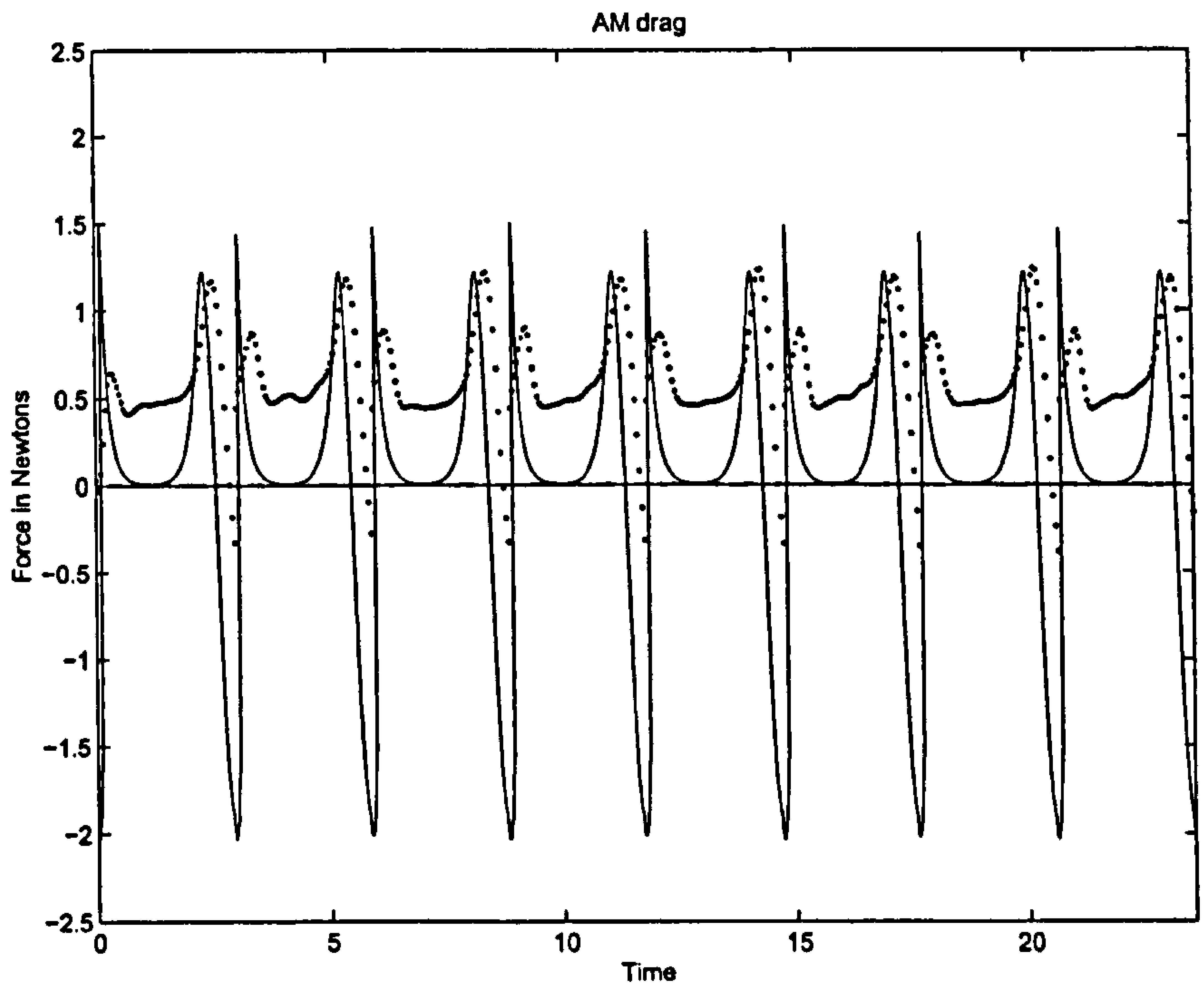


Figure 64: Added mass correction to drag for Robofly (solid line) versus measured drag (dotted line). Chain line is average predicted force.

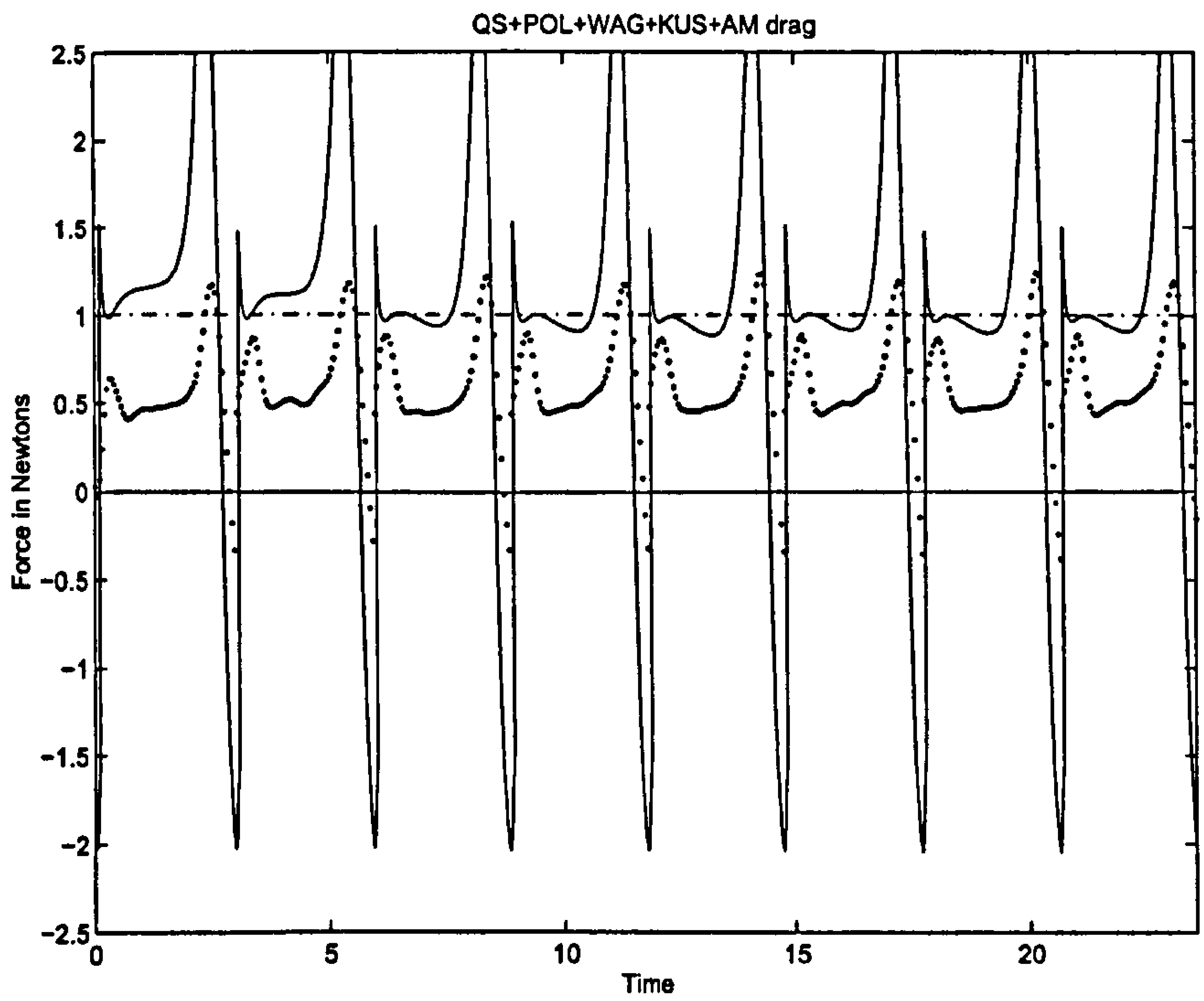


Figure 65: Total drag with added mass for Robofly dataset (solid line) versus measured drag (dotted line). Chain line is average predicted force.

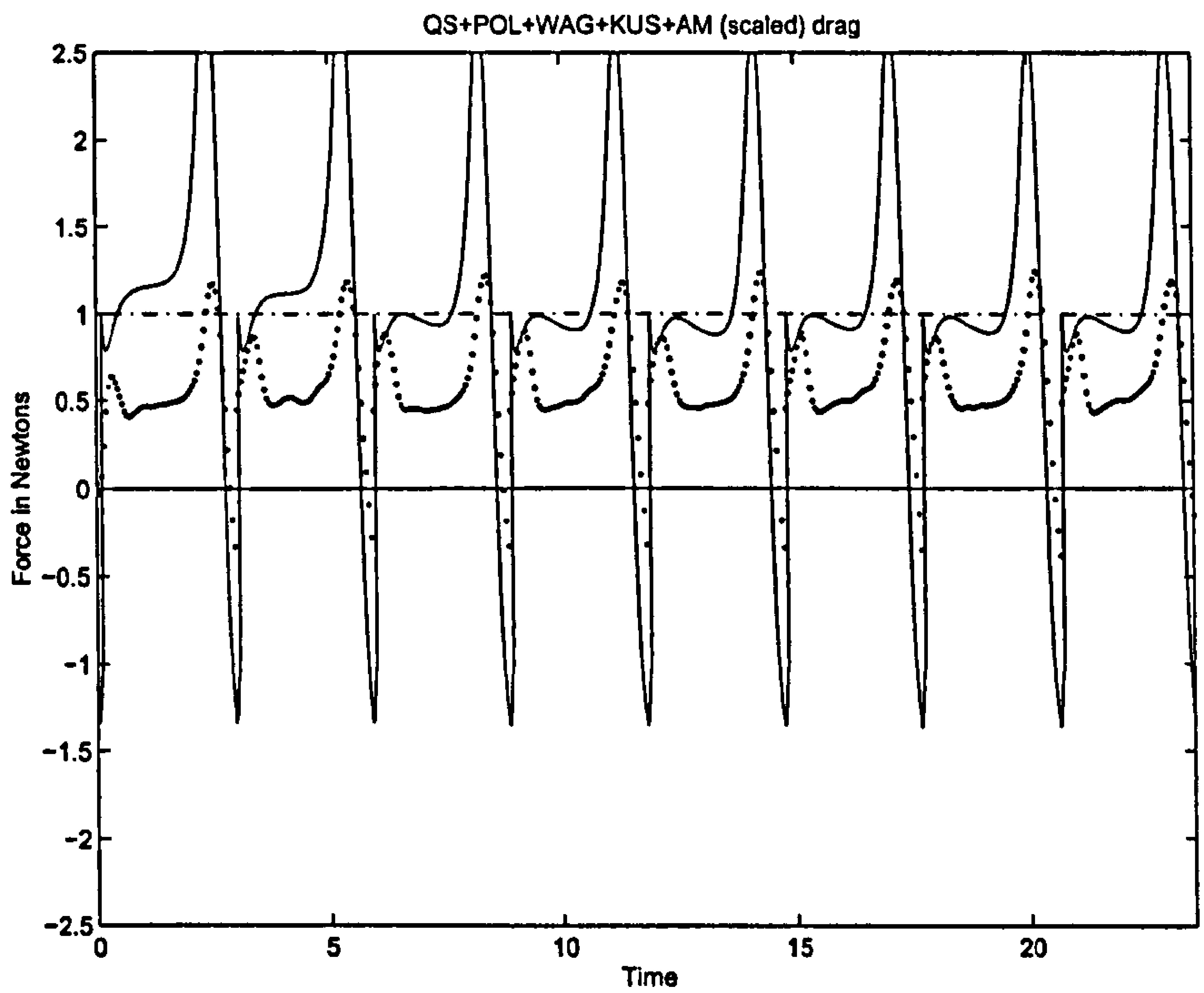


Figure 66: Total drag with added mass for Robofly dataset, including Dirichlet added mass forces, and half of the Kutta-Joukowski added mass force (solid line) versus measured lift (dotted line). Chain line represents average predicted force. Compare with Figure 65.

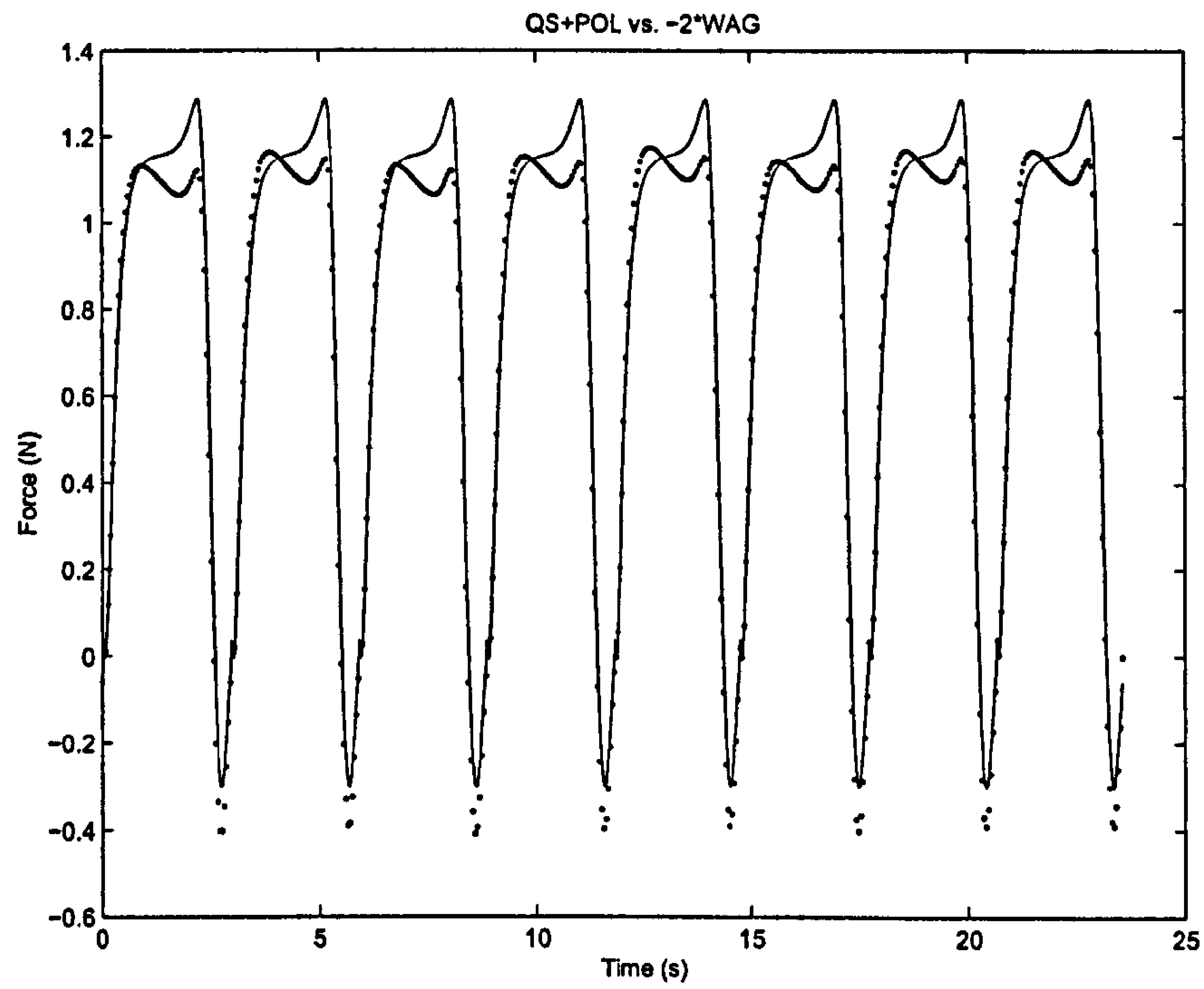


Figure 67: Comparison of predicted Polhamus-corrected quasi-steady lift (solid line) with minus twice the Wagner primary wake lift effect (dotted line). Note the strong correlation.

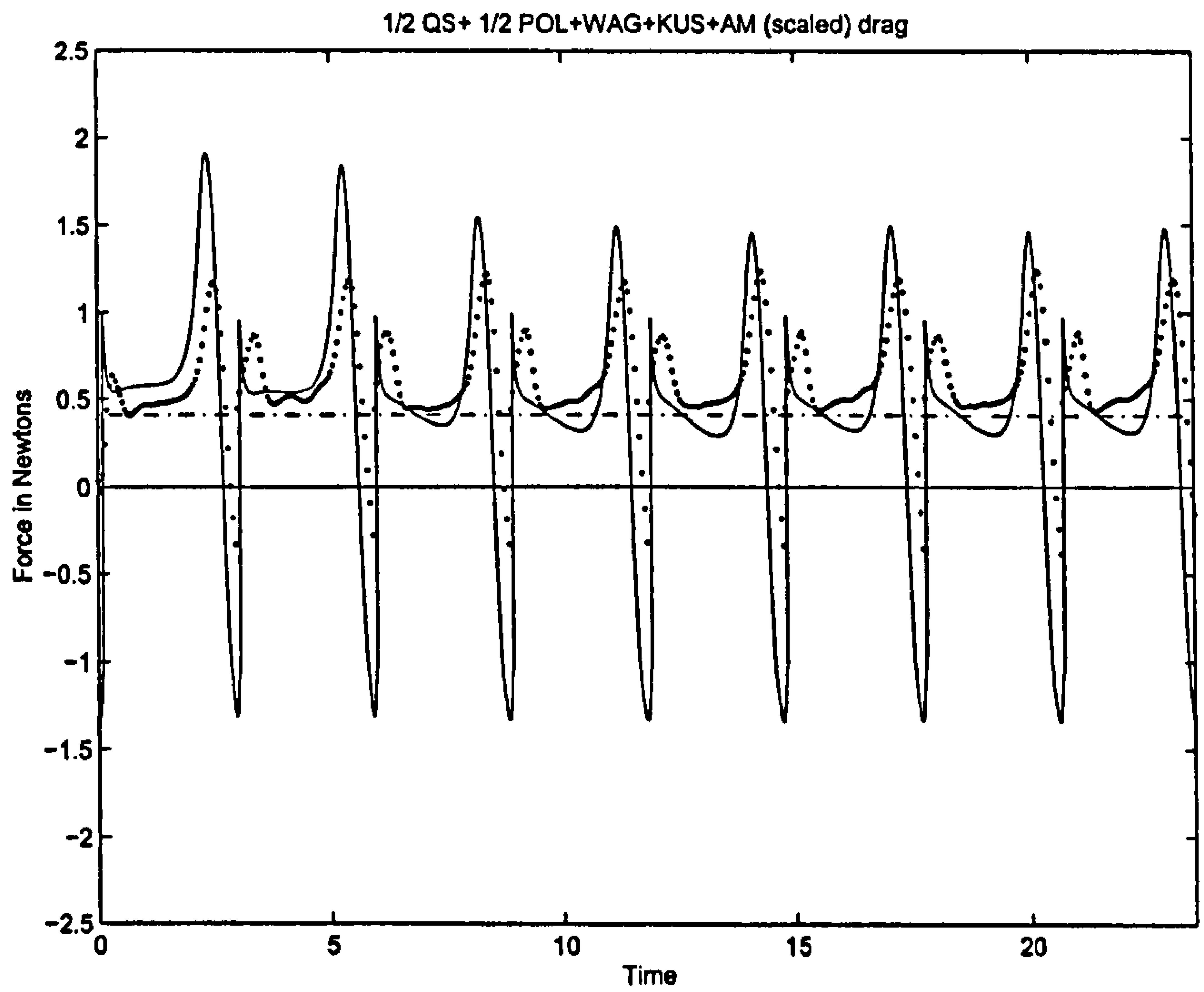


Figure 68: Total Robofly drag, as in Figure 65, but minus half of quasi-steady and Polhamus contribution (solid line) versus measured drag (dotted line). Chain line represents average predicted force.

16 Dataset: FMAV 50/2

The FMAV 50/2 is a theoretical design, based on an overall weight of 50g, with a wing length of 15cm, flapping at a cycle frequency of 20 Hz. The results of this run are included to highlight some effect of our model that are not apparent from the Robofly dataset alone. The full set of results for this run will not be shown, only the ones that are of special interest. The results of interest are the effect of plunging on the quasi-steady forces, the effect of treating the cycle as part of a repeating pattern, rather than an impulsive start, and illustrating that the added mass averages to 0 during a cycle. Also, this dataset is used as an illustration of a purely analytical dataset, with a prescribed wing shape, that can use wing shape parameters to form wing integrals of the forces, rather than numerical integration.

16.1 Geometry

The wing geometry is loosely based on that of the hoverfly, as shown in Figure 69. It is made up of four quarter ellipses, with the maximum chord at 75% of the tip radius. The hinge line is at 25% of chord. The tip radius is 150mm, and the maximum chord 50mm, giving an aspect ratio[3] of $12/\pi$.

16.2 Kinematics

The kinematics are based on a simple Lissajous curve, giving a figure-of-eight motion with a horizontal stroke plane, and two antisymmetric strokes. The sweeping and plunging motions are both sinusoidal, with the plunging motion being upwards at reversal, and downwards during midstroke. The sweep amplitude is 60°, so the sweep covers a segment of 120°. The plunging amplitude is 1/8 of this. The kinematics and resulting velocities are show in Figures 70 and 71.

The pitching motion is based on the expression $\frac{1}{2}(\sin(4\,tt) + 4\,tt) + \pi/2$, where tt is the phase angle $2\pi\ t/T$. This was chosen because the velocity tends asymptotically to 0 at the interface between translational and rotational motion. This is shown in Figure 72.

Code parameters

- RADIAL POINTS: 12, evenly distributed. Inner point at $r = 0$, outer at $r = 1$.
- TIME POINTS: 2048 evenly distributed across a single cycle.
- ROTATION: 180°. Wing is horizontal during translation.
- PERIOD = 1/20
- RHO = 1.225. This is the fluid density for air (kg/m^3)
- DATALENGTH = full cycle. The data represent a single, closed cycle.

[3]Using the definition that aspect ratio is R^2/A, where R is wingtip radius, and A is wing area

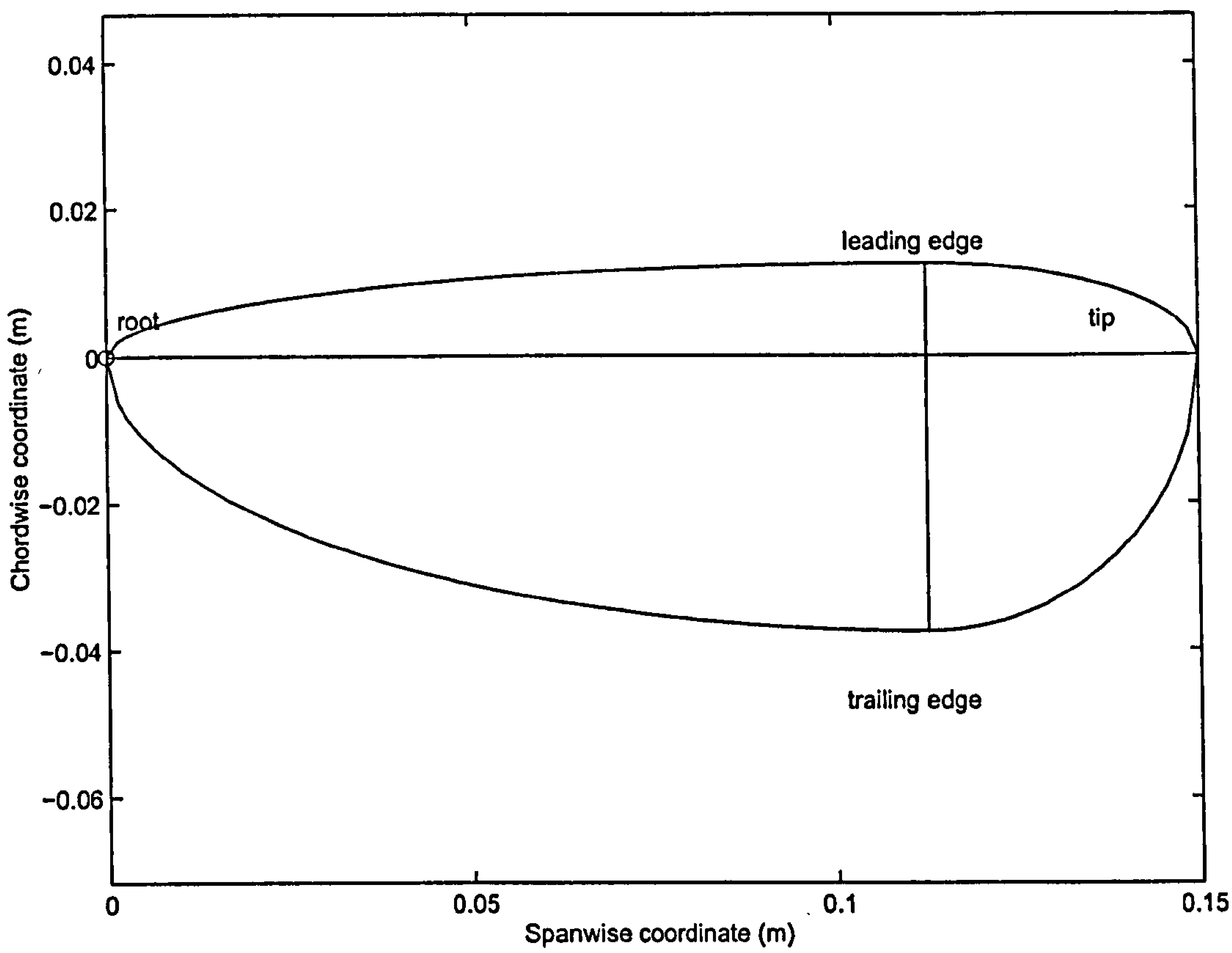


Figure 69: FMAV 50/2 wing shape: the root of the wing is located at the circle, the horizontal and vertical lines are the hinge line and maximum chord line, respectively.

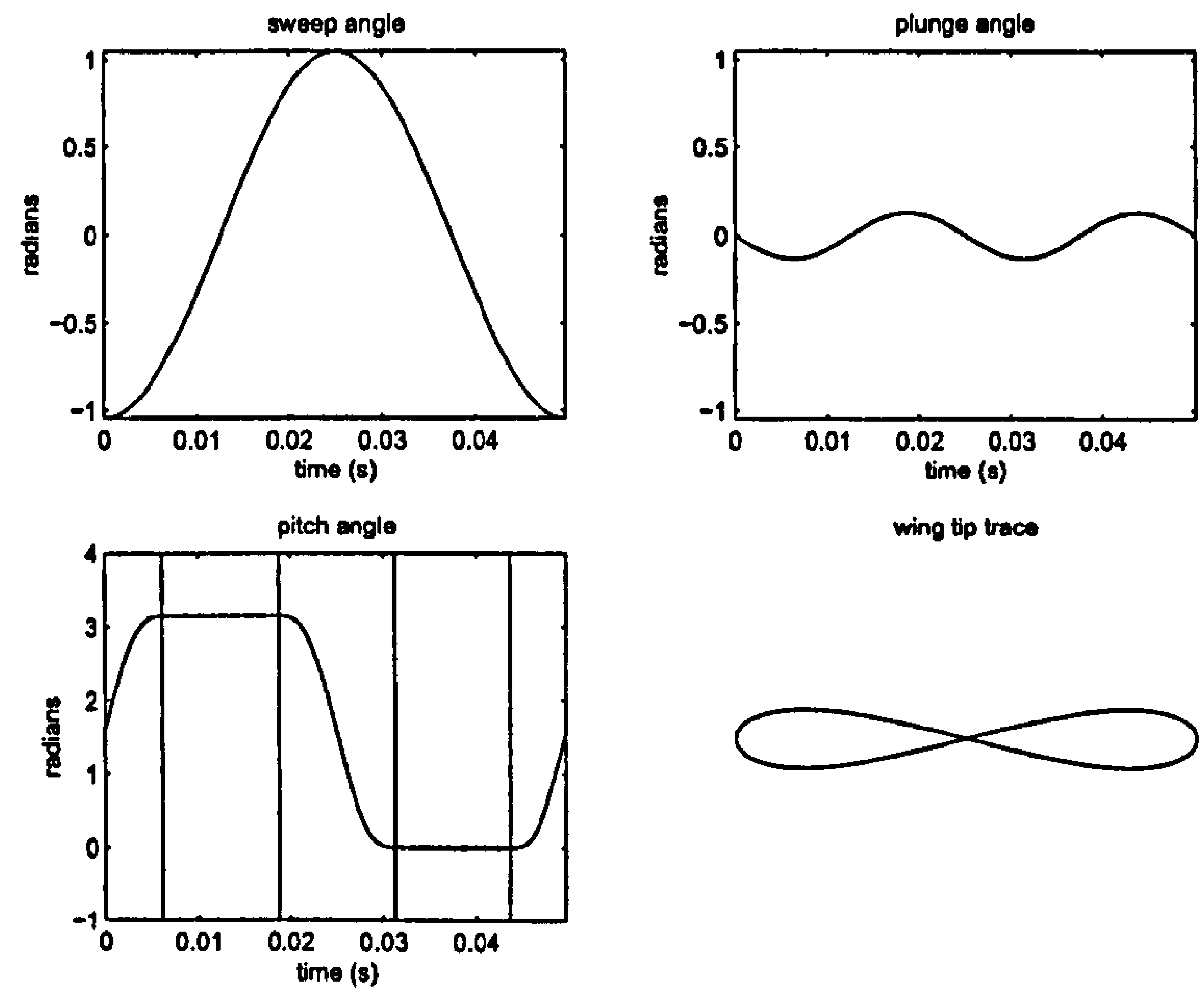


Figure 70: FMAV 50/2 wing kinematics: the first stroke starts forwards (negative sweep angle θ) with the wing vertical. The vertical lines in the pitch angle plot are the delimiters between rotational and translational motion. The tip trace is in the θ, ψ spherical coordinate system. Angles are given in radians.

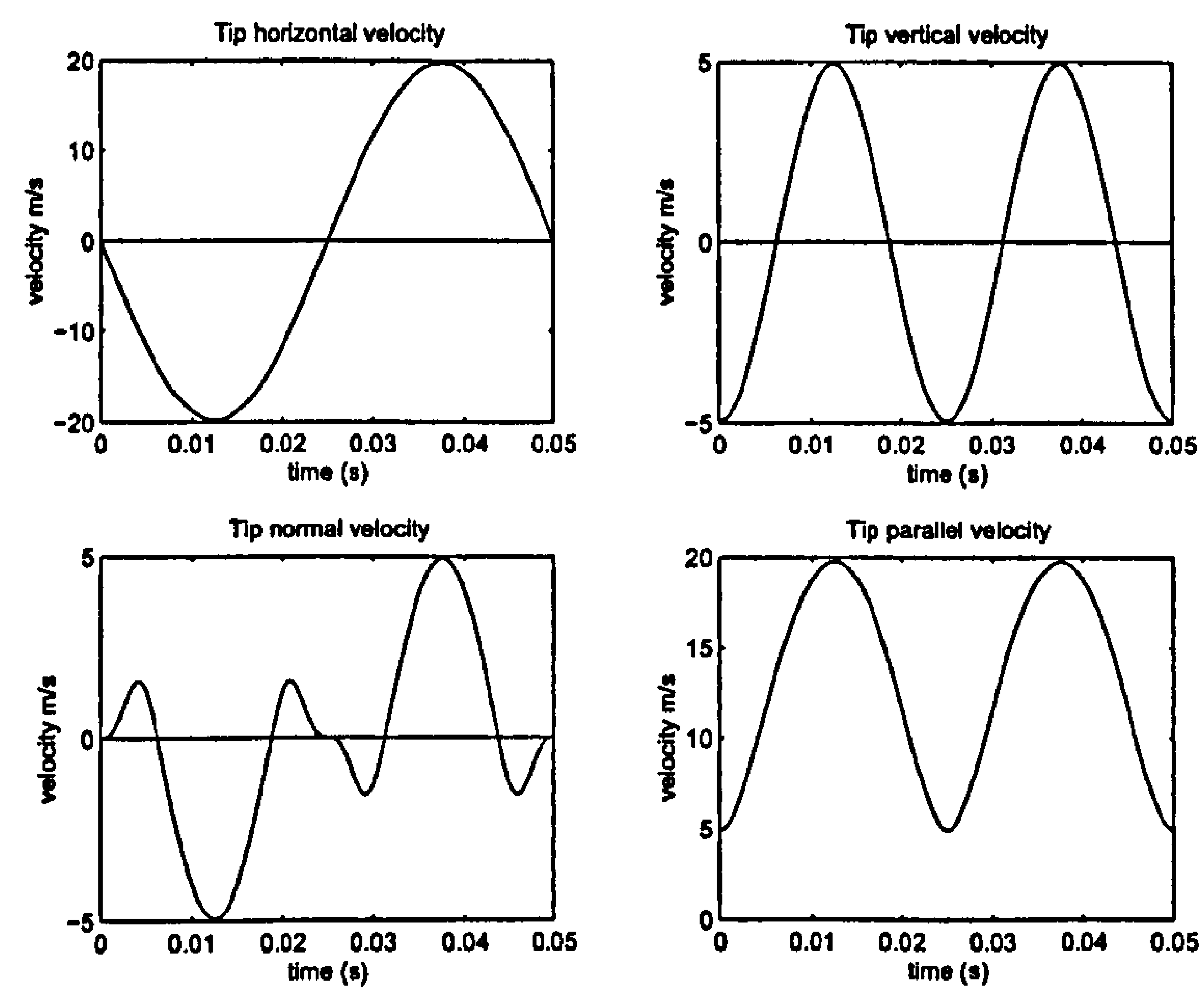


Figure 71: FMAV 50/2 wingtip velocities: the velocities are of the fluid relative to the wing.

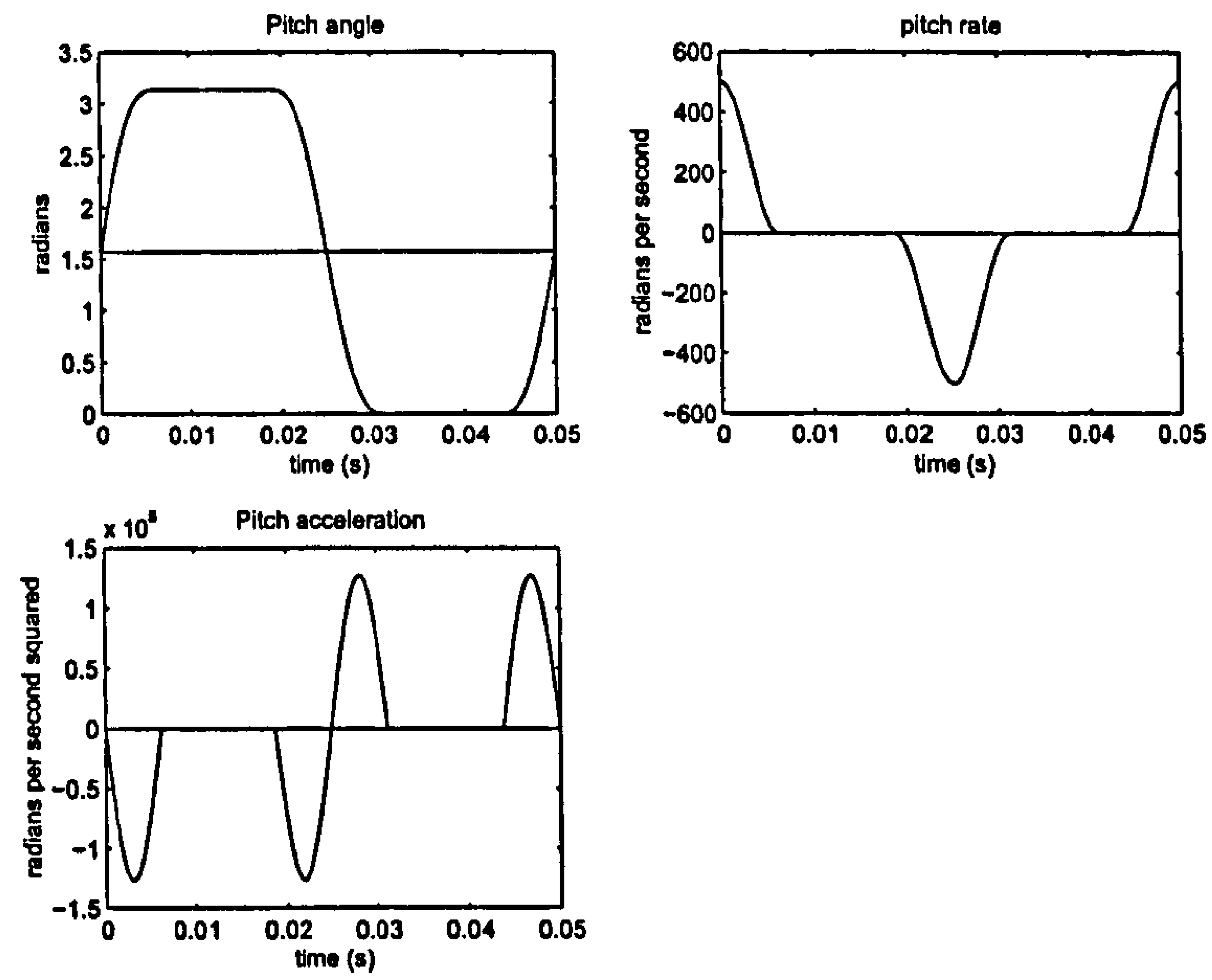


Figure 72: FMAV 50/2 wing pitching: note that the velocity asymptotes gradually to 0. The horizontal line on the pitch angle figure represents the vertical ($\pi/2$).

- FIRSTSTEP = wrap. Because the data represent a closed cycle, we wrap the first value of C_L
- NWAK = 4. The wake is formed using four full cycles.
- WAKEMETHOD = full wake. We form the full wake based on the cycles given by NWAK at once, rather than growing it from the start.
- USEPOLHAMUS = yes. C_L for the Wagner and Küssner effects are the effective C_L, as modified by the Polhamus correction.
- TAILFLAG = 1. Reversal times are based on the trailing edge position.

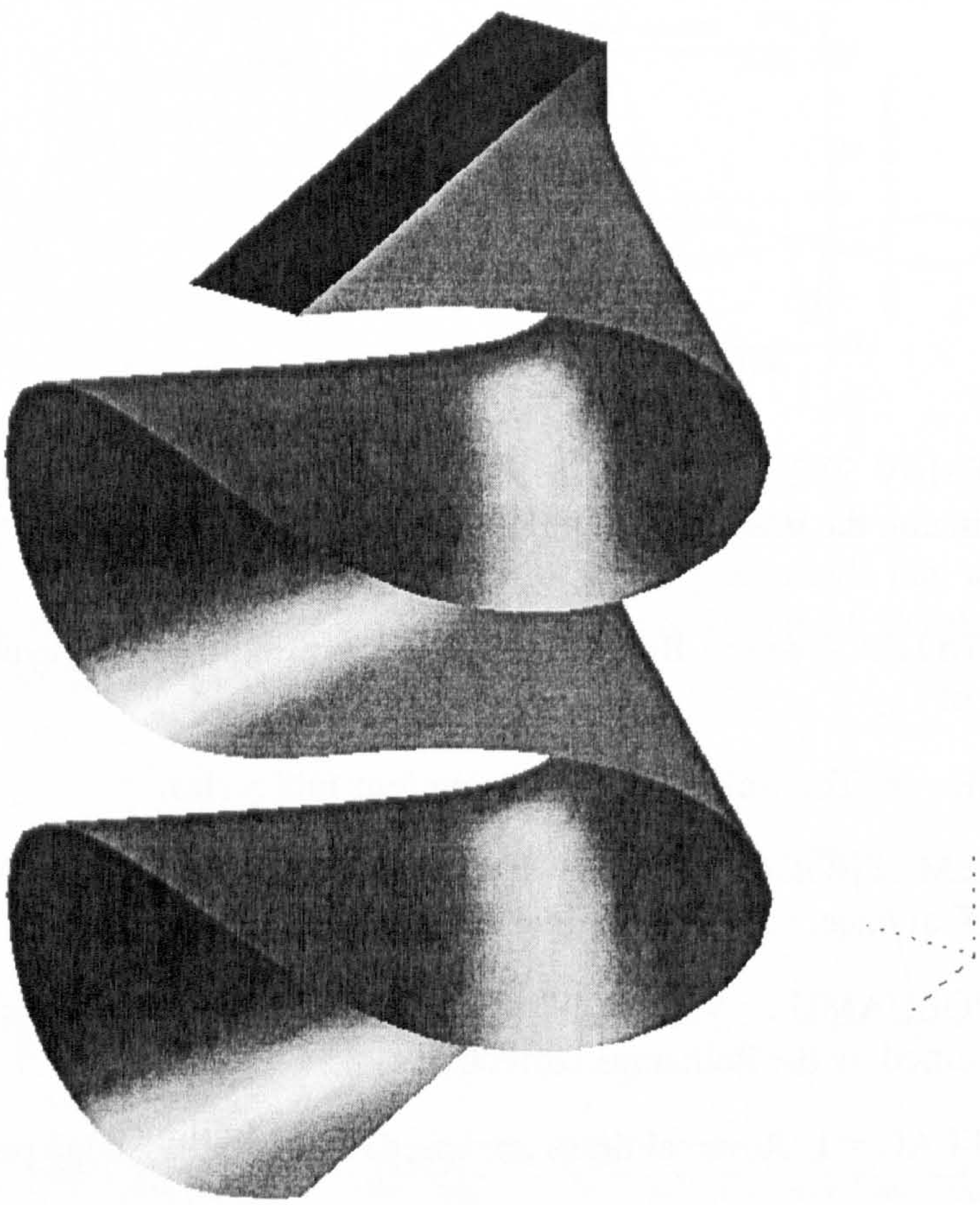


Figure 73: Sample $3D$ Lissajous wake surface. Compare with Figure 31. The dark rectangle at the top is a thin, flat rectangular wing, undergoing a figure-of-eight motion similar to the FMAV 50/2 kinematics given. The hinge of the wing is at the upper right corner in the figure. The $3D$ wake is the surface swept by the trailing edge, convected downwards at constant velocity.

17 Results for FMAV 50/2

This study has been included for the sake of highlighting some effects of our model that are not apparent from the Robofly dataset.

Considering the horizontal quasi-steady "drag" force in Figure 75, it can be seen that during the translation it is actually in the direction of motion, due to the wing plunging downwards. This is clearly an unrealistic result, but once the Polhamus correction has been added, as shown in Figures 76 and 77 the horizontal force ceases to be in the direction of motion. This is because the forwards horizontal force will be a suction force that the Polhamus effect rotates to become vertical.

The vertical added mass force is shown in Figure 79. This is shown to illustrate that the mean added mass force for a cycle is in fact 0. For the Robofly dataset, the mean added mass force came to about 1-2% of the root-mean-square value, because of the use of sampled data.

Finally, by setting *firststep='wrap'* the impulsive starting effect of the primary wake have been eliminated, assuming that the current stroke is just one in a long series of identical strokes. Therefore, the primary wake effect in Figure 78 is also symmetrical between strokes.

A sample wake shape for this type of motion can be seen in Figure 73.

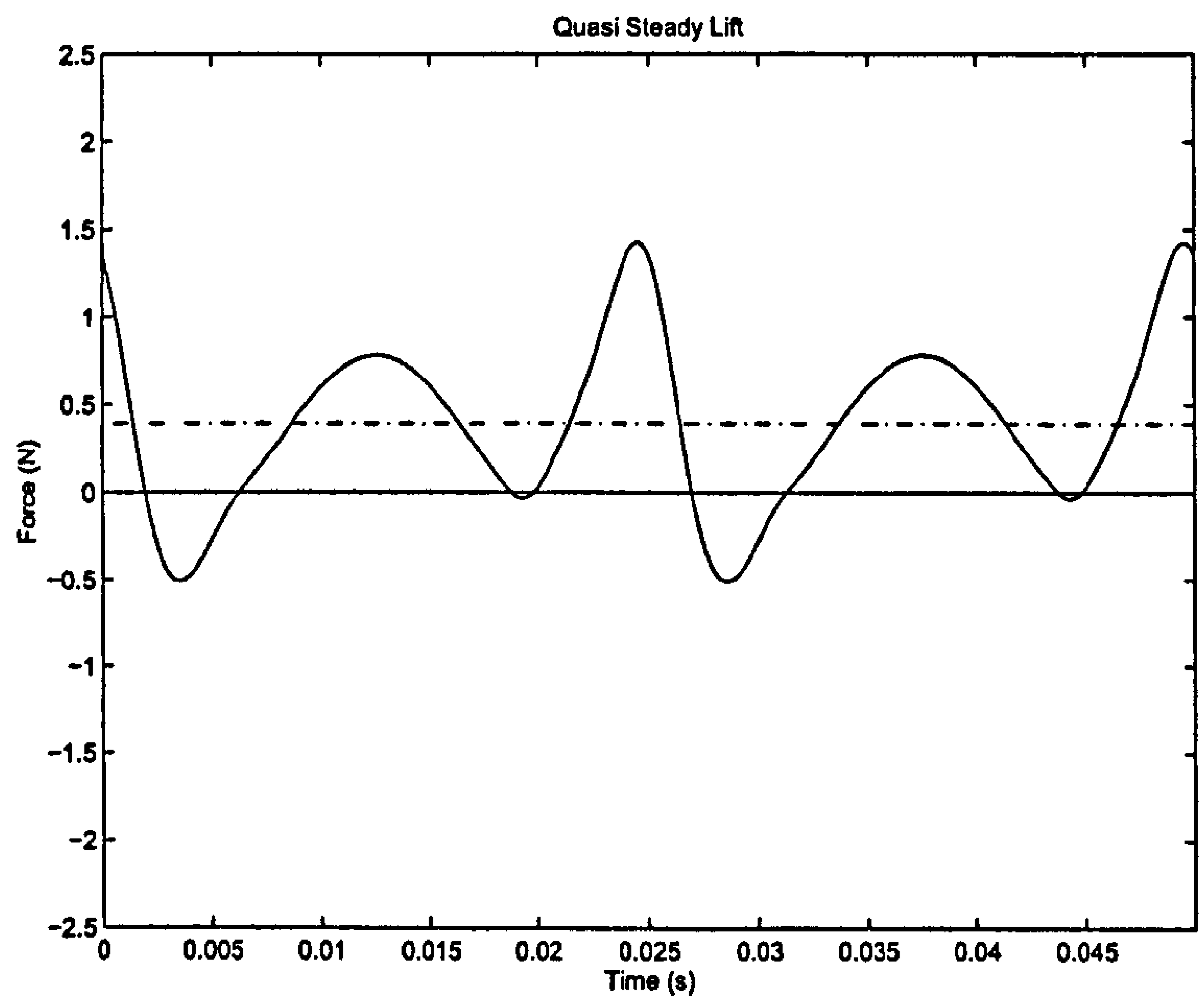


Figure 74: FMAV 50/2 quasi-steady lift (F_{VQW}) - solid line. Chain line is average predicted force.

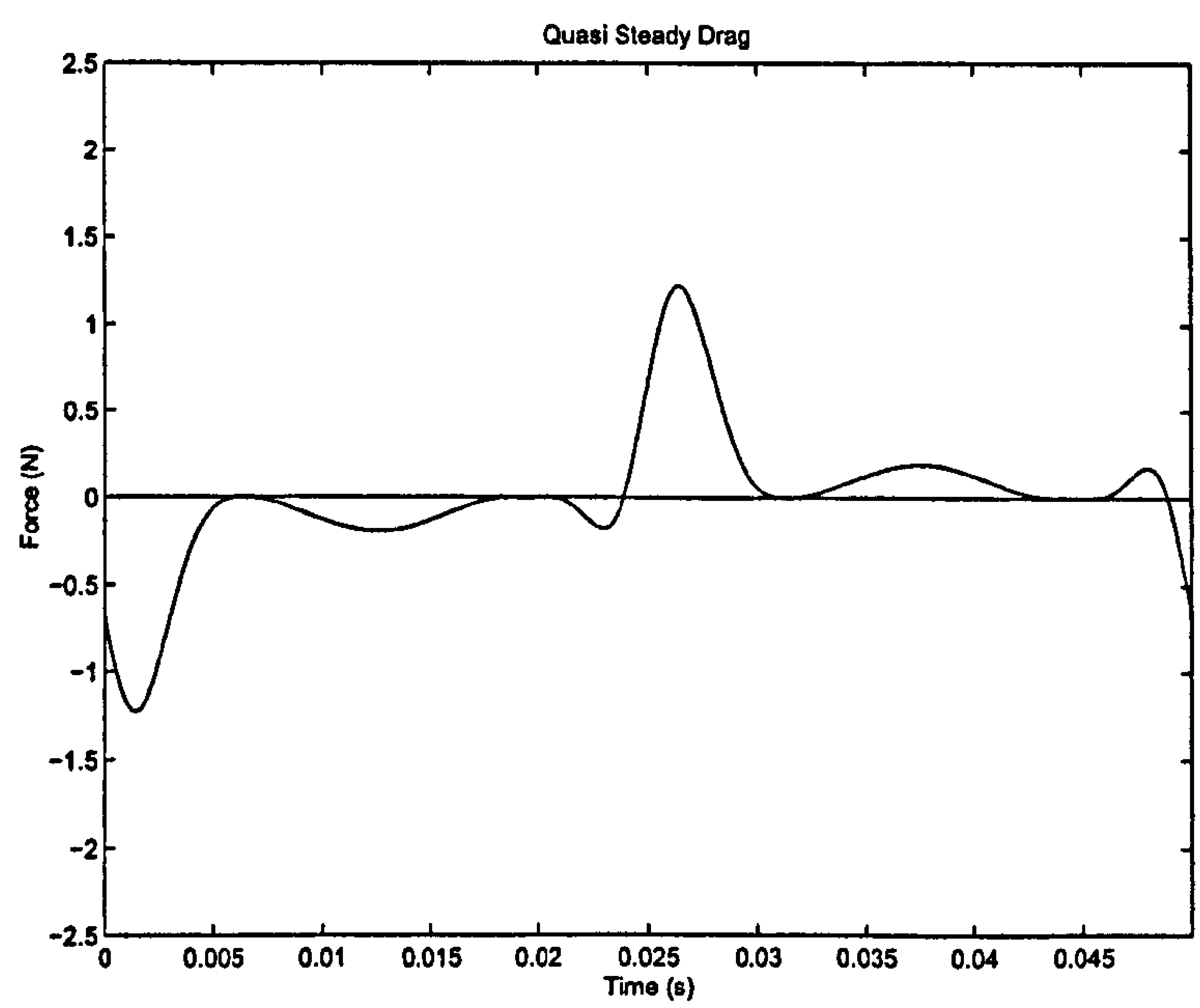


Figure 75: FMAV 50/2 quasi-steady "drag" . (F_{HQW}) - solid line. Average predicted force is approximately 0.

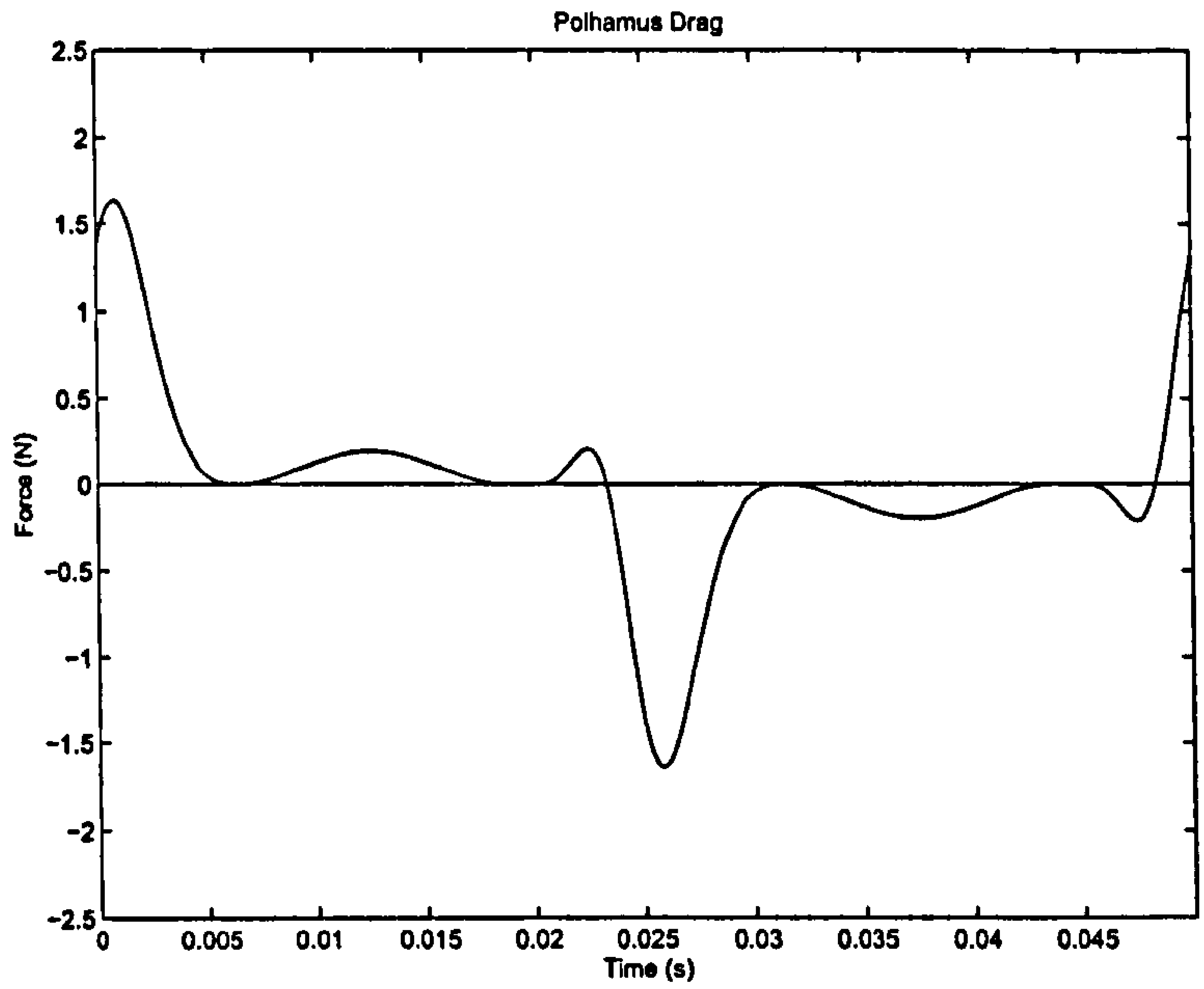


Figure 76: FMAV 50/2 Polhamus correction to horizontal force. Average predicted force is approximately 0.

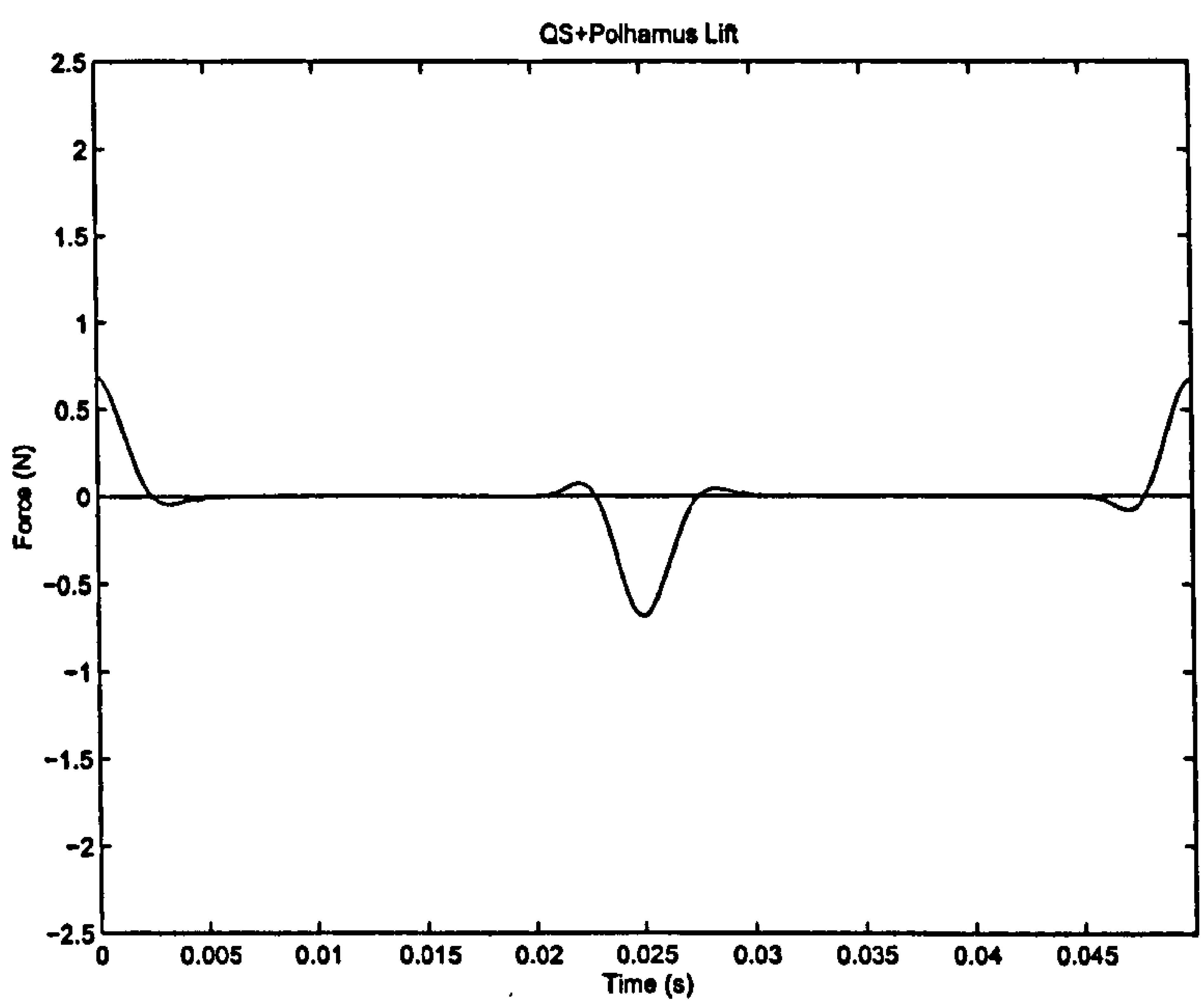


Figure 77: FMAV 50/2 horizontal force, corrected for Polhamus effect. Average predicted force is approximately 0.

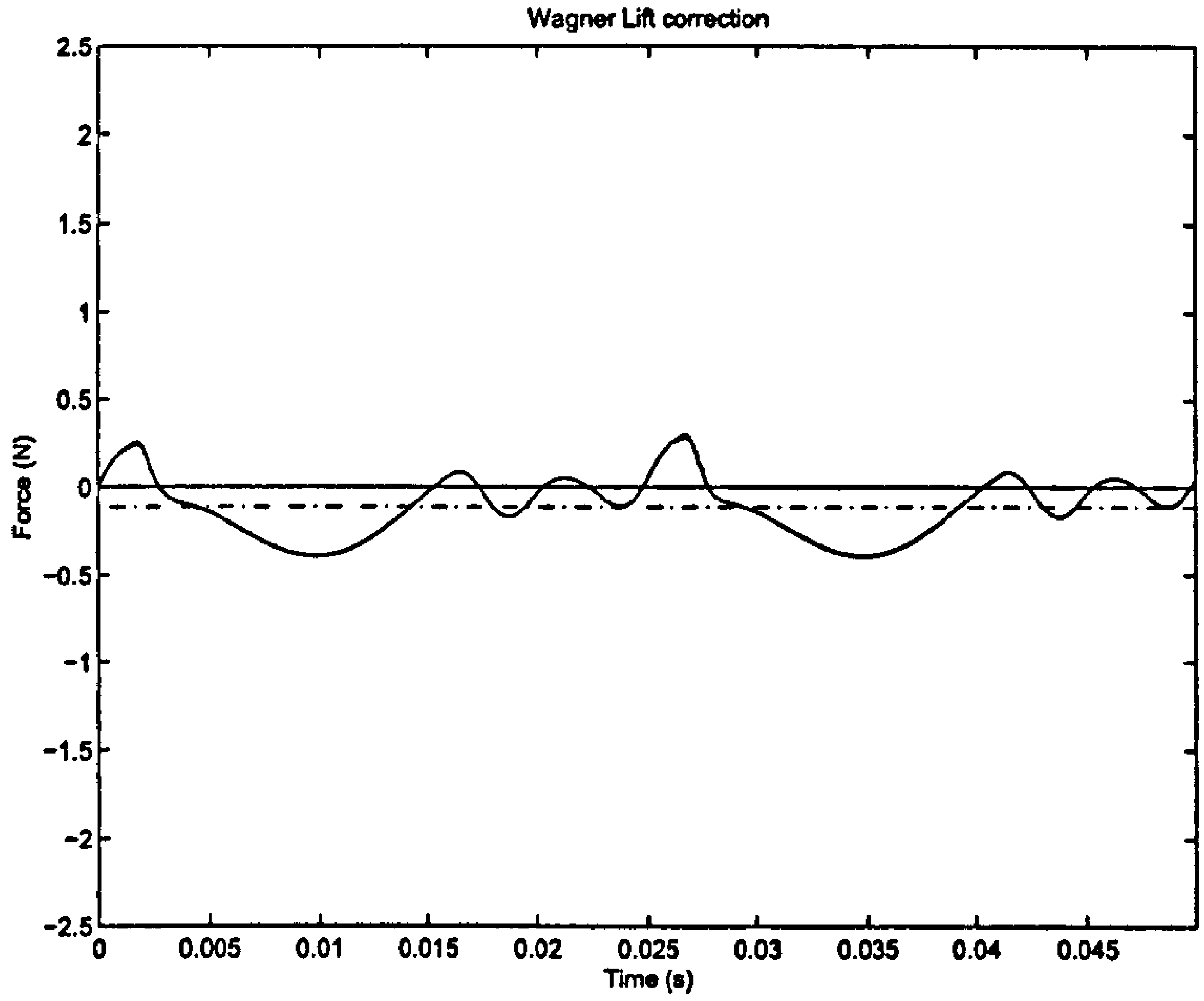


Figure 78: FMAV 50/2 primary (Wagner) wake lift correction (solid line). Chain line is average predicted force.

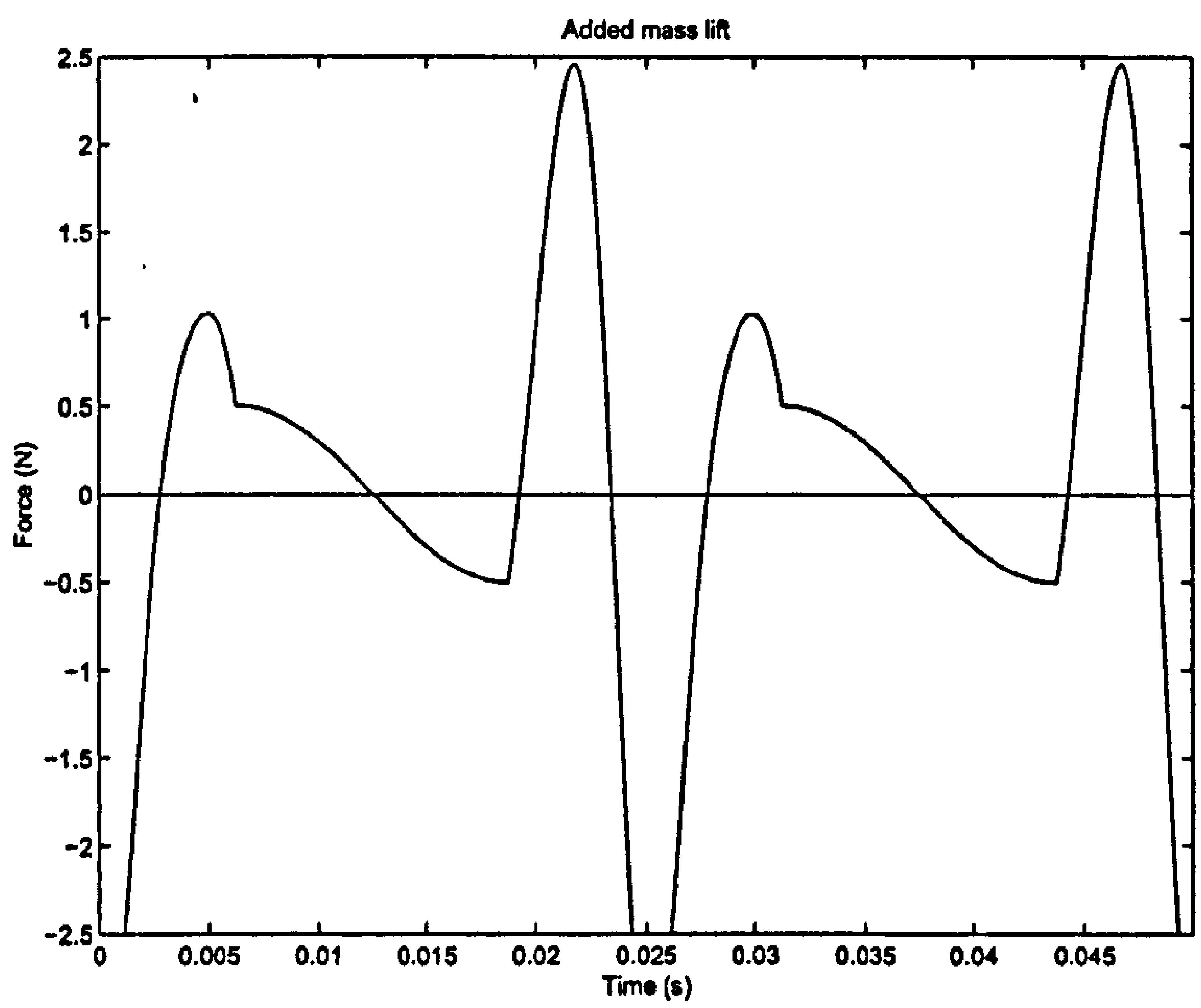


Figure 79: FMAV 50/2 added mass lift. Average predicted force is approximately 0.

18 Discussion

18.1 Discussion of results

18.1.1 Result overview

The Robofly data for both lift and drag show a similar form - a sharp trough near the reversal point, accompanied by peaks immediately before and after reversal. Between these two peaks (in the midstroke) the forces are lower, but increasing gradually towards the second peak.

For the predicted forces, both the lift and drag show a similar form - the Polhamus corrected quasi-steady force over-predicts the measured force by a factor of 2, and has no first peak after reversal. The wake effect is to reduce these forces at midstroke. Finally, the added mass contribution has very little effect at midstroke, but causes a sharp reduction in forces at reversal, and an increase immediately after, reducing the predicted value at reversal to match the trough in the measured data, and introducing the first peak just after reversal, again to match the measured data.

These observations cannot be assumed generally to apply to other kinematics - for example the FMAV 50/2 kinematics of Section 17 show a marked force peak at midstroke.

18.1.2 Radial force distribution

Consider the radial distribution of the lift forces in Figures 48 to 57. The radial position of the peaks is most apparent from the radial distribution plot of peak force in Figure 58. As mentioned in Section 15 the radial position with the greatest quasi-steady force per unit span is slightly outboard of the radius where the semichord is greatest. This is because translational velocities increase towards the tip, so the force per unit surface area increases further from the root. However, beyond the radius where peak chord occurs, the wing starts to taper to a point, reducing the surface area per span. Therefore, the radial position with the greatest quasi-steady force per unit span will be where the product r^2b is greatest. In Figures 49 and 58 it can be seen that this occurs at about 80% of tip radius.

This radial position will obviously be different for other wing shapes, but it will generally hold that the quasi-steady force per metre span is highest slightly outboard of the maximum chord, for any reasonably smooth chord distribution. The radial distribution of the other four force components are broadly similar, with some notable variation: the magnitude of the Polhamus component is directly related to the suction part of the quasi-steady force. Therefore, unsurprisingly, the radial distribution of the Polhamus force component is almost identical to that of the quasi-steady force. For the Wagner and Küssner components, it was expected that the radial distribution would be fuller towards the root. Although the shed vorticity increases towards the tip, so does the distance travelled in semichords, which governs how quickly these effects decay. The distance travelled in semichords increases towards the tip partly because of the greater distance travelled, but also because the semichord decreases towards the tip. Defining the maximum force radius ($\hat{r}$, the radial position where the forces have their greatest magnitude), it can be seen that the magnitudes of the two wake components do, indeed, drop off much faster tipwards of $\hat{r}$. However, surprisingly, values of $\hat{r}$ for the the wake force components are further tipwards than for the quasi-steady case, and the radial distribution of the wake forces rootwards of $\hat{r}$ is not notably fuller than the quasi-steady case.

18.1.3 Rigid versus flexible wing surface

The most rapid changes in force, and the peaks and troughs of the force occur near the reversal points, because of the high rotation velocities at those points. It is postulated that this is an effect of the wings being rigid, and this concentration will be less pronounced for wings with flexible surfaces. Because the wings are rigid, the normal velocities due to the rotation at reversal are high. This affects all the force components. Allowing the surface to deform introduces elasticity into the response of the wing to the kinematics, attenuating the effect of sharp rotation.

This point is important because an eventual physical embodiment of an FMAV will almost certainly need a flexible wing surface, partly for structural reasons and partly to reduce the peak loads during rotation.

18.1.4 Viscosity

The potential force modelling used in this thesis required the omission of viscous forces, especially in the form of skin friction and base (pressure) drag. There is no simple way of introducing corrections for these, apart from using empirical corrections.

18.2 Evaluation of results

This section deals with the accuracy and utility of the model. It is mainly restricted to the results for the Robofly dataset, since this is the only set with measured data to compare with. It is concluded that the added mass model is not very good, as it over-predicts the effect of added mass due to ignoring the effect of the primary wake on the added mass. The lift results are acceptable as a first-order model, in that they capture the general shape and overall scale of the lift force. Especially gratifying is the way they accurately capture the loss of lift due to the impulsive start. The drag results, however, are much less good, primarily because the drag effect of the primary wake on the wing is not modelled. Using the Polhamus correction to force coefficient for the purpose of wake effect seems especially promising, but will require more validation, *after* the added mass model has been refined. A simple model to correct for the primary wake effect on drag has been proposed in Section 15.2. Although this is too weak theoretically to use when predicting forces, it is used to support the postulation that omitting primary wake effects is probably the most important source of error in the drag force prediction.

Moment data were not available for the Robofly experiment. This is unfortunate, as it makes it impossible to validate the expressions for the moments. The added mass moment is especially a concern, as the added mass forces were modelled without the effect of the wake.

Similarly, the effect of omitting Polhamus corrections from the pitching moment cannot be evaluated.

Nonetheless, it is gratifying that a purely inviscid model has managed to get as close at it did to the actual results, which are for a very viscous and unsteady flight regime. The model underpredicted the average lift by only 9%. Note also that the expected operating regime of the FMAV is considerably less viscous than that of Dickinson's experiment. Dickinson had a Reynolds number of order 10^2, while for the FMAV kinematics of Section 16 the Reynolds number is of the order 10^5. This is calculated using the same expression as Dickinson[4]. The result of this higher Reynolds number should be to reduce the effect of viscosity, and hopefully make the model more accurate when modelling the aerodynamics of the FMAV.

[4]Using the expression for Reynolds number on page 2608 of [2], with our kinematic data from Section 16, and the kinematic viscosity equal to $1.5 \times 10^{-5} m^2/s$. Note that these values are not directly comparable with standard translational aerofoil Reynolds numbers, since they use peak values of velocity and semichord.

Part V
Conclusion and further work

In this part, the assumptions used when making the model are summarised in Section 19.1, and some conclusions drawn on the usability of the model and the code in Section 19.2. Ideas for future refinement are given in Section 20.

19 Conclusions

First, the assumptions used while creating the model will be listed.

19.1 Assumptions

- The wing root is stationary.
- The wing is thin and flat.
- The hinge point is at the same point on the chord for all spanwise sections, when wing shape factors are used.
- The body does not affect the airflow, and is ignored.
- The far-field free stream is stationary.
- The flow is entirely inviscid.
- The flow separates sharply from the leading edge, causing total loss of leading edge suction.
- The flow always reattaches, and forms a stable leading edge vortex.
- The effect of the LEV is to rotate the leading edge suction force by 90°, to become a normal force component.
- The direction of the above rotation is in the direction of the normal velocity at the leading edge.
- The LEV dissipates immediately when shed.
- The flow leaves the trailing edge smoothly, satisfying the Kutta-Joukowski condition.
- The wake is treated as a thin, globally stationary filament of vorticity.
- The wake does not decay or dissipate.
- The wake is split into single-stroke elements, each of which is assumed to be a straight line.
- The wake moves under constant downwash velocity u_i, without deforming under its own induced velocity.
- The above movement is discretised into a set of steps at each reversal.
- Each wake segment is assumed to be behind the wing until reversal, where all previous wakes jump downwards by a distance based on the average predicted downwash velocity.

19.2 Theory conclusions

A model has been developed for calculating highly unsteady lift of insect-like flapping wings, and embodied in **MATLAB** code. This model is simplified and modular, for the purpose of giving better insight into the various effects that act on the wing. However, this has come at the expense of considerable simplification, in order to enable the use of known solutions to standard unsteady problems. The greatest limitations of the model are: 1) no modelling of viscous forces (this was neccesary to obtain an analytical potential model), and 2) the effect of the wake on added mass is incomplete - although the Küssner function includes the effect of added mass, the Wagner function does not.

The model was tested on two datasets: one for a prescribed geometry and kinematics of a proposed FMAV wing design, the other data from an experiment on Dickinson's Robofly. The second dataset was used for model validation, from which it was concluded that the average circulatory lift predicted was within 9% of the measured. However, the non-circulatory lift (added mass effect) was a poor fit - so although added mass does not contribute a net force over a cycle, some features of the shape of the lift trace are lost, and the peak loads are being over-predicted. For drag, both the circulatory and non-circulatory component showed poor correlation. This is partly because of the above mentioned problems with added mass, and the fact that the wake model does not model primary wake drag. However, it is also suspected to be mainly due to the fact that viscous drag is omitted entirely - consider for example the odd result of Figure 75, where the drag force is in the direction of the motion.

The main conclusion is that this model is a considerable simplification. This was done for the purpose of making it possible to embody the model in non-iterative code, and to give quantitative insight into the meaning of the results. These assumptions mean the model does not predict forces accurately enough for peak loading or flight dynamics modelling. Hopefully, further refinement will allow this. However, the time evolution of lift has been captured well and it has been shown that only the added mass component is not modelled with the required accuracy. This indicates the soundness of the approach and shows the advantage of modularity.

19.3 Code conclusions

The code runs to about 160kb of **MATLAB** code, with a runtime less than five minutes on a reasonably modern system. Unlike standard CFD code, it does not rely on successive approximation by iterating the code. This means runtime is much lower, and allows us to use some "sloppy" code practice, that gives better code legibility at the expense of runtime.

Considerable effort has been expended to make the code proof against data and method legacy, and to leave it open to further refinement by keeping it strictly modular: one module per aerodynamic effect, calculation or dataset.

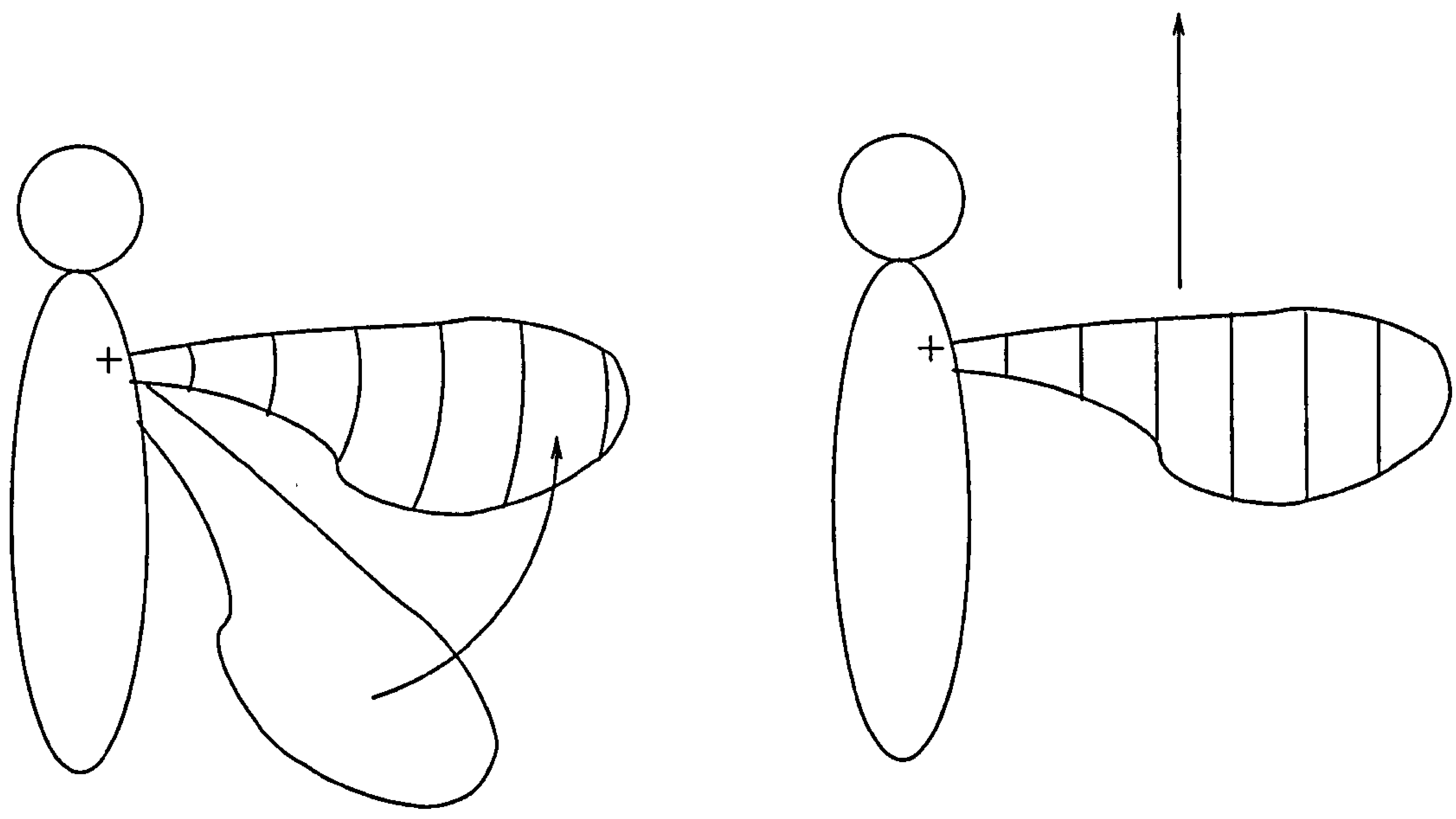


Figure 80: Radial chord example.

20 Further work

20.1 Theory refinement

20.1.1 Radial chord

Consider a wing sweeping horizontally in still air, at zero angle of attack, so the wing is also horizontal. As can be seen from Figure 80, the effective chord lines in this case are arcs centered on the hinge - these are points where the velocity due to the sweeping motion are the same. Similarly, consider the same wing sweeping horizontally at 90° angle of attack, so the wing is vertical - the effective chord lines are now straight.

From the above, it can be seen that the effective chord of a rotating wing is not straight, but depends on the inflow angle of the wing. It is proposed that the effective chord distribution can be expressed as the summation of two "pseudochords" - one purely radial, one purely linear.

Development of this theory had to be abandoned because of time constraints. Briefly, it was assumed that the forces would scale with $\bar{u}_{\mathrm{T}}^2$, of which $\cos(\alpha^2)$ would be along the wing (and thus radial chord), and $\sin(\alpha^2)$ would be across the wing (and thus linear chord). An analytical issue arose because the velocity due to wing pitching is always based on linear chord, so the radial chord calculation would have to be expressed as a mixture of radial and linear chord elements. Although this is not insurmountable, other parts of the model were felt to warrant more attention.

20.1.2 Wake shape

The wake shape used in this model is obviously unrealistic, with large, discrete jumps in location, causing it to be discontinuous. Although this was initially chosen in order to attempt to form a Loewy-like analytical summation, this has not worked. One positive result of this is that more complex wake shapes can be used, since summation over the wake is performed directly, without further simplifying assumptions made by Loewy. This author did experiment with wake shapes based on constant downwards velocity u_i, trying to model the entirety of the wake as a Küssner type effect. The drawback of this was that the Küssner effect kicks in slowly, whereas the Wagner effect starts immediately - thus, the effects of impulsive starts are being under-predicted. If a continuous wake filament is used, more thought will have to go into how the Wagner and Küssner functions can be combined. An example of this is the Miles gust model, which is described in [3].

20.1.3 Added mass and the wake

As mentioned, a flaw in the model is that the effect of the wake on the added mass is not modelled. Doing this precisely would require an iterative CFD model. The same holds for the effect of the wake on quasi-steady forces. Similarly to how the simplified case of Wagner was used for the quasi-steady forces, there may be some mileage in deriving similar expressions for the wake effect on added mass for simple cases. Note, however, that while the Wagner function reduces to a function of C_L and distance travelled only, the equivalent expressions for primary wake effect on added mass will require the entire kinematics of the wing, and the distance travelled.

The most immediate solution to this is from the Theodorsen function [28], where he treated the bound Kutta-Joukowski vorticity and wake vorticity as a single filament, and calculated the entirety of the forces caused by it. The Theodorsen function cannot be used in unmodified form, however, as he makes assumptions about constant forward velocity and cyclic pitching, in order to use the reduced frequency parameter, to reduce the solution to a single, analytical expression. However, the original integrals of Theodorsen can be used, and integrated numerically along the primary wake.

20.1.4 Wing shape parameters

The use of wing shape parameters to calculate the forces on the entire wing, has been demonstrated in Section 8.8. This method is appealing in that it allows greater insight into the effect of the wing shape on the various aerodynamic components, and allows fast and accurate calculation of wing forces without relying on numerical integration. However, the method used assumes that the hinge location a is constant for the entire span. This can be generalised to a varying hinge location by creating additional wing shape parameters that include the variation of a along the span. This is a trivial exercise in mathematics, but does require that a can be expressed analytically as a function of r.

20.2 Code refinement

The code, as it stands, is not optimised for speed. This has been commented on in Section 13. When the model has reached a greater stage of refinement, it may be worth speeding up the code for the sake of being able to use it in automated iterative refinement of wing kinematics and shape. For now, however, it is felt that ease of development and bug-tracking outweighs runtime.

Part VI
Appendices

A Mathematical results

A.1 Identities

A.1.1 Differential identities

$$
\begin{aligned}
f(g)' &= f'(g)\,g' && (189)\\
(fg)' &= f'(g) + g'(f) && (190)\\
\frac{df(x,y,z,...)}{dx} &= \frac{\partial x}{\partial x}\frac{\partial f}{\partial x} + \frac{\partial y}{\partial x}\frac{\partial f}{\partial y} + \frac{\partial z}{\partial x}\frac{\partial f}{\partial z} + ... \quad \text{chain rule} && (191)\\
\frac{d}{dx}f &= \left(\frac{d}{df}x\right)^{-1} \quad \text{not for partial derivatives.} && (192)
\end{aligned}
$$

A.1.2 Trigonometric Identities

$$
\begin{aligned}
SC &= \frac{1}{2}S_2 && (193)\\
C^2 &= \frac{1}{2}(1 + C_2) && (194)
\end{aligned}
$$

A.1.3 Coordinate transformations

$$
\begin{aligned}
bS_\theta &= b\sqrt{1 - x'^2} && (195)\\
x &= C_\theta \quad \text{Assumes x normalised} && (196)\\
\frac{dx}{d\theta} &= -S_\theta && (197)\\
\frac{ds}{d\theta} &= 2\pi r && (198)\\
& && (199)
\end{aligned}
$$

A.1.4 Identities involving Q

Note that the following uses the identity $Q = \sqrt{1 - x^2}$. All can be found in Gradshteyn and Ryzhik [50], abbreviated to GR in the following. The right arrow symbol $\rightarrow$ is followed by the reference to where the identity was obtained from. The left arrow symbol $\leftarrow$ is followed by necessary conditions for the identity to be true.

$$
\int \frac{dx}{Q} = \frac{-1}{1}\text{asin}\left(\frac{-2x}{\sqrt{4}}\right) \quad \rightarrow \text{GR2.261 P99} \ \leftarrow c, D < 0 \qquad (200)
$$

$$\begin{aligned}
&= -\text{asin}(-x) && \rightarrow 200 && (201)\\
\int_{-1}^{1}\frac{dx}{Q} &= -\left[\text{asin}(-x)\right]_{-1}^{1} \\
&= -[(-\pi/2)-(\pi/2)] \\
&= \pi && && (202)\\
& && && (203)
\end{aligned}$$

$$\begin{aligned}
\int Q\,dx &= \frac{xQ}{2}+\frac{1}{2}\int_{-1}^{1}\frac{dx}{Q} && \rightarrow \text{GR2.262.1 P102} && (204)\\
\int_{-1}^{1} Q\,dx &= \left[\frac{xQ}{2}\right]_{-1}^{1}+\frac{1}{2}\pi && \rightarrow 202 \\
&= 0+\frac{\pi}{2} && && (205)
\end{aligned}$$

$$\begin{aligned}
\int\frac{x\,dx}{Q} &= \frac{Q}{c}-\frac{b}{2c}\int\frac{dx}{Q} && \rightarrow \text{GR2.264.2 p100} && (206)\\
\int\frac{x\,dx}{Q} &= -Q-0 && && (207)\\
\int_{-1}^{1} &= -\left[Q\right]_{-1}^{1} \\
&= 0 && && (208)
\end{aligned}$$

$$\begin{aligned}
\int\frac{x^2\,dx}{Q} &= \left(\frac{x}{2c}-\frac{3b}{4c^2}\right)Q+\left(\frac{3b^2}{8c^2}-\frac{a}{2c}\right)\int\frac{dx}{Q} && \rightarrow \text{GR2.264.3 p100}\\
\int\frac{x^2\,dx}{Q} &= \frac{x}{-2}Q+(-\frac{1}{-2})\int\frac{dx}{Q} \\
&= -\frac{1}{2}(xQ)+\frac{1}{2}\int\frac{dx}{Q} \\
&= -\frac{1}{2}\left[xQ\right]+\frac{1}{2}\int\frac{dx}{Q} && && (209)\\
\int_{-1}^{1}\frac{x^2\,dx}{Q} &= -\frac{1}{2}\left[xQ\right]_{-1}^{1}+\frac{1}{2}[\pi] && \rightarrow 202 && (210)\\
&= 0+\frac{\pi}{2} && && (211)
\end{aligned}$$

$$\int x^n\text{asin}(x) = \frac{x^{n+1}}{n+1}\text{asin}(x)-\frac{1}{n+1}\int\frac{x^{n+1}}{Q} \quad \rightarrow \text{GR2.831 p253} \quad (212)$$

$$
\begin{aligned}
\int \operatorname{asin}(x) &= x\operatorname{asin}(x) - \int \frac{x}{Q} \qquad \rightarrow 212 \\
\int_{-1}^{1} \operatorname{asin}(x) &= \operatorname{asin}(1) + \operatorname{asin}(-1) - \int_{-1}^{1} \frac{x}{Q} \\
&= 0
\end{aligned} \tag{213}
$$

$$
\begin{aligned}
\int x \operatorname{asin}(x) &= \frac{x^2}{2}\operatorname{asin}(x) - \frac{1}{2}\int \frac{x^2}{Q} \qquad \rightarrow 212 \\
\int_{-1}^{1} x \operatorname{asin}(x) &= \frac{1}{2}\operatorname{asin}(1) - \frac{1}{2}\operatorname{asin}(-1) - \frac{1}{2}\int \frac{x^2}{Q} \\
\int_{-1}^{1} x \operatorname{asin}(x) &= \frac{1}{2}\frac{\pi}{2} - \frac{1}{2}\frac{-\pi}{2} - \frac{1}{2}\int \frac{x^2}{Q} \\
\int_{-1}^{1} x \operatorname{asin}(x) &= \frac{\pi}{2} - \frac{1}{2}\int \frac{x^2}{Q} \\
&= \frac{\pi}{2} - \frac{1}{2}\frac{\pi}{2} \rightarrow 211 \\
&= \frac{\pi}{4}
\end{aligned} \tag{214}
$$

A.2 Other proofs

A.2.1 R.M.S. velocity

When forming the denominator for lift coefficient $\frac{1}{2}\rho\bar{u}_{\mathrm{T}}^2 A$ we use the total velocity $\bar{u}_{\mathrm{T}}$, which varies along the wing when there is a pitching velocity. Since this is an expression for the kinetic energy of the flow, we use the mean value of $\bar{u}_{\mathrm{T}}^2$, integrated along the chord, which becomes:

$$
\bar{u}_{\mathrm{T}}^2 = \bar{u}_{\mathrm{T}}^2 + \dot{\beta}^2 b^2(1/3 + a^2) - 2u_N\dot{\beta}ba \tag{215}
$$

This velocity is only zero at complete standstill, i.e. $\bar{u}_{\mathrm{T}} = \dot{\beta} = 0$.

At any given position on the wing, the local velocity of the (stationary) air relative to the wing is:

$$
\begin{aligned}
\bar{u_{\mathrm{T}}}^2 &= u_{pe}^2 + u_{ne}^2 \\
&= u_p^2 + \left(u_n + \dot{\beta}b(\xi - a)\right)^2 \\
&= u_p^2 + u_n^2 + \dot{\beta}^2 b^2(\xi - a)^2 + 2u_n\dot{\beta}b(x - a) \\
&= \bar{u}_{\mathrm{T}}^2 + \dot{\beta}^2 b^2(\xi - a)^2 + 2u_n\dot{\beta}b(x - a)
\end{aligned} \tag{216}
$$

The mean of this is found from the integral along ξ:

$$\begin{aligned}
\bar{u_{\mathrm{T}}}^2 &= \frac{1}{2}\int_{-1}^{+1} \bar{u}_{\mathrm{T}E}^2 \, d\xi \\
&= \frac{1}{2}\left[u_p^2 x + u_n^2 x + \left(\frac{1}{3}x^3 + a^2 x - a x^2\right)\dot{\beta}^2 b^2 + 2u_n\dot{\beta}b\left(\frac{1}{2}x^2 - ax\right)\right]_{-1}^{+1} \\
&= \frac{1}{2}\left(2u_p^2 + 2u_n^2 + \left(\frac{2}{3} + 2a^2 - 0\right)\dot{\beta}^2 b^2 + 2u_n\dot{\beta}b(0 - 2a)\right) \\
&= \bar{u}_{\mathrm{T}}^2 + \dot{\beta}^2 b^2\left(\frac{1}{3} + a^2\right) - 2u_n\dot{\beta}ba
\end{aligned} \tag{217}$$

for the case where $a = -\frac{1}{2}$:

$$\bar{u_{\mathrm{T}}}^2 = \bar{u}_{\mathrm{T}}^2 + \frac{7}{12}\dot{\beta}^2 b^2 + u_n\dot{\beta}b \tag{218}$$

Prove that $\bar{u_{\mathrm{T}}}^2$ is only 0 when stationary, by finding conditions for $\bar{u_{\mathrm{T}}}^2 = 0$. Find the condition for $\dot{\beta}$ that gives $\bar{u_{\mathrm{T}}}^2 = 0$. Do this by finding the solution so $0 = A\dot{\beta}^2 + B\dot{\beta} + C$, where $A = \bar{u}_{\mathrm{T}}^2$, $B = -2u_n ba$ and $C = b^2(1/3 + a^2)$. This is a second order polynomial with solution:

$$\begin{aligned}
\dot{\beta}_0 &= \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \\
&= \frac{2u_n ba \pm \sqrt{4u_n^2 b^2 a^2 - 4(\bar{u}_{\mathrm{T}}^2 b^2 a^2 + \bar{u}_{\mathrm{T}}^2 b^2/3)}}{2\bar{u}_{\mathrm{T}}^2} \\
&= \frac{2u_n ba \pm 2b\sqrt{u_n^2 a^2 - \bar{u}_{\mathrm{T}}^2 a^2 - \bar{u}_{\mathrm{T}}^2/3}}{2\bar{u}_{\mathrm{T}}^2} \\
&= \frac{2u_n ba \pm 2b\sqrt{a^2(u_n^2 - \bar{u}_{\mathrm{T}}^2) - \bar{u}_{\mathrm{T}}^2/3}}{2\bar{u}_{\mathrm{T}}^2}
\end{aligned} \tag{219}$$

The only purely real solution to this is $\dot{\beta}_0 = 0$ when $\bar{u}_{\mathrm{T}} = 0$. All other conditions will cause the expression under the square root to be negative, so the result becomes complex. this is because $u_n^2 < \bar{u}_{\mathrm{T}}^2$ always, so $a^2(u_n^2 - \bar{u}_{\mathrm{T}}^2)$ is always negative or 0. The second part $-\bar{u}_{\mathrm{T}}^2/3$ is also always negative or 0.

A.2.2 Duhamel's integral

The following is a generalised theory for the response of a system to an input. In our case the system is the aerofoil in unsteady motion, with the in- and output being the wing kinematics and wing lift—but for this section we will deal with it as an abstract "system".

Consider the response $y(t)$ of the system, modelled by $\Phi(t)$, to a step change in input $\Delta x(t)$ that occurs at time 0:

$$y(t) = \Delta x(0)\,\Phi(t) \tag{220}$$

This assumes linearity, so the output scales with the input. From causality, $\Phi(<0) = 0$.

Assuming time invariance, the above can be generalized to an input at time t_1:

$$y(t) = \Delta x(t_1)\,\Phi(t-t_1). \tag{221}$$

Another effect of linearity is that the output due to several inputs can simply be superposed linearly:

$$\begin{aligned} y(t) &= \Delta x(t_1)\,\Phi(t-t_1) + \Delta x(t_2)\,\Phi(t-t_2) + ... && (222)\\ &= \sum_n x(t_n)\,\Phi(t-t_n) && (223) \end{aligned}$$

which can be refined to infinitely small timesteps, so that in the limit

$$y(t) = \int_{-\infty}^{t} \frac{dx(\sigma)}{d\sigma}\Phi(t-\sigma)\,d\sigma. \tag{224}$$

The variable σ represents the delays t_1, t_2, ..., and is of the same units and scope as t. This is sometime referred to as the *dummy* or *integration* variable.

Now assuming zero input until $t = 0$, we get

$$y(t) = \int_{0}^{t} \frac{dx(\sigma)}{d\sigma}\Phi(t-\sigma)\,d\sigma \tag{225}$$

However, in order to include the effect of an impulsive input at $t = 0$, we have to split this integral into two time regimes:

$$y(t) = \int_{0^-}^{0^+} \frac{dx(\sigma)}{d\sigma}\Phi(t-\sigma)\,d\sigma + \int_{0^+}^{t} \frac{dx(\sigma)}{d\sigma}\Phi(t-\sigma)\,d\sigma, \tag{226}$$

where 0^- and 0^+ are immediately before and after 0, respectively. The first term can then be integrated to give:

$$y(t) = x(0)\,\Phi(t) + \int_{0}^{t} \frac{dx(\sigma)}{d\sigma}\Phi(t-\sigma)\,d\sigma \tag{227}$$

This is Duhamel's equation. There is also a slightly generalized form for the case when the first input is not received at $t = 0$, but at $t = t_0$:

$$y(t) = x(t_0)\,\Phi(t-t_0) + \int_{t_0}^{t} \frac{dx(\sigma)}{d\sigma}\Phi(t-\sigma)\,d\sigma \tag{228}$$

The three assumptions for this equation are:

- Time invariance: the system always responds identically to an input, irrespective of when the input is received.
- Linearity: the output scales with the input, and the output of several different inputs can be superposed (summed).
- Causality: the system does not respond before the input is received.

Note that for most of the expressions where we use Duhamel the variable of integration is not t, but s, the semichord distance travelled; this doesn't change the expression, but care must be taken to avoid confusing this with the Laplacian variable $s = \sigma + i\omega$, which is commonly used in control theory.

B Code listing

The code is reproduced here only so the reader can refer to it and understand the method used. It is *not* necessary to copy the code from this listing - it is freely available by contacting the author.

Firstly, a note on the displayed output. Because some of the lines in the code are longer than can be displayed on the page, they have been wrapped. Any line without a line number is actually a continuation of the line before. This is important because some of the lines, as printed, will not work in **MATLAB** if input with the extra carriage returns.

Some elements are common to all the below files. The first line is the function definition, which describes which inputs the function expects, and what it will return. The commented lines immediately following that explains what the inputs mean, and what the function does. This is the information that is displayed when you type *help* foo.

The first functional part of the code starts with *switch nargin*. This is simply a switch for the number of inputs that have been received - any values that are missing are given default values in the *case* commands immediately below. This is the input switch.

Similarly, the output switch towards the end of the function starts with *switch gimme*, and simply decides which of the variables that have been calculated in the function to return. This is the output switch. Although this part of the code can become rather large in some functions, it is entirely trivial. For the sake of keeping the pagecount down, and to differentiate this part of the code from the important parts, the output switch part of the code is shown in a smaller font.

Elements starting with *for ri=1:nr* and *for ti=1:nt* are radial and timewise stepping, respectively. In using these iterative models, we lose some runtime, especially because **MATLAB** is good at fast calculations for matrices. The advantage of this is that the code is a good deal clearer, and it is much easier to locate problems at certain positions. For example, the inner and outer radial position often need special treatment, to avoid divide by zero errors. For both the datasets, the inner position is at a standstill - thus, all velocities, forces and wake contributions there are set forcibly to 0. Similarly, at the tip, the wing semichord is 0 - therefore the forces and wake contributions there are set forcibly to 0. For the Robofly dataset, the inner 60mm of the wing are used for measuring equipment. This is accommodated by setting all force and wake contributions out to 60mm to 0.

Some elements start with *if verb* or *if show*. These are checks to see if the user has requested extra information to be displayed via the variables *verb* and *show*. The code immediately following such statements is to display additional information at runtime, or plot the results to graphs. It doesn't affect the running of the code.

Most of the code is commented, and should be fairly self-explanatory. The function **master_wag** and **master_kus** have warranted detailed explanation, because of their complexity. The reader may wish to refer directly to these on pages 210 and 220.

Finally, a note on nomenclature: The "lift" referred to below is everywhere the force F_V. Similarly, the "drag" is everywhere the force F_H.

B.1 Data Functions

B.1.1 geom

This is the geometry datafile for the FMAV-50/2 dataset. It defines the wing geometry, and performs wing shape parameter calculations.
Line 29 is a flag wether the alternative geometry in **geom2** should be used instead.
Line 65 creates the default vector of radial positions, used by all other functions. *r* is normalised w.r.t. tip radius.
lines 109-113 creates the semichord *b*, which is not normalised.
line 112 *dledr* is the slope of the leading edge - note this is *not* normalised, it is true geometry.
line 224 onwards calculates wing shape parameters. This is done symbolically. Note that one of the wing shape parameters can't be formed symbolically by **MATLAB** so is calculated numerically instead. This costs 30-40 seconds of runtime.

```
1 function out = main(gimme,type,r,verb,show);
2 %out = geom(gimme,'t'|'r',r,verb,show)
3 %datafile. All geometric data is obtained by calling this function
4 %gimme is a string specifying which information to return
5 % 't'|'r' is a string specifying rotary or translational chord
6 % r the normalised radius. This can be a vector.
7 % if r is a string, the calculation is performed for max values
8 % verb is a 0|1 flag if additional information should be shown
9 % show is a 0|1 flag if data should be shown as a figure
10
11
12 %Last edited 7.11.02 by CBP
13 %This geometry datafile is for the 2-elipse wingshape
14 last_edited = '07.Nov.03';
15 last_run = date;
16
17 %flag if we should be using dickinson data, instead
18 %Note this is hardcoded.
19 dickinson = 0;
20 if dickinson
21     switch nargin
22     case 0
23         disp([mfilename ' error: must have at least one input']);
24     case 1
25         type = 't'; r = 'tip'; verb = 0;show = 0;
26     case 2
27         r = 'tip';verb = 0;show = 0;
28     case 3
29         verb = 0; show = 0;
30     case 4
31         show = 0;
```

```
    case 5
        %do nothing
    case 6
        disp('too many input arguments');
    end
    out = geom2(gimme,type,r,verb,show);
    return
    warning('should not be here')
end
%End of Dickinson part

switch nargin
case 0
    disp([mfilename ' error: must have at least one input']);
case 1
    type = 't'; r = 'tip'; verb = 0;show = 0;
case 2
    r = 'tip';verb = 0;show = 0;
case 3
    verb = 0; show = 0;
case 4
    show = 0;
case 5
    %do nothing
case 6
    disp('too many input arguments');
end


R = 0.15;
c = R/3;
B = c/2;
c_default = linspace(-1,1,10);
r_default = linspace(0,1,12);
hinge= -1/2; %hinge at 1/4 chord

switch type
case 't';
    if verb>1 disp('translational chord');end
case 'r';
    if verb>1 disp('rotational chord');end
otherwise
    disp(['geom function error - chord type must be r or t: '
        num2str(type)]);
    out = -1;
    return;
```

```
end

%symbolic expression for chord
tipflag = 0;
if isstr(r) %if r is a string (typically 'tip') assume we are
    calculating tip values
    r = 1; b = B; tipflag = 1;
else
    %assume r is the normalised radius we want to calculate at,
        possibly a vector
    if max(r)>1 disp([mfilename ' warning: received radius in
        excess of 1, radius should be normalised']);end
    if min(r)<0 disp([mfilename ' warning: received radius below
        0, radius is normalised, from the hinge']);end
end

r0 = 0.75; %point where we change elipses, also max chord
[err,r0index] = min(abs(r0-r));

%symbolic expressions for the wing shape
if length(gimme)==2 & (gimme == 'b1' | gimme == 'b2')
    r = sym('r','real');
    b1 = sqrt(1-(1-r/r0).^2);
    tempa = (r-r0).^2; tempb = (1-r0)^2;
    b2 = sqrt(1-tempa./tempb);
else
    b = sqrt(1-(1-r/r0).^2);
    FIND = find(r>r0);
    tempa = (r-r0).^2; tempb = (1-r0)^2;
    b(FIND) = sqrt(1-tempa(FIND)./tempb);
    b = B * b;
end

%if tipflag, all chords are maximum
if tipflag b = ones(size(r));end

%translational chord
le = (+1+hinge) * b; %find leading
te = (-1+hinge) * b; %trainling edge
dledr = [0 diff(le)./diff(r)]/R; %leading edge slope
b = (le - te)/2; %semichord

%show wingshape
if show hold off;plot(r*R,le,'b-',r*R,te,'b-');hold on;axis equal;
    end

```

```
%convert to rotational chord, if requested
if type == 'r'
    [r,le] = l2rc(le,r,length(r),0);
    [r,te] = l2rc(te,r,length(r),0);
end

%Wing shape identifier, used for calls to ntharm below
shape = '2e';

%Output switch
if isstr(gimme)
   switch type
   case 't';
      switch gimme
      case {'id','ID'}
         out = ['Geometry: 2-elipse planform, hinge ' num2str((hinge+1)/2)];
      case 'area'
         out = 2 * B * R * ntharm(1,0,shape);
      case 'R'
        out = R;
      case 'B'
        out = c/2;
     case 'b'
        out = b;
      case 'c'
        out = c;
      case 'hinge'
        out = hinge;
     case 'shape'
        out = shape;
     case 'b1'
        out = b1;
     case 'b2'
        out = b2;
     case {'change','r0'}
        out = r0;
      case 'b1r0'
        out = ntharm(1,0);
      case 'b1r1'
        out = ntharm(1,1);
      case 'b1r2'
        out = ntharm(1,2);
     case 'b1r3'
        out = ntharm(1,3);
      case 'b2r1'
        out = ntharm(2,1);
     case 'b2r2'
        out = ntharm(2,2);
      case 'b3r0'
        out = ntharm(3,0);
     case 'b3r1'
        out = ntharm(3,1);
     case 'b4r0'
        out = ntharm(4,0);
     case 'b4r1'
        out = ntharm(4,1);
     case 'b5r0'
        out = ntharm(5,0);
     case 'b1r0P'
        out = ntharm(1,0,1);
      case 'b1r1P'
        out = ntharm(1,1,1);
      case 'b1r2P'
        out = ntharm(1,2,1);
     case 'b1r3P'
        out = ntharm(1,3,1);
      case 'b2r1P'
        out = ntharm(2,1,1);
     case 'b2r2P'
        out = ntharm(2,2,1);
      case 'b3r0P'
        out = ntharm(3,0,1);
     case 'b3r1P'
        out = ntharm(3,1,1);
     case 'b4r0P'
        out = ntharm(4,0,1);
     case 'le'
        out = le;
     case 'te'
        out = te;
     case 'dledr'
```

```
            out = dledr;
        case 'c_default';
            out = c_default;
        case 'r_default';
            out = r_default;
        case 'r0index'
            out = r0index;
        case 'dr'
            dr = [0 diff(geom('r_default'))];
            out = dr;
         otherwise
            disp([mfilename ' error, unknown string passed:' gimme])
            out = -1;
            return
        end
    case 'r';
        disp('Warning - rotary chord requested, not yet implemented')
        out = -1;
        return
    end
    if verb disp(gimme);end
    return
end
```

```
function out = ntharm(m,n,le,shape,verb);
%ntharm: calculates wing shape parameters
%out = ntharm(m,n,le,b,r);
%wing parameter is b_m r_n
%le=1 uses correction for leading edge slope (for Polhamus)
%this implemenatation is analytical

%Created by C.Pedersen
%Last edited 27.12.02

switch nargin
case {0,1}
    disp([mfilename ' error, need at least 2 inputs']); return;
case 2
    shape = 'default'; verb = 0; le = 0;
case 3
    shape = 'default'; verb = 0;
case 4
    le = 0;
case 5
    %do nothing
otherwise
    disp([mfilename ' error, too many input arguments']); return;
end

r = sym('r','real'); %this is normalised radius.


switch shape
case 'default'
    r0 = geom('r0');
    b1 = geom('b1');
    b2 = geom('b2');
```

```
    B = geom('B');
    R = geom('R');
    hinge = geom('hinge');
case '2e'  %2-elipse planform
    r0 = 3/4;    %point where the functions change over
    b1 = sqrt(1-(1-r/r0).^2);
    tas = (r - r0)^2; tbs = (1-r0)^2;
    b2 = sqrt(1-tas/tbs);
    B = 0.025;
    R = 0.15;
    hinge = -.5;
case 'r' %rectangular wing
    b1 = 1;
    b2 = 1;
    r0 = 0.5;
    B = 0.025;
    R = 0.15;
    hinge = -.5;
case 'triangle'
    r0 = 0;
    b1 = 1;
    b2 = 1 - (r-r0)/(1-r0);
    B = 0.025;
    R = 0.15;
    hinge = -0.5;
otherwise
    disp([mfilename ' error, unknown type received: ' num2str(shape)]); return
end

if le
    symbolic = 1;
    if symbolic
        warnstate = warning; warning off
        %some integrals are not properly symbolic, but still work
        db1 = diff(b1,r)*(hinge+1)*B/R;
        db2 = diff(b2,r)*(hinge+1)*B/R;
        corr1 = simple(sqrt(1+db1^2));
        corr2 = simple(sqrt(1+db2^2)); %leading edge correction factor
        out1= int(b1^m * r^n * corr1,r,0,r0);
        out2= int(b2^m * r^n * corr2,r,r0,1); %cant integrate entire region
        out  = abs(out1 + out2);
        warning(warnstate)
    else %numerical calculation
```

```
        R = geom('R');
        B= geom('B');
        r = 0:1/10000:1;
        dr = [0 diff(r)];
        le = geom('le','t',r);
        b = geom('b','t',r);
        dle = [0 diff(le)./diff(r)/R];
        corrP = sqrt(1+dle.^2);
        out = sum(b.^m .* r.^n .* dr .* corrP);
    end
else
    out= int(b1^m * r^n,r,0,r0) + int(b2^m * r^n,r,r0,1);
end

if verb
    if le
        disp(['    arm b' num2str(m) 'r' num2str(n) ' with le correction'])
    else
        disp(['    arm b' num2str(m) 'r' num2str(n)])
    end
end
out = double(out); %converts the symbolic result to a number
```

B.1.2 kine

This is the kinematics datafile for the FMAV-50/2 dataset. It defines the wing kinematics, and runtime parameters.
Line 21 is a flag wether the alternative kinematics in **kine** should be used instead.
Line 58 sets the default number of timesteps *nt=2048*. This is rather high, for the purpose of smooth plots. Almost identical results are obtained using *nt=512*, with the greatest difference being in the primary wake effect. *nt* should be a power of 4, for the sake of smooth transition from rotating to translating motion.
lines 108–172 define the basic kinematics of the wing tip. Note the velocities and accelerations are formed analytically at every timestep, not numerically differentiated, to avoid numerical noise.

```
1  function out = main(gimme,nt,verb,show,show_type);
2  %out = kine(gimme,nt,verb,show,show_type)
3  %datafile. All kinematic data is obtained by calling this function
4  %gimme is a string specifying which information to return
5  %nt is the number of timesteps
6  % verbose is a 0|1 flag if additional information should be shown
       (typically for debugging)
7  % show is a 0|1 flag if data should be shown as a figure
8
9  %Part of Project Mekado
10 %Called by: All
11 %Calls: None
12
13 %Last edited 18.11.02 by CBP
14 %contact pedersen@rmcs.cranfield.ac.uk
15 %This kinematic datafile is for the lissajous trajectory, as
       outlined in document meki-js01
16 last_edited='18.Nov.02';
17 last_run=date;
18
19 %flag if we should be using dickinson data, instead
20 %Note this is hardcoded.
21 dickinson = 0;
22 if dickinson
23     switch nargin
24     case 0
25     disp([mfilename ' error: needs at least 1 input']);
26     case 1
27         %no timestep number given, use default
28         cycle = 0; %default timestep to use
29         verb = 0;
30         show = 0;
```

```
        show_type = 'basic';
    case 2
        show = 0;
        verb = 0;
        show_type = 'basic';
    case 3
        show = 0;
        show_type = 'basic';
    case 4
        show_type = 'basic';
    case 5
        %do nothing
    otherwise
        disp([mfilename ' error: too many inputs'])
    end
    out = kine2(gimme,cycle,verb,show,show_type);
    return
    disp('wrong place')
end
%end of dickinson part

%Input switch
switch nargin
case 0
    disp([mfilename ' error: needs at least 1 input']);
case 1
    %no timestep number given, use default
    nt = 2048; %default number of timesteps
    verb = 0;
    show = 0;
    show_type = 'basic';
case 2
    show = 0;
    verb = 0;
    show_type = 'basic';
case 3
    show = 0;
    show_type = 'basic';
case 4
    show_type = 'basic';
case 5
    %do nothing
otherwise
    disp([mfilename ' error: too many inputs'])
end

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% define period and air density
%
period = 1/20;
f = 1/period;
dt = period / nt; %timestep
rho = 1.225; %fluid density
w = 2*pi*f; %radian frequency
nwak=1; %number of cycles in full wake
firststep = 'w'; %vauels are (w)rap, (s)mooth or (i)mpulse.
%decides how to form the first step for wagner and kussner
datalength = 'f'; %values are (f)ull or (o)ther: length of data — full cycle or other.
wakemethod = 'f'; %wether to form (f)ull secondary vortex sets, or (g)row them from the start time
polmethod = 'lt'; %which polhamus method to use
usepolhamus = 'y'; %wether to use polhamus to correct cl for wake calculations
tailflag = 1;  %wether to calculate wake location and reversal based on tailing edge, or hinge velocity
tshow = [1 200 500]; %which timesteps to show
rshow = 9; %which radial position to show
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb
    disp(['Period ' num2str(period)]);
    disp(['Timesteps ' num2str(nt)]);
    disp(['Fluid Density ' num2str(rho)]);
end

t = 0:dt:period-dt;
tt = w * t;
dt = period/nt * Der(t,t);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%create pitch angle
%
%
n1 = floor(nt/8);
n2 = floor(3*nt/8);
nmid = floor(nt/2);
n3 = floor(5*nt/8);
n4 = floor(7*nt/8);

p = .5 * (sin(4*tt) + tt*4) + pi/2;
```

```
p(n1:n2) = pi*ones(size(p(n1:n2)));
p(n2+1:n3) = -p(n2+1:n3)+pi+p(n2+1);
p(n3:n4) = 0*ones(size(p(n1:n2)));
p(n4+1:nt) = p(n4+1:nt)-p(n4+1);

dp = 2*(cos(4*tt)+1) * w;
dp(1:n1)   = dp(1:n1);
dp(n1:n2) = zeros(size(p(n1:n2)));
dp(n2+1:n3) = -dp(n2+1:n3);
dp(n3:n4) = zeros(size(p(n1:n2)));

ddp = -8 * w^2 * sin(4*tt);
ddp(n1:n2) = zeros(size(ddp(n1:n2)));
ddp(n2:n3) = - ddp(n2:n3);
ddp(n3:n4) = zeros(size(ddp(n1:n2)));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% scale rotation
dummy = 180;
p = dummy/180 * (p-pi/2) + pi/2;
dp = dummy/180 * dp;
ddp = dummy/180 * ddp;

SP = sin(p);
CP = cos(p);

%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create sweep angle (back positive)
%
%
A = 120; %total angle swept;
A = A *pi /180 /2; %amplitude swept
phi = -A * cos(tt);
dphi =  A * w * sin(tt);
ddphi = A * w^2 * cos(tt);
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create plunge angle (down positive)
```

```
%
psi   = -A/8 * sin(2*tt);
dpsi = -A/4 * w * cos(2*tt);
ddpsi=  A/2 * w^2 * sin(2*tt);
clear A;
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%get variables needed
R = geom('R','t');
B = geom('B','t');
hinge = geom('hinge','t');

%note these velocities are of the flow relative to the wing
uht = -R   * dphi;
duht = -R .* ddphi;

uvt  = R   * dpsi;
duvt  = R * ddpsi;

unt  = uht .* SP + uvt.*CP;
upt  =   uht .* CP - uvt.*SP;

dunt = duht .* SP + duvt .* CP + 2 * dp .* upt;
%careful: note coriolis term
dupt = duht .* CP - duvt .* SP  - 2 * dp .* unt;
%careful of the coriolis term here, too

ut   = abs(uht + sqrt(-1)*uvt);
ut2  = abs(unt + sqrt(-1)*upt);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% convert to x,y,z coordiantes
%
if gimme == 'x' | gimme == 'y' | gimme == 'z'
      tip_basic = [0 R 0]; %rest position of tip
      for i=1:nt
            TIP_R = rotator(tip_basic,phi(i),psi(i),0);
            x(i) = TIP_R(1);
            y(i) = TIP_R(2);
            z(i) = TIP_R(3);
      end
end
%
%
```

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

%output switch
if isstr(gimme)
    switch gimme
    case {'id','ID'}
        out = ['Kinematics: Lissajous, frequency ' num2str(f)];
    case 'nt'
        out = nt;        if verb disp('Timesteps');end
    case 'nrot'
        out = nrot;      if verb disp('second halfstroke start index');end
    case 'dt'
        out = dt;        if verb disp('Timestep length');end
    case 'rho'
        out = rho;       if verb disp('Density');end
    case 'nt'
        out = nt;        if verb disp('Number of timesteps');end
    case 'T'
        out = period;    if verb disp('period');end
    case 'f'
        out = f;         if verb disp('frequency');end
    case 'ut'
        out = ut;       if verb disp('tip total velocity');end
    case 'ut2'
        out = ut2;       if verb disp('tip total velocity 2');end
    case 'uht'
        out = uht;       if verb disp('tip horizontal velocity');end
    case 'uvt'
        out = uvt;       if verb disp('tip vertical velocity');end
   case 'ut'
        out = ut;        if verb disp('tip total velocity');end
    case 'unt'
        out = unt;       if verb disp('tip normal velocity');end
     case 'upt'
        out = upt;       if verb disp('tip paralelle velocity');end
    case 'duht'
        out = duht;       if verb disp('tip horz acceleration');end
    case 'duvt'
        out = duvt;       if verb disp('tip vert acceleration');end
    case 'dunt'
        out = dunt;       if verb disp('tip norm acceleration of fluid');end
    case 'dupt'
        out = dupt;       if verb disp('tip parl acceleration of fluid');end
    case 'dp'
        out = dp;        if verb disp('pitching velocity (rad/s)');end
    case 'ddp'
        out = ddp;        if verb disp('pitching acceleration (rad/s^2)');end
    case 'phi'
        out = phi;       if verb disp('sweep angle');end
    case 'dphi'
        out = dphi;       if verb disp('sweep angular velocity');end
    case 'ddphi'
        out = ddphi;       if verb disp('sweep angular acceleration');end
    case 'psi'
        out = psi;       if verb disp('plunge angle');end
    case 'dpsi'
        out = dpsi;      if verb disp('plunge anglular velocity');end
    case 'ddpsi'
        out = ddpsi;      if verb disp('plunge anglular acceleration');end
    case 'SP'
        out = SP;        if verb disp('sin(pitch)');end
    case 'CP'
        out = CP;        if verb disp('cos(pitch)');end
    case 'pitch'
        out = p;         if verb disp('pitch');end
     case 't'
        out = t;         if verb disp('time');end
     case 'tt'
        out = tt;         if verb disp('phase time');end
    case 'x'
        out = x;         if verb disp('x of hinge');end
    case 'y'
        out = y;         if verb disp('y of hinge');end
    case 'z'
        out = z;         if verb disp('z of hinge');end
    case 'firststep'
        out = firststep;    if verb disp('method for first step');end
    case 'datalength'
        out = datalength;  if verb disp('data length');end
    case 'usepolhamus'
        out = usepolhamus; if verb disp('adjust wake for polhamus flag');end
    case 'tailflag'
        out = tailflag; if verb disp('tailing edge flag');end
    case 'wakemethod'
        out = wakemethod; if verb disp('wake method');end
    case 'tshow'
        out = tshow; if verb disp('which timesteps to show');end
```

```
    case 'rshow'
        out = rshow; if verb disp('which radial position to show');end
    case 'nwak'
        out = nwak; if verb disp('number of cycles in full wake');end
    case 'polmethod'
        out = polmethod; if verb disp('polhamus method to use');end
    otherwise
        disp([mfilename ' error: unknown string received:' num2str(gimme)])
    end
end
```

B.1.3 geom2 and kine2

These are the alternative datafiles for the Robofly experiment. They are almost identical to the above, except:
The wing shape parameters are calculated numerically.
The velocity and acceleration terms are formed by numerical differentiation of the position. This gives some noise, which is corrected for manually.
The data is read from a file, not formed analytically.

geom2

```
function out = main(in,type,r,verb,show);
%out = geom(gimme,'t'|'r',r,verb,show)
%core datafile. All geometric data is obtained by calling this function
%what is a string specifying which information to return
% 't'|'r' is a string specifying wether you want rotary or translational chord
% r the normalised radius where you want the geometry for. This can be a vector
% if r is a string, the tip calculation is performed, for maximum radius and chord
% verb is a 0|1 flag if additional information should be shown (typically for debugging)
% show is a 0|1 flag if data should be shown as a figure

%Last edited 13.May.03 by CBP
%This geometry datafile is for the Dickinson wingshape.
last_edited='13.may.03';
last_run=date;

%Input switch
switch nargin
case 0
    disp([mfilename ' error: must have at least one input']);
case 1
    type = 't'; r = 'tip'; verb = 0;show = 0;
case 2
    r = 'tip'; verb = 0;show = 0;
case 3
    verb = 0; show = 0;
case 4
    show = 0;
case 5
    %do nothing
```

```
case 6
    disp('too many input arguments');
end

data = 'rundata/dick_wing';
load(data);

R = 0.25;
B = max(b);
c = 2*B;
%r_default loaded from file
% hinge is not a constant !
[err,r0index] = min(abs(B-b)); %point where chord is maximum

switch type
case 't';
    if verb>1 disp('translational chord');end
case 'r';
    if verb>1 disp('rotational chord');end
otherwise
    disp(['geom function error - chord type must be r or t: '
        num2str(type)]);
    out = -1;
    return;
end


tipflag = 0;
%if r is a string (typically 'tip') assume we are calculating
    using max values
if isstr(r)
    r = ones(size(hinge)); b = ones(size(hinge))*B; tipflag = 1;
else
    %assume r is the normalised radius we want to calculate at,
        possibly a vector
    if max(r)>1 disp([mfilename ' warning: received radius in
        excess of 1, radius should be normalised']);end
    if min(r)<0 disp([mfilename ' warning: received radius below
        0, radius is normalised, from the hinge']);end
end
r0 = r(r0index);

%if tipflag, all chords are maximum
if tipflag b = ones(size(r));end

%translational chord
```

```
le = (+1+hinge) .* b; %leading edge
te = (-1+hinge) .* b; %trailing edge

a = warning;
warning off
dledr = [0 diff(le)./diff(r)]/R; %leading edge slope
warning(a)
clear a;

%show wingshape
if show hold off;plot(r*R,le,'b-',r*R,te,'b-');hold on;axis equal;
    end

%convert to rotational chord, if requested
if type == 'r'
    [r,le] = l2rc(le,r,length(r),0);
    [r,te] = l2rc(te,r,length(r),0);
end

%Identifier, used for calls to ntharm below
shape = 'dick';

%Output switch
if isstr(in)
   switch type
   case 't';
      switch in
      case {'id','ID'}
          out = ['Geometry: Dickinson wing'];
      case 'area'
          out = 2 * B * R * geom('b1r0');
      case 'R'
         out = R;
      case 'B'
         out = c/2;
     case 'b'
         out = b;
      case 'c'
         out = c;
      case 'hinge'
         out = hinge;
     case 'shape'
         out = shape;
     case 'b1'
         out = b1;
     case 'b2'
         out = b2;
     case {'change','r0'}
         out = r0;
     case 'b0r0'
         out = ntharm(0,0);
      case 'b1r0'
         out = ntharm(1,0);
      case 'b1r1'
         out = ntharm(1,1);
      case 'b1r2'
         out = ntharm(1,2);
     case 'b1r3'
         out = ntharm(1,3);
      case 'b2r1'
         out = ntharm(2,1);
     case 'b2r2'
         out = ntharm(2,2);
      case 'b3r0'
         out = ntharm(3,0);
     case 'b3r1'
         out = ntharm(3,1);
     case 'b4r0'
         out = ntharm(4,0);
```

```
        case 'b4r1'
            out = ntharm(4,1);
        case 'b5r0'
            out = ntharm(5,0);
        case 'b1r0P'
            out = ntharm(1,0,1);
         case 'b1r1P'
            out = ntharm(1,1,1);
         case 'b1r2P'
            out = ntharm(1,2,1);
        case 'b1r3P'
            out = ntharm(1,3,1);
         case 'b2r1P'
            out = ntharm(2,1,1);
        case 'b2r2P'
            out = ntharm(2,2,1);
         case 'b3r0P'
            out = ntharm(3,0,1);
        case 'b3r1P'
            out = ntharm(3,1,1);
        case 'b4r0P'
            out = ntharm(4,0,1);
        case 'le'
            out = le;
        case 'te'
            out = te;
        case 'dledr'
            out = dledr;
        case 'c_default';
            out = c_default;
        case 'r_default';
            out = r_default;
        case 'r0index'
            out = r0index;
        case 'dr'
            %dr is the spanwise length of each element (normalised)
            dr = [0 diff(geom('r_default'))];
            %set the second value of dr to 0
            %this represents the inner 9 cm of the wing
            %where the measuring equipment is
            %so assume it generates no forces
            dr(2) = 0;
            out = dr;
         otherwise
            disp([mfilename ' error, unknown string passed:' in])
            out = -1;
            return
        end
    case 'r';
        disp('Warning - rotary chord requested, not yet implemented')
        out = -1;
        return
    end
    if verb disp(in);end
    return
end


function out = ntharm(m,n,pol,b,r);
%ntharm: calculates wing shape parameters
%out = ntharm(m,n,le,b,r);
%wing parameter is b_m r_n
%le=1 uses correction for leading edge slope (for Polhamus)
%this implemenatation is numeric, not analytical

%Created by C.Pedersen
%Last edited 13.May.03

switch nargin
case {0,1}
    disp([mfilename ' error, need at least 2 inputs']); return;
case 2
    pol = 0; shape = 'dick'; verb = 0;
```

```
case 3
    shape = 'dick'; verb = 0;
case 4
    verb = 0;
case 5
    %do nothing
otherwise
    disp([mfilename ' error , too many input arguments']); return;
end

r = sym('r','real'); %this is normalised radius.

R= geom('R');
B= geom('B');
r= geom('r_default');
dr = geom('dr','t',r);
le = geom('le','t',r);
b = geom('b','t',r)/B;
dle = geom('dledr','t',r);

%Leading edge correction (Polhamus)
corrP = sqrt(1+dle.^2).^pol;
sum(b.^m .* r.^n .* dr);
out = sum(b.^m .* r.^n .* dr .* corrP);


if verb
    if le
        disp(['    arm b' num2str(m) 'r' num2str(n) ' with le correction'])
    else
        disp(['    arm b' num2str(m) 'r' num2str(n)])
    end
end
out = double(out); %converts the (possibly) symbolic result to a number
```

kine2

```
function out = main(gimme,cycle,verb,show,show_type);
%out = kine(gimme,nt,verb,show,show_type)
%datafile. All kinematic data is obtained by calling this function
%gimme is a string specifying which information to return
%nt is the number of timesteps
% verbose is a 0|1 flag if additional information should be shown (typically for debugging)
% show is a 0|1 flag if data should be shown as a figure


%Last edited 13.5.03 by CBP
%This kinematic datafile is for the triangular wave of dickinson's robofly
%With rotation leading the reversal

last_edited='13.May.03';
last_run = date;

%load data from file
load 'rundata/dick_kine'

%Input switch
switch nargin
case 0
    disp([mfilename ' error: needs at least 1 input']);
    return
case 1
    cycle = 1;
    verb = 0;
    show = 0;
    show_type = 'basic';
case 2
    show = 0;
    verb = 0;
    show_type = 'basic';
case 3
    show = 0;
    show_type = 'basic';
case 4
    show_type = 'basic';
case 5
    %do nothing
otherwise
    disp([mfilename ' error: too many inputs'])
```

```
end

tailflag = 1; %calculate reversal based on tailing edge
nrot = 'find'; %automatically find reversal times
firststep = 'i'; %values are (w)rap, (i)mpulsive or (s)mooth
%decides how to form the first step for wagner and kussner
datalength = 'f'; %values are (f)ull, (o)ther
%length of data - full cycle, or other.
wakemethod = 'g'; %values are (g)row from start, or (f)ull
polmethod = 'lt'; %which polhamus method to use
usepolhamus = 'y'; %adjust cl for polhamus when calculating wake (y)es or (n)o
rshow = 14; %which radial position to plot
tshow = [1 50 100 200 250]; %which time positions to plot
nwak = 1;

%Choose which timesteps to use
switch cycle
case 0
    ti = 1:2356; firststep = 'i'; datalength = 'o';
case 1
    ti = 2:297; firststep = 'i';
case 2
    ti = 298:598;
case 3
    ti = 599:891;
case 4
    ti = 892:1185
case 5
    ti = 1186:1480;
case 6
    ti = 1481:1774;
case 7
    ti = 1774:2068;
case 8
    ti = 2069:2356;
case 9
    ti = 2:598; firststep = 'i'; datalength = 'o';
otherwise
    disp([mfilename ' error: cycle number must be 0-9, but is ' num2str(cycle)]);
end

%dont extract relevant cycle until last step.
%slower, but derivatives are more accurate
nt = length(ti);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% define period and air density
%
period = 3;
f = 1/period;
dt = period / nt;
rho = 870; %mineral oil
w = 2*pi*f;
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if verb
    disp(['Period ' num2str(period)]);
    disp(['Timesteps ' num2str(nt)]);
    disp(['Fluid Density ' num2str(rho)]);
end

%phase time
tt = w * t;

%pitch angle
p = rot / 180 * pi + pi/2;
p = p';
t = t';
SP = sin(p);
CP = cos(p);

%pitching rate
dp = der(p,t);
dp(2) = 1.01; %manually set first value to give a good fit
dp(590) = 1.06;
dp(1179) = 1.04;
dp(1768) = 1.035;

%pitching acceleration
ddp = der(dp,t);

%There is some numerical noise, which is aggrevated by double-differentiating
%we manually smooth this:
ddp(2) = -5.18; ddp(3) = -5.11; %manually set first two values
ddp(590) = -5.18; ddp(591) = -5.18;
ddp(1179) = -5.18; ddp(1180) = -5.18;
```

```
ddp(1768) = -5.15; ddp(1769) = -5.16;

%sweep and plunge angles
phi = -az'/180 * pi;
psi = -el'/180 * pi;

dphi = der(phi,t);
dpsi = der(psi,t);

ddphi = der(dphi,t);
ddpsi = der(dpsi,t);

%manually adjust values to be smooth
dphi(2) = -0.25;
ddphi(2) = 5.2; ddphi(3)=5.25;

%get geometric variables needed
R = geom('R','t');
B = geom('B','t');
hinge = geom('hinge','t');

%note these velocities are of the flow relative to the wing
uht = -R  * dphi;
duht = -R .* ddphi;

uvt = R  * dpsi;
duvt = R * ddpsi;

unt = uht .* SP + uvt.*CP;
upt =  uht .* CP - uvt.*SP;

dunt = duht .* SP + duvt .* CP + 2 * dp .* upt;
%careful: note the coriolis term
dupt = duht .* CP - duvt .* SP - 2 * dp .* unt;
%careful of the coriolis term here, too

ut  = abs(uht + sqrt(-1)*uvt);
ut2 = abs(unt + sqrt(-1)*upt);

%position in x,y,z coordinate system
if gimme == 'x' | gimme == 'y' | gimme == 'z'
    tip_basic = [0 R 0]; %rest position of tip
    for i=ti
        TIP_R = rotator(tip_basic,phi(i),psi(i),0);
        x(i) = TIP_R(1);
        y(i) = TIP_R(2);
```

```
            z(i) = TIP_R(3);
        end
end

%output switch
if isstr(gimme)
    switch gimme
    case {'id','ID'}
        out = ['Kinematics: Dickinson, frequency ' num2str(f)];
    case 'nt'
        out = nt;        if verb disp('Number of timesteps');end
    case 'nrot'
        out = nrot;      if verb disp('second halfstroke start index');end
    case 'dt'
        out = dt;        if verb disp('Timestep length');end
    case 'rho'
        out = rho;       if verb disp('Density');end
    case 'T'
        out = period;    if verb disp('period');end
    case 'f'
        out = f;         if verb disp('frequency');end
    case 'ut'
        out = ut(ti);       if verb disp('tip total velocity');end
    case 'ut2'
        out = ut2(ti);      if verb disp('tip total velocity 2');end
    case 'uht'
        out = uht(ti);      if verb disp('tip horizontal velocity');end
    case 'uvt'
        out = uvt(ti);      if verb disp('tip vertical velocity');end
   case 'ut'
        out = ut(ti);       if verb disp('tip total velocity');end
    case 'unt'
        out = unt(ti);      if verb disp('tip normal velocity');end
     case 'upt'
        out = upt(ti);      if verb disp('tip paralelle velocity');end
    case 'duht'
        out = duht(ti);      if verb disp('tip horz acceleration');end
    case 'duvt'
        out = duvt(ti);      if verb disp('tip vert acceleration');end
    case 'dunt'
        out = dunt(ti);      if verb disp('tip norm acceleration');end
    case 'dupt'
        out = dupt(ti);      if verb disp('tip parl acceleration');end
   case 'dp'
        out = dp(ti);       if verb disp('pitching velocity (rad/s)');end
    case 'ddp'
        out = ddp(ti);       if verb disp('pitching acceleration (rad/s^2)');end
    case 'phi'
        out = phi(ti);      if verb disp('sweep angle');end
    case 'dphi'
        out = dphi(ti);      if verb disp('sweep angular velocity');end
    case 'ddphi'
        out = ddphi(ti);      if verb disp('sweep angular acceleration');end
    case 'psi'
        out = psi(ti);      if verb disp('plunge angle');end
    case 'dpsi'
        out = dpsi(ti);     if verb disp('plunge anglular velocity');end
    case 'ddpsi'
        out = ddpsi(ti);     if verb disp('plunge anglular acceleration');end
    case 'SP'
        out = SP(ti);       if verb disp('sin(pitch)');end
    case 'CP'
        out = CP(ti);       if verb disp('cos(pitch)');end
    case 'pitch'
        out = p(ti);        if verb disp('pitch');end
     case 't'
        out = t(ti)-t(ti(1));        if verb disp('time');end
     case 'tt'
        out = tt(ti)-tt(ti(1));        if verb disp('phase time');end
    case 'x'
        out = x(ti);        if verb disp('x of hinge');end
    case 'y'
        out = y(ti);        if verb disp('y of hinge');end
    case 'z'
        out = z(ti);        if verb disp('z of hinge');end
    case 'firststep'
        out = firststep;    if verb disp('method for first step');end
    case 'datalength'
        out = datalength;  if verb disp('data length');end
    case 'tailflag'
        out = tailflag;     if verb disp('tail flag');end
    case 'ti'
        out = ti;       if verb disp('time indexes');end
    case 'wakemethod'
        out = wakemethod;  if verb disp('wake method');end
    case 'usepolhamus'
```

```
        out = usepolhamus; if verb disp('adjust wake for polhamus flag');end
    case 'rshow'
        out = rshow; if verb disp('which radial position to show');end
    case 'tshow'
        out = tshow; if verb disp('which radial position to show');end
    case 'nwak'
        out = nwak; if verb disp('number of times to repeat main cycle for full wake');end
    case 'polmethod'
        out = polmethod; if verb disp('polhamus method to use');end
    otherwise
        disp([mfilename ' error: unknown string received:' num2str(gimme)])
    end
else
    message(toc,[mfilename ' error: need a string for gimme: ' num2str(gimme)]);
end
```

B.2 Calculation Functions

These functions perform the calculations of section 8 to 11.

B.2.1 qs

This performs the quasi-steady calculations of section 8. Remember that the lift and drag referred to are actually F_V and F_H.
Lines 110–125 form the normal and parallel components both as a function of the vertical and horizontal, and as direct functions of the velocity. This is done for the sake of cross-checking.
Lines 129-131 form the bound vorticity of the wing.

```
1 function out = main(gimme,uh,uv,pitch,dp,r,b,hinge,verb,show);
2 %calculates the quasi-steady results.
3 %out = qs('gimme',uh,uv,pitch,dp,r,b,hinge,verb,show);
4 %NOTE: full-wing values use tip values of uh,uv,R and maximum B,
     and geometry data from GEOM.M
5 %inputs must all be grids of values
6
7 %Created 21.4.03 by CP
8 last_edited='21.Apr.03';
9 last_run=date;
10
11 %parse input, and preamble
12 switch nargin
13 case 0
14     gimme = 'LW'; uh = kine('uht'); uv = kine('uvt'); pitch = kine
         ('pitch'); dp = kine('dp');
15     r = 1; b = geom('B'); hinge = geom('hinge'); verb = 0; show
         = 0;
16 case 1
17     uh = kine('uht'); uv = kine('uvt'); pitch = kine('pitch'); dp
         = kine('dp');
18     r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
         = 0;
19 case 2
20     uv = kine('uvt'); pitch = kine('pitch'); dp = kine('dp');
21     r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
         = 0;
22 case 3
23     pitch = kine('pitch'); dp = kine('dp');
24     r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
         = 0;
25 case 4
26     dp = kine('dp');
```

```
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
        = 0;
case 5
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
        = 0;
case 6
    b = geom('B'); hinge = geom('hinge');  verb = 0; show = 0;
case 7
    hinge = geom('hinge');  verb = 0; show = 0;
case 8
    verb = 0; show = 0;
case 9
    show = 0;
case 10
    %do nothing
otherwise
    disp([mfilename ' error: too many input arguments'])
    return
end

%calculate lift and drag forces
SP = sin(pitch); CP = cos(pitch);
un = uh.*SP + uv.*CP;
up = uh.*CP - uv.*SP;
ut = abs(uh + sqrt(-1)*uv);

%mean square total velocity along chord
ut2mean = ut.^2 + dp.^2 .* b.^2 .* (1/3 + hinge^2) - 2 * hinge .*
    un .* b .* dp;


rho = kine('rho');
%Normal Force per meter span
VEL = uh; ANG = SP;
L1 = 2*pi*rho*b*(VEL .* un);
L2 = 2*pi*rho*b^2*(dp.*ANG*(-hinge).*un);
L3 = 2*pi*rho*dp*b^2*(.5-hinge).*VEL;
L4 = 2*pi*rho*dp.^2*b^3.*ANG*(hinge^2-hinge/2);
L = L1 + L2 + L3 + L4;

%Drag Force per meter span
VEL = uv; ANG = CP;
D1 = 2*pi*rho*b*(VEL .* un);
D2 = 2*pi*rho*b^2*(dp.*ANG*(-hinge).*un);
D3 = 2*pi*rho*dp*b^2*(.5-hinge).*VEL;
D4 = 2*pi*rho*dp.^2*b^3.*ANG*(hinge^2-hinge/2);
```

```
D = D1 + D2 + D3 + D4;

%Lift Coefficient
if gimme(1) == 'C'
    den = rho * b * ut2mean;
    VEL = uh; ANG = SP;
    CL1 = L1./den;
    CL2 = L2./den;
    CL3 = L3./den;
    CL4 = L4./den;
    CL = CL1 + CL2 + CL3 + CL4;

    %Drag Coefficient
    VEL = uv; ANG = CP;
    CD1 = 2*pi*(VEL .* un)./ut2mean;
    CD2 = -2*pi*b.*(dp.*ANG*(.5+hinge).*un)./ut2mean;
    CD3 = 2*pi*dp.*b.*(.5-hinge).*VEL./ut2mean;
    CD4 = 2*pi*dp.^2.*b.^2.*ANG*hinge^2./ut2mean;
    CD = CD1 + CD2 + CD3 + CD4;
end

R = geom('R');
b3r0 = geom('b3r0','t'); b2r1 = geom('b2r1','t'); b1r2 = geom('
    b1r2','t');
%Lift force for entire wing
%Note, only works if received velocities are tip values, r,b are
    maximum values
LW1 = R * L1 * b1r2;
LW2 = R * L2 * b2r1;
LW3 = R * L3 * b2r1;
LW4 = R * L4 * b3r0;
LW = LW1 + LW2 + LW3 + LW4;

%Drag force for entire wing
%Note, only works if received velocities are tip values, r,b are
    maximum values
DW1 = R * D1 * b1r2;
DW2 = R * D2 * b2r1;
DW3 = R * D3 * b2r1;
DW4 = R * D4 * b3r0;
DW = DW1 + DW2 + DW3 + DW4;

%Normal force for entire wing
%N = L .* CP - D .* SP;
N = 2 * pi * rho * b * (un + dp .*b * (1/2-hinge) ) .* up;
NW = LW .* CP - DW .* SP;
```

```
%Paralelle force (away from tip , ie tip suction);
P1 = L1  .* SP + D1 .* CP;
P2 = L2  .* SP + D2 .* CP;
P3 = L3  .* SP + D3 .* CP;
P4 = L4  .* SP + D4 .* CP;
P  = 2 * pi * rho * b * (un + dp .* b * (1/2 - hinge)) .* (un + dp * b .* (-hinge));

PW1 = R * P1 * b1r2; %this is in-plane paralelle force
PW2 = R * P2 * b2r1;
PW3 = R * P3 * b2r1;
PW4 = R * P4 * b3r0;
PW = PW1 + PW2 + PW3 + PW4;

%wing circulation per meter span
B = geom('B');
Gamma1 = 2 * pi * b * un;                          %note this is total bound gamma
Gamma2 = 2 * pi * b * b * dp * (.5 - hinge);       %not just for the upper surface
Gamma = Gamma1 + Gamma2;

if gimme(1) == 'M'
    %root moments on wing
    %moment in vertical (upwards) direction
    MX1 = L1 .* r * R; %moment per meter span
    MX2 = L2 .* r * R; %moment per meter span
    MX3 = L3 .* r * R; %moment per meter span
    MX4 = L4 .* r * R; %moment per meter span
    MX = MX1 + MX2 + MX3 + MX4; %total vertical moment per meter span

    b1r3 = geom('b1r3'); b2r2 = geom('b2r2'); b3r1 = geom('b3r1');
    MXW1 = L1*R^2*b1r3; %moment for entire wing
    MXW2 = L2*R^2*b2r2; %moment for entire wing
    MXW3 = L3*R^2*b2r2; %moment for entire wing
    MXW4 = L4*R^2*b3r1; %moment for entire wing
    MXW = MXW1 + MXW2 + MXW3 + MXW4; %total vertical moment for entire wing

    %root moments on wing
    %moment in horizontal (backwards) direction
    MY1 = D1 .* r * R; %moment per meter span
    MY2 = D2 .* r * R; %moment per meter span
    MY3 = D3 .* r * R; %moment per meter span
```

```
    MY4 = D4 .* r * R; %moment per meter span
    MY = MY1 + MY2 + MY3 + MY4; %total vertical moment per meter
        span

    b1r3 = geom('b1r3'); b2r2 = geom('b2r2'); b3r1 = geom('b3r1');
    MYW1 = D1*R^2*b1r3; %moment for entire wing
    MYW2 = D2*R^2*b2r2; %moment for entire wing
    MYW3 = D3*R^2*b2r2; %moment for entire wing
    MYW4 = D4*R^2*b3r1; %moment for entire wing
    MYW = MYW1 + MYW2 + MYW3 + MYW4; %total vertical moment for
        entire wing

    %pitching moments (this is total moment for lift& drag
        combined)
    b4r0 = geom('b4r0');

    MP1 = zeros(size(D1));
    MP2 = 2 * pi * rho * (hinge + 0.5) * b^2 * un .* uh .* CP;
    MP3 = zeros(size(D3));
    MP4 = 2 * pi * rho * dp .* (-hinge^2) .* b.^3 .* up;
    MP = MP1 + MP2 + MP3 + MP4; %pitching moment per m span
    clear a;

    %for these, require max values, ie tip velocities and b=B.
    MPW1 = MP1 * R * b2r2;
    MPW2 = MP2 * R * b3r1;
    MPW3 = MP3 * R * b3r1;
    MPW4 = MP4 * R * b4r0;
    MPW = MPW1 + MPW2 + MPW3 + MPW4; %pitching moment for entire
        wing
end

%output switch
switch gimme
case 'N'
    out = N;   if verb disp([mfilename ' returning Normal force per m span']);end
case 'P'
    out = P;   if verb disp([mfilename ' returning suction force per m span']);end
case 'NW'
    out = NW; if verb disp([mfilename ' returning Normal force for wing']);end
case 'PW'
    out = PW; if verb disp([mfilename ' returning suction force for wing']);end
case {'MYW1'}
    out = MYW1; if verb disp([mfilename ' returning horz moment for wing, part 1']);end
case {'MYW2'}
    out = MYW2; if verb disp([mfilename ' returning horz moment for wing, part 2']);end
case {'MYW3'}
    out = MYW3; if verb disp([mfilename ' returning horz moment for wing, part 3']);end
case {'MYW4'}
    out = MYW4; if verb disp([mfilename ' returning horz moment for wing, part 4']);end
case {'MYW'}
    out = MY; if verb disp([mfilename ' returning horz moment  for wing']);end
case {'MY1'}
    out = MY1; if verb disp([mfilename ' returning horz moment per span, part 1']);end
case {'MY2'}
    out = MY2; if verb disp([mfilename ' returning horz moment per span, part 2']);end
case {'MY3'}
    out = MY3; if verb disp([mfilename ' returning horz moment per span, part 3']);end
case {'MY4'}
    out = MY4; if verb disp([mfilename ' returning horz moment per span, part 4']);end
```

```
case {'MY'}
    out = MY; if verb disp([mfilename ' returning horz moment per span']);end
case {'MP2'}
    out = MP2; if verb disp([mfilename ' returning pitching moment per span, part 2']);end
case {'MP1'}
    out = MP1; if verb disp([mfilename ' returning pitching moment per span, part 1']);end
case {'MP'}
    out = MP; if verb disp([mfilename ' returning pitching moment per span']);end
case {'MPW2'}
    out = MPW2; if verb disp([mfilename ' returning pitching moment for wing, part 2']);end
case {'MPW1'}
    out = MPW1; if verb disp([mfilename ' returning pitching moment for wing, part 1']);end
case {'MPW'}
    out = MPW; if verb disp([mfilename ' returning pitching moment for wing']);end
case {'ML1','MX1'}
    out = MX1; if verb disp([mfilename ' returning upward root moment per span, part 1']);end
case {'ML2','MX2'}
    out = MX2; if verb disp([mfilename ' returning upward root moment per span, part 2']);end
case {'ML3','MX3'}
    out = MX3; if verb disp([mfilename ' returning upward root moment per span, part 3']);end
case {'ML4','MX4'}
    out = MX4; if verb disp([mfilename ' returning upward root moment per span, part 4']);end
case {'ML','MX'}
    out = MX; if verb disp([mfilename ' returning upward root moment per span']);end
case {'MLW1','MXW1'}
    out = MXW1; if verb disp([mfilename ' returning upward root moment on wing, part 1']);end
case {'MLW2','MXW2'}
    out = MXW2; if verb disp([mfilename ' returning upward root moment on wing, part 2']);end
case {'MLW3','MXW3'}
    out = MXW3; if verb disp([mfilename ' returning upward root moment on wing, part 3']);end
case {'MLW4','MXW4'}
    out = MXW4; if verb disp([mfilename ' returning upward root moment on wing, part 4']);end
case {'MLW','MXW'}
    out = MXW; if verb disp([mfilename ' returning upward root moment on wing']);end

case {'gamma','Gamma','GB','circ'}
    out = Gamma; if verb disp([mfilename ' returning bound circulation per meter span']);end
case {'ut2'}
    out = ut2mean;  if verb disp([mfilename ' returning mean square total velocity']);end
case {'CL1','CLT1','cl1'}
    out = CL1; if verb disp([mfilename ' returning lift coefficient, component 1']);end
case {'CL2','CLT2','cl2'}
    out = CL2; if verb disp([mfilename ' returning lift coefficient, component 2']);end
case {'CL3','CLR1','cl3'}
    out = CL3; if verb disp([mfilename ' returning lift coefficient, component 3']);end
case {'CL4','CLR2','cl4'}
    out = CL4; if verb disp([mfilename ' returning lift coefficient, component 4']);end
case {'CD1','CDT1','cd1'}
    out = CD1; if verb disp([mfilename ' returning drag coefficient, component 1']);end
case {'CD2','CDT2','cd2'}
    out = CD2; if verb disp([mfilename ' returning drag coefficient, component 2']);end
case {'CD3','CDR1','cd3'}
    out = CD3; if verb disp([mfilename ' returning drag coefficient, component 3']);end
case {'CD4','CDR2','cd4'}
    out = CD4; if verb disp([mfilename ' returning drag coefficient, component 4']);end
case {'CD','dragcoeff'}
    out = CD; if verb disp([mfilename ' returning drag coefficient']);end
case {'CL','liftcoeff'}
    out = CL; if verb disp([mfilename ' returning lift coefficient']);end
case {'L','lift'}
    out = L; if verb disp([mfilename ' returning lift per meter span']);end
case {'L1','lift1'}
    out = L1; if verb disp([mfilename ' returning lift per meter span, component 1']);end
case {'L2','lift2'}
    out = L2; if verb disp([mfilename ' returning lift per meter span, component 2']);end
case {'L3','lift3'}
    out = L3; if verb disp([mfilename ' returning lift per meter span, component 3']);end
case {'L4','lift4'}
    out = L4; if verb disp([mfilename ' returning lift per meter span, component 4']);end
case {'D','drag'}
    out = D; if verb disp([mfilename ' returning drag per meter span']);end
case {'D1','drag1'}
    out = D1; if verb disp([mfilename ' returning drag per meter span, component 1']);end
case {'D2','drag2'}
    out = D2; if verb disp([mfilename ' returning drag per meter span, component 2']);end
case {'D3','drag3'}
    out = D3; if verb disp([mfilename ' returning drag per meter span, component 3']);end
case {'D4','drag4'}
    out = D4; if verb disp([mfilename ' returning drag per meter span, component 4']);end
case {'LW','liftwing'}
    out = LW; if verb disp([mfilename ' returning lift for entire wing']);end
case 'LW1'
    out = LW1; if verb disp([mfilename ' returning lift for entire wing, component 1']);end
case 'LW2'
    out = LW2; if verb disp([mfilename ' returning lift for entire wing, component 2']);end
case 'LW3'
    out = LW3; if verb disp([mfilename ' returning lift for entire wing, component 3']);end
case 'LW4'
    out = LW4; if verb disp([mfilename ' returning lift for entire wing, component 4']);end
```

```
case 'DW'
    out = DW; if verb disp([mfilename ' returning drag for entire wing']);end
case 'DW1'
    out = DW1; if verb disp([mfilename ' returning drag for entire wing, component 1']);end
case 'DW2'
    out = DW2; if verb disp([mfilename ' returning drag for entire wing, component 2']);end
case 'DW3'
    out = DW3; if verb disp([mfilename ' returning drag for entire wing, component 3']);end
case 'DW4'
    out = DW4; if verb disp([mfilename ' returning drag for entire wing, component 4']);end
case 'P1'
    out = P1; if verb disp([mfilename ' returning tip suction per m span, component 1']);end
case 'P2'
    out = P2; if verb disp([mfilename ' returning tip suction per m span, component 2']);end
case 'P3'
    out = P3; if verb disp([mfilename ' returning tip suction per m span, component 3']);end
case 'P4'
    out = P4; if verb disp([mfilename ' returning tip suction per m span, component 4']);end
case 'PW1'
    out = PW1; if verb disp([mfilename ' returning tip suction for wing, component 1']);end
case 'PW2'
    out = PW2; if verb disp([mfilename ' returning tip suction for wing, component 2']);end
case 'PW3'
    out = PW3; if verb disp([mfilename ' returning tip suction for wing, component 3']);end
case 'PW4'
    out = PW4; if verb disp([mfilename ' returning tip suction for wing, component 4']);end
case 'PW'
    out = PW; if verb disp([mfilename ' returning tip suction for wing']);end
otherwise
    disp([mfilename '.m error: unknown gimme ' num2str(gimme)])
end
```

B.2.2 am

This performs the added-mass calculations of section 9.

```
function out = main(gimme,uh,duh,uv,duv,pitch,dp,ddp,r,b,hinge,verb,show);
%calculates the added mass forces.
%out = am('gimme',uh,duh,uv,duv,pitch,dp,ddp,r,b,hinge,verb,show);
%NOTE: full-wing values use tip values of uh,uv,R and maximum B, and geometry data from GEOM.M
%inputs must all be grids of values

%returns added mass lift or drag

%Created 21.2.03 by CP
%last edited 24.5.03 by CP
last_edited='24.05.03 by CBP';
last_run = date;

%parse input
switch nargin
case 0
    gimme = 'LA'; uh = kine('uht'); duh = kine('duht'); uv = kine('uvt'); duv = kine('duvt');
    pitch = kine('pitch'); dp = kine('dp'); ddp = kine('ddp');
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show = 0;
case 1
    uh = kine('uht'); duh = kine('duht'); uv = kine('uvt'); duv = kine('duvt');
    pitch = kine('pitch'); dp = kine('dp'); ddp = kine('ddp');
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show = 0;
case 2
    duh = kine('duht'); uv = kine('uvt'); duv = kine('duvt');
    pitch = kine('pitch'); dp = kine('dp'); ddp = kine('ddp');
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show = 0;
case 3
    uv = kine('uvt'); duv = kine('duvt');
    pitch = kine('pitch'); dp = kine('dp'); ddp = kine('ddp');
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show = 0;
case 4
    duv = kine('duvt');
    pitch = kine('pitch'); dp = kine('dp'); ddp = kine('ddp');
```

```
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
        = 0;
case 5
    pitch = kine('pitch'); dp = kine('dp'); ddp = kine('ddp');
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
        = 0;
case 6
    dp = kine('dp'); ddp = kine('ddp');
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
        = 0;
case 7
    ddp = kine('ddp');
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
        = 0;
case 8
    r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
        = 0;
case 9
    b = geom('B'); hinge = geom('hinge');  verb = 0; show = 0;
case 10
    hinge = geom('hinge');  verb = 0; show = 0;
case 11
    verb = 0; show = 0;
case 12
    show = 0;
case 13
    %do nothing
otherwise
    disp([mfilename ' error: too many input arguments'])
    nargin
    return
end



%load basic kinematics from datafile
SP = sin(pitch); CP = cos(pitch);
ut = abs(uh + sqrt(-1)*uv);
a = hinge;
up = uh .* CP - uv .* SP;
un =  uh .* SP + uv .* CP;

rho = kine('rho');

%Lift and drag per M span
```

```
L1 = pi * rho * b^2 * (duh .* SP .* CP + duv .* CP.^2); %dirichlet part one
L2 = pi * rho * b^2 * dp .* (2 .* up .* CP - un .* SP); %dirichlet part two
L3 = pi * rho * b^3 * (-ddp .* a .* CP + dp.^2 .* a .* SP ); % dirichlet part three
L4 = 2 * pi * rho * b^2 * (duh .* SP .* CP + duv .* CP.^2); %kutta part one
L5 = 2 * pi * rho * b^2 * dp .* (2 .* up .* CP - un .* SP); %kutta part two
L6 = 2 * pi * rho * b^3 * (a-1/2).* (-ddp .* CP + dp.^2 .* SP); % kutta part three
L = L1 + L2 + L3 + L4 + L5 + L6; %total lift
LD = L1 + L2 + L3; %Dirichlet

D1 = pi * rho * b^2 * (-duh .* SP.^2 - duv .*SP .* CP); %dirichlet part one
D2 = pi * rho * b^2 * dp .* (-2 .* up .* SP - un .* CP); % dirichlet part two
D3 = pi * rho * b^3 * (a) * (ddp .* SP + dp.^2 .* CP ); %dirichlet part three
D4 = 2 * pi * rho * b^2 * (-duh .* SP.^2 - duv .* SP .* CP); % kutta part one
D5 = 2 * pi * rho * b^2 * dp .* (-2 .* up .* SP - un .* CP); % kutta part two
D6 = 2 * pi * rho * b^3 * (a-1/2).* (ddp .* SP + dp.^2 .* CP); % kutta part three
D = D1 + D2 + D3 + D4 + D5 + D6; %total drag
DD = L1 + L2 + L3; %Dirichlet

%Wing Integrals
%only works if received velocities are tip values, and r,b are maximum values
R = geom('R');
b2r1 = geom('b2r1','t'); b3r0 = geom('b3r0','t');
LW1 = L1 * R * b2r1;
LW2 = L2 * R * b2r1;
LW3 = L3 * R * b3r0;
LW4 = L4 * R * b2r1;
LW5 = L5 * R * b2r1;
LW6 = L6 * R * b3r0;
LW = LW1 + LW2 + LW3 + LW4 + LW5 + LW6;

%drag force for entire wing
%only works if received velocities are tip values, r,b are maximum values
```

```
DW1 = D1 * R * b2r1;
DW2 = D2 * R * b2r1;
DW3 = D3 * R * b3r0;
DW4 = D4 * R * b2r1;
DW5 = D5 * R * b2r1;
DW6 = D6 * R * b3r0;
DW = DW1 + DW2 + DW3 + DW4 + DW5 + DW6;

N = L .* CP - D .* SP;
P = L .* SP + D .* CP;
NW = LW .* CP - DW .* SP;
PW = LW .* SP + DW .* CP;

%Choose output
switch gimme
case 'N'
    out = N;   if verb disp([mfilename ' returning Normal force per m span']);end
case 'P'
    out = P;   if verb disp([mfilename ' returning suction force per m span']);end
case 'NW'
    out = NW; if verb disp([mfilename ' returning Normal force for wing']);end
case 'PW'
    out = PW; if verb disp([mfilename ' returning suction force for wing']);end
case {'L','lift'}
    out = L; if verb disp([mfilename ' returning lift per meter span']);end
case {'LD'}
    out = LD; if verb disp([mfilename ' returning dirichlet lift per meter span']);end
case {'L1','lift1'}
    out = L1; if verb disp([mfilename ' returning lift per meter span, component 1']);end
case {'L2','lift2'}
    out = L2; if verb disp([mfilename ' returning lift per meter span, component 2']);end
case {'L3','lift3'}
    out = L3; if verb disp([mfilename ' returning lift per meter span, component 3']);end
case {'L4','lift4'}
    out = L4; if verb disp([mfilename ' returning lift per meter span, component 4']);end
case {'L5','lift5'}
    out = L5; if verb disp([mfilename ' returning lift per meter span, component 5']);end
case {'L6','lift6'}
    out = L6; if verb disp([mfilename ' returning lift per meter span, component 6']);end
case {'D','drag'}
    out = D; if verb disp([mfilename ' returning drag per meter span']);end
case {'DD','drag'}
    out = DD; if verb disp([mfilename ' returning dirichlet drag per meter span']);end
case {'D1','drag1'}
    out = D1; if verb disp([mfilename ' returning drag per meter span, component 1']);end
case {'D2','drag2'}
    out = D2; if verb disp([mfilename ' returning drag per meter span, component 2']);end
case {'D3','drag3'}
    out = D3; if verb disp([mfilename ' returning drag per meter span, component 3']);end
case {'D4','drag4'}
    out = D4; if verb disp([mfilename ' returning drag per meter span, component 4']);end
case {'LW','liftwing'}
    out = LW; if verb disp([mfilename ' returning lift for entire wing']);end
case 'DW'
    out = DW; if verb disp([mfilename ' returning drag for entire wing']);end
case 'P1'
    out = P1; if verb disp([mfilename ' returning tip suction per m span, component 1']);end
case 'P2'
    out = P2; if verb disp([mfilename ' returning tip suction per m span, component 2']);end
case 'N1'
    out = N1; if verb disp([mfilename ' returning normal force per m span, component 1']);end
case 'N2'
    out = N2; if verb disp([mfilename ' returning normal force per m span, component 2']);end
case 'PW1'
    out = PW1; if verb disp([mfilename ' returning tip suction for wing, component 1']);end
case 'PW2'
    out = PW2; if verb disp([mfilename ' returning tip suction for wing, component 2']);end
case 'NW1'
    out = NW1; if verb disp([mfilename ' returning normal force for wing, component 1']);end
case 'NW2'
    out = NW2; if verb disp([mfilename ' returning normal force for wing, component 2']);end
case 'LW1'
    out = LW1; if verb disp([mfilename ' returning lift for wing, component 1']);end
case 'LW2'
    out = LW2; if verb disp([mfilename ' returning lift for wing, component 2']);end
case 'LW3'
    out = LW3; if verb disp([mfilename ' returning lift for wing, component 3']);end
case 'LW4'
```

```
    out = LW4; if verb disp([mfilename ' returning lift for wing, component 4']);end
case 'LW5'
    out = LW5; if verb disp([mfilename ' returning lift for wing, component 5']);end
case 'LW6'
    out = LW6; if verb disp([mfilename ' returning lift for wing, component 6']);end
case 'DW1'
    out = DW1; if verb disp([mfilename ' returning drag for wing, component 1']);end
case 'DW2'
    out = DW2; if verb disp([mfilename ' returning drag for wing, component 2']);end
case 'DW3'
    out = DW3; if verb disp([mfilename ' returning drag for wing, component 3']);end
case 'DW4'
    out = DW4; if verb disp([mfilename ' returning drag for wing, component 4']);end
case 'DW5'
    out = DW5; if verb disp([mfilename ' returning drag for wing, component 5']);end
case 'DW6'
    out = DW6; if verb disp([mfilename ' returning drag for wing, component 6']);end
case 'PW'
    out = PW; if verb disp([mfilename ' returning tip suction for wing']);end
otherwise
    disp([mfilename '.m error: unknown gimme ' num2str(gimme)])
end
```

B.2.3 pol

This performs the Polhamus lift correction of section 10, for a single spanwise location.
Line 60 obtains the leading edge thrust from **qs**.
line 61 then forms the leading edge suction, based on the sweep.
lines 67-98 decide which way to turn the force.
lines 101-123 decides how much to scale the force by.
Finally, lines 130–135 forms the Polhamus corrections, *L_pol* and *D_pol*. These should be added to the quasi-steady forces.

```
1 function out = main(gimme,uh,uv,pitch,dp,r,b,hinge,corr,verb,show)
2 %out = pol('gimme',uh,uv,pitch,dp,r,b,hinge,corr,verb,show)
3 %calculates the polhamus correction to lift
4
5 %created 20.5.03 by C.B.Pedersen
6 last_edited='20.May.03';
7 last_run=date;
8
9 %input switch
10 switch nargin
11 case 0
12     gimme = 'LW'; uh = kine('uht'); uv = kine('uvt'); pitch = kine
           ('pitch'); dp = kine('dp');
13     r = 1; b = geom('B'); hinge = geom('hinge'); verb = 0; show
           = 0;
14 case 1
15     uh = kine('uht'); uv = kine('uvt'); pitch = kine('pitch'); dp
           = kine('dp');
16     r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
           = 0;
17 case 2
18     uv = kine('uvt'); pitch = kine('pitch'); dp = kine('dp');
19     r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
           = 0;
20 case 3
21     pitch = kine('pitch'); dp = kine('dp');
22     r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
           = 0;
23 case 4
24     dp = kine('dp');
25     r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
           = 0;
26 case 5
27     r = 1; b = geom('B'); hinge = geom('hinge');  verb = 0; show
           = 0;
```

```
case 6
    b = geom('B'); hinge = geom('hinge');  verb = 0; show = 0;
case 7
    hinge = geom('hinge');  verb = 0; show = 0;
case 8
    corr = 1; r0 = 1; verb = 0; show = 0;
case 9
    verb = 0; show = 0;
case 10
    show = 0;
case 11
    %do nothing
otherwise
    disp([mfilename ' error: too many input arguments'])
    return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHOOSE METHOD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
method = kine('polmethod'); %Note this is hard-coded
%first letter is:
%     l rotate to the side un is at the le
%     m rotate to the mean un side
%     r rotate to the rear point un side
%     c always rotate clockwise (as seen from root)
%     u always rotate so upwards
%
%second letter is:
%    t rotate entire tip suction
%    f rotate only a fraction based on the amount of chord where un
     is the same sign as unle

P = qs('P',uh,uv,pitch,dp,r,b,hinge,0,0);
S = P .* corr; %actual suction, corrected for leading edge sweep

ut2 = qs('ut2',uh,uv,pitch,dp,r,b,hinge,0,0);
B = geom('B'); r0 = geom('r0');

un = uh .* sin(pitch) + uv .* cos(pitch);
if r==0
    un = 0;
else
    un        = un * r0/r;    %un at the characterisitc point, here
        greatest chord
end
```

```
unle     = un + B * dp * (-hinge-1); %unle = normal velocity at
    leading edge
unme     = un + B * dp * (-hinge);   %unme = normal velocity at
    midpoint edge
unre     = un + B * dp * (-hinge+0.5); %unle = normal velocity at
    rear neutral point edge

turn = ones(size(uh));
switch method(1)
case 'l'
    %rotate according to leading edge normal velocity
    %always rotate enture wing
    I = find(unle<0); turn(I) = -1;
case 'm'
    %rotate according to midpoint normal velocity
    I = find(unme<0); turn(I) = -1;
case 'r'
    I = find(unre<0); turn(I) = -1;
case 'c'
    %always clockwise
    %do nothing
case 'u'
    %rotate so rotated vector is always upwards
    I = find(pitch > pi/2); %find where pitch>pi/2
    turn(I) = -1;
otherwise
    disp([mfilename '.m error: rotation method not recognised'])
    return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% decide how much to scale the suction force by
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
scale = ones(size(unle));
xo = ones(size(unle));
switch method(2)
case 't'
    if verb>3 message(toc,'using total polhamus scaling');end
    %do nothing
case 'l'
    if verb>3 message(toc,'using linear polhamus scaling');end
    %find point where sing of normal flow reverses, xo
    I = find(dp*b); %find point where this is not zero
    %at points where it is zero, scaling is one, as already set
    xo(I) = hinge - un(I)./(dp(I)*b);
```

```
    xo = min(1,xo); %if xo is off the trailing edge, set at
        trailing edge
    J = find(xo<-1); xo(J) = 1; %if xo falls off the leading edge
        , set at trailing edge
    scale(I) = (xo(I)+1)/2;
case 'd'
    disp('not yet implemented')
otherwise
    disp([mfilename '.m error rotation method not recognized'])
    return
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% modify the forces
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%force change due to the tip suction (note difference between P
    and S)


SP = sin(pitch); CP = cos(pitch);
P_pol = -P;%subtract the P suction force
N_pol = scale .* turn .* S; %but add the S suction force
L_pol = P_pol.*SP + N_pol.*CP;
D_pol = P_pol.*CP - N_pol.*SP;

switch gimme
case 'P_pol'
    out = P_pol;  if verb disp([mfilename ' returning Polhamus suction force per m span']);end
case 'N_pol'
    out = N_pol;  if verb disp([mfilename ' returning Polhamus normal force per m span']);end
case 'L_pol'
    out = L_pol;  if verb disp([mfilename ' returning Polhamus lift force per m span']);end
case 'D_pol'
    out = D_pol;  if verb disp([mfilename ' returning Polhamus drag force per m span']);end
case 'turn'
    out = turn;   if verb disp([mfilename ' returning Polhamus turn direction']);end
case 'scale'
    out = scale;  if verb disp([mfilename ' returning Polhamus scaling']);end
otherwise
    disp([mfilename '.m error: unknown gimme ' num2str(gimme)])
end
```

B.2.4 wagner

Calculates the effect of the primary wake, as described in section 11. It uses the numerical equivalent of the Duhamel integral, simply the summation of a number of discrete steps in the property. As will be shown later, the property in question in C_L. Note that depending on the value of *gimme*, it will return either the perturbation, or the total result.

```
function out = main(DA,s,gimme,verb,show);
%out = wagner(DA,s,'gimme',verb,show)
%returns the wanger effect on property A
%DA is the step change in property A at distances s
%s is the distance from the wing DA occured, normalised wrt semichords
% NOTE SEMICHORDS, NOT CHORDS
% 'gimme' governs what is returned
% 'corr' is the change in A due to wagner effect, summed for all changes
% 'loca' is as above, but for every DA individually (not summed)
% 'tota' is wagner correction _plus_ sum of steps DA.
% show, verb return figures and verbose data if set to 1.

%created on 22.11.02 CBP
%last edited 27.11.02 CBP
%last edited 20.4.03 CBP - added check for negative s
last_edited='20.Apr.03';
last_run=date;


%%%%%%%%%
%check input
switch nargin
case {0,1}
   disp([mfilename ' error: need two inputs'])
   return
case 2
   gimme = 'corr';verb = 0;show = 0;
case 3
   verb = 0; show = 0;
case 4
   show = 0;
case 5
   %do nothing
otherwise
   disp([mfilename ' error: too many inputs received'])
   return
end

```

```
if min(s)<0 disp([mfilename ' warning - bad input: distance less
    than zero']);end
%sorts the input by increasing distance s
%[s,I] = sort(s);
%DA = DA(I);
%clear I;


wag = - 0.165 * exp(-0.041 * s) - 0.335 * exp(-0.3 *s); %this is
    the wagner correction function, as approximated by jones
wag_loca = wag .* DA;
wag_corr = sum(wag_loca);
wag_tota = wag_corr + sum(DA);

switch gimme
case 'corr'
    out = wag_corr;
    if verb
        disp([mfilename ' returning total wagner correction']);
    end
case 'loca'
    out = wag_loca;
    if verb
        disp([mfilename ' returning local wagner correction']);
    end
case 'tota'
    out = wag_tota;
    if verb
        disp([mfilename ' returning total change in property with wagner correction']);
    end
otherwise
    disp([miflename ' error, bad input gimme: ' gimme])
    return
end

if show
    figure
    subplot(2,2,1)
    plot(s,'ko')
    title('distace in semichords')
    xlabel('index')
    ylabel('distance')

    subplot(2,2,2)
    plot(s,DA,'ko')
    title('step changes in property')
    xlabel('distance in semichords')
    ylabel('step change')

    subplot(2,2,3)
    plot(s,wag,'k.')
    title('wagner function at points')
    xlabel('distance in semichords')
    ylabel('wagner correction function')

    subplot(2,2,4)
    plot(s,wag_loca,'k.')
    title('wagner correction at points')
    xlabel('distance in semichords')
    ylabel('wagner correction')
end
```

B.2.5 kussner

Calculates the effect of the secondary wake, as described in section 11. It uses the numerical equivalent of the Duhamel integral, simply the summation of a number of discrete steps in the property. As will be shown later, the property in question in C_L. Note that depending on the value of *gimme*, it will return either the perturbation, or the total result.

```
function out = main(DA,s,gimme,verb,show);
%Calculation function.
%out = kussner(DA,s,'gimme',verb,show
%returns the kussner effect on property A
%DA is the step change in property A at distances s
%s is the distance from the wing DA occured, normalised wrt semichords
% NOTE SEMICHORDS, NOT CHORDS
% 'gimme' governs what is returned
% 'corr' is the change in A due to kussner effect, summed for all changes
% 'loca' is as above, but for every DA individually (not summed)
% 'tota' is kussner correction _plus_ sum of steps DA.
% show, verb return figures and verbose data if set to 1.

%created on 10.3.03 CBP
%last edited 10.3.03 CBP
last_edited='10.mar.03';
last_run=date;

%input switch
switch nargin
case {0,1}
    disp([mfilename ' error: need two inputs'])
    return
case 2
    gimme = 'corr';verb = 0;show = 0;
case 3
    verb = 0; show = 0;
case 4
    show = 0;
case 5
    %do nothing
otherwise
    disp([mfilename ' error: too many inputs received'])
    return
end

kus = - 0.5 * exp(-0.13 * s) - 0.5 * exp(-s); %this is the kussner perturbation function
```

```
kus_loca = kus .* DA;
kus_corr = sum(kus_loca);
kus_tota = kus_corr + sum(DA);

if min(s)<0 disp([mfilename 'warning - bad input: distance less
    than zero']);end
```

```
%output switch
switch gimme
case 'corr'
    out = kus_corr; if verb disp([mfilename ' returning total kussner correction']);end
case 'loca'
    out = kus_loca; if verb disp([mfilename ' returning local kussner correction']);end
case 'tota'
    out = kus_tota; if verb disp([mfilename ' returning total change in property with kussner correction']);end
otherwise
    disp([miflename ' error, bad input gimme: ' gimme])
    return
end
```

B.3 Master Functions

B.3.1 master_qsam and numerical_qsam

These use **qs** and **am** to calculate the quasi-steady and added-mass forces on the wing. **master_qsam** uses wing shape parameters to find the total lift for the wing. **numerical_qsam** uses numerical summation across the radial stations.

master_qsam

Lines 60–63 are the vertical, horizontal, normal and parallel added mass forces, respectively.

```
1 function main(path,verb,show,fast)
2 %calculates the forces and moments on the wing using wing shape
     parameters.
3 %the kinematics and geometry are read from the functions geom.m
     and kine.m as needed
4 %all functions are documented by typing "help functionname"
5
6 %created 10.6.02 by C.B.Pedersen
7 %last edited 12.3.03 BY CBP.
8 last_edited='12.Mar.03';
9 last_run=date;
10
11
12 %parse the input
13 switch nargin
14 case 0
15     disp([mfilename 'error: the path for rundata must be specified
          '])
16     return
17 case 1
18     verb = 0;
19     show = 0;
20     fast = 0;
21 case 2
22     show = 0;
23     fast = 0;
24 case 3
25     fast = 0;
26 case 4
27     %do nothing
28 otherwise
29     disp([mfilename ' error, too many input arguments'])
30 end
31 save temp verb show fast
32
```

```
if verb timegone = toc; disp([num2str(round(timegone)) ' Quasi steady calculation']);end
id = 'qs_1_quasi';
if fast
    load([path id])
    load temp verb show fast;
    if verb disp('skipped, loading data from file');end
else
    LW = qs('LW');          %quasi steady vertical force
    DW = qs('DW');          %quasi steady horizontal force
    FW = LW + sqrt(-1)*DW;       %total force (complex)


save([path id])
end
if verb timegone = toc; disp([num2str(round(timegone)) ' Done']); disp(' ');end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Added mass contribution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb timegone = toc; disp([num2str(round(timegone)) ' Added mass']);end
id = 'qs_2_addm';
if fast>1
    load([path id])
    load temp verb show fast;
    if verb disp('skipped, loading data from file');end
else
    LA = am('LW');
    DA = am('DW');
    NA = am('NW');
    PA = am('PW');

    save([path id])
end
if verb timegone = toc; disp([num2str(round(timegone)) ' Done']); disp(' ' );end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Moments
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb timegone = toc; disp([num2str(round(timegone)) ' Moments' ]);end
id = 'qs_3_moments';
```

```
if fast>2
    load([path id])
    load temp verb show fast;
    if verb disp('skipped, loading data from file');end
else
    MX = qs('MXW');        %vertical moment
    MZ = qs('MYW');        %horizontal moment
    MY = qs('MPW'); %pitch moment, am disabled

save([path id])
end
if verb timegone = toc; disp([num2str(round(timegone)) ' Done']);
   disp(' ');end

if verb
    timegone = toc;
    disp(['completed in ' num2str(round(timegone)) ' seconds'])
end

id = 'qs_final';
save([path id])
return
```

numerical_qsam

Lines 51–56 and 80–85 are the summation across the wing.

```
1 function main(path,verb,show,fast)
2 %numerical_qsam(path,verb,show,fast)
3 %calculates the forces and moments on the wing using numerical
     integration
4 %the kinematics and geometry are read from the functions geom.m
     and kine.m as needed
5
6 %created 13.Apr.03 by C.B.Pedersen
7 last_edited='13.Apr.03';
8 last_run=date;
9
10 %parse the input
11 switch nargin
12 case 0
13     disp([mfilename 'error: the path for rundata must be specified
          '])
14     return
15 case 1
16     verb = 0;
17     show = 0;
18     fast = 0;
19 case 2
20     show = 0;
21     fast = 0;
22 case 3
23     fast = 0;
24 case 4
25     %do nothing
26 otherwise
27     disp([mfilename ' error, too many input arguments'])
28 end
29 save temp verb show fast
30
31 if verb timegone = toc; disp([num2str(round(timegone)) ' Quasi
     steady calculation']);end
32 id = 'qs_1_quasi';
33 if fast
34     load([path id])
35     load temp verb show fast;
36     if verb disp('skipped, loading data from file');end
37 else
38     R = geom('R'); r = geom('r_default'); hinge = geom('hinge','t'
          ,r); dr = geom('dr'); nr = length(r);
```

```
    uht = kine('uht'); uvt = kine('uvt'); pitch = kine('pitch'); dp = kine('dp'); b = geom('b','t',r);
    t = kine('t'); nt = length(t);
    if length(hinge) == 1
        hinge = ones(1,nr)*hinge;
    end

    for ri=1:nr;
        L(ri,:)   = qs('L',uht*r(ri),uvt*r(ri),pitch,dp,r(ri),b(ri),hinge(ri),0,0); %lift per span at each station
        D(ri,:)   = qs('D',uht*r(ri),uvt*r(ri),pitch,dp,r(ri),b(ri),hinge(ri),0,0);
        P(ri,:)   = qs('P',uht*r(ri),uvt*r(ri),pitch,dp,r(ri),b(ri),hinge(ri),0,0);
        N(ri,:)   = qs('N',uht*r(ri),uvt*r(ri),pitch,dp,r(ri),b(ri),hinge(ri),0,0);
    end
    for ti=1:nt
        LW(ti) = sum(L(:,ti).*dr')*R;
        DW(ti) = sum(D(:,ti).*dr')*R;
        PW(ti) = sum(P(:,ti).*dr')*R;
        NW(ti) = sum(N(:,ti).*dr')*R;
    end

save([path id])
end
if verb timegone = toc; disp([num2str(round(timegone)) ' Done']); disp(' ');end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Added mass contribution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb timegone = toc; disp([num2str(round(timegone)) ' Added mass']);end
id = 'qs_2_addm';
if fast>1
    load([path id])
    load temp verb show fast;
    if verb disp('skipped, loading data from file');end
else
    duht = kine('duht'); duvt = kine('duvt'); ddp = kine('ddp');
    for ri=1:nr; %force per m span
        LAL(ri,:)  = am('L',uht*r(ri),duht*r(ri),uvt*r(ri),duvt*r(ri),pitch,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
        LALD(ri,:) = am('LD',uht*r(ri),duht*r(ri),uvt*r(ri),duvt*r(ri),pitch,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
```

```
        PAL(ri ,:)    = am('P' ,uht*r(ri),duht*r(ri),uvt*r(ri),duvt*r
            (ri),pitch ,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
        NAL(ri ,:)    = am('N' ,uht*r(ri),duht*r(ri),uvt*r(ri),duvt*r
            (ri),pitch ,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
        DAL(ri ,:)    =  am('D' ,uht*r(ri),duht*r(ri),uvt*r(ri),duvt*r
            (ri),pitch ,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
    end
    for ti=1:nt
        LA(ti) = sum(LAL(:, ti).*dr')*R;
        LAD(ti)= sum(LALD(:, ti).*dr')*R;
        DA(ti) = sum(DAL(:, ti).*dr')*R;
        PA(ti) = sum(PAL(:, ti).*dr')*R;
        NA(ti) = sum(NAL(:, ti).*dr')*R;
    end
save([path id])
end
if verb timegone = toc; disp([num2str(round(timegone)) ' Done']);
   disp(' ');end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Moments
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb timegone = toc; disp([num2str(round(timegone)) ' Moments'
   ]);end
id = 'qs_3_moments';
if fast>2
    load([path id])
    load temp verb show fast;
    if verb disp('skipped, loading data from file');end
else
    %Note moments have been disabled for speed
    %for ri=1:nr
        %MQ1(ri ,:) =   qs('MY1',uht*r(ri),uvt*r(ri),pitch ,dp,r(ri),
            b(ri),hinge(ri),0,0); %pitching moment
        %MQ2(ri ,:) =   qs('MY2',uht*r(ri),uvt*r(ri),pitch ,dp,r(ri),
            b(ri),hinge(ri),0,0); %pitching moment
        %MQ3(ri ,:) =   qs('MY3',uht*r(ri),uvt*r(ri),pitch ,dp,r(ri),
            b(ri),hinge(ri),0,0); %pitching moment
        %MQ4(ri ,:) =   qs('MY4',uht*r(ri),uvt*r(ri),pitch ,dp,r(ri),
            b(ri),hinge(ri),0,0); %pitching moment
        %MQ(ri ,:) =   qs('MY',uht*r(ri),uvt*r(ri),pitch ,dp,r(ri),b
            (ri),hinge(ri),0,0); %pitching moment

        %MA1(ri ,:) = am('MP1',upt*r(ri),dupt*r(ri),unt*r(ri),dunt*
            r(ri),pitch ,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
```

```
        %MA2(ri,:) = am('MP2',upt*r(ri),dupt*r(ri),unt*r(ri),dunt*
            r(ri),pitch,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
        %MA3(ri,:) = am('MP3',upt*r(ri),dupt*r(ri),unt*r(ri),dunt*
            r(ri),pitch,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
        %MA4(ri,:) = am('MP4',upt*r(ri),dupt*r(ri),unt*r(ri),dunt*
            r(ri),pitch,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
        %MA5(ri,:) = am('MP5',upt*r(ri),dupt*r(ri),unt*r(ri),dunt*
            r(ri),pitch,dp,ddp,r(ri),b(ri),hinge(ri),0,0);
        %MA(ri,:) = am('MP',upt*r(ri),dupt*r(ri),unt*r(ri),dunt*r
            (ri),pitch,dp,ddp,r(ri),b(ri),hinge(ri),0,0);

        %end
    %MP = MQ; %+ MA; added mass disabled    %pitch moment

    %for ti=1:nt
     %   MQW(ti) = sum(MQ(:,ti).*dr)*R;
        %MAW(ti) = sum(MA(:,ti).*dr)*R;
        %end
    %MPW = MQW; %added mass disabled + MAW;

save([path id])
end
if verb timegone = toc; disp([num2str(round(timegone)) ' Done']);
    disp(' ');end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOTTING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if show
    figure
    plot([kine('tt') kine('tt')+2*pi],[LW LW]) %plots lift across
        to full strokes, to check for end effects
    title('lift across two full cycles, to check for end effects')



    figure                  %plots some sample locations and force
        vectors
    showme = [1800 10 300 500 800 1023];
    dummy = show_kine('2d',showme,[DW(showme) ; LW(showme)]);
    title('some sample force vectors')

    figure
    subplot(2,2,1)
    plot(kine('tt'),LA)
    title('added mass - vert')
```

```
    subplot(2,2,2)
    plot(kine('tt'),DA)
    title('added mass -horz')

    subplot(2,2,3)
    plot(kine('tt'),abs(LA+sqrt(-1)*DA))
    title('added mass - total')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if verb
    timegone = toc;
    disp(['completed in ' num2str(round(timegone)) ' seconds'])
end

id = 'qs_final';
save([path id])
return
```

B.3.2 master_polhamus and numerical_pol

These calculate the Polhamus correction, as described in section 10. **master_pol** uses wing shape parameters to calculate the total force on the wing, while **numerical_pol** uses numerical summation.

master_polhamus

This is very similar to the calculation function **pol**. The functionality of **pol** is reproduced here, because it has been used for extensive testing of other means of calculating the Polhamus correction, which have not always lent themselves easily to a function call.

```
function main(path,verb,show,fast)
%master_polhamus(path,verb,show,fast,method)
%calculates the polhamus correction to lift
%path is the path where data will be saved
%verb is verbosity level (0 = quiet)
%show is amount of figures to plot (0=none, 1= some, 2+ all)
%fast skips part of the calculation by using data from previous
    runs

%created 1.4.03 by C.B.Pedersen
%last edited 1.6.03 BY CBP.
last_edited='1.Jun.03';
last_run=date;

%parse the input
switch nargin
case 0
    disp([mfilename ' error: the path for rundata must be
        specified'])
    return
case 1
    verb = 0; show = 0; fast = 0; method = 'unle';
case 2
    show = 0;    fast = 0;
case 3
    fast = 0;
case 4
    %do nothing
otherwise
    disp([mfilename ' error, too many input arguments'])
end
save temp verb show fast

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHOOSE METHOD
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
method = kine('polmethod');
%first letter is:
%    l rotate to the side un is at the le
%    m rotate to the mean un side
%    r rotate to the rear point un side
%    c always rotate clockwise (as seen from root)
%    u always rotate so upwards
%
%second letter is:
%   t rotate entire tip suction
%   f rotate only a fraction based on the amount of chord where un is the same sign as unle
%   d die during rotation.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% QUASI-STEADY DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb message(toc,'Quasi-steady data');end
    %timegone = toc; disp([num2str(round(timegone)) ' Quasi-steady data']);end
id = 'pol_1_qsdata';
if fast>0
    load([path id])
    load temp verb show fast;
    if verb disp('skipped, loading data from file');end
else
    %Load results from the quasi-steady calculation
    pitch = kine('pitch'); dp = kine('dp');
    r = geom('r_default'); b = geom('b','t',r); R = geom('R');
    hinge = geom('hinge'); dledr = geom('dledr','t',r);
    unt = kine('unt'); uht = kine('uht'); uvt = kine('uvt');
    nr = length(r); nt = length(pitch);
    %bsic geometric results - these can take some time
    if verb message(toc,'Geometric data (can take a while)');end
    b1r2P = geom('b1r2P'); b1r2 = geom('b1r2');%note the difference in geom call
    b2r1P = geom('b2r1P'); b2r1 = geom('b2r1');
    b3r0P = geom('b3r0P'); b3r0 = geom('b3r0');
    b3r1P = geom('b3r0P'); b3r1 = geom('b3r1');
    b4r0P = geom('b4r0P'); b4r0 = geom('b4r0');
    if verb message(toc,'Geometric data got');end

    %max suction forces
    P1 = qs('P1');
    P2 = qs('P2');
```

```
    P3 = qs('P3');
    P4 = qs('P4');


    PW1 = R * P1 * b1r2;
    PW2 = R * P2 * b2r1;
    PW3 = R * P3 * b2r1;
    PW4 = R * P4 * b3r0;
    PW = PW1 + PW2 + PW3 + PW4; %paralelle force

    SW1 = R * P1 * b1r2P;
    SW2 = R * P2 * b2r1P;
    SW3 = R * P3 * b2r1P;
    SW4 = R * P4 * b3r0P;
    SW  = SW1 + SW2 + SW3 + SW4; %suction force (higher because of Le sweep)

    if show
        figure
        plot(PW2)
        hold on
        plot(SW2,'k')
        plot(0,0,'kx')
    end
    pol_ratio = SW./PW; %the amount sweep increases suction by

save test_data
save([path id])
end
if verb message(toc,'Done');disp(' ');end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% decide which way to turn
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb message(toc,'Choosing turn direction');end
    %timegone = toc; disp([num2str(round(timegone)) ' Quasi-steady data ']);end
id = 'pol_2_turn_direction';
if fast>1
    load([path id])
    load temp verb show fast;
    if verb disp('skipped, loading data from file');end
else
    turn = ones(1,nt); %all rotating clockwise initially
    %rotate all lift the same way.
    %use greatest chord point
```

```
    r0 = geom('r0'); B = geom('B');%radius where chord is maximum
    un      = unt * r0;
    unle    = un + B * dp * (-hinge-1); %unle = normal velocity at leading edge
    unme    = un + B * dp * (-hinge);   %unme = normal velocity at midpoint edge
    unre    = un + B * dp * (-hinge+0.5); %unle = normal velocity at rear neutral point edge

    switch method(1)
    case 'l'
        %rotate according to leading edge normal velocity
        %always rotate enture wing
        I = find(unle<0); turn(I) = -1;
    case 'm'
        %rotate according to midpoint normal velocity
        I = find(unme<0); turn(I) = -1;
        %if 2d mesh
        %[I,J] = find(unme<0);
        %for i=1:length(I)
        %    turn(I(i),J(i)) = -1;
        %end
    case 'r'
        I = find(unre<0); turn(I) = -1;
        %rotate according to rear neutral normal velocity
        %[I,J] = find(unre<0);
        %for i=1:length(I)
        %    turn(I(i),J(i)) = -1;
        %end
    case 'c'
        %always clockwise
        %do nothing
    case 'u'
        %rotate so rotated vector is always upwards
        I = find(pitch > pi/2); %find where pitch>pi/2
        turn(I) = -1;
        %for i=1:length(I)
        %    turn(:,I(i)) = -1;
        %end
    otherwise
        disp([mfilename '.m error: rotation method not recognised'])
        return
    end
save([path id])
end
```

```
if verb message(toc,'Done');disp(' ');end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% decide how much to scale the suction force by
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
scale = ones(size(unle));
switch method(2)
case 't'
    %do nothing
case 'f'
    disp('not yet implemented')
case 'd'
    disp('not yet implemented')
otherwise
    disp([mfilename '.m error rotation method not recognized'])
    return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% modify the forces
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%force change due to the tip suction (note difference between P
    and S)
%have to do radial stepping, or turn isn't a consistent vecotur



    SP = sin(pitch); CP = cos(pitch);
    P_pol = -PW;%subtract the P suction force
    N_pol = scale .* turn .* SW; %but add the S suction force
    L_pol = P_pol.*SP + N_pol.*CP;
    D_pol = P_pol.*CP - N_pol.*SP;
```

numerical_pol

```
function main(path,verb,show,fast)
%numerical_polhamus(path,verb,show,fast,method)
%calculates the polhamus correction to lift
%path is the path where data will be saved
%verb is verbosity level (0 = quiet)
%show is amount of figures to plot (0=none, 1= some, 2+ all)
%fast skips part of the calculation by using data from previous runs

%created 1.4.03 by C.B.Pedersen
%last edited 1.4.03 BY CBP.
last_edited='1.Apr.03';
last_run=date;

%input switch
switch nargin
case 0
    disp([mfilename ' error: the path for rundata must be specified'])
    return
case 1
    verb = 0; show = 0; fast = 0; method = 'unle';
case 2
    show = 0;    fast = 0;
case 3
    fast = 0;
case 4
    %do nothing
otherwise
    disp([mfilename ' error, too many input arguments'])
end
save temp verb show fast

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHOOSE METHOD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
method = kine('polmethod');
%first letter is:
%     l rotate to the side un is at the le
%     m rotate to the mean un side
%     r rotate to the rear point un side
%     c always rotate clockwise (as seen from root)
%     u always rotate so upwards
%
```

```
%second letter is:
%    t rotate entire tip suction
%    f rotate only a fraction based on the amount of chord where un is the same sign as unle
%    d die during rotation.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% QUASI-STEADY DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb message(toc,'Quasi-steady data');end
    %timegone = toc; disp([num2str(round(timegone)) ' Quasi-steady data ']);end
id = 'pol_1_qsdata';
if fast>0
    load([path id])
    load temp verb show fast;
    if verb message(toc,'skipped, loading data from file');end
else
    %Load results from the quasi-steady calculation
    pitch = kine('pitch'); dp = kine('dp');
    r = geom('r_default'); dr = geom('dr'); b = geom('b','t',r); R = geom('R');
    hinge = geom('hinge'); dledr = geom('dledr','t',r);
    unt = kine('unt'); uht = kine('uht'); uvt = kine('uvt');
    nr = length(r); nt = length(pitch);
    upt = kine('upt'); dupt = kine('dupt'); unt = kine('unt'); dunt = kine('dunt');
    if length(hinge) == 1
        hinge = ones(1,nr)*hinge;
    end

    if verb message(toc,'calculating suction force');end

    %suction forces
    dledr = geom('dledr','t',r); corr = sqrt(1+dledr.^2);
     for ri=1:nr
        if verb>1 message(toc,['radial position ' num2str(ri)]); end
        L_pol(ri,:) = pol('L_pol',uht*r(ri),uvt*r(ri),pitch,dp,r(ri),b(ri),hinge(ri),corr(ri),0,0);
        D_pol(ri,:) = pol('D_pol',uht*r(ri),uvt*r(ri),pitch,dp,r(ri),b(ri),hinge(ri),corr(ri),0,0);
        N_pol(ri,:) = pol('N_pol',uht*r(ri),uvt*r(ri),pitch,dp,r(ri),b(ri),hinge(ri),corr(ri),0,0);
        P_pol(ri,:) = pol('P_pol',uht*r(ri),uvt*r(ri),pitch,dp,r(ri),b(ri),hinge(ri),corr(ri),0,0);
```

```
        scale(ri ,:) = pol('scale',uht*r(ri),uvt*r(ri),pitch,dp,r(
           ri),b(ri),hinge(ri),corr(ri),0,0);
        turn(ri ,:)  =  pol('turn',uht*r(ri),uvt*r(ri),pitch,dp,r(
           ri),b(ri),hinge(ri),corr(ri),0,0);
    end

    for ti = 1:nt
         PW(ti) = sum(P_pol(:, ti).*dr')*R;
         NW(ti) = sum(N_pol(:, ti).*dr')*R;
         LW(ti) = sum(L_pol(:, ti).*dr')*R;
         DW(ti) = sum(D_pol(:, ti).*dr')*R;

     end

    _pol_ratio = abs(NW./PW); %the amount le sweep increases
        suction by

save([path id])

end
if verb message(toc,'Done');disp(' ');end

save([path 'pol_final'])
```

B.3.3 master_wag

This masterfile calculates the effect of the primary wake, modelling it using the Wagner function. It divides into four main subroutines:

- **wag_1_init**, which initializes data, mainly by reading it from **kine** and **geom**
- **wag_2_liftcoeff**, which forms the lift coefficient
- **wag_3_wagner_effect**, which splits the wake into individual stroke segments, and applies the Wagner function to them
- **wag_4_correct_lift**, which turns the Wagner-modified lift coefficients into full force values

wag_1_init

Here, the function loads the kinematic and geometric data. Note also that it loads a number of runtime parameters from **kine**. These will be explained when they are used in the code.

wag_2_liftcoeff

line 74 is the start of a radial stepping loop, that persists until line 111. It performs the following calculations at each spanwise station:
lines 75–78 simply check if r or b are 0 - if so, there will be no lift or Wagner effect, and the results are forcibly set to 0, rather than calculated. This is done to avoid divide by 0 errors.
line 81 calls **qs** to return *ute2m*, the mean square velocity at that spanwise station. This is put in an *nr*-by-*nt* matrix for later reference.
line 82 calls **qs** again, this time to return C_L, the vertical force coefficient. This is based on the vertical force F_V divided by *rho b ute2m*. This is the "lift coeffience" that will be used throughout the following. It is placed in a *nr*-by-*nt*, as above. lines 83–95 deal with the Polhamus correction to the lift coefficient, firstly checking the runtime parameter *usepolhamus* to see if it should be used. If it should, it gets the Polhamus lift correction by calling **pol**, , forms the lift coefficient for this correction, and adds it to the original lift coefficient. lines 96–108 deals with *Dcl*, the step change in lift coefficient. This is formed by calling our function **der**, which is similar to the inbuilt function **diff**, except that is returns a full-length differential vector, by assuming the data can be wrapped around - so the first value of *Dcl* is the step increase from the last value of *cl* to the first. This is the case for when our data forms a full cycle. There are two exceptions to this case, which are checked by the variable *firststep*. If the data represents an impulsive start, the first step will be the first value of *cl*. Alternatively, if the data is supposed to close, but doesn't, we can force the first step to be smooth, by setting it equal to the second step. This can occur either because of measuring noise, or sampling rate mismatch with the flapping frequency, so the data isn't exactly a full cycle.

wag_3_wagner_effect

Again, we perform the calculations for each spanwise section separately - line 129 starts a spanwise stepping loop that persist until line 215.
Lines 132–134 simply checks to see if the radius is 0, and forces the results to be 0 if it is.
Lines 136–143 calculate *Ddist* the distance travelled for every timestep, and *rev* the vector of reversal points (zero everywhere apart from one point at the reversal). It uses the variable *tailflag* to decide wether to use the velocity of the trailing or hinge location for this.
Line 146 creates *nrev*, which is simply the index of the points where reversal occurs.
Lines 148–155 checks if the data is wrappable via *firststep*, and if it is, shifts the data so the first reversal point is at the start of the vector.
Lines 157–165 corrects *nrev* so it has a leading value of 1 and at tailing value of *nt*. This is done for splitting the data into strokes.
Lines169–175 splits the data into single-stroke segments, counts through the strokes, and calculates *dist*, the distance travelled within the current stroke only. Note that it is absolute, and increases from 0 at the start of each stroke.
Lines 182–214 then apply the Wagner function to each individual stroke. for this, we need to use timewise stepping (line 186) through all the timesteps of the current stroke. Line 188 forms *distw*, which is the distance to each point in the current wake, previous to the current timestep. This is always positive, with the value for the current timestep being 0. (the distance to the wake element that has just been shed is 0). Lines 191–192 call **wagner** to return the Wagner perturbation correction, as a vector of all the contributions of points in the wake, and sums these for a total perturbation correction, *wag*. Note this could be done as a single call to **wagner**, with the *gimme* flag set to *tota*. However, in lines 194–210 we plot the contributions of each point in the wake, if desired. This is partly for error-checking, but also for additional insight, as seen in section 15.

Note that although we split the data into single stroke segments, and treat the start of each stroke as an impulsive start, in that we remove all accumulated Wagner contributions for the previous wake, we treat the step change in lift coefficient as continuous (i.e. we do not set the value of *Dcl* at the start of every stroke to the value of *cl* at the start of the stroke, but rather use the change from the end of the last stroke). This is because doing so would introduce a large, discrete step change in life coefficient that does not match the reality we are modelling. Although we spilt the lift coefficient into strokes for calculation purposes, it is in fact continuous.

Lines 225–233 checks to see if the data was shifted in lines 148–155, and if is was, shifts it back to the original timesteps.
Lines 273–296 re-creates the full forces from the coefficients, by multiplying the coefficients by *den*. Note that unlike the calculation that created *cl* above, this uses the actual area of each segments to create the lift per segment, not lift per m span. These are then summed to create the full lift values for the wing.
Finally, line 331 highlights an important limitation of the code: the predicted drag effect of the primary wake is 0.

```
function main(path,verb,show,fast)
%master_wagner(path,verb,show,fast)
%caluclates the wagner effect

%created 10.6.02 by C.B.Pedersen
%last edited 12.3.03 BY CBP.
%last edited 19.4.03 by CBP - changed Wagner to be faster.
%last edited 21.4.03 by CBP - now use qs_cl to get lift coefficient
%last edited 15.5.03 by CBP - refine first DCLDS point
%last edited 16.5.03 by CBP - added chopping into multiple strokes
last_edited='16.May.03';
last_run=date;

%input switch
switch nargin
case 0
    disp([mfilename 'error: the path for rundata must be specified '])
    return
case 1
    verb = 0;
    show = 0;
    fast = 0;
case 2
    show = 0;
    fast = 0;
case 3
    fast = 0;
case 4
    %do nothing
otherwise
    disp([mfilename ' error, too many input arguments'])
end
save temp verb show fast

if verb timegone = toc; disp([num2str(round(timegone)) ' Data initialisation']);end
id = 'wag_1_init';
if fast
    load([path id])
    load temp verb show fast;
    if verb disp('skipped, loading data from file');end
else
    tshow = kine('tshow'); rshow = kine('rshow');
    tailflag = kine('tailflag'); %wether to calculate reversal
```

```
    based on tail or hinge position
    datalength = kine('datalength'); %wether the data is a full , half or other part of a cycle
    firststep = kine('firststep'); %method to use on the first dCL /dt step
    usepolhamus = kine('usepolhamus'); %wether to adjust cl for polhamus when calculating wake effect
    nt = kine('nt'); r = geom('r_default','t'); nr = length(r); tt = kine('tt');
    b = geom('b','t',r); rho = kine('rho'); ut = kine('ut');
    R = geom('R'); upt = kine('upt'); unt = kine('unt'); dp = kine ('dp');
    hinge = geom('hinge');
    if length(hinge) == 1
        hinge = ones(1,nr)*hinge;
    end

save([path id])
end
if verb message(toc,' Done');disp(' ');end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Form lift coefficient
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb message(toc,' Lift coefficient');end
id = 'wag_2_liftcoeff';
if fast>1
    load([path id])
    load temp verb show fast;
    if verb message(toc,'skipped , loading data from file');end
else
    uht = kine('uht');
    uvt = kine('uvt');
    pitch = kine('pitch');
    dp = kine('dp');
    for ri=1:nr
        if b(ri) ==0 | r(ri) == 0
            cl(ri,1:nt) = zeros(1,nt);
            Dcl(ri,1:nt) = zeros(1,nt);
            ute2m(ri,:) = zeros(1,nt);
        else
            uh(ri,:) = uht .* r(ri);
            uv(ri,:) = uvt .* r(ri);
            ute2m(ri,:) = qs('ut2',uh(ri,:),uv(ri,:),pitch,dp,r(ri),b(ri),hinge(ri),0,0);
            cl(ri,:) = qs('CL',uh(ri,:),uv(ri,:),pitch,dp,r(ri),b(
```

```
ri),hinge(ri),0,0);
            switch usepolhamus
            case 'y'
                if verb & ri==2 message(toc,'using polhamus correction to cl');end
                dledr = geom('dledr','t',r); corr = sqrt(1+dledr.^2);
                L_pol(ri,:) = pol('L_pol',uh(ri,:),uv(ri,:),pitch,dp,r(ri),b(ri),hinge(ri),corr(ri),0,0);
                clp(ri,:) = L_pol(ri,:)./(rho * b(ri) .* ute2m(ri,:));
                cl(ri,:) = cl(ri,:) + clp(ri,:);
            case 'n'
                %do nothing
            otherwise
                disp('bad value received for usepolhamus')
                return
            end
            Dcl(ri,:) = der(cl(ri,:),1:nt);
            %need to change the first value based on what data we are using
            switch firststep
            case 'w'
                %wrap data - already calculated above
            case 'i'
                Dcl(ri,1) = cl(ri,1); %impulsive start
            case 's'
                Dcl(ri,1) = Dcl(ri,2); %smooth by setting equal to second step
            otherwise
                message(toc,['bad value for firststep: ' firststep]);
                return
            end
        end
    end
save([path id])
end
if verb message(toc,' Done');disp(' ');end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%wagner effect
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if verb message(toc,' Wagner effect');end
```

```
id = 'wag_3_wagner_effect';
if fast>2
    load([path id])
    load temp verb show fast;
    if verb message(toc,'skipped, loading data from file');end
else
    %find the distance travelled per timestep
    uht = kine('uht'); dt = kine('dt');
    for ri=1:nr
        if verb>1 message(toc,['now at radial index ' num2str(ri)]);end
        if r(ri)==0
            Ddist(ri,:) = zeros(1,nt);
            rev(ri,:) = zeros(1,nt);
            shift(ri) = 0;
        else
            if tailflag %wether to base reversals and distance travelled on hinge or trailing edge
                uhte(ri,:) = uht*r(ri) + dp * b(ri) * (+1 -hinge(ri)) .* sin(pitch); %velocity at trailing edge
                Ddist(ri,:) = abs(uhte(ri,:) .* dt);
                rev(ri,:) = find_crossings('vector',uhte(ri,:));
            else
                Ddist(ri,:) = abs(uht * r(ri)) .* dt;
                rev(ri,:) = find_crossings('vector',uht);
            end

            %create reversal points for this radial position
            nrev = find(rev(ri,:));

            if firststep == 'w';
                shift(ri) = nrev(1); %amount to shift data by so first reversal is at index 1
                if verb > 1 message(toc,['shifting data by ' num2str(shift(ri)-1)]);end
                %shifts the data so the first reversal point is a index 1
                Ddist(ri,:) = shifter(Ddist(ri,:),shift(ri));
                Dcl(ri,:) = shifter(Dcl(ri,:),shift(ri));
                rev(ri,:)   = shifter(rev(ri,:),shift(ri));
            end

            if nrev(1) ~=1
                nrev = [1 nrev];
            end

```

```
        lrev = length(nrev);
        if nrev(lrev) ~=nt;
            nrev = [nrev nt];
        end
        clear lrev;

        %split into strokes

        for stroke=1:length(nrev)-1
            t1 = nrev(stroke); %start index of this stroke
            t2 = nrev(stroke+1)-1; %end index of this stroke
            for ti=t1:t2;
                %distance travelled in this stroke
                dist(ri,ti) = sum(Ddist(ri,t1:ti));
            end
        end
    end

    if b(ri)==0 | r(ri) == 0
        wag(ri,1:nt) = zeros(1,nt);
    else
     for stroke = 1:length(nrev)-1
      t1 = nrev(stroke); %start index of this stroke
      t2 = nrev(stroke+1)-1; %end index of this stroke
     %Dcl(ri,t1) = 0;
      for ti=t1:t2;
       %steps through time this stroke
       distw = dist(ri,ti)-dist(ri,t1:ti); %distance to points in wake, not distance travelled

       if verb >2 & ti/100 == floor(ti/100); timegone = toc; disp([num2str(round(timegone)) 'time step ' num2str(ti)]);end
       wag_loca = wagner(Dcl(ri,t1:ti),distw/b(ri),'loca',0,0); %wagner correction for every point in the wake
       wag(ri,ti) = sum(wag_loca); %correction only, summation of all contributions for this stroke

       if show & ti == tshow & ri == rshow
        figure
        subplot 221
        plot(Dcl(ri,1:ti));
        title('Delta(cl)')

        subplot 222
        plot(wag_loca)
```

```
            title('wagner correftion per timestep')

            subplot 223
            plot(wag(ri,1:ti))
            title('total wagner correction')
            xlabel('date')

            subplot 224
            xlabel(['Radial station ' num2str(ri)])
          end
        end
      end
    end
  end

save([path id])
end
if verb message(toc,' Done');disp(' ');end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Shift back to original time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if shift
    if verb message(toc,'un-shifting data');end
    for ri=1:nr
        Ddist(ri,:) = shifter(Ddist(ri,:),-shift(ri));
        dist(ri,:)  = shifter(dist(ri,:),-shift(ri));
        Dcl(ri,:) = shifter(Dcl(ri,:),-shift(ri));
        wag(ri,:)   = shifter(wag(ri,:),-shift(ri));
    end
end

%Show results
if show
    figure
    subplot 221
    surf(tt,r,wag);
    shading interp
    axis([0 2*pi 0 1 min(min(cl)) max(max(cl))])
    title('wagner correction coefficient')
    %view([0 0 1])
    colorbar

    subplot 222
    surf(tt,r,cl);
```

```
    axis([0 2*pi 0 1 min(min(cl)) max(max(cl))])
    title('original quasi steady lift coefficient')
    shading interp
    %view([0 0 1])
    colorbar

    subplot 223
    surf(tt,r,cl+wag);
    axis([0 2*pi 0 1 min(min(cl)) max(max(cl))])
    title('wagner-compensated lift coefficient')
    shading interp
    %view([0 0 1])
    colorbar

    subplot 224
    surf(tt,r,cl+wag);
    axis([0 2*pi 0 1 min(min(cl)) max(max(cl))])
    title('wagner-compensated lift coefficient')
    shading interp
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Turn from coefficients into full values again
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if verb message(toc,' Correcting lift');end
id = 'wag_4_Correct_Lift';
if fast>3
    load([path id])
    load temp verb show fast;
    if verb disp('skipped, loading data from file');end
else
    %turn from coefficients into full values
    dr = geom('dr');
    da = dr .* R * 2 .* b; %area of each spanwise segment;
    for ri=1:nr
        den(ri,:) = .5 * rho * ute2m(ri,:) .* da(ri);
    end
    lds = cl.*den; %original lift
    lw = wag.*den; %wagner correction
    lqw = (wag + cl).*den; %wagner corrected lift

    for i=1:nt
        L(i) = sum(lds(:,i));   %total QS force on entire wing
        W(i) = sum(lw(:,i));    %total WAG force on entire wing
        LW(i) = sum(lqw(:,i));  %total QS + WAG force on entire
```

```
            wing
    end
save([path id])
end
if verb timegone = toc; disp([num2str(round(timegone)) ' Done']);
    disp(' ');end


if show
    ymax = max(max(max(lds)),max(max(lqw)));
    figure
    subplot 221
    surf(tt,r,lds)
    title('original lift')
    shading interp
    axis([0 2*pi 0 1 -ymax ymax])

    subplot 222
    surf(tt,r,lw)
    title('wagner correction')
    shading interp
    axis([0 2*pi 0 1 -ymax ymax])

    subplot 223
    surf(tt,r,lqw)
    title('wagner-corrected lift')
    shading interp
    axis([0 2*pi 0 1 -ymax ymax])

    subplot 224
    plot(tt,L,'k')
    hold on
    plot(tt,LW,'b');
    plot([min(tt) max(tt)],[mean(L) mean(L)],'m')
    plot([min(tt) max(tt)],[mean(LW) mean(LW)],'g')
    title('original qs lift (black) vs wagner corrected')
end

L_wag = sum(lw); %for entire span
D_wag = zeros(size(L_wag)); %Wagner does not create drag
save([path 'wag_final'])
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END OF MAIN
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

B.3.4 master_kus

This masterfile calculates the effect of the secondary wakes, using the Küssner function. It divides into six main subroutines:

- **kus_1_init**, which initializes data, mainly by reading it from **kine** and **geom**
- **kus_2_wakevect**, which forms vectors of wake location and the vorticity at each
- **kus_3_influence_coefficients**, which calculates the influence coefficients and induced velocity at each timestep
- **kus_4_CL**, which uses the induced velocity to form a perturbation to the lift and force coefficients
- **kus_5_Kuessner_effect**, which applies the Küssner function to the perturbation of the lift and drag coefficients
- **kus_6_force_reconstruct**, which forms the actual forces from the coefficients.

kus_1_init

The function loads the kinematic and geometric data. Note also that it loads a number of runtime parameters from **kine**. These will be explained when they are used in the code.
Line 100 finds the bound vorticity *GB* of the wing, by calling **qs**. This is the vorticity per m span at every radial step.
Line 102 finds the vorticity shed into the wake at every timestep *DGW*, as the numerical differential of *GB*. Note that, like with the Wagner masterfunction, we use **der**, and assume the data wraps.
Lines 105–114 deals with the case when data does not wrap, via the variable *firststep*. This, like Wagner above, sets either an impulsive start, or smooths the first value of *DGW*.
Lines 119–128 calculates the downwash velocity *ui* and offset between cycles *h*. This uses the mean lift calculated from the quasi-steady calculation, and the equation of section 7.5.1.

kus_2_wakevect

We perform radial stepping, in a loop that start at line 145 and ends at line 226.
Lines 146–152 forcibly set results for the root to 0, to avoid divide by 0 errors.
Lines 154–169 form the distance travelled per timestep *Ddist* and the reversal points *rev*. It uses the runtime parameter *tailflag* to decided wether to base these calculations on velocities at the tail or the hinge.
Lines 172–179 checks *firststep* to see if the data is wrappable, and if it is, shifts the data so the first reversal is at the first index.
Lines 181–187 ensure that the index of reversal points *nrev* has a first value of 1, and a last value of *nt*. This is done for the sake of splitting the wake into single-stroke elements.

Lines 190–201 finds the horizontal direction of the first stroke, which is used for calculating wake location
Lines 204–223 steps through the strokes, forming *xwak*, which is the horizontal location of the wake shed at every timestep. Note that this location is in the spherical coordinate system.
Lines 208–210 deal with a special case, the last stroke. For this we want the stroke to end at the last timestep, because we have forcibly set the last timestep as a reversal point, and the first timestep too. This would cause and extra stroke of length 1 index between the last and first value of the time series (if we are wrapping), so we add 1 to the end index of the last stroke, and ignore the last reversal point at *nt*.
Lines 211–223 creates *dist*, the distance travelled within the current stroke, *direction*, the direction of the current stroke, *xwak* as described above, *GW*, the vorticity in the wake (this was already expressed in *DGW*, but this line is used to modify the wake vorticity when running test cases), and *wakenum*, the count of the current stroke, starting from 0. *wakenum* leads directly to finding *zwak*, which is the wake number times the vertical offset between strokes. Note: the offset is between strokes, half that of the offset between cycles. Note also this is positive, and counts up with *wakenum* from 0.
Lines 228–256 create a full wake, if requested via *wakemethod='f'*. For this, rather than the wake growing from the first timestep, we create a fully-formed secondary wake, of length *nwak* times the original data. It assumes the data is a single, closed cycle, because otherwise we can't be wrapping it. Most of the variables are simply wrapped, with their original value appended to the end, with the exception of *wakenum* on line 246, which has the number of wakes in the first cycle added to it (so it is a continuous upward count, rather than suddenly starting from 0 again), and *zwak* on line 243, which is offset by the distance caused by the induced velocity during the first cycle. Again, this is so it keeps counting up, and doesn't suddenly reset to 0.

kus_3_influence_coefficients

We step radially, starting at line 273 up to line 348.
Line 279 creates *toff* which is the offset to the index just before the last cycle in the secondary wake. if *nwak* is 1 (i.e. we aren't using a fully formed wake), *toff=0*.
Lines 280–291 creates *nrev*, exactly as above, except this time we use *toff* to find *nrev* for the last cycle, not the first.
lines 293–345 step through the strokes in the last cycle, then steps through timestep within the current stroke in lines 300–344. At each timestep, it calculates the offset distance to each previous point in the wake (including earlier strokes). *xoff* is the horizontal offset distance (in the spherical coordinate system) from the leading edge of the wing to the point in the wing, noting that *xwak(ri,ti)* is the current horizontal position of the trailing edge. Similarly, *zoff* is the vertical offset distance.
Lines 307–316 uses the offsets above to form the influence coefficients *hinf, vinf* of the 2-D Biot-Savart equation, and multiplies the influence coefficient for every point in the wake by the vorticity of that point in the wake, to find the vector of velocity induced by every point in the wake, *hw, vw*. These are then summed to give *hwl,vwl*, the total induced velocity

due to the secondary wakes. Note that this has to be calculated for every timestep, so the calculation time goes with the number of timesteps squared.

kus_4_CL

Again, we use radial stepping, in a loop from line 374–423 This calculates the vertical and horizontal force coefficients, by calling qs. The coefficients are calculated twice: *cl,cd* before adding the induced velocity of the wake, and *cl2,cd2* after.
As in master_wag, we use *usepolhamus* to decide wether to adjust the coefficients for Polhamus effect, in lines 391–420.
Finally, we form *CL,CD*, the perturbation of *cl,cd* due to the secondary wake induced velocity, in lines 425–426. These are the quantities we will be applying the Küssner effect to.

kus_5_Kuessner_effect

In this subroutine, we apply the Küssner effect to the perturbation of *cd,cl* calculated above. We use radial stepping, in a loop from line 445–505.
Lines 451–469 form *DCL,DCD*, the step changes in *CL,CD*. As before, they use *firststep* to decide how to treat the first point in the series.
Lines 471–483 finds *nrev* for the last cycle, as done in subroutine 3.
Lines 484–504 step though strokes, while lines 491–503 step through timesteps within the current stroke.
At each timestep, we calculate *distw*, which is the horizontal distance from the trailing edge to a point in the wake. Unlike the offset calculated in subroutine 3, *distw* is always positive, increasing from 0 at the current timestep. WE then say this is the penetration distance of the *CD,CL* change, so call kussner in lines 495 and 498 to return the Küssner perturbation contribution, for every previous point in the wake.
We call Kussner again in lines 500,501 to get the total perturbation contribution. This could also have been done by summing the local contributions, but during development this method was used to cross-check the results of Kussner.

kus_6_force_reconstruct

Finally, just like in master_wag, we convert the coefficients back into forces. Note that the resulting forces are the force per element, not per m span. These forces are then summed across all radial positions to form the total force on the wing due to the secondary wake.

```
1 function main(path,verb,show,fast)
2 %master_kus(verb,show,fast)
3 %calculates the induced velocity due to secondary wakes (using
     loewy approximations)
4 %then applies this as a perturbation velocity using Kussner's
     theory
```

```
%verb is the verbosity: 0=no feedback, 1=some feedback, 2-4= detailed fedback
%show is the amount of data to plot: 0=none 1=some 2+=all
%fast is a flag for how much of the code to skip, loading data from a previous run


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parsing input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Edited 3.5.03 by CP Purely numeric: removed spatial mapping
%Edited 1.5.02 by CP correcting induced velocity to be based on 3d biot-savart
%Edited 21.4.03 by CP correcting CL calcualtion
%Now use velocity-corrected qs calcualtion
last_edited='3.May.03';
last_run=date;

%input switch
switch nargin
case 0
    disp([mfilename ' error, must specify a path for rundata'])
    return
case 1
    verb = 0;
    show = 0;
    fast = 0;
case 2
    show = 0;
    fast = 0;
case 3
    fast = 0;
case 4
    %do nothing
otherwise
    disp([mfilename ' error: too many input arguments'])
    return
end

save temp verb show fast %saves passed information you don't want overwritten

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

```
% non-run inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%special inputs that cause the program to display extended information , rather than run normally
switch verb
case -3
    disp([mfilename ' has 6 runlevels , see help ' mfilename ' for more'])
    return
case -2
    disp([mfilename ' runtime data path ' path])
    return
case -1
    disp([mfilename ' last edited ' last_edited ' and last run ' date])
    return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INITIAL VALUES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if fast & verb disp('Skipping some calculations');end

if verb message(toc,'s Initialising variables');end
id = 'kus_1_init';
if fast
    load([path id])
    load temp verb show fast;
    if verb message(toc,'Loading from previous run');end
else
    rshow = 14; tshow =2300; %which radial and time positions to show
    nwak = kine('nwak'); %number of times to repeat the full cycle if forming a full wake
    tailflag = kine('tailflag'); %wether to calculate reversal based on tail or hinge position
    datalength = kine('datalength'); %wether the data is a full , half or other part of a cycle
    firststep = kine('firststep'); %method to use on the first dCL/dt step
    tailflag = kine('tailflag'); %wether to use the corrected location at te or just the hinge
    wakemethod = kine('wakemethod'); %(f)ull or (g)row.
    usepolhamus = kine('usepolhamus'); %wether to adjust cl for polhamus when calculating wake effect
```

```
nt = kine('nt');
t = kine('t'); tt = kine('tt');  dt = kine('dt');
r = geom('r_default','t'); nr = length(r);
dr = geom('dr');
b = geom('b','t',r,0,0); hinge = geom('hinge');
phi = kine('phi'); phi = phi(1:nt);
Dphi = max(phi) - min(phi);
if length(hinge) == 1;
    hinge = ones(1,nr)*hinge;
end


%find bound vorticity GB, and DGW is step change in wake
    vorticity
unt = kine('unt'); dp = kine('dp'); pitch = kine('pitch');
uht = kine('uht'); uvt = kine('uvt'); R = geom('R');

for ri=1:nr
    un = unt * r(ri);
    %bound vorticity per meter span, at each radial step
    GB(ri,:) = qs('gamma',uht*r(ri),uvt*r(ri),pitch,dp,r(ri),b
        (ri),hinge(ri));
    %shed vorticity into the wake
    DGW(ri,:) = -Der(GB(ri,:),1:nt);

    %need to change the first value based on what data we are
        using
    switch firststep
    case 'w'
        %wrap data - already calculated above
    case 'i'
        DGW(ri,1) = -GB(ri,1); %impulsive start
    case 's'
        DGW(ri,1) = DGW(ri,2); %smooth by setting equal to
            second step
    otherwise
        message(toc,['bad value for firststep: ' firststep]);
        return
    end
end

%calulate average induced vertical velocity
load([path,'qs_final'],'LW')
LW_mean = mean(LW);
R = geom('R'); rho = kine('rho');

```

```
    Phi_prime = Dphi/(2*pi); %fraction of a full revolution convered
    As = pi * R^2 * Phi_prime;
    ui = sqrt(LW_mean/(rho * As));
    T = kine('T');
    h = ui * T; %vertical offset between wakes (between full strokes , not halfstrokes);
    clear LW;

save([path id])
end
if verb timegone = toc; disp([num2str(round(timegone)) 's Done']); disp(' ');end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SECONDARY WAKE VECTORS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb timegone = toc; disp([num2str(round(timegone)) 's Calculating wake gamma']);end
id = 'kus_2_wakevect';
if fast>1
    load([path id])
    load temp verb show fast;
    if verb disp('Loading from previous run');end
else
    shift = zeros(1,nr);
    for ri=1:nr if verb>1 message(toc,['Radial position ' num2str(ri)]);end
        if r(ri) ==0
            Ddist(ri ,:) = zeros(1,nt);
            rev(ri ,:) = zeros(1,nt);
            shift(ri) = 0;
            xwak(ri ,:) = zeros(1,nt);
            zwak(ri ,:) = zeros(1,nt);
            GW(ri ,:) = zeros(1,nt);
        else
            if tailflag %wether to base reversals and distance travelled on hinge or trailing edge
                %velocity at trailing edge
                uhte(ri ,:) = uht*r(ri) + dp * b(ri) * (+1 -hinge(ri)) .* sin(pitch);
                %Distance covered per timestep
                Ddist(ri ,:) = abs(uhte(ri ,:) .* dt);
                %REversal points
                rev(ri ,:) = find_crossings('vector',uhte(ri ,:));
            else
```

```
            Ddist(ri ,:) = abs(uht * r(ri)) .* dt;
            rev(ri ,:) = find_crossings('vector',uht);
        end

        nrev = find(rev(ri ,:));
        if ~sum(rev(ri ,:))
            nrev = 1;
        end


        if firststep == 'w';
            if verb & ri==2 message(toc,'wrapping data');end
            %shifts the data so the first reversal point is at
                index 1
            shift(ri) = nrev(1);
            Ddist(ri ,:) = shifter(Ddist(ri ,:),shift(ri));
            rev(ri ,:)   = shifter(rev(ri ,:),shift(ri));
            nrev = find(rev(ri ,:));
        end

        if nrev(1) ~=1
            nrev = [1 nrev];
        end

        if nrev(length(nrev)) ~=nt;
            nrev = [nrev nt];
        end

        %find the direction of the first stroke
        if tailflag
            uchar = uhte(ri ,:);
        else
            uchar = uht;
        end
        first_direction(ri) = sign(uchar(1));
        if first_direction(ri) == 0
            first_direction(ri) = sign(uchar(2));
        end
        if first_direction == 0
            disp('error, inital velocity is 0');
        end

        %step through strokes
        xwak(ri ,1:nt) = zeros(1,nt);
        for stroke=1:length(nrev)-1
            t1 = nrev(stroke); %start index of this stroke
```

```
                t2 = nrev(stroke+1)-1;%end index of this stroke
                if stroke == length(nrev)-1
                    t2 = t2 +1;
                end
                wakenum(ri,t1:t2) = stroke-1;
                direction(ri,t1:t2) = -1*(-1).^wakenum(ri,t1:t2)*first_direction(ri);
                for ti=t1:t2;
                    %distance travelled in this stroke
                    dist(ri,ti) = sum(Ddist(ri,t1:ti));
                end
                xwak(ri,t1:t2) = dist(ri,t1:t2).*direction(ri,t1:t2);

                if t1~=1
                    xwak(ri,t1:t2) = xwak(ri,t1:t2)+xwak(ri,t1-1);
                end
                zwak(ri,t1:t2) = ui * T/2 * wakenum(ri,t1:t2);
                GW(ri,t1:t2) = DGW(ri,t1:t2);
            end
        end
    end

    if wakemethod == 'f'
        %Turn the wake variables into nwak times their original length
        if verb message(toc,'using full wakemethod');end
        if firststep ~='w'
            disp([mfilename ' warning, when using full wake model, first step should be set to (w)rap'])
        end
        xw = xwak; %temporary variables
        zw = zwak;
        gw = GW;
        dw = dist;
        wn = wakenum;
        rw = rev;

        for ni=2:nwak
            xw = [xw xwak];
            zw = [zw zwak + ni * T * ui];
            dw = [dw dist];
            gw = [gw GW];
            wn = [wn ni+wakenum];
            rw = [rw rev];
        end
```

```
        xwak = xw;
        zwak = zw;
        dist = dw;
        GW = gw;
        rev = rw;
        clear xw;clear zw; clear gw; clear dw;
        clear rw; clear wn; clear rw;
    end

save([path id])
clear nrev;
if verb message(toc,' Done');disp(' ');end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INFLUENCE COEFFICIENTS and velocities
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb timegone = toc; disp([num2str(round(timegone)) ' forming
    influence coefficients']);end
id = 'kus_3_influence_coefficients';
if fast>2
    load([path id])
    load temp verb show fast;
    if verb disp('Loading from previous run');end
else
    for ri=1:nr
        if verb>1 message(toc,[' Now at radial index ' num2str(ri)
            ]);end
        if b(ri)==0 | r(ri) == 0
            vwl(ri,:) = zeros(1,nt*nwak);
            hwl(ri,:) = zeros(1,nt*nwak);
        else
                toff = nt*(nwak-1); %this is offset so we
                    calculate for last cycle
                nrev = find(rev(ri,toff+1:toff+nt))+toff;
                if ~sum(rev(ri,toff+1:nt*nwak))
                    nrev = toff+1;
                end

                if nrev(1) ~=toff+1
                    nrev = [toff+1 nrev];
                end

                if nrev(length(nrev)) ~=nt*nwak;
                    nrev = [nrev nt*nwak];
                end
```

```
for stroke = 1:length(nrev)-1 if verb >1 message(toc ,[
    'stroke ' num2str(stroke)]);end
    t1 = nrev(stroke); %start index of this stroke
    t2 = nrev(stroke+1)-1; %end index of this stroke
    if stroke == length(nrev)-1
        t2 = t2 + 1;    %for the last stroke, we want
            the full length
    end
    %steps through time this stroke
    for ti=t1:t2;
        if verb >2 & ti/100 == floor(ti/100); timegone
            = toc; disp([num2str(round(timegone)) '
            time step ' num2str(ti)]);end

        %offset distance
        xoff = xwak(ri,1:t1-1) -xwak(ri,ti)-cos(pitch(
            ti-toff))*2*b(ri); %only the previous wakes
        zoff = -zwak(ri,1:t1-1) +zwak(ri,ti)+sin(pitch
            (ti-toff))*2*b(ri);

        %influence coeficcients (vertical and
            horizontal)
        vinf = (1/2/pi)*xoff ./ (abs(xoff.^2) + abs(
            zoff.^2));
        hinf = (1/2/pi)*zoff ./ (abs(xoff.^2) + abs(
            zoff.^2));
        %induced velocity contribution for every point
            in wake
        vw = GW(ri,1:t1-1).*vinf;
        hw = GW(ri,1:t1-1).*hinf;

        %total contribution for entire wake
        vw1(ri,ti) = sum(vw);
        hw1(ri,ti) = sum(hw);

        if show & ri==rshow & ti==tshow
            figure
            subplot 221
            plot(vinf)
            title('vinf')

            subplot 222
            plot(hinf)
            title('hinf')

```

```
                        subplot 223
                        plot(xoff)
                        title('xoff')

                        subplot 224
                        plot(zoff)
                        title('zoff')

                        figure
                        plot3(xoff,zoff,vinf)

                        figure
                        plot(uhte(rshow,:))
                        t1
                        t2
                        save kus_data
                    end
                end
            end
        end
    end
save([path id])
if verb message(toc,' Done');disp(' ');end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Resolve as normal and parallele
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
save test_data
for ri=1:length(r)
    [pwl(ri,:),nwl(ri,:)] = cz_resolve(vwl(ri,toff+1:toff+nt),hwl(
        ri,toff+1:toff+nt),pitch);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Lift Coefficients
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb message(toc,'Lift coefficients');end
id = 'kus_4_CL';
if fast>3
    load([path id])
    load temp verb show fast;
    if verb message(toc,'Loading from previous run');end
else
    %form velocity matricies
```

```
%   B = geom('B');
    for ri=1:length(r) %radial stepping
        if verb>1 message(toc,[' radial position ' num2str(ri)]);
           end
        if b(ri)==0 | r(ri) == 0
            cl(ri,1:nt) = zeros(1,nt);
            cl2(ri,1:nt) = zeros(1,nt);
            cd(ri,1:nt) = zeros(1,nt);
            cd2(ri,1:nt) = zeros(1,nt);
            uh(ri,1:nt) = uht(1:nt)*r(ri);
            uv(ri,1:nt) = uvt(1:nt)*r(ri);
        else
            uh(ri,:) = uht * r(ri); uv(ri,:) = uvt * r(ri);
            %Lift coefficient before wake velocity correction
            cl(ri,:) = qs('CL',uh(ri,:),uv(ri,:),pitch,dp,r(ri),b(
               ri),hinge(ri));
            %Lift coefficient after wake velocity correction
            cl2(ri,:)= qs('CL',uh(ri,:) + hwl(ri,toff+1:toff+nt),
               uv(ri,:) +vwl(ri,toff+1:toff+nt),pitch,dp,r(ri),b(
               ri),hinge(ri));
            cd(ri,:) = qs('CD',uh(ri,:),uv(ri,:),pitch,dp,r(ri),b(
               ri),hinge(ri));
            cd2(ri,:)= qs('CD',uh(ri,:) + hwl(ri,toff+1:toff+nt),
               uv(ri,:) +vwl(ri,toff+1:toff+nt),pitch,dp,r(ri),b(
               ri),hinge(ri));
            switch usepolhamus
            case 'n'
                %do nothing
            case 'y'
                if verb & ri == 2 message(toc,'using polhamus
                   correction to cl');end
                %mean square velocity before wake correction
                ut2 = qs('ut2',uh(ri,:),uv(ri,:),pitch,dp,r(ri),b(
                   ri),hinge(ri));
                %mean square velocity after wake correction
                ut2kus = qs('ut2',uh(ri,:) + hwl(ri,toff+1:toff+nt
                   ),uv(ri,:) +vwl(ri,toff+1:toff+nt),pitch,dp,r(
                   ri),b(ri),hinge(ri));
                %leading edge slope
                dledr = geom('dledr','t',r); corr = sqrt(1+dledr
                   .^2);

                %Polhamus lift effects
                L_pol(ri,:) = pol('L_pol',uh(ri,:),uv(ri,:),pitch,
                   dp,r(ri),b(ri),hinge(ri),corr(ri),0,0);
```

```
                    L_pol2(ri,:) = pol('L_pol',uh(ri,:)+hwl(ri,toff+1:
                        toff+nt),uv(ri,:) + vwl(ri,toff+1:toff+nt),
                        pitch,dp,r(ri),b(ri),hinge(ri),corr(ri),0,0);
                    D_pol(ri,:) = pol('D_pol',uh(ri,:),uv(ri,:),pitch,
                        dp,r(ri),b(ri),hinge(ri),corr(ri),0,0);
                    D_pol2(ri,:) = pol('D_pol',uh(ri,:)+hwl(ri,toff+1:
                        toff+nt),uv(ri,:) + vwl(ri,toff+1:toff+nt),
                        pitch,dp,r(ri),b(ri),hinge(ri),corr(ri),0,0);

                    clp(ri,:) = L_pol(ri,:)./(rho .* b(ri) .* ut2);
                    clp2(ri,:) = L_pol2(ri,:)./(rho .* b(ri) .* ut2kus
                        );
                    cdp(ri,:) = D_pol(ri,:)./(rho .* b(ri) .* ut2);
                    cdp2(ri,:) = D_pol2(ri,:)./(rho .* b(ri) .* ut2kus
                        );

                    cl(ri,:) = cl(ri,:) + clp(ri,:);
                    cl2(ri,:) = cl2(ri,:) + clp2(ri,:);
                    cd(ri,:) = cd(ri,:) + cdp(ri,:);
                    cd2(ri,:) = cd2(ri,:) + cdp2(ri,:);

                    clear L_pol; clear L_pol2; clear D_pol; clear
                        D_pol2;
                    clear ut2; clear ut2kus;
                end
            end
        end

CL = cl2-cl; %wake perturbation CL
CD = cd2-cd; %wake perturbation CD

%test data C = ones(size(C));

save([path id]);
if verb message(toc,'Done');disp(' ');end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Kuessner effect
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb message(toc,'Kuessner effect');end
id = 'kus_5_Kuessner_effect';
if fast>4
    load([path id])
    load temp verb show fast;
    if verb disp('Loading from previous run');end
```

```
else
    kus_loca = zeros(nr,nt);
    for ri=1:nr %radial stepping
     if verb>1 message(toc,['s radial ' num2str(ri)]);end
     if b(ri)==0 | r(ri) == 0
     kus_L(ri,1:nt) = zeros(1,nt);
     kus_D(ri,1:nt) = zeros(1,nt);
     else
      DCL(ri,:) = der(CL(ri,:),1:nt); %step changes in C
      DCD(ri,:) = der(CD(ri,:),1:nt); %step changes in C
      %change first value depending on method
      switch firststep
      case 'w'
       %no nothing - already calculated above
      case 'i'
       DCL(ri,1) = cl2(ri,1); %impulsive start
       DCD(ri,1) = cd2(ri,1); %impulsive start
      case 'o'
       DCL(ri,1) = -DCL(ri,1); %first step is opposite of last one
            in series
       DCD(ri,1) = -DCD(ri,1); %first step is opposite of last one
            in series
      case 's'
       DCL(ri,1) = DCL(ri,2); %smooth by setting equal to second
          step
       DCD(ri,1) = DCD(ri,2); %smooth by setting equal to second
          step
      otherwise
       message(toc,['bad value for firststep: ' firststep]);
       return
      end

      toff = (nwak-1)*nt;
      nrev = find(rev(ri,toff+1:toff+nt))+toff; %step through the
         last values
      if ~sum(rev(ri,toff+1:toff+nt))
       nrev = toff+1;
      end

      if nrev(1) ~=toff+1
       nrev = [toff+1 nrev];
      end

      if nrev(length(nrev)) ~=nt*nwak;
       nrev = [nrev nt*nwak];
      end
```

```
          for stroke = 1:length(nrev)-1
           t1 = nrev(stroke); %start index of this stroke
           t2 = nrev(stroke+1)-1; %end index of this stroke
           if stroke == length(nrev)-1
            t2 = t2 + 1;    %for the last stroke, we want the full length
           end

           for ti=t1:t2; if verb >2 & ti/100 == floor(ti/100); message(toc,['s time step ' num2str(ti)]);end
            distw = dist(ri,ti)-dist(ri,t1:ti); %distance to points in wake, not distance travelled
            if (show>0 & ri==rshow & ti == tshow)
             %kussner correcto for every point in the wake
             kus_loca_L(ri,t1-toff:ti-toff) = kussner(DCL(ri,t1-toff:ti-toff),distw./b(ri),'loca',0,1);
            else
             %kussner correcto for every point in the wake
             kus_loca_L(ri,t1-toff:ti-toff) = kussner(DCL(ri,t1-toff:ti-toff),distw./b(ri),'loca',0,0);
            end
            kus_L(ri,ti-toff) = kussner(DCL(ri,t1-toff:ti-toff),distw./b(ri),'tota',0,0);
            kus_D(ri,ti-toff) = kussner(DCD(ri,t1-toff:ti-toff),distw./b(ri),'tota',0,0);
           end
          end
        end
    end
save([path id])
if verb message(toc,'Done');disp(' ');end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reconstruction
%go from coefficients to actual forces
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if verb message(toc,'Reconstructing forces from coefficients');end
id = 'kus_6_force_reconstruct';
if fast>5
    load([path id])
    load temp verb show fast;
    if verb message(toc,'Loading from previous run');end
else
    for ri=1:nr
        %Mean square velocity before and after wake effect
```

```
        ut2(ri,:) = qs('ut2',uh(ri,:),uv(ri,:),pitch,dp,r(ri),b(ri
            ),hinge(ri));
        ut2kus(ri,:) = qs('ut2',uh(ri,:) + hwl(ri,toff+1:toff+nt),
            uv(ri,:) +vwl(ri,toff+1:toff+nt),pitch,dp,r(ri),b(ri),
            hinge(ri));

        LQ(ri,:) = cl(ri,:) .* ut2(ri,:) * rho * b(ri) * R * dr(ri
            ); %original lift, numerical sum
        LK(ri,:) = kus_L(ri,:).* ut2kus(ri,:)*rho*b(ri) * R * dr(
            ri); %perturbation due to kuessner effect
        LT(ri,:) = LK(ri,:) + LQ(ri,:); %total lift
        DQ(ri,:) = cd(ri,:) .* ut2(ri,:) * rho * b(ri) * R * dr(ri
            ); %original lift, numerical sum
        DK(ri,:) = kus_D(ri,:).* ut2kus(ri,:)*rho*b(ri) * R * dr(
            ri); %perturbation due to kuessner effect
        DT(ri,:) = DK(ri,:) + DQ(ri,:); %total lift
    end
save([path id])
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Shift back to original time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if shift
    if verb message(toc,'un-shifting data');end
    for ri=1:nr
        Ddist(ri,:) = shifter(Ddist(ri,:),-shift(ri));
        dist(ri,:)  = shifter(dist(ri,:),-shift(ri));
        DCL(ri,:) = shifter(DCL(ri,:),-shift(ri));
        wag(ri,:)    = shifter(wag(ri,:),-shift(ri));
        LK(ri,:) = shifter(LK(ri,:),-shift(ri));
        DK(ri,:) = shifter(DK(ri,:),-shift(ri));
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Total lift correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L_kus = sum(LK); %sums corrections across span
D_kus = sum(DK);

save([path 'kus_final'])
return

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Additional functions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [parl,norm] = cz_resolve(vert,horz,angle);
%resolves velocities parallel and normal to wing

norm = vert .* cos(angle) + horz .* sin(angle);
parl = vert .*-sin(angle) + horz .* cos(angle);

return
```

B.4 Run Functions

There are two of these, and both are rather simple. They simply call the master functions in order, and tell them where to save the results. **master** performs an analytical evaluation (using wing shape parameters), while **master_num** uses numerical summation.

Master

```
%master
%note that no variables are created
%they have to be loaded from the folder \rundata
%Last edited 12.Mar.03 by CP

clear all; close all
tic

path = 'c:\data\math\matlab\mekado\current\rundata\';
verb = [1 1 2 2]; %verbosity level
show = [0 0 0 8]; %show level
skip = [0 0 0 0]; %amount of subroutines to skip

if verb
    disp(geom('id'))    %displays the geometry being used
    disp(kine('id'))    %displays the kinematics being used
end

disp('**** Quasi steady ***')
master_qsam(path,verb(1),show(1),skip(1))
disp(['******* Done ********'])
disp(' ')

disp('***** Polhamus ******')
master_polhamus(path,verb(2),show(2),skip(2))
disp(['******* Done ********'])
disp(' ')

disp('****** Wagner *******')
master_wag(path,verb(3),show(3),skip(3))
disp(['******* Done ********'])
disp(' ')


disp('**** Secondary wake *****')
master_kus(path,verb(4),show(4),skip(4))
disp(['******* Done ********'])
disp(' ')
```

Master_num

```
%master_num
%this is the numeric runfile.
%it runs all the other files
%note that no variables are created
%they have to be loaded from the folder \rundata
%Last edited 21.May.03 by CP


clear all
close all
tic

path = 'rundata/';
verb = [1 1 1 1]; %verbosity level
show = [0 0 0 0]; %show level
skip = [0 0 0 0]; %amount of subroutines to skip

if verb
    disp(geom('id'))    %displays the geometry being used
    disp(kine('id'))    %displays the kinematics being used
end

disp('**** Quasi steady ***')
numerical_qsam(path,verb(1),show(1),skip(1))
disp(['******* Done ********'])
disp(' ')

disp('***** Polhamus ******')
numerical_polhamus(path,verb(2),show(2),skip(2))
disp(['******* Done ********'])
disp(' ')

disp('****** Wagner *******')
master_wag(path,verb(3),show(3),skip(3))
disp(['******* Done ********'])
disp(' ')


disp('**** Secondary wake *****')
master_kus(path,verb(4),show(4),skip(4))
disp(['******* Done ********'])
disp(' ')
```

B.5 Miscellaneous functions

These are calculation-level functions, with a very limited scope, typically a single task.

B.5.1 der

This calculates the numerical differential of a variable x wrt variable t. It assumes that x forms a closed path, so it can wrap the data around the last to first value.

```
function dx = Der(x,t);
%[dx,dy] = Der(x,t);
%returns the derivative of x wrt t. The deriavtive is of the same
    length as the original.
%assumes that  x forms a closed paths, so the last value of x and
    dx is the same as the first.
%requires time values 0:dt:t-dt, not dt:dt:T
dx = diff(x);
dt = diff(t);

dx = [dx(length(dx)) dx]; %adds an extra value on the end so
    vectors are same length
dt = [dt(length(dt)) dt];

dx = dx./dt;
```

B.5.2 find_crossings

This finds the points where a variable x crosses zero (changes sign), low-pass filtering the data to avoid multiple crossings in close succession, e.g. for noisy data.

```
function out = main(gimme,data ,deadspace ,verb ,show);
%nrev = find_crossings(gimme,data , deadspace , verb , show);
%find the points where the sign of vector DATA changes
%GIMME is either 'index ' or 'vector '
%    'index ' returns the indexes where sign changes
%    'vector ' returns a full length vector of zeros , with ones at the crossing points
%for noisy data , we sometimes get multiple crossings in close succesion
%the input DEADSPACE governs the minimum number of points between crossings
%if crossings are closer than this , the later ones are ignored.
%deadspace is by default 5% of the length of DATA
%VERB (verbose) is a flag wether additional information should be shown
%SHOW is a flag to plot the results of the function

%created 16.5.03 by C.Pedersen
last_edited='16-May-2003';
last_run = date;

%assign default values to missing inputs
switch nargin
case {0,1}
    warning('need at least two inputs')
    return
case 2
    deadspace = floor(length(data)/20);
    verb =0;show = 0;
case 3
    verb = 0;show = 0;
case 4
    show = 0;
case 5
    %do nothing
otherwise
    warning('too many inputs - exiting')
    return
end

nt = length(data);
S = sign(data);
```

```
nrev = find(S(2:nt)-S(1:nt-1));

%form vector
rev = zeros(1,nt);
rev(nrev) = 1;

for i=1:length(nrev)
    %index of points covered by the deadspace
    dead_index = nrev(i)+1:nrev(i)+1+deadspace;
    %sets all reversal points in deadspace to 0
    rev(dead_index) = zeros(size(dead_index));
end
%nrev is values where rev is still not 0;
[error,nrev] = find(rev);
%restores rev to original length (can have got longer when
    searching deadspace)
rev = rev(1:nt);

%Output switch
switch gimme
case 'index'
    out = nrev; if verb disp('index of reversal points');end
case 'vector'
    out = rev; if verb disp('full length vector');end
otherwise
    disp([mfilename ' error: unknown input for gimme: ' gimme]);
    return
end

if verb
    disp(['found ' num2str(length(nrev)) ' reversal points'])
    disp(['in ' num2str(nt) ' datapoints'])
end

%Display functions
if show
    plot(data)
    hold on
    if exist('nrev')
        plot(nrev,zeros(size(nrev)),'go')
        plot([nrev(1) nrev(1)+1+deadspace],[data(nrev(1)) data(
            nrev(1))],'k-')
    end
    title('reversal points')
end
```

B.5.3 message

Displays the time elapsed, along with a given string, to the run window.

```
function message(time,message_text);

disp([num2str(round(time)) ' ' message_text]);
```

B.5.4 rotator

Rotates an arbitrary vector in 3-D, using the euler angles. In our code, this is used to find the x, y, z location of the tip.

```
function X = rotator(x,phi ,psi ,ang);
%xr = rotator(x,phi,psi,ang)
%rotates by euler angles ang (pitch), phi (sweep) and psi (plunge)
% x is a matrix of 3xn
ANG = [cos(ang) 0 -sin(ang) ; 0 1 0; sin(ang) 0 cos(ang)]';
PSI = [ 1  0 0 ; 0 cos(psi) sin(psi) ; 0 -sin(psi) cos(psi)]';
PHI = [ cos(phi) sin(phi) 0 ; -sin(phi) cos(phi) 0; 0 0 1]';

X = (PHI * PSI * ANG * x')';

%show the final transformation matrix by the following commands
%clear all
%syms('x', 'y', 'z', 'real')
%syms('phi', 'psi', 'ang', 'real')
%X = rotator([x y z],phi,psi,ang);
%X'
```

References

[1] R. Żbikowski. Global optimisation solvers based on trajectory methods: Preliminary report. Technical Report DAPS/RZ/55/2000, Cranfield University, RMCS Shrivenham, July 2000.

[2] S. P. Sane and M. H. Dickinson. The control of flight force by a flapping wing: Lift and drag production. *Journal of Experimental Biology*, 204:2607–2625, 2001.

[3] J. G. Leishman. *Principles of Helicopter Aerodynamics*. Cambridge University Press, Cambridge, England, 2000.

[4] R. Żbikowski. On aerodynamic modelling of an insect-like flapping wing in hover for micro air vehicles. *Philosophical Transactions of the Royal Society of London (Series A: Mathematical, Physical and Engineering Sciences)*, 360(1791):273–290, 2002.

[5] E. L. Houghton and P. W. Carpenter. *Aerodynamics for Engineering Students*. Edward Arnold, London, Fourth edition, 1993.

[6] J. W. S. Pringle. *Insect Flight*, volume 52 of *Oxford Biology Readers*. Oxford University Press, Glasgow, 1975.

[7] J. Katz and A. Plotkin. *Low Speed Aerodynamics*. Cambridge University Press, 2001.

[8] T. J. Mueller. *Fixed and Flapping Wing Aerodynamics for Micro Air Vehicle Applications*, volume 195 of *Progress in Astronautics and Aeronautics*. American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia, US, 2001.

[9] C. P. Ellington and J. R. Usherwood. Lift and drag characteristics of rotary and flapping wings. In T. J. Mueller, editor, *Fixed and Flapping Wing Aerodynamics for Micro Air Vehicle Applications*, volume 195 of *Progress in Astronautics and Aeronautics*, pages 231–246. American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia, US, 2001.

[10] K. C. Hall and S. R. Hall. A rational engineering analysis of the efficiency of flapping flight. In T. J. Mueller, editor, *Fixed and Flapping Wing Aerodynamics for Micro Air Vehicle Applications*, volume 195 of *Progress in Astronautics and Aeronautics*, pages 249–272. American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia, US, 2001.

[11] J. N. Newman. *Marine Hydrodynamics*. MIT Press, London, England, 1977.

[12] L. M. Milne-Thomson. *Theoretical hydrodynamics*. Dover Publications, inc., Mineola, New York, USA, 5th edition, 1996.

[13] L. I. Sedov. *Two–Dimensional Problems in Hydrodynamics and Aerodynamics*. Interscience, John WIley & Sons Inc., London, First edition, 1965.

[14] E. C. Polhamus. A concept of the vortex lift of sharp-edge delta wings based on a leading-edge-suction analogy. NASA Technical Note TN D-3767, National Aeronautics and Space Administration, 1966.

[15] R. G. Bradley, C. W. Smith, and I. C. Bhateley. Vortex-lift prediction for complex wing planforms. *Journal of Aircraft*, 10(6):379–381, 1973.

[16] J. W. Purvis. Analytical prediction of vortex lift. *Journal of Aircraft*, 18(4):225–230, 1981.

[17] J. E. Lamar. The use and characteristics of vortical flows near a generating aerodynamic surface: A perspective. *Progress in Aerospace Science*, 34(3–4):167–217, 1998. (Erratum in Volume 34, Number 7–8, page 543).

[18] C. P. Ellington. The aerodynamics of hovering insect flight. I. The quasi-steady analysis. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, 305(1122):1–15, 1984.

[19] C. P. Ellington. The aerodynamics of hovering insect flight. II. Morphological parameters. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, 305(1122):17–40, 1984.

[20] C. P. Ellington. The aerodynamics of hovering insect flight. III. Kinematics. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, 305(1122):41–78, 1984.

[21] C. P. Ellington. The aerodynamics of hovering insect flight. IV. Aerodynamic mechanisms. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, 305(1122):79–113, 1984.

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

[23] C. P. Ellington. The aerodynamics of hovering insect flight. VI. Lift and power requirements. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, 305(1122):145–181, 1984.

[24] C. van den Berg and C. P. Ellington. The vortex wake of a 'hovering' model hawkmoth. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, 352(1351):317–328, 1997.

[25] C. P. Ellington, C. van den Berg, A. P. Willmott, and A. L. R. Thomas. Leading-edge vortices in insect flight. *Nature*, 384(6610):626–630, 1996.

[26] J. M. Birch and M. H. Dickinson. Spanwise flow and the attachment of the leading-edge vortex on insect wings. *Nature*, 412(6848):729–733, 2001.

[27] J. A. Walker. Rotational lift: something different or more of the same? *Journal of Experimental Biology*, 205:3783–3792, 2002.

[28] T. Theodorsen. General theory of aerodynamic instability and the mechanism of flutter. NACA Report 496, National Advisory Committee for Aeronautics, 1935.

[29] B. G. van der Wall and J. G. Leishman. On the influence of time-varying flow velocity on unsteady aeordynamics. *Journal of the American Helicopter Society*, 39(4):25–36, 1994.

[30] H. Wagner. Über die Entstehung des dynamischen Auftriebes von Tragflügeln. *Zeitschrift für Angewandte Mathematik und Mechanik*, 5(1):17–35, 1925.

[31] H. G. Küssner. Zusammenfassender Bericht über den instationären Auftrieb von Flügelen. *Luftfahrtforschung*, 13(12), 1936.

[32] G.H. Keugelan and L.H. Carpenter. Forces on cylinders and plates in an osscilating fluid. *Journal of research of the national bureau of standards*, 60(5):423–440, 1958.

[33] E. Huse. Resonant heave dumping of tension leg platforms. In *Offshore Technology Conference, paper OTC6317*, 1990.

[34] R. Dudley. Atmospheric oxygen, giant paleozoic insects and the evolution of aerial locomotory performance. *Journal of Experimental Biology*, 201(March):1043–1050, 1998.

[35] A. R. Ennos. *The biomechanics of flight in Diptera*. PhD thesis, Exeter University, Biological Sciences Department, Exeter, UK, 1987.

[36] R. J. Wootton. Functional morphology of insect wings. *Annual Review of Entomology*, 37:113–140, 1992.

[37] R. J. Wootton. Support and deformability in insect wings. *Journal of Zoology*, 193(April):447–468, 1981.

[38] A. R. Ennos. The kinematic and aerodynamics of the free flight of some diptera. *Journal of Experimental Biology*, 142:49–85, 1989.

[39] T. Weiss-Fogh. Quick estimates of flight fitness in hovering animals, including novel mechanisms for lift production. *Journal of Experimental Biology*, 59:169–230, 1973.

[40] M. J. Hemsch and J. M. Luckring. Connection between leading-edge sweep, vortex lift, and vortex strength for delta wings. *Journal of Aircraft*, 25(5):473–475, 1990.

[41] R. McNeill-Alexander. Smokescreen lifted on insect flight. *Nature*, 384:609–610, 1996.

[42] T. von Kármán and W. R. Sears. Airfoil theory for non-uniform motion. *Journal of the Aeronautical Sciences*, 5(10):379–390, 1938.

[43] G. G. Stokes. On the effect of the internal friction of fluids on the motion of pendulums. *Transations of Cambridge Philosophical Society*, 9(8), 1851.

[44] B. S. Massey. *Mechanics of Fluids*. Chapman and Hall, London, Seventh edition, 1998.

[45] J. B. Marion and S. T. Thornton. *Classical dynamics of particles and systems*. Harcourt College Publishers, New York, London, 4th edition, 1995.

[46] S. P. Sane and M. H. Dickinson. The aerodynamic effects of wing rotation and a revised quasi-steady model of flapping flight. *Journal of Experimental Biology*, 205(JEB3926):1087–1096, 2002.

[47] R. G. Loewy. A two-dimensional approximation to the unsteady aerodynamics of rotary wings. *Journal of the Aeronautical Sciences*, 24(2):81–92,144, 1957.

[48] M. H. Dickinson, F.O. Lehmann, and S.P. Sane. Wing rotation and the aerodynamic basis for insect flight. *Science*, 284(1):1954–1960, 1999.

[49] J. D. DeLaurier. An aerodynamic model for flapping-wing flight. *The Aeronautical Journal*, 97:125–130, 1993.

[50] I. S. Gradshteyn and I. M. Ryzhik. *Table of integrals series and products*. Academic Press, New york and London, 1994.