DEVELOPMENT AND VALIDATION OF AN UNSTEADY PANEL CODE TO MODEL AIRFOIL AEROMECHANICAL RESPONSE

By

AARON M. MCCLUNG

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the degree of
MASTER OF SCIENCE
July 2004

DEVELOPMENT AND VALIDATION OF AN UNSTEADY PANEL CODE TO MODEL AIRFOIL AEROMECHANICAL RESPONSE

By

AARON M. MCCLUNG

Approved as to style and content by:

Eric A. Falk, Chair

Andrew S. Arena, Member

Gary E. Young, Member

Dean, Graduate College

TABLE OF CONTENTS

LIST OF FIGURES

NOTATION

Φ	Velocity Potential
ρ	Density
Γ	Total circulation strength
γ	Circulation strength per unit length
λ	Source strength per unit length
Λ	Total source strength
$\mathbf{n}$	Normal Vector
$\mathbf{V}$	Velocity Vector
$\mathbf{q}$	Velocity Vector
r_{ij}	Radius from poing j to point i
$m_{c.v.}$	Mass of fluid inside the control volume
m_{out}	Mass flux out of a control volume
m_{in}	Mass flux into a control volume
$\mathbf{p}$	Momentum
$\mathbf{a}$	Acceleration
$\mathbf{F}$	Force
ds	Differential along a surface
$\mathbf{f}$	Body force
τ_{ij}	Fluid shear stress
p	Pressure
μ	Viscocity coefficient
δ_{ij}	Kronecker delta function
ω	Angular velocity of a fluid element
ζ	Vorticity of a fluid element
L	Lift
C_l	Coefficient of Lift
C_d	Coefficient of Drag
C_m	Coefficient of Moment
c	Chord, length of the airfoil section
b	Semi-chord $= \frac{c}{2}$
s	$= Ut/b$, Semi-chord location
f_i	Body Force
τ_{ij}	Fluid Shear Stress
Q_h	$= -L$, Generalized force along the +z-axis
Q_α	$= M_y$, Generalized moment about the elastic axis
m	Mass

h	Vertical translation of the airfoil, positive for deflection along the -z-axis
α	Angle between the airfoil centerline and the mean freestream flow
I_α	Mass moment of inertial per unit span about axis $x = ba$
S_α	$= mbx_\alpha$, Static mass imbalance per unit span about axis $x = ba$
ω_h	$= \sqrt{K_h/m}$, uncoupled natural frequency in bending
ω_α	$= \sqrt{K_\alpha/I_\alpha}$, uncoupled natural frequency in torsion
K_h	Bending spring stiffness
K_α	Torsional spring stiffness
k	$= \omega b/U$, Reduced frequency

CHAPTER 1

INTRODUCTION

Aeroelastic condsiderations affect a wide range of disciplines. With respect to turbomachinery, particularly the area of high-cycle fatigue, aerodynamic forcing of internal components due to rotor-stator interactions can significantly impact engine life-cycle and maintenance requirements.

To better understand the influence of aerodynamic damping, on high-cycle fatigue, the influence of aerodynamic damping on forced structural response must first be be examined. As a first step towards this goal, this thesis develops a computational tool through which the influence of aerodynamic damping can be isolated and systematically studied.

1.1 Goals

The goal of this thesis is to develop and validate a computation tool which will enable the systematic investigation into wake induced stuctural responce. The computational tool is based loosely on a Hess-Smith [5] type unsteady panel code written by Ron Hugo [7, 9] which has been modified to include a freestream gust model and an airfoil structural model. By incorporating the capability to model arbitrary freestream gusts into the unsteady panel code, and coupling the panel code with a structural model, the time-domain response of a body due to an arbitrary freestream disturbance can be computed.

1.2 Organization

This thesis is presented in five parts. The first part is an overview of the governing fluid dynamic equations and the derivation of velocity potential which governs the inviscid and incompressible flowfield, as well as the derivation and description of related theorems and concepts which are necessary for the formulation of the numeric solution. The second part describes the formulation of two dimensional panel methods in three sections, starting with the formulation to solve the steady-state flowfield about a non-lifting body, adding the Kutta condition to solve the steady-state flowfield about a lifting body, and then accounting for time-dependent effects to solve the time-dependent flowfield about a lifting-body undergoing arbitrary motion. The third part of the thesis expands on the time-dependent panel method formulation by adding a freestream gust model which represents time dependent freestream pertubations using discrete vortex elements, and a two degree of freedom structural model which allows the responce of an arbitrary body due to aerodynamic forcing to be determined. The fourth part compares the developed panel code against classic analytic solutions for unsteady aerodynamics, and the last part demonstrates the application of the developed panel code to a forced responce problem using a solution which couples the freestream gust model with the structural model.

CHAPTER 2

FUNDAMENTALS

Before panel codes are discussed, this chapter defines several relations and terms used throughout the later discussion. The first section in the present chapter discusses the derivation of basic governing equations for fluid flow. The second section discusses potential flow and applies basic governing equations to the solution of potential flowfields. The last two sections relate terms and definitions used later in this thesis.

2.1 Governing Equations

The fundamental equations governing fluid flow are derived here from the relationships between density, momentum, and energy, and their time rates of change inside a control-volume.

2.1.1 Continuity

The continuity equation relates the time rate of change of mass inside a control-volume to the mass flux through the control-surface. The integral form of the continuity equation can be derived by beginning with a statement of mass inside a control-volume, such as

$$m_{c.v.} = \int_{c.v.} \rho \, dV \tag{2.1}$$

Based on Eq. (2.1), the time rate of change of mass inside the control-volume, $\partial m_{c.v.}/\partial t$, is given by

$$\frac{\partial m_{c.v.}}{\partial t} = \frac{\partial}{\partial t} \int_{c.v.} \rho \, dV \tag{2.2}$$

Mass flux through the control-surface can also be stated as

$$\dot{m}_{in} - \dot{m}_{out} = -\int_{c.s.} \rho q_i n_i \, dS \tag{2.3}$$

If mass is conserved, the net mass flux through the control-surface must equal the time rate of change of the mass within the control-volume, leading to the integral form of the continuity equation [8].

$$\frac{\partial}{\partial t} m_{c.v.} = \frac{\partial}{\partial t} \int_{c.v.} \rho \, dV = -\int_{c.s.} \rho q_i n_i \, dS = \dot{m}_{in} - \dot{m}_{out} \tag{2.4}$$

The divergence theorem states that given a vector q_i, the integral of the normal component of q_i relative to the control-surface equals the integral of the gradient of q_i inside the corresponding control-volume.

$$\int_{c.s.} q_i n_i \, dS = \int_{c.v.} \frac{\partial}{\partial x_i} q_i \, dV \tag{2.5}$$

By applying Eq. (2.5) to the integral form of the conservation equation, Eq. (2.4), the following simplification can be made

$$\int_{c.s.} \rho q_i n_i \, dS = \int_{c.v.} \frac{\partial}{\partial x_i} \left(\rho q_i \right) \, dV \tag{2.6}$$

Thus, Eq. (2.4) can be reduced to

$$\frac{\partial}{\partial t} \int_{c.v.} \rho \, dV + \int_{c.v.} \frac{\partial}{\partial x_i} \left(\rho q_i \right) \, dV = 0 \tag{2.7}$$

or

$$\int_{c.v.} \left(\frac{\partial}{\partial t} \rho + \frac{\partial}{\partial x_i} \left(\rho q_i \right) \right) \, dV = 0 \tag{2.8}$$

Since the volume integral in Eq. (2.8) must equal zero for any arbitrary control-volume, it must also hold that

$$\frac{\partial}{\partial t} \rho + \frac{\partial}{\partial x_i} \left(\rho q_i \right) = 0 \tag{2.9}$$

producing the differential form of the continuity equation [8].

2.1.2 Momentum

The momentum equation relates the time rate of change of fluid momentum through a control-volume to the forces acting on the control-volume. Momentum is a vector quantity, p_j, defined by the product of mass and the corresponding velocity vector.

$$p_j = mq_j \tag{2.10}$$

For a control-volume, the summation of forces acting on the volume equal the time rate of change of the control-volume momentum.

$$\sum_{c.v.} F_j = \frac{\partial}{\partial t}\left(mq_j\right)_{c.v.} \tag{2.11}$$

When Eq. (2.11) is incorporated with the continuity equation, Eq. (2.4) becomes

$$\sum_{c.v.} F_j = \frac{\partial}{\partial t}(mq_j)_{c.v.} = \frac{\partial}{\partial t}\int_{c.v.} \rho q_j \, dV + \int_{c.s.} \rho q_j q_i n_i \, dS \tag{2.12}$$

Forces acting on the control-volume may be either body forces, surface forces, or both.

$$\sum_{c.v.} F_{body_j} = \int_{c.v.} \rho f_j \, dV \tag{2.13}$$

$$\sum_{c.s.} F_{surface_j} = \int_{c.s.} \tau_{ij} n_i \, dS \tag{2.14}$$

Thus, substituting Eqs. (2.12) –(2.14) into Eq. (2.11) gives the integral form of the momentum equation.

$$\frac{\partial}{\partial t}\int_{c.v.} \rho q_j \, dV + \int_{c.s.} \rho q_j q_i n_i \, dS = \int_{c.v.} \rho f_j \, dV + \int_{c.s.} \tau_{ij} n_i \, dS \tag{2.15}$$

Applying the divergence theorem to Eqs. (2.12) and (2.14) allows simplification of Eq. (2.15)

$$\int_{c.s.} \rho q_j q_i n_i \, dS = \int_{c.v.} \frac{\partial}{\partial x_i}\left(\rho q_j q_i\right) \, dV \tag{2.16}$$

$$\int_{c.s.} \tau_{ij} n_i \, dS = \int_{c.v.} \frac{\partial}{\partial x_i} \tau_{ij} \, dV \tag{2.17}$$

$$\int_{c.v.} \left(\frac{\partial}{\partial t}(\rho q_j) + \frac{\partial}{\partial x_i}(\rho q_j q_i) - \rho f_j - \frac{\partial}{\partial x_i}\tau_{ij} \right) dV = 0 \tag{2.18}$$

Again, since the volume integral must equal zero for any arbitrary control-volume, it holds that

$$\frac{\partial}{\partial t}(\rho q_j) + \frac{\partial}{\partial x_i}(\rho q_j q_i) = \rho f_j + \frac{\partial}{\partial x_i}\tau_{ij} \tag{2.19}$$

producing the differential form of the momentum equation.

2.1.3 Navier-Stokes

If the assumption is made that the fluid is Newtonian (i.e. the stress components τ_{ij} are linearly related to the derivatives $\partial q_i / \partial x_j$), then the following substitution has been widely accepted

$$\tau_{ij} = -\left(p + \frac{2}{3}\mu \frac{\partial q_k}{\partial x_k} \right) \delta_{ij} + \mu \left(\frac{\partial q_i}{\partial x_j} + \frac{\partial q_j}{\partial x_i} \right) \tag{2.20}$$

Thus, Eq. (2.20) can be substituted into Eq. (2.19), giving conservative form of the Navier-Stokes relation [8].

$$\frac{\partial}{\partial t}(\rho q_j) + \frac{\partial}{\partial x_i}(\rho q_j q_i) = \rho f_i - \frac{\partial}{\partial x_j}\left(p + \frac{2}{3}\mu \frac{\partial q_k}{\partial x_k} \right) + \frac{\partial}{\partial x_i}\left[\mu \left(\frac{\partial q_i}{\partial x_j} + \frac{\partial q_j}{\partial x_i} \right) \right] \tag{2.21}$$

2.1.4 Euler

Depending on the flow regime, the Navier-Stokes equations can be simplified. For example, low-speed flow about a thin airfoil outside of the boundary layer can be assumed to be incompressible, $\rho =$ constant, and inviscid, $\mu = 0$, if the airfoil is at a conservative angle of attack and large Reynolds Numbers. With these two assumptions, Eq. (2.21) simplifies to the Euler equation.

$$\frac{\partial}{\partial t} q_j + \frac{\partial}{\partial x_i}(q_j q_i) = f_j + \frac{1}{\rho}\frac{\partial p}{\partial x_j} \tag{2.22}$$

2.2 Potential Flow

The potential flow assumption is of interest here because it describes the flow regime examined in the current investigation.

2.2.1 Velocity Potential

If a flowfield can be considered incompressible, then the continuity equation, Eq. (2.9), simplifies to $\partial q_i/\partial x_i = 0$. If the flowfield is also inviscid, $\mu = 0$, then vorticity in the flowfield must remain constant with respect to time, $\partial\zeta/\partial t = 0$. Given these assumptions, a scaler potential function Φ exists that is a solution to the Laplace equation describing the flowfield

$$\frac{\partial^2}{\partial x_j^2}\Phi = 0 \tag{2.23}$$

The potential function, Φ, is often denoted the velocity potential because the velocity field is equal to the gradient of Φ.

$$q_j = \frac{\partial}{\partial x_j}\Phi \tag{2.24}$$

Inversely, the potential at any point, P, in the flowfield can be calculated from any arbitrary reference point, P_0, by integrating the velocity field along any path between P_0 and P

$$\Phi(x_1, x_2, x_3) = \int_{P_0}^{P} x_1\, dx_1 + x_2\, dx_2 + x_3\, dx_3 = \int_{P_0}^{P} \frac{\partial\Phi}{\partial x_1}\, dx_1 + \frac{\partial\Phi}{\partial x_2}\, dx_2 + \frac{\partial\Phi}{\partial x_3}\, dx_3 \tag{2.25}$$

Note that with the assumptions of irrotationality and incompressibility, the integrand of Eq. (2.25) is an exact differential, and as such the potential is independent of the integration path. [8]

2.2.2 Superposition

Because the velocity potential describes the potential flowfield, and is the solution to the Laplace equation, it holds that [1]:

1. *Any irrotational incompressible flow has a velocity potential and stream function (for two-dimensional flow) that both satisfy Laplace's equation.*
2. *Conversely, any solution of Laplace's equation represents the velocity potential or stream function (two-dimensional) for an irrotational, incompressible flow.*

Since the Laplace equation is a second-order, linear, partial differential equation, it holds that the sum of two or more particular solutions is also a valid solution. Thus, a complex flowfield with a total potential Φ can be modeled as the superposition of multiple potential solutions, Φ_k, giving

$$\Phi = \sum \Phi_k \tag{2.26}$$

2.2.3 Boundary Conditions

Since solving the Laplace equation is a boundary value problem, applying the correct boundary conditions is essential. The two physical phenomon considered here are the no-flow boundary condition at the fluid-body interface, and the farfield condition forcing body-induced disturbances to decay to zero strength far from the body. There are two types of boundary condition formulations, the "direct" Neumann boundary condition, and the "indirect" Dirichlet boundary condition. The Dirichlet boundary condition is not explained here because it is not employed in this investigation. See References [1] and [8] for a full explanation.

The Neumann boundary condition specifies the normal velocity on the fluid-body boundary must equal zero,

$$\frac{\partial \Phi}{\partial n} = 0 \tag{2.27}$$

and the potential field due to the presence of the body must be negligible in the farfield ($r \to \infty$).

$$\lim_{r \to \infty} \left(\Phi_{body} \right) = 0 \tag{2.28}$$

2.3 Theorems And Relations

2.3.1 Bernoulii

To compute pressure in a potential flow, the relation between potential and velocity, $q_j = \partial\Phi/\partial x_j$, and the assumption of a conservative body force with potential E, $f_j = -\partial E/\partial x_j$, are substituted in to the Euler equation, Eq. (2.22).

$$\frac{\partial}{\partial x_j}\left(E + \frac{p}{\rho} + \frac{{q_j}^2}{2} + \frac{\partial\Phi}{\partial t}\right) = 0 \tag{2.29}$$

Thus, upon spatial integration

$$E + \frac{p}{\rho} + \frac{{q_j}^2}{2} + \frac{\partial\Phi}{\partial t} = C(t) \tag{2.30}$$

where $C(t)$ is a spatially independent constant over the entire flowfield, but is a function of time. This is the Bernoulli equation [8]. Because the left hand side of Eq. (2.30) is constant over the entire flowfield at a given point in time, pressure and velocity can be compared at different points in the flow if the potential is known.

2.3.2 Coefficient of Pressure

The pressure coefficient is a non-dimensional parameter relating pressure between two different locations in a flowfield.

$$C_p = \frac{p_\infty - p}{\frac{1}{2}\rho {q_j}^2_\infty} \tag{2.31}$$

Using the Bernoulli equation, Eq. (2.30), the pressure difference in Eq. (2.31) becomes

$$p_\infty - p = \rho\left[E + \frac{{q_j}^2}{2} + \frac{\partial\Phi}{\partial t}\right]_\infty - \rho\left[E + \frac{{q_j}^2}{2} + \frac{\partial\Phi}{\partial t}\right]_p \tag{2.32}$$

If care is taken with the choice of reference point, denoted by ∞, such that it exists at a location in the farfield not influenced by any body-induced disturbances, then the change in potential with time can be neglected at the reference point.

$$\frac{\partial \Phi_\infty}{\partial t} = 0 \tag{2.33}$$

If the reference point is also chosen such that the difference in the body forces is negligible,

$$E_\infty = E_p \tag{2.34}$$

then Eq. (2.32) reduces to

$$p_\infty - p = \rho \left[\frac{{q_j}_\infty^2}{2} - \frac{{q_j}_p^2}{2} - \frac{\partial \Phi_p}{\partial t} \right] \tag{2.35}$$

Dividing the pressure difference by the free stream dynamic pressure, $\frac{1}{2}\rho {q_j}_\infty^2$, Eq. (2.35) becomes

$$C_p = 1 - \frac{{q_j}_p^2}{{q_j}_\infty^2} - \frac{2}{{q_j}_\infty^2}\frac{\partial \Phi_p}{\partial t} \tag{2.36}$$

2.4 Angular Velocity, Vorticity, and Circulation

2.4.1 Motion of a Fluid Element

Motion of a fluid element is comprised of translation, rotation, and deformation, where each type of motion is usually caused by different phenomena in the flowfield. Translation is caused by a uniform velocity, where all parts of the element move at the same velocity, disallowing deformation and rotation. Rotation and deformation occur when velocity gradients exist across the element, as can be the case when viscous effects are not negligible.

2.4.2 Angular Velocity and Vorticity

The angular velocity of a fluid element relates to the element deformation caused by a velocity gradient. Generally these velocity gradients are caused by shear stresses. The angular velocity of a fluid element, ω_i, is defined as the curl of the velocity vector, or

$$\omega_i = -\frac{1}{2}\varepsilon_{ijk}\frac{\partial q_j}{\partial x_k} \tag{2.37}$$

Another measure of fluid angular velocity, used to simplify several equations, is vorticity, defined as twice the angular velocity.

$$\zeta_i = 2\omega_i = -\varepsilon_{ijk}\frac{\partial q_j}{\partial x_k} \tag{2.38}$$

2.4.3 Circulation

Circulation, Γ, is a measure of the vorticity in a fluid region, and equals the integral of the vorticity normal to a surface, S.

$$\Gamma = \int_S \zeta_i n_i \, dS \tag{2.39}$$

By substituting the definition of vorticity, Eq. (2.38), into Equation (2.39) and using Stokes Theorm [8],

$$\int_S -\varepsilon_{ijk}\frac{\partial q_j}{\partial x_k} n_i \, dS = \oint_C q_i \, dx_i \tag{2.40}$$

circulation can be defined as

$$\Gamma = \oint_C q_i \, dx_i \tag{2.41}$$

2.4.4 Kelvin's Theorem

Kelvin's theorem relates the time rate of change of circulation in a potential flow inside a closed region C. Simply stated, it states that the time rate of change of circulation in a closed fluid region must equal zero,

$$\frac{D\Gamma}{Dt} = 0 \tag{2.42}$$

or the total circulation of a closed fluid region is constant with time.

In the case of a lifting body, the body carries some bound circulation related to the body lift. If the body is at steady state, then lift and circulation are constant with time, and Eq. (2.42) is satisfied. For a body that is not at steady state, lift and circulation are functions of time. Therfore, to satisfy Eq. (2.42), another source of equal and opposite circulation must exist in the closed region. From physical observations, the additional circulation is known to be confined to a wake behind the body, Γ_{wake}, giving

$$\frac{D\Gamma}{Dt} = \frac{(\Gamma_{body} + \Gamma_{wake})}{\Delta t} = 0 \tag{2.43}$$

CHAPTER 3

PANEL CODES

This chapter describes the solution of two-dimensional potential flowfields using the Smith-Hess panel method [5]. The description starts with the solution of the flowfield about a non-lifting body, incorporates the Kutta condition to account for bound circulation, and then incorporates time-dependent effects to solve for time-dependent flowfields using the method of Basu and Hancock [3] as modified by Ardonceau [2].

The solution of the inviscid and incompressible flowfield about a non-lifting body represents the fundamental case to which a panel method can be applied. It also provides a starting point to describe the basic implementation of the panel method which will be expanded upon for the later lifting-body and time-dependent solutions.

3.1 Non-lifting Body

As described earlier, the inviscid and incompressible flowfield about a non-lifting body can be described by a potential field, which is the combination of the body and freestream potentials.

$$\Phi = \Phi_{body} + \Phi_{\infty} \tag{3.1}$$

To model the body potential, a distributed strength source sheet (of strength $\lambda(s)$) is placed along the fluid-body interface, s, as illustrated in Figure 3.1. This allows the body potential at an arbitrary point in the flow, $P\left(x_1, x_2\right)$, to be computed in terms of the potential due to the source sheet.

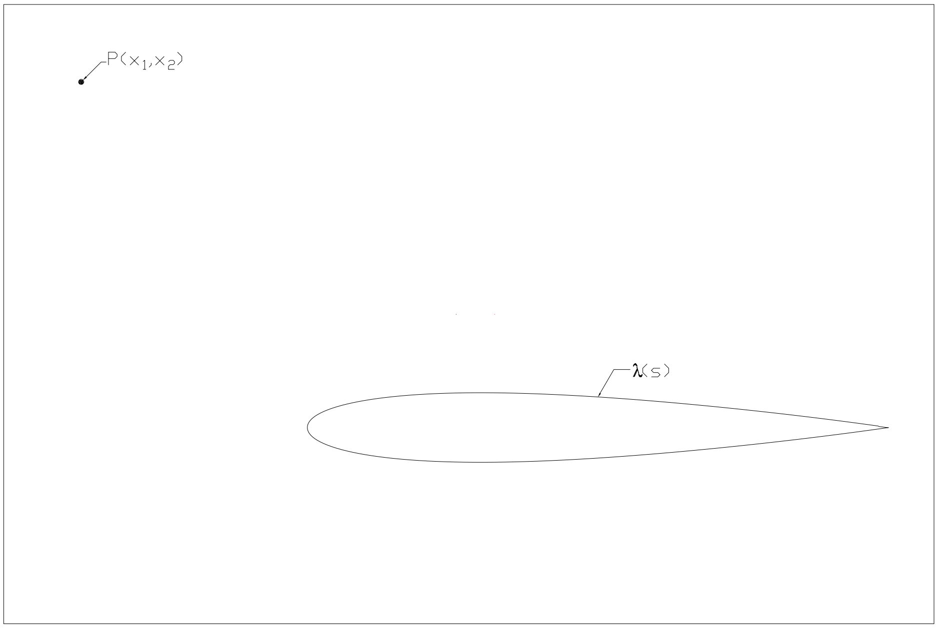

Figure 3.1. Airfoil modeled with a continuous source sheet.

$$\Phi_{body}(P) = \int_s \frac{\lambda(s)}{2\pi} \ln r \, ds \tag{3.2}$$

Correspondingly, of freestream flow is uniform, parallel to the x_1-axis, and the origin is a reference point where $\Phi_\infty(0,0) = 0$, the potential due to the freestream at point P is

$$\Phi_\infty(P) = q_{j\infty} x_j \tag{3.3}$$

Substituting Eqs. (3.2) and (3.3) into Eq. (3.1), gives the total potential at point P due to both the body and freestream.

$$\Phi(P) = \int_s \frac{\lambda(s)}{2\pi} \ln r \, ds + q_{j\infty} x_j \tag{3.4}$$

The only unknown parameter in Eq. (3.4) is the body source distribution, $\lambda(s)$. However, by applying the no-flow Neumann boundary condition from Eq. (2.27),

$$q_j n_j = n_j \frac{\partial \Phi}{\partial x_j} = 0 \tag{3.5}$$

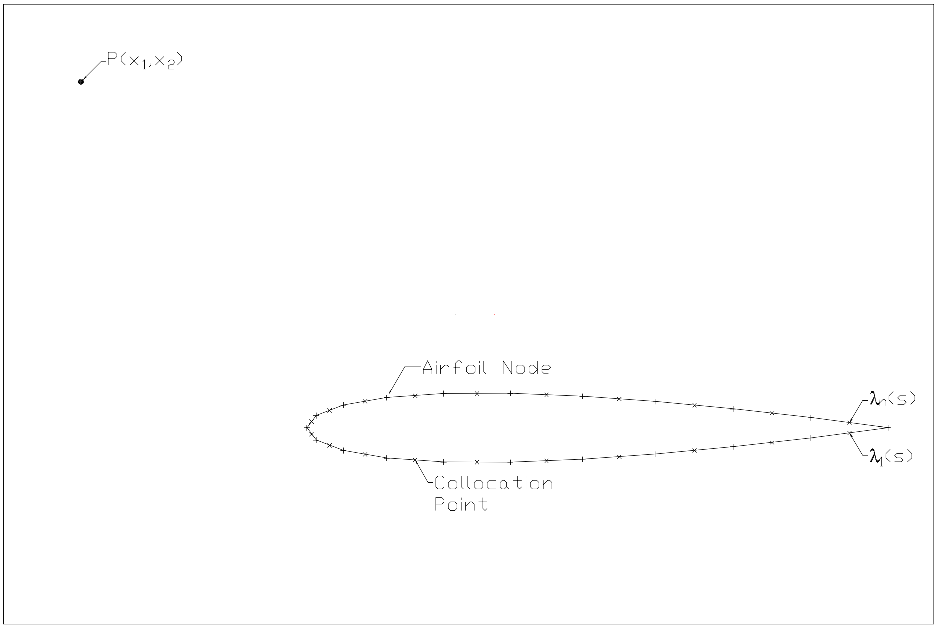

Figure 3.2. Airfoil discretized into constant strength source elements.

to the total potential at the fluid-body interface,

$$n_j \frac{\partial \Phi}{\partial x_j} = \int_s n_j \frac{\partial}{\partial x_j} \frac{\lambda(s)}{2\pi} \ln r \, ds + q_{j\infty} n_j = 0 \tag{3.6}$$

the unknown source distribution can be determined. Unfortunately, solving Eq. (3.6) for the source distribution is a non-trivial exercise for all but the simplest geometries. However, by applying geometric simplifications, determining the body source distribution as a function of body geometry and freestream conditions can be reduced to solving a set of linear equations.

3.1.1 Discretization

By discretizing the continuous source distribution, shown in Figure 3.1, into a series of straight segments, or panels, as shown in Figure 3.2, Eq. (3.6) may be reduced to a set of dependent linear equations. For this discussion, each panel represents a unique distributed source element having a constant source strength along the length of the element. A further simplification is made in that the no-flow boundary condition is not enforced at all locations on the body. Rather, the no-flow boundary conditions are applied to a single location, or collocation point, at the midpoint of each panel, as shown in Figure 3.2.

By discretizing the body into N panels, numbered clockwise from panel 1 at the lower body trailing-edge to panel N at the upper body trailing edge, the potential at the collocation point of any panel, panel α, can be determined as a function of freestream potential, body geometry, and panel strength distribution along the body. In this manner, the potential on panel α due to a source element on panel β and the freestream is

$$\Phi_{\alpha\beta} = \frac{\lambda_\beta}{2\pi} \int_\beta \ln r_{\alpha\beta} \, ds_\beta + q_{j\infty} x_{j\alpha} \tag{3.7}$$

The potential on panel α due to the entire body can be calculated using superposition. Thus, the potential on panel α due to the entire body is the sum of the potential due to the N panels on the body.

$$\Phi_\alpha = \sum_{\beta=1}^{N} \left(\frac{\lambda_\beta}{2\pi} \int_\beta \ln r_{\alpha\beta} \, ds_\beta \right) + q_{j\infty} x_{j\alpha} \tag{3.8}$$

Applying the no-flow boundary condition, Eq. (2.27), to Eq. (3.8) gives the normal velocity on panel α due to the body and freestream.

$$q_{n_\alpha} = \sum_{\beta=1}^{N} \left(\frac{\lambda_\beta}{2\pi} \int_\beta \frac{\partial}{\partial n_\beta} \ln r_{\alpha\beta} \, ds_\beta \right) + q_{j\infty} n_{j_\alpha} = 0 \tag{3.9}$$

As in Eq. (3.6), the source strengths in Eq. (3.9) are the unknown. However, because the parameters in the integrand of Eq. (3.9) are based strictly on body geometry, the integral can be replaced by a geometric influence coefficient, $a_{\alpha\beta}$, which represents the geometric influence of panel β on panel α.

$$a_{\alpha\beta} = \frac{1}{2\pi} \int_\beta \frac{\partial}{\partial n_\alpha} \ln r_{\alpha\beta} \, ds_\beta \tag{3.10}$$

Using the influence coefficient method, the no-flow normal condition on panel α, given previously in Eq. 3.9 becomes

$$\sum_{\beta=1}^{N} (\lambda_\beta \, a_{\alpha\beta}) = -q_{j\infty} n_{j_\alpha} \tag{3.11}$$

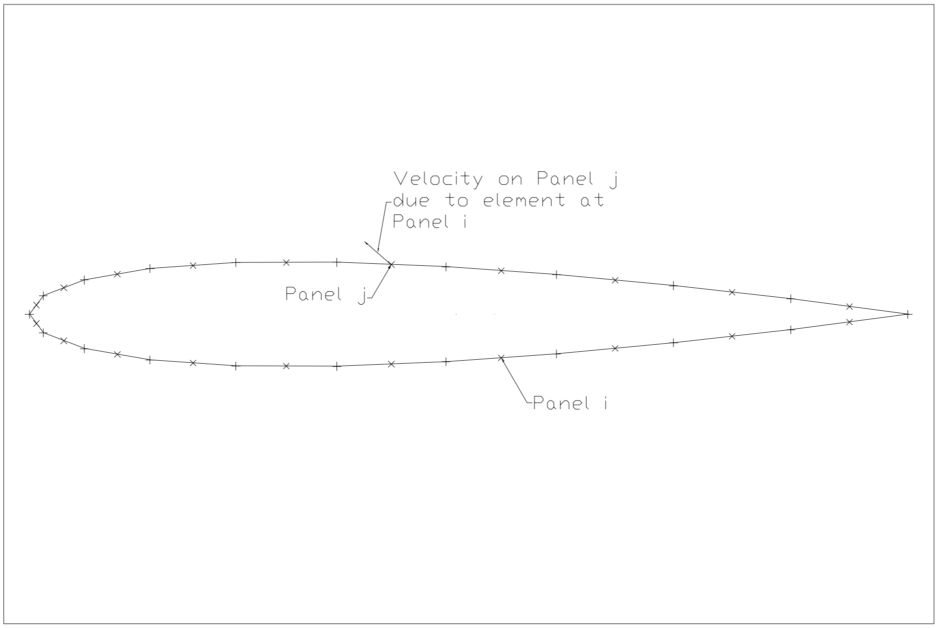

Figure 3.3. Constant Strength Panel Discretization

Equation (3.11) is the basis for a set of linear equations relating the unknown panel source strengths λ_β to the no-flow boundary condition. This system of equations begins with the influence matrix, $A_{\alpha\beta}$, which is made up of the influence coefficients, $a_{\alpha\beta}$, based only on the body geometry.

$$A_{\alpha\beta} = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1N} \\ a_{21} & a_{22} & ... & a_{2N} \\ ... & ... & & ... \\ a_{N1} & a_{N2} & ... & a_{NN} \end{pmatrix} \tag{3.12}$$

The element strengths for each panel are stored in the column vector x_β.

$$x_\beta = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ ... \\ \lambda_N \end{pmatrix} \tag{3.13}$$

Finally, the column vector B_α represents normal velocity components at the collocation point not induced by the body, such as the freestream normal velocity.

$$B_\alpha = \begin{pmatrix} -q_{j\infty} n_{j_1} \\ -q_{j\infty} n_{j_2} \\ \ldots \\ -q_{j\infty} n_{j_N} \end{pmatrix} \tag{3.14}$$

Combined, these matrices and vectors form a system of equations $A_{\alpha\beta} x_\beta = B_\alpha$, or

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1N} \\ a_{21} & a_{22} & \ldots & a_{2N} \\ \ldots & \ldots & & \ldots \\ a_{N1} & a_{N2} & \ldots & a_{NN} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \ldots \\ \lambda_N \end{pmatrix} = \begin{pmatrix} -q_{j\infty} n_{j_1} \\ -q_{j\infty} n_{j_2} \\ \ldots \\ -q_{j\infty} n_{j_N} \end{pmatrix} \tag{3.15}$$

the solution of which is trivial, or non-unique, describing the potential-flow about the non-lifting body. In physical terms, the trivial solution does not include the effects of bound circulation about the body, and therefore does not model lift.

3.2 Lifting Body

To model the effects of lift and bound circulation about a body, additional constraints must be considered. To model bound circulation on the body, a set of constant strength vortex panels, each of the same strength, are added to the existing source panel discretization. Since each vortex panel has the same strength, only a single variable must be added to the set of linear equations modeling the non-lifting solution. The additional variable necessitates an additional constraint to solve for the vortex panel strength. This additional constraint is provided by the Kutta condition, which is based on observations of physical flow phenomena about a lifting body, or airfoil, with a sharp trailing-edge.

3.2.1 Kutta Condition

The Kutta condition is a means to relate possible potential-flow solutions about a body to observed physical flow characteristics, thereby generating a unique solution for the flowfield. The general definition of the Kutta condition specifies that the flow must detach from the airfoil at the airfoil trailing-edge and that the trailing-edge has zero loading [8]. The aforementioned potential-flow solution described for the non-lifting body possess a trailing-edge singularity, thus the Kutta condition as specified can not be satisfied at the airfoil trailing-edge. For a lifting body, a commonly used first approximation is employed in which the zero loading condition is enforced on the panels adjacent to the airfoil trailing-edge.

For a discretized airfoil, the condition of zero trailing-edge loading is approximately satisfied by specifying equal pressure on the airfoil upper and lower trailing-edge panels. The unsteady Bernoulli equation, Eq. (2.30), is used to relate the fluid flow on the upper, u, and lower, l, panels, giving

$$\left[\frac{p}{\rho} + \frac{q_j^2}{2} + \frac{\partial \Phi}{\partial t}\right]_l = \left[\frac{p}{\rho} + \frac{q_j^2}{2} + \frac{\partial \Phi}{\partial t}\right]_u \tag{3.16}$$

For a steady-state flow, the time-dependent potential terms can be neglected in Eq. (3.16), and the condition of equal pressure simplifies to the specification of equal flow velocity on the airfoil upper and lower trailing-edge panels.

$$q_{j_l} = q_{j_u} \tag{3.17}$$

Thus, Eq. (3.17) provides the additional constraint necessary to solve for the unique panel strengths on the airfoil in a steady-state flow.

3.2.2 Equations

Placing vortex panels along the airfoil does not change the no-flow boundary condition described in Eq. (2.27), but the potential at panel α due to panel β and the freestream must now include the influence of the discreyized vortex sheet.

$$\Phi_{\alpha\beta} = \frac{\lambda_\beta}{2\pi} \int_\beta \ln r_{\alpha\beta}\, ds_\beta - \frac{\gamma}{2\pi} \int_\beta \theta_{\alpha\beta}\, ds_\beta \tag{3.18}$$

Accordingly, the potential at panel α due to the N source panels, N vortex panels, and the freestream influence along the body becomes

$$\Phi_\alpha = \sum_{\beta=1}^{N} \left(\frac{\lambda_\beta}{2\pi} \int_\beta \ln r_{\alpha\beta}\, ds_\beta \right) - \gamma \sum_{\beta=1}^{N} \left(\frac{1}{2\pi} \int_\beta \theta_{\alpha\beta}\, ds_\beta \right) + q_{j\infty} x_{j_\alpha} \tag{3.19}$$

Hence, the normal velocity on panel α due to the N panels and freestream can be writen in terms of the potential gradient normal to the body at panel α

$$q_{n_\alpha} = \sum_{\beta=1}^{N} \left(\frac{\lambda_\beta}{2\pi} \int_\beta \frac{\partial}{\partial n_\alpha} \ln r_{\alpha\beta}\, ds_\beta \right) - \gamma \sum_{\beta=1}^{N} \left(\frac{1}{2\pi} \int_\beta \frac{\partial}{\partial n_\alpha} \theta_{\alpha\beta}\, ds_\beta \right) + q_{j\infty} n_{j_\alpha} = 0 \tag{3.20}$$

Again, the integrand for the circulatory term in Eq. (3.20) is based solely on body geometry and therefore may be calculated as a geometric influence coefficient, b_α, representing the influence of the N discretized vortex panels on panel α.

$$b_\alpha = - \sum_{\beta=1}^{N} \left(\frac{1}{2\pi} \int_\beta \frac{\partial}{\partial n_\alpha} \theta_{\alpha\beta}\, ds_\beta \right) \tag{3.21}$$

Substituting the influence coefficients, Eq. (3.10) and Eq. (3.21), the no-flow boundary condition gives

$$\sum_{\beta=1}^{N} (\lambda_\beta\, a_{\alpha\beta}) + \gamma\, b_\alpha = -q_{j\infty} n_{j_\alpha} \tag{3.22}$$

Equation (3.22) still provides N equations, but there are now $N + 1$ variables (N source strengths, λ_α, and one vortex strength, γ) describing the potential field about the lifting body. The Kutta condition, Eq. (3.17), provides the $N + 1$'th condition needed to solve the linear system of equations for the source and vortex strengths.

Using the no-flow boundary condition to simplify the Kutta condition (i.e. all flow on the trailing-edge panels must be tangential) the tangential flow velocity on panel α can calculated in terms of the potential gradient along the body. The tangential flow velocity on panel α due to panel β and the freestream is therefore

$$q_{s_{\alpha\beta}} = \frac{\lambda_\beta}{2\pi} \int_\beta \frac{\partial}{\partial s_\beta} \ln r_{\alpha\beta} \, ds_\beta + \frac{\gamma}{2\pi} \int_\beta \frac{\partial}{\partial s_\alpha} \theta_{\alpha\beta} \, ds_\beta + q_{j_\infty} s_j \tag{3.23}$$

giving a tangential flow velocity on panel α due to all N body panels of

$$q_{s_\alpha} = \sum_{\beta=1}^{N} \left(\frac{\lambda_\beta}{2\pi} \int_\beta \frac{\partial}{\partial s_\alpha} \ln r_{\alpha\beta} \, ds_\beta \right) + \gamma \sum_{\beta=1}^{N} \left(\frac{1}{2\pi} \int_\beta \frac{\partial}{\partial s_\alpha} \theta_{\alpha\beta} \, ds_\beta \right) + q_{j_\infty} s_j \tag{3.24}$$

Examining Eq. (3.24), two new influence coefficients are introduced, $c_{\alpha\beta}$, the tangential flow component along panel α due to source panel β

$$c_{\alpha\beta} = \frac{1}{2\pi} \int_\beta \frac{\partial}{\partial s_\alpha} \ln r_{\alpha\beta} \, ds_\beta \tag{3.25}$$

and d_α, the tangential flow component along panel α due to the N body vortex panels.

$$d_\alpha = \sum_{\beta=1}^{N} \left(\frac{1}{2\pi} \int_\beta \frac{\partial}{\partial s_\alpha} \theta_{\alpha\beta} \, ds_\beta \right) \tag{3.26}$$

Rewriting the steady-state Kutta condition, Eq. (3.17), in terms of the geometric influence coefficients,

$$q_{s_l} = \sum_{\beta=1}^{N} (\lambda_\beta \; c_{1\beta}) + \gamma \; d_1 = \sum_{\beta=1}^{N} (\lambda_\beta \; c_{N\beta}) + \gamma \; d_N = q_{s_u} \tag{3.27}$$

and rearranging to position the terms on the left hand side,

$$\sum_{j=1}^{n} (\lambda_j \;\; (c_{nj} - c_{1j})) + \gamma \;\; (d_n - d_1) = 0 \tag{3.28}$$

gives the Kutta condition in a suitable form to incorporate into the system of linear equations, Eq. (3.15).

Rewriting Eq. (3.15) to include the vortex influence and the Kutta condition, the $A_{\alpha\beta}$ matrix becomes,

$$A_{\alpha\beta} = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1N} & b_1 \\ a_{21} & a_{22} & ... & a_{2N} & b_2 \\ ... & ... & & ... & ... \\ a_{N1} & a_{N2} & ... & a_{NN} & b_N \\ (c_{N1} - c_{11}) & (c_{N2} - c_{12}) & ... & (c_{NN} - c_{1N}) & (d_N - d_1) \end{pmatrix} \tag{3.29}$$

the x_β vector becomes,

$$x_j = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ ... \\ \lambda_N \\ \gamma \end{pmatrix} \tag{3.30}$$

and the B_α vector becomes,

$$B_i = \begin{pmatrix} -q_{j_\infty} n_{j_1} \\ -q_{j_\infty} n_{j_2} \\ ... \\ -q_{j_\infty} n_{j_N} \\ 0 \end{pmatrix} \tag{3.31}$$

Combining Eqs. (3.29), through (3.31) gives the linear system of equations which model the flowfield about the lifting body,

$$\begin{pmatrix} a_{11} & a_{12} & ... & a_{1N} & b_1 \\ a_{21} & a_{22} & ... & a_{2N} & b_2 \\ ... & ... & & ... & ... \\ a_{N1} & a_{N2} & ... & a_{NN} & b_N \\ (c_{N1} - c_{11}) & (c_{N2} - c_{12}) & ... & (c_{NN} - c_{1N}) & (d_N - d_1) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ ... \\ \lambda_N \\ \gamma \end{pmatrix} = \begin{pmatrix} -q_{i_\infty} n_{i_1} \\ -q_{i_\infty} n_{i_2} \\ ... \\ -q_{i_\infty} n_{i_N} \\ 0 \end{pmatrix} \tag{3.32}$$

providing a unique solution to the flowfield which includes the effects of lift and bound circulation in the solution.

3.3 Time-Dependent Solutions

The non-lifting and lifting body solutions described in Sections 3.1 and 3.2 provide methods to model steady-state flowfields about a body in a uniform freestream flow. If the body is in motion relative to the freestream, or if the freestream includes perturbations about its time-average mean, assumptions neglecting time-dependent terms are no longer valid and a time-dependent solution methodology must be found.

The basic formulation of a time-dependent solution is similar to that of the lifting body solution; i.e. the body is discretized using source and vortex panel discretization, the no-flow boundary condition provides N linear equations, and the Kutta condition provides the one additional relation necessary to formulate a unique lifting-body solution. The difference between the time-dependent and steady-state solutions is the application of the Kutta condition and the incorporation of a model to account for the airfoil wake.

The following method describes a solution for the time-dependent flowfield about an airfoil in motion relative to the influence of an otherwise uniform freestream flow.

3.3.1 Frame of Reference

The choice of coordinate system and reference frame determine the complexity of the mathematical model. For this discussion, the no-flow boundary condition is calculated in a body-fixed coordinate system that is allowed to translate and pitch in the global reference frame with velocity $q_{j_{rel}}$ and pitch rate Ω.

3.3.2 Wake

In a viscous solution for attached flow over an airfoil, a low energy boundary layer along the airfoil is shed into the freestream flow from the airfoil trailing edge to form the airfoil wake. The wake represents a pertubation of the freestream flow aft of the airfoil due to the flow about the airfoil. The wake is significant because it can have a profound influence on the flow about the airfoil, even as the wake convects with the freestream. Since viscous effects are neglected in a potential-flow solution, an airfoil wake must be modeled in a way representing the effect of shed bound circulation to satisfy the Kelvin theorm. As such, the wake is often called a "time history" because it represents the change in bound circulation on the airfoil with time.

Because the inviscid wake represents the change in bound circulation on the airfoil with time, or the shed circulation, it is possible to model the wake using a set of discrete vortex elements. Basu and Hancock [3] use a set of point vortices and a single constant strength vortex panel to model the inviscid time-dependent wake. The strength and orientation of the wake vortex panel, or wake panel, play a key roll in satisfying the time-dependent Kutta condition (explained in detail later in this chapter). The the strength of the wake panel is dependent on the amount of circulation shed by the airfoil between time steps, and wake panel orientation is determined by the Kutta condition.

After the Kutta condition has been satisfied, calculations necessary to determine the time-dependent flowfield solution have been performed, and any necessary post solution calculations have been completed, all wake vortices are convected with the local flow in preparation for the next time step. The wake panel is not convected with the local flow, however, rather the wake panel is replaced by a single point vortex of strength equal to the shed circulation from the previous timestep. This new point vortex is then allowed to convect with the local flow. In this manner, the wak panel and point vortices model shed circulation from the airfoil, which in turn can influence the flow about the airfoil.

3.3.3 Unsteady Kutta Condtion

The specification of the time-dependent Kutta condition is similar to the steady-state specification described in Section 3.2.1. The difference is in the application of the time-dependent Kutta condition, which can no longer neglect time-dependent terms in the unsteady Bernoulli equation relating pressures on the upper and lower airfoil trailing-edge panels. The inclusion of time-dependent terms means that the time-dependent Kutta condition is a quadratic equation which must be solved iteratively.

Two applications of the time-dependent Kutta condition are described below, the method of Basu and Hancock [3], and the modification of that method by Ardonceau [2] used in the current investigation.

3.3.3.1 Basu-Hancock

Basu and Hancock propose that "...there is no definitive statement of the Kutta condition for a steady airfoil, each mathematical model requiring its own consistent 'Kutta' condition to ensure a unique solution..." [3] Based on that statement, the assumption that the flow separates from the airfoil at the airfoil trailing-edge, zero loading exists across the shed vorticity at the trailing-edge, and zero loading occurs across the trailing-edge elements of the airfoil, Basu and Hancock propose the folowing mathematical model for the Kutta condition. This model determines the orientation, θ_k, length, Δ_k, and strength, $(\gamma_w)_k$, of the wake panel at time t_k.

Beginning with the unsteady Bernoulli equation applied at the airfoil trailing-edge panels

$$\left[\frac{p}{\rho} + \frac{q_j^2}{2} + \frac{\partial \Phi}{\partial t}\right]_l = \left[\frac{p}{\rho} + \frac{q_j^2}{2} + \frac{\partial \Phi}{\partial t}\right]_u \tag{3.33}$$

and specifying equal pressure at the trailing-edge,

$$\frac{p_l - p_u}{\rho} = \frac{q_{j_u}^2}{2} - \frac{q_{j_l}^2}{2} + \frac{\partial \Phi_u}{\partial t} - \frac{\partial \Phi_l}{\partial t} = 0 \tag{3.34}$$

a quadratic relation develops for the flow velocity on the upper and lower airfoil trailing-edge panels. Because the velocity relation is not linear, an iterative solution is necessary to determine the orientation and strenth of the wake panel which satisfies the Kutta condition.

Using the Kelvin theorem, Eq (2.42), the rate of change of circulation about the airfoil must be balanced by the rate of change of the shed circulation in its wake,

$$\frac{\Delta_k \left(\gamma_w\right)_k}{\Delta t} = -\frac{\partial \Gamma}{\partial t} = -\frac{\Gamma_{k-1} - \Gamma_k}{\Delta t} \tag{3.35}$$

or the change in circulation about the airfoil from t_{k-1} to t_k must be balanced by an equal and opposite circulation about the wake panel.

$$\Delta_k \left(\gamma_w\right)_k = \Gamma_k - \Gamma_{k-1} \tag{3.36}$$

The rate of change of potential across the airfoil trailing-edge is related to the rate of change in circulation by

$$\frac{\partial \left(\Phi_l - \Phi_u\right)}{\partial t} = \frac{\partial \Gamma}{\partial t} \tag{3.37}$$

Therefore, substituting Eq. (3.37) into Eq. (3.34) relates the upper and lower trailing-edge velocities to the rate of change of circulation about the airfoil.

$$\frac{q_{j_u}^2 - q_{j_l}^2}{2} + \frac{\Gamma_k - \Gamma_{k-1}}{t_k - t_{k-1}} = 0 \tag{3.38}$$

Substituting Eq. (3.36) into Eq. (3.38) gives the circulation strength about the wake panel in terms of trailing-edge panels velocities and wake panel length.

$$\left(\gamma_w\right)_k = \frac{\left(t_k - t_{k-1}\right)\left(q_l^2 - q_u^2\right)}{2\Delta_k} \tag{3.39}$$

Wake panel orientation is determined by local velocity on the wake panel, neglecting the effect of the wake panel on itself,

$$\tan\theta_k = \frac{\left(q_{1_w}\right)_k}{\left(q_{2_w}\right)_k} \tag{3.40}$$

and wake panel length is proportional to the magnitude of the local velocity and the time step.

$$\Delta_k = \left(q_{j_w}\right)_k \left(t_k - t_{k-1}\right) \tag{3.41}$$

3.3.3.2 Ardonceau

Ardonceau proposed a modification to Basu and Hancock's Kutta condition based on experimental studies [2]. The modified solution method is nearly identical to that of Basu and Hancock, but the wake panel geometry is altered. Instead of allowing the wake panel to change both orientation and length, the wake panel orientation is fixed along the bisector between the upper and lower trailing-edge panels.

$$\theta_k = \frac{\theta_u + \theta_l}{2} \tag{3.42}$$

The length of the Ardonceau wake panel then equals the average of the trailing-edge panel velocities porportional to the time step.

$$\Delta_k = \frac{1}{2}\left(q_{j_u} + q_{j_l}\right)_k \left(t_k - t_{k-1}\right) \tag{3.43}$$

The calculation of wake panel strength is the same as Eq. (3.39).

3.3.4 Method of Solution

Regardless of the mathematical formulation of the unsteady Kutta condition, the solution methods are the same. As in the steady-state solutions, the N source strengths, one vortex strength, and freestream along the body are related through the no-flow boundary condition which gives a system of N linear equations. As outlined above, however, the Kutta condition becomes a quadratic relation in an unsteady flow which must be solved using an iterative techique.

The no-flow boundary condition for the time-dependent solution also includes induced velocity terms due to body motion relative to the freestream and induced velocity terms due to the airfoil wake. Modifying Eq. (3.22) to include the effects of body rotation,

$$q_{j_{rotation}} = \Omega \times r_\alpha \tag{3.44}$$

body translation,

$$q_{j_{translation}} = q_{j_{rel}} \tag{3.45}$$

and the influence of the wake panel and point vortices,

$$q_{j_{wake}} = \gamma_w \; b_{\alpha N+1} + \sum_{\beta=1}^{k-1} \Gamma_\beta \left(\frac{\partial}{\partial n_\alpha} \frac{\theta_{\alpha\beta}}{2\pi} \right) \tag{3.46}$$

the time dependent no-flow relation becomes

$$\sum_{\beta=1}^{N} (\lambda_\beta \; a_{\alpha\beta}) + \gamma \; \sum_{\beta=1}^{N} b_{\alpha\beta} + \gamma_w \; b_{\alpha N+1} + \sum_{\beta=1}^{k-1} \Gamma_\beta \left(\frac{\partial}{\partial n_\alpha} \frac{\theta_{\alpha\beta}}{2\pi} \right) + (q_{j_\infty} + \Omega \times r_\alpha + q_{j_{rel}})\, n_{j_\alpha} = 0 \tag{3.47}$$

Rearranging to place all non-source terms on the right hand side gives

$$\sum_{\beta=1}^{N} (\lambda_\beta \; a_{\alpha\beta}) = -\gamma \; \sum_{\beta=1}^{N} b_{\alpha\beta} - \gamma_w \; b_{\alpha N+1} - \sum_{\beta=1}^{k-1} \Gamma_\beta \left(\frac{\partial}{\partial n_\alpha} \frac{\theta_{\alpha\beta}}{2\pi} \right) - (q_{j_\infty} + \Omega \times r_\alpha + q_{j_{rel}})\, n_{j_\alpha} \tag{3.48}$$

Note that Eq. (3.48) is very similar to Eq. (3.11) but with extra terms on the right hand side. Therefore, rewriting Eq. (3.14) to include the new terms of Eq. (3.48) gives

$$B_i = \begin{pmatrix} -\gamma \; \sum_{\beta=1}^{N} b_{1\beta} - \gamma_w \; b_{1N+1} - \sum_{\beta=1}^{k-1} \Gamma_\beta \left(\frac{\partial}{\partial n_1} \frac{\theta_{1\beta}}{2\pi} \right) - (q_{j_\infty} + \Omega \times r_1 + q_{j_{rel}})\, n_{j_1} \\ -\gamma \; \sum_{\beta=1}^{N} b_{2\beta} - \gamma_w \; b_{2N+1} - \sum_{\beta=1}^{k-1} \Gamma_\beta \left(\frac{\partial}{\partial n_2} \frac{\theta_{2\beta}}{2\pi} \right) - (q_{j_\infty} + \Omega \times r_2 + q_{j_{rel}})\, n_{j_2} \\ \dots \\ -\gamma \; \sum_{\beta=1}^{N} b_{N\beta} - \gamma_w \; b_{NN+1} - \sum_{\beta=1}^{k-1} \Gamma_\beta \left(\frac{\partial}{\partial n_N} \frac{\theta_{N\beta}}{2\pi} \right) - (q_{j_\infty} + \Omega \times r_N + q_{j_{rel}})\, n_{j_N} \end{pmatrix} \tag{3.49}$$

Substituting Eq. (3.49) for Eqs. (3.14) in Eq. (3.15) gives a linear system of equations for the time-dependent solution.

$$
\begin{pmatrix}
a_{11} & a_{12} & \dots & a_{1n} \\
a_{21} & a_{22} & \dots & a_{2n} \\
\dots & \dots & & \dots \\
a_{n1} & a_{n2} & \dots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
\lambda_1 \\
\lambda_2 \\
\dots \\
\lambda_n
\end{pmatrix}
=
$$

$$
\begin{pmatrix}
-\gamma\ \sum_{\beta=1}^{N} b_{1\beta} - \gamma_w\ b_{1N+1} - \sum_{\beta=1}^{k-1} \Gamma_\beta \left(\frac{\partial}{\partial n_1}\frac{\theta_{1\beta}}{2\pi}\right) - (q_{j\infty} + \Omega \times r_1 + q_{j_{rel}})\, n_{j_1} \\
-\gamma\ \sum_{\beta=1}^{N} b_{2\beta} - \gamma_w\ b_{2N+1} - \sum_{\beta=1}^{k-1} \Gamma_\beta \left(\frac{\partial}{\partial n_2}\frac{\theta_{2\beta}}{2\pi}\right) - (q_{j\infty} + \Omega \times r_2 + q_{j_{rel}})\, n_{j_2} \\
\dots \\
-\gamma\ \sum_{\beta=1}^{N} b_{N\beta} - \gamma_w\ b_{NN+1} - \sum_{\beta=1}^{k-1} \Gamma_\beta \left(\frac{\partial}{\partial n_N}\frac{\theta_{N\beta}}{2\pi}\right) - (q_{j\infty} + \Omega \times r_N + q_{j_{rel}})\, n_{j_N}
\end{pmatrix}
\tag{3.50}
$$

An iterative solution scheme is used to find the unique solution satisfying both the system of N linear equations and one quadratic relation.

The iterative Kutta condition assumes initial values for the wake panel orientation, length, and circulation strength at the initialization of the simulation. The inital values are used in the solution of N linear equations to determine the strength of the body source elements. The calculated body source strengths are then used to recalculate the orientation, length, and circulation strength of the wake panel, and the process is repeated until the orientation, length, and circulation strength of the wake panel meet a given convergence criteria.

CHAPTER 4

CODE DESCRIPTION

To facilitate investigations into the interaction between elastic airfoil response and arbitrary aerodynamic forcing, two components are added to the time-dependent panel code described in Section 3.3. The first component is a gust model originally proposed by Basu and Hancock [3] which uses singularity elements to model the influence of a sharp edge gust. The second component is a structural model which, when coupled with the aerodynamic model, determines the airfoil responce to aerodynamic forcing. This chapter describes the gust and structural models, as well as their implementation and integration into the unsteady panel code.

4.1 Frame of Reference

The source and vortex elements modeling the airfoil, wake, and freestream perturbations are tracked in a Lagrangian reference frame. The origin of this reference frame is located at the leading edge of the undisturbed airfoil, with the airfoil trailing edge lying on the x_1 axis.

4.2 Gust Model

The gust model uses singularity elements to induce velocity perturbations about an otherwise uniform freestream flow. Because this investigation is initially interested in the effect of transverse velocity perturbations, or perturbations perpendicular to the time-averaged freestream flow, the gust is modeled by a set of vortex sheets. For this discussion, a vortex sheet will be defined as a collection of vortex elements, each sharing at least one end point with a neighboring vortex element. Collectively, these vortex elements produce a continuous

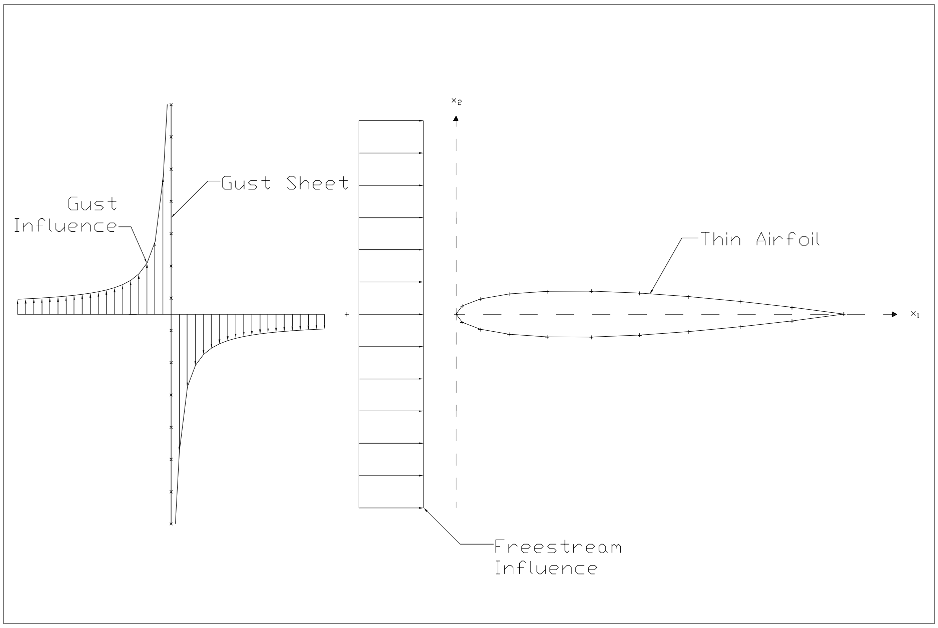

Figure 4.1. Influence of a Vortex sheet located in the Freestream flow compared to the Freestream influence.

vorticity “sheet” that convects with the freestream flow. Each vortex sheet possess a finite amount of bound circulation that remains constant with time, and is initially distributed evenly along the length of the sheet. The influence of a single gust sheet in the freestream flow is shown in Figure 4.1.

By placing gust sheets into the flowfield prior to initialization of the simulation, and specifying a time-invariant total circulation about each gust sheet, Kelvin’s Theorm is implicitly satisfied. Thus, the Kutta condition discussed in Section 3.3 remains valid.

4.2.1 Deformation

The key to properly modeling the transverse gust, such that the gust responds to the influence of the airfoil and its wake, lies in modeling gust convection. To allow the gust sheet to deform and react to the influence of the airfoil, wake, and other elements in the flowfield, each gust sheet is discretized into a finite number of panels, or elements. At the end of each computational time step, the gust sheet is convected by propagating the endpoints of each gust element with the local fluid flow. In this manner, the gust sheet is alowed to deform

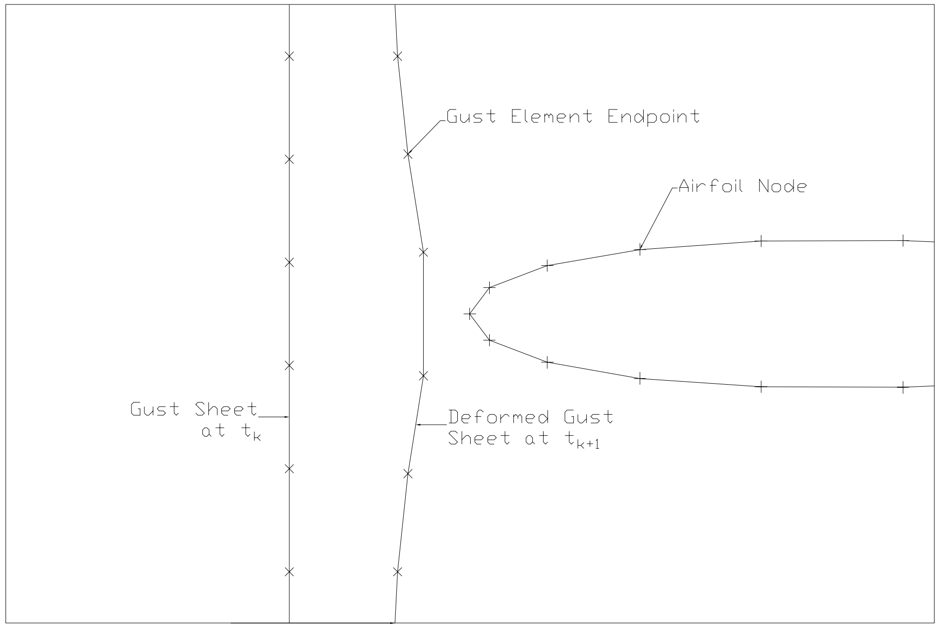

Figure 4.2. Deformation of a vortex sheet approaching the airfoil leading edge.

due to local velocity gradients in the flow. Figure 4.2 illustrates the deformation of a gust sheet as it approaches the airfoil leading edge. Because the total bound circulation about each gust element is time-invariant, the influence of each gust element on the surrounding fluid flow is a function of element length.

4.2.2 Airfoil-Gust Interaction

Each gust sheet initialized upstream of the airfoil eventually encounters the airfoil as it convects with the freestream flow. Since the gust sheet provides aerodynamic forcing for forced-response simulations, proper modeling of airfoil-gust interaction is a critical aspect of the overall gust model.

To properly model gust sheet influence on the airfoil, the gust sheet must propagate around the airfoil, not propagate through the airfoil. Therefore, the continous gust sheet must be “split” when the gust sheet encounters the forward-most edge of the airfoil, allowing one section of the sheet to convect across the upper airfoil surface and the other to convect

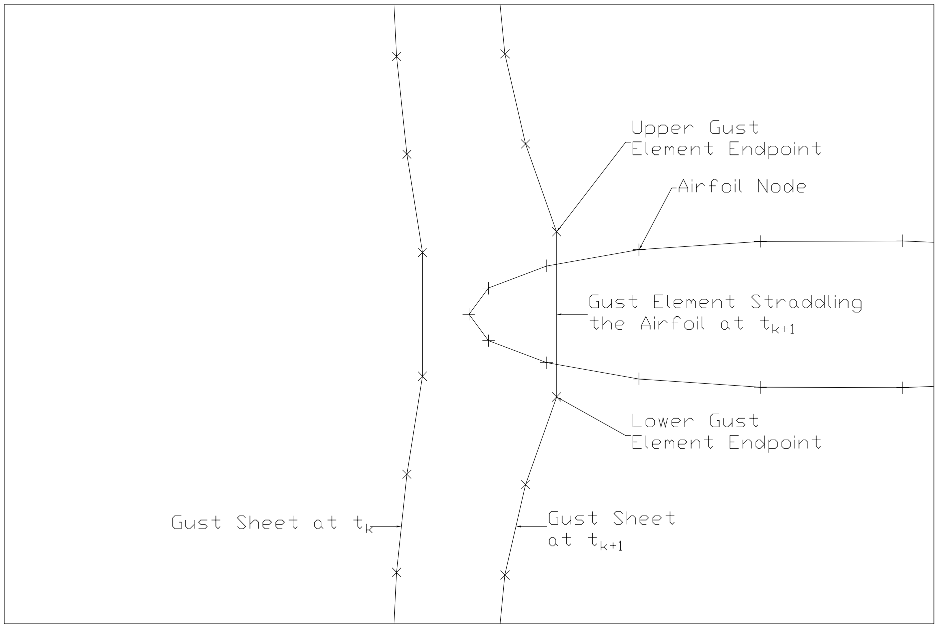

Figure 4.3. Case One: Gust element straddling the airfoil.

along the lower airfoil surface. Techniques used to determine if and when a gust sheet must be split, and methods used to split each gust sheet are described below.

Two distinct cases can arise when a gust sheet reaches the airfoil. Case One involves a gust element straddling the airfoil leading edge. In this case, one endpoint attempts to convect above the airfoil while the other endpoint convects below, as illustrated in Figure 4.3. To maintain the no-flow boundary condition, the gust element straddling the airfoil must be split into two separate elements, one ending on the upper airfoil surface and one ending on the lower airfoil surface. The two "new" elements must also posses a combined bound circulation equal to the bound circulation of the original gust element to satisfy Kelvin's theorem.

Case Two involves a gust element, or pair of elements sharing an endpoint, where the endpoint attempts to convect into the airfoil interior, as illustrated in Figure 4.4. To maintain the no-flow boundary condition, the erroneous endpoint must be to the either the upper or lower airfoil surface. In this case, no new gust elements are created, and each affected elements retains its original bound circulation.

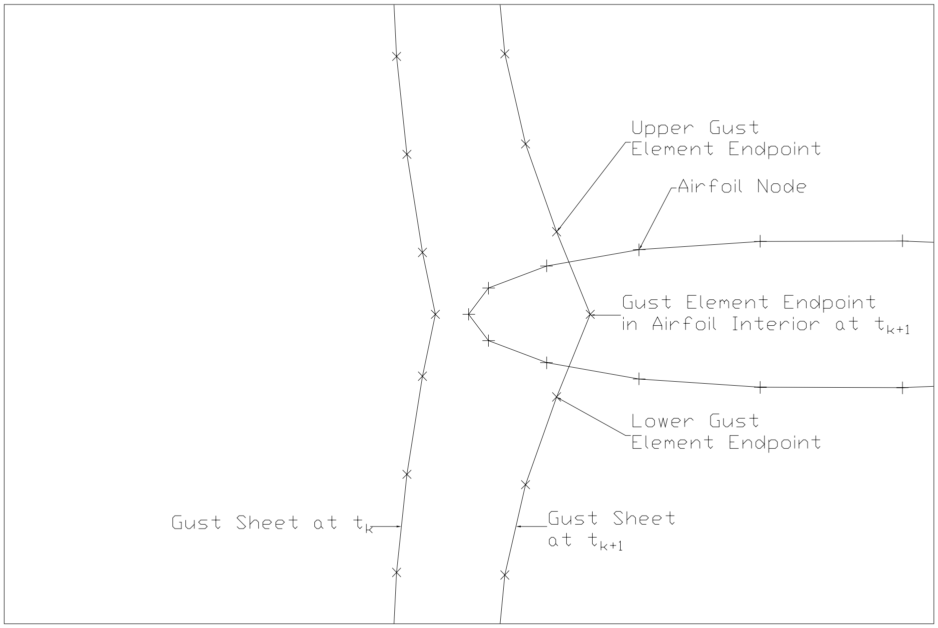

Figure 4.4. Case Two: Gust element endpoint convected into the airfoil.

In each case, once the erroneous elements have been relocated to the airfoil surface, the element endpoints lying on the airfoil surface are convected along the airfoil at the surface tangential velocity, until the gust element propagates past the airfoil trailing edge.

4.2.2.1 Determining Gust Element Condition

To ensure a gust sheet does not breach the airfoil interior, the following conditions are checked for each gust element after it is convected in preparation for the next time step.

1. Does the gust element currently terminate on the airfoil surface?

2. Is either element endpoint located between the airfoil leading and trailing edges in the x_1 direction?

3. Is one element endpoint located above the airfoil while the other is located below the airfoil in the x_2-direction?

4. Is either gust element endpoint located inside the airfoil surface?

Based on the four conditions, the state of the gust element with respect to the airfoil can be determined. If condition 1 is true, the gust element must be convected along the airfoil at the surface tangential velocity instead of the local flow velocity. If condition 1 is false and conditions 2 and 3 are true, the gust element is an example of Case One and must be split. If condition 1 is false and conditions 2 and 4 are true, the gust element is an example of Case Two and the element endpoint must be relocated to the airfoil surface. If conditions 1 through 4 are false, the gust element is located in the freestream and no gust sheet modifications are required.

4.2.2.2 Case One

Case One involves splitting a gust element straddling the airfoil, as illustrated in Figure 4.3, and determining the bound circulation about the split gust elements. Because the airfoil may be at some arbitrary orientation relative to the time-averaged freestream flow, the airfoil leading-edge node may not be the airfoil node the gust sheet first encounters, therefore the term "forward most" edge, or node, will be defined as the node closest to the gust when the gust impacts the airfoil. In addition, depending on airfoil orientation and the influence of the freestream (including the gust), the upstream stagnation point on the airfoil may not correspond to either the forward-most airfoil node or the airfoil leading edge. This distinction is subtle, as illustrated in Figure 4.5. The variance in x_1 location between the leading edge and stagnation point may only be a few hundredths of a chord length, but the influence of this variance on resulting airfoil forcing can be significant.

For example, consider the case where a gust element is split about the airfoil leading edge at time t_{k+1}, but the upstream stagnation point on the airfoil does not correspond to the airfoil leading edge. If the upstream stagnation point is located on the lower airfoil surface, the gust element ending on the lower surface between the airfoil leading edge and the upstream stagnation point will convect towards the airfoil leading edge at time t_{k+2} instead of towards the airfoil trailing edge, as desired. This process is depicted in Figures 4.6 and

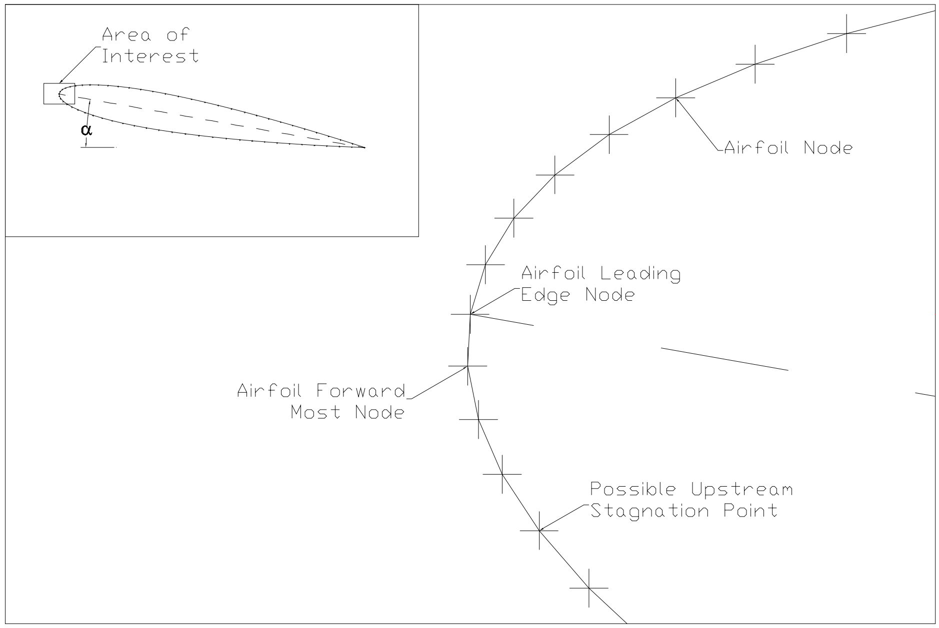

Figure 4.5. Airfoil leading edge vs. the airfoil forward-most node.

4.7. In fact, the lower gust element will eventually propagate around the leading edge and convect towards the airfoil trailing edge along the upper surface. This will stretch the gust element through the airfoil, invalidating the no-flow boundary condition.

A similar circumstance occurs if the gust element is simply split about the upstream stagnation point. For example, if the upstream stagnation point does not correspond to the leading edge, but rather lies on the lower airfoil surface, the gust element propagating above the airfoil will stretch through the airfoil and end on the lower airfoil surface, as illustrated in Figure 4.8. The upper gust element will eventually propagate around the airfoil leading edge before it propagates towards the trailing edge along the upper airfoil surface, as desired, but this gives the gust element an undue influence on the airfoil as it propagates around the airfoil leading edge and will invalidate the no-flow boundary condition.

Because the upstream stagnation point and the forward-most node both exhibit large influences on the gust element, gust elements straddling the airfoil leading edge are split about both the forward-most airfoil node and the upstream stagnation point. In this manner, if the upstream stagnation point is located on the lower airfoil surface, the lower gust element will

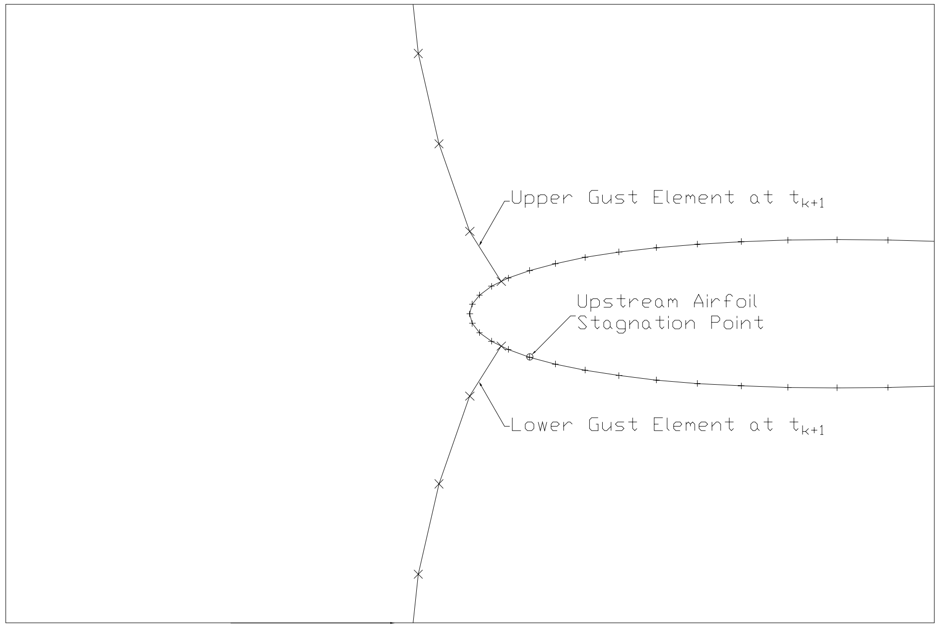

Figure 4.6. Gust element split about the leading edge with the upstream stagnation point on the lower airfoil surface at time t_{k+1}.

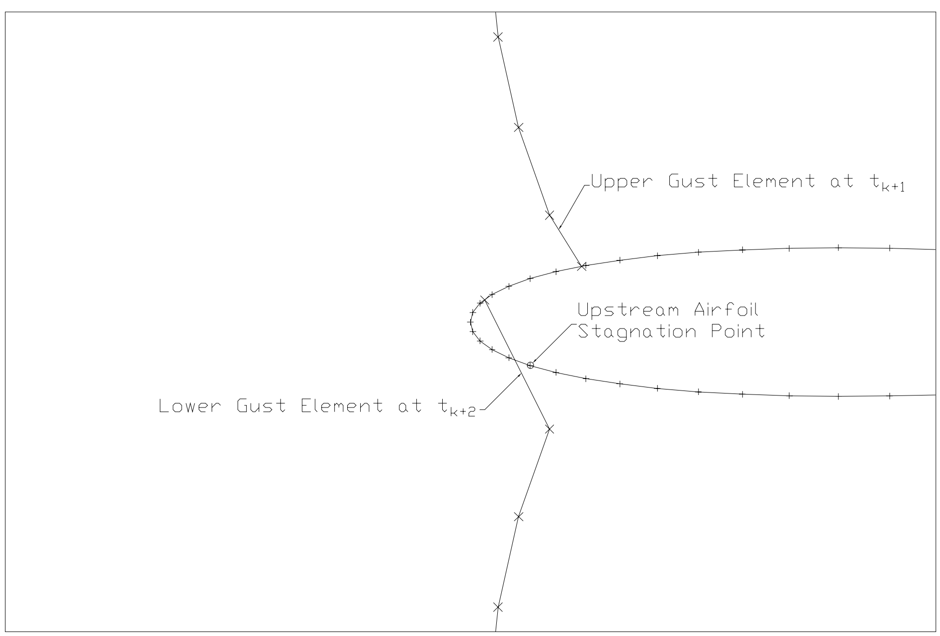

Figure 4.7. Gust element split about the leading edge with the upstream stagnation point on the lower airfoil surface at time t_{k+2}.

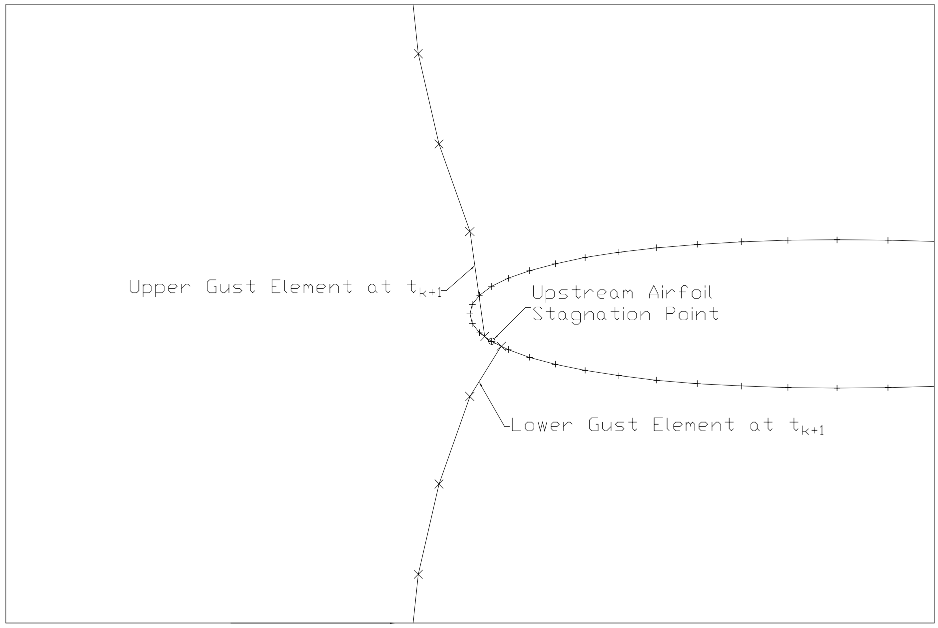

Figure 4.8. Gust element split about the upstream stagnation point.

convect along the airfoil from the upstream stagnation point while the upper gust element will convect along the airfoil from the forward-most airfoil node, and visa versa for an upstream stagnation point located on the upper airfoil surface. In most cases, this distinction is negligible, but the method ensures that split gust elements will not convect towards the airfoil leading edge, or stretch through the airfoil surface.

4.2.2.3 Implementation

As mentioned, once it has been determined that a gust element straddles the airfoil, the element must be split into two "new" elements, one convecting above the airfoil and one convecting below the airfoil. Because the unsteady panel code models the flowfield using discrete time steps, it is unlikely that the instant a gust element impacts the forward-most airfoil node will correspond exactly to a panel code time step. Therefore, an interpolation routine is employed to accurately determine the instant in time a gust element impacts the forward-most airfoil node, as illustrated in Figure 4.9. It is necessary to know this time because, for example, if a gust element impact occurs at midway between timesteps, the

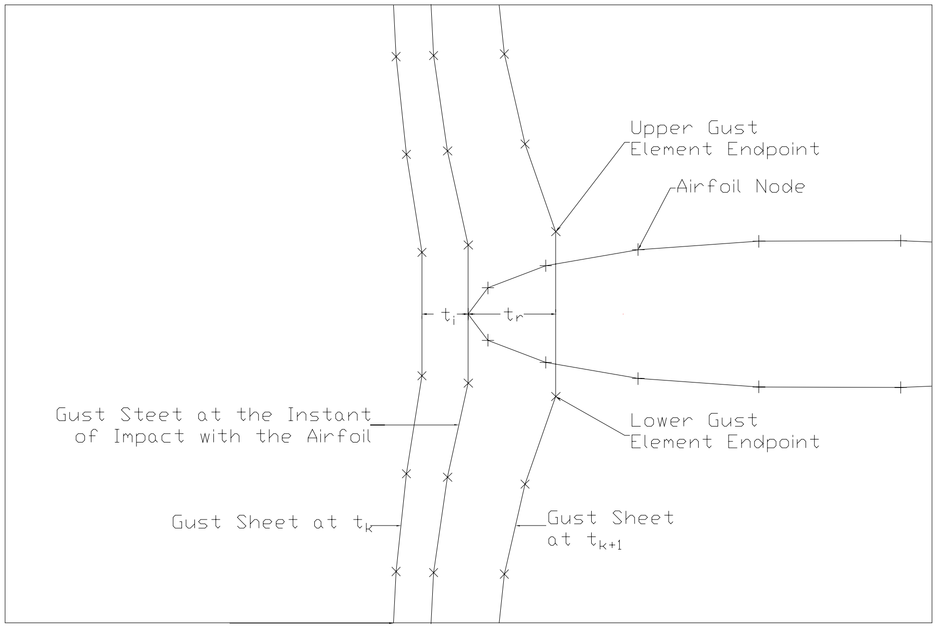

Figure 4.9. Interpolation to determine the time of Gust-Airfoil impact.

element should be convected along the airfoil during the remaining amount of the time step after being split. In addition to accurately determining the instant in time that a gust element impacts the airfoil, the interpolation routine also provides information regarding what percentage of the original gust element should convect above and below the airfoil. Knowing these percentages is necessary so proper fractions of the original bound circulation can be assigned to each "new" gust element, thereby maintaining a constant total circulation in the flow.

4.2.2.4 Case Two

Case Two involves a gust element, or pair of elements, possessing an endpoint that convects into the closed airfoil surface, as illustrated in Figure 4.4. This is the less common of the two cases, and for a simulation with a suitably small time step only occurs if the initial gust sheet contains an element possessing an endpoint close to the x_1 axis. Therfore, in an effort to simplify the panel code, this case is controled through well considered initial discretization of the gust sheet.

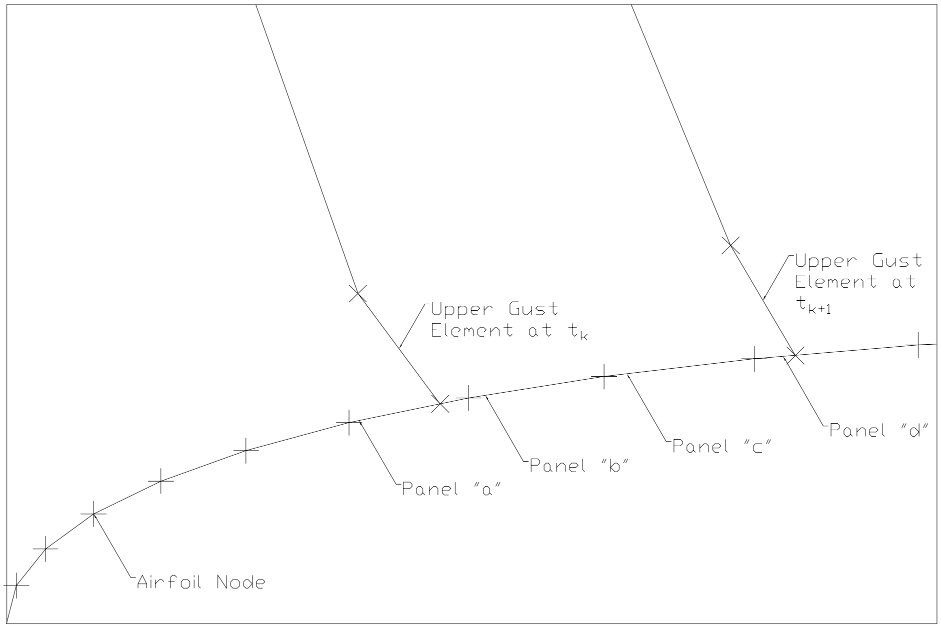

Figure 4.10. Gust element convection along the upper airfoil surface.

4.2.3 Convection

For a gust element ending on the airfoil surface, the endpoint on the surface is convected at the surface tangential velocity instead of the local flow velocity. Because the airfoil itself is discretized into a set of discrete panels and the no-flow boundary condition is enforced only at each panel midpoint, a gust element endpoint convected at the local flow velocity could convect into the airfoil surface, or off the airfoil surface into the freestream flow. Basu and Hancock [3] calculated the surface tangential velocity at the gust element endpoint by interpolating tangential velocities across adjacent airfoil panels. The interpolated surface tangential velocity value was then multiplied by the local time step to find the distance the element endpoint should convect along the airfoil surface. This method provides a good first aproximation for coarse airfoil discretizations, but fails for finely discretized airfoils in locations where a large velocity gradient exists between adjacent panels, such as at the airfoil leading edge.

To acount for large tangential velocity gradients, an alternate method of convecting a gust element along the airfoil surface has been developed. This alternate method estemates the amount time nessisary to convect the gust endpoint along a surface panel based on the length of the surface panel and the surface tangential velocity at the panel midpoint. The estimated time to convect the gust element endpoint to the end of the surface panel is compared to the amount of time remaining in the computational timestep. Based on whether the estimated time is greater than the remaning time step, a decision is made to convect the endpoint a fractional distance along the surface panel, based on the surface tangential velocity and the remaining time step, or to convect the endpoint to the end of the current surface panel, and repeat the time estimation on the next surface panel.

For example, to convect the gust element endpoint initially located at some location along Panel a, as depicted in Figure 4.10, the distance between the gust endpoint and the downstream node of Panel a is used with the surface tangential velocity at the midpoint of Panel a to estimate the amount of time necessary to convect the gust endpoint to the downstream node of Panel a. If the estimated time to convect the gust endpoint to the end of Panel a is less than the local time step, Δt, or for convenience, the time remaining, t_r, then the gust endpoint is relocated to the downstream airfoil node shared by Panels a and b, and the estimated time is subtracted from t_r. In this manner, using the lengths of Panels b, c, and d, along with their respective tangential velocities, the time necessary to convect the gust endpoint across Panels b, c, and d is estimated to be greater than t_r, but the time necessary to convect the gust endpoint across only Panels b and c is less than t_r. Thus, the gust endpoint is relocated to the shared airfoil node between Panels c and d, and the estimated time to convect the gust endpoint across Panels b and c is subtracted from t_r. Because the time necessary to convect across Panel d is greater than t_r, the gust endpoint is relocated a fractional distance along Panel d, as determined using the surface tangential velocity at the midpoint of Panel d and t_r.

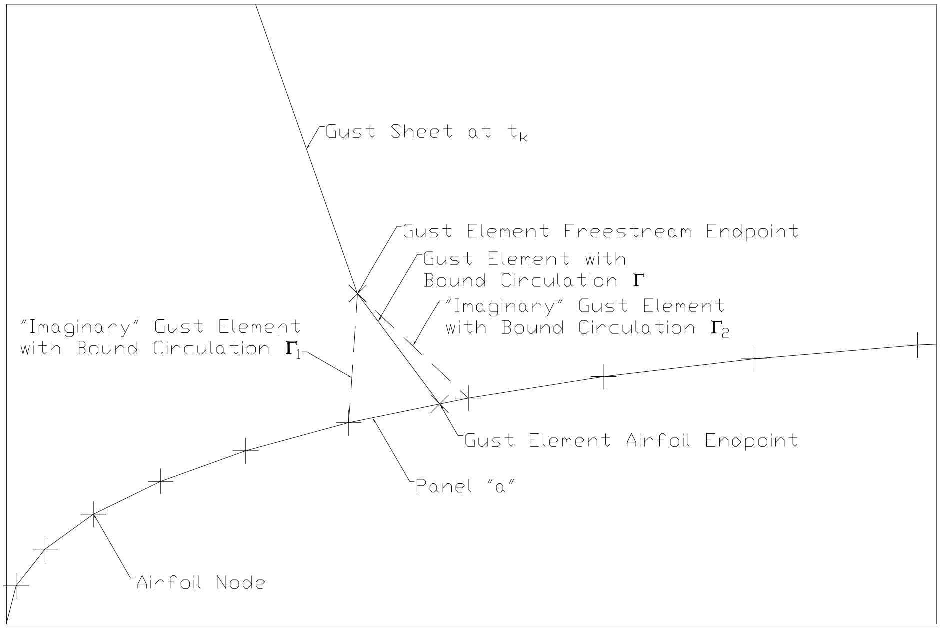

Figure 4.11. Gust element convection along the upper airfoil surface.

4.2.4 Gust Influence on the Airfoil

Since the gust sheet is composed of singularity elements, each with an influence proportional to $1/r$, the influence of a gust element ending on the airfoil depends on the proximity of the element endpoint to an airfoil collocation point. If a gust element ends on a collocation point, r approaches zero and the influence of that gust element becomes infinite. This skews the flowfield solution in a non-physical manner. Basu and Hancock [3] prevented this possibility by replacing the each gust element ending on the airfoil with a pair of "imaginary" elements, illustrated in Figure 4.11. The two imaginary gust elements share the freestream endpoint with the original gust element, but instead of terminating at some location along an airfoil panel, airfoil panel a, with the original gust element, the imaginary element pair terminate at corresponding endpoints of airfoil panel a. The imaginary elements share the bound circulation of the original gust element in a manner dependent on the location of the original element endpoint on panel a. As such, the influence of the gust continues to propagate across the airfoil surface but the possibility of discontinuities arrising due to a gust element coexisting with an airfoil colocation point is eliminated.

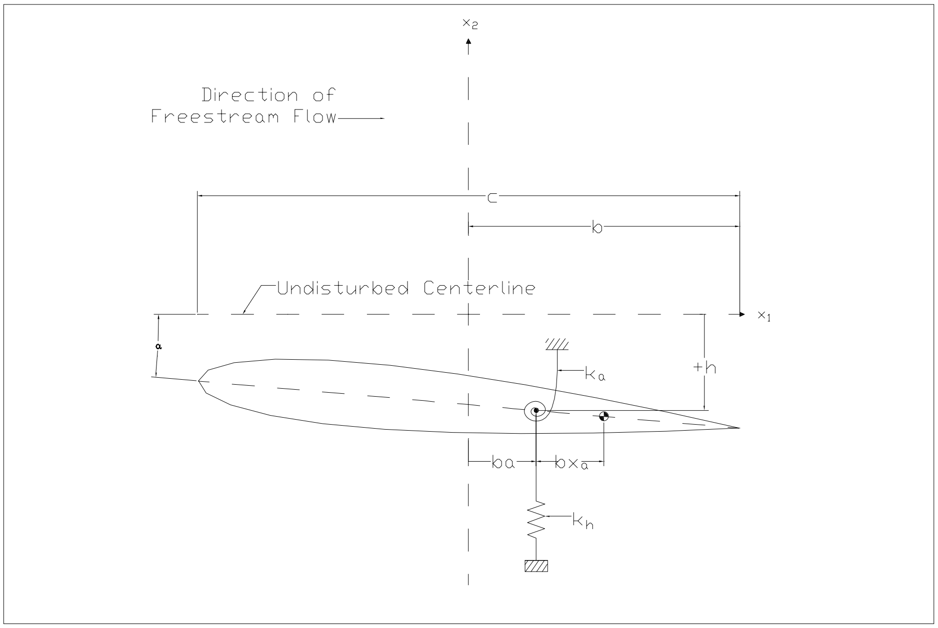

Figure 4.12. Pitching and Plunging Airfoil

4.3 Free Response

The inclusion of an airfoil structural model enables the unsteady panel code to model time-dependent airfoil response due to arbitrary and self-induced aerodynamic forcing.

4.3.1 Model

The airfoil structural model is a two degree of freedom (TDOF) spring-mass system allowing coupled airfoil motion in rotation and translation, or pitch and plunge. Figure 4.12 shows a basic schematic detailing parameters important to the model. The equations governing two-dimensional body motion in terms of sectional characteristic and generalized external forces are

$$m\ddot{h} + S_\alpha \ddot{\alpha} + m\omega_h^2 h = Q_h \tag{4.1a}$$

$$S_\alpha \ddot{h} + I_\alpha \ddot{\alpha} + I_\alpha \omega_\alpha^2 \alpha = Q_\alpha \tag{4.1b}$$

For a thin airfoil, the generalized external forces correspond to aerodynamic lift and moment about the elastic axis.

$$Q_h = - L \tag{4.2a}$$

$$Q_\alpha = M_y \tag{4.2b}$$

For compatibility with the developed unsteady panel code, which calculates non-dimensional forces and moments through integration of instantanious surface-pressure coefficents, Eq. (4.1) is non-dimensionalized with respect to chord, freestream velocity, time, and mass. The resulting non-dimensional equations of motion are then rewritten in terms of the non-dimensional sectional characteristics, such as density ratio, μ, radius of gyration, r_α, static imbalance, x_α, reduced bending frequency, k_h, reduced pitching frequency, k_α, normalized plunge, $\hat{h}$, normalized pitch, $\hat{\alpha}$, and non-dimensional time, $\hat{t}$.

$$\mu\hat{h}'' + \frac{x_\alpha\mu}{2}\hat{\alpha}'' + \mu k_h^2\hat{h} = \frac{2}{\pi}C_l \tag{4.3a}$$

$$\frac{x_\alpha\mu}{2}\hat{h}'' + \frac{r_\alpha^2\mu}{4}\hat{\alpha}'' + \frac{r_\alpha^2\mu k_\alpha^2}{4}\hat{\alpha} = \frac{2}{\pi}C_{m_y} \tag{4.3b}$$

Expressing Eq. (4.3) in matrix notation,

$$\Big[M\Big]\Big\{X\Big\}'' + \Big[K\Big]\Big\{X\Big\} = \Big\{F\Big\} \tag{4.4}$$

where

$$\begin{bmatrix} M \end{bmatrix} = \begin{bmatrix} \mu & \frac{x_\alpha \mu}{2} \\ \frac{x_\alpha \mu}{2} & \frac{r_\alpha^2 \mu}{4} \end{bmatrix} \tag{4.5a}$$

$$\begin{bmatrix} K \end{bmatrix} = \begin{bmatrix} \mu k_h^2 & 0 \\ 0 & \frac{r_\alpha^2 \mu k_\alpha^2}{4} \end{bmatrix} \tag{4.5b}$$

$$\begin{Bmatrix} X \end{Bmatrix} = \begin{Bmatrix} \hat{h} \\ \hat{\alpha} \end{Bmatrix} \tag{4.5c}$$

$$\begin{Bmatrix} F \end{Bmatrix} = \frac{2}{\pi} \begin{Bmatrix} C_l \\ C_{m_y} \end{Bmatrix} \tag{4.5d}$$

The second derivative in Eq. (4.4) can be isolated on the LHS,

$$\begin{Bmatrix} X \end{Bmatrix}'' = \begin{bmatrix} M \end{bmatrix}^{-1} \begin{Bmatrix} F \end{Bmatrix} - \begin{bmatrix} M \end{bmatrix}^{-1} \begin{bmatrix} K \end{bmatrix} \begin{Bmatrix} X \end{Bmatrix} \tag{4.6}$$

allowing the equations of motion to be writen as a set of coupled first order ordinary differential equations.

$$\begin{Bmatrix} X \end{Bmatrix}' = \begin{Bmatrix} Y \end{Bmatrix} \tag{4.7a}$$

$$\begin{Bmatrix} Y \end{Bmatrix}' = \begin{bmatrix} M \end{bmatrix}^{-1} \begin{Bmatrix} F \end{Bmatrix} - \begin{bmatrix} M \end{bmatrix}^{-1} \begin{bmatrix} K \end{bmatrix} \begin{Bmatrix} X \end{Bmatrix} \tag{4.7b}$$

Airfoil orientation and position at time t_{k+1} is determined by solving Eq. (4.7) with a fourth-order Runge-Kutta method using non-dimensional aerodynamic forces computed at time t_k.

4.3.2 Solution

The aerodynamic solution and TDOF structural model are coupled directly in the developed unsteady panel method to calculate free and forced response of an arbitrary thin airfoil. The non-dimensional forces and moments calulated at each time step are used as inputs to the structural model, predicting airfoil orientation and position at the next time-step. The new airfoil position and orientation are used to calculate new non-dimensional aerodynamic forces, and the process is repeated.

4.4 Forced Response

By coupling the gust model described in Section 4.2 and the structural model described in Section 4.3, airfoil responce to arbitrary gust induced forcing can be modeled. As will be shown in Chapter 5, the influence of multiple gust sheets can be superimposed to model periodic freestream disturbance having arbitrary shapes, frequencies, and amplitudes. Thus, airfoil responce due to external forcing can be systematically studied by varying the characteristics of the freestream gust.

CHAPTER 5

CODE VERIFICATION

To verify the accuracy and applicability of the developed panel code, a set of test cases were examined. These test cases compare unsteady panel code simulations with fundamental problems in unsteady aerodynamics having known analytical or computational solutions. In this manner, the accuracy and applicability of the panel code is established prior to its extension to problems of interest not having known solutions.

5.1 Wagner

The Wagner problem, one of the fundamental problems in unsteady aerodynamics, explores the lift response of a flat plate to a flowfield which is instantaneously accelerated from one equilibrium state to another. The problem demonstrates the effect of body wake development on lift and moment during transition between equilibrium states.

5.1.1 Description

Consider a stationary flat plate, or airfoil of infinitesimal thickness, at some angular orientation relative to a freestream flow, α_0, illustrated in Figure 5.1. At time $t < 0$, the magnitude of the freestream relative to the flat plate is zero, $q_\infty = 0$. Since the no-flow boundary condition is implicitly satisfied, the body produces zero lift, and perhaps more importantly, carries zero bound circulation. At time $t = 0$, the freestream instantaneously accelerates to a finite non-zero velocity, $q_\infty = c$. By applying the unsteady Kutta condition and no-flow boundary condition discussed in Section 3.3, a lifting solution can be found for the flat plate.

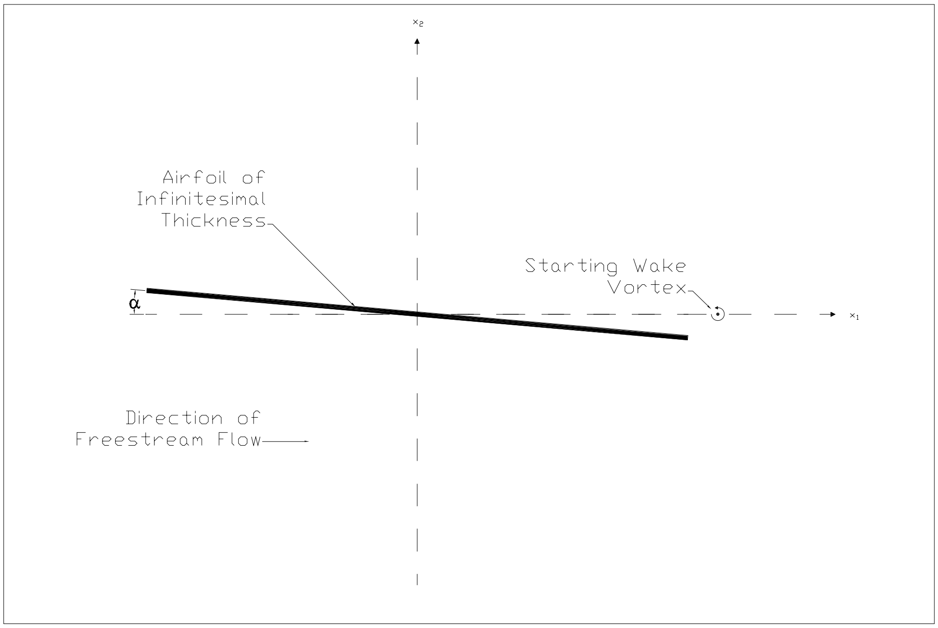

Figure 5.1. Flat plate at time $t = 0$

It should be recalled from Section 3.3 that the body wake for an inviscid solution represents shed bound vorticity from the body which is necessary to satisfy Kelvin's theorem. As such, the shed vorticity magnitude in the wake at time $t = 0$ equals the magnitude of the bound circulation change about the body, but in the opposite direction. The shed circulation caused by the flowfield transition between equilibrium states is often called a "starting vortex" because the magnitude of this vortex is significantly greater then the rest of the wake. Shed vorticity in the wake produces an aerodynamic downwash on the body, influencing the no-flow boundary condition. Wake influence on lift is normally of a small magnitude relative to the freestream and the relative body motion, but in the case of a starting vortex where the shed circulation magnitude is on the same order as the bound circulation about the body, the wake-induced downwash suppresses lift generation on the body. As such, the starting vortex significantly influences lift development on the body until the starting vortex propagates into the far field.

5.1.2 Solution

By modeling the induced body wake as a continuous vortex sheet of varying strength, originating at the body trailing edge and oriented parallel to the freestream flow, Wagner [4] developed a time-accurate solution for lift on an instantaneously accelerated flat plate.

$$L = 2\pi b \rho q^2 \alpha_0 \phi\left(s\right) \tag{5.1}$$

This solution depends on a modified Bessel function known as the Wagner function, $\phi\left(s\right)$.

$$\phi\left(s\right) = \frac{1}{2\pi i}\int_{-\infty}^{\infty} \frac{C\left(k\right)}{k}\, e^{iks}\, dk \tag{5.2}$$

An approximate representation [4] of the Wagner function has been computed as,

$$\phi\left(s\right) \approx 1 - 0.165e^{-0.0455s} - 0.335e^{-0.3s} \tag{5.3}$$

the solution of which is shown in Figure 5.2, along with the solution to the approximate Kussner function described in Section 5.4.

5.1.3 Comparison

To assist verification of the developed panel code, lift solutions for thin symmetric airfoils computed using the panel code are compared the Wagner lift solution for a flat plate. Panel code solutions were obtained for instantaneously accelerated NACA 0006, 00010, and 0014 airfoils oriented at $\alpha_0 = 1$, 2, and 4 deg relative to a uniform freestream in the x_1 direction. Solutions were computed using non-dimensionalized time steps of 0.005, 0.075, and 0.010, corresponding to 4000, 3000, and 2000 computational iterations, respectively. Calculated lift coefficients for each simulation were normalized by corresponding steady-state lift values, allowing a comparison to the Wagner function, Eq. (5.3).

Note that differing fundamental assumptions between the panel code and the Wagner solution affect direct comparison of the results. For example, the Wagner solution assumes

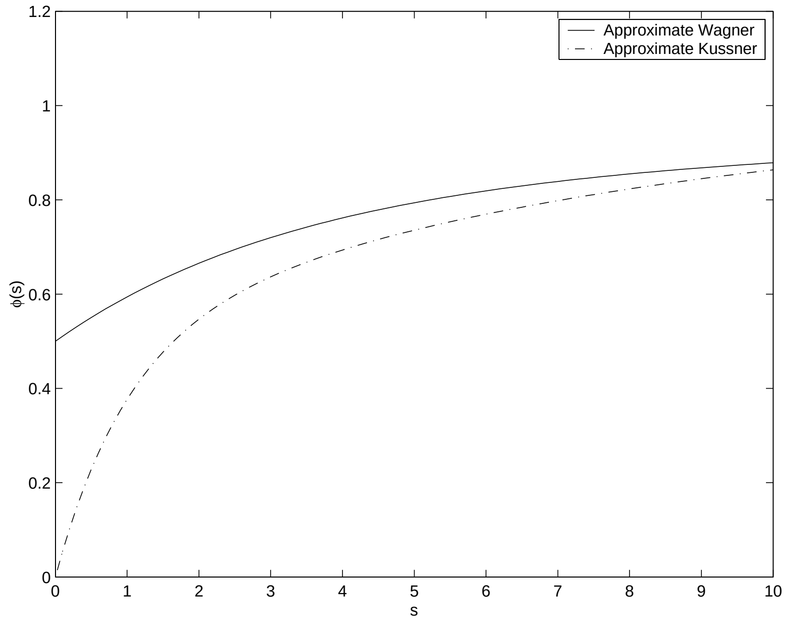

Figure 5.2. Solutions for the approximate Wagner function, Eq. (5.3), and the approximate Kussner function, Eq. (5.19)

the body wake is a continous vortex sheet convecting at the mean freestream velocity, while the panel code discretizes the wake into a set of discrete vortices convecting at the local velocity. Also, the Wagner solution models a flat plate with negligible thickness, while the panel code models a thin symmetric airfoil.

Figure 5.3 compares panel code solutions for airfoils of different thicknesses to the Wagner function. The panel code solutions in Figure 5.3 are computed for NACA 0006, 0010, and 0014 airfoils at $\alpha_0 = 1\,\text{deg}$ using a normalized time step of 0.005. As airfoil thickness decreases, the panel solutions approach the Wagner function.

Figure 5.4 compares panel code solutions for a single airfoil at several orientation angles to the Wagner function. Panel code solutions in Figure 5.4 are computed for a NACA 0010 airfoil at $\alpha_0 = 1$, 2, and 4 deg using a normalized time step of 0.010. As Figure 5.4 shows, airfoil orientation does not have a discernable effect on normalized lift.

Figure 5.5 compares panel code simulations for a single airfoil thickness and orientation but at varying normalized time steps. Panel code solutions in Figure 5.5 are computed for a

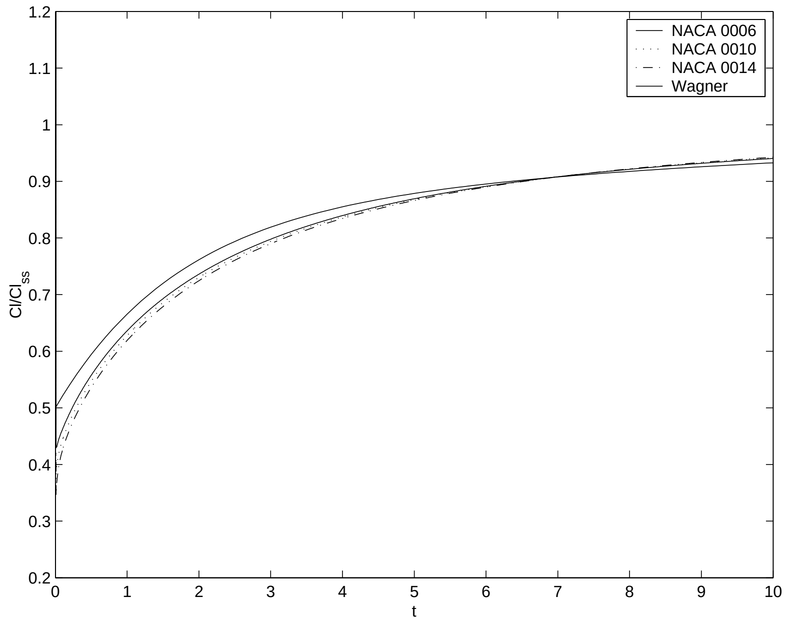

Figure 5.3. Normalized lift for the NACA 0006, 0010, and 0014 airfoils at $\alpha_0 = 1$ deg using a normalized time step of 0.005 compared to Eq. (5.3)

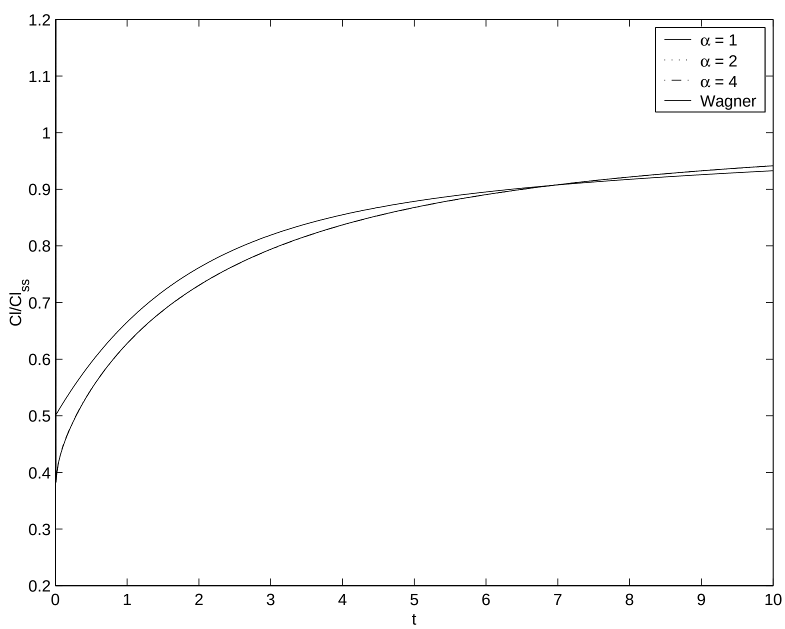

Figure 5.4. Normalized lift on a NACA 0010 at $\alpha_0 = 1$, 2, and 4 deg using a normalized time step of 0.010 compared to Eq. (5.3)

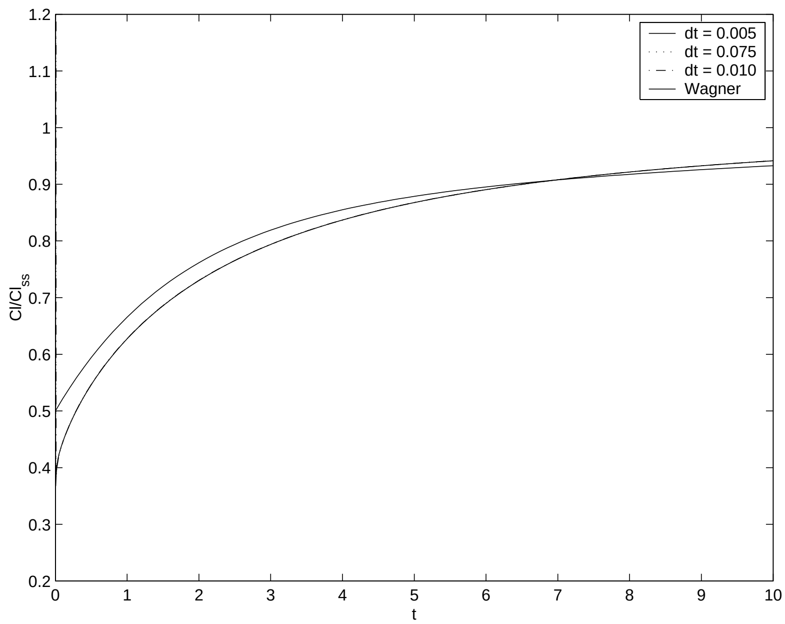

Figure 5.5. Normalized lift on a NACA 0010 at $\alpha_0 = 2\,\text{deg}$ computed using non-dimensionalized time steps of 0.005, 0.075, and 0.010 compared to Eq. (5.3)

NACA 0010 airfoil at $\alpha_0 = 2\,\text{deg}$ using non-dimensionalized time steps of 0.005, 0.075, and 0.010. The panel code solutions show no significant dependence on the selected normalized time steps.

Since neither time step nor orientation significantly affects the panel code solutions, differences between the panel code and Wagner solutions can be attributed primarily to airfoil thickness effects. Despite their differences, however, good overall agreement exists between the two lift solutions.

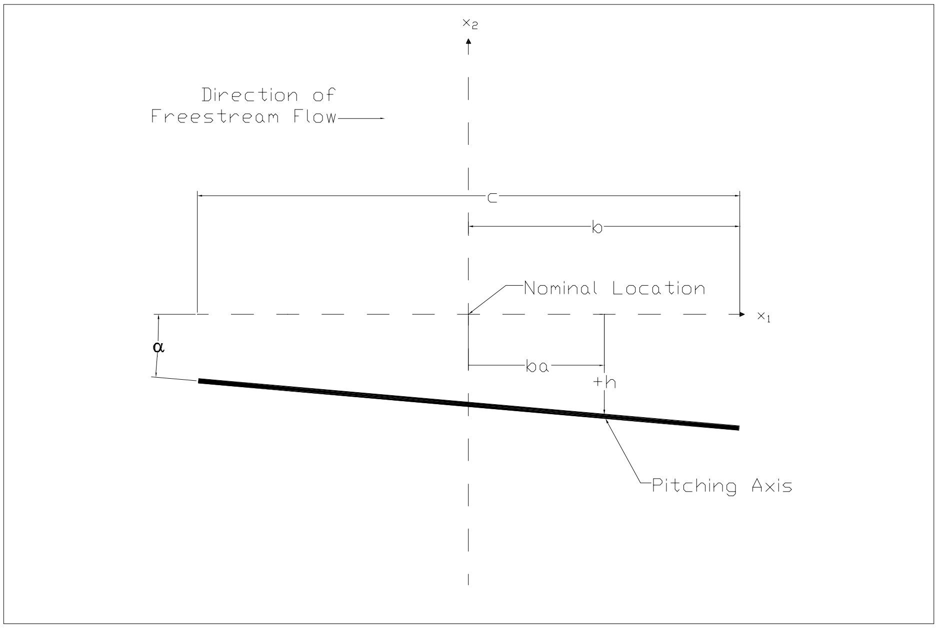

Figure 5.6. Notation used to describe the Theodorsen pitching and plunging flat plate

5.2 Theodorsen

The Theodorsen problem, or the problem of a periodically pitching and plunging airfoil in an otherwise steady uniform freestream flow, demonstrates the effect of body motion and time-dependent wake on unsteady lift and moments.

5.2.1 Description

For an airfoil translating and rotating relative to an otherwise uniform freestream flow, induced flow perturbations near the airfoil surface due to its relative motion can be significant. For an airfoil undergoing periodic translational and rotational relative motion, the influence of the induced surface flow perturbation is a function of motion frequency and amplitude. In addition to motion induced flow perturbations, wake circulation will induce velocity perturbations which also influence the unsteady airfoil lift and moment as a function of motion frequency and amplitude. Figure 5.6 illustrates common notation used to describe the Theodorsen problem.

5.2.2 Solution

Using conformal mapping techniques and a wake model similar to that employed in the Wagner solution, Theodorsen developed an analytic solution for lift and moment on a flat plate undergoing periodic pitching and plunging. The solution relates induced lift and moment on a flat plate to reduced frequency of the relative motion.

$$L = L_{NC} + L_C\, C(k) \tag{5.4}$$

$$M = M_{NC} + M_C\, C(k) \tag{5.5}$$

Note that the Theodorsen lift and moment solutions separate non-circulatory terms, the irrotational component of lift and moment due to body motion,

$$L_{NC} = \pi \rho b^2 \left[\ddot{h} + U\dot{\alpha} - ba\ddot{\alpha}\right] \tag{5.6}$$

$$M_{NC} = \pi \rho b^2 \left[ba\ddot{h} - Ub\left(\frac{1}{2} - a\right)\dot{\alpha} - b^2\left(\frac{1}{8} - a^2\right)\ddot{\alpha}\right] \tag{5.7}$$

from circulatory terms, the rotational component of lift and moment necessitated by the Kutta condition.

$$L_C = 2\pi\rho U b\left[\dot{h} + U\alpha + b\left(\frac{1}{2} - a\right)\dot{\alpha}\right] \tag{5.8}$$

$$M_C = 2\pi\rho U b^2\left(a + \frac{1}{2}\right)\left[\dot{h} + U\alpha + b\left(\frac{1}{2} - a\right)\dot{\alpha}\right] \tag{5.9}$$

The distinction between non-circulatory and circulatory terms is of importance because circulatory terms depend on motion reduced frequency, as related through the Theodorsen function.

$$C(k) = \frac{H_1^2(k)}{H_1^2(k) + iH_0^2(k)} \tag{5.10}$$

Combining Eqs. (5.6) through (5.8) with Eqs. (5.4) and (5.5) produces time-dependent Theodorsen lift and moment equations for a flat plate undergoing periodic pitching and plunging relative to the freestream flow.

$$L = \pi \rho b^2 \left[\ddot{h} + U\dot{\alpha} - ba\ddot{\alpha} \right] + 2\pi\rho U b \left[\dot{h} + U\alpha + b\left(\frac{1}{2} - a\right)\dot{\alpha} \right] C(k) \tag{5.11}$$

$$\begin{aligned} M_y =& \pi\rho b^2 \left[ba\ddot{h} - Ub\left(\frac{1}{2} - a\right)\dot{\alpha} - b^2\left(\frac{1}{8} - a^2\right)\ddot{\alpha} \right] + \\ & 2\pi\rho U b^2 \left(a + \frac{1}{2}\right)\left[\dot{h} + U\alpha + b\left(\frac{1}{2} - a\right)\dot{\alpha} \right] C(k) \end{aligned} \tag{5.12}$$

5.2.3 Comparison

To further assist verification of the unsteady panel code, namely the effects of relative body motion, computed solutions for thin symetric airfoils undergoing periodic pitching, periodic plunging, and periodic pitching and plunging are compared to the corresponding Theodorsen solution for a flat plate. As with comparisons to the Wagner function, the effects of thickness and wake model limit direct comparison between the panel code and analytic solutions.

5.2.3.1 Pure Pitching

Panel code solutions for NACA 0006, 0010, and 0014 airfoils pitching relative to the freestream flow were computed for reduced frequencies of $k = 0.25$ and 0.75 and amplitudes of $\bar{\alpha} = 1$, 2, and 4 deg about the airfoil quarter-chord location. Time-dependent lift and moment for a flat plate undergoing similar motion were also computed using Eqs. (5.11) and (5.12).

Figures 5.7 through 5.9 demonstrate the effect of airfoil thickness on panel code lift and moment solutions, as compared to the Theodorsen solution. Panel code solutions in Figures 5.7 through 5.9 were computed for NACA 0006, 0010, and 0014 airfoils pitching at a reduced frequency of $k = 0.25$ and an amplitude of $\bar{\alpha} = 2\,\text{deg}$.

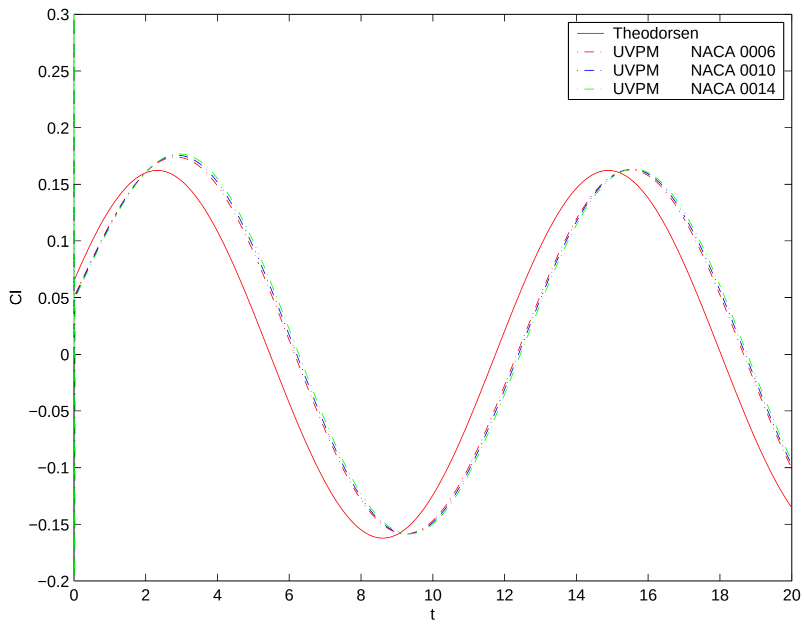

Figure 5.7. C_l vs. Time for NACA 0006, 0010, and 0014 airfoils pitching about the quarter-chord at a reduced frequency of $k = 0.25$ and amplitude of $\bar{\alpha} = 2\,\text{deg}$

For pure pitching at small reduced frequencies, airfoil thickness exhibits a small influence on the phase between the panel code and Theodorsen lift solutions as well as the amplitude ratio of the two solutions. However, both amplitude and phase of the panel code solution approach the Theodorsen solution as airfoil thickness decreases.

Figures 5.10 through 5.12 shows the relative agreement between the panel code lift and moment to the Theodorsen solution, for a range of pitching amplitudes. Panel code solutions for Figures 5.10 through 5.12 were computed for a NACA 0010 airfoil pitching at a reduced frequency of $k = 0.25$, and pitching amplitudes of $\bar{\alpha} = 1$, 2, and 4 deg. For pure pitching at a constant reduced frequency, pitching amplitude does not appear to exhibit a significant influence on either the phase between the panel code and Theodorsen lift solutions or the lift ratio between the two solutions. The lift ratio, computed as the maximum panel code lift coefficent divided by the maximum Theodorsen lift coefficient, remains constant around 1.08 for the pitching amplitudes computed.

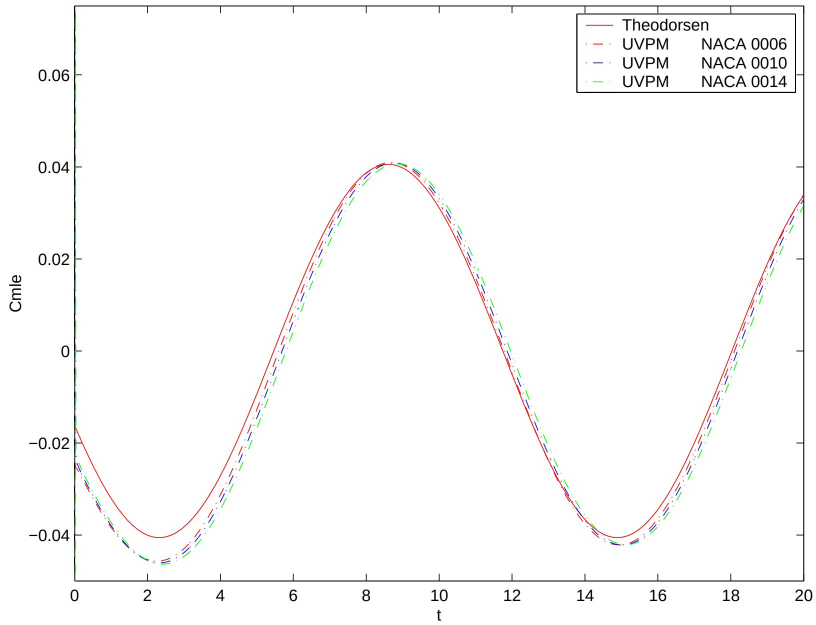

Figure 5.8. $C_{m_{le}}$ vs. Time for NACA 0006, 0010, and 0014 airfoils pitching about the quarter-chord at a reduced frequency of $k = 0.25$ and amplitude of $\bar{\alpha} = 2\,\text{deg}$

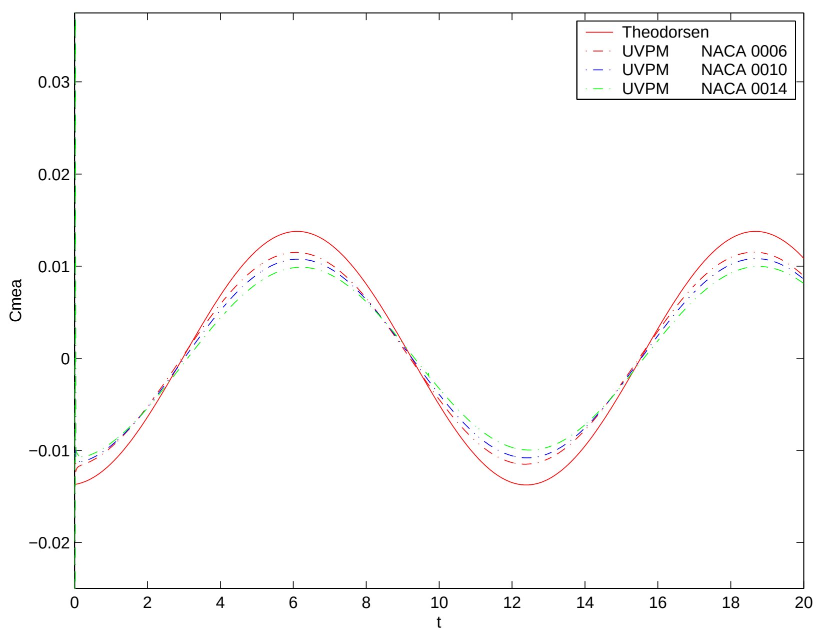

Figure 5.9. $C_{m_{ea}}$ vs. Time for NACA 0006, 0010, and 0014 airfoils pitching about the quarter-chord at a reduced frequency of $k = 0.25$ and amplitude of $\bar{\alpha} = 2\,\text{deg}$

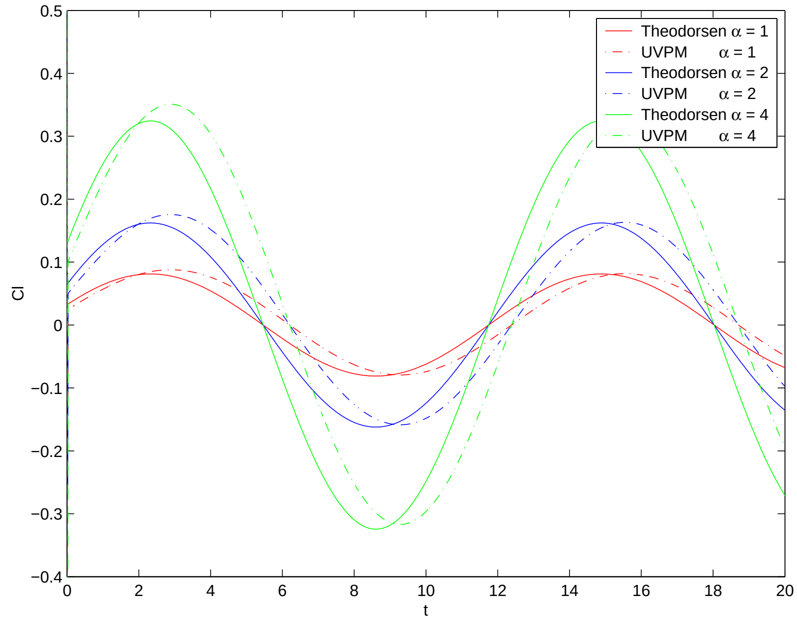

Figure 5.10. C_l vs. Time for a NACA 0010 airfoil pitching about the quarter-chord at a reduced frequency of $k = 0.25$ and amplitudes of $\bar{\alpha} = 1$, 2, and 4 deg

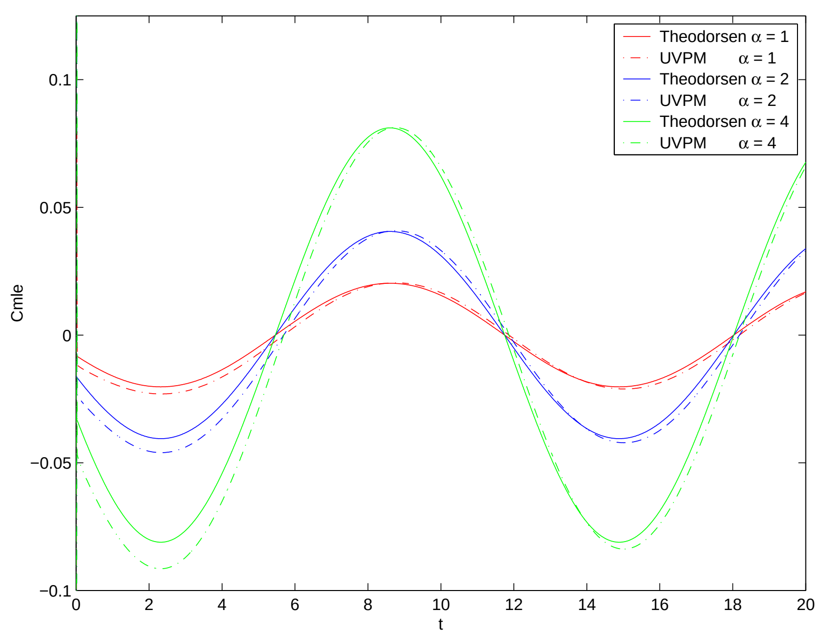

Figure 5.11. $C_{m_{le}}$ vs. Time for a NACA 0010 airfoil pitching about the quarter-chord at a reduced frequency of $k = 0.25$ and amplitudes of $\bar{\alpha} = 1$, 2, and 4 deg

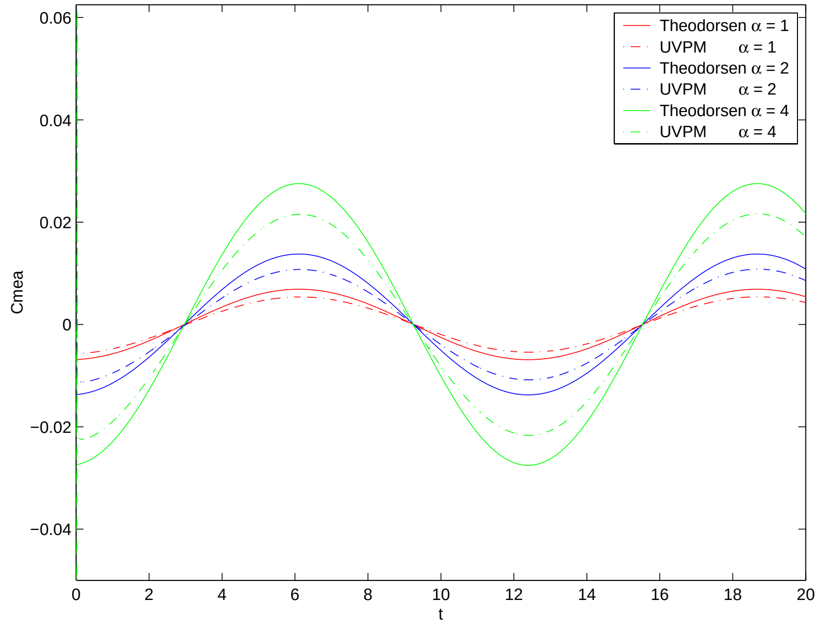

Figure 5.12. $C_{m_{ea}}$ vs. Time for a NACA 0010 airfoil pitching about the quarter-chord at a reduced frequency of $k = 0.25$ and amplitudes of $\bar{\alpha} = 1$, 2, and 4 deg

Figures 5.13 through 5.15 show the relative agreement of the panel code lift and moment solutions to the Theodorsen solution, for a range of reduced frequencies. Panel code solutions in Figures 5.13 through 5.15 are a NACA 0010 airfoil pitching at reduced frequencies of $k =$ 0.25 and 0.75 with a pitching amplitude of $\bar{\alpha} = 2\,\text{deg}$. For pure pitching at a constant amplitude, reduced frequency does not appear to exhibit an influence on the phase between the panel code and Theodorsen lift solutions, but does appear to influence the amplitude ratio between the two solutions. It appears that the pase between the panel code and Theodorsen lift solutions remains constant as reduced frequency varies, however, the amplitude ratio between the panel code and Theodorsen lift solution decreases as reduced frequency increases.

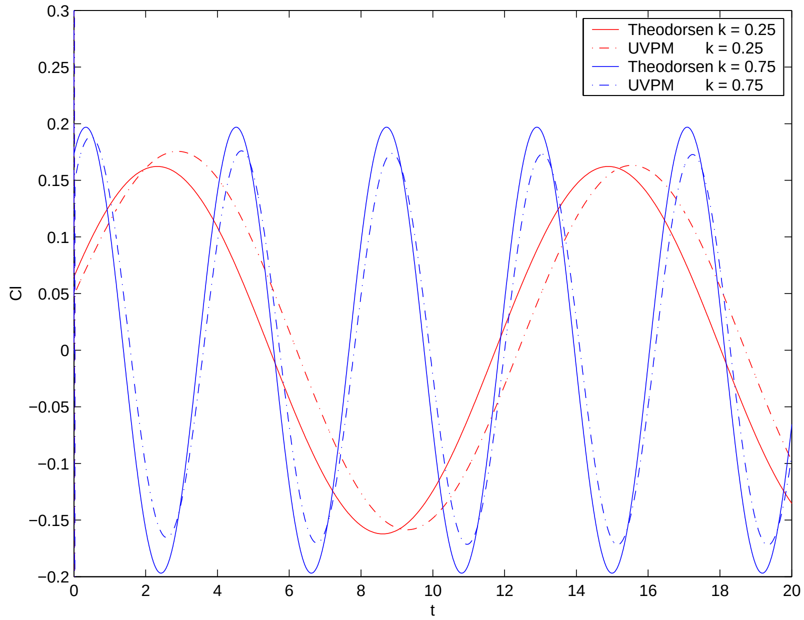

Figure 5.13. C_l vs. Time for a NACA 0010 airfoil pitching at reduced frequencies of $k = 0.25$ and 0.75 with an amplitude of $\bar{\alpha} = 2\,\text{deg}$

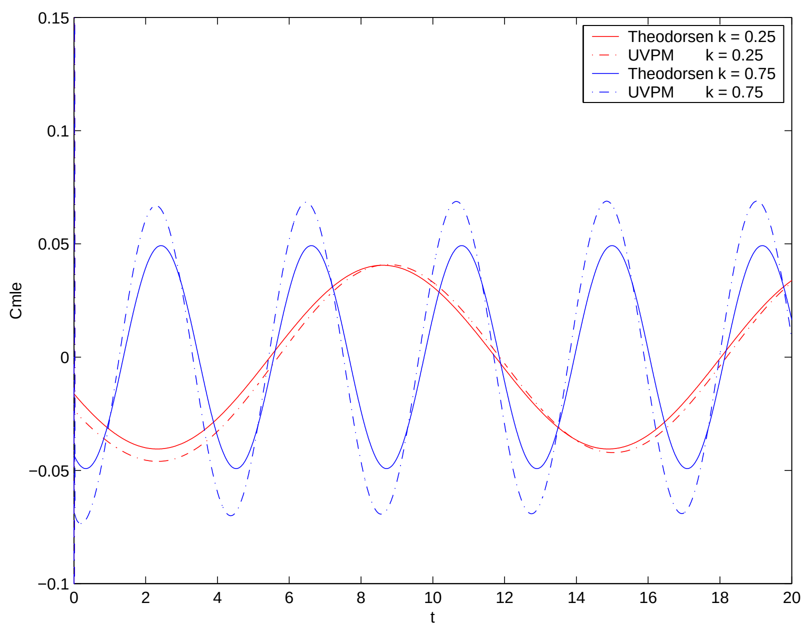

Figure 5.14. $C_{m_{le}}$ vs. Time for a NACA 0010 airfoil pitching about the quarter-chord at reduced frequencies of $k = 0.25$ and 0.75 with an amplitude of $\bar{\alpha} = 2\,\text{deg}$

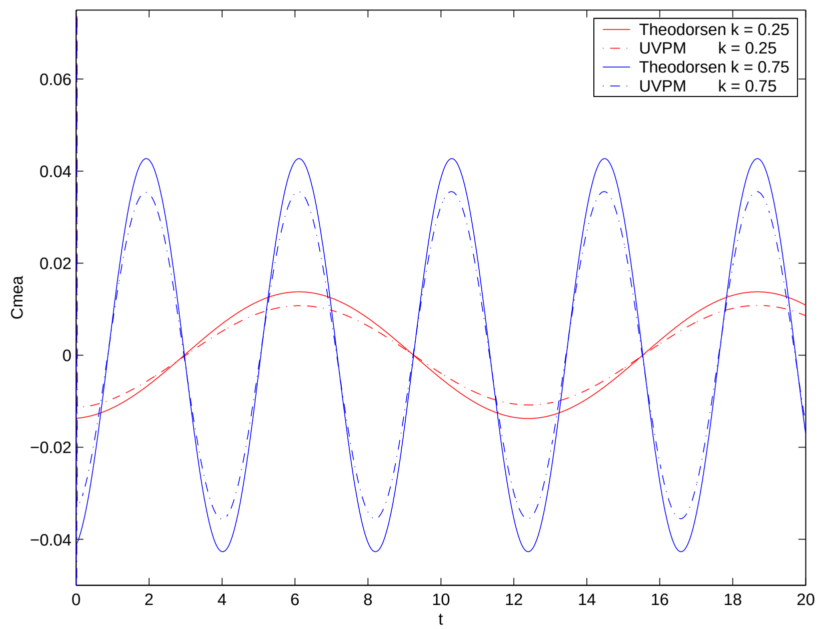

Figure 5.15. $C_{m_{ea}}$ vs. Time for a NACA 0010 airfoil pitching about the quarter-chord at reduced frequencies of $k = 0.25$ and 0.75 with an amplitude of $\bar{\alpha} = 2\,\mathrm{deg}$

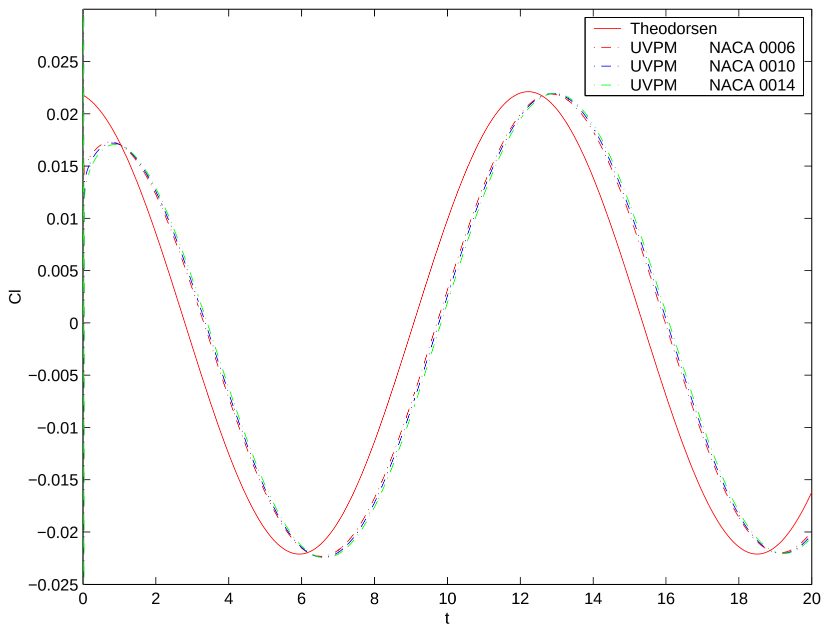

Figure 5.16. C_l vs. Time for NACA 0006, 0010, and 0014 airfoils plunging at a reduced frequency of $k = 0.25$ and an amplitude of $\bar{h} = 0.025$

5.2.3.2 Pure Plunging

The unsteady panel code was used to generate solutions for NACA 0006, 0010, and 0014 airfoils plunging at reduced frequencies of $k = 0.25$ and 0.75 with plunging amplitudes of $\bar{h} = 0.010$, 0.025, and 0.050 relative to the mean freestream flow. Time-dependent lift and moment for a flat plate undergoing similar motion were also computed using Eqs. (5.11) and (5.12). The half-chord was used to calculate moments about the elastic axis.

Figures 5.16 through 5.18 demonstrate the effect of airfoil thickness on panel code lift and moment solutions, as compared to the Theodorsen solution. Panel code solutions in Figures 5.16 through 5.18 were computed for NACA 0006, 0010, and 0014 airfoils plunging at a reduced frequency of $k = 0.25$ and amplitude of $\bar{h} = 0.025$. For pure plunging at small reduced frequencies, as in the case of pure pitching, airfoil thickness exhibits a small influence on the phase between the panel code and Theodorsen lift solutions as well as the amplitude ratio of the two solutions. However, both amplitude and phase of the panel code solution approach the Theodorsen solution as airfoil thickness decreases.

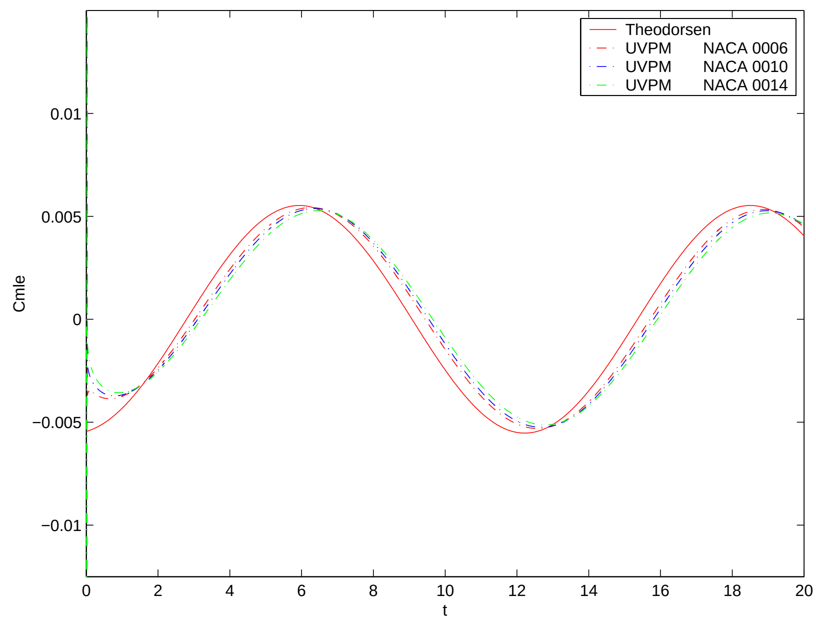

Figure 5.17. $C_{m_{le}}$ vs. Time for NACA 0006, 0010, and 0014 airfoils plunging at a reduced frequency of $k = 0.25$ and an amplitude of $\bar{h} = 0.025$

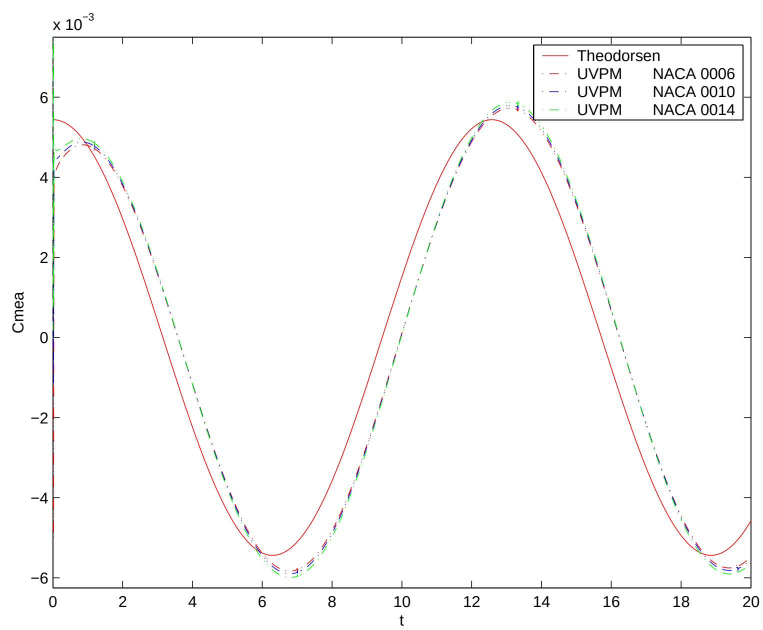

Figure 5.18. $C_{m_{ea}}$ vs. Time for NACA 0006, 0010, and 0014 airfoils plunging at a reduced frequency of $k = 0.25$ and an amplitude of $\bar{h} = 0.025$

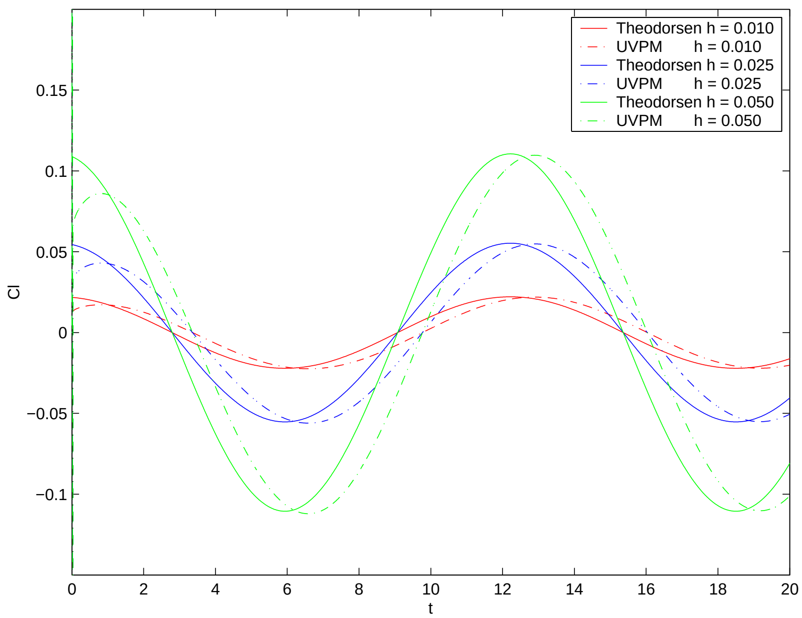

Figure 5.19. C_l vs. Time for a NACA 0010 airfoil plunging at a reduced frequency of $k = 0.25$ and amplitudes of $\bar{h} = 0.010$, 0.025, and 0.050

Figures 5.19 through 5.21 show the relative agreement between the panel code lift and moment solution to the Theodorsen solution for different plunging amplitudes. Panel code solutions in Figures 5.19 through 5.21 were computed for a NACA 0010 airfoil plunging at a reduced frequency of $k = 0.25$ and amplitudes of $\bar{h} = 0.010$, 0.025, and 0.050. As is the case for pure pitching, for pure plunging at a constant reduced frequency, pitching amplitude does not appear to exhibit a significant influence on either the phase between the panel code and Theodorsen lift solutions or the lift ratio between the two solutions. The amplitude ratio remains constant around 0.99 for the plunging amplitudes computed.

Figures 5.22 though 5.24 show relative agreement between the panel code lift and moment solution to the Theodorsens solution at different reduced frequencies. Panel code solutions in Figures 5.22 though 5.24 were computed for a NACA 0010 airfoil plunging at reduced frequencies of $k = 0.25$ and 0.75 and amplitude of $\bar{h} = 0.025$. As is the case for pure pitching, for pure plunging at a constant amplitude, reduced frequency does not appear to exhibit an influence on the phase between the panel code and Theodorsen lift solutions, but does appear

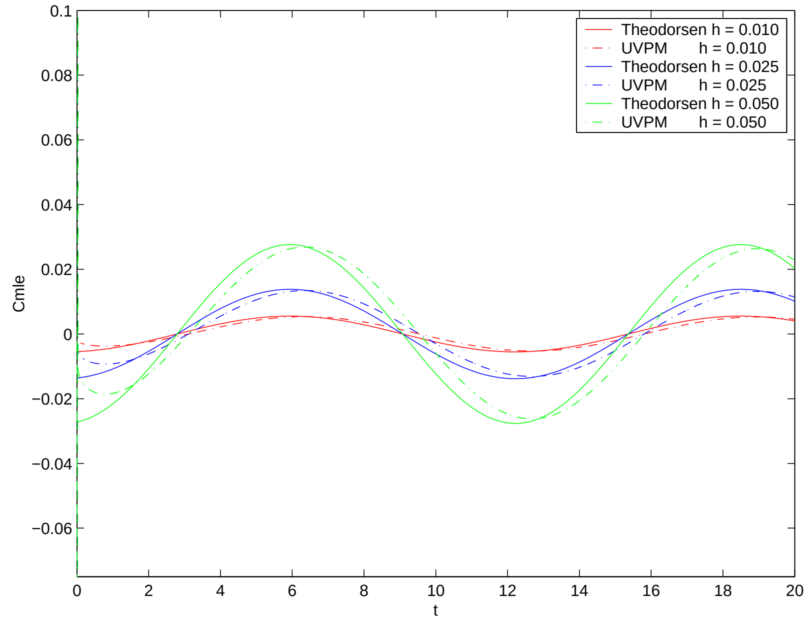

Figure 5.20. $C_{m_{le}}$ vs. Time for a NACA 0010 airfoil plunging at a reduced frequency of $k = 0.25$ and amplitudes of $\bar{h} = 0.010$, 0.025, and 0.050

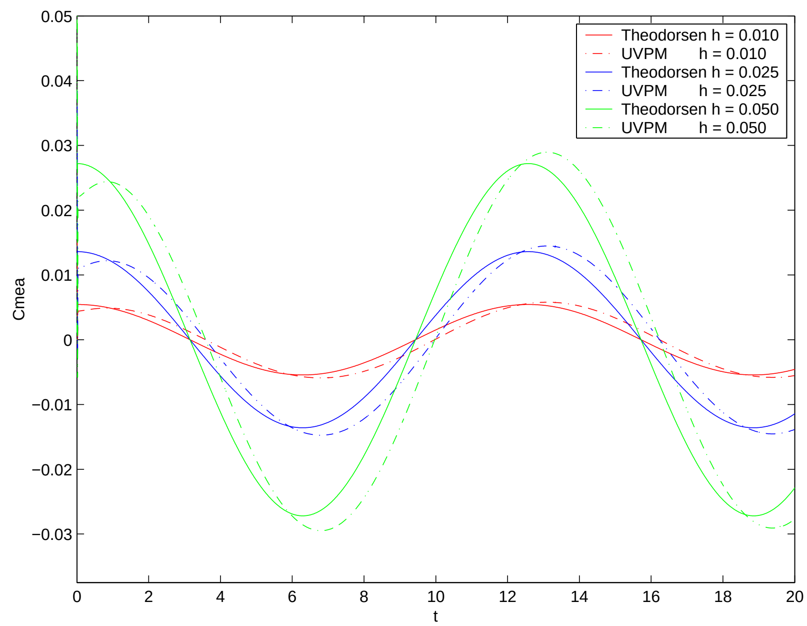

Figure 5.21. $C_{m_{ea}}$ vs. Time for a NACA 0010 airfoil plunging at a reduced frequency of $k = 0.25$ and amplitudes of $\bar{h} = 0.010$, 0.025, and 0.050

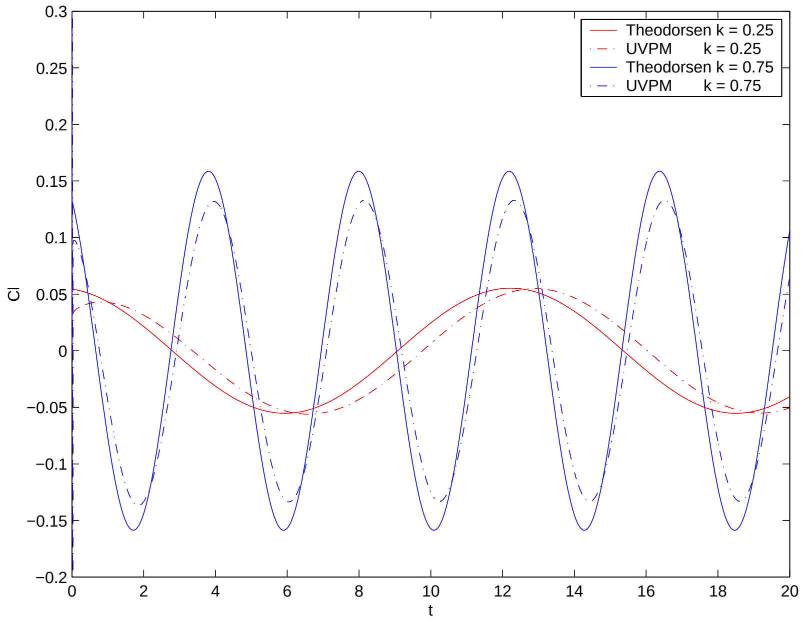

Figure 5.22. C_l vs. Time for a NACA 0010 airfoil plunging at reduced frequencies of $k = 0.25$ and 0.75 and an amplitude of $\bar{h} = 0.025$

to influence the amplitude ratio between the two solutions. It appears that the pase between the panel code and Theodorsen lift solutions remains constant as reduced frequency varies, however, the amplitude ratio between the panel code and Theodorsen lift solution decreases as reduced frequency increases.

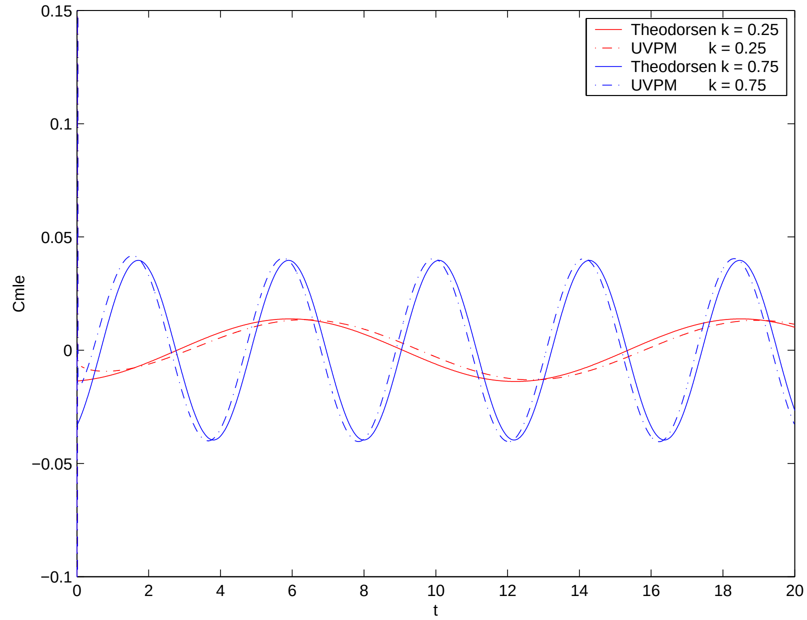

Figure 5.23. $C_{m_{le}}$ vs. Time for a NACA 0010 airfoil plunging at reduced frequencies of $k = 0.25$ and 0.75 and an amplitude of $\bar{h} = 0.025$

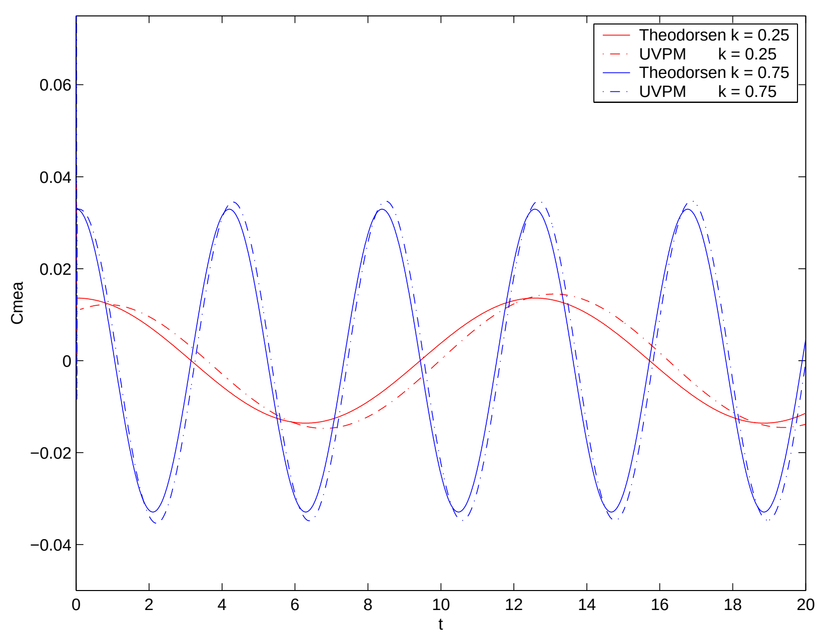

Figure 5.24. $C_{m_{ea}}$ vs. Time for a NACA 0010 airfoil plunging at reduced frequencies of $k = 0.25$ and 0.75 and an amplitude of $\bar{h} = 0.025$

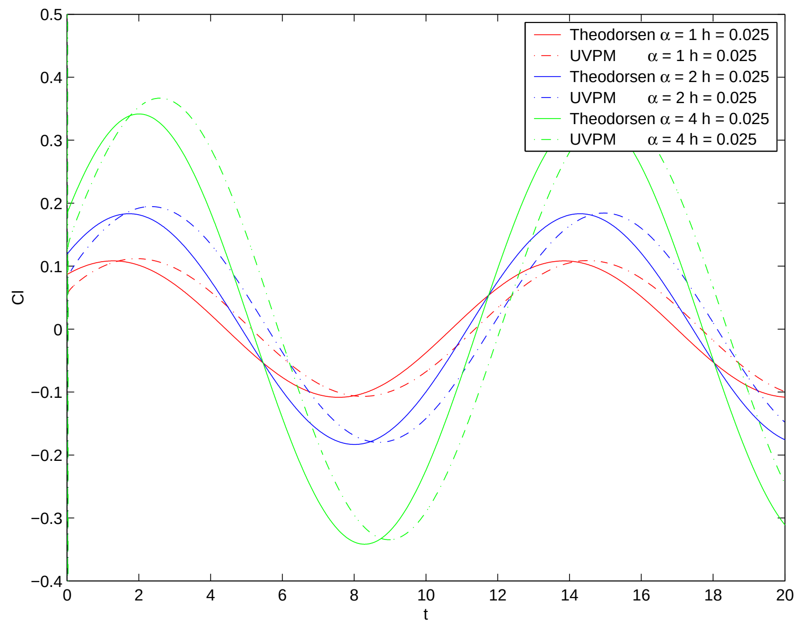

Figure 5.25. C_l vs. Time for a NACA 0010 airfoil pitching and plunging about $x = c/4$ at $k = 0.25$, $\bar{\alpha} = 1$, 2, and 4 deg, and $\bar{h} = 0.025$.

5.2.3.3 Combined Pitching and Plunging

Because solutions for an airfoil undergoing combined pitching and plunging motion represent a superposition of solutions for pure pitching and pure plunging, which have already been examined, this section will use a subset of the previously examined test cases to demonstrate that the panel code properly models combined pitching and plunging. It is expected that the same observations on the effects of amplitude and reduced frequency for the pure pitching and pure plunging cases will hold for the combined pitching and plunging.

Figures 5.25 through 5.27 demonstrate the relative agreement between the panel code lift and moment solutions and the Theodorsen solution. Panel code solutions in Figures 5.25 through 5.27 are for a NACA 0010 airfoil pitching and plunging about the quarter-chord at a reduced frequency of $k = 0.25$ with amplitudes of $\bar{\alpha} = 1$, 2, and 4 deg, and $\bar{h} = 0.025$.

Figures 5.28 through 5.30 demonstrate the relative agreement between the panel code lift and moment solutions and the Theodrsen solution. Panel code solutions in Figures 5.28 through 5.30 are for a NACA 0010 airfoil pitching and plunging about the quarter-chord at

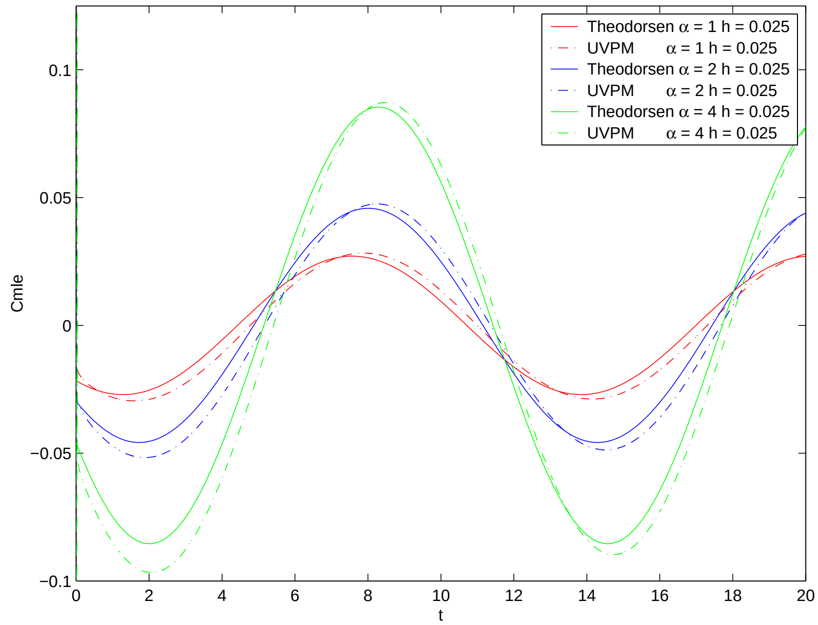

Figure 5.26. $C_{m_{le}}$ vs. Time for a NACA 0010 airfoil pitching and plunging about $x = c/4$ at $k = 0.25$, $\bar{\alpha} = 1$, 2, and 4 deg, and $\bar{h} = 0.025$.

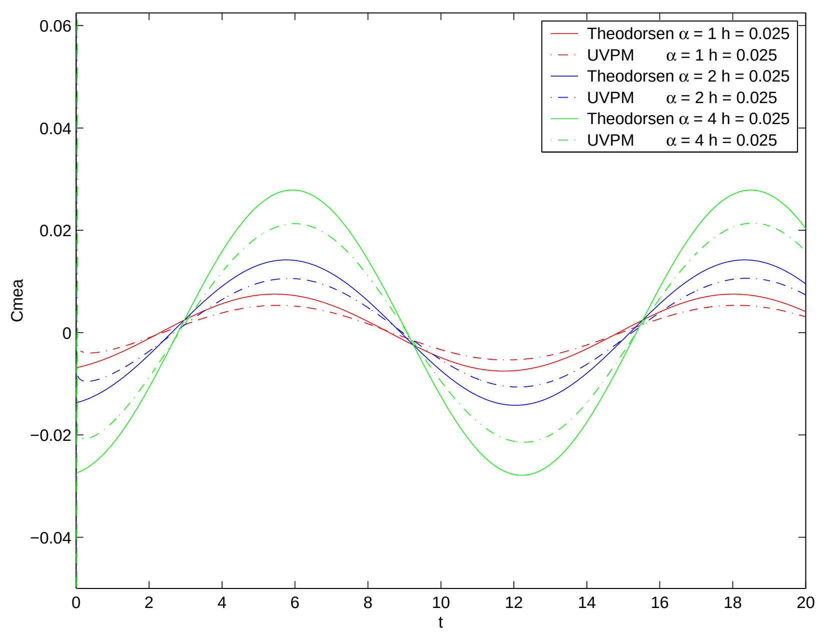

Figure 5.27. $C_{m_{ea}}$ vs. Time for a NACA 0010 airfoil pitching and plunging about $x = c/4$ at $k = 0.25$, $\bar{\alpha} = 1$, 2, and 4 deg, and $\bar{h} = 0.025$.

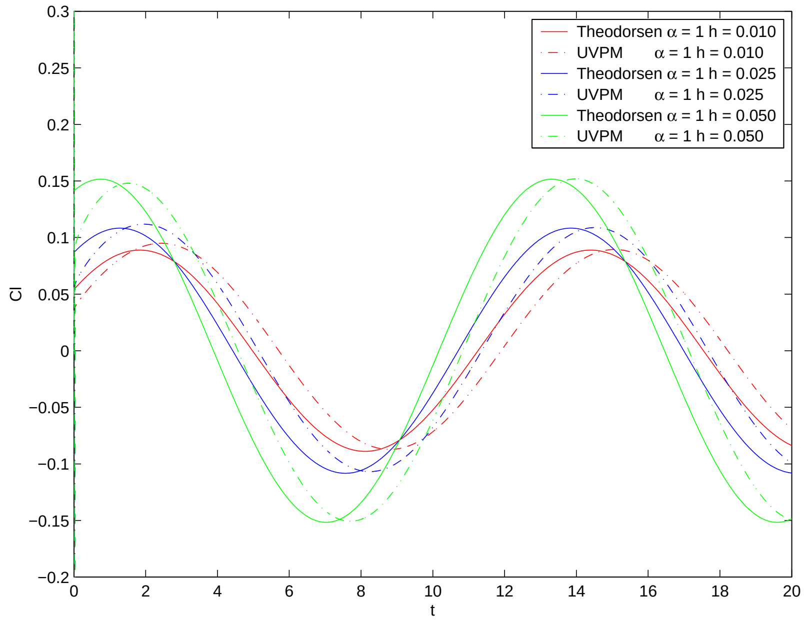

Figure 5.28. C_l vs. Time for a NACA 0010 airfoil pitching and plunging about $x = c/4$ at $k = 0.25$, $\bar{\alpha} = 2$ deg, and $\bar{h} = 0.010$, 0.025, and 0.050.

a reduced frequency of $k = 0.25$ with a amplitudes of $\bar{\alpha} = 1$ deg and $\bar{h} = 0.010$, 0.025, and 0.050.

5.2.3.4 Discussion

As observed in the pure pitching and pure plunging examples, the panel code solution showed small variations in both phase and amplitude as compared to the Theodorsen solution. It was also shown that these small variations were dependent only on airfoil thickness and reduced frequency.

Since the variations do not do not show a dependence on motion amplitude, the variation between the two solutions may be attributed to differences inherent in the wake models. As described earlier, the Theodorsen solution models the shed bound vorticity as a continuous vortex sheet of variable strength, released from the undisturbed airfoil trailing edge. The vortex sheet then convects with the time-averaged freestream flow, essentially confining the vortex sheet to the x_1 axis. The panel code models the shed circulation as a set of discrete vortex elements, released from the airfoil trailing edge location at each time step. The discrete

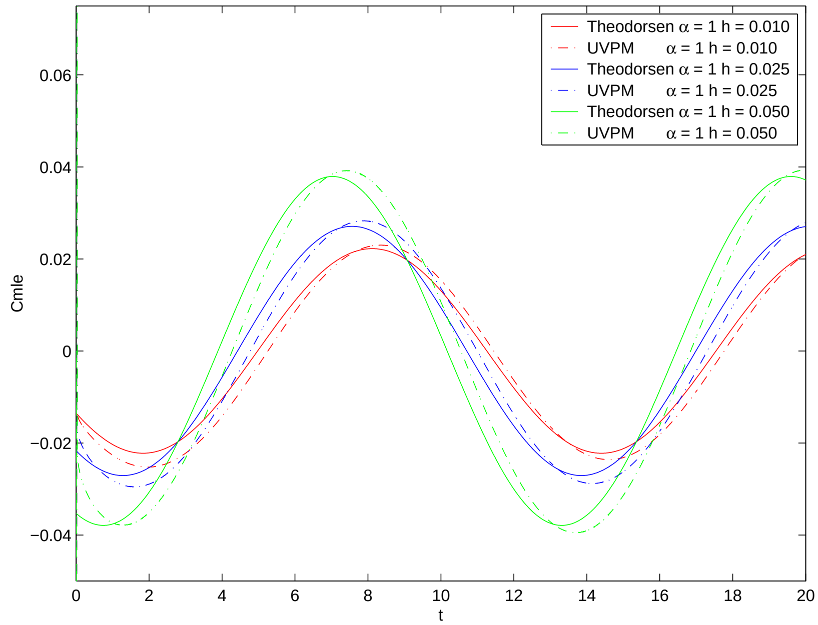

Figure 5.29. $C_{m_{le}}$ vs. Time for a NACA 0010 airfoil pitching and plunging about $x = c/4$ at $k = 0.25$, $\bar{\alpha} = 2$ deg, and $\bar{h} = 0.010$, 0.025, and 0.050.

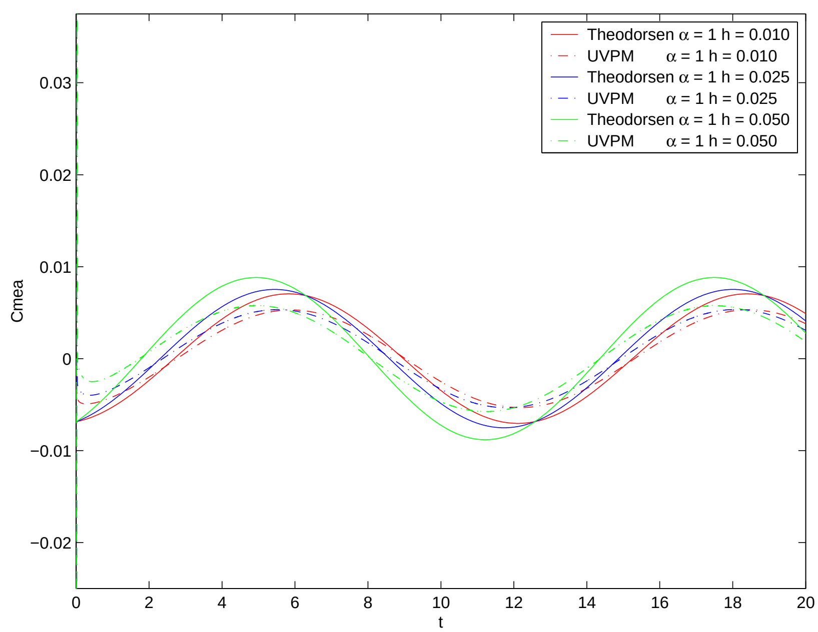

Figure 5.30. $C_{m_{ea}}$ vs. Time for a NACA 0010 airfoil pitching and plunging about $x = c/4$ at $k = 0.25$, $\bar{\alpha} = 2$ deg, and $\bar{h} = 0.010$, 0.025, and 0.050.

vortex elements are allowed to convect with the instantaneous local flowfield, and thus the wake is allowed to deform in time. By convecting the wake at the local velocity and allowing the wake to deform, the panel code wake model induces a different influence on the airfoil than the Theodorsen wake model, which would be dependent on reduced frequency.

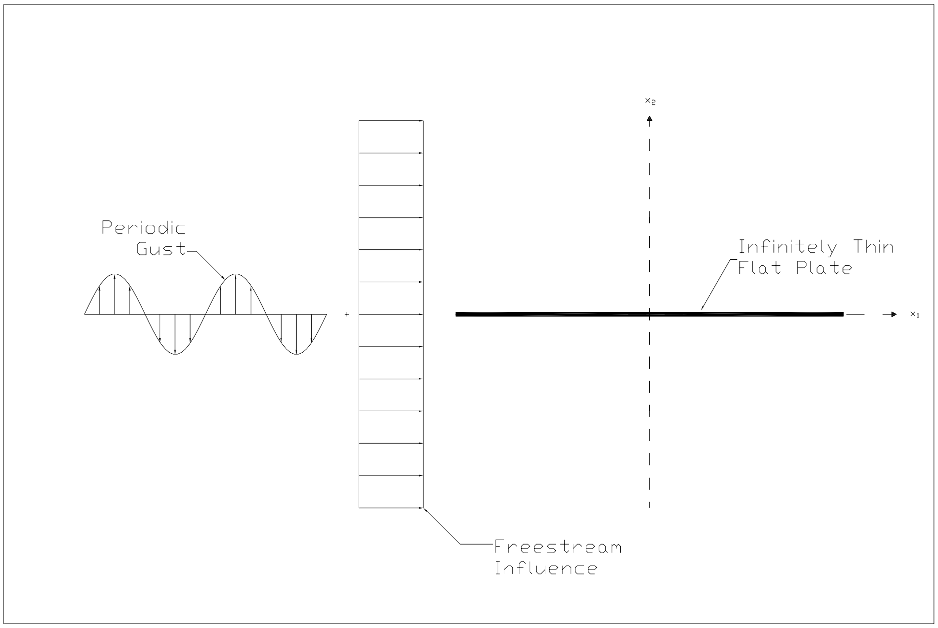

Figure 5.31. Stationary plate of infinitesimal thickness with periodic transverse gust

5.3 Sears Periodic Gust

The Sears periodic gust problem examines time-dependent lift and moment generated on a stationary airfoil under the influence of a time-averaged uniform freestream flow with sinusoidal transverse velocity perturbations, or transverse gusts.

5.3.1 Description

Consider a stationary airfoil immersed in a freestream flow where the time-averaged flow is aligned with the airfoil chord. If the airfoil is symmetric, it will not generate lift. However, if a transverse velocity perturbation is introduced into the freestream, the velocity perturbation will induce a local time-dependent angle of attack change on the airfoil that will induce unsteady lift. The Sears gust problem investigates the influence of periodic transverse velocity perturbations on unsteady lift and moment generated on a flat plate.

5.3.2 Solution

By assuming the transverse gust is not influenced by the presence of the airfoil, i.e. the "frozen gust" approximation, the downwash induced by the gust on the airfoil as a function of time can be writen as

$$w = \bar{w}e^{i\omega\left(t-\frac{x}{U}\right)} = \bar{w}e^{ik(s-x^*)} \tag{5.13}$$

Using a wake model similar to that of Wagner and Theodorsen, the relation for wake induced downwash on the airfoil as a function of reduced gust frequency is given by the Sears function.

$$S(k) = \frac{2}{\pi k\left[H_0^2(k) - iH_1^2(k)\right]} \tag{5.14}$$

Modifying the no-flow boundary condition to include both gust and wake induced downwash, Sears lift and moment solutions for a flat plate under the influence of a sinusoidal transverse gust become

$$L = 2\pi\rho U\bar{w}be^{i\omega t}\,S(k) \tag{5.15}$$

$$M_y = L\left(\frac{1}{2}+a\right)b \tag{5.16}$$

5.3.3 Comparison

The panel code can implement two separate methods to model a periodic transverse gust. The first method, refered to later as the modified panel code, accounts for the influence of the periodic gust directly by modifying the implementation of the no-flow boundary condition on the airfoil surface. The modified boundary condition includes the influence of the velocity pertubation by replacing the constant freestream velocity term with a time and position dependent function. This time and position dependent function must then be included in every other calculation which depends on the freestream velocity, such as the computation of unsteady surface pressures and the convection routines used to convect wake elements with the local flowfield. The application of this method to other problems, such as forced responce, is limited because gust influence on the airfoil is directly modeled as a function of time and location in the flowfield, and as such does not allow for gust deformation due to body or wake influences.

The second method employs the gust model descibed in Section 4.2. To use this discrete gust model, the continuous periodic gust is discretized into a set of gust sheets which propagate across the airfoil at the local flow velocity. This method does not require modifications to the original no-flow boundary condition since the gust sheet is composed of constant strength vortex elements whose influence was included in the original no-flow boundary condition. Since the gust sheets convect with the local flowfield, this method allows for gust deformation due to body and wake influences. The comparison of this method to the Sears solution is only limited by the discretization of the continuous periodic gust into a corresponding gust sheet representation. As such, care must be taken in choosing the method of discretization, since different representations of the same continuous gust will result in different lift and moment solutions.

5.3.3.1 Modified No-Flow Boundary Condition

Figures 5.32 through 5.37 demonstrate the effect of airfoil thickness on panel code lift and moment solutions as compared to the Sears lift and moment solutions for reduced frequencies of $k = 0.25$, 1.0, and 4.0. Panel code solutions in Figures 5.32 through 5.37 were computed for NACA 0006, 0010, 0012, and 00014 airfoils under the influence of a continuous sinusoidal gust having an amplitude of $\bar{w} = 0.01$ using the modified no-flow boundary condition.

Airfoil thickness exhibits a small influence on the phase between the modified panel code and Sears lift solutions as well as the amplitude ratio of the two solutions. The difference in amplitude of the lift solution computed by the panel code is attributed to the effects of airfoil thickness, because the solutions approach, but do not reach the Sears solution as airfoil thickness decreases. It is interesting to note that the amplitude ratio, computed as the maximum panel code lift coefficent divided by the maximum Sears lift coefficent, remains constant at roughly 0.7 for a NACA 0010 airfoil regardless of reduced frequency. The

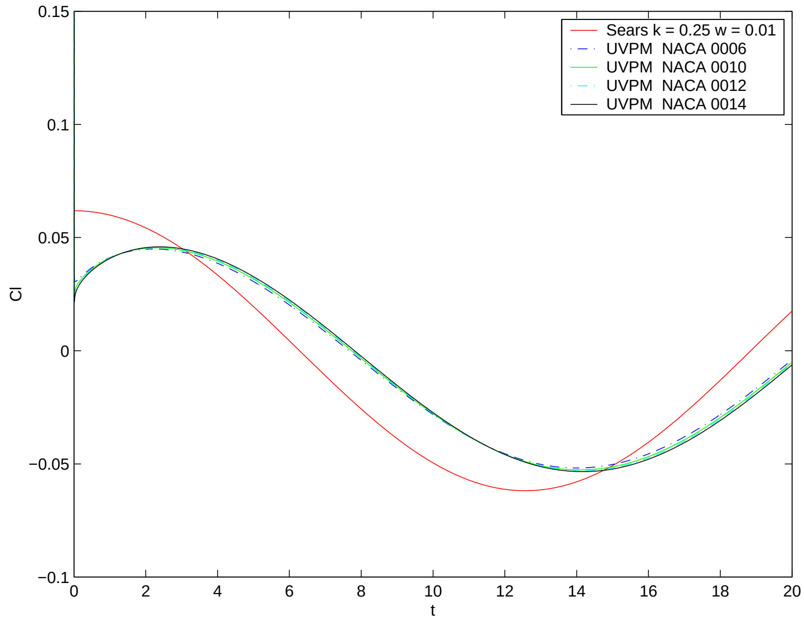

Figure 5.32. C_l vs. Time for the Sears solution compared to the alternate panel code solution for NACA 0006, 0010, 0012, and 0014 airfoils under the influence of a sinusoidal gust with a reduced frequency of $k = 0.25$ and a gust amplitude of $\bar{w} = 0.01$

influence of reduced frequency on the phase of the solutions is more dramatic. The phase of the modified panel code solution lags the Sears solution at small reduced frequencies, and shifts such that it leads at higher reduced frequencies. Because the amplitude ratio does not appear to be influenced by reduced frequency, it is assumed that differences in wake models, as discussed in Section 5.2.3.4, are responsible for the phase shift with reduced frequency.

5.3.3.2 Vortex Sheet Gust Model

Figures 5.38 and 5.39 compares lift and moment coefficents calculated by the panel code utilizing the freestream gust model to a corresponding Sears solution. Panel code solutios in Figures 5.38 and 5.39 were computed for a NACA 0010 airfoil oriented at $\alpha_0 = 0.0\,\text{deg}$ to the time-averaged freestream. The gust, having a reduced frequency of $k = 1.0$ and amplitude of $\bar{w} = 0.01$, was modeled using a set of six gust sheets per gust period for five and a half periods upstream of the airfoil. The strength of each gust sheet is based on the velocity pertubation at the gust sheet's initial x_1 location in the flowfield. Figure 5.40 shows

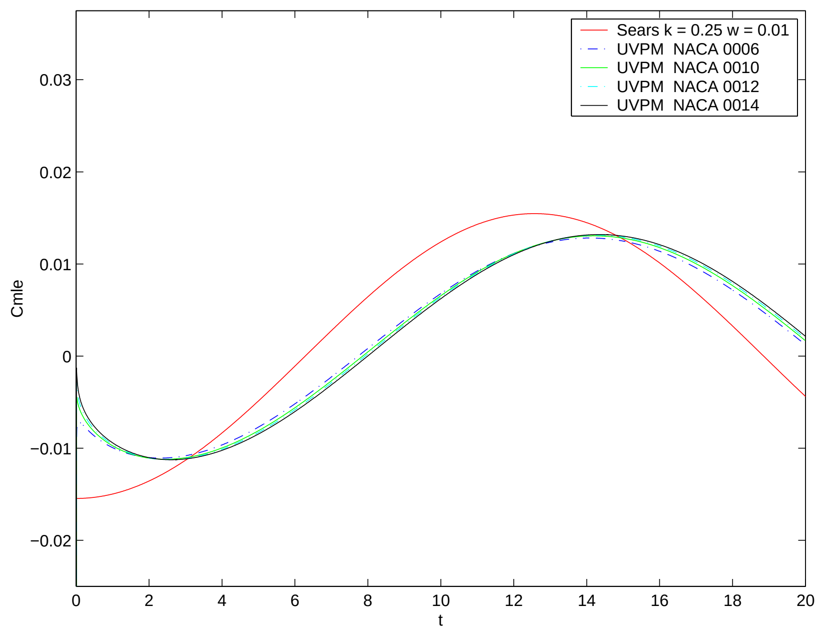

Figure 5.33. $C_{m_{le}}$ vs. Time for the Sears solution compared to the alternate panel code solution for NACA 0006, 0010, 0012, and 0014 airfoils under the influence of a sinusoidal gust with a reduced frequency of $k = 0.25$ and a gust amplitude of $\bar{w} = 0.01$

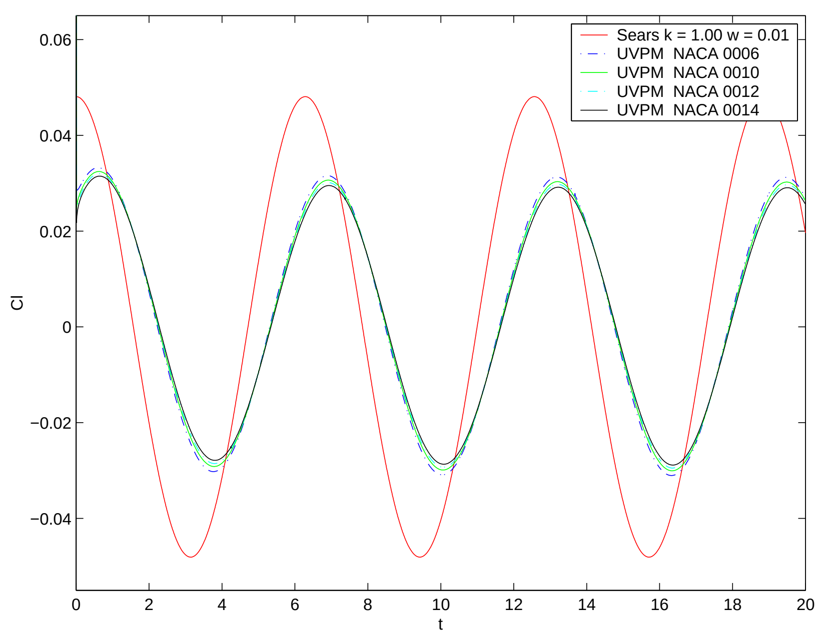

Figure 5.34. C_l vs. Time for the Sears solution compared to the alternate panel code solution for NACA 0006, 0010, 0012, and 0014 airfoils under the influence of a sinusoidal gust with a reduced frequency of $k = 1.0$ and a gust amplitude of $\bar{w} = 0.01$

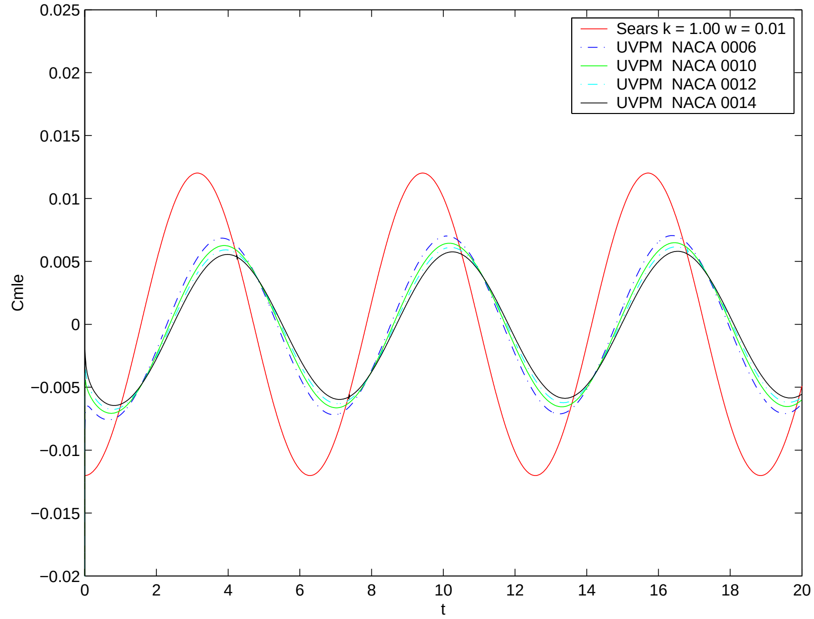

Figure 5.35. $C_{m_{le}}$ vs. Time for the Sears solution compared to the alternate panel code solution for NACA 0006, 0010, 0012, and 0014 airfoils under the influence of a sinusoidal gust with a reduced frequency of $k = 1.0$ and a gust amplitude of $\bar{w} = 0.01$

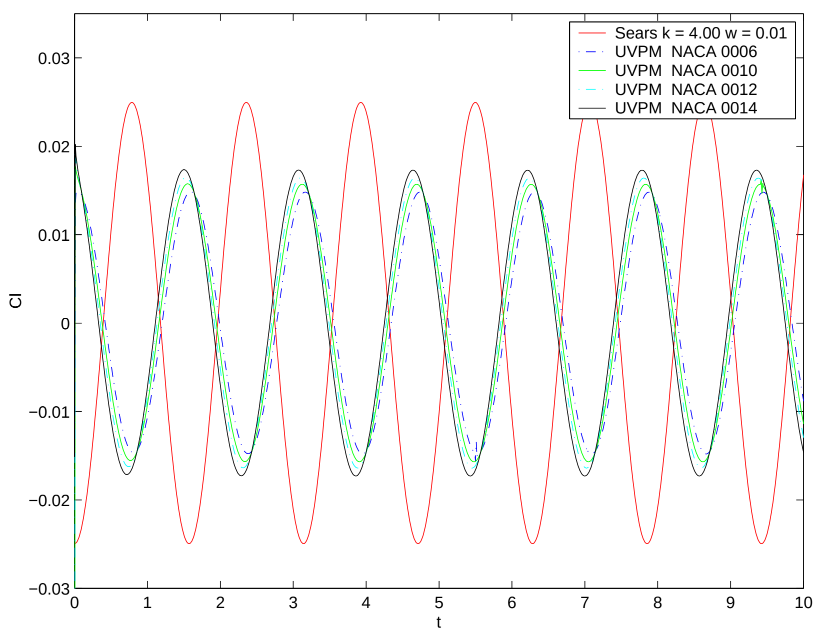

Figure 5.36. C_l vs. Time for the Sears solution compared to the alternate panel code solution for NACA 0006, 0010, 0012, and 0014 airfoils under the influence of a sinusoidal gust with a reduced frequency of $k = 4.0$ and a gust amplitude of $\bar{w} = 0.01$

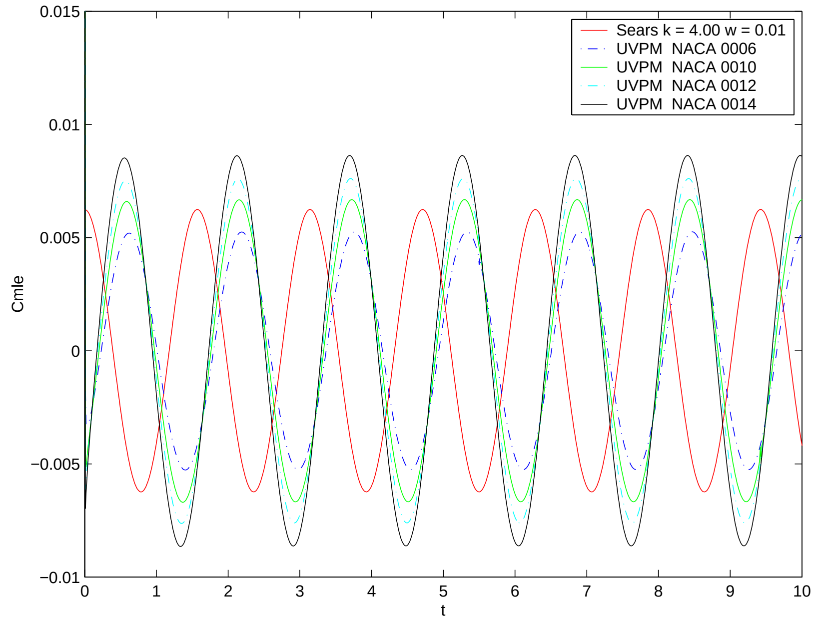

Figure 5.37. $C_{m_{le}}$ vs. Time for the Sears solution compared to the alternate panel code solution for NACA 0006, 0010, 0012, and 0014 airfoils under the influence of a sinusoidal gust with a reduced frequency of $k = 4.0$ and a gust amplitude of $\bar{w} = 0.01$

circulation strength per unit length about each gust sheet relative to the initial x_1 location of the sheet. One drawback to the use of the discrete gust model is that the gust sheets do not produce a sinusoidal velocity pertubation. The induced pertubation due to the gust sheets closely resembles a set of sharp edge gusts, as shown in Figure 5.41. Figure 5.41 combines a visualization of the location of the airfoil, wake, and gust sheets in the top panel, the instantaneous pressure coefficients along the airfoil in the lower left panel, and the coefficent of lift time-history in the lower right panel. Velocity vectors representing the freestream velocity in the x_2 direction, sampled at locations upstream of the airfoil along the x_1 axis and scaled by a factor of 100, have been added to the location plot in the top panel.

The lift coefficent computed by the panel code shown in Figure 5.38 overshoots the lift coefficient predicted by the Sears solution. This overshoot is due to the discretization of the continuous gust. The maximum velocity pertubation, $\bar{w}$, induced by the of the set of gust sheets is closer to 0.02 than 0.01, as shown by the velocity vectors in Figure 5.41. Therefore, a second simulation was computed for using a different discretization of the freestream gust.

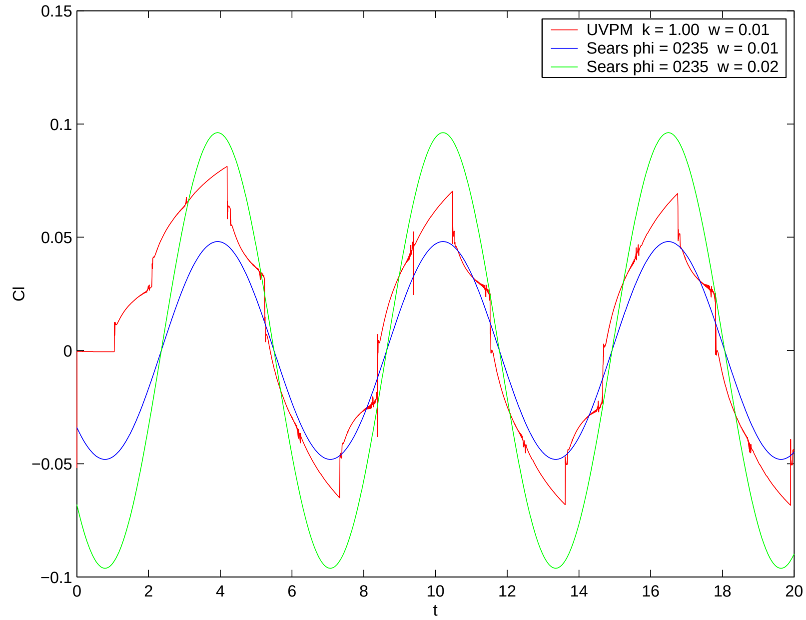

Figure 5.38. C_l vs. Time for the Sears solution compared to the panel code solution for NACA 0010 airfoil under the influence of a periodic freestream gust with a reduced frequency of $k = 1.0$ and gust amplitude of $\bar{w} = 0.01$, sampled at 6 times the reduced frequency.

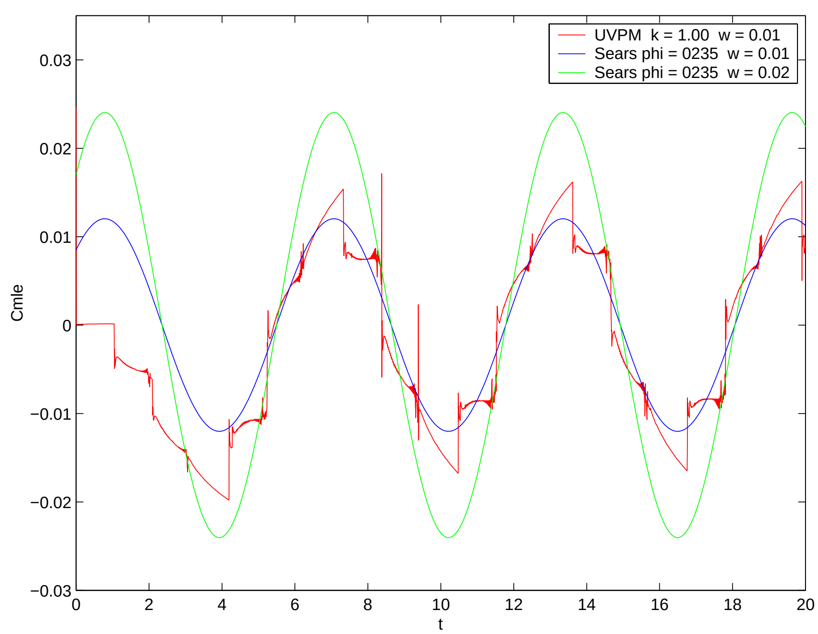

Figure 5.39. $C_{m_{le}}$ vs. Time for the Sears solution compared to the panel code solution for NACA 0010 airfoil under the influence of a periodic freestream gust with a reduced frequency of $k = 1.0$ and gust amplitude of $\bar{w} = 0.01$, sampled at 6 times the reduced frequency.

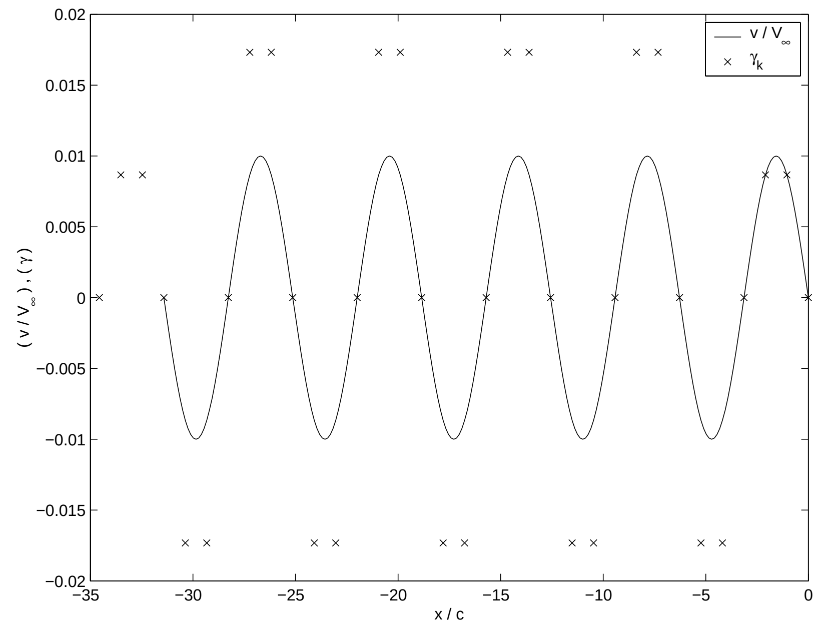

Figure 5.40. Gust sheet circulation per unit length vs. initial x/c location for a periodic freestream gust with a reduced frequency of $k = 1.0$ and gust amplitude of $\bar{w} = 0.01$, sampled at 6 times the reduced frequency.

The alternate gust discretization uses a set of four gusts sheets per gust period for five and a half periods upstream of the airfoil. Again, the panel code solution was computed for a NACA 0010 airfoil oriented at $\alpha_0 = 0.0\,\mathrm{deg}$ to the time-averaged freestream. Figures 5.42 and 5.43 compare the lift and moment coefficents calculated by the panel code utilizing the new freestream gust discretization to the Sears solution. Figure 5.44 shows the circulation strength per unit length about each gust sheet in relation to the initial x_1 location of the sheet, as well as the amplitude of the velocity pertubation induced by the continuous gust sheet. Figure 5.45 shows the same visualization as Figure 5.41, including the velocity vectors representing the freestream velocity in the x_2 direction.

Using four gust sheets per gust period to discretize the continuous gust does not produce the graduated velocity pertubation which is possible by using a larger number of gust sheets per gust period, but the maximum velocity pertubation does match the maximum value of the continuous pertubation. As such, the the lift and moment coefficients closely match the coefficients predicted by the Sears solution.

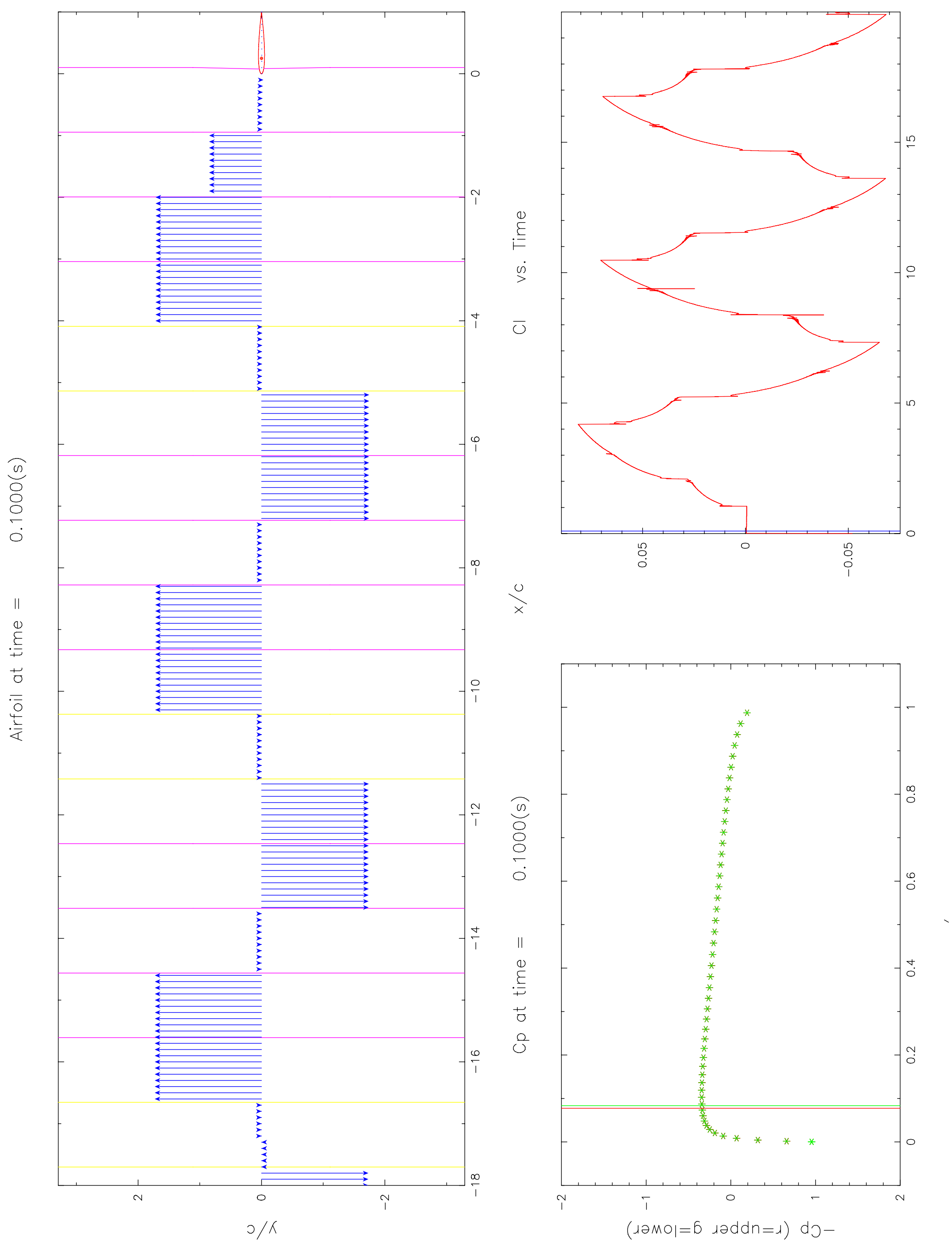

Figure 5.41. Visualization showing the location of the airfoil, wake, gust sheets, and selected x_2 velocities in the top panel, instantaneous C_p vs. x/c in the lower left panel, and C_l vs. t in the lower right panel.

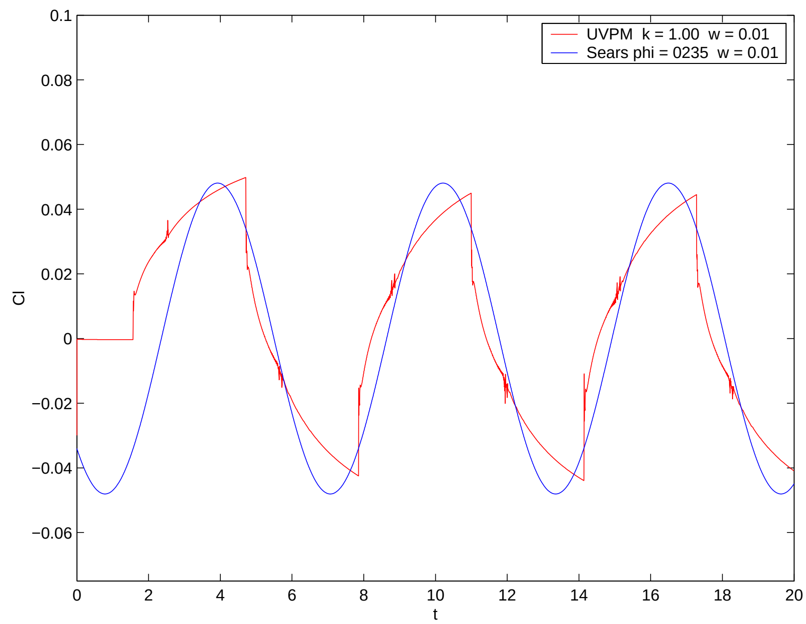

Figure 5.42. C_l vs. Time for the Sears solution compared to the panel code solution for NACA 0010 airfoil under the influence of a periodic freestream gust with a reduced frequency of $k = 1.0$ and gust amplitude of $\bar{w} = 0.01$, sampled at 4 times the reduced frequency.

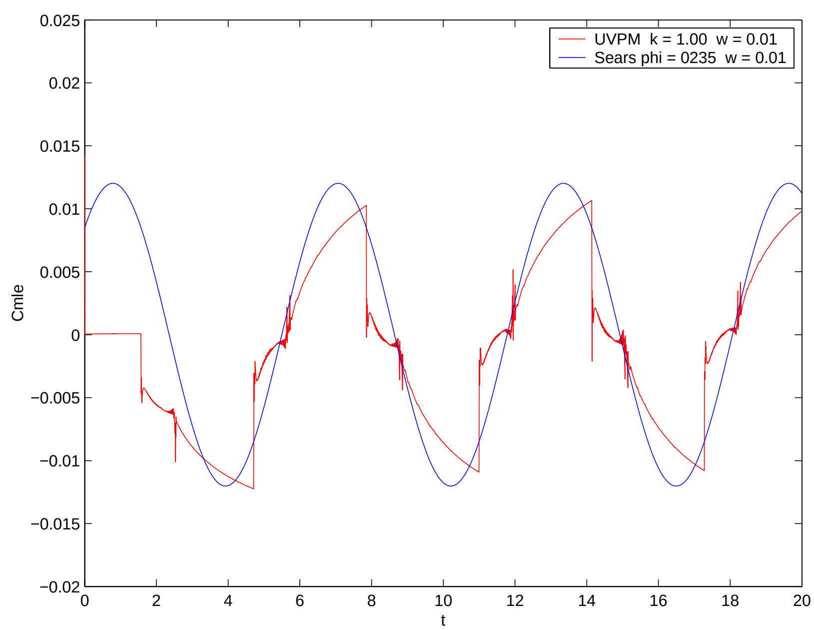

Figure 5.43. $C_{m_{le}}$ vs. Time for the Sears solution compared to the panel code solution for NACA 0010 airfoil under the influence of a periodic freestream gust with a reduced frequency of $k = 1.0$ and gust amplitude of $\bar{w} = 0.01$, sampled at 4 times the reduced frequency.

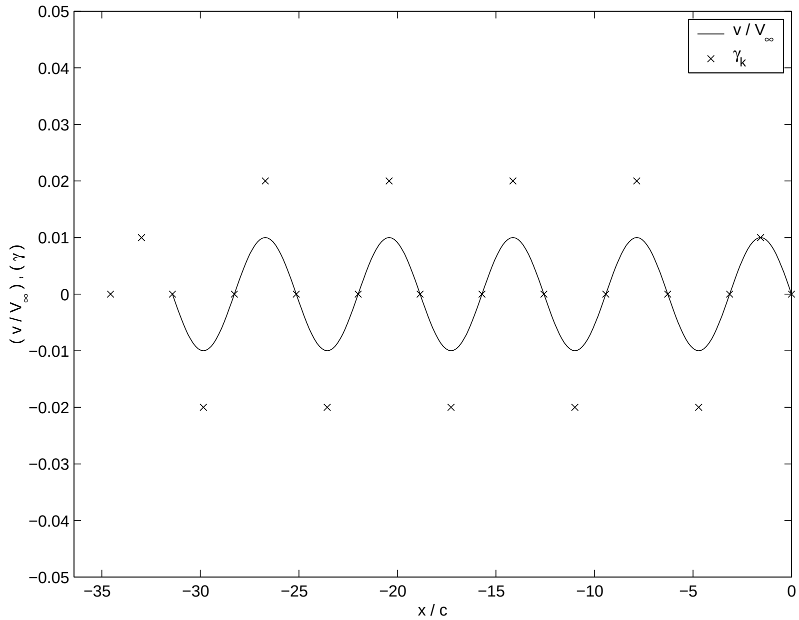

Figure 5.44. Gust sheet circulation per unit length vs. initial x/c location for a periodic freestream gust with a reduced frequency of $k = 1.0$ and gust amplitude of $\bar{w} = 0.01$, sampled at 4 times the reduced frequency.

Alternate methods of discretizing the continuous gust are possible, and a higher order discretization could be used which would provide a closer match to the Sears solution. However, it has been shown that the gust model can be used to model periodic freestream perturbations in a manner which allows the gust to deform due to the influence of the airfoil and airfoil wake.

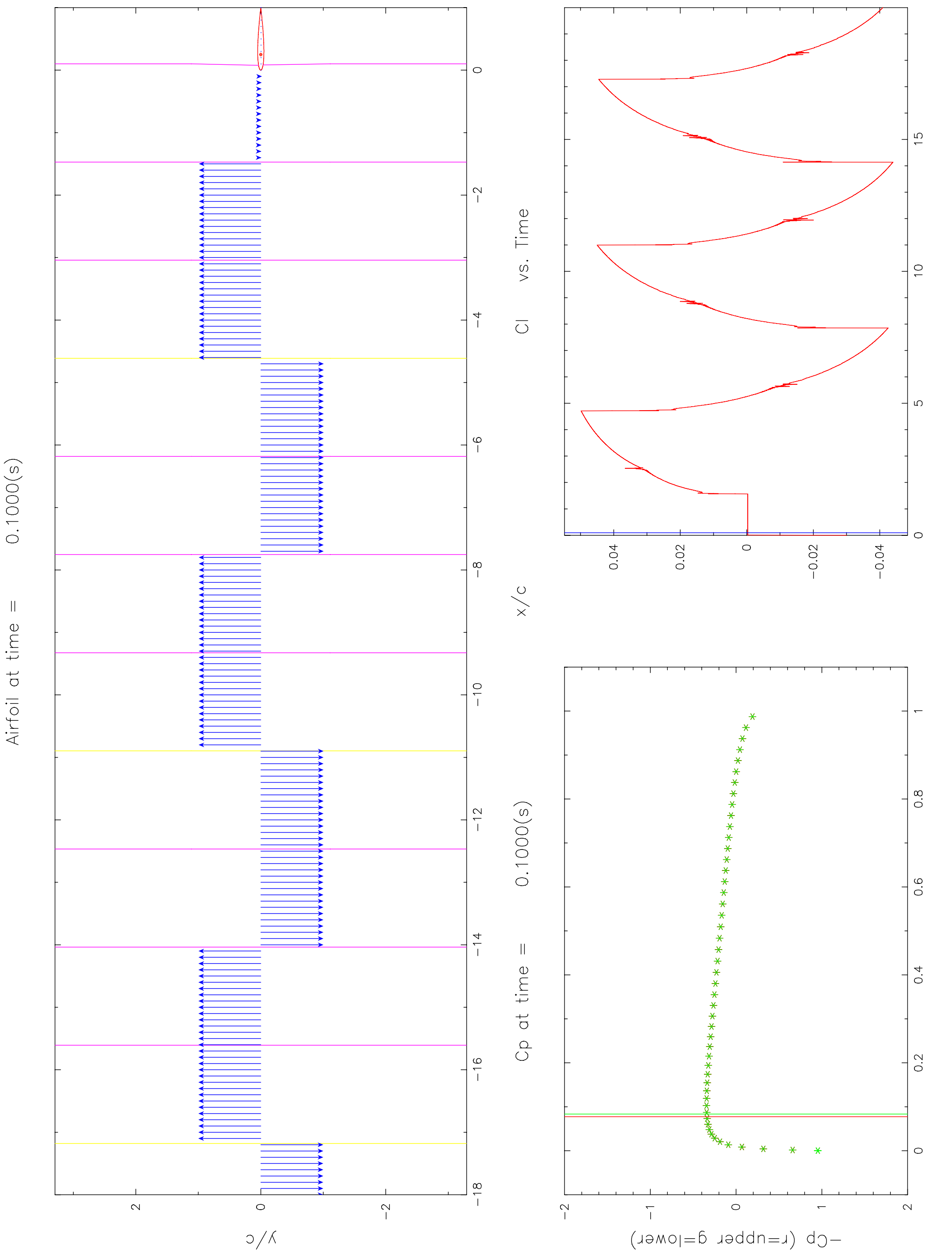

Figure 5.45. Visualization at $t = 2.0$ showing the location of the airfoil, wake, gust sheets, and selected x_2 velocities in the top panel, instantaneous C_p vs. x/c in the lower left panel, and C_l vs. t in the lower right panel.

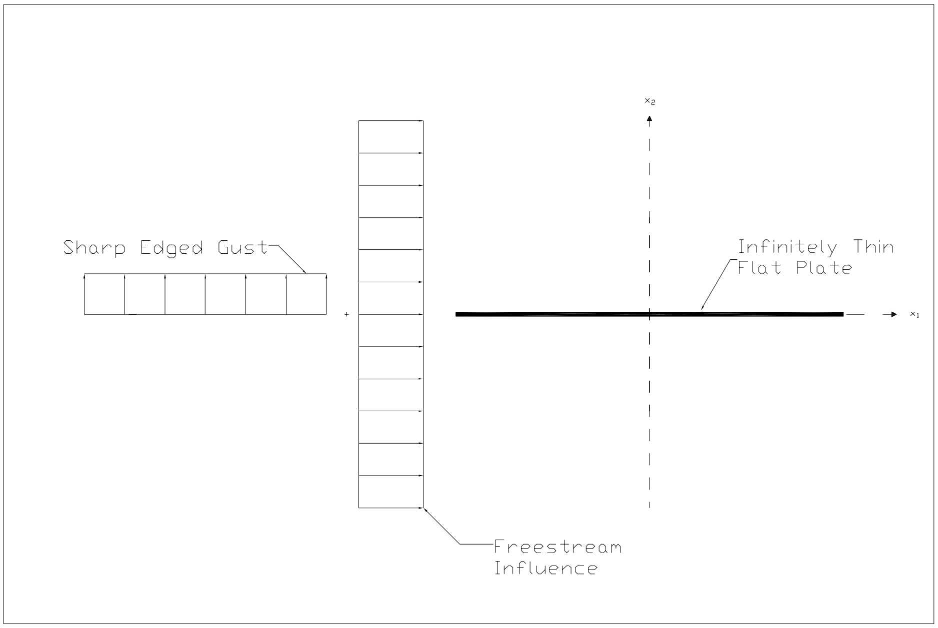

Figure 5.46. Stationary plate of infinitesimal thickness with sharp edge transverse gust

5.4 Kussner's Sharp Edge Gust

The Kussner sharp edge gust problem examines the lift and moment development on an airfoil in response to the a sudden change of incidence induced by a sharp edged transverse velocity perturbation. As with the Wagner problem, described section 5.1, the Kussner sharp edge gust demonstrates the effect of the body wake development on the airfoil lift and moment during the transition between equilibrium states.

5.4.1 Description

The Kussner sharp edge gust is an extension of the Sears periodic gust problem descibed in Section 5.3, but models a single gust propagating across the airfoil instead of the periodic gust. Using the same problem formulation, the sharp edge gust is modeled as a Fourier combination of the periodic gust. In this manner, the Kussner solution can be described using the same notation as the Sears problem

5.4.2 Solution

The solution for the Kussner sharp edge gust also starts with the influence of the velocity pertubation on the body, but in this case the sharp edge gust is modeled using a Fourier integral of the Sears periodic gust.

$$w_g = \frac{\bar{w}}{2\pi} \int_{-\infty}^{\infty} \frac{e^{i\frac{kU}{b}\left(t-\frac{b}{U}-\frac{x}{U}\right)}}{ik}\, dk = \frac{\bar{w}}{2\pi} \int_{-\infty}^{\infty} \frac{e^{ik(s-1)}}{ik}\, dk \tag{5.17}$$

This means the influence of the wake on the airfoil lift and moment in response to the sharp edged gust is a Fourier integral of the Sears function, Eq. (5.14). This is commonly refered to as the Kussner function.

$$\psi(s) = \frac{1}{2\pi i} \int_{-\infty}^{\infty} \frac{S(k)\, e^{ik(s-1)}}{k}\, dk \tag{5.18}$$

The Kussner function represents the ratio of instantaneous lift to the steady-state lift after the gust has past the airfoil. A commonly used approximation for the Kussner function is,

$$\psi(s) \approx 1 - 0.500e^{-0.130s} - 0.500e^{-s} \tag{5.19}$$

as is shown in Figure 5.2. Using Eqs. (5.17) and (5.18), lift and moment on a flat plate under the influence of a sharp edge gust is

$$L = 2\pi\rho U w_0 b \cdot \psi(s) \tag{5.20}$$

$$M_y = L\left(\frac{1}{2} + a\right) b \tag{5.21}$$

5.4.3 Comparison

The effect of a single sharp edge gust is analagous to the Wagner problem described in Section 5.1. Both problems examine the influence of wake development on lift and moment buildup for an airfoil transitioning between equilibrium states. In the case of the Wagner problem, the change in equilibrium states is due to a change in the freestream velocity magnitude relative to an airfoil held at a constant non-zero angle of attack, while in the case of the Kussner problem, the change in equilibrium is due to a change in the relative angle of attack between an airfoil held at a constant orientation and the mean freestream flow due to the influence of a sharp edged gust.

The Kussner problem provides a second verification of the gust model descibed in Section 4.2. As was shown in Figures 5.41 and 5.45, the influence of a set of gust sheets is analogous to the influence of a set of superimposed sharp edge gusts. Here, the influence of a single gust sheet, and a pair of gust sheets, will be compared to the Kussner solution.

5.4.3.1 Transient Panel Code Solution

To assist verification of the developed panel code, computed solutions for lift on thin symetrical airfoils are compared to predicted lift due to Kussners sharp edge gust. The panel code generated transient solutions for NACA 0006, 00010, and 0014 airfoils oriented at $\alpha_0 =$ 1, 2, and 4 deg relative to a uniform freestream parallel to the x_1 direction. Solutions were computed using non-dimensionalized time steps of 0.005, 0.075, and 0.010 corresponding to 4000, 3000, and 2000 iterations, respectively. Calculated lift coefficient for each simulation were normalized by corresponding steady-state lift values, allowing a comparison to the Kussner function, Eq. (5.19). It should be noted that these are the same panel code solutions used in the Wagner comparison presented in Section 5.1.3.

Figure 5.47 compares panel code solutions for airfoils of different thicknesses to the Kussner function. The panel code solutions are computed for NACA 0008, 0010, and 0012 airfoils at $\alpha_0 = 1\,\text{deg}$ using a normalized time step of 0.005. As in the case of the Wagner problem, the panel code solutions approach the Kussner function as airfoil thickness decreases

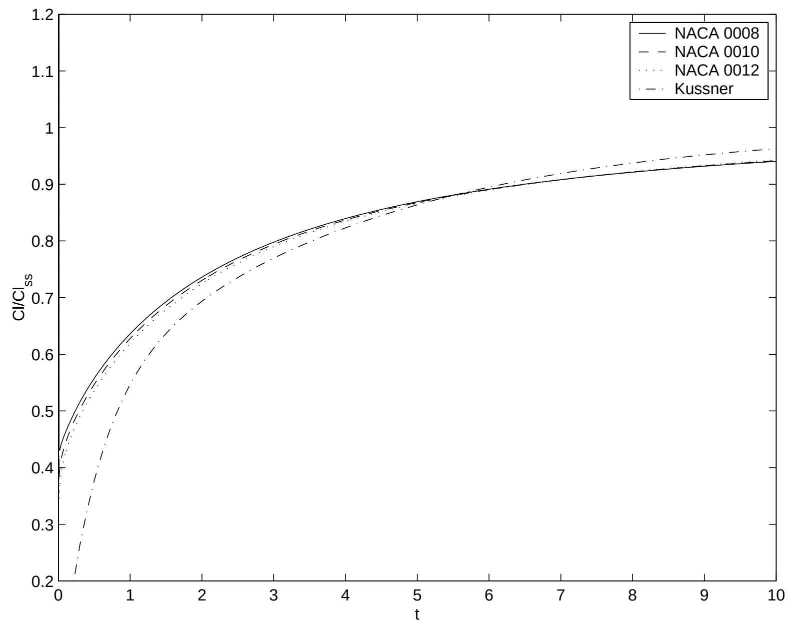

Figure 5.47. Transient lift solutions normilized by the corresponding steady state lift for NACA 0008, 0010, and 0012 airfoils oriented at at $\alpha_0 = 1\,\text{deg}$ relative to the time-averaged freestream computes using a normalized time step of 0.005 compared to Eq. (5.19)

Since it was shown in Figures 5.4 and 5.5 that orientation and time step have a negligible infuence on the panel code solution, the comparisons for variying orientation and time step will be omitted here.

5.4.3.2 Single and Double Gust Sheets

Figures 5.48 and 5.49 compare the lift and moment coefficients for a single gust sheet initiated three chord lengths upstream of a NACA 0010 airfoil to the lift coefficient predicted by the corresponding Kussner solution. The panel code solution was computed for $\alpha_0 = 0$ using a non-dimensional time step of 0.010. The gust sheet possessed a bound circulation per unit length of γ = -0.02, corresponding to the Kussner solution for a gust strength of $\bar{w} = 0.01$.

Figure 5.50 shows a visualization of the location of the airfoil, wake, and gust sheets in the top panel, the instantaneous pressure coefficients along the airfoil in the lower left panel, and the coefficent of lift time-history in the lower right panel. Velocity vectors representing

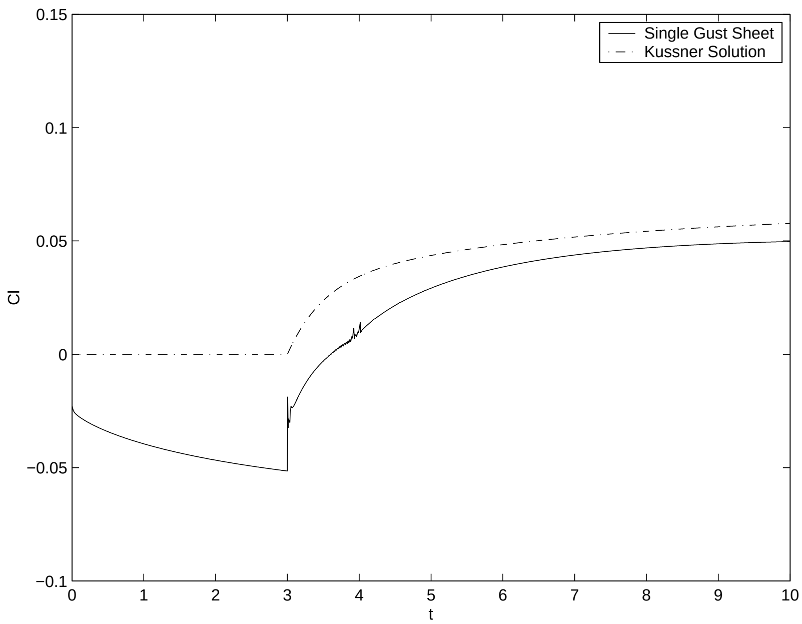

Figure 5.48. C_l for a single gust sheet of strength $\gamma = -0.02$ propagating across a NACA 0010 airfoil oriented at $\alpha_0 = 0.0$ to the time-averaged freestream computed using a time step of 0.010 compared to the Kussner sharp edge gust with an amplitude of $\bar{w} = 0.01$.

the freestream velocity in the x_2 direction, sampled at locations in the flowfield and scaled by an arbitrary factor have been added to the location plot in the top panel.

The panel code solution for a single gust sheet shows good agreement with the shape of the predicted Kussner solution, but has an offset in lift an moment due to the influence of the gust as it approaches the airfoil. It turns out that the initial influence can be negated by using a pair of gust sheets of equal but opposite circulation strength. This pair of gust sheets closely model the influence of two sharp edged gusts offset by some period of time.

Figures 5.51 and 5.52 compare the lift and moment coefficients for a pair of gust sheets initiated at three and five chord lengths upstream of a NACA 0010 airfoil to the corresponding Kussner solution. The panel code solution was computed for $\alpha_0 = 0$ using a non-dimensional time step of 0.010. The gust sheets possesed bound circulation per unit length of $\gamma =$ -0.02 and 0.02, respectively, corresponding to a Kussner solution for gust strengths of $\bar{w} = 0.02$ and -0.02 located at $\tau = 3$ and 5.

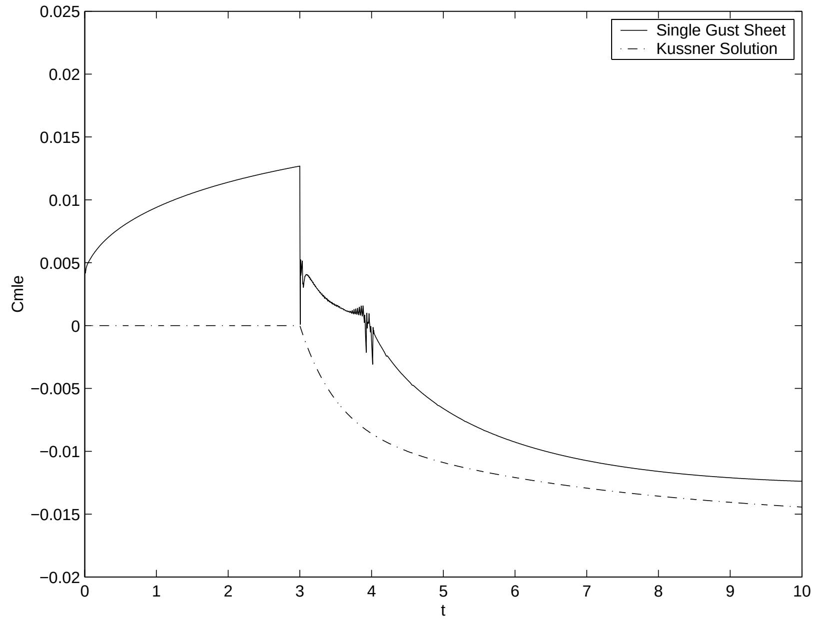

Figure 5.49. $C_{m_{le}}$ for a single gust sheet of strength $\gamma = -0.02$ propagating across a NACA 0010 airfoil oriented at $\alpha_0 = 0.0$ to the time-averaged freestream computed using a time step of 0.010 compared to the Kussner sharp edge gust with an amplitude of $\bar{w} = 0.01$.

Figure 5.53 shows a visualization of the location of the airfoil, wake, and gust sheets in the top panel, the instantaneous pressure coefficients along the airfoil in the lower left panel, and the coefficent of lift time-history in the lower right panel. Velocity vectors representing the freestream velocity in the x_2 direction, sampled at locations in the flowfield and scaled by an arbitrary factor have been added to the location plot in the top panel.

The agreement shown between the panel code solutions utilizing the freestream gust model and the Kussner solutions for superimposed sharp edged gusts provides a second verification of the freestream gust model.

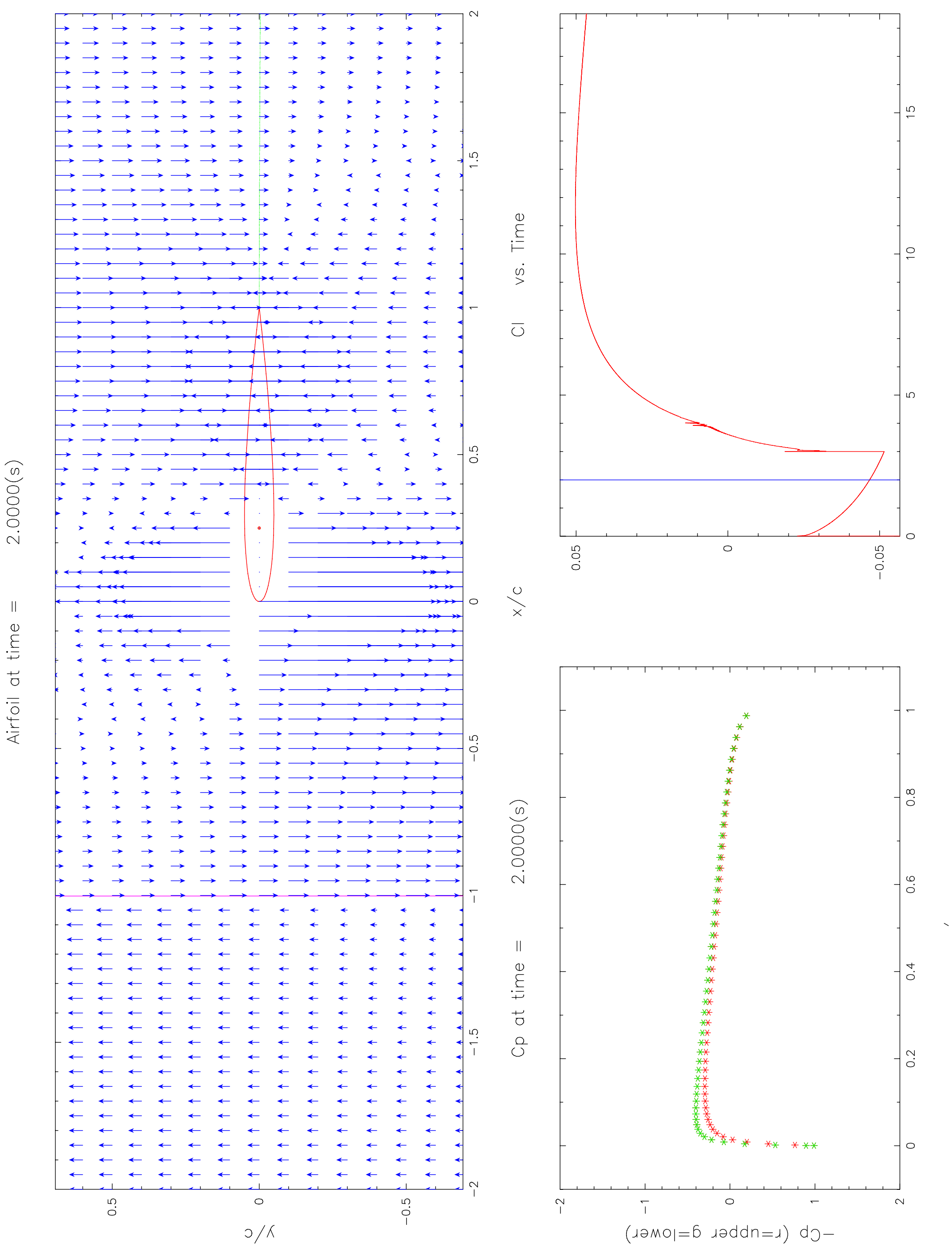

Figure 5.50. Visualization at $t = 2.0$ showing the location of the airfoil, wake, gust sheets, and selected x_2 velocities in the top panel, instantaneous C_p vs. x/c in the lower left panel, and C_l vs. t in the lower right panel.

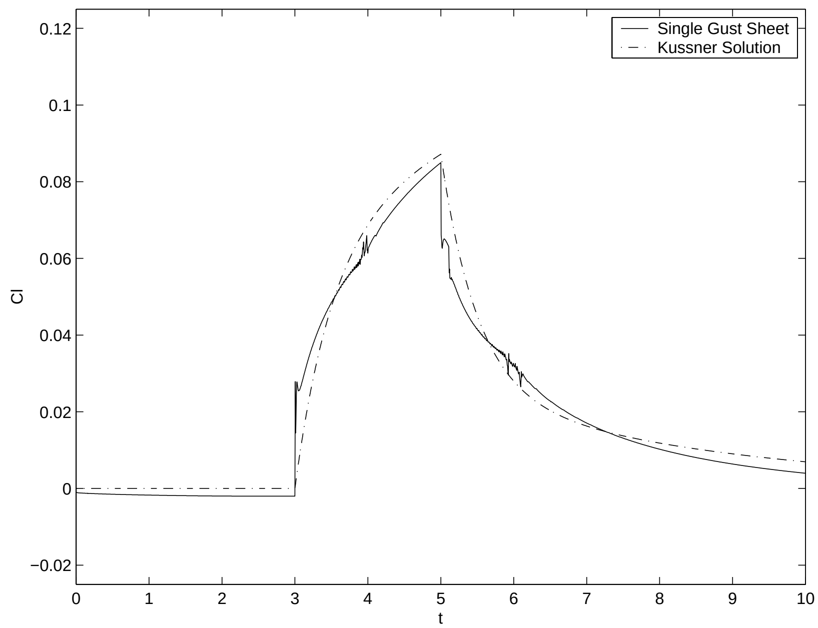

Figure 5.51. C_l for a pair of gust sheets of strength $\gamma =$ -0.02 and 0.02 propagating across a NACA 0010 airfoil oriented at $\alpha_0 = 0.0$ to the time-averaged freestream computed using a time step of 0.010 compared to the Kussner sharp edge gusts with amplitudes of $\bar{w} = 0.01$ and -0.01.

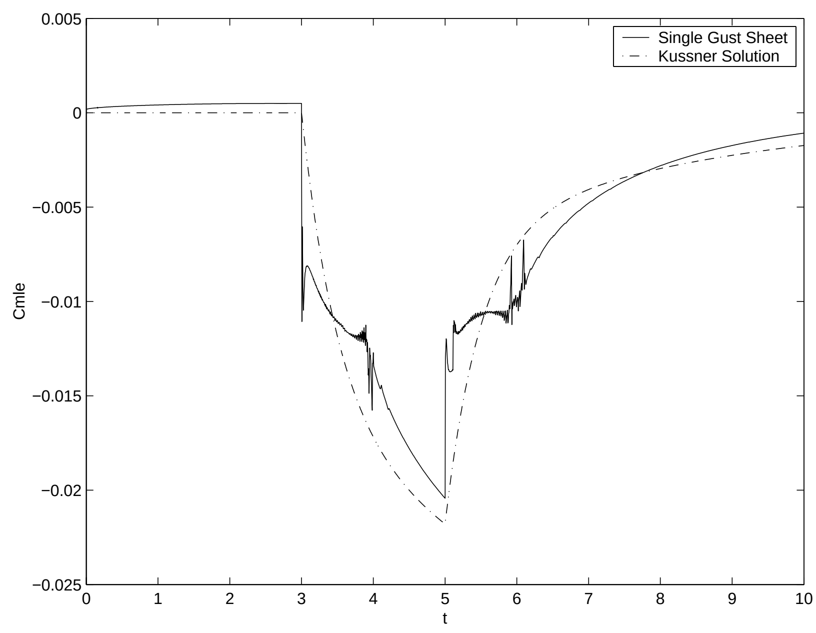

Figure 5.52. $C_{m_{le}}$ for a pair of gust sheets of strength $\gamma =$ -0.02 and 0.02 propagating across a NACA 0010 airfoil oriented at $\alpha_0 = 0.0$ to the time-averaged freestream computed using a time step of 0.010 compared to the Kussner sharp edge gusts with amplitudes of $\bar{w} = 0.01$ and -0.01.

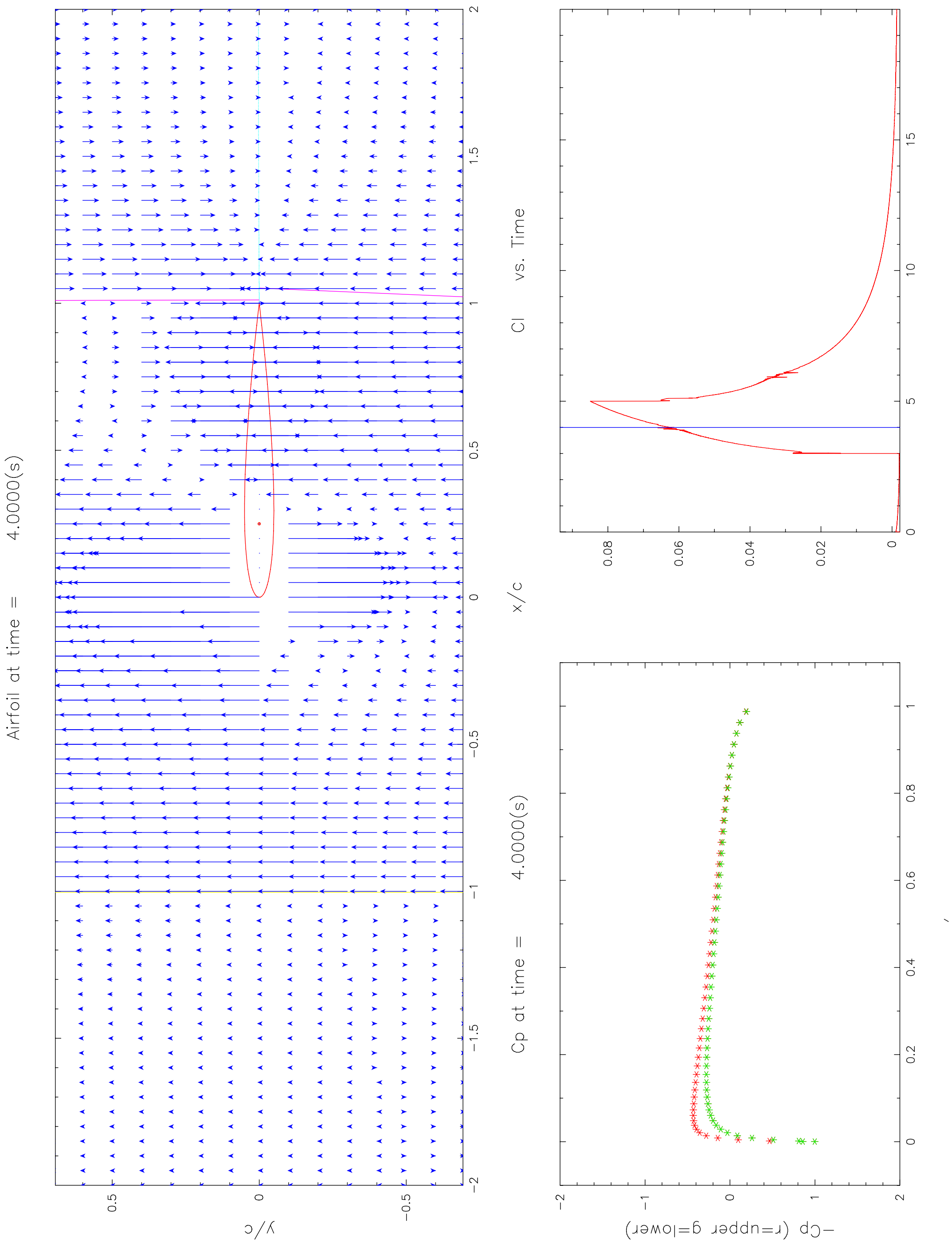

Figure 5.53. Visualization at $t = 4.0$ showing the location of the airfoil, wake, gust sheets, and selected x_2 velocities in the top panel, instantaneous C_p vs. x/c in the lower left panel, and C_l vs. t in the lower right panel.

5.5 Free Response

The responce of an elastic airfoil to aerodynamic forcing is a phenomenon which is of interest to a wide range of aerodynamic fields. Flutter, the divergent structural responce due to self induced aerodynamic forcing, has been well studied because of its impact on the design of aircraft, turbo-machinery, and civil structures. Several analytic techniques have been developed which provide a means to predict the flutter boundary, the point at which the damping of at least one mode goes to zero correponding to the transition from a stable to an unstable aeroelastic system. [10, 11, 4, 6] The flutter boundary is of interest here because it provides a means to correlate the time-domain panel code solution with the classical frequency-domain techniques used to predict the onset of flutter.

Classical frequency-domain techniques based on the work of Theodorsen [10, 11], assume that body motion at the flutter boundary is harmonic. Thus, the problem of finding the flutter boundary reduces to finding the flight conditions which produce harmonic body motion. The assumption of harmonic motion has the added benefit in that it allows for the use of linear or quasi-steady aerodynamic models, which when coupled with the equations of motion, results in a eigen-value problem. Classical techniques are limited in that they only identify the location of the flutter boundary and do not accurately predict modal dampening for flight conditions which are not close to the flutter boundary.

The p-k method [6], which is utilized for this validation, combines an arbitrary aerodynamic model with a structural model incorporating modal dampening. This allows for the responce of the system at a variety of flight conditions to be estimated, as well as the determination of the flutter boundary. The flutter boundary correlates to the flight condition where dampening for at least one of the structural modes goes to zero.

5.5.1 Solution

The p-k method starts with the equations of motion given in Eq. (4.1), restated here with suitable substitutions for convenience.

$$m\ddot{h} + mbx_\alpha\ddot{\alpha} + k_h h = -L \tag{5.22a}$$

$$mbx_\alpha\ddot{h} + I_\alpha\ddot{\alpha} + k_\alpha\alpha = M_y \tag{5.22b}$$

Assuming harmonic motion,

$$\alpha = \bar{\alpha}e^{\nu t} \tag{5.23a}$$

$$h = \bar{h}e^{\nu t} \tag{5.23b}$$

$$L = \bar{L}e^{i\omega t} \tag{5.23c}$$

$$M_y = \bar{M}_y e^{i\omega t} \tag{5.23d}$$

and rewriting the moment about the elastic axis in terms of Lift and the moment about the quarter chord,

$$M_y = M_{\frac{c}{4}} + b\left(\frac{1}{2} + a\right)L \tag{5.24}$$

the equations of motion become

$$m\nu^2\bar{h}e^{\nu t} + mbx_\alpha\nu^2\bar{\alpha}e^{\nu t} + k_h\bar{h}e^{\nu t} = -\bar{L}e^{i\omega t} \tag{5.25a}$$

$$mbx_\alpha\nu^2\bar{h}e^{\nu t} + I_\alpha\nu^2\bar{\alpha}e^{\nu t} + k_\alpha\bar{\alpha}e^{\nu t} = \bar{M}_{\frac{c}{4}}e^{i\omega t} + b\left(\frac{1}{2} + a\right)\bar{L}e^{i\omega t} \tag{5.25b}$$

The forcing functions in terms of coefficent representations of lift and moment are

$$\bar{L} = \frac{L}{e^{i\omega t}} = -\pi\rho_\infty b^3\omega^2\left[l_h\left(k, M_\infty\right)\frac{\bar{h}}{b} + l_\alpha\left(k, M_\infty\right)\bar{\alpha}\right] \tag{5.26a}$$

$$\bar{M}_{\frac{c}{4}} = \frac{M_{\frac{c}{4}}}{e^{i\omega t}} = \pi\rho_\infty b^4\omega^2\left[m_{\frac{c}{4}h}\left(k, M_\infty\right)\frac{\bar{h}}{b} + m_{\frac{c}{4}\alpha}\left(k, M_\infty\right)\bar{\alpha}\right] \tag{5.26b}$$

where the coefficents in terms of the Theodorsen periodic lift and moment are

$$l_h(k, M_\infty) = 1 - \frac{i2C(k)}{k} \tag{5.27a}$$

$$l_\alpha(k, M_\infty) = -\frac{2C(k)}{k^2} - \frac{i(1-2a)C(k)}{k} - \frac{i}{k} - a \tag{5.27b}$$

$$m_{\frac{c}{4}h}(k, M_\infty) = \frac{1}{2} \tag{5.27c}$$

$$m_{\frac{c}{4}\alpha}(k, M_\infty) = -\frac{i}{k} + \frac{1}{8} - \frac{a}{2} \tag{5.27d}$$

Simplifying Eq. (5.25) by $e^{i\omega t}$ and rewriting the forcing function in terms of Eq. (5.26) gives

$$m\nu^2\bar{h} + mbx_\alpha\nu^2\bar{\alpha} + k_h\bar{h} = \pi\rho_\infty b^3\omega^2\left[l_h(k, M_\infty)\frac{\bar{h}}{b} + l_\alpha(k, M_\infty)\bar{\alpha}\right] \tag{5.28a}$$

$$\begin{aligned} mbx_\alpha\nu^2\bar{h} + I_\alpha\nu^2\bar{\alpha} + k_\alpha\bar{\alpha} =& \pi\rho_\infty b^4\omega^2\left[m_{\frac{c}{4}h}(k, M_\infty)\frac{\bar{h}}{b} + m_{\frac{c}{4}\alpha}(k, M_\infty)\bar{\alpha}\right] - \\ & \pi\rho_\infty b^4\omega^2\left[\frac{1}{2} + a\right]\left[l_h(k, M_\infty)\frac{\bar{h}}{b} + l_\alpha(k, M_\infty)\bar{\alpha}\right] \end{aligned} \tag{5.28b}$$

where $\nu = i\omega$. Simplifying Eq. (5.28) and collecting terms

$$\left[\frac{m\nu^2 + k_h}{\pi\rho_\infty b^3\omega^2}\right]\bar{h} + \left[\frac{mbx_\alpha\nu^2}{\pi\rho_\infty b^3\omega^2}\right]\bar{\alpha} = \left[l_h(k, M_\infty)\frac{\bar{h}}{b} + l_\alpha(k, M_\infty)\bar{\alpha}\right] \tag{5.29a}$$

$$\begin{aligned} \left[\frac{mbx_\alpha\nu^2}{\pi\rho_\infty b^4\omega^2}\right]\bar{h} + \left[\frac{I_\alpha\nu^2 + k_\alpha}{\pi\rho_\infty b^4\omega^2}\right]\bar{\alpha} =& \left[m_{\frac{c}{4}h}(k, M_\infty)\frac{\bar{h}}{b} + m_{\frac{c}{4}\alpha}(k, M_\infty)\bar{\alpha}\right] - \\ & \left[\frac{1}{2} + a\right]\left[l_h(k, M_\infty)\frac{\bar{h}}{b} + l_\alpha(k, M_\infty)\bar{\alpha}\right] \end{aligned} \tag{5.29b}$$

for which appropriate substitutions are made gives.

$$\left[p^2\mu V^2 + \mu\sigma^2 - l_h V^2 k^2\right]\frac{\bar{h}}{b} + \left[p^2\mu x_\alpha V^2 - l_\alpha V^2 k^2\right]\bar{\alpha} = 0 \tag{5.30a}$$

$$\begin{aligned} &\left[p^2\mu x_\alpha V^2 - m_{\frac{c}{4}h}V^2k^2 + \left(\frac{1}{2} + a\right)l_h V^2 k^2\right]\frac{\bar{h}}{b} + \\ &\left[p^2\mu r_\alpha^2 V^2 + \mu r_\alpha^2 - m_{\frac{c}{4}\alpha}V^2k^2 + \left(\frac{1}{2} + a\right)l_\alpha V^2 k^2\right]\bar{\alpha} = 0 \end{aligned} \tag{5.30b}$$

Equation (5.30) can be rewriten in matrix form

$$\begin{bmatrix} N_1 & N_2 \\ N_3 & N_4 \end{bmatrix} \begin{pmatrix} \frac{\bar{h}}{b} \\ \bar{\alpha} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{5.31}$$

where the coeffients N_i are

$$N_1 = p^2 \mu V^2 + \mu \sigma^2 - l_h V^2 k^2 \tag{5.32a}$$

$$N_2 = \frac{m b x_\alpha \nu^2}{\pi \rho_\infty b^3 \omega^2} \tag{5.32b}$$

$$N_3 = p^2 \mu x_\alpha V^2 - m_{\frac{c}{4}h} V^2 k^2 + \left(\frac{1}{2} + a\right) l_h V^2 k^2 \tag{5.32c}$$

$$N_4 = p^2 \mu r_\alpha^2 V^2 + \mu r_\alpha^2 - m_{\frac{c}{4}\alpha} V^2 k^2 + \left(\frac{1}{2} + a\right) l_\alpha V^2 k^2 \tag{5.32d}$$

The only non-trivial solution to Eq. (5.31) is when the determinant of $N = 0$ and $p = ik$.

$$\det \begin{bmatrix} N_1 & N_2 \\ N_3 & N_4 \end{bmatrix} = 0 \tag{5.33}$$

This values of p which satisfy the non-trivial solution can be found for a given flight condition through an iterative method.

1. Assume an initial value for reduced frequency k
2. Calculate forcing based on k and M_∞
3. Calculate p by solving $\det[N] = 0$
4. Set $k = \Im(p)$
5. Repeat step 2 through 4 until the values of p and k converge

In this manner, the frequency of forced osscilation and modal dampening can be determined for a range of flight conditions, and this trend can be used to determine the flutter boundary by finding the flight condition for which modal dampening for any mode goes to zero, or $\Re(p) = 0$.

5.5.2 Comparison

To verify the panel code structural model, the flutter boundary as computed from a set of time-domain panel code solutions is compared to the flutter boundary estimated using the frequency-domain p-k method.

The first step in determining the flutter boundary for either solution method is specifying the sectional structural charictarisics of the airfoil to be modeled. The structural characteristics are specified using the following non-dimensional parameters; axis location, a, radius of gyration, r_α, static unbalance, x_α, denity ratio, μ, pitching natural frequency, ω_α, and plunging natural frequency, ω_h.

Given a set of structural charictaristics, the flutter boundary is estimated using the p-k method as outlined in Section 5.5.1. This estimated flutter boundary is then used as a reference point for a set of panel code solutions modeling freestream velocity at, above, and below the estimated flutter velocity. Modal frequency and damping is then calculated from the airfoil motion history for each panel code solution. Thus, the flutter boundary can be found by determining the freestream velocity where modal damping for at least one mode goes to zero.

Two methods can be used Thus, to model different "flight" conditions about the flutter boundary, the reduced modal freqiencies are varied by a value of 2% in a range between 90% and 110% of the estimated flutter boundary.

The flutter boundary was estimated for a NACA 0007 airfoil with the following structural charictarisics; $a = -1/5$, $r_a = 0.48$, $x_a = 0.10$, $\mu = 20.0$, and $\omega_h/\omega_\alpha = 2/5$. Using the p-k method, the flutter boundary was found at a freestream velocity of $U_f = 2.17$, which

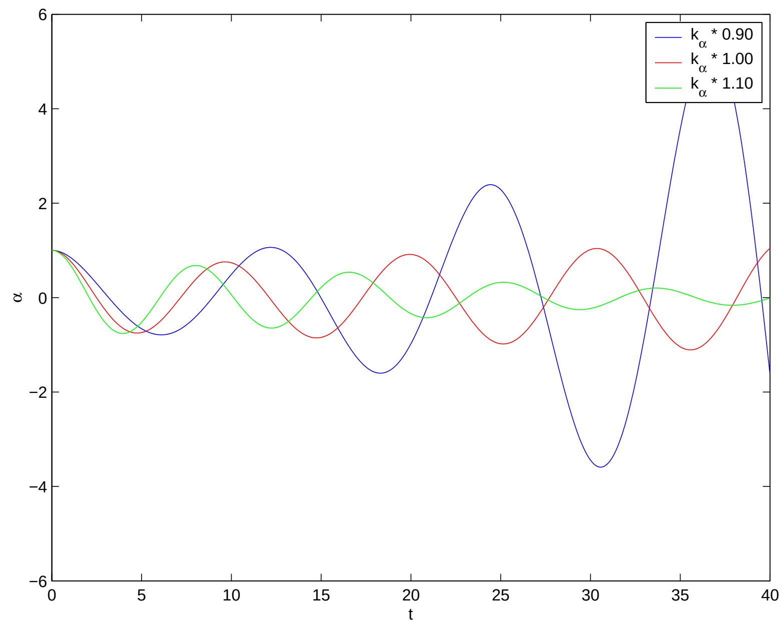

Figure 5.54. Pitch history for the panel code free response simulation above, at, and below the predicted flutter boundary.

corresponds to a reduced pitching frequency of $k_\alpha = 0.92$ and reduced pitching frequency of $k_h = 0.36$.

Panel code solutions were computed at reduced frequencies ranging from 90% to 110% of the reduced pitching and plunging frequencies in steps of 0.02%. Each simulation was initiated as a transient solution with the airfoil initially oriented at $\alpha_0 = 1\,\text{deg}$ relative to the time-averaged freestream flow, and computed using a non-dimensional time step of 0.01 for 4000 iterations.

Figures 5.54 through 5.58 show the time history of pitch, plunge, lift and moment coefficents, and plunge vs. pitch. Panel code simulations in Figures 5.54 through 5.58 represent the solution for reduced frequencies at 90%, 100%, and 110% of the reduced frequencies at the flutter boundary. It can be observed from Figures 5.54 and 5.55 that the panel code flutter boundary will be at a lower freestream velocity than predicted by the p-k method.

To determine modal damping, the assumption is made that the motion is harmonic close to the flutter boundary. Therefore, pitch and plunge can be approximated by the following

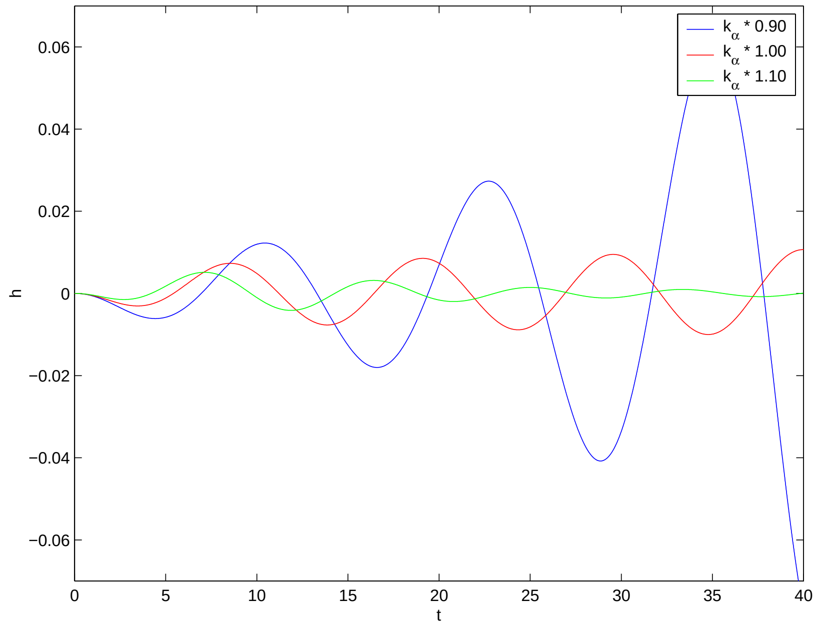

Figure 5.55. Plunge history for the panel code free response simulation above, at, and below the predicted flutter boundary.

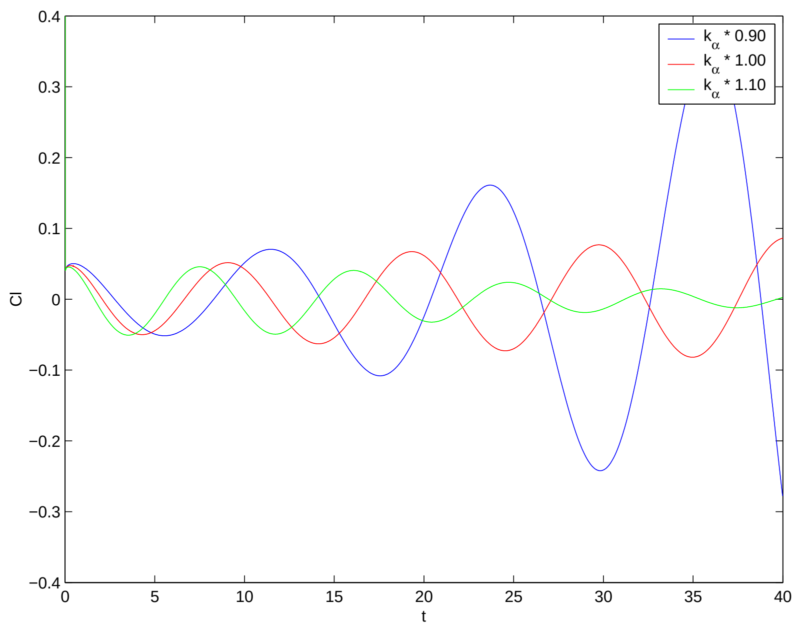

Figure 5.56. C_l history for the panel code free response simulation above, at, and below the predicted flutter boundary.

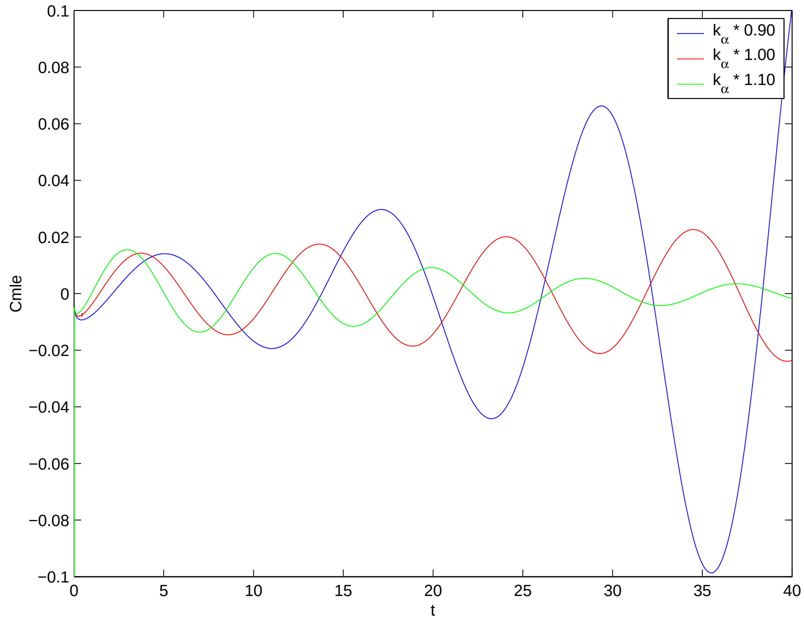

Figure 5.57. $C_{m_{le}}$ history for the panel code free response simulation above, at, and below the predicted flutter boundary.

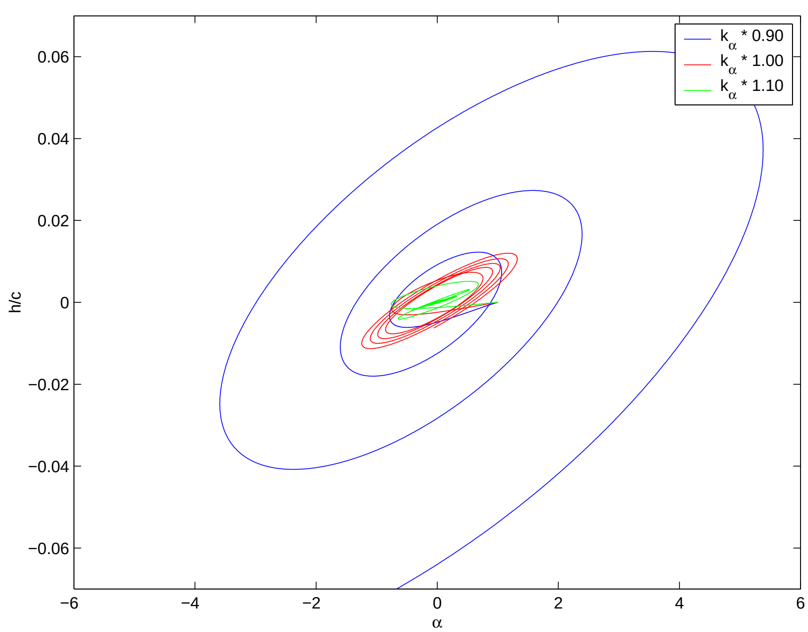

Figure 5.58. Plunge vs. Pitch for the panel code free response simulation above, at, and below the predicted flutter boundary.

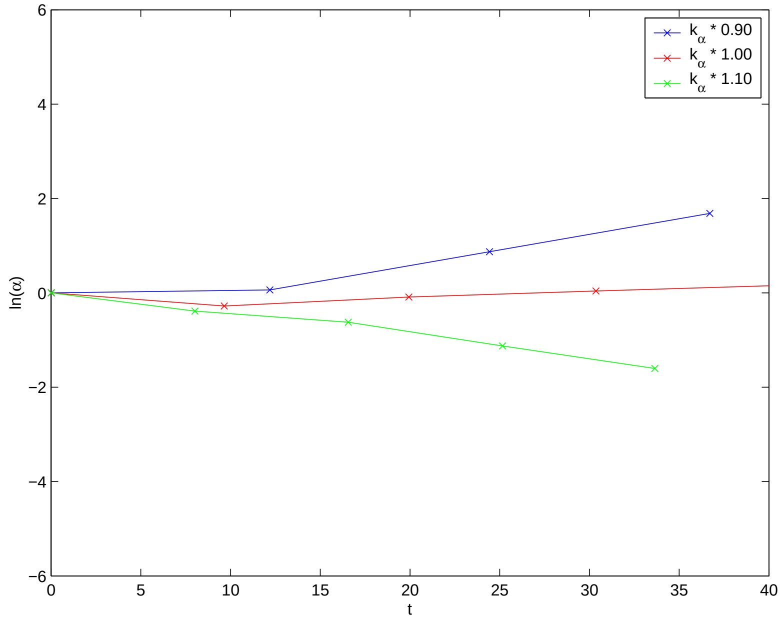

Figure 5.59. Determining modal damping for pitch.

equations.

$$\alpha\left(t\right) = \alpha_0 \exp\left(\eta_\alpha t\right) \sin\left(k_\alpha t\right) \tag{5.34}$$

$$h\left(t\right) = h_0 \exp\left(\eta_h t\right) \sin\left(k_h t\right) \tag{5.35}$$

By assuming damped harmonic motion, the damping factor corresponds to the slope of the line through the local maxima of $\ln\left(\alpha\left(t\right)\right)$ and $\ln\left(h\left(t\right)\right)$. Figures 5.59 and 5.60 show the natural log of the local maxima for pitch and plunge.

Figures 5.61 and 5.62 show modal damping and frequency normalized by the reduced pitching frequency as a function of freestream velocity for both the panel code solutions and the p-k method. The panel code solution differs from the p-k method in that it predicts the flutter boundary at a slightly smaller freestream velocity, and both modal damping and frequency coalesce at the flutter boundary.

Differences between the two solutions are to be expected due to the nature of the solutions. The panel code is a non-linear aerodynamic solver while the p-k method is based on the Theodorsen solution, which is the subject of Section 5.2. The p-k method assumes harmonic

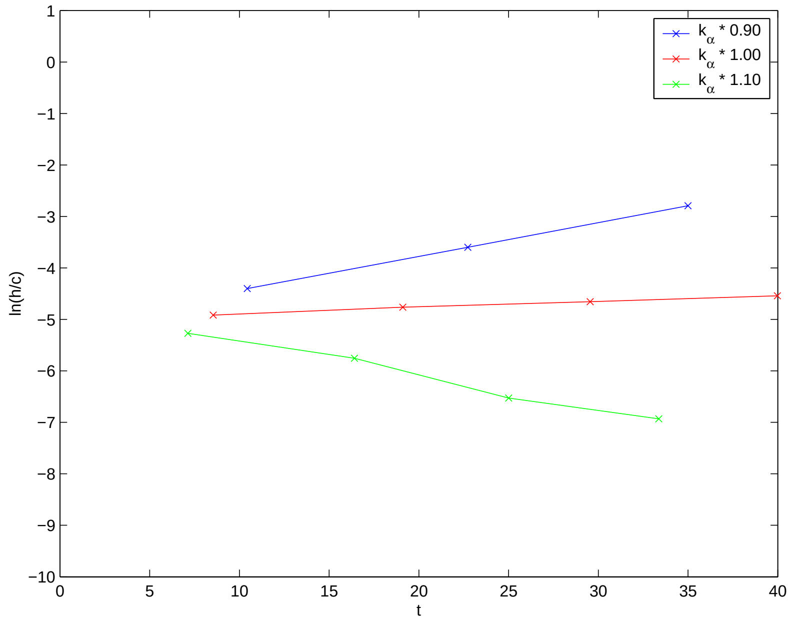

Figure 5.60. Determining modal damping for plunge.

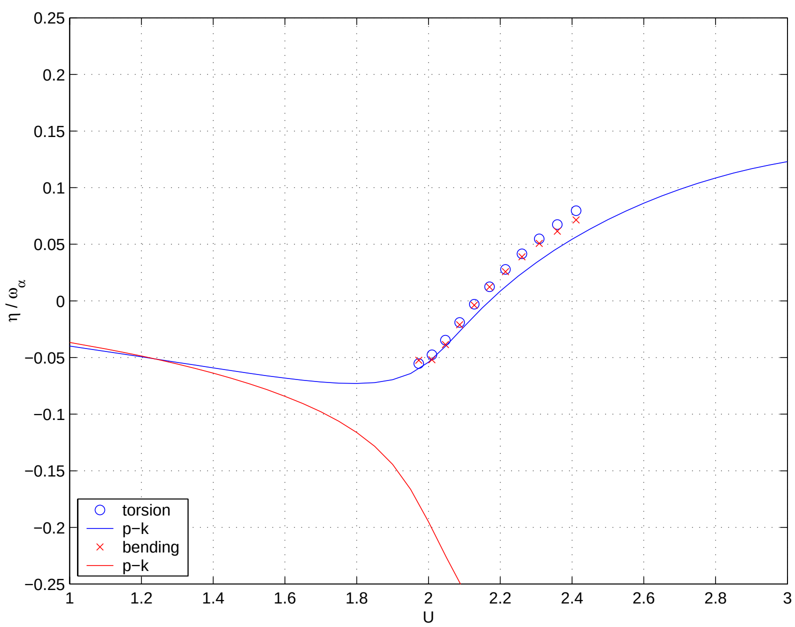

Figure 5.61. Normalized modal damping vs. freestream velocity for the panel code solution compared to the p-k method.

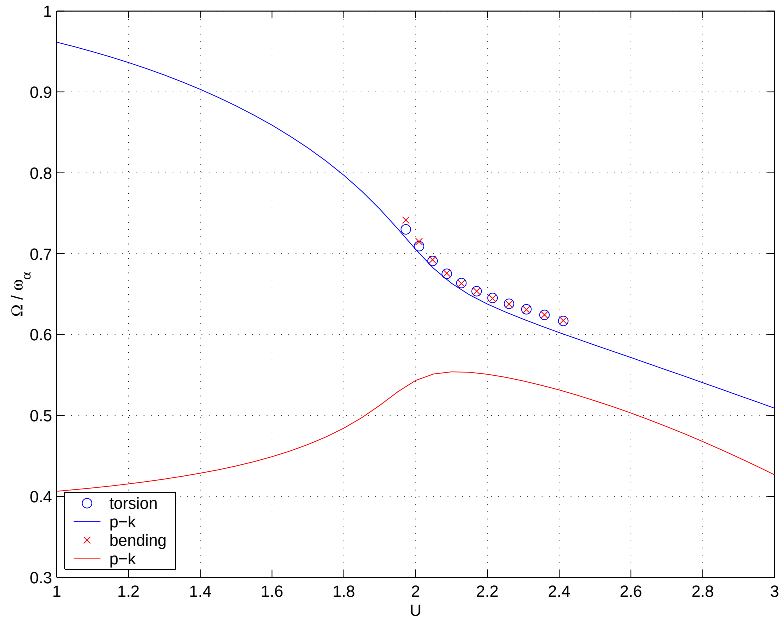

Figure 5.62. Normalized modal frequency vs. freestream velocity for the panel code solution compared to the p-k method.

motion while the panel makes no assumption of mode shape except in the calculation of modal damping. Despite these differences, the panel code shows a good agreement as to location of the flutter boundary with the p-k method.

CHAPTER 6

FORCED RESPONSE

This chapter provides a brief examination of airfoil forced responce as modeled by the panel code.

6.1 Description

The panel code models forced responce by combining the freestream gust model described in Section 4.2 with the structural model described in Section 4.3. The freestream gust model was shown to model a periodic freestream disturbance comparable to the Sears periodic gust in Section 5.3. It was also shown that the influence of the gust model is highly dependent on the discretization of the freestream gust into representative gust sheets. The structural model was shown to predict the self excited flutter boundary when coupled with the panel code aerodynamic model in Section 5.5.

Using the combined models, a set of panel code solutions were computed to determine the response of an airfoil close to its flutter boundary to the influence of a set of freestream gusts possesing a gust frequency at intervals about the airfoil reduced pitching frequency.

The solutions were computed for an NACA 0007 airfoil with the same structural charictaristics as were used to verify the flutter boundary in Section 5.5. As in Section 5.5, each simulation was initiated as a transient solution and computed using a non-dimensional time step of 0.01, however, the airfoil was given an initial orientation of $\alpha_0 = 0\,\text{deg}$ relative to the time-averaged freestream flow. Each simulation used the same freestream gust discretization of four gust sheets per gust period as used in Section 5.3, but the gust period was varied for each simulaiton to force the airfoil at its reduced pitching frequency.

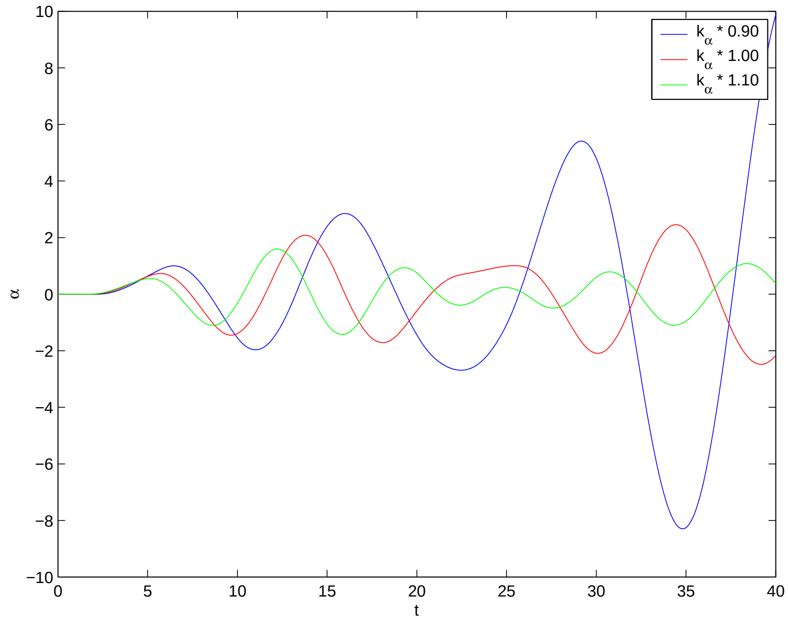

Figure 6.1. Pitch history for the panel code forced response simulation above, at, and below the predicted flutter boundary.

Figures 6.1 through 6.5 show the time history of pitch, plunge, lift and moment coefficents, and plunge vs. pitch for the forced responce solutions. Panel code solutions in Figures 6.1 through 6.5 represent the solution for a NACA 0007 airfoil having reduced pitching and plunging frequencies of 90%, 100%, and 110% of the reduced frequencies at the flutter boundary under the influence of a periodic freestream gust with a gust frequency corresponds to the airfoil pitching frequency.

Without performing a comprehensive study into airfoil responce due to a preiodic gust, no conclusions will be drawn about the accuracy of the time domain motion, however, the breif study does show that the freestream gust model can be coupled with the airfoil structural model to produce airfoil responce due to external aerodynamic forcing.

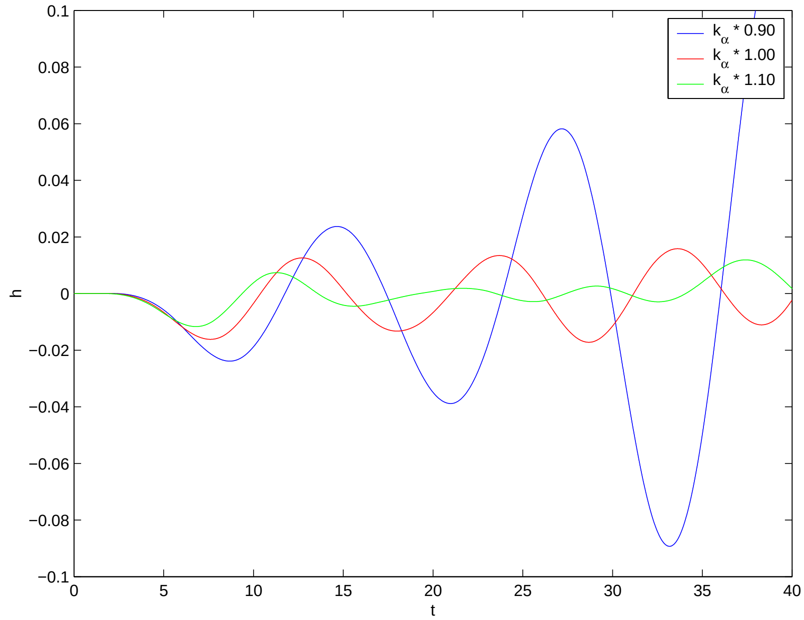

Figure 6.2. Plunge history for the panel code forced response simulation above, at, and below the predicted flutter boundary.

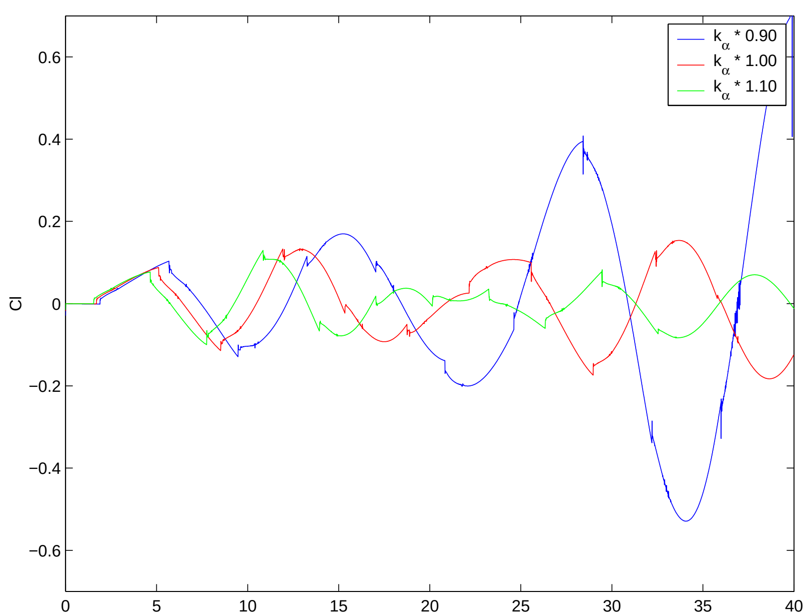

Figure 6.3. C_l history for the panel code forced response simulation above, at, and below the predicted flutter boundary.

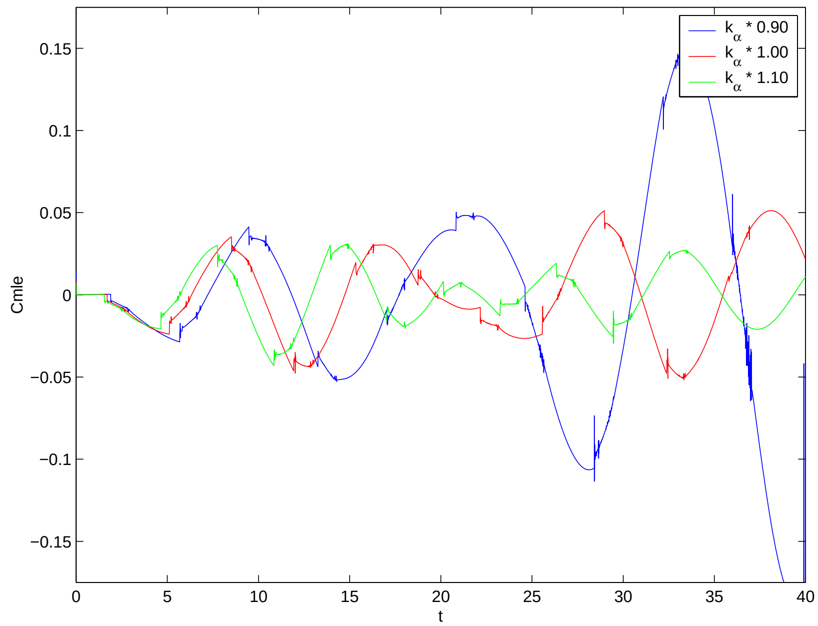

Figure 6.4. $C_{m_{le}}$ history for the panel code forced response simulation above, at, and below the predicted flutter boundary.

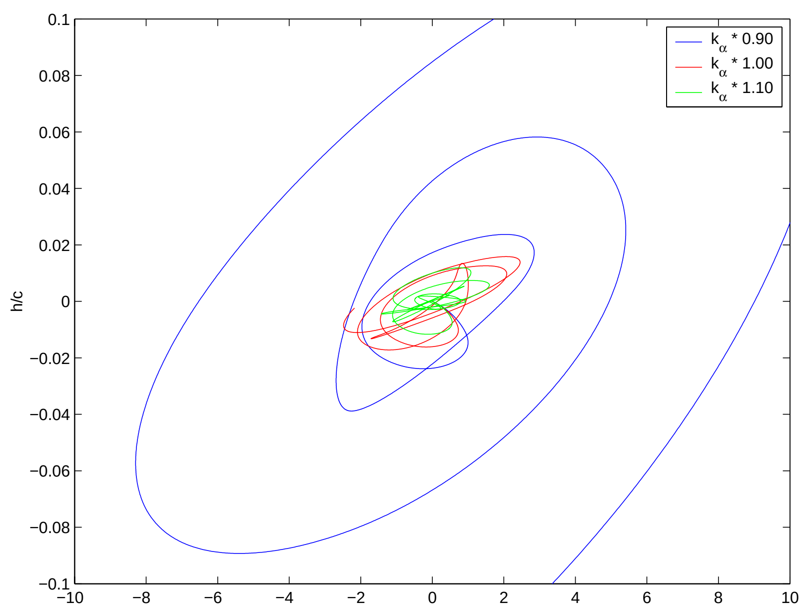

Figure 6.5. h/c vs. α for the panel code forced response simulation above, at, and below the predicted flutter boundary.

CHAPTER 7

SUMMARY AND CONCLUSIONS

An unsteady panel code has been developed as a computational tool to investigate the influence of aerodynamic damping on airfoil aeromechanical response. The inclusion of a freestream gust model enables the panel code to simulate freestream gust perturbations of arbitrary shape and magnitude. The inclusion of a TDOF structural model also enables the panel code to model structural response due to both self-induced and gust-induced aerodynamic forcing.

7.1 Validation

The panel code was compared to classic problems in unsteady aerodynamics having known analytic solutions. The panel code compared favorably to the Wagner solution for an instantaneous change in airfoil attitude. This comparison demonstrated the effect of proper modeling of the unsteady Kutta condition and wake model on lift development.

The panel code also compared favorably to the Theodorsen solution for a pitching and plunging airfoil. Small variations in phase and amplitude were observed between the panel code and Theodorsen lift solutions, but the differences were reasonable considering the separate wake model formulations and airfoil thickness influence. This comparison demonstrated the effect of proper formulation of the unsteady boundary condition on an airfoil undergoing relative motion, providing further validation of the unsteady Kutta condition and panel code wake model.

The modified panel code solution compared favorably to the Sears solution for a periodic transverse velocity perturbation. As with the Theodorsen solution, small variations in phase and amplitude were observed between the modified panel code and Sears solutions, but again, these differences are reasonable since the Sears solution uses the same wake formulation as Theodorsen. Comparison of the panel code employing the freestream gust model did not compare quite as favorably to the Sears solution. Agreement between the Sears solution and the freestream gust model was highly dependent on gust discretization into an appropriate set of gust sheets. Agreement in lift amplitude and phase was achieved using a discretization of four gust sheets per gust period, but the shape of the panel code lift curve resembled a superposition of sharp edge gusts rather than the sinusoidal shape of the Sears solution. Thus, this comparison showed the freestream gust model could be used to model periodic velocity perturbations, but the resulting solution is highly dependent on gust discretization and introduces additional frequency content into the solution.

The panel code solution compared favorably to the Kussner solution for a sharp-edge gust. This favorable comparison was expected since the Kussner solution represents a Fourier integral of the Sears solution; however, the Kussner solution showed better agreement with the freestream gust model than anticipated. Finally, the panel code solution predicted a flutter boundary within a 3 percent difference of the flutter boundary predicted using the p-k method. This verified that the coupled aerodynamic-structural model could predict airfoil structural response to self-induced aerodynamic forcing.

7.2 Extension

Having demonstrated the favorable ability of the panel code to model classic unsteady aerodynamic problems, the freestream gust model and structural model were coupled to demonstrate panel code forced-response prediction capabilities. No conclusions were drawn from these forced response simulations with regard to aerodynamic damping or system stability. However, the simulations demonstrate the panel code can model forced structural response due to periodic aerodynamic excitation.

7.3 Discussion and Recommendations

The panel code solution shows greatest dependence on freestream gust model parameters and discretization of a continuous gust. As such, it is recommended that alternate gust discretization methods be investigated prior to the application of the developed panel method to aerodynamic damping investigations of interest. In addition, the panel code solution is only valid for flowfields where incompressible, inviscid assumptions hold true. This limits the panel code application to low Mach numbers, small pitch and plunge perturbations, and low reduced frequencies. If necessary, various techniques could be implemented in the code to expand the applicability of the solution, such as compressibility correction factors and flow separation models. Such corrections are well established and may be easily implemented and validated.

7.4 Contributions of Present Work

Many investigations into aeromechanical response decouple the aerodynamic and structural models by calculating aerodynamic forcing on a stationary airfoil, which is then used to determine structural response using assumed modes or an equivalent technique. This does not account for aerodynamic forcing due to structural response. Time-accurate solutions coupling the aerodynamic and structural models are possible using a finite element approach, but can be computationally expensive for realistic configurations.

The developed panel code couples the aerodynamic and structural solutions to determine time-accurate structural response accounting for both freestream and self-induced aerodynamic forcing. The panel code provides a first order solution, limited only by the potential flow assumptions, which can be used to systematically study aeromechanical response and determine directions for further study.

BIBLIOGRAPHY

[1] Anderson, J. *Fundamentals of Aerodynamics*, 1 ed. McGraw Hill, New York, 1984.

[2] Ardonceau, P. L. Unsteady pressure distribution over a pitching airfoil. *AIAA Journal 27* (1989), 660–662.

[3] Basu, B. C., and Hancock, G. J. The unsteady motion of a two-dimensional aerofoil in incompressible inviscid flow. *Journal of Fluid Mechanics 87* (1978), 159–68.

[4] Blisplinghoff, R. L., Ashley, H., and Halfman, R. L. *Aeroelasticity*, 2 ed. Dover, New York, 1996.

[5] Hess, J. L., and Smith, A. M. O. Calculation of potential flow about arbitrary bodies. In *Progress in Aeronautical Science*. Pergamon Press, New York, 1966, pp. 1–138.

[6] Hodges, Dewey H, and Pierce, G. Alvin. *Introduction to Structural Dynamics and Aeroelasticity*, 1 ed. Cambridge, New York, 2002.

[7] Hugo, R., and Jumper, E. Controling unsteady lift using unsteady trailing-edge flap motions. *AIAA Paper 92-0275* (Jan 1992).

[8] Katz, J., and Plotkin, A. *Low-Speed Aerodynamics*, 2 ed. Cambridge University Press, New York, 2001.

[9] Rennie, R. M., and J., Jumper E. Experimental measurements of dynamic control surface effectiveness. *Journal of Aircraft 33* (1996), 880–887.

[10] Theodorsen, Theodore. General theory of aerodynamic instability and the mechanism of flutter. Tech. Rep. NACA Report 496, NACA, 1935.

[11] Theodorsen, Theodore, and Garrick, I E. Mechanism of flutter.a theoretical and experimental investigation of the flutter problem. Tech. Rep. NACA Report 685, NACA, 1940.

APPENDIX A

UVPM

A.1 Revision 120

```
      program uvpm
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c-----------------------------------------------------------------------
c     rev 120
c        -  based on rev 109
c           Note,
c              foille = forward most node
c              fv_tan = stagnation point node
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c-----------------------------------------------------------------------
c  Precompiler Definitions (uses fpp)
c-----------------------------------------------------------------------

c-----------------------------------------------------------------------
c  Declare Variables
c-----------------------------------------------------------------------

      IMPLICIT NONE
c     -----------------------------------------------------------------
c     Common Blocks / Included Files
c     -----------------------------------------------------------------

         include 'lengths.inc'
c      parameter( lpanel = 300, liter = 6500, lphi = 101 , lfree = 1001)
         include 'airfoil.inc'
         include 'calc.inc'
         include 'const.inc'
         include 'debug.inc'
         include 'file.inc'
         include 'forces.inc'
         include 'freeresp.inc'
         include 'freevort.inc'
         include 'gau.inc'
         include 'graph.inc'
         include 'iterative.inc'
         include 'motion.inc'
         include 'param.inc'
         include 'phi.inc'
         include 'relax.inc'
         include 'strengths.inc'
         include 'velocities.inc'
         include 'wake.inc'
         include 'wakepannel.inc'
c     -----------------------------------------------------------------
c     - Function Parameters
         integer  namlen
         integer  rootlen
c     - Iteration Variables
         integer i
         integer j
```

```
        integer k
c     - Steady State Flag
        integer  sstate
c     - Graphics Variables
        INTEGER pgopen
        INTEGER istat(10)
        REAL xmin, xmax, ymin, ymax
        REAL fxmin, fxmax, fymin, fymax
        REAL xmina, xmaxa, ymina, ymaxa
      INTEGER just, axis
      CHARACTER*70 title
      REAL pgscale
      REAL xtemp(liter)
      REAL ytemp(liter)
      REAL ztemp(liter)
      INTEGER ltemp
c     - Other
        INTEGER  Kutta
        Integer  iter

        REAL*8   local_nor, local_tan
c     - Gauss Solver
        REAL*8 rhs
c     - Influence variables
c     - Velocity components due to rotation
        REAL*8 ru(lpanel)          !  x component of velocity at panel
                                   !  midpoint due to airfoil rotation
        REAL*8 rv(lpanel)          !  y component of velocity at panel
                                   !  midpoint due to airfoil rotation
c     - Velocity components due to free stream
        REAL*8 fsu                 !  x component of velocity at panel
                                   !  midpoint due to freestream
        REAL*8 fsv                 !  y component of velocity at panel
                                   !  midpoint due to freestream
c     - Summation variables used in influence calculations
        REAL*8 tsum                !  summation of tangential velocity
                                   !  component
        REAL*8 usum                !  summation of x dir velocity
                                   !  component
        REAL*8 vsum                !  summation of y dir velocity
                                   !  component
c     - Velocity components due to vortex panels on the airfoil
        REAL*8 vu(lpanel)          !  x vel influence coefficient at
                                   !  panel i due to a unit strength
                                   !  vortex at panel j
        REAL*8 vv(lpanel)          !  y vel influence coefficient at
                                   !  panel i due to a unit strength
                                   !  vortex at panel j
c     - Velocity components due to vortex sheets in the freestream
        REAL*8 fsvu(lpanel)        !  x dir component of velocity at
                                   !  panel midpoint due to vortex
                                   !  sheet (gust)
        REAL*8 fsvv(lpanel)        !  y dir component of velocity at
                                   !  panel midpoint due to vortex
                                   !  sheet (gust)
```

```
c     - Velocity components due to source panel influence
        REAL*8 su(lpanel,lpanel)   !  unit strength source influence
                                   !  coefficient at panel i due to
                                   !  panel j (x dir component)
        REAL*8 sv(lpanel,lpanel)   !  unit strength source influence
                                   !  coefficient at panel i due to
                                   !  panel j (y dir component)

c     - Summation variables used in calculation of airfoil vortex panel
c       influence and vortex sheet influence
        REAL*8 wvu(lpanel)         !  u vel influence coefficient at
                                   !  panel i due to a unit strength
                                   !  Wake vortex
        REAL*8 wvv(lpanel)         !  v vel influence coefficient at
                                   !  panel i due to a unit strength
                                   !  Wake vortex

c     - Wake Panel Coef
        REAL*8 wpu(lpanel)         !  u vel influence coefficient at
                                   !  panel i due to a unit strength
                                   !  vortex at the wake panel
        REAL*8 wpv(lpanel)         !  v vel influence coefficient at
                                   !  panel i due to a unit strength
                                   !  vortex at the wake panel

c     - Calculation variables used in wake panel calc, and graphics
        REAL*8 u1
        REAL*8 u2
        REAL*8 v1
        REAL*8 v2

c     - Working Variables
        REAL*8 dist                !  length of a panel
        REAL*8 dx                  !  x-length of a panel
        REAL*8 dy                  !  y-length of a panel

c     -
        real*8 theta

c     - Phi Integration
        REAL*8 intgrl(0:liter,lpanel) !  Potential at the Leading Edge
        REAL*8 phi_le
        REAL*8 phi_temp1
        REAL*8 phi_temp2

        REAL*8 vsqare

c     - Forces
        REAL*8 casum               !  Axial Pressure
        REAL*8 cnsum               !  Normal Pressure
        REAL*8 cmsum               !  Moment Coefficient
        REAL*8 dxmom               !  Used to calculate foil moment
        REAL*8 dymom               !  Used to calculate foil moment
        REAL*8 xmidmom             !  Used to calculate foil moment
        REAL*8 ymidmom             !  Used to calculate foil moment

c     - Vortex Sheet Parameters
        real*8 xfsv(10,2)
        real*8 yfsv(10,2)
        real*8 gfsv(10,2)
        integer  nfsv_t
        real*8 temp_cos, temp_sin
        real*8 travel
        real*8 time                !  Used in sheet splitting

        REAL*8 dxj
        REAL*8 dxjp
```

```
         REAL*8 dyj
         REAL*8 dyjp
         real*8 mj
         real*8 mjp
c     - Time
         real etime
         real elapsed
         real extime(2)

c     ------------------------------------------------------------------
c     Namelists
c     ------------------------------------------------------------------

      namelist /vpm_in/    f_foil, x0, y0, f_responce, mu, k_a, k_h,
     & r_a, x_a, f_mot, f_vort, idump1, idump2, debug,
     & debug_wake, i_debug, relax_gammaw, relax_delk, relax_thetk

      namelist /graph/     graphics, savegif,
     & zm_field, zm_field_x, zm_field_y

c      namelist /testing/   test_fs_split, test_fs_inf, fs_inf_scale,
c     & test_fs_inf_pause

      namelist /phi_int/   npi, x_far, y_far

      namelist /init/      sstate
c     ------------------------------------------------------------------
c     Format Statements
c     ------------------------------------------------------------------

      include 'format.inc'
c-----------------------------------------------------------------------
c  Start Program Runtime
c-----------------------------------------------------------------------

      write(6,*)'Starting UVPM'
c-----------------------------------------------------------------------
c  -  Program Initialization:
c-----------------------------------------------------------------------

      write(6,*)'Start Program Initalization'
!-----------------------------------------------------------------------
!----- Initialize Graphics
      if (graphics.eq.1) then
         !- Window 1 -  Airfoil
            istat(1) = pgopen('/xserve')
            if (istat(1).le.0) stop
            call PGASK(.false.)
!         !- Window 2 -  CP
!            istat(2) = pgopen('/xserve')
!            if (istat(2).le.0) stop
!            call PGASK(.false.)
!         !- Window 3 -  Forces
!            istat(3) = pgopen('/xserve')
!            if (istat(3).le.0) stop
!            call PGASK(.false.)
      end if
c     ------------------------------------------------------------------
c     -  HARD-CODED PARAMETERS
c     ------------------------------------------------------------------

c     -  Convergence Criteria
         dkcon =  1D-6!0.0001   !  Convergence Criteria
         tkcon =  1D-6!0.0001   !  Convergence Criteria
```

```
      gwcon =  1D-6!0.0001   !  Convergence Criteria
c     -  Wake Grouping - Currently Disabled
         ngv = 10          !  Number of vortices to group past grouping
                           !   distance
         nwv = 0           !
         vgd = 50.         !  Chord Lengths Down Stream to start
                           !  Grouping Vortices
c     -  Constants
         pi          = 4.D0*datan(1.0D0)  ! 3.1416
         pi2inv      = 1.D0/(2.D0*pi)     ! 0.3183
         rho_fluid   = 0.002377D0         ! slug/ft^3
         chord       = 1.0D0
         alphafs     = 0.0D0
c     -  Initializations
         omega = 0.D0
         hdot = 0.D0
c     ------------------------------------------------------------------
c     -  Get Configuration File from Command Line Input
c     ------------------------------------------------------------------
         call getarg(1,f_config)
         if (f_config(1:1).eq.' ') then
            write(*,*) 'INPUT FILE NOT SPECIFIED. PROGRAM STOPPED.'
            stop
         end if
         len_config = namlen(f_config)
         write(6,*)'-  Input File is >"',f_config(1:len_config),'"'
         call getarg(2,fn)
         if (fn(1:1).eq.' ') then
            sstate = 0
         else if (fn(1:1).eq.'1') then
            sstate = 1
         end if
c     ------------------------------------------------------------------
c     -  File Unit Definitions
c     ------------------------------------------------------------------
         i_airfoil   = 30     !  Output - Store Airfoil Node Locations
         i_foil      = 40     !  Input  - Read In Airfoil Coords (UIUC)
                              !   Specified in i_config
         i_force     = 48     !  Output - Calculated Force and Moments
         i_config    = 34     !  Input  - Configuration File (vpm_in.dat)
         i_mot       = 22     !  Input  - Motion History File Defined in
                              !   i_config
         i_readme    = 42     !  Output - Stores Relevant Run Information
         i_tan       = 24     !  Output - Tangential Velocity
         i_vortex    = 33     !  Output -
         i_elements  = 47     !  Output - Element Locations and Strengths,
                              !   Source sheet, Vortex sheet, point vortex
         i_ani       = 49     !  Data output
         i_temp      = 50     !  Temp debug
         i_pressure  = 51     !  pressure
c     ------------------------------------------------------------------
c     -  Read Input Files
c     ------------------------------------------------------------------
      write(6,*)'- Read Input Files'
c     -  Main Config File Specified on the Command Line
         write(6,*)'-  Start Main Config File'
!         call read_input()
```

```
         len_config = namlen(f_config)
         fn = f_config(1:len_config)
         open(Unit=i_config,File=fn,Status='unknown')
            read(i_config,nml=vpm_in)
            read(i_config,nml=graph)
c            read(i_config,nml=testing)
            read(i_config,nml=phi_int)
         close(unit=i_config)
         write(6,*)'-  Finish Main Config File'

c     -  Read Motion Files
         write(6,*)'-  Start Motion History'
         write(6,*)'"',f_mot,'"'
         call read_motion()
         write(6,*)'-  Finish Motion History'

!c       -  Plot Motion History
!            call plot_motion(istat(1))
!            pause

c     -  Read Airfoil
         write(6,*)'-  Start Airfoil Coordinates'
         write(6,*)'"',f_foil,'"'
         call read_foil()
         write(6,*)'-  Finish Airfoil Coordinates'

!c       -  Plot Airfoil
!            call plot_airfoil(istat(1),1.E0,0)
!            pause

c     -  Read Vortex Locations
         if (f_vort.ne.'none') then
            write(6,*)'-  Start Vortex Locations'
            write(6,*)'"',f_vort,'"'
            call read_freevort()
            write(6,*)'-  Finish Vortex Locations'
         end if

!c       -  Plot Airfoil
!            call plot_airfoil(istat(1),1.E0,0)
!            pause

c     ---------------------------------------------------------------
c     -  Initialize Output Files
c     ---------------------------------------------------------------
c     -  Initalize Lift File
         write(6,*)'-  Open Force Output'
!         write(fn,'("out\force.out")')

         len_config = rootlen(f_config)
         if (len_config.gt.0) then
            fn=f_config(1:len_config)//'.lft'
          else
            write(fn,'("out\vpm_in.lft")')
         end if
         write(*,*)'"',fn,'"'

         OPEN(UNIT=i_force,FILE=fn,STATUS='unknown')

c     -  Initialize Pressure File
         len_config = rootlen(f_config)
         if (len_config.gt.0) then
            fn=f_config(1:len_config)//'.cp'
          else
            write(fn,'("out\vpm_in.cp")')
```

```
         end if
         write(*,*)'"',fn,'"'
         OPEN(UNIT=i_pressure,FILE=fn,STATUS='unknown')
c     -  Initalize Animation File
         write(6,*)'-  Open Animation Output'
c         write(fn,'("out\data.out")')
         len_config = rootlen(f_config)
         if (len_config.gt.0) then
            fn=f_config(1:len_config)//'.ani'
          else
            write(fn,'("out\vpm_in.ani")')
         end if
         write(*,*)'"',fn,'"'
         OPEN(UNIT=i_ani,FILE=fn,STATUS='unknown')
            write(i_ani,*)nodtot
            write(i_ani,*)nodle
            write(i_ani,*)dt
            write(i_ani,*)nstep
            write(i_ani,*)idump1
            write(i_ani,*)idump2
            write(i_ani,*)x0
            write(i_ani,*)y0

         len_config = rootlen(f_config)
         if (len_config.gt.0) then
            fn=f_config(1:len_config)//'.tmp'
          else
            write(fn,'("out\vpm_in.lft")')
         end if
         write(*,*)'"',fn,'"'
         OPEN(UNIT=i_temp,FILE=fn,STATUS='unknown')

c     ------------------------------------------------------------------
c     -  Set Initial Conditions Based on Mothis
c     ------------------------------------------------------------------
      write(6,*)'-  Set Initial Conditions'
c  REV 100
!         alpha(0)    =  alpha(1)
         cd_s(0)     =  0.
         cl_s(0)     =  0.
         clp         =  0.0
         cmeap       =  0.0
         cmle_s(0)   =  0.
         cmo_s(0)    =  0.
         gamma(0)    =  0.
         gammaw(0)   =  0.
         t           =  0
         t_s(0)      =  0.*dt
         thetk1      =  thetk
         cd          =  0.
         cl          =  0.
         clp         =  0.0
         cmle        =  0.
!         cmo         =  0.
         delk        =  dt
         thetk       =  0.0
         eta_a       =  0D0
```

```
         eta_h        =  0D0
         ufre         =  1D0
         rho_fluid    =  2.377D-3


c-----------------------------------------------------------------------
c  End Program Initalization
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------

c-----------------------------------------------------------------------
c  BEGIN THE TIME STEPPING
c-----------------------------------------------------------------------
      write(6,*)'Start Time Stepping: ',nstep,' Iterations'
c  ---------------------------------------------------------------------
c     -  Start Time Iteration
c  ---------------------------------------------------------------------
      do 1000 t=0,nstep
!      do 1000 t=0,1
c     ------------------------------------------------------------------
c     - ROTATE THE AIRFOIL AND FIND PANEL MIDPOINTS, ANGLES
c     ------------------------------------------------------------------
!         call plot_airfoil(istat(1),1.E0,0)

         call rotate_foil()

!         call plot_airfoil(istat(1),1.E0,0)
c     ------------------------------------------------------------------
c     -  Evaluate Parameters That Change For Each Time Step But Not Each
c        Iteration
c
c         The Freestream, Source Panels (Airfoil), Vortex Panels (Airfoil),
c         Vortex Panels (Wake), Discrete Vortices, Airfoil Perimeter
c         Panel Midpoint Velocities Due To Rotation
c     ------------------------------------------------------------------

      do 220 i=1,nodtot
c     -  Find Airfoil Perimeter
!         dx=x(i+1)-x(i)
!         dy=y(i+1)-y(i)
!         dist=sqrt(dx*dx+dy*dy)
!         perim=perim+dist
c     ------------------------------------------------------------------
c     -  Freestream Influence on i'th panel:

         ! Global Frame (Eularian)
           fsu= uexp
           fsv= 0.
         ! i'th Panel Frame (Lagrangian)
!            fst(i)= costhe(i)*fsu +  sinthe(i)*fsv
!            fsn(i)=-sinthe(i)*fsu +  costhe(i)*fsv
c            pst(i)=(xmid(i)-xphi)*fsu + (ymid(i)-yphi)*fsv
c     ------------------------------------------------------------------
c     -  Influence Due To Rotation on i'th panel
C        Evaluate Panel Midpoint Velocities Due To Rotation

         ru(i)=-(xmid(i)-xmidp(i))/dt
         rv(i)=-(ymid(i)-ymidp(i))/dt
!         rt(i)=  costhe(i)*ru(i) + sinthe(i)*rv(i)
!         rn(i)= -sinthe(i)*ru(i) + costhe(i)*rv(i)
c     ------------------------------------------------------------------
c     -  Airfoil Influence:
C        Find Contribution Of j'th Panel Due To Source Panels And
C        Vortex Panels On The i'th Panel
```

```
!         nsum=0.
!         tsum=0.
        usum=0.
        vsum=0.

        do 200 j=1,nodtot

           call calc_panel_inf(x(j),y(j),x(j+1),y(j+1),xmid(i),ymid(i),
    & costhe(j),sinthe(j))

!       -  Unit Source Normal on panel i due to Panel j (nxn Array)
            su(i,j)= vel(1)
            sv(i,j)= vel(2)
!            st(i,j)= vel(1)* costhe(i) + vel(2)* sinthe(i)
!            sn(i,j)= vel(1)*-sinthe(i) + vel(2)* costhe(i)

!       -  Unit Vortex Normal on panel i due to all other panels
           usum =  usum + vel(3)
           vsum =  vsum + vel(4)
!           tsum =  tsum + vel(3)* costhe(i) + vel(4)* sinthe(i)
!           nsum =  nsum + vel(3)*-sinthe(i) + vel(4)* costhe(i)

 200    continue

!        vn(i)=nsum
!        vt(i)=tsum
        vu(i)=usum
        vv(i)=vsum
c     ------------------------------------------------------------------
c     -  Wake Influence on the i'th Panel: DISCRETE VORTICES

!        nsum=0.
!        tsum=0.
        usum=0.
        vsum=0.

        do 210 j=1,nwv

           call calc_pt_inf(xvort(j),yvort(j),xmid(i),ymid(i),vort(j))

           usum = usum + vel(1)
           vsum = vsum + vel(2)
!           tsum = tsum + vel(1)* costhe(i) + vel(2)* sinthe(i)
!           nsum = nsum + vel(1)*-sinthe(i) + vel(2)* costhe(i)

 210    continue

!        wvn(i)=nsum
!        wvt(i)=tsum
        wvu(i)=usum
        wvv(i)=vsum

c     ------------------------------------------------------------------
c     -  Freestream Vortex Influence on the i'th panel: SHEET VORTICES

!        nsum=0.
!        tsum=0.
        usum=0.
        vsum=0.

        do 215 j=1,nfsv

                 ! check if vortex ends on a panel
                 if ((fsvort(j,7).ne.0).or.(fsvort(j,6).ne.0)) then
                    nfsv_t = 2  ! split panel
                  else
                    nfsv_t = 1  ! single panel
                 end if

                 ! for each sub-vortex (k) (if split)
                 do k = 1,nfsv_t
```

```
                    ! create sub-vortex
                    ! Vortex Start Point
                    if (k.eq.1) then
                       if ( fsvort(j,6) .ne. 0 ) then
!                            xfsv(k,1) = x(fsvort(j,6)+1)
!                            yfsv(k,1) = y(fsvort(j,6)+1)
                          xfsv(k,1) = x(fsvort(j,6))
                          yfsv(k,1) = y(fsvort(j,6))
                        else
                          xfsv(k,1) = fsvort(j,1)
                          yfsv(k,1) = fsvort(j,2)
                       end if
                    else if (k.eq.2) then
                       if ( fsvort(j,6) .ne. 0 ) then
!                            xfsv(k,1) = x(fsvort(j,6))
!                            yfsv(k,1) = y(fsvort(j,6))
                          if (fsvort(j,6) .gt. (1)) then
                             xfsv(k,1) = x(fsvort(j,6)-1)
                             yfsv(k,1) = y(fsvort(j,6)-1)
                           else
                             xfsv(k,1) = x(nodtot+2)
                             yfsv(k,1) = y(nodtot+2)
                          end if
!                            xfsv(k,1) = x(fsvort(j,6)-1)
!                            yfsv(k,1) = y(fsvort(j,6)-1)
                        else
                          xfsv(k,1) = fsvort(j,1)
                          yfsv(k,1) = fsvort(j,2)
                       end if
                    end if
                    ! Vortex End Point
                    if (k.eq.1) then
                       if ( fsvort(j,7) .ne. 0 ) then
                          xfsv(k,2) = x(fsvort(j,7))
                          yfsv(k,2) = y(fsvort(j,7))
                        else
                          xfsv(k,2) = fsvort(j,3)
                          yfsv(k,2) = fsvort(j,4)
                       end if
                    else if (k.eq.2) then
                       if ( fsvort(j,7) .ne. 0 ) then
                          xfsv(k,2) = x(fsvort(j,7)+1)
                          yfsv(k,2) = y(fsvort(j,7)+1)
                        else
                          xfsv(k,2) = fsvort(j,3)
                          yfsv(k,2) = fsvort(j,4)
                       end if
                    end if
                 end do ! (k)
                 ! find length of surface panel
                 if ( fsvort(j,7) .ne. 0 ) then
                    ! ends on panel
                    dist = sqrt((x(fsvort(j,7))-x(fsvort(j,7)+1))**2+
     &                (y(fsvort(j,7))-y(fsvort(j,7)+1))**2 )
                  else if ( fsvort(j,6) .ne. 0 ) then
                    ! starts on panel
!                     dist = sqrt((x(fsvort(j,6)-1)-x(fsvort(j,6)))**2+
```

```
!     &                (y(fsvort(j,6)-1)-y(fsvort(j,6)))**2 )
                      if (fsvort(j,6) .gt. (1)) then
                         dx = x(fsvort(j,6)-1) - x(fsvort(j,6))
                         dy = y(fsvort(j,6)-1) - y(fsvort(j,6))
                         dist = sqrt( (dx)**2 + (dy)**2 )
                       else
                         dx = x(nodtot+2) - x(fsvort(j,6))
                         dy = y(nodtot+2) - y(fsvort(j,6))
                         dist = sqrt( (dx)**2 + (dy)**2 )
                      end if
                   end if
                   ! for each sub-vortex
                   do k = 1,nfsv_t
                      if ( fsvort(j,7) .ne. 0 ) then
                         ! if split panel by end
                         ! distance from end of vortex to end of subpanel
                         gfsv(k,2) = sqrt( (xfsv(k,2) - fsvort(j,3))**2 +
     &                    (yfsv(k,2) - fsvort(j,4))**2 ) / dist
                         ! calc total gamma for sub-vortex
                         gfsv(k,1) = fsvort(j,5) * (1 - gfsv(k,2))
                       else if ( fsvort(j,6) .ne. 0 ) then
                         ! if split panel by Start
                         ! distance from end of vortex to end of subpanel
                         gfsv(k,2) = sqrt( (xfsv(k,1) - fsvort(j,1))**2 +
     &                    (yfsv(k,1) - fsvort(j,2))**2 ) / dist
                         ! calc total gamma for sub-vortex
                         gfsv(k,1) = fsvort(j,5) * (1 - gfsv(k,2))
                       else
                         ! calc total gamma for sub-vortex
                         gfsv(k,1) = fsvort(j,5)
                      end if
                   end do ! (k)
                   ! for each sub-vortex
                   do k = 1,nfsv_t
                      ! length of sub-vortex
                      dist = sqrt( (xfsv(k,2) - xfsv(k,1))**2 +
     &                 (yfsv(k,2) - yfsv(k,1))**2 )
                      temp_cos = (xfsv(k,2) - xfsv(k,1)) / dist
                      temp_sin = (yfsv(k,2) - yfsv(k,1)) / dist
                      !- Find Influence of sub-vortex (k) of vortex (j)
                      !  on panel (i)
                      call calc_panel_inf(xfsv(k,1),yfsv(k,1),
     &                  xfsv(k,2),yfsv(k,2),xmid(i),ymid(i),temp_cos,
     &                  temp_sin)
                      ! Calculate Velocities at midpoints
                         usum =  usum + vel(3) * gfsv(k,1) / dist
                         vsum =  vsum + vel(4) * gfsv(k,1) / dist
                   end do !(k) sub-vortex
  215   continue
            fsvu(i)= usum
            fsvv(i)= vsum
c     ------------------------------------------------------------------
```

```
c     END FSVORT
c     ----------------------------------------------------------------
  220 continue
c     ----------------------------------------------------------------
c     - Iterate to find Wake Panel Location based on non-changing
c       parameters
c        - Seed gamma and gammaw to help convergence
            if (t.ge.1) then
               gamma(t)=gamma(t-1)
               gammaw(t)=gammaw(t-1)
            else
               gamma(0) = 0.0D0
               gammaw(0) = 0.0D0
            end if
c        add thetk calculation based on angle of TE panels
         iter=0
  230    continue
c        - Increment iter
            iter = iter + 1
            if (iter.gt.400) then
               write(*,*)'Wake Panel Iterations Exceeded 400'
               stop
            end if
c        - Position wake panel based on new delk and thetk
            x(nodtot+2)=x(1)+cos(thetk)*delk
            y(nodtot+2)=y(1)+sin(thetk)*delk
c        - Find Midpoint of wake panel
            xmid(nodtot+1)=.5*(x(nodtot+1)+x(nodtot+2))
            ymid(nodtot+1)=.5*(y(nodtot+1)+y(nodtot+2))
c        - Store delk and thetk for comparison
            delk1    =  delk
            thetk1   =  thetk
            gammaw1  =  gammaw(t)
c        - EVALUATE INFLUENCE COEFFICIENT AT PANEL I DUE TO WAKE PANEL
            j=nodtot+1
            do 240 i=1,nodtot
               call calc_panel_inf(x(j),y(j),x(j+1),y(j+1),xmid(i),
     &               ymid(i),cos(thetk),sin(thetk))
               wpu(i) = vel(3)
               wpv(i) = vel(4)
!               wpt(i) = vel(3)* costhe(i) + vel(4)* sinthe(i)
!               wpn(i) = vel(3)*-sinthe(i) + vel(4)* costhe(i)
  240       continue
c     ----------------------------------------------------------------
c        - ASSEMBLE MATRIX FOR THE GAUSS SOLVER
      if ((sstate.eq.0).or.(t.ge.1)) then
         !-------------------------------------------------------------
         ! Time Dependent
            do 260 i=1,nodtot
            ! rhs = rotation, fs, bound vortex, wake vortex, wake panel,
            !       Freestream Vortex
               rhs = -local_nor(ru(i),rv(i),costhe(i),sinthe(i))
     &               -local_nor(fsu,fsv,costhe(i),sinthe(i))
     &               -local_nor(vu(i),vv(i),costhe(i),sinthe(i))*
     &               gamma(t)
```

```
     &                 -local_nor(wvu(i),wvv(i),costhe(i),sinthe(i))
     &                 -local_nor(wpu(i),wpv(i),costhe(i),sinthe(i))*
     &                 gammaw(t)
     &                 -local_nor(fsvu(i),fsvv(i),costhe(i),sinthe(i))
            do 250 j=1,nodtot
               a(i,j)=local_nor(su(i,j),sv(i,j),costhe(i),sinthe(i))
  250       continue
            a(i,nodtot+1)=rhs
  260    continue
      else
         !---------------------------------------------------------------
         !  Steady State
         do 265 i=1,nodtot
            rhs = -local_nor(fsu,fsv,costhe(i),sinthe(i))
            do 255 j=1,nodtot
               ! Source
               a(i,j)=local_nor(su(i,j),sv(i,j),costhe(i),sinthe(i))
  255       continue
            ! Gamma
            a(i,nodtot+1)=local_nor(vu(i),vv(i),costhe(i),sinthe(i))
            ! RHS
            a(i,nodtot+2)=RHS
!            a(i,nodtot+2)=-fsn(i)
  265    continue
            i = nodtot+1
            do 275 j=1,nodtot
               ! Source Kutta
               a(i,j)=local_tan(su(1,j),sv(1,j),costhe(1),sinthe(1))
     &                 +local_tan(su(nodtot,j),sv(nodtot,j),
     &                  costhe(nodtot),sinthe(nodtot))
  275       continue
            ! Gamma Kutta
            a(i,nodtot+1)=local_tan(vu(1),vv(1),costhe(1),sinthe(1))
     &                 +local_tan(vu(nodtot),vv(nodtot),
     &                  costhe(nodtot),sinthe(nodtot))
            ! Gamma RHS
            a(i,nodtot+2)=0.!-fst(1)-fst(nodtot)
      end if   ! sstate
c     ------------------------------------------------------------------
c        -  CALL THE GAUSSIAN SOLVER
            if ((sstate.eq.0).or.(t.ge.1)) then
               call gauss(1,nodtot)
            else
               call gauss(1,nodtot+1)
            end if
c     ------------------------------------------------------------------
c        -  RETRIEVE SOLUTION
            if ((sstate.eq.0).or.(t.ge.1)) then
               do 270 i=1,nodtot
                  q(i)=a(i,nodtot+1)
  270          continue
            else
               do 285 i=1,nodtot
                  q(i)=a(i,nodtot+2)
c                  write(6,*)' q=',q(i)
  285          continue
            ! Vortex
               gamma(t) = a(nodtot+1,nodtot+2)
               gammaw(t) = 0.0
            end if
```

```
c     ------------------------------------------------------------------
c        -  CALCULATE THE TANGENTIAL VELOCITIES ON PANELS 1 AND NODTOT
           do 290 k=1,2
              i=k
              if(k.eq.2) i=nodtot
              ! Tangential/Normal Velocity Due to Airfoil Panels
                 tsum=0.
                 usum=0.
                 vsum=0.
              ! - Source
                 do 280 j=1,nodtot
                    tsum = tsum + local_tan(su(i,j),sv(i,j),costhe(i)
     &               ,sinthe(i))*q(j)
                    usum = usum + su(i,j)*q(j)
                    vsum = vsum + sv(i,j)*q(j)
  280            continue
              ! - Sum Velocity Components
              vtan(i) = local_tan(ru(i),rv(i),costhe(i),sinthe(i))
     &               +local_tan(fsu,fsv,costhe(i),sinthe(i))
     &               +local_tan(vu(i),vv(i),costhe(i),sinthe(i))*
     &               gamma(t)
     &               +local_tan(wvu(i),wvv(i),costhe(i),sinthe(i))
     &               +local_tan(wpu(i),wpv(i),costhe(i),sinthe(i))*
     &               gammaw(t)
     &               +local_tan(fsvu(i),fsvv(i),costhe(i),sinthe(i))
     &               +tsum
                 if (i.eq.1) then
                    u1=fsu+usum+vu(i)*gamma(t)+wpu(i)*gammaw(t)+
     &               wvu(i)+fsvu(i)
                    v1=fsv+vsum+vv(i)*gamma(t)+wpv(i)*gammaw(t)+
     &               wvv(i)+fsvv(i)
                 else
                    u2=fsu+usum+vu(i)*gamma(t)+wpu(i)*gammaw(t)+
     &               wvu(i)+fsvu(i)
                    v2=fsv+vsum+vv(i)*gamma(t)+wpv(i)*gammaw(t)+
     &               wvv(i)+fsvv(i)
                 end if
  290      continue
c     ------------------------------------------------------------------
c        -  SOLVE FOR THE VELOCITY AT THE MIDPOINT OF THE WAKE PANEL
c     ------------------------------------------------------------------
c        -  EVALUATE STRENGTH OF VORTICITY ON THE WAKE PANEL
        if ((sstate.eq.0).or.(t.ge.1)) then
           gammaw(t)=dt/(2.D0*delk)*(vtan(nodtot)**2.D0-vtan(1)**2.D0)
           gamma(t)=gamma(t-1)-gammaw(t)*delk/perim
        end if ! (t.ge.1)
c     ------------------------------------------------------------------
c        -  SOLVE FOR DELK AND THETK
c          - Basu and Hancock Method
c             Basu, B. C. and Hancock, G. J.,
c             "The Unsteady Motion of a Two-Dimensional Aerofoil in
c             Incompressible Inviscid Flow," Journal of Fluid Mechanics,
c             Vol. 87, 1978, pp. 159-168.
!             thetk=atan2(vwp,uwp)
!             delk=sqrt(uwp**2+vwp**2)*dt
```

```
c           - Ardonceau's Method
c              Ardonceau, Pascal L.,
c              "Unsteady Pressure Distribution Over A Pitching Airfoil,"
c              AIAA Journal, Vol. 27, 1989, pp. 660-662.
            dx=(u1+u2)*5.D-1*dt
            dy=(v1+v2)*5.D-1*dt
            delk  = delk  + relax_delk  * (sqrt(dx**2+dy**2) - delk)
            dx = x(1) - x(2)
            dy = y(1) - y(2)
            dist = dy / dx
            dx = x(1) - x(nodtot)
            dy = y(1) - y(nodtot)
            dist = (dist + dy / dx) / 2.D0
            thetk = thetk + relax_thetk * (datan2(dy,dx) - thetk)
            thetk = datan2(dist,1.D0)
c     ----------------------------------------------------------------
c     print Debug Data for convergence
c     - r91
      if (debug.ne.0) then
         if (iter.le.1) then
            write(6,49)
            write(6,49)'iter','delk','thetk','gamma','gammaw',
     &       'd_gammaw','conv','u1','u2','v1','v2','vtan(1)',
     &       'vtan(nodtot)'
         endif
         write(6,50)real(iter),delk,thetk,gamma(t),gammaw(t),
     &    abs(gammaw(t)-gammaw1), gwcon, u1, u2, v1, v2,vtan(1),
     &    vtan(nodtot)
      endif
c     ----------------------------------------------------------------
c     -  Check for Convergence of delk and thetk
!         if(abs(thetk-thetk1).lt.tkcon.and.abs(delk-delk1).lt.dkcon)then
         if ((sstate.eq.0).or.(t.ge.1)) then
            if( abs(gammaw(t)-gammaw1)  .lt.  gwcon )then
               go to 320   ! Exit
            else
               go to 230   ! Re-iterate
            end if
         end if ! (t.ge.1)
c     ----------------------------------------------------------------
c     -  CONVERGENCE ACHIEVED
  320 continue
c     - r91
      if (debug.ne.0) then
         write(6,49)
      endif
c---------------------------------------------------------------------
c  - Pressure and Force Calculation
c---------------------------------------------------------------------
c     -  Place Phi Integration points
         if (debug.ne.0) then
            write(i_debug,*)'  - Place Integration Points'
         endif
c     ----------------------------------------------------------------
c     -  Sine Placement from x_far to LE
         dx=x(nodle)-x_far
         dy=y(nodle)-y_far
```

```
        do 325 i=1,npi
           theta = dreal(i-1)/dreal(npi) * pi/2.0D0
           xpi(i) = dsin(theta)*dx + x_far;
           ypi(i) = dsin(theta)*dy + y_far;
 325    continue
c-----------------------------------------------------------------------
c  CALCULATION OF TANGENTIAL VELOCITY ON EACH PANEL DUE TO
c  SOURCE PANELS, VORTEX PANELS, WAKE PANEL, DISCRETE VORTICES
c
      if (debug.ne.0) then
         write(i_debug,*)'  - Calc Tangential Velocity on each Panel'
      endif

      do 360 i=1,nodtot

      ! Tangential/Normal Velocity Due to Airfoil Panels
         tsum=0.
         usum=0.
         vsum=0.

         ! - Source
         do 365 j=1,nodtot
            tsum = tsum + local_tan(su(i,j),sv(i,j),costhe(i)
     &       ,sinthe(i))*q(j)
            usum = usum + su(i,j)*q(j)
            vsum = vsum + sv(i,j)*q(j)
 365    continue

        vtan(i) = local_tan(ru(i),rv(i),costhe(i),sinthe(i))
     &          +local_tan(fsu,fsv,costhe(i),sinthe(i))
     &          +local_tan(vu(i),vv(i),costhe(i),sinthe(i))*
     &          gamma(t)
     &          +local_tan(wvu(i),wvv(i),costhe(i),sinthe(i))
     &          +local_tan(wpu(i),wpv(i),costhe(i),sinthe(i))*
     &          gammaw(t)
     &          +local_tan(fsvu(i),fsvv(i),costhe(i),sinthe(i))
     &          +tsum

        vxdir(i)=fsu+usum+vu(i)*gamma(t)+wpu(i)*gammaw(t)+
     &     wvu(i)+fsvu(i)
        vydir(i)=fsv+vsum+vv(i)*gamma(t)+wpv(i)*gammaw(t)+
     &     wvv(i)+fsvv(i)

        if (vtan(i).lt.0) then
           fv_tan = i+1
        end if

 360  continue

c       Nessisary For Fsvortex Convection Routine
        call calc_pt_vel(xmid(nodtot+1),ymid(nodtot+1),1)

        vtan(nodtot+1) = vel(1)*cos(thetk) + vel(2)*sin(thetk)
c-----------------------------------------------------------------------
c  CALCULATION OF VELOCITY AT INTEGRATING POINTS
c
      do 390 i=1,npi

         call calc_pt_vel(xpi(i),ypi(i),0)

         upi(i) = vel(1)
         vpi(i) = vel(2)

 390  continue
c-----------------------------------------------------------------------
```

```
c  EVALUATE THE VELOCITIES AT THE DISCRETE VORTICES
c
      do 420 i=1,nwv

         call calc_pt_vel(xvort(i),yvort(i),0)

         udv(i) = vel(1)
         vdv(i) = vel(2)

  420 continue
c-----------------------------------------------------------------------
c  EVALUATE THE VELOCITIES AT THE FREE STREAM VORTICES
c
      do 425 i=1,nfsv

         !- Start Point
            call calc_pt_vel(fsvort(i,1),fsvort(i,2),0)

            vfsvort(i,1) = vel(1)
            vfsvort(i,2) = vel(2)

         !- End Point
            call calc_pt_vel(fsvort(i,3),fsvort(i,4),0)

            vfsvort(i,3) = vel(1)
            vfsvort(i,4) = vel(2)

  425 continue
c-----------------------------------------------------------------------
c  EVALUATION OF PHI INTEGRATION UP TO THE LEADING EDGE
      if (debug.ne.0) then
         write(i_debug,*)'  - Evaluate PHI Integration up to Leading Edg
     &e'
      endif
c     -----------------------------------------------------------------
c     INTEGRATE UP TO LEADING EDGE
c     Assumption: phi is zero, or constant for all t at point 1, or the
c                 farfield

         phi_le = 0.
         do i=2,npi
            ! Distance from previous point to point i
               dx=(xpi(i) - xpi(i-1)) * upi(i-1) !(upi(i) + upi(i-1))/2.0
               dy=(ypi(i) - ypi(i-1)) * vpi(i-1) !(vpi(i) + vpi(i-1))/2.0
            ! Calculate phi at point i based on phi at previous point and
            !  velocity at current point
               phi_le = phi_le + dx + dy
         end do

         dx=x(nodle)-xpi(npi)
         dy=y(nodle)-ypi(npi)
         phi_le = phi_le + upi(npi)*dx + vpi(npi)*dy
c     -----------------------------------------------------------------
c     INTEGRATE PHI ALONG THE LOWER SURFACE

         phi_temp1 = phi_le
         do 440 i=nodle-1,1,-1
            ! Find Distance along panel
               dx = x(i) - x(i+1)
               dy = y(i) - y(i+1)
            ! find phi at panel end point
               phi_temp2 = phi_temp1 + vxdir(i)*dx + vydir(i)*dy
            ! phi at midpoint is average of phi at endpoints
               intgrl(t,i)= (phi_temp1 + phi_temp2) / 2.0
            ! swap temp variables
               phi_temp1 = phi_temp2
```

```
  440    continue
c     ---------------------------------------------------------------
c     INTEGRATE PHI ALONG THE UPPER SURFACE
         phi_temp1 = phi_le
         do 450 i=nodle,nodtot,1
            ! Find Distance along panel
               dx = x(i+1) - x(i)
               dy = y(i+1) - y(i)
            ! find phi at panel end point
               phi_temp2 = phi_temp1 + vxdir(i)*dx + vydir(i)*dy
            ! phi at midpoint is average of phi at endpoints
               intgrl(t,i)= (phi_temp1 + phi_temp2) / 2.0
            ! swap temp variables
               phi_temp1 = phi_temp2
  450    continue
c--------------------------------------------------------------------
c  CALCULATE SURFACE PRESSURES
c  (NOTE:  THE EVALUATION OF V.DX MAY BE WRONG AND
c   THE X AND Y COMPONENTS OF VELOCITY ON EACH
c   PANEL MAY NEED TO BE EVALUATED)
      if (debug.ne.0) then
         write(i_debug,*)'  - Calculate Surface Pressure'
      endif
c     ---------------------------------------------------------------
c     Calculate CP around airfoil
      do 460 i = 1,nodtot
c        Calculate Cp panel(i)
            vsqare=vtan(i)**2
      if (sstate.eq.0) then
         if(t.eq.0) then
           cp(i)=1.-vsqare-2.*intgrl(t,i)/dt
         else if(t.eq.1) then
           cp(i)=1.-vsqare-2.*(intgrl(t,i)-intgrl(t-1,i))/dt
         else
           cp(i)=1.-vsqare-2.*(intgrl(t-2,i)+3.*intgrl(t,i)-
     +             4.*intgrl(t-1,i))/dt/2.
         end if
      else
         if(t.eq.0) then
           cp(i)=1.-vsqare !- 2.*intgrl(t,i)/dt
         else if(t.eq.1) then
           cp(i)=1.-vsqare-2.*(intgrl(t,i)-intgrl(t-1,i))/dt
         else
           cp(i)=1.-vsqare-2.*(intgrl(t-2,i)+3.*intgrl(t,i)-
     +             4.*intgrl(t-1,i))/dt/2.
         end if
      end if ! (sstate.eq.0)
  460 continue
c--------------------------------------------------------------------
c  EVALUATE AERODYNAMIC FORCES
      if (debug.ne.0) then
         write(i_debug,*)'  - Calc Aerodynamic Forces'
      endif
      cnsum=0.
      casum=0.
      cmsum=0.
      cl = 0.
      cd = 0.
      cmle = 0.
```

```
      cmea = 0.
      do 475 i=1,nodtot
         !- Cl
            cnsum = -cp(i) * (x(i+1)-x(i))
            cl = cl + cnsum
         !- Cd
            casum = -cp(i) * (y(i+1)-y(i))
            cd = cd + casum
         !- Cm
            cmle=cmle+cnsum*(x(nodle)-xmid(i))-casum*(y(nodle)-ymid(i))
            cmea=cmea+cnsum*(x0      -xmid(i))-casum*(y0      -ymid(i))
  475 continue
c---------------------------------------------------------------------
c---------------------------------------------------------------------
c  Export Data For Each Time Step
      if (debug.ne.0) then
         write(i_debug,*)'  - Export Data'
      endif
c---------------------------------------------------------------------
c  FORCE DATA
c---------------------------------------------------------------------
      write(i_force,58)float(t)*dt*chord/uexp,mothis(t),
     &alpha(t)*180./pi,cl,cd,cmle,cmea,delk,thetk
      write(i_pressure,55) (cp(i),i=1,nodtot)

      if((t.eq.0).or.(MOD(t,idump1).eq.0).or.(MOD(t,idump2).eq.0)) then
c         call print_iter()
               write(i_ani,*)    t
               write(i_ani,56)   gamma(t),gammaw(t)
               write(i_ani,58)   float(t)*dt*chord/uexp,mothis(t),
     &alpha(t)*180./pi,cl,cd,cmle
               write(i_ani,56)   (x(i),i=1,nodtot+2)
               write(i_ani,56)   (y(i),i=1,nodtot+2)
               write(i_ani,56)   (q(i),i=1,nodtot)
               write(i_ani,55)   (xmid(i),i=1,nodtot)
               write(i_ani,55)   (ymid(i),i=1,nodtot)
               write(i_ani,55)   (cp(i),i=1,nodtot)
               write(i_ani,55)   (vtan(i),i=1,nodtot)
               write(i_ani,55)   (vnor(i),i=1,nodtot)
               write(i_ani,57)   (xvort(i),i=1,t)
               write(i_ani,57)   (yvort(i),i=1,t)
               write(i_ani,57)   (vort(i),i=1,t)
            ! Free Stream Vortices
               write(i_ani,*)nfsv
               do k =1,7
                  write(i_ani,57)   (fsvort(i,k),i=1,nfsv)
               end do

      end if
      write(i_temp,55)(intgrl(t,i),i=1,nodtot),(vtan(i),i=1,nodtot),
     & (cp(i),i=1,nodtot)
c---------------------------------------------------------------------
c  CONVECT WAKE PANEL AND WAKE VORTICES
c
c  REV 92
c  REV 100
```

```
!      if (t.ge.1) then
      if ((sstate.eq.0).or.(t.ge.1)) then

      if (debug.ne.0) then
         write(i_debug,*)'  - Convect Wake Panel and Vortices'
      endif

      nwv=nwv+1
      sumvort=0.
      do 480 i=nwv-1,1,-1
         vort(i+1)=vort(i)
         xvort(i+1)=xvort(i)+udv(i)*dt
         yvort(i+1)=yvort(i)+vdv(i)*dt
         nvort(i+1)=nvort(i)
         sumvort=sumvort+vort(i+1)
  480 continue
      vort(1)=gammaw(t)*delk
!      vort(1)=gammaw(t) ! REV 52
      xvort(1)=xmid(nodtot+1)+.5*(u1+u2)*dt
      yvort(1)=ymid(nodtot+1)+.5*(v1+v2)*dt
      nvort(1)=1
c     ------------------------------------------------------------------
c        Convect Freestream Vorts
c     ------------------------------------------------------------------
      ! Step Through All Sheet Vortices and Convect
      do j = 1,nfsv

      ! Upper surface

         ! Check if sheet ends on a surface
         if (fsvort(j,7).ne.0) then

!  rev 120  -  check if stagnation point has moved past gust element
!              endpoint. If so, move element endpoint with stagnation
!              point

            if (fsvort(j,7).lt.fv_tan) then
               fsvort(j,7) = fv_tan
               fsvort(j,3) = x(fsvort(j,7))
               fsvort(j,4) = y(fsvort(j,7))
            end if

            ! Initialize Time
               time = dt
            ! Calculate Distance to travel along surface based
            !  on panel tangential velocity
               travel = vtan(fsvort(j,7)) * time

            do while (travel .gt. 0.0)

               ! Calculate distance from vortex to end of panel
               !  i.e. distance remaining on the panel
                  dx = x(fsvort(j,7)+1) - fsvort(j,3)
                  dy = y(fsvort(j,7)+1) - fsvort(j,4)
                  dist = sqrt( (dx)**2 + (dy)**2 )

!             ! Move end along panels until travel < dist
!                while_flag = 1
!
               if (travel .ge. dist) then
                  if (fsvort(j,7) .le. (nodtot)) then

                     ! subtract time to go length of panel from timestep
                        time = time - dist / vtan(fsvort(j,7))
                     ! Move index to the next panel
```

```
                    fsvort(j,7) = fsvort(j,7) + 1
                 ! place endpoint at end of panel
                    fsvort(j,3) = x(fsvort(j,7))
                    fsvort(j,4) = y(fsvort(j,7))
                 ! Calculate Distance to travel along surface based
                 !  on panel tangential velocity
                    travel = vtan(fsvort(j,7)) * time

                else if (fsvort(j,7) .eq. (nodtot+1)) then

                 ! Move index to the next panel
                    fsvort(j,7) = fsvort(j,7) + 1
                 ! place endpoint at end of panel
                    fsvort(j,3) = x(fsvort(j,7))
                    fsvort(j,4) = y(fsvort(j,7))
                 ! Calculate remaining distance to go past end of
                 !  wake panel
                    travel = travel - dist
                 ! Place end past panel
                    fsvort(j,3) = fsvort(j,3) + dx * travel / dist
                    fsvort(j,4) = fsvort(j,4) + dy * travel / dist
                 ! Zero Panel
                    fsvort(j,7) = 0
                 ! Zero Travel
                    travel = 0.0

              end if

           else if (travel .lt. dist) then
              if (fsvort(j,7) .le. (nodtot+1)) then

                 fsvort(j,3) = fsvort(j,3) + dx * travel / dist
                 fsvort(j,4) = fsvort(j,4) + dy * travel / dist

                 travel = 0.0

               else

              end if

           end if
        end do
     else ! does not end on panel
        ! Move endpoints at local velocity if do not end on surface
           fsvort(j,3) = fsvort(j,3) + dt * vfsvort(j,3)   ! x stop
           fsvort(j,4) = fsvort(j,4) + dt * vfsvort(j,4)   ! y stop
     end if

  ! Lower Surface
     ! Check if sheet ends on a surface
     if (fsvort(j,6).ne.0) then

!  rev 120  -  check if stagnation point has moved past gust element
!              endpoint. If so, move element endpoint with stagnation
!              point

        if (fsvort(j,6).gt.fv_tan) then
           fsvort(j,6) = fv_tan
           fsvort(j,1) = x(fsvort(j,6))
           fsvort(j,2) = y(fsvort(j,6))
        end if

        ! Initialize Time
           time = dt
        ! Calculate Distance to travel along surface based
        !  on panel tangential velocity
```

```
if (fsvort(j,6) .gt. (1)) then
   travel = abs(vtan(fsvort(j,6)-1)) * time
 else
   travel = abs(vtan(nodtot+1)) * time
end if

do while (travel .gt. 0.0)
   ! Calculate distance from vortex to end of panel
   !  i.e. distance remaining on the panel

   if (fsvort(j,6) .gt. (1)) then
      dx = x(fsvort(j,6)-1) - fsvort(j,1)
      dy = y(fsvort(j,6)-1) - fsvort(j,2)
      dist = sqrt( (dx)**2 + (dy)**2 )
    else
      dx = x(nodtot+2) - fsvort(j,1)
      dy = y(nodtot+2) - fsvort(j,2)
      dist = sqrt( (dx)**2 + (dy)**2 )
   end if

   if (travel .ge. dist) then
      if (fsvort(j,6) .gt. (2)) then

         ! subtract time to go length of panel from timestep
            time = time - dist / abs(vtan(fsvort(j,6)-1))
         ! Move index to the next panel
            fsvort(j,6) = fsvort(j,6) - 1
         ! place endpoint at end of panel
            fsvort(j,1) = x(fsvort(j,6))
            fsvort(j,2) = y(fsvort(j,6))
         ! Calculate Distance to travel along surface based
         !  on panel tangential velocity
            travel = abs(vtan(fsvort(j,6)-1)) * time

      else if (fsvort(j,6) .eq. (2)) then

         ! subtract time to go length of panel from timestep
            time = time - dist / abs(vtan(fsvort(j,6)-1))
         ! Move index to the next panel
            fsvort(j,6) = fsvort(j,6) - 1
         ! place endpoint at end of panel
            fsvort(j,1) = x(fsvort(j,6))
            fsvort(j,2) = y(fsvort(j,6))
         ! Calculate Distance to travel along surface based
         !  on panel tangential velocity
            travel = abs(vtan(nodtot+1)) * time

        else if (fsvort(j,6) .eq. (1)) then

         ! place endpoint at end of panel
            fsvort(j,1) = x(nodtot+2)
            fsvort(j,2) = y(nodtot+2)
         ! Calculate remaining distance to go past end of
         !  wake panel
            travel = travel - dist
         ! Place end past panel
            fsvort(j,1) = fsvort(j,1) + dx * travel / dist
            fsvort(j,2) = fsvort(j,2) + dy * travel / dist
         ! Zero Panel
            fsvort(j,6) = 0
         ! Zero Travel
            travel = 0.0
```

```
              end if
           else if (travel .lt. dist) then
              if (fsvort(j,6) .ge. (1)) then
                 fsvort(j,1) = fsvort(j,1) + dx * travel / dist
                 fsvort(j,2) = fsvort(j,2) + dy * travel / dist
                 travel = 0.0
               else if (fsvort(j,6) .eq. (1)) then
              end if
           end if
        end do
      else ! does not end on panel
         ! Move endpoints at local velocity if do not end on surface
           fsvort(j,1) = fsvort(j,1) + dt * vfsvort(j,1)   ! x stop
           fsvort(j,2) = fsvort(j,2) + dt * vfsvort(j,2)   ! y stop
      end if
c     ------------------------------------------------------------------
c        Split Freestream Vorts that Straddle foille
c     ------------------------------------------------------------------

c     ------------------------------------------------------------------
c     Check for panel straddling foille and split
c     ------------------------------------------------------------------
c  START CONDITIONAL
C 1   !  Not Already End On Airfoil
         if ( (fsvort(j,6) + fsvort(j,7)).eq. 0) then
C 2   !  Starting or ending vortex x coord is past x(foille)
         if ((fsvort(j,1) .ge. x(foille))
     &    .or.(fsvort(j,3) .ge. x(foille))) then
C 3   !  Starting or ending vortex x coord is less then x(1)
         if ((fsvort(j,1) .le. x(1)).and.(fsvort(j,3) .le. x(1))) then
C 4
         dx = fsvort(j,2) - y(foille)
         dy = fsvort(j,4) - y(foille)
         !  Vortex y coord straddle y(foille)
         if ((dx*dy) .le. 0.0) then
C 5
         dx = fsvort(j,1) - fsvort(j,3)
         dy = fsvort(j,2) - fsvort(j,4)
         dist = sqrt( (dx)**2 + (dy)**2 ) ! length of vortex panel j
         dyj = fsvort(j,2) - y(foille)       ! y location at LE
         dxj = fsvort(j,1) - dyj / dy * dx   ! Corresponding x location
         dyj = y(foille)
      !  Check that panel is past foil leading edge
         if ((dxj .ge. x(foille)) .or.
     &    (abs(dxj - x(foille)) .lt. 1.0e-6)) then

         dyj = fsvort(j,2) - (y(foille) + y(fv_tan))/2.0
                                             ! y location at LE
         dxj = fsvort(j,1) - dyj / dy * dx   ! Corresponding x location
         dyj = (y(foille) + y(fv_tan))/2.0
```

```
        ! delta to le correlation
          dx = fsvort(j,1) - dxj
          dy = fsvort(j,2) - dyj
        ! ratio of distance to le equiv to length of vortex panel j
          dist = sqrt( (dx)**2 + (dy)**2 ) / dist

!!!     write(*,*) ' dist = ', dist
    ! Fill Temp Array with Location at time t
      do k = 1,4
         fstemp(k) = fsvort(j,k) - vfsvort(j,k)*dt
      end do

      ! From a point on the Panel at t to the stagnation pt
         dxj = fstemp(1)+(fstemp(3)-fstemp(1))*dist
         dyj = fstemp(2)+(fstemp(4)-fstemp(2))*dist
      ! From a point on the Panel at t+1 to the stagnation pt
         dxjp = fsvort(j,1)+(fsvort(j,3)-fsvort(j,1))*dist
         dyjp = fsvort(j,2)+(fsvort(j,4)-fsvort(j,2))*dist
      ! Slopes of each point to the stagnation pt
         mj = (dyj - y(fv_tan+1)) / (dxj - x(fv_tan+1))
         mjp = (dyjp - y(fv_tan+1)) / (dxjp - x(fv_tan+1))
      ! If the slopes are not the same, Change point on
      !  panels and reiterate at 495,
      ! Else, continue

    ! Find Distance traveled from point at t to the Stagnation point
      dx = sqrt((dyj-y(foille))**2 + (dxj-x(foille))**2)
    ! Find Distance traveled from the Stagnation point to point at t+1
      dy = sqrt((dyjp-y(foille))**2 + (dxjp-x(foille))**2)
    ! Find the portion of dt left after stagnation point reached
      mj = dy / (dx + dy) * dt
      if (dist .ge. 0.999) then
         fsvort(j,7) = foille
         nfsv_t = 0
         fsvort(j,3) = x(foille)
         fsvort(j,4) = y(foille)
      else if (dist .le. 0.001) then
         fsvort(j,6) = foille
         nfsv_t = 0
         fsvort(j,1) = x(foille)
         fsvort(j,2) = y(foille)
      else
         nfsv_t = 1
         ! Add one panel to existing sheet
           nfsv = nfsv + 1
         ! Shift Current Panels by One
           do i = nfsv, j+1, -1
              ! Panel Locations
                do k = 1,7
                   fsvort(i,k) = fsvort(i-1,k)
                end do
```

```
                  ! Panel Velocities
                     do k = 1,4
!                         vfsvort(i,k) = vfsvort(i-1,k)
                        if (i.eq.(j+1)) then
                           vfsvort(i,k) = 0.
                        else
                           vfsvort(i,k) = vfsvort(i-1,k)
                        end if
                     end do
               end do
            ! Set Ending vortex
               k = j
               if (fv_tan .gt. foille) then
                  fsvort(k,7) = fv_tan
               else
                  fsvort(k,7) = foille
               end if
               fsvort(k,3) = x(fsvort(k,7))
               fsvort(k,4) = y(fsvort(k,7))
               fsvort(k,5) = fsvort(k,5) * dist
            ! Set Starting vortex
               k = j + 1
               if (fv_tan .lt. foille) then
                  fsvort(k,6) = fv_tan
               else
                  fsvort(k,6) = foille
               end if
               fsvort(k,1) = x(fsvort(k,6))
               fsvort(k,2) = y(fsvort(k,6))
               fsvort(k,5) = fsvort(k,5) * (1 - dist)
c                  ! Stop Execution
c                     goto 9999
         end if ! dist .le. 0.001
         do k = j,j + nfsv_t
         time = mj
         if (fsvort(k,7).ne.0) then
            travel = vtan(fsvort(k,7)) * time
            do while (travel .gt. 0.0)
               ! Calculate distance from vortex to end of panel
               !  i.e. distance remaining on the panel
                  dx = x(fsvort(k,7)+1) - fsvort(k,3)
                  dy = y(fsvort(k,7)+1) - fsvort(k,4)
                  dist = sqrt( (dx)**2 + (dy)**2 )
!            ! Move end along panels until travel < dist
!               while_flag = 1
!
               if (travel .ge. dist) then
                  if (fsvort(k,7) .lt. (nodtot)) then
                     ! subtract time to go length of panel from timestep
                        time = time - dist / vtan(fsvort(k,7))
                     ! Move index to the next panel
                        fsvort(k,7) = fsvort(k,7) + 1
                     ! place endpoint at end of panel
                        fsvort(k,3) = x(fsvort(k,7))
                        fsvort(k,4) = y(fsvort(k,7))
                     ! Calculate Distance to travel along surface based
```

```
            !  on panel tangential velocity
               travel = vtan(fsvort(k,7)) * time
          else if (fsvort(k,7) .eq. (nodtot)) then
            ! subtract time to go length of panel from timestep
               time = time - dist / vtan(fsvort(k,7))
            ! Move index to the next panel
               fsvort(k,7) = fsvort(k,7) + 1
            ! place endpoint at end of panel
               fsvort(k,3) = x(fsvort(k,7))
               fsvort(k,4) = y(fsvort(k,7))
            ! Calculate Distance to travel along surface based
            !  on panel tangential velocity
               travel = vtan(fsvort(k,7)) * time
          else if (fsvort(k,7) .eq. (nodtot+1)) then
            ! Move index to the next panel
               fsvort(k,7) = fsvort(k,7) + 1
            ! place endpoint at end of panel
               fsvort(k,3) = x(fsvort(k,7))
               fsvort(k,4) = y(fsvort(k,7))
            ! Calculate remaining distance to go past end of
            !  wake panel
               travel = travel - dist
            ! Place end past panel
               fsvort(k,3) = fsvort(k,3) + dx * travel / dist
               fsvort(k,4) = fsvort(k,4) + dy * travel / dist
            ! Zero Panel
               fsvort(k,7) = 0
            ! Zero Travel
               travel = 0.0
         end if
      else if (travel .lt. dist) then
         if (fsvort(k,7) .le. (nodtot+1)) then
            fsvort(k,3) = fsvort(k,3) + dx * travel / dist
            fsvort(k,4) = fsvort(k,4) + dy * travel / dist
            travel = 0.0
          else
         end if
      end if
   end do
end if

if (fsvort(k,6).ne.0) then
   ! Calculate Distance to travel along surface based
   !  on panel tangential velocity
   if (fsvort(k,6) .gt. (1)) then
      travel = abs(vtan(fsvort(k,6)-1)) * time
    else
      travel = abs(vtan(nodtot+1)) * time
   end if
   do while (travel .gt. 0.0)
      ! Calculate distance from vortex to end of panel
      !  i.e. distance remaining on the panel
      if (fsvort(k,6) .gt. (1)) then
         dx = x(fsvort(k,6)-1) - fsvort(k,1)
```

```
     dy = y(fsvort(k,6)-1) - fsvort(k,2)
     dist = sqrt( (dx)**2 + (dy)**2 )
   else
     dx = x(nodtot+2) - fsvort(k,1)
     dy = y(nodtot+2) - fsvort(k,2)
     dist = sqrt( (dx)**2 + (dy)**2 )
  end if

  if (travel .ge. dist) then
     if (fsvort(k,6) .gt. (2)) then
        ! subtract time to go length of panel from timestep
          time = time - dist / abs(vtan(fsvort(k,6)-1))
        ! Move index to the next panel
          fsvort(k,6) = fsvort(k,6) - 1
        ! place endpoint at end of panel
          fsvort(k,1) = x(fsvort(k,6))
          fsvort(k,2) = y(fsvort(k,6))
        ! Calculate Distance to travel along surface based
        !  on panel tangential velocity
          travel = abs(vtan(fsvort(k,6)-1)) * time

     else if (fsvort(k,6) .eq. (2)) then
        ! subtract time to go length of panel from timestep
          time = time - dist / abs(vtan(fsvort(k,6)-1))
        ! Move index to the next panel
          fsvort(k,6) = fsvort(k,6) - 1
        ! place endpoint at end of panel
          fsvort(k,1) = x(fsvort(k,6))
          fsvort(k,2) = y(fsvort(k,6))
        ! Calculate Distance to travel along surface based
        !  on panel tangential velocity
          travel = abs(vtan(nodtot+1)) * time

       else if (fsvort(k,6) .eq. (1)) then
        ! place endpoint at end of panel
          fsvort(k,1) = x(nodtot+2)
          fsvort(k,2) = y(nodtot+2)
        ! Calculate remaining distance to go past end of
        !  wake panel
          travel = travel - dist
        ! Place end past panel
          fsvort(k,1) = fsvort(k,1) + dx * travel / dist
          fsvort(k,2) = fsvort(k,2) + dy * travel / dist
        ! Zero Panel
          fsvort(k,6) = 0
        ! Zero Travel
          travel = 0.0

     end if

  else if (travel .lt. dist) then

     if (fsvort(k,6) .ge. (1)) then

        fsvort(k,1) = fsvort(k,1) + dx * travel / dist
        fsvort(k,2) = fsvort(k,2) + dy * travel / dist

        travel = 0.0

      else if (fsvort(k,6) .eq. (1)) then
     end if
```

```
            end if
         end do
      end if
      end do ! k
c      pause
C 5
      end if
C 4
      end if   !  if ((dx*dy) .lt. 0.0) then
C 3
      end if
C 2
      end if   !  if ((fsvort(j,1) .ge. x(foille)).or.(fsvort(j,3)
               !  .ge. x(foille))) then
C 1
      end if   !  if ( (fsvort(j,6) + fsvort(j,7)).eq. 0) then
c  END CONDITIONAL
      end do ! j (nfsv)
c  REV 92
      end if   ! (t.ge.1)
c------------------------------------------------------------------------
c  RETURN TO NEXT TIME STEP
c
      if (debug.ne.0) then
         write(i_debug,*)'  - End Time Step'
      endif
!------------------------------------------------------------------------
!  Print Forces to screen
!------------------------------------------------------------------------
!      if((t.eq.idump1).or.(idump1.eq.0).or.(MOD(t,idump2).eq.0)) then
      if((t.eq.0).or.(MOD(t,idump1).eq.0)) then
      if (t.le.1) write(6,49)'t','h','alpha','cl','cmle','cmea','iter',
     & 'GAMMA','GAMMAW'!,'phi_le','phi_us','phi_ls'
      write(6,50)float(t)*dt*chord/uexp,mothis(t),alpha(t)*180./pi,cl,
     &cmle,cmea,dreal(iter)/1.D3,gamma(t)*perim,gammaw(t)*delk
!     &phi_le,
!     &intgrl(t,nodtot),
!     &intgrl(t,1)
      end if
      clp = cl
      cmeap = cmea
      cl_s(t) = cl
      cmle_s(t) = cmle
      cmea_s(t) = cmea
      cmo_s(t) = cmle + cl/4.
      cd_s(t) = cd
      t_s(t) = t*dt
 1000 continue
!------------------------------------------------------------------------
!------------------------------------------------------------------------
!  Close Open Data Files
      write(6,*)'Closing Data Files'
      if (i_debug .NE. 6) then   ! Close Debug if open
         close(UNIT=i_debug)
```

```
      endif
      CLOSE(UNIT=i_force)
!-----------------------------------------------------------------------
!  Print Summary Output
!-----------------------------------------------------------------------
c      call print_readme()
!-----------------------------------------------------------------------
!  Print Status to Screen
!-----------------------------------------------------------------------
      write(6,*)'Finished'
      elapsed = etime(extime)
      write(*,*) 'Executed in ', elapsed, '(s) CPU time.'
      write(*,*) extime(1), '(s) user time, ',  extime(2),
     &'(2) system time.'
!-----------------------------------------------------------------------
 9999 continue
!-----------------------------------------------------------------------
!----- Close Open Graphics Windows
      if (graphics.eq.1.) then
!         call pgslct(istat(3))
!         call pgclos
!         call pgslct(istat(2))
!         call pgclos
         call pgslct(istat(1))
         call pgclos
      end if
      stop
      end
!-----------------------------------------------------------------------
c  End Program Unpanel
!-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c  Include Files
c-----------------------------------------------------------------------
c     (-G77 Start)
         include 'read_foil.f'
         include 'read_freevort.f'
         include 'read_motion.f'
         include 'plot_motion.f'
         include 'plot_airfoil.f'
         Include 'rotate_foil_r109.f'
         include 'rk4_resp.f'
         Include 'rotate_pt.f'
         include 'calc_panel_inf.f'
         include 'calc_point_inf.f'
         include 'gauss.f'
         include 'calc_pt_vel.f'
c-----------------------------------------------------------------------
c  Subroutines and Functions
c-----------------------------------------------------------------------
C     namlen DIRECTLY from Tim Cowan's Euler3d utilities.
c*********************************************************************c
c*********************************************************************c
      integer function namlen( filen )
c
c*********************************************************************c
c
      character*72 filen
```

```
c
c----------------------------------------------------------------------c
c
      namlen = 0
      do i = 72,1,-1
          if ( filen(i:i) .ne. ' ' ) then
              namlen = i
              goto 101
          endif
      enddo
  101 return
c
      end
c**********************************************************************c
c**********************************************************************c

      integer function rootlen( filen )
c
c**********************************************************************c
c
      character*72 filen
c
c----------------------------------------------------------------------c
c
      namlen = 0
      do i = 72,1,-1
          if (filen(i:i) .eq. '.') then
              rootlen = i-1
              goto 101
          endif
      enddo
  101 return
c
      end
c**********************************************************************c

c**********************************************************************c
c**********************************************************************c

      integer function dirlen( filen )
c
c**********************************************************************c
c
      character*72 filen
c
c----------------------------------------------------------------------c
c
      namlen = 0
      do i = 72,1,-1
          if ((filen(i:i) .eq. '/').or.(filen(i:i) .eq. '\')) then
              dirlen = i-1
              goto 101
          endif
      enddo
  101 return
c
      end
c**********************************************************************c
c-----------------------------------------------------------------------

      real*8   function local_tan(u,v,costhe,sinthe)
c     ------------------------------------------------------------------
      implicit none
      real*8   u,v,costhe,sinthe

         local_tan = u*costhe + v*sinthe

      return
      end
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------

      real*8   function local_nor(u,v,costhe,sinthe)
c     ------------------------------------------------------------------
```

```
      implicit none
      real*8   u,v,costhe,sinthe
         local_nor = -u*sinthe + v*costhe
      return
      end
c-----------------------------------------------------------------------
```

APPENDIX B

COMMON FILES

B.1 Common Variables Declarations

The common files contain variables used in multiple locations. The files are organized around Common Blocks and variable usage.

B.1.1 lengths.inc

```
!----------------------------------------------------------------------
!-- Lengths.inc
!----------------------------------------------------------------------
      INTEGER lpanel
      INTEGER liter
      INTEGER lphi
      integer lfree
      integer knd
      parameter( lpanel = 300, liter = 10000, lphi = 201 , lfree = 3001,
     & knd = 4 )
!----------------------------------------------------------------------
```

B.1.2 airfoil.inc

```
!----------------------------------------------------------------------
!-- airfoil.inc
!----------------------------------------------------------------------
      common /airfoil/ nodle, nodtot, x0, y0, x, y, xmid, ymid, xmidp,
     &ymidp, costhe, sinthe, perim, foille

         INTEGER nodle           !  node of leading edge
         INTEGER nodtot          !  total number of nodes
         REAL*8 x0               !  x-coordinate of the Elastic Axis
         REAL*8 y0               !  y-coordinate of the Elastic Axis
         REAL*8 x(lpanel)        !  x-coordinate of node location
         REAL*8 y(lpanel)        !  y-coordinate of node location
         REAL*8 xmid(lpanel)     !  x-coordinate of middle of panel
         REAL*8 ymid(lpanel)     !  y-coordinate at middle of panel
         REAL*8 xmidp(lpanel)    !  x-coordinate of middle of panel
                                 !   previous time step
         REAL*8 ymidp(lpanel)    !  y-coordinate of middle of panel
                                 !   previous time step
         REAL*8 costhe(lpanel)   !  array of cosine of angle of panels
                                 !   with the x axis
         REAL*8 sinthe(lpanel)   !  array of sine of angle of panels
                                 !   with x-axis
         REAL*8 perim            !  perimeter of the airfoil
         integer  foille
!----------------------------------------------------------------------
```

B.1.3 calc.inc

```
!-----------------------------------------------------------------------
!-- calc.inf
!-----------------------------------------------------------------------
      common /pt_inf/ vel
      real*8 vel(4)
!-----------------------------------------------------------------------
```

B.1.4 const.inc

```
!-----------------------------------------------------------------------
!-- const.inc
!-----------------------------------------------------------------------
      common /const/ pi, pi2inv, rho_fluid
      REAL*8 pi                !  3.141593...
      REAL*8 pi2inv            ! 1/(2*pi)
      real*8 rho_fluid         ! Density of fluid
!-----------------------------------------------------------------------
```

B.1.5 debug.inc

```
!-----------------------------------------------------------------------
!-- debug.inc
!-----------------------------------------------------------------------
      common /debugstatus/ debug

      integer debug
      integer debug_wake
!-----------------------------------------------------------------------
```

B.1.6 file.inc

```
!-----------------------------------------------------------------------
!-- file.inc
!-----------------------------------------------------------------------
      common /file/  i_readme, i_airfoil, i_config, i_debug, i_mot,
     &i_pressure, i_force, i_foil, i_tan, i_vortex, i_elements,
     &i_data, idump1, idump2, fn, f_debug, f_mot, f_foil, foil_desc,
     &f_vort, i_temp

      ! File Identifiers
         INTEGER i_readme
         INTEGER i_airfoil
         INTEGER i_config
         INTEGER i_direc
         INTEGER i_debug
         INTEGER i_mot
         INTEGER i_pressure
         INTEGER i_force
         INTEGER i_foil
         INTEGER i_tan
         INTEGER i_vortex
         INTEGER i_elements
         INTEGER i_data
         integer i_ani
         integer i_ani2
         INTEGER idump1
         INTEGER idump2
```

```
         INTEGER len_config
         integer i_temp
         integer i_phi
         integer i_samp
         integer i_resp

      ! File Names
         CHARACTER*72 fn
         CHARACTER*72 f_debug
         CHARACTER*72 f_mot
         CHARACTER*72 f_foil
         CHARACTER*72 foil_desc
         CHARACTER*72 f_vort
         CHARACTER*72 f_config
         CHARACTER*72 f_samp
!-----------------------------------------------------------------------
```

B.1.7 forces.inc

```
!-----------------------------------------------------------------------
!-- forces.inc
!-----------------------------------------------------------------------
      common /forces/ cl, cd, cmle, cmea, cp, cl_s, cmle_s, cmea_s,
     & cmo_s, cd_s, t_s, clp, cmeap

      REAL*8 cl                !  Coefficient of Lift
      REAL*8 cd                !  Coefficient of Drag
      REAL*8 cmle              !  Set equal to cmsum
      REAL*8 cmea              !  Set equal to cmsum
      REAL*8 cp(lpanel)        !  Cp on each panel

      real*8 cl_s(0:liter)     !  Cl Time History
      real*8 cmle_s(0:liter)   !  Cmle Time History
      real*8 cmea_s(0:liter)   !  Cmea Time History
      real*8 cmo_s(0:liter)    !  Cmo Time History
      real*8 cd_s(0:liter)     !  Cd Time History
      real*8 t_s(0:liter)      !  Time History

      real*8 clp               !  CL for previous time step
      real*8 cmeap             !  cmea for previous time step
!-----------------------------------------------------------------------
```

B.1.8 freeresp.inc

```
!-----------------------------------------------------------------------
!-- freeresp,inc
!-----------------------------------------------------------------------
      common   /freeresp/ mu, k_a, k_h, r_a, x_a,
     & omega, hdot, eta_a, eta_h, ufre, f_responce

      integer  f_responce
      real*8   mu
      real*8   k_a
      real*8   k_h
      real*8   r_a
      real*8   x_a

      real*8   omega
      real*8   hdot
```

```
      real*8   eta_a
      real*8   eta_h

      real*8   ufre
!-----------------------------------------------------------------------
```

B.1.9 freevort.inc

```
!-----------------------------------------------------------------------
!-- freevort.inc
!-----------------------------------------------------------------------
      common /freevort/ fsvort, fv_split, fv_tan, nfsv

      integer  nfsv              !  Number of free stream vortex panels
      real*8     fsvort(lfree, 7)
      !  Store Free Stream Vortex Location and Strength
      !     (n,1) Panel Starting Location x coordinate
      !     (n,2) Panel Starting Location y coordinate
      !     (n,3) Panel Ending Location x coordinate
      !     (n,4) Panel Ending Location y coordinate
      !     (n,5) Vortex Strength
      !     (n,6) Panel Vortex Starts on if along airfoil
      !     (n,7) Panel Vortex Ends on if along airfoil

      real*8  vfsvort(lfree, 4)
      !  Stores Free Stream Vortex Velocity
      !     (n,1) Panel Starting Location x velocity
      !     (n,2) Panel Starting Location y velocity
      !     (n,3) Panel Ending Location x velocity
      !     (n,4) Panel Ending Location y velocity

      real*8  fstemp(5)
      !  Temporary calculation storage for first 5 values of fsvort()

      integer fv_split
      integer fv_tan
      integer stag_pt
!-----------------------------------------------------------------------
```

B.1.10 gau.inc

```
!-----------------------------------------------------------------------
!-- gau.inc
!-----------------------------------------------------------------------
      common /gau/ a
REAL*8 a(lpanel+1,lpanel+1)      !  [A] For Gauss Solver
!-----------------------------------------------------------------------
```

B.1.11 graph.inc

```
!-----------------------------------------------------------------------
!-- graph.inc
!-----------------------------------------------------------------------
      common /graph/ graphics, savegif

      INTEGER  graphics    ! Determines if Graphical output should be
                           !  displayed at runtime
      INTEGER  savegif     ! Determines if Graphical output should be
                           !  Saved as Gifs at runtime
```

```
      INTEGER  zm_field
      real     zm_field_x(2)  ! (1) min   (2) max
      real     zm_field_y(2)  ! (1) min   (2) max
!-------------------------------------------------------------------------
```

B.1.12 iterative.inc

```
!-------------------------------------------------------------------------
!-- iterative.inc
!-------------------------------------------------------------------------
      common   /iterative/ t
      INTEGER t            !  time (iteration)
!-------------------------------------------------------------------------
```

B.1.13 motion.inc

```
!-------------------------------------------------------------------------
!-- motion.inc
!-------------------------------------------------------------------------
      common /motion/   alpha, mothis, nstep
      INTEGER  nstep             !  total number of time steps
      REAL*8   alpha(0:liter)    !  airfoil angle of attack (deg)
      REAL*8   mothis(0:liter)
!-------------------------------------------------------------------------
```

B.1.14 param.inc

```
!-------------------------------------------------------------------------
!-- param.inc
!-------------------------------------------------------------------------
      common /param/ uexp, dt, alphafs, chord
      REAL*8 uexp             !  Free Stream Velocity
      REAL*8 dt               !  time step
      REAL*8 alphafs          !  Free Stream Angle of Attack
      REAL*8 chord            !  Chord Length of the airfoil
!-------------------------------------------------------------------------
```

B.1.15 phi.inc

```
!-------------------------------------------------------------------------
!-- phi.inc
!-------------------------------------------------------------------------
      common /phi/   x_far, y_far, xpi, ypi, upi, vpi, npi
      ! Integrating Points
         integer  npi      !  Number Of Integrating Points
         real*8 x_far
         real*8 y_far
         REAL*8 xpi(lphi)  !  x-coordinate of phi integration point
         REAL*8 ypi(lphi)  !  y-coordinate of phi integration point
         REAL*8 upi(lphi)  !  x-velocity component at phi integration
                           !  point
         REAL*8 vpi(lphi)  !  y-velocity component at phi integration
                           !  point
!-------------------------------------------------------------------------
```

B.1.16 relax.inc

```
!-----------------------------------------------------------------------
!-- relax.inc
!-----------------------------------------------------------------------
      common /relax/ relax_delk, relax_thetk, relax_gammaw

      real*8 relax_delk
      real*8 relax_thetk
      real*8 relax_gammaw
!-----------------------------------------------------------------------
```

B.1.17 strengths.inc

```
!-----------------------------------------------------------------------
!-- strengths.inc
!-----------------------------------------------------------------------
      common /strengths/ q, gamma, gammaw
      REAL*8 q(lpanel)         !  strength of each source panel
      REAL*8 gamma(0:liter)    !  vorticity on airfoil
      REAL*8 gammaw(0:liter)   !  vorticity on wake panel
!-----------------------------------------------------------------------
```

B.1.18 velocities.inc

```
!-----------------------------------------------------------------------
!-- velocities.inc
!-----------------------------------------------------------------------
      common /velocities/ vtan, vnor
      REAL*8 vtan(lpanel)      !  tangential velocity on panel
      REAL*8 vnor(lpanel)      !  tangential velocity on panel

      REAL*8 vxdir(lpanel)     !  tangential velocity on panel
      REAL*8 vydir(lpanel)     !  tangential velocity on panel
!-----------------------------------------------------------------------
```

B.1.19 wake.inc

```
!-----------------------------------------------------------------------
!-- wake.inc
!-----------------------------------------------------------------------
      common /wake/ nwv, ngv, vgd, nvort, xvort, yvort, vort, udv, vdv

      INTEGER nwv           !  Total # of wake vortices (including "big"
                            !   ones)
      INTEGER ngv           !  number of small vortices past vortice
                            !   grouping distance before they are grouped
                            !   into a "big" vortice
      REAL*8 vgd            !  vortice grouping distance (nondimensional,
                            !   in chord lengths)
      INTEGER nvort(liter)  !  # of vortices that have been grouped into
                            !   wake vortex i
      REAL*8 xvort(liter)   !  x-coordinate of discrete vortex location
      REAL*8 yvort(liter)   !  y-coordinate of discrete vortex location
      REAL*8 vort(liter)    !  strength of discrete vortex
      REAL*8 udv(liter)     !  x-velocity component at discrete vortex
      REAL*8 vdv(liter)     !  y-velocity component at discrete vortex
!-----------------------------------------------------------------------
```

B.1.20 wakepannel.inc

```
!----------------------------------------------------------------------
!-- wakepanel.inc
!----------------------------------------------------------------------
      common /wakepannel/  delk, delk1, dkcon, tkcon, thetk, thetk1
      REAL*8 delk       !  length of wake panel
      REAL*8 delk1      !  length of wake panel (last iteration)
      REAL*8 dkcon      !  convergence of delk
      REAL*8 tkcon      !  convergence of thetk
      REAL*8 thetk      !  angle of wake panel with x-axis
      REAL*8 thetk1     !  angle of wake panel with x-axis (last
                        !   iteration)
      real*8 gwcon
      real*8 gammaw1
!----------------------------------------------------------------------
```

B.1.21 graph_cons.inc

```
!----------------------------------------------------------------------
!-- graph_cons.inc
!----------------------------------------------------------------------
      character*20 color_n(0:15)
      data color_n /'Black ',  'White ', 'Red ', 'Green ', 'Blue ',
     & 'Cyan ', 'Magenta ', 'Yellow ', 'Orange ', 'Green + Yellow ',
     & 'Green + Cyan ', 'Blue + Cyan ', 'Blue + Magenta ',
     & 'Red + Magenta ', 'Dark Gray ', 'Light Gray '/

      !---------------------------------------------------------------
      ! Index Color
      ! ---   --------------------- ----  ----  ----   ----  ----  ----
      !   0   Black (background)      0, 0.00, 0.00   0.00, 0.00, 0.00
      !   1   White (default)         0, 1.00, 0.00   1.00, 1.00, 1.00
      !   2   Red                   120, 0.50, 1.00   1.00, 0.00, 0.00
      !   3   Green                 240, 0.50, 1.00   0.00, 1.00, 0.00
      !   4   Blue                    0, 0.50, 1.00   0.00, 0.00, 1.00
      !   5   Cyan (Green + Blue)   300, 0.50, 1.00   0.00, 1.00, 1.00
      !   6   Magenta (Red + Blue)   60, 0.50, 1.00   1.00, 0.00, 1.00
      !   7   Yellow  (Red + Green) 180, 0.50, 1.00   1.00, 1.00, 0.00
      !   8   Red + Yellow (Orange) 150, 0.50, 1.00   1.00, 0.50, 0.00
      !   9   Green + Yellow        210, 0.50, 1.00   0.50, 1.00, 0.00
      !  10   Green + Cyan          270, 0.50, 1.00   0.00, 1.00, 0.50
      !  11   Blue + Cyan           330, 0.50, 1.00   0.00, 0.50, 1.00
      !  12   Blue + Magenta         30, 0.50, 1.00   0.50, 0.00, 1.00
      !  13   Red + Magenta          90, 0.50, 1.00   1.00, 0.00, 0.50
      !  14   Dark Gray               0, 0.33, 0.00   0.33, 0.33, 0.33
      !  15   Light Gray            0, 0.66, 0.00   0.66, 0.66, 0.66
!----------------------------------------------------------------------
```

APPENDIX C

INPUT FILES

C.1 Configuration File

```
!------------------------------------------------------------------------
!  Configuration file, set Namelists for runtime parameters
!------------------------------------------------------------------------
!------------------------------------------------------------------------
! &vpm_in
!  -----------------------------------------------------------------------
!  F_FOIL      -  Input File With Airfoil Coordinates
!  X0          -  x/c Location Of Elastic Axis
!  Y0          -  y/c Location Of Elastic Axis
!
!  F_RESPONCE  -  Calculate Airfoil Free Elastic Responce if > 0
!  B_RATIO     -  Ratio Of Pitching Frequency To Plunging Frequency
!                 (  Omega_alpha / Omega_h  )
!  MU          -  Normalized Density Ratio
!                 (  m / pi*rho*b^2         )
!  X_ALPHA     -  Dimensionless Static Imbalance
!                 (  sqrt(S_alpha / m*b     )
!  R_ALPHA     -  Dimensionless Radius Of Gyration
!                 (  I_alpha^2.....         )
!
!  F_MOT       -  Arbitrary Motion Input File
!                 (  CSV Or Space Delimited )
!  F_VORT      -  Starting Location for Free Stream Vortex Sheets
!                 (  CSV Or Space Delimited )
!                 -- Note, use 'none' for filename if there are no free
!                    stream vorticies
!
!  IDUMP1      -  A Single Time Step To Save Data At
!  IDUMP2      -  A Time Step Multiple To Save Data At
!
!  DEBUG       -  Show Debug Data/Comments
!  DEBUG_WAKE  -  Not Used
!  I_DEBUG     -  Where to Send Debug Info, 6 = screen, any other
!                 integers save to file
!
!  RELAX_GAMMAW   -  Over/Under Relaxation Factors for Wake Panel
!                    Iterations
!  RELAX_DELK     -  Over/Under Relaxation Factors for Wake Panel
!                    Iterations
!  RELAX_THETK    -  Over/Under Relaxation Factors for Wake Panel
!                    Iterations
!
!------------------------------------------------------------------------
! &graph - Set Graphics Parameters
! -----------------------------------------------------------------------
!
!  GRAPHICS    -  To Show Graphics, set GRAPHICS > 0
!  SAVEGIF     -  To Save graphics as gifs instead of sending to
```

```
!                   display, set SAVEGIF > 0
!  ZM_FIELD    -  Zoom Flowfield Display
!                 0  =  Default
!                 1  =  Zoom to dimensions specified in ZM_FIELD_X and
!                       ZM_FIELD_Y
!                 2  =  Convects View Specified At t=0 by ZM_FIELD_X and
!                       ZM_FIELD_Y With The Free Stream Velocity
!  ZM_FIELD_X  -  Two Element Vector Specifying MIN and MAX Region
!  ZM_FIELD_Y
!-------------------------------------------------------------------------
! &phi_int - Set Phi Integration Parameters
! ------------------------------------------------------------------------
!  npi         -  Number of points to place between leading edge and
!                 Phi = 0
!  x_far       -  x location to place point where Phi = 0
!  y_far       -  y location to place point where Phi = 0
!
!-------------------------------------------------------------------------
&vpm_in
   F_FOIL   =  'in\airfoils\c060p066\n0012.100',
   X0       =  0.25,
   Y0       =  0.0,

   F_RESPONCE  =  0,
   B_RATIO     =  1,
   MU          =  20.,
   X_ALPHA     =  0.0,
   R_ALPHA     =  0.5,

   F_MOT    =  'in\motion\wa01t005.mot',

   F_VORT   =  'none',

   IDUMP1      =  1,
   IDUMP2      =  1,

   DEBUG       =  0,
   DEBUG_WAKE  =  0,
   I_DEBUG     =  6,

   RELAX_GAMMAW   =  1.0,
   RELAX_DELK     =  1.0,
   RELAX_THETK    =  1.0
/
&graph
   GRAPHICS    =  0,
   SAVEGIF     =  0,
   ZM_FIELD    =  0,
   ZM_FIELD_X  =  -.05 1.05
   ZM_FIELD_Y  =  -0.15  0.15
/
&phi_int
   npi   =  100
   x_far =  -10.0
   y_far =  0.0
/
```

C.2 Airfoil Coordinates

```
NACA0.12
1.00000000  0.00000000
0.86666667 -0.01760195
0.73333333 -0.03264190
```

```
0.60000000 -0.04518009
0.47410810 -0.05421574
0.34986683 -0.05933467
0.23519668 -0.05890533
0.13740798 -0.05205985
0.06273483 -0.03908178
0.01593772 -0.02123112
0.00000000  0.00000000
0.01593772  0.02123112
0.06273483  0.03908178
0.13740798  0.05205985
0.23519668  0.05890533
0.34986683  0.05933467
0.47410810  0.05421574
0.60000000  0.04518009
0.73333333  0.03264190
0.86666667  0.01760195
1.00000000 -0.00000000
```

C.3 Motion History

```
 Free Stream Velocity (uinf),,
   1.000000     ,,
 Time Incrememnt - (dt),,
  .005,,
 tstep (integer),alpha (deg), mothis (chord)
  0.0000000E+00 ,   1.000000     ,  0.0000000E+00
  9.9999998E-03 ,   1.000000     ,  0.0000000E+00
  2.0000000E-02 ,   1.000000     ,  0.0000000E+00
  2.9999999E-02 ,   1.000000     ,  0.0000000E+00
  3.9999999E-02 ,   1.000000     ,  0.0000000E+00
  4.9999997E-02 ,   1.000000     ,  0.0000000E+00
  5.9999999E-02 ,   1.000000     ,  0.0000000E+00
  7.0000000E-02 ,   1.000000     ,  0.0000000E+00
  7.9999998E-02 ,   1.000000     ,  0.0000000E+00
  8.9999996E-02 ,   1.000000     ,  0.0000000E+00
  9.9999994E-02 ,   1.000000     ,  0.0000000E+00
  0.1100000     ,   1.000000     ,  0.0000000E+00
  0.1200000     ,   1.000000     ,  0.0000000E+00
  0.1300000     ,   1.000000     ,  0.0000000E+00
  0.1400000     ,   1.000000     ,  0.0000000E+00
  0.1500000     ,   1.000000     ,  0.0000000E+00
  0.1600000     ,   1.000000     ,  0.0000000E+00
  0.1700000     ,   1.000000     ,  0.0000000E+00
  0.1800000     ,   1.000000     ,  0.0000000E+00
  0.1900000     ,   1.000000     ,  0.0000000E+00
  0.2000000     ,   1.000000     ,  0.0000000E+00
```

C.4 Free Stream Vortices

```
x1          y1          x2          y2          GAMMA
-1.000000   2.000000    -1.000000   1.900000    0.0100000
-1.000000   1.900000    -1.000000   1.800000    0.0100000
-1.000000   1.800000    -1.000000   1.700000    0.0100000
-1.000000   1.700000    -1.000000   1.600000    0.0100000
-1.000000   1.600000    -1.000000   1.500000    0.0100000
-1.000000   1.500000    -1.000000   1.400000    0.0100000
-1.000000   1.400000    -1.000000   1.300000    0.0100000
-1.000000   1.300000    -1.000000   1.200000    0.0100000
-1.000000   1.200000    -1.000000   1.100000    0.0100000
-1.000000   1.100000    -1.000000   1.000000    0.0100000
-1.000000   1.000000    -1.000000   0.900000    0.0100000
-1.000000   0.900000    -1.000000   0.800000    0.0100000
-1.000000   0.800000    -1.000000   0.700000    0.0100000
-1.000000   0.700000    -1.000000   0.600000    0.0100000
```

-1.000000	0.600000	-1.000000	0.500000	0.0100000
-1.000000	0.500000	-1.000000	0.400000	0.0100000
-1.000000	0.400000	-1.000000	0.300000	0.0100000
-1.000000	0.300000	-1.000000	0.200000	0.0100000
-1.000000	0.200000	-1.000000	0.100000	0.0100000
-1.000000	0.100000	-1.000000	-0.100000	0.0200000
-1.000000	-0.100000	-1.000000	-0.200000	0.0100000
-1.000000	-0.200000	-1.000000	-0.300000	0.0100000
-1.000000	-0.300000	-1.000000	-0.400000	0.0100000
-1.000000	-0.400000	-1.000000	-0.500000	0.0100000
-1.000000	-0.500000	-1.000000	-0.600000	0.0100000
-1.000000	-0.600000	-1.000000	-0.700000	0.0100000
-1.000000	-0.700000	-1.000000	-0.800000	0.0100000
-1.000000	-0.800000	-1.000000	-0.900000	0.0100000
-1.000000	-0.900000	-1.000000	-1.000000	0.0100000
-1.000000	-1.000000	-1.000000	-1.100000	0.0100000
-1.000000	-1.100000	-1.000000	-1.200000	0.0100000
-1.000000	-1.200000	-1.000000	-1.300000	0.0100000
-1.000000	-1.300000	-1.000000	-1.400000	0.0100000
-1.000000	-1.400000	-1.000000	-1.500000	0.0100000
-1.000000	-1.500000	-1.000000	-1.600000	0.0100000
-1.000000	-1.600000	-1.000000	-1.700000	0.0100000
-1.000000	-1.700000	-1.000000	-1.800000	0.0100000
-1.000000	-1.800000	-1.000000	-1.900000	0.0100000
-1.000000	-1.900000	-1.000000	-2.000000	0.0100000

APPENDIX D
GRAPHICS ROUTINES

D.1 Plotting Routines

Plot and compare output.

D.1.1 Compare Data r10

```
      PROGRAM  compare
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
!  rev 10
!     -  Remove comments and unnecessary code for Publication
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
!----- Variables
      IMPLICIT NONE

      !- Include Common Variable Definitions
         INCLUDE 'graph_cons.inc'

      !- Array Length Parameters
         INTEGER liter
         INTEGER lcompare
         PARAMETER( liter = 10000, lcompare = 10 )

      !- Data Variables
         REAL TIME(liter,lcompare)
         REAL h(liter,lcompare)
         REAL alpha(liter,lcompare)
         REAL cl(liter,lcompare)
         REAL CMLE(liter,lcompare)
         REAL CMEA(liter,lcompare)
         REAL CD(liter,lcompare)
         INTEGER  nstep(lcompare)
         CHARACTER*20 titles(lcompare)

      !- Code Variables
         REAL temp
         CHARACTER*72   fn(lcompare)
         INTEGER  i_data, i, j

      !- Graphics Variables
         INTEGER pgopen
         INTEGER istat(10)
         REAL xmin
         REAL xmax
         REAL ymin
         REAL ymax
       INTEGER just, axis
       CHARACTER*70 title
       REAL pgscale
```

```
       REAL xtempa(liter,lcompare)
       REAL ytempa(liter,lcompare)
       REAL ztempa(liter,lcompare)
       REAL xtemp(liter)
       REAL ytemp(liter)
       REAL ztemp(liter)

       INTEGER ltemp

         INTEGER nsets
         INTEGER gtype
         INTEGER gforce
         INTEGER reread

         INTEGER namlen
         INTEGER leng

         real*8   rate(liter)
         real*8   peaks_t(liter)
         real*8   peaks_amp(liter)
         integer  npeaks

         real*8   troughs_t(liter)
         real*8   troughs_amp(liter)
         integer  ntroughs

      !- Initalize Variables
         i_data = 11

!----- Format Statements

!----- Body

!-------- Prompt for File Input
      write(*,*)' Number of Datasets to Compare?'
      read(*,*) nsets

      do j=1,nsets
         write(*,*)' Name of set (',j,') relative to cd (72 char max)'
         read(*,*)fn(j)
      end do

!-------- Plot Parameters

      istat(1) = pgopen('/xserve')
c      istat(1) = pgopen('?')
      if (istat(1).le.0) stop
      call PGASK(.false.)

!-------- Read Input Files
 1000 continue
      reread = 0
      do j = 1,nsets
         open(UNIT=i_data, FILE=fn(j), status='unknown')
         i = 1
         do while(.true.)
            ! Read coordinates into x(i) and y(i)
 1010       format(9(E20.10,1X))
            read(i_data,fmt=1010,end=1020) time(i,j),h(i,j),alpha(i,j),
     & CL(i,j),CD(i,j),CMLE(i,j),CMEA(i,j) !,temp
            i=i+1
         end do
 1020    continue
```

```
         nstep(j) = i-1
!         write(*,*)'j ',j,' i ',i,' nstep ',nstep(j)
         close(UNIT=i_data)
      end do



!-------- Define Titles
         titles(1) = 'Cl    vs. Time'
         titles(2) = 'Cmle  vs. Time'
         titles(3) = 'Cmea  vs. Time'
         titles(4) = 'Cd    vs. Time'
         titles(5) = 'Alpha vs. Time'
         titles(6) = 'h/c   vs. Time'
         titles(7) = 'h/c   vs. Alpha'


      gforce = 1
      do while(gforce.gt.0)

         if (gforce.eq.1) then
            write(*,*)'--------------------------'
            do i = 1,7

               write(*,'(a,I2,a,a)')'    (',i,')   ',titles(i)
            end do
!            write(*,*)'    (2)   Cm    vs. Time'
!            write(*,*)'    (3)   Cd    vs. Time'
!            write(*,*)'    (4)   Alpha vs. Time'
!            write(*,*)'    (5)   h/c   vs. Time'
!            write(*,*)'    (6)   Cl    vs. Alpha'
            write(*,*)'--------------------------'

            gtype = 0
            do while ((gtype.lt.1).or.(gtype.gt.7))
               write(*,*)' Pick Parameter to Plot'
               read(*,*)gtype
            end do

            !---  Fill Temp Arrays With Chosen Values
               do j=1,nsets
                  do i=1,nstep(j)
                     !-- Fill xtempa
                        if ((1.le.gtype).and.(gtype.le.6)) then
                           xtempa(i,j)=time(i,j)
                        else if ((7.eq.gtype)) then
                           xtempa(i,j)=alpha(i,j)
                        end if

                     !-- Fill ytempa
                        if ((1.eq.gtype))then
                           ytempa(i,j)=cl(i,j)
                        else if ((2.eq.gtype))then
                           ytempa(i,j)=CMLE(i,j)
                        else if ((3.eq.gtype))then
                           ytempa(i,j)=CMEA(i,j)
                        else if ((4.eq.gtype))then
                           ytempa(i,j)=cd(i,j)
                        else if ((5.eq.gtype))then
                           ytempa(i,j)=alpha(i,j)
                        else if ((6.eq.gtype))then
                           ytempa(i,j)=h(i,j)
                        else if ((7.eq.gtype))then
```

```
                    ytempa(i,j)=h(i,j)
                 end if
              end do
           end do

        end if

!---  Find Limits to Plot
        if (gforce.eq.2) then
           write(*,*)' X-axis Limits (xmin, xmax)'
           read(*,*)xmin,xmax
           write(*,*)' Y-axis Limits (xmin, xmax)'
           read(*,*)ymin,ymax
        else if (gforce.eq.1) then
           !------  x-axis
              xmin = 0
              xmax = 0
              if ((1.le.gtype).and.(gtype.le.6)) then
                 do j = 1,nsets
                    xmax = max(xmax,time(nstep(j),j))
                 end do
              else if ((7.eq.gtype)) then
                 do j = 1,nsets
                    do i = 1,nstep(j)
                       xmin = min(xmin,(xtempa(i,j))*1.1)
                       xmax = max(xmax,(xtempa(i,j))*1.1)
                    end do
                 end do
              end if

           !------  y-axis
              ymin = 0.
              ymax = 0.
              do j = 1,nsets
                 do i = 1,nstep(j)
                    ymin = min(ymin,(ytempa(i,j))*1.1)
                    ymax = max(ymax,(ytempa(i,j))*1.1)
                 end do
              end do
        end if

        ! Select Graphics Window
           call pgslct(istat(1))
c        CALL PGERAS
           !-- Save as Gif
           if (gforce.eq.3) then
              write(*,*)'File name to Save As (no extension)'
              read(*,*) fn(nsets+1)
              leng = namlen(fn(nsets+1))
              write(fn(nsets+2),'(a,".ps/cps")')fn(nsets+1)(1:leng)
              write(*,*)fn(nsets+2)
              istat(4) = pgopen(fn(nsets+2))
              call pgslct(istat(4))
           end if


           call pgbbuf()
        ! Color Index
           call pgsci(1)
        ! Line Style
           call pgsls(1)
        ! Axis Properties
           just = 0
```

```
          axis = 0

          call PGENV (XMIN, XMAX, YMIN, YMAX, JUST, AXIS)
       ! Label Axes
          call pglab('','',titles(gtype))

       ! Set Line Style
          call pgsls(1)

       do j = 1,nsets
          ! Set Color Index
        call pgsci(j+1)  ! Red
          ! Plot Line
             do i = 1,nstep(j)
                xtemp(i) = xtempa(i,j)
                ytemp(i) = ytempa(i,j)
             end do

             call pgline(nstep(j),xtemp,ytemp)

c            write(title,)

          call pgsch(0.75)
          i = namlen(fn(j))
          write(title,'("- ",A," = ",A,A,A)')color_n(j+1),
     &     '',fn(j)(1:i),''
          write(*,*)title
          call PGMTXT ('T', -real(1+j), 1./20., 0.0, title)
       end do

       call pgebuf()

       if (gforce.eq.3) then
          call pgclos
       end if

      !---  Options
       write(*,*)'----------------------------------'
       write(*,*)'    (0)   Exit'
       write(*,*)'    (1)   Plot Another Parameter'
       write(*,*)'    (2)   Zoom Current Plot'
       write(*,*)'    (3)   Save Current Plot'
       write(*,*)'    (4)   Reload Data'
       write(*,*)'----------------------------------'

       gforce = -1
       do while ((gforce.lt.0).or.(gforce.gt.4))
          write(*,*)' Option?'
          read(*,*)gforce
       end do

       if (gforce.eq.4) then
          gforce = 0
          reread = 1
       end if

      end do

       if (reread.eq.1) goto 1000

      call pgslct(istat(1))
      call pgclos
      end
```

```
!-----------------------------------------------------------------------
!---  FUNCTIONS
!-----------------------------------------------------------------------
      INTEGER FUNCTION namlen( filen )

      CHARACTER*72 filen
!-----------------------------------------------------------------------
      namlen = 0
      do i = 72,1,-1
          if ( filen(i:i) .ne. ' ' ) then
              namlen = i
              goto 101
          endif
      enddo
  101 return
      end
!-----------------------------------------------------------------------
```

D.2 Animation Routines

Used to animate output.

D.2.1 Animate r21

```
      PROGRAM  animate
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
!  rev 21
!     -  Remove comments and unnecessary code for Publication
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
!----- Variables
      IMPLICIT NONE
      !- Include Common Variable Definitions
         INCLUDE 'lengths.inc'
         INCLUDE 'airfoil.inc'
         INCLUDE 'wake.inc'
         INCLUDE 'strengths.inc'
         INCLUDE 'motion.inc'
         INCLUDE 'freevort.inc'
         INCLUDE 'forces.inc'
         INCLUDE 'graph_cons.inc'
      !- Array Length Parameters
         INTEGER lcompare
         PARAMETER( lcompare = 3 )
      !- Data Variables
         REAL*8 time(liter)
         REAL*8 vtan(liter)
         REAL*8 vnor(liter)
         REAL*8 trash
         REAL temp
         REAL*8 dt, dx, dy, dist
         INTEGER  gforce, gtype
         INTEGER  nstep_force
      !- Data Counters
         INTEGER  idump1,idump2
         INTEGER  m,t,imax,step
         REAL     tstart, tstop, gpoints
         INTEGER  istart, istop, gshow, grepeat, gzoom
```

```
      !- Code Variables
         CHARACTER*60   fn(4)
         CHARACTER*60   ftemp
         CHARACTER*20   titles(10)
         CHARACTER*8    atitle(10)
         INTEGER  i_ani, i_force, i, j, k, ani, i_ani2
         INTEGER namlen
         INTEGER leng
         INTEGER leng2
      !- Graphics Variables
         INTEGER  pgopen
         INTEGER  istat(10)
         REAL     xmin, xmax, ymin, ymax
         REAL     fxmin, fxmax, fymin, fymax
         REAL     xminv, xmaxv, yminv, ymaxv
       INTEGER  just, axis
       CHARACTER*70   title
       REAL     pgscale, pgscale_vf, pgscale_vf_u, pgscale_vf_v
       REAL     xtemp(liter)
       REAL     ytemp(liter)
       REAL     ztemp(liter)
       INTEGER  ltemp
!  Set type of diaplay, 1 or 3 windows
         INTEGER  nwindow
      !  Viewports
         REAL     xmina, xmaxa, ymina, ymaxa
         integer  UNITS
      !  Write Graphics
         integer  savegif
         integer  iter

         CHARACTER*70 garb_c
         real     garb_f
         real*8   garb_d
         integer  garb_i


         integer  nsamp, i_samp
!           lfree = 3001
         REAL*8 xsamp(liter)  !  x-coordinate of phi integration point
         REAL*8 ysamp(liter)  !  y-coordinate of phi integration point
         REAL*8 usamp(liter)  !  x-velocity at phi integration point
         REAL*8 vsamp(liter)  !  y-velocityat phi integration point

!------------------------------------------------------------------------
!----- Format Statements
         INCLUDE 'format.inc'

!------------------------------------------------------------------------
!----- Initalize Variables
      !- File Identifiers
         i_ani    = 11
         i_ani2   = 53
         i_force  = 12
         i_samp      = 52     !  Velocities at the phi integration points

      !- Runtime
         m        = 30
         t        =  1
         step     =  5
         dt       =  0.01
         gpoints  = -1
         gzoom    = -1
```

```
         nodtot   = m

      !- Graphics
         gforce   = 1
         gtype    = 1
         savegif  = 0

         pgscale = -1.0
         pgscale_vf = -1.0

      !- Define Titles
         titles(1) = 'Cl    vs. Time'
         titles(2) = 'Cm    vs. Time'
         titles(3) = 'Cd    vs. Time'
         titles(4) = 'Alpha vs. Time'
         titles(5) = 'h/c   vs. Time'
         titles(6) = 'Cl    vs. Alpha'

!-----------------------------------------------------------------------
!----- Prompt for Input File

         write(*,*)' Name of Dataset (60 char max, no extension)'
         read(*,*)   fn(1)

         write(*,*)' Number of Graphics Windows (1,3)'
         read(*,*)   nwindow

!-----------------------------------------------------------------------
!----- Initialize Graphics
      !- Window 1 -  Airfoil
         istat(1) = pgopen('/xserve')
         if (istat(1).le.0) stop
         call PGASK(.false.)

      if (nwindow.ne.1) then
      !- Window 2 -  CP
         istat(2) = pgopen('/xserve')
         if (istat(2).le.0) stop
         call PGASK(.false.)
      !- Window 3 -  Forces
         istat(3) = pgopen('/xserve')
         if (istat(3).le.0) stop
         call PGASK(.false.)
      end if
!-----------------------------------------------------------------------
!----- Read Force Data
         leng = namlen(fn(1))
         write(*,*)'leng = ',leng
         write(ftemp,'(a,".lft")')fn(1)(1:leng)
         write(*,*)'"',ftemp,'"'
         open(UNIT=i_force, FILE=ftemp, status='unknown')
         i = 1
         do while(.true.)
            ! Read coordinates into x(i) and y(i)
            read(i_force,fmt=58,end=1020) time(i),mothis(i),alpha(i),
     &       cl_s(i),cd_s(i),cmle_s(i)
            i=i+1
         end do
 1020    continue
         nstep_force = i-1
!         write(*,*)'j ',j,' i ',i,' nstep ',nstep(j)
         close(UNIT=i_force)
```

```
!----------------------------------------------------------------------
!----- Set Repeat Loop
         grepeat = 1
      do while(grepeat.gt.0)

!----------------------------------------------------------------------
!----- Read Animation Data

      !- Animation File (*.ani), BINARY
         leng = namlen(fn(1))
         write(*,*)'leng = ',leng
         write(ftemp,'(a,".ani2")')fn(1)(1:leng)
         write(*,*)'"',ftemp,'"'
         open(UNIT=i_ani2,file=ftemp,status='old',FORM='unformatted',
     &    ERR=1030)
         goto 1040
 1030    continue

         WRITE(*,*) ' File ERROR'
         i_ani2 = 0
      !- Animation File (*.ani), ASCII
         leng = namlen(fn(1))
         write(*,*)'leng = ',leng
         write(ftemp,'(a,".ani")')fn(1)(1:leng)
         write(*,*)'"',ftemp,'"'
         open(UNIT=i_ani,file=ftemp,status='old',FORM='formatted',
     &    ERR=9999)
 1040    continue


         leng = namlen(fn(1))
         write(*,*)'leng = ',leng
         write(ftemp,'(a,".samp")')fn(1)(1:leng)
         write(*,*)'"',ftemp,'"'
        open(UNIT=i_samp,file=ftemp,status='unknown',FORM='unformatted')


            write(*,*)'=========================='
            write(*,*)'- File open'
      if (i_ani2.ne.0) then
         read(i_ani2)nodtot
            write(*,*)'   nodtot =',nodtot
         read(i_ani2)nodle
            write(*,*)'   nodle  =',nodle
         read(i_ani2)dt
            write(*,*)'   dt     =',dt
         read(i_ani2)nstep
            write(*,*)'   nstep  =',nstep
         read(i_ani2)idump1
            write(*,*)'   idump1 =',idump1
         read(i_ani2)idump2
            write(*,*)'   idump2 =',idump2
         read(i_ani2)x0
            write(*,*)'   x0     =',x0
         read(i_ani2)y0
            write(*,*)'   y0     =',y0
      else
         read(i_ani,*)nodtot
            write(*,*)'   nodtot =',nodtot
         read(i_ani,*)nodle
            write(*,*)'   nodle  =',nodle
         read(i_ani,*)dt
            write(*,*)'   dt     =',dt
```

```
         read(i_ani,*)nstep
            write(*,*)'   nstep  =',nstep
         read(i_ani,*)idump1
            write(*,*)'   idump1 =',idump1
         read(i_ani,*)idump2
            write(*,*)'   idump2 =',idump2
         read(i_ani,*)x0
            write(*,*)'   x0     =',x0
         read(i_ani,*)y0
            write(*,*)'   y0     =',y0
      end if

            write(*,*)'-------------------------'
            write(*,*)'   interval =',real(nstep)*dt,' (s)'

!------------------------------------------------------------------------
!----- Prompt for Display Options
      if (grepeat.eq.1) then

         write(*,*)'=========================='
         write(*,*)' Display Options'
         write(*,*)'  (1)   Show Specified Time'
         write(*,*)'  (2)   Animate Interval'
         write(*,*)'  (3)   Animate All Data'
         write(*,*)'=========================='

         ani = -1
         ! Error Check Input
         do while ((ani.lt.1).or.(3.lt.ani))
            write(*,*)' Option?'
            read(*,*)ani
         end do

         !- Option One - Show Specified Time
         if (ani.eq.1) then
            ! Get Time To Show
            write(*,*)'Show Time x? (s)'
            read(*,*)tstop
            istop = (tstop/dt)
         !- Option Two - Animate Interval
         else if (ani.eq.2) then
            write(*,*)'Start at time x? (s)'
            read(*,*)tstart
               istart=(tstart/dt)
            write(*,*)'Stop at time x? (s)'
            read(*,*)tstop
               istop= (tstop/dt)
         !- Option Three - Animate All
         else if (ani.eq.3) then
            istart = 0
            istop  = nstep
         end if

!------------------------------------------------------------------------
!----- Set Parameters for Display
         step = idump2
         imax = (imax/step)*step
         write(*,*)'imax   = ',imax
         write(*,*)'istart = ',istart
         write(*,*)'istop  = ',istop

         gzoom = -1  ! Set Default Zoom
      end if
```

```
!-----------------------------------------------------------------------
!----- Find Limits if repeat
      if (grepeat.eq.2) then
         write(*,*)' X-axis Limits (xmin, xmax)'
         read(*,*)xmina,xmaxa
         write(*,*)' Y-axis Limits (xmin, xmax)'
         read(*,*)ymina,ymaxa
         gzoom = 1   ! Set Custom Zoom
      end if

!-----------------------------------------------------------------------
!----- Animate Loop

         iter = 0
         do t = 0,istop,step
            iter = iter + 1

      !---- Read Data
         if (i_ani2.ne.0) then
            read(i_ani2)   garb_i
            read(i_ani2)  gamma(t),gammaw(t)
!            read(i_ani2,58)  time(t),mothis(t),alpha(t),cl_s(t),cd_s(t),
!     &        cmle_s(t)
            read(i_ani2)  garb_d,garb_d,garb_d,garb_d,garb_d,garb_d
            read(i_ani2)  (x(i),i=1,nodtot+2)
            read(i_ani2)  (y(i),i=1,nodtot+2)
            read(i_ani2)  (q(i),i=1,nodtot)

            read(i_ani2)  (xmid(i),i=1,nodtot)
            read(i_ani2)  (ymid(i),i=1,nodtot)
            read(i_ani2)  (cp(i),i=1,nodtot)
            read(i_ani2)  (vtan(i),i=1,nodtot)
            read(i_ani2)  (vnor(i),i=1,nodtot)
            read(i_ani2)  (xvort(i),i=1,t)
            read(i_ani2)  (yvort(i),i=1,t)
            read(i_ani2)  (vort(i),i=1,t)

            !---- Free Stream Vortices
            read(i_ani2)nfsv
            do k =1,7
               read(i_ani2)   (fsvort(i,k),i=1,nfsv)
            end do
         else
            read(i_ani,*)   trash
            read(i_ani,56)  gamma(t),gammaw(t)
!            read(i_ani,58)  time(t),mothis(t),alpha(t),cl_s(t),cd_s(t),
!     &        cmle_s(t)
            read(i_ani,58)  trash,trash,trash,trash,trash,trash
            read(i_ani,56)  (x(i),i=1,nodtot+2)
            read(i_ani,56)  (y(i),i=1,nodtot+2)
            read(i_ani,56)  (q(i),i=1,nodtot)

            read(i_ani,55)  (xmid(i),i=1,nodtot)
            read(i_ani,55)  (ymid(i),i=1,nodtot)
            read(i_ani,55)  (cp(i),i=1,nodtot)
            read(i_ani,55)  (vtan(i),i=1,nodtot)
            read(i_ani,55)  (vnor(i),i=1,nodtot)
            read(i_ani,57)  (xvort(i),i=1,t)
            read(i_ani,57)  (yvort(i),i=1,t)
            read(i_ani,57)  (vort(i),i=1,t)

            !---- Free Stream Vortices
```

```
       read(i_ani,*)nfsv
       do k =1,7
          read(i_ani,57)   (fsvort(i,k),i=1,nfsv)
       end do
    end if

 !---- Sampled Data
       if (pgscale_vf.gt.0) then
          read(i_samp) garb_i,nsamp,
&           (xsamp(i),ysamp(i),usamp(i),vsamp(i),i=1,nsamp)
       end if


 !---- Calculate Panel Angles
       do i=1,nodtot
          dx=x(i+1)-x(i)
          dy=y(i+1)-y(i)
          dist=sqrt(dx*dx+dy*dy)
          sinthe(i)=dy/dist
          costhe(i)=dx/dist
       enddo

 !- Plot Data
 !---- Check If Display Single, Display All, Or Display Range
       gshow = 0
       if ((ani.eq.1).and.(t.eq.istop)) then
          gshow = 1
       else if ((ani.eq.2).and.(istart.le.t).and.(t.le.istop)) then
          gshow = 1
       else if (ani.eq.3) then
          gshow = 1
       end if

 !- Flowfied Plot ----------------------------------------------------
 if (gshow.eq.1) then

 ! Start Buffer

 ! Select Display to plot to

 if (nwindow.eq.1) then
    if (savegif.ne.0) then
       leng = namlen(fn(2))
       leng2 = namlen(fn(3))
       write(fn(4),'(a,"",i5.5".",a,"/",a)')fn(2)(1:leng),iter,
&           fn(3)(1:leng2),fn(3)(1:leng2)
       write(*,*)fn(4)
       istat(4) = pgopen(fn(4))
       if (istat(4).le.0) stop
       call pgslct(istat(4))
    else
       call pgslct(istat(1))
    end if
    CALL PGPAGE
    CALL PGSVP(0.05,0.95,0.5,0.92)
 else
    if (savegif.ne.0) then
       leng = namlen(fn(2))
       leng2 = namlen(fn(3))
       write(fn(4),'(a,"ff",i5.5".",a,"/",a)')fn(2)(1:leng),iter,
&           fn(3)(1:leng2),fn(3)(1:leng2)
       write(*,*)fn(4)
       istat(4) = pgopen(fn(4))
```

```
            if (istat(4).le.0) stop
            call pgslct(istat(4))
         else
            call pgslct(istat(1))
         end if
      end if

      ! Get actual viewport dimensions
       UNITS = 3   ! (Pixels)
       call PGQVP(UNITS, xminv,xmaxv,yminv,ymaxv)

      ! Axes
        call pgsci(1)
      ! Set Up Axes
       if (gzoom.lt.0) then
        xmin = -1.
        xmax = istop*dt+2.5
        ymin = 0.
        ymax = 0.
        ymin = min(ymin,-abs((xmax - xmin)/5))
        ymax = max(ymax, abs((xmax - xmin)/5))
       else
!        if (nwindow.ne.1) then
         if (((xmaxa - xmina)/(xmaxv - xminv)) .gt.
     &      ((ymaxa - ymina)/(ymaxv - yminv))) then
          xmin = xmina
          xmax = xmaxa
          ymin = (ymaxa+ymina)/2.0 -
     &        (xmaxa-xmina)*(ymaxv - yminv)/(xmaxv - xminv)/2.0
          ymax = (ymaxa+ymina)/2.0 +
     &        (xmaxa-xmina)*(ymaxv - yminv)/(xmaxv - xminv)/2.0
         else
          xmin = (xmaxa+xmina)/2.0 -
     &        (ymaxa-ymina)*(xmaxv - xminv)/(ymaxv - yminv)/2.0
          xmax = (xmaxa+xmina)/2.0 +
     &        (ymaxa-ymina)*(xmaxv - xminv)/(ymaxv - yminv)/2.0
          ymin = ymina
          ymax = ymaxa
         end if
       end if

                  just = 1
                  axis = 0
               if (nwindow.ne.1) then
                  call PGENV (XMIN, XMAX, YMIN, YMAX, JUST, AXIS)
               else
                  CALL PGSWIN(xmin, xmax, ymin, ymax)
                  call pgsch(0.5)
                  CALL PGBOX ('BCTSN', 0.0, 0, 'BCTSVN', 0.0, 0)
                  call pgsch(0.75)
               end if
               ! title
            write(title,'("Airfoil at time =",f12.4,"(s)")')real(t)*dt

                  call pglab ('x/c','y/c',title)

               ! Set Character (Arrow) Size
                  call pgsch(0.25)

               ! Mark Center of rotation
                  call pgsci(8) ! orange
                  xtemp(1) = x0
                  ytemp(1) = y0
```

```
            call pgpt(1,xtemp,ytemp,8)
            call pgsci(2) ! red
            xtemp(1) = x0
            ytemp(1) = y0 - mothis(t)
            call pgpt(1,xtemp,ytemp,8)



         ! Mark Wake Vortices
            call pgsci(2) ! Blue
            call pgsch(1.0)
            do j = 1,t
               xtemp(1) = abs(vort(j))
               call pgsch(100.*xtemp(1))
               if (vort(j).gt.0) then
                  call pgsci(3)  ! Green
                  xtemp(1) = 3   ! *
                else if (vort(j).eq.0) then
                  call pgsci(3)  ! Green
                  xtemp(1) = 1   ! .
                else if (vort(j).lt.0) then
                  call pgsci(5)  ! L. Blue
                  xtemp(1) = 2   ! +
               end if
               ! + if CW, x if ccw
               xtemp(2) = xvort(j)
               ytemp(2) = yvort(j)
               call pgpt(1,xtemp(2),ytemp(2),int(xtemp(1)))
!               call pgpt(1,xvort(j),yvort(j),-1)
            end do

         ! Plot Airfoil
               call pgsch(gpoints)
               call pgsci(2)

               do j = 1,nodtot+1
                  xtemp(j) = x(j)
                  ytemp(j) = y(j)
               end do
               call pgline(nodtot+1,xtemp,ytemp)

               if (gpoints.gt.0) then
                  call pgpt(nodtot,xtemp,ytemp,2)
               end if

!               call pgsci(4)
!               do i = 1,nodtot
!
!                  dx = x(i+1) - x(i)
!                  dy = y(i+1) - y(i)
!                  dist = sqrt(dx**2 + dy**2)
!
!                  xtemp(1) = xmid(i) - dy/dist * 0.005
!                  ytemp(1) = ymid(i) + dx/dist * 0.005
!
!                  call pgpt(1,xtemp,ytemp,2)
!
!               end do

               call pgsci(4)
                  xtemp(1)=x(nodtot+1)
                  xtemp(2)=x(nodtot+2)
```

```
            ytemp(1)=y(nodtot+1)
            ytemp(2)=y(nodtot+2)
         call pgline(2,xtemp,ytemp)
         if (gpoints.gt.0) then
            call pgpt(2,xtemp,ytemp,2)
         end if

      ! Mark Free Stream Vortices
         call pgsci(3) ! Blue
         call pgsch(.25)
         do j = 1,nfsv
            if (fsvort(j,5).ge.0) then
               call pgsci(6)  ! Purple
             else
               call pgsci(7)  ! Yellow
            end if

            xtemp(1) = fsvort(j,1)
            xtemp(2) = fsvort(j,3)

            ytemp(1) = fsvort(j,2)
            ytemp(2) = fsvort(j,4)

            call pgline(2,xtemp,ytemp)

            call pgsch(gpoints)
            if (gpoints.gt.0) then
               call pgpt(2,xtemp,ytemp,2)
            end if
         end do

! Plot Velocity Vectors
   if (pgscale.gt.0) then
      call pgsch(.25)
      do j = 1,nodtot,1

      ! Plot tangential velocities at midpoints
         call pgsci(8) ! Orange
         xtemp(1)=xmid(j)
         xtemp(2)=xtemp(1)+vtan(j)* costhe(j)*pgscale
         ytemp(1)=ymid(j)
         ytemp(2)=ytemp(1)+vtan(j)* sinthe(j)*pgscale
         call pgarro(xtemp(1),ytemp(1),xtemp(2),ytemp(2))
      end do
   end if   ! Velocity Vectors


! Plot Vector Field
   if (pgscale_vf.gt.0) then
      call pgsch(.25)
      do j = 1,nsamp

      ! Plot tangential velocities at midpoints
         call pgsci(4) ! Blue
         xtemp(1)=xsamp(j)
         xtemp(2)=xtemp(1)+usamp(j)*pgscale_vf*pgscale_vf_u
         ytemp(1)=ysamp(j)
         ytemp(2)=ytemp(1)+vsamp(j)*pgscale_vf*pgscale_vf_v
         call pgarro(xtemp(1),ytemp(1),xtemp(2),ytemp(2))
      end do
   end if   ! Velocity Vectors

      ! Set Character (Arrow) Size
```

```
               call pgsch(1.)

 !- Cp Plot -------------------------------------------------------

 if (nwindow.eq.1) then
    if (savegif.ne.0) then
       call pgslct(istat(4))
    else
       call pgslct(istat(1))
    end if
    CALL PGSVP(0.05,0.45,0.05,0.4)
 else
    if (savegif.ne.0) then
       call pgslct(istat(4))
       call pgclos

       leng = namlen(fn(2))
       leng2 = namlen(fn(3))
       write(fn(4),'(a,"cp",i5.5".",a,"/",a)')fn(2)(1:leng),iter,
&          fn(3)(1:leng2),fn(3)(1:leng2)
       write(*,*)fn(4)
       istat(4) = pgopen(fn(4))
       call pgslct(istat(4))
    else
       call pgslct(istat(2))
    end if
 end if

    ! Set Line Style
       call pgsls(1)
    ! Set Color Index
       call pgsci(1)
    ! Set Axis Limits
       xmin = -0.1
       xmax = 1.1
       ymin = 2.
       ymax = -2.
       just = 0
       axis = 0
    if (nwindow.ne.1) then
       call PGENV (XMIN, XMAX, YMIN, YMAX, JUST, AXIS)
    else
       CALL PGSWIN(xmin, xmax, ymin, ymax)
       call pgsch(0.5)
       CALL PGBOX ('BCTSN', 0.0, 0, 'BCTSVN', 0.0, 0)
       call pgsch(0.75)
    end if
    ! Set Title
       write(title,'("Cp at time =",f12.4,"(s)")')real(t)*dt
       call pglab('x/c','-Cp (r=upper g=lower)',title)
    ! Plot Cp Points
       do j = 1,nodtot
          xtemp(j) = xmid(j)
          ytemp(j) = cp(j)
       end do
       call pgsci(2)
       call pgpt(nodtot,xtemp,ytemp,3)
       call pgsci(3)
       call pgpt(nodle,xtemp,ytemp,3)

    ! Mark Free Stream Vortices
       call pgsci(3) ! Blue
       do j = 1,nfsv
```

```
         if ((fsvort(j,6)+fsvort(j,7)).ne.0) then
            if (fsvort(j,7).ne.0) then
               call pgsci(2)
               xtemp(1) = fsvort(j,3)
               xtemp(2) = fsvort(j,3)
            else
               call pgsci(3)
               xtemp(1) = fsvort(j,1)
               xtemp(2) = fsvort(j,1)
            end if

            ytemp(1) = ymin
            ytemp(2) = ymax
            call pgline(2,xtemp,ytemp)
         end if

      end do


!- Force Plot -----------------------------------------------------
if (nwindow.eq.1) then
   if (savegif.ne.0) then
      call pgslct(istat(4))
   else
      call pgslct(istat(1))
   end if
   CALL PGSVP(0.55,0.95,0.05,0.4)
else
   call pgslct(istat(3))
end if

!---  Fill Temp Arrays With Chosen Values
do i=1,nstep_force
   !-- Fill xtempa
      if ((1.le.gtype).and.(gtype.le.5)) then
         xtemp(i)=time(i)
      else if ((6.eq.gtype)) then
         xtemp(i)=alpha(i)
      end if

   !-- Fill ytempa
      if ((1.eq.gtype))then
         ytemp(i)=cl_s(i)
      else if ((2.eq.gtype))then
         ytemp(i)=cmle_s(i)
      else if ((3.eq.gtype))then
         ytemp(i)=cd_s(i)
      else if ((4.eq.gtype))then
         ytemp(i)=alpha(i)
      else if ((5.eq.gtype))then
         ytemp(i)=mothis(i)
      else if (6.eq.gtype) then
         ytemp(i)=cl_s(i)
      end if
end do

!- Find Limits to Plot
if (gforce.eq.1) then
   !------  x-axis
      fxmin = 0
      fxmax = 0
      if ((1.le.gtype).and.(gtype.le.5)) then
         fxmax = max(fxmax,time(nstep_force))
```

```
      else if ((6.eq.gtype)) then
         do i = 1,nstep_force
            fxmin = min(fxmin,(xtemp(i))*1.1)
            fxmax = max(fxmax,(xtemp(i))*1.1)
         end do
      end if

   !------  y-axis
      fymin = 0.
      fymax = 0.
      do i = 3,nstep_force
         fymin = min(fymin,(ytemp(i))*1.1)
         fymax = max(fymax,(ytemp(i))*1.1)
      end do
end if

call pgsch(1.0)

! Color Index
   call pgsci(1)
! Text Scale
   call pgsch(1.0)
! Line Style
   call pgsls(1)
! Axis Properties
   just = 0
   axis = 0
if (nwindow.ne.1) then
   call PGENV (fxmIN, fxmAX, fymIN, fymAX, JUST, AXIS)
else
   CALL PGSWIN(fxmIN, fxmAX, fymIN, fymAX)
   call pgsch(0.5)
   CALL PGBOX ('BCTSN', 0.0, 0, 'BCTSVN', 0.0, 0)
   call pgsch(0.75)
end if
! Label Axes
   call pglab('','',titles(gtype))

! Set Line Style
   call pgsls(1)

! Set Color Index
call pgsci(2)  ! Red
! Plot Line
call pgline(nstep_force,xtemp,ytemp)

write(title,'("- ",A," = ",A)')color_n(1+1),fn(1)(1:i)

!- Plot Vertical line at time(t)
   call pgsci(4)  ! Blue
   xtemp(1) = time(t)
   xtemp(2) = time(t)
   ytemp(1) = fymin
   ytemp(2) = fymax
   call pgline(2,xtemp,ytemp)

   if (savegif.ne.0) then
      call pgslct(istat(4))
      call pgclos
   end if

end if   ! if gshow
```

```
      end do
      if (i_ani.ne.0) then
         close(UNIT=i_ani2)
      else
         close(UNIT=i_ani)
      end if

      if (savegif.ne.0) then
         savegif = 0
      end if

!----------------------------------------------------------------------
!----- Options Menu

         write(*,*)'=========================='
         write(*,*)' Program Options'
         write(*,*)' '
         write(*,*)'  (1 )   Plot Again'
         write(*,*)'  (2 )   Zoom Current Plot'
         write(*,*)'  (3 )   Replay Animation'
         write(*,*)'  (4 )   Toggle Points'
         write(*,*)'  (5 )   Toggle Velocity'
         write(*,*)' '
         write(*,*)'  (6 )   Change Force Plot'
         write(*,*)' '
         write(*,*)'  (7 )   Step Back by ',idump2*dt,'(s)'
         write(*,*)'  (8 )   Step Forward by ',idump2*dt,'(s)'
         write(*,*)' '
         write(*,*)'  (9 )   Save Graphics'
         write(*,*)' '
         write(*,*)'  (11)   Plot Vector Field'
         write(*,*)' '
         write(*,*)'  (20)   Exit'
         write(*,*)'=========================='

      ! Error Check Input
         grepeat = -1
         do while ((grepeat.lt.1).or.(20.lt.grepeat))
            write(*,*)' Option?'
            read(*,*)grepeat
!            if ((grepeat.lt.10).and.(5.lt.grepeat)) then
!               grepeat=-1
!            end if
         end do

      ! Check Exit Case (Exits for grepeat.le.0 )
         if (grepeat.eq.20)   grepeat = 0

      ! Check For Toggle Points
         if (grepeat.eq.4) then
            if (gpoints.le.0) then
               write(*,*)' Scale Factor for Points? (+ real)'
               read(*,*)gpoints
            else
               gpoints = -1
            end if
         end if

      ! Check for Toggle Velocity
         if (grepeat.eq.5) then
            if (pgscale.le.0) then
               write(*,*)' Scale Factor for Vectors? (+ real)'
               read(*,*)pgscale
```

```
          else
             pgscale = -1
          end if
       end if

    ! Check for Vector Field
       if (grepeat.eq.11) then
!           if (pgscale_vf.le.0) then
             write(*,*)' Scale Factor for Vectors? (+real=on,-1=off)'
             read(*,*)pgscale_vf
             write(*,*)' Scale Factor for Vectors (x)? (+ real)'
             read(*,*)pgscale_vf_u
             write(*,*)' Scale Factor for Vectors (y)? (+ real)'
             read(*,*)pgscale_vf_v
       end if

    ! Check for Change force plot
       if (grepeat.eq.6) then
       !---  Options
          write(*,*)'----------------------------------'
          write(*,*)'    (0)   Exit'
          write(*,*)'    (1)   Plot Another Parameter'
          write(*,*)'    (2)   Zoom Current Plot'
          write(*,*)'----------------------------------'

          gforce = -1
          do while ((gforce.lt.0).or.(gforce.gt.3))
             write(*,*)' Option?'
             read(*,*)gforce
          end do

          ! Check gforce
             if (gforce.eq.1) then
                write(*,*)'-------------------------'
                do i = 1,6

                   write(*,'(a,I2,a,a)')'    (',i,')   ',titles(i)
                end do
!                write(*,*)'    (2)   Cm    vs. Time'
!                write(*,*)'    (3)   Cd    vs. Time'
!                write(*,*)'    (4)   Alpha vs. Time'
!                write(*,*)'    (5)   h/c   vs. Time'
!                write(*,*)'    (6)   Cl    vs. Alpha'
                write(*,*)'-------------------------'

                gtype = 0
                do while ((gtype.lt.1).or.(gtype.gt.6))
                   write(*,*)' Pick Parameter to Plot'
                   read(*,*)gtype
                end do
             end if

             if (gforce.eq.2) then
                write(*,*)' X-axis Limits (fxmin, fxmax)'
                read(*,*)fxmin,fxmax
                write(*,*)' Y-axis Limits (xmin, xmax)'
                read(*,*)fymin,fymax
             end if
       end if


    ! Check for Step forward or backwards
       if ((7.le.grepeat).and.(grepeat.le.8)) then
```

```
          ! Set Show Single Time
             ani = 1
          if (grepeat.eq.7) istop = istop - idump2
          if (grepeat.eq.8) istop = istop + idump2
       end if

    !-- Save as Graphics
       if (grepeat.eq.9) then
          grepeat = 3
          savegif = nwindow

          write(*,*)'path to save to?'
          read(*,*) fn(2)
          write(*,*)'Format to Save to? (gif,ps,cps)'
          read(*,*) fn(3)
          iter = 1
          leng = namlen(fn(2))
          leng2 = namlen(fn(3))
          write(fn(4),'(a,i5.5".",a,"/",a)')fn(2)(1:leng),iter,
     &       fn(3)(1:leng2),fn(3)(1:leng2)
          write(*,*)fn(4)
       end if


!------------------------------------------------------------------------
!----- Repeat
      end do

!------------------------------------------------------------------------
!----- Close Open Graphics Windows

      if (nwindow.ne.1) then
      call pgslct(istat(3))
      call pgclos
      call pgslct(istat(2))
      call pgclos
      end if
      call pgslct(istat(1))
      call pgclos

!------------------------------------------------------------------------
!----- End Program

 9999 continue
      end

!------------------------------------------------------------------------
!---  FUNCTIONS
!------------------------------------------------------------------------
      INTEGER function namlen( filen )
      IMPLICIT NONE
      CHARACTER*72 filen
      INTEGER i

      namlen = 0
      do i = 72,1,-1
          if ( filen(i:i) .eq. ' ' ) then
              namlen = i-1
          endif
      enddo
      end
!------------------------------------------------------------------------
```

VITA

Aaron M. McClung

Candidate for the degree of

Masters of Science

Thesis: DEVELOPMENT AND VALIDATION OF AN UNSTEADY PANEL CODE TO MODEL AIRFOIL AEROMECHANICAL RESPONSE

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education: Graduated from Newman Smith High School, Carrollton, Texas in May 1996; received Bachelor of Science in Aerospace Engineering from Oklahoma State University, Stillwater, Oklahoma in July 2000; Completed the requirements for the Masters of Science in Mechanical Engineering at Oklahoma State University in July 2004.

Experience: Employed by Sequoyah Engineering, Inc. of Oklahoma City, Oklahoma, as a staff engineer from May 2000 to December 2001; employed by Oklahoma State University Department of Mechanical and Aerospace Engineering, Stillwater, Oklahoma, as a Graduate Teaching Assistant from January 2002 to December 2003; employed by Oklahoma State University Department of Mechanical and Aerospace Engineering, Stillwater, Oklahoma, as a Graduate Research Assistant from June 2002 to May 2004.

Professional Memberships: Association for Unmanned Vehicle Systems International