

(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A274 867



DTIC
ELECTE
JAN 25 1994
S C D



THESIS

COMPUTATIONAL INVESTIGATION OF THE
COMPRESSIBLE DYNAMIC STALL CHARACTERISTICS OF
THE SIKORSKY SSC-A09 AIRFOIL

by

Thomas A. Johnston

September, 1993

Thesis Advisor:
Co-Advisor:

M. F. Platzer
J. A. Ekaterinaris

Approved for public release; distribution is unlimited.

94-02052



94 1 24 059

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified			1b Restrictive Markings		
2a Security Classification Authority			3 Distribution/Availability of Report		
2b Declassification/Downgrading Schedule			Approved for public release; distribution is unlimited.		
4 Performing Organization Report Number(s)			5 Monitoring Organization Report Number(s)		
6a Name of Performing Organization Naval Postgraduate School		6b Office Symbol (if applicable) AA	7a Name of Monitoring Organization Naval Postgraduate School		
6c Address (city, state, and ZIP code) Monterey CA 93943-5000			7b Address (city, state, and ZIP code) Monterey CA 93943-5000		
8a Name of Funding/Sponsoring Organization		6b Office Symbol (if applicable)	9 Procurement Instrument Identification Number		
Address (city, state, and ZIP code)			10 Source of Funding Numbers		
			Program Element No	Project No	Task No
			Work Unit Accession No		
11 Title (include security classification) COMPUTATIONAL INVESTIGATION OF THE COMPRESSIBLE DYNAMIC STALL CHARACTERISTICS OF THE SIKORSKY SSC-A09 AIRFOIL					
12 Personal Author(s) Johnston, Thomas A.					
13a Type of Report Master's Thesis		13b Time Covered From To	14 Date of Report (year, month, day) September 1993		15 Page Count 261
16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.					
17 Cosati Codes			18 Subject Terms (continue on reverse if necessary and identify by block number)		
Field	Group	Subgroup	Dynamic Stall, Supercritical Airfoil, Navier-Stokes Solution, Shock-induced Boundary Layer Separation		
19 Abstract (continue on reverse if necessary and identify by block number)					
Steady and unsteady two-dimensional flowfield analysis was conducted for a Sikorsky SSC-A09 airfoil in compressible, high Reynolds number flows. Limited verification with experimental measurement was achieved. Computational methods included a steady, linear panel method with compressibility corrections; a laminar and turbulent boundary layer method; an unsteady, linear panel method; and a numerical solution method of the thin layer, compressible, Navier-Stokes equations using a body-fitted C-type computational grid. The Baldwin-Lomax, two-layer, zero-equation turbulence model was used. Wind tunnel wall interference effects were ignored. Steady and unsteady airloads and instantaneous flow pictures are presented. In steady flow with little or no separation, computed lift, drag, pitching moment, and skin friction coefficients, as well as displacement thickness and boundary layer velocity profiles at several angles-of-attack were generally found to be in good agreement with experimental data.					
20 Distribution/Availability of Abstract _X_ unclassified/unlimited _ same as report _ DTIC users			21 Abstract Security Classification Unclassified		
22a Name of Responsible Individual Platzer, M. F.			22b Telephone (include Area Code) 408-656-2058		22c Office Symbol AA/PI

DD FORM 1473.84 MAR

83 APR edition may be used until exhausted

security classification of this page

All other editions are obsolete

Unclassified

Approved for public release; distribution is unlimited.

COMPUTATIONAL INVESTIGATION OF THE
COMPRESSIBLE DYNAMIC STALL CHARACTERISTICS OF
THE SIKORSKY SSC-A09 AIRFOIL

by

Thomas A. Johnston
Commander, United States Navy
B.S., United States Naval Academy, 1978

Submitted in partial fulfillment
of the requirements for the degree of

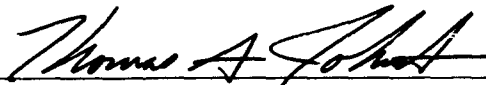
MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

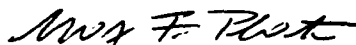
September 1993

Author:

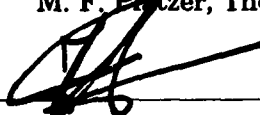


Thomas A. Johnston

Approved by:



M. F. Platzer, Thesis Advisor



J. A. Ekaterinaris, Co-Advisor



D. J. Collins, Chairman

Department of Aeronautics and Astronautics

ABSTRACT

Steady and unsteady two-dimensional flowfield analysis was conducted for a Sikorsky SSC-A09 airfoil in compressible, high Reynolds number flows. Limited verification with experimental measurement was achieved. Computational methods included a steady, linear panel method with compressibility corrections; a laminar and turbulent boundary layer method; an unsteady, linear panel method; and a numerical solution method of the thin layer, compressible, Navier-Stokes equations using a body-fitted C-type computational grid. The Baldwin-Lomax, two-layer, zero-equation turbulence model was used. Wind tunnel wall interference effects were ignored. Steady and unsteady airloads and instantaneous flow pictures are presented. In steady flow with little or no separation, computed lift, drag, pitching moment, and skin friction coefficients, as well as displacement thickness and boundary layer velocity profiles at several angles-of-attack were generally found to be in good agreement with experimental data.

DTIC QUALITY INSPECTED 8

Accession For	
NTIS CRA&I	☒
DTIC TAB	☒
Unannounced	☒
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	STEADY, LINEAR PANEL CODE	5
	A. POTENTIAL FLOW THEORY/BACKGROUND	5
	1. Reference Frame	6
	2. Airfoil Nomenclature	8
	3. Singularity Distribution	9
	4. Influence Coefficients	10
	5. Coefficient of Pressure	10
	6. Force and Moment Coefficients	11
	7. Thin Airfoil Theory Prediction	13
	8. The Supercritical Airfoil	14
	B. CODE VALIDATION	15
	1. Computer Codes	15
	2. The NACA 0012 Symmetric Airfoil	16
	3. The Eppler E585 Airfoil	17
	C. THE SIKORSKI SSC-A09 SUPERCRITICAL AIRFOIL	18
	1. Airfoil Geometry	18
	2. Lorber and Carta Experimental Data	18
	3. Panel Number Effects on Force and Pitching Moment Coefficients	19
	4. Force and Pitching Moment Results	20
III.	CEBECI 2-D LAMINAR & TURBULENT BOUNDARY LAYER CODE	
	A. SHEAR LAYER THEORY/BACKGROUND	37

1.	Turbulence Model	40
2.	Transition Model	41
B.	BL2D.F OVERVIEW	42
1.	Program Hints	43
C.	The NACA 0012 AIRFOIL	44
D.	THE SIKORSKI SSC-A09 AIRFOIL	45
IV.	UNSTEADY, LINEAR PANEL CODE	60
A.	THEORY/BACKGROUND	60
1.	Flow Model	60
2.	Boundary Conditions	61
3.	Solution Scheme	61
B.	THE NACA 0012 AIRFOIL	62
C.	THE SIKORSKY SSC-A09 AIRFOIL	63
1.	Ramp Motion	63
2.	Sinusoidal Motion	64
V.	NAVIER-STOKES CODE	74
A.	THEORY/BACKGROUND	74
B.	NUMERICAL SCHEME	75
1.	Boundary Conditions	78
2.	Numerical Implementation	79
3.	Computational Grid	80
4.	Program NS.F User's Guide	80
C.	SIKORSKY SSC-A09 RESULTS	80
1.	Steady State Motion	80
2.	Unsteady Motion	83
VI.	CONCLUSIONS AND RECOMMENDATIONS	164

LIST OF REFERENCES	167
APPENDIX A	169
A. BASIC UNIX COMMANDS	169
B. PROGRAM AIRFOIL.F	170
C. PROGRAM PANEL.F	171
D. PROGRAM BL2D.F	172
E. VISUALIZATION ROUTINES	173
F. AIRFOIL.F & PANEL.F & BL2D.F SOURCE CODES	176
APPENDIX B	189
A. UPOT.IN NAME LIST	189
B. PROGRAM UPOT.F SOURCE CODE	190
APPENDIX C	211
A. NS.IN NAME LIST	211
B. BATCH & GRAPHICS INTERFACE CODES	212
C. PROGRAM NS.F SOURCE CODE	220
APPENDIX D	247
A. MICHEL'S EMPIRICAL CORRELATION METHOD	247
INITIAL DISTRIBUTION LIST	252

TABLE OF SYMBOLS

a	Speed of Sound
A	Reduced Pitch Rate; $= (\dot{\alpha}c/2U)$
c	Airfoil Chord length
C_d	Section Pressure Drag Coefficient
C_F	Skin Friction Coefficient
C_l	Section Lift Coefficient
C_m	Section Moment Coefficient
C_p	Pressure Coefficient; $= (P-P_\infty)/q$
C_x	X-Force Coefficient
C_y	Y-Force Coefficient
e	Total Energy per Unit Volume
F	Inviscid Flux Vector, ξ
G	Inviscid Flux Vector, ζ
k	Reduced Frequency; $= (\omega c/2U)$
M	Mach Number
$\hat{n}$	Unit Normal Vector
N	Total Number of Panels
p	Pressure
P_r	Prandtl Number
q	Dynamic Pressure; $= \frac{1}{2}\rho U^2$
$q(s)$	Source Strengths
Q	Conservative Variables Vector
r	Scalar Distance between two points
Re	Reynolds Number
S	Viscous Fluxes
$\hat{t}$	Unit Tangential Vector
u	Velocity Component, x-direction
U	Freestream Velocity Magnitude
v	Velocity Component, y-direction
α	Geometric Angle-of-Attack
$\alpha_{L=0}$	Angle-of-Attack at Zero Lift

γ	Ratio of Specific Heats
$\gamma(s)$	Vortex Strengths
γ_{tr}	Intermittency Factor
Γ	Circulation
δ^*	Displacement Thickness
ϵ_m	Turbulent Eddy-Viscosity
μ	Dynamic Viscosity
ν	Kinematic Viscosity
ρ	Density
ϕ	Velocity Potential
ω	Oscillation Frequency

Subscript Indices:

i, j	Indicator for Airfoil Panels and Nodes
n	Normal Component
k	Time indice
t	Tangential Component
tr	Transition
x	X-Component
y	Y-Component
∞	Free Stream Static

Operators:

∂	Partial Derivative
∇	Gradient
∇^2	Laplacian
Σ	Summation
$\int$	Integral
$\oint$	Surface Integral
U_{xx}	Second Derivative with Respect to X
U_{yy}	Second Derivative with Respect to Y

I. INTRODUCTION

Historically aeronautical engineers have had only wind tunnel and flight test experiments to validate aerodynamic theory, often at great expense. In today's world of powerful supercomputers and advanced personal computers with vast memory capability, flowfield solutions once thought impossible or prohibitively expensive are becoming feasible in this new age of Computational Fluid Dynamics (CFD). CFD has become an effective research tool in understanding complicated fluid dynamics phenomena. Indeed, as illustrated in Figure 1.1, the Theory, Experiment, and CFD triad complement each other as

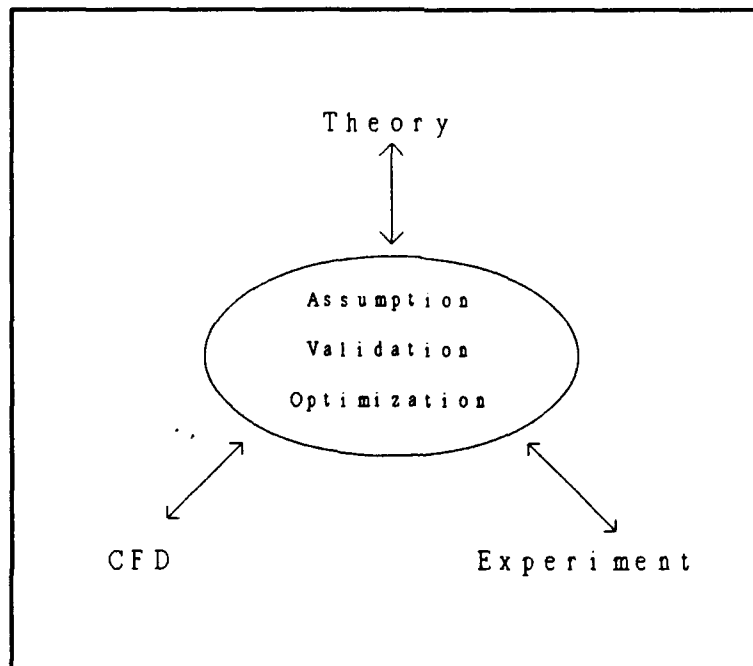


Figure 1.1

well as add a new dimension to the validation process. Experiment and CFD can now be used to complement and optimize each other. CFD results allow in-depth understanding of many physical processes. Transition and turbulence models can be verified and code robustness (convergence time span) can be optimized. Within the CFD framework, it was once thought that the full Navier-Stokes equations would have to be solved to obtain realistic flows over an airfoil executing maneuvers in a viscous, compressible medium. Numerical scheme accuracy and convergence rates are complicated by the various length scales of the viscous effects near the airfoil and those of the surrounding inviscid flow field. It has been proven that more cost effective solution methods can be employed that make realistic simplifications to the governing equations and allow moving away from the supercomputer to the personal computer thus yielding beneficial results at greatly reduced time and cost. Ultimately, the overall goal would be the completion of the design process using only CFD methods.

A current field of intense investigation is the aerodynamics of a rapidly pitching airfoil. Two effects are of major interest:

- Augmented lift created during dynamic stall while performing aircraft combat maneuvers (ACM) in high Reynolds number flows.
- Dynamic stall on a retreating helicopter rotor blade during high-speed forward flight.

Dynamic lift and stall are dominated by the generation of a vortex near the leading edge of the suction surface and its subsequent convection over the airfoil surface. This sequence of events is discussed and shown in great detail in Chapter V.

The intent of this thesis is the CFD investigation of the unsteady aerodynamics of a Sikorsky SSC-A09 airfoil undergoing high pitch rate maneuvers. The experimental results of Lorber and Carta [Ref. 11] are used for validation of the computed solutions. The investigation goals are:

- Determine the influence of the leading edge stall vortex on the unsteady aerodynamic response during and after stall.
- Determine the location of any separation bubbles.
- Determine the location and extent of the boundary layer transition.
- Determine compressibility effects in inviscid and viscous flows.
- Determine the effect of any supersonic regions and shock waves created during pitch up ramp or sinusoidal maneuvers.
- Accurately predict pressure loads, forces, and moments.
- Determine the most efficient and cost effective CFD approach that achieves the desired level of accuracy.

In the following sections the methods which were used to analyze the above flow phenomena are presented first. A presentation of the numerical results and comparisons with experiment follows. Each section includes an Appendix which contains a complete user's guide for the reader who wishes to apply the codes to similar problems. Finally, a discussion of all the results is presented and some conclusions with recommendations for future research are given.

II. STEADY, LINEAR PANEL CODE

A. POTENTIAL FLOW THEORY/BACKGROUND

The flow field is assumed to be steady, incompressible, inviscid and irrotational. A steady flow field implies the fluid velocity and pressure depend only on the spatial coordinates and not on time. Flow field incompressibility implies that the divergence (the time rate of change of volume of a moving fluid element per unit volume) of the velocity vector is zero as indicated in Equation 2.1, and that density is a constant throughout.

$$\nabla \cdot \vec{V} = 0 \quad (2.1)$$

Flow field irrotationality implies that vorticity is zero everywhere, Equation 2.2, and that a scalar function must exist such that the velocity is given by the scalar function's gradient as shown in Equation 2.3.

$$\nabla \times \vec{V} = 0 \quad (2.2)$$

$$\nabla \phi = \vec{V} \quad (2.3)$$

Consequently, irrotational flows are often described as 'potential flows'.

A flow field that is both incompressible and irrotational must satisfy Laplace's equation:

$$\nabla^2 \phi = \phi_{xx} + \phi_{yy} = 0 \quad (2.4)$$

Since Laplace's equation is a linear homogeneous second order partial differential equation, the principle of superposition holds. Complicated flows can be created by linearly combining elementary flows that are both incompressible and irrotational. Uniform, source, and vortex flows are examples that meet these conditions (Anderson [Ref.2]).

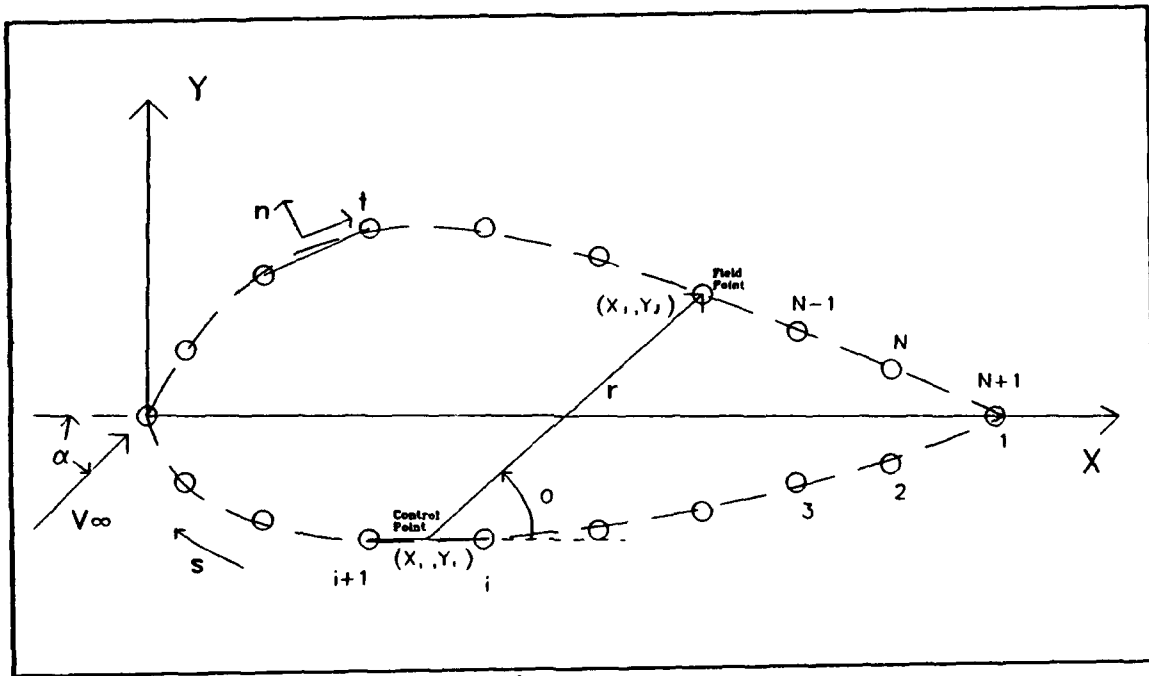


Figure 2.1
Airfoil Geometry and Coordinate System

1. Reference Frame

The two-dimensional airfoil geometry and (x,y) and (r,θ) coordinate systems are described in Figure 2.1. The airfoil surface is divided into a number (N) of straight line

segments normally called 'panels'. $N+1$ surface points, normally called nodes, distinguish the N panels. Numbering convention starts from the lower trailing edge and proceeds clockwise around the airfoil making the first and last point the same. Panel length is arbitrary, but enforcement of the trailing edge Kutta condition (the trailing edge flow must depart smoothly since it is a stagnation point) requires that the first and last panel length be the same. Unit normal vectors, $\hat{n}$, are perpendicular, positive outward from the panel surface. Unit tangent vectors, $\hat{t}$, are parallel to the panel surface, positive in the clockwise direction.

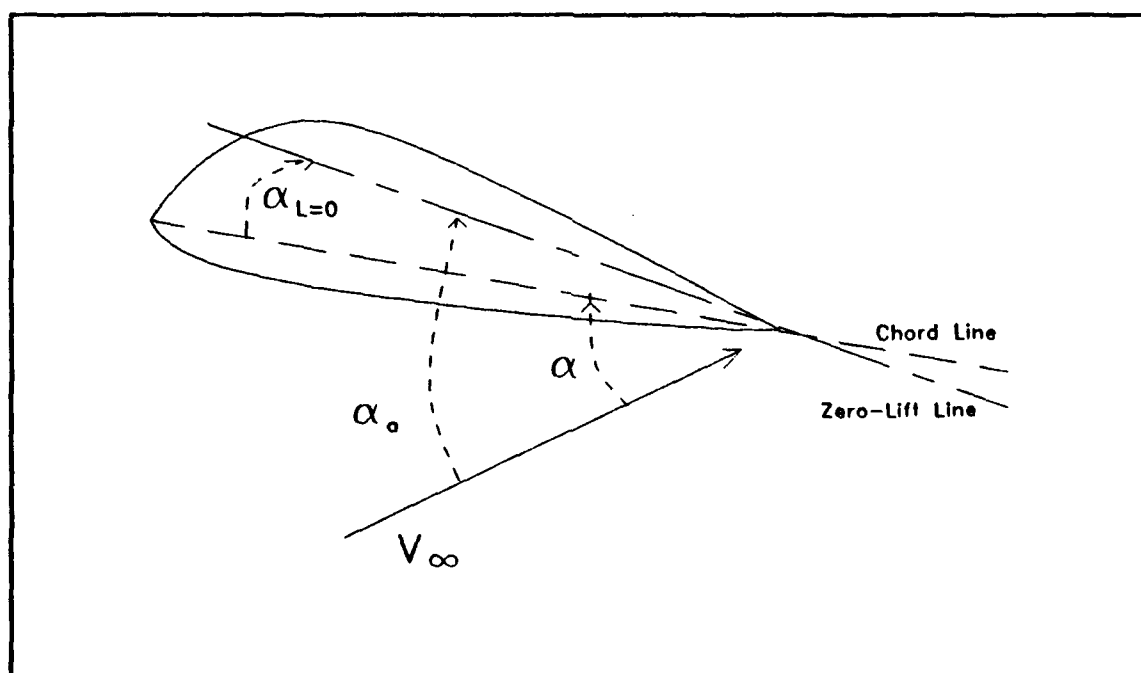


Figure 2.2
Angle-of-Attack Standardization

2. Airfoil Nomenclature

Standard airfoil nomenclature is used as displayed in Figure 2.2. Summary of nomenclature (Kuethe and Chow [Ref. 10]):

- **Chord Line.** The straight line connecting the leading and trailing edges.
- **Chord.** The distance between the leading and trailing edges along the chord line.
- **Mean Camber Line.** The locus of points one-half way between the upper and lower surface measured perpendicular to the mean camber line itself.
- **Symmetric Airfoil.** An airfoil where the mean camber and chord lines are the same.
- **Aerodynamic Center.** The point on the airfoil where the moment is independent of angle-of-attack.
- **Center of Pressure.** The location where the resultant of a distributed load effectively acts on a body. The point about which the aerodynamic moment is zero.
- **Geometric Angle-of-Attack (α).** The angle between V_∞ and the chord line.
- **Zero-Lift Line.** A line on the airfoil parallel to the flight path and passing through the trailing edge when the airfoil is oriented to create zero lift. The zero-lift and chord line are the same for a symmetric airfoil.

- **Angle-of-Attack at Zero Lift ($\alpha_{L=0}$).** The angle between the chord and zero-lift lines.
- **Absolute Angle-of-Attack (α_a).** The angle between V_∞ and the zero-lift line.

$$\alpha_a = \alpha - \alpha_{L=0} \quad (2.5)$$

3. Singularity Distribution

The airfoil velocity potential (Φ) is determined by decomposing the potential flow field into a free stream flow, and placing a source and vortex distribution at each control point (mid point) of each panel. Vortex flows provide circulation/lift and here vortex strength (γ) is fixed. Source flows accurately represent body thickness. Source distributions (q) are allowed to vary from panel to panel. The total potential is described below. These integrals are calculated along the surface contour s in polar coordinates.

$$\Phi_{total} = \phi_\infty + \phi_{source} + \phi_{vortex} \quad (2.6)$$

$$\phi_\infty = V_\infty \times [x \cos\alpha + y \sin\alpha] \quad (2.7)$$

$$\phi_{source} = \int_s \left\{ \frac{q(s)}{2\pi} \ln r \right\} ds \quad (2.8)$$

$$\phi_{vortex} = - \int_s \left\{ \frac{\gamma(s)}{2\pi} \theta \right\} ds \quad (2.9)$$

Integration is performed on each panel along a straight line where q_i and γ are constant and then all the panels summed. The velocity is then obtained from $\nabla\phi$.

4. Influence Coefficients

Influence coefficients provide an algebraic system of linear simultaneous equations that ease numerical solution. An influence coefficient is defined by the velocity induced at a field point (on the airfoil surface) by a unit strength singularity (Source and Vortex) distribution on one panel. Nowak [Ref. 13], Teng [Ref. 15], and Tuncer [Ref. 16] provide detailed analysis of geometrical quantities, equations and the numerical solution scheme.

a. Boundary Conditions

Two boundary conditions must be satisfied. The first is the flow tangency condition at all control points (the mid point of each panel). This is accomplished by requiring the normal component of velocity at the control point to be zero for all panels. The second, the Kutta condition requires smooth flow leaving the trailing edge, and is accomplished by equating the upper and lower pressures at the trailing edge. This is enforced by equating the tangential velocities on the first and N^{th} panel.

5. Coefficient of Pressure (C_p)

Once the source strengths (q_i) and vortex strength (γ) are calculated, the normalized velocity $(V_{\text{total}}/V_\infty)_i$ is computed

at each control point. Using Bernoulli's equation, Equation 2.10, the incompressible flow Coefficient of Pressure is computed.

$$C_p = \frac{P - P_\infty}{q_\infty} = 1 - \left\{ \frac{V_{total}}{V_\infty} \right\}^2 \quad (2.10)$$

a. Pressure Compressibility Correction

For low Mach number flows, less than $M=.3$, the density variation in an inviscid flow is negligible (less than a 5% variation, Anderson [Ref. 2]). For higher, subsonic ($M_\infty < .7$) Mach number flows, a compressibility correction to the incompressible data is achieved by using the 'Prandtl-Glauert' rule derived from small perturbation, linearized velocity potential theory:

$$C_{P_{comp}} = \frac{C_{P_{incomp}}}{\sqrt{1 - M_\infty^2}} \quad (2.11)$$

6. Force and Moment Coefficients

The force and moment coefficients are computed by integration/summation of the pressure distribution assuming a constant C_p on each panel. The total force on a single panel would be $C_{p_i} \cdot ds$. Figure 2.3 details the required geometry. Airfoil-fixed forces for panel i are:

$$\sin\beta = \frac{dy}{ds} \quad \cos\beta = \frac{dx}{ds} \quad (2.12)$$

$$C_{F_{x_i}} = C_{P_i} \times ds \times \sin\beta = C_{P_i} \times dy \quad (2.13)$$

$$C_{F_{y_i}} = C_{P_i} \times ds \times \cos\beta = C_{P_i} \times dx \quad (2.14)$$

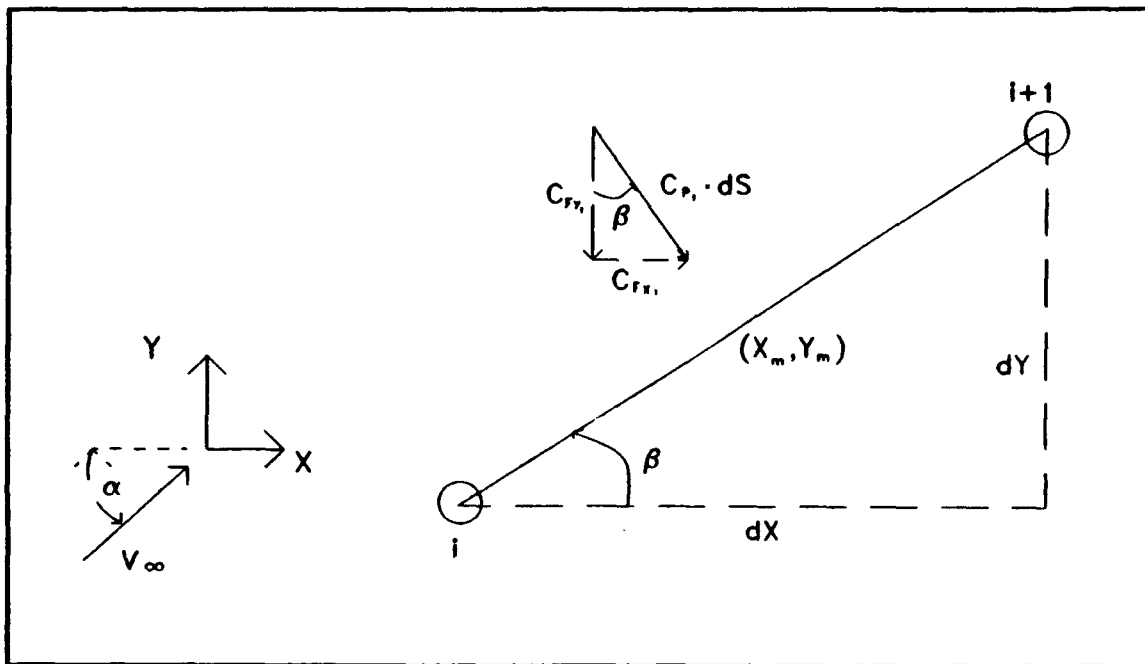


Figure 2.3
Force and Moment Geometry

Integration of forces over N panels with respect to the airfoil-fixed coordinate system (negative sign on C_{F_x} due to the sign convention of positive tangential velocities in the clockwise direction around the airfoil) are shown in Equations 2.14 and 2.15:

$$C_{F_x} = - \sum_{i=1}^N C_{P_i} \{ y_{i+1} - y_i \} \quad (2.15)$$

$$C_{F_y} = \sum_{i=1}^N C_{P_i} \{ x_{i+1} - x_i \} \quad (2.16)$$

Rotation with respect to the free stream direction (tangent for drag and perpendicular for lift) is described in Equations 2.17 and 2.18:

$$C_l = C_{F_y} \cos \alpha - C_{F_x} \sin \alpha \quad (2.17)$$

$$C_d = C_{F_x} \cos \alpha + C_{F_y} \sin \alpha \quad (2.18)$$

The moment is taken about the quarter-chord point from each control point (x_m, y_m) and summed:

$$C_m = \sum_{i=1}^N C_{P_i} \{ (x_{i+1} - x_i) (x_m - .25) + (y_{i+1} - y_i) y_m \} \quad (2.19)$$

7. Thin Airfoil Theory Prediction

One of the basic assumptions of two-dimensional inviscid theory holds that the flow always closes smoothly and completely around the trailing edge, therefore, integrally producing **zero pressure drag** (d'Alembert's paradox). Drag is primarily due to viscous effects which generate frictional

shear forces at the surface eventually causing flow separation. D'Alembert's paradox is also true for subsonic compressible flow since the compressible and incompressible pressure coefficients differ only by a constant. This can be proven since there is no locally supersonic flow that would create wave drag. Inviscid flow theory has proven to be in good agreement with experiment in the linear region of the $C_{l\alpha}$ curve where there is no flow separation (Anderson [Ref. 2]). Thin airfoil theory predicts that for a symmetric airfoil:

- The lift-curve-slope is 2π .
- The aerodynamic center and the center of pressure are at the quarter-chord point.

For a cambered airfoil:

- The lift-curve-slope is 2π .
- Only the aerodynamic center is at the quarter-chord point and the center of pressure varies with C_l .

8. The Supercritical Airfoil

Airfoil quality and efficiency are measured by its L/D which determines aerodynamic efficiency and C_{Lmax} which determines stall speed and is critically dependent upon airfoil thickness. The supercritical airfoil was the result of Richard T. Whitcomb's (working at The National Aeronautics and Space Administration - NASA) development of two-dimensional turbulent airfoils with good transonic behavior, improved drag divergence Mach numbers, and good low-speed

maximum lift and stall characteristics. The concept was based on obtaining locally supersonic flow on the upper surface with an isentropic recompression. As the airflow approaches the speed of sound, a local area of supersonic flow extending vertically appears over the upper surface. On a conventional airfoil, this flow would terminate in a shock wave at about mid-chord producing significant losses. Separation of the boundary layer is then aggravated by the shock induced pressure rise superimposing on an adverse pressure gradient. The supercritical airfoil allows the shock to position itself significantly aft of mid-chord producing a more even upper surface pressure distribution. The resulting airfoil series was characterized by a large leading edge radius, less curvature across the upper surface middle region (limiting flow acceleration), and aft camber where its influence is a maximum (Harris [Ref. 8]).

B. CODE VALIDATION

1. Computer Codes

Many panel codes based on steady, incompressible, inviscid flow over arbitrary airfoils have been developed. This paper uses versions written, and subsequently modified, by Nowak [Ref. 13] and Teng [Ref. 15]. Required input consists of angle-of-attack in degrees and the number of airfoil panels. Normalized velocities and pressure coefficients at each control point are produced. A complete

users guide for the Airfoil.f and Panel.f programs are provided in Appendix A.

2. The NACA 0012 Symmetric Airfoil

a. Geometry and Output Verification

A NACA 0012 symmetric airfoil was chosen to investigate the effects of increasing panel number and compressibility. Figure 2.4 illustrates a typical 100 panel airfoil generated by the airfoil.f program. Excellent agreement between calculated Pressure Coefficient and results obtained by Anderson [Ref. 2] at 9° angle-of-attack is shown in Figure 2.5. Compressibility effects, an increased suction peak with increasing Mach number, on a symmetric airfoil are also demonstrated.

b. Forces and Moment Comparison

Lift, drag, and pitching-moment coefficient as a function of angle-of-attack and panel number are displayed in Figures 2.6 through 2.8. The number of panels has little effect on calculated lift and only a slight effect on calculated moment. However, Figure 2.8 graphically displays the wide variation of calculated drag as a function of panel number. It is important to note that this 'calculated' drag is not real. In reality, the integral pressure drag should be zero as indicated in section A.7. As this figure illustrates, the suction peak forces cannot be exactly resolved when a summation is made over N discrete panels. Further

investigation, Figure 2.9, reveals that airfoil thickness also plays an important role. The leading edge suction peak is more easily resolved on thicker airfoils. Compressibility effects are displayed in Figures 2.10 through 2.12 - Lift, drag, and pitching-moment coefficient magnitude increase with increasing Mach number.

c. The Aerodynamic Center

Thin airfoil theory predicts the aerodynamic center to be at the quarter-chord point (section A.7). Figure 2.13 illustrates the pitching-moment coefficient as a function of Mach number and pivot point (the point about which all moments are taken). The aerodynamic center was located at 26.05% for both $M=0.2$ and $M=0.4$. Kuethe and Chow [Ref. 10] state that the position of the aerodynamic center is a function of airfoil thickness, geometry (camber), and viscosity. Here the thickness effect is seen as moving the aerodynamic center aft.

3. The Eppler E585 Airfoil

This airfoil was designed for sailplanes in low Reynolds number flows. A 71 panel geometry is displayed in Figure 2.14. The angle-of-attack for zero lift is 5.53° . Good agreement was achieved between the panel code calculation and Eppler's [Ref. 7] measured velocity distributions, Figure 2.15. Only slight variation was identified at the trailing edge that is easily resolved by splining in additional panels. Figures 2.16 through 2.19 display compressibility effects on

this cambered airfoil. Compressibility enhances lift and a more pronounced suction peak is observed.

C. THE SIKORSKY SSC-A09 SUPERCRITICAL AIRFOIL

1. Airfoil Geometry

This is a 9% thick, supercritical airfoil (section A.8) used in the Lorber and Carta experiment [Ref. 11]. The original geometry consisted of 132 surface coordinates (131 panels) as shown in Figure 2.20. The trailing edge was modified, Figure 2.21, to meet Kutta condition requirements: A sharp trailing edge, and the first and last panel having the same length. The resulting surface coordinates were manually entered into the points.dat input file.

2. Lorber and Carta Experimental Data

Lorber and Carta [Ref. 11] completed an experiment studying the aerodynamics of dynamic stall penetration at constant pitch rate and free stream Mach numbers of 0.2 through 0.4 corresponding to a Reynolds number of two through four million using the Sikorsky SSC-A09 airfoil. The two-dimensional tunnel experiment obtained dynamic stall data at conditions representative of full-scale helicopter rotor blades and maneuverable combat aircraft. A 17.3 inch chord wing was oscillated in pitch using both ramp and sinusoid motion. Wind tunnel wall effects were not accounted for.

Detailed aerodynamic response was obtained from 72 miniature pressure transducers and eight surface hot film

gages. Unsteady data included 36 constant speed ramps and nine sinusoidal oscillations. Ramp motion was a modified motion consisting of an initial delay, a constant rate increase to maximum, and then a second delay at maximum. Force and pitching-moment coefficients were determined by integrating pressures over the airfoil using the following:

$$C_N = \frac{1}{qC} \int (P_{low} - P_{up}) dx \quad (2.20)$$

$$C_C = \frac{1}{qC} \int (P_{low} - P_{up}) \frac{dy}{dx} dx \quad (2.21)$$

$$C_M = \frac{1}{qC^2} \int (P_{low} - P_{up}) (x - 0.25c) dx \quad (2.22)$$

$$C_L = C_N \cos \alpha - C_C \sin \alpha \quad (2.23)$$

$$C_D = C_C \cos \alpha + C_N \sin \alpha \quad (2.24)$$

3. Panel Number Effects on Forces and Moment

Lift, drag, and pitching-moment coefficient as a function of panel number and angle-of-attack are illustrated in Figures 2.22 through 2.24. Panel density was evenly increased around the leading edge using a spline program to stimulate peak suction resolution. A total of 184 panels was found to minimize the calculated drag resulting in a maximum C_d of .004 at 15° angle-of-attack. As before, the lift

coefficient was found insensitive and the moment coefficient was found to be only slightly sensitive to panel number.

4. Force and Moment Results

Panel computed and Lorber and Carta measured pressure coefficient as a function of Mach number and angle-of-attack is illustrated in Figures 2.25 through 2.32. Reasonable agreement was achieved at small angles-of-attack (0° to 9°). Increasing angle-of-attack and using compressibility corrections caused deviation from measured values.

Lift coefficient as a function of Mach number and angle-of-attack for calculated and Lorber and Carta measured values are displayed in Figure 2.33. Only slight deviation is observed at $M=0.2$ through 10° angle-of-attack. However, the compressibility effect calculated by panel.f was in the opposite direction (increasing C_{L_e} with increasing Mach number) to that measured by Lorber and Carta.

Moment coefficient as a function of Mach number, angle-of-attack, and pivot point calculated by panel.f and measured by Lorber and Carta are displayed in Figures 2.34 and 2.35. The general compressibility effect is accurately predicted and good correlation was achieved at lower angle-of-attacks. The aerodynamic center was located at 25.05%.

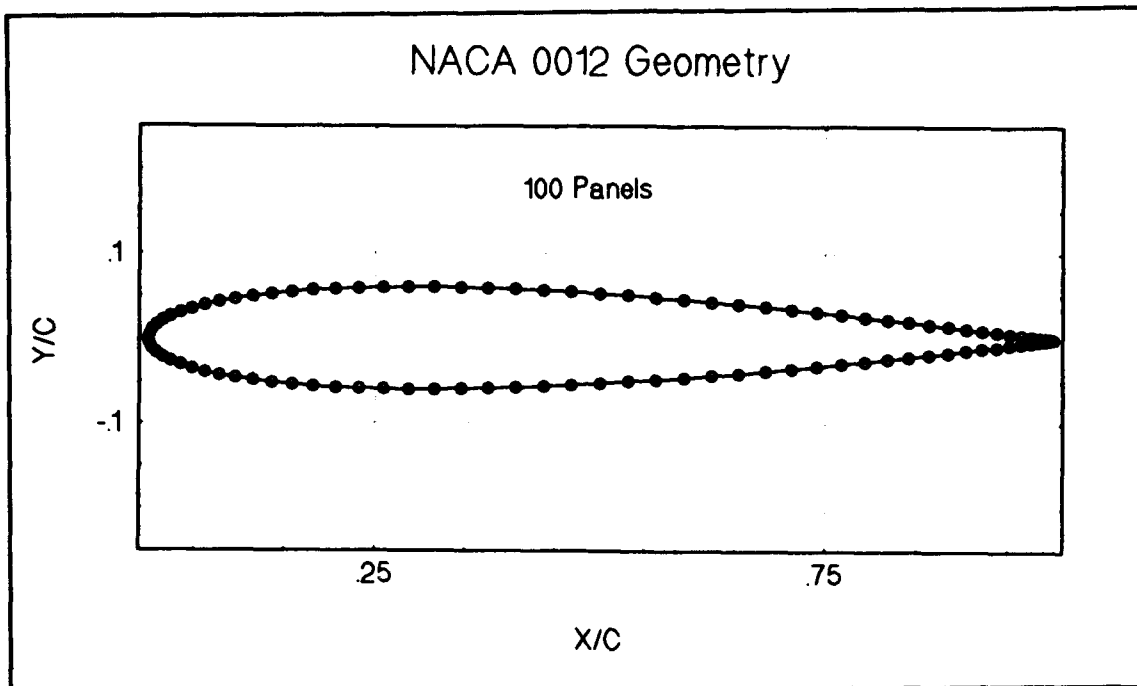


Figure 2.4

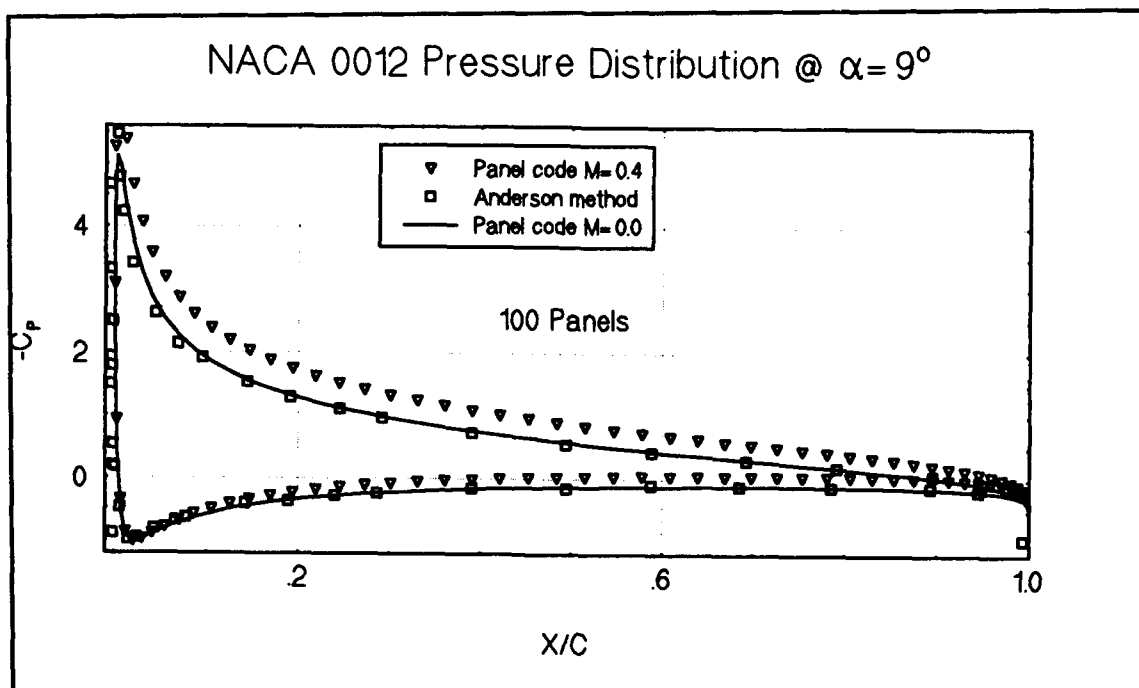


Figure 2.5

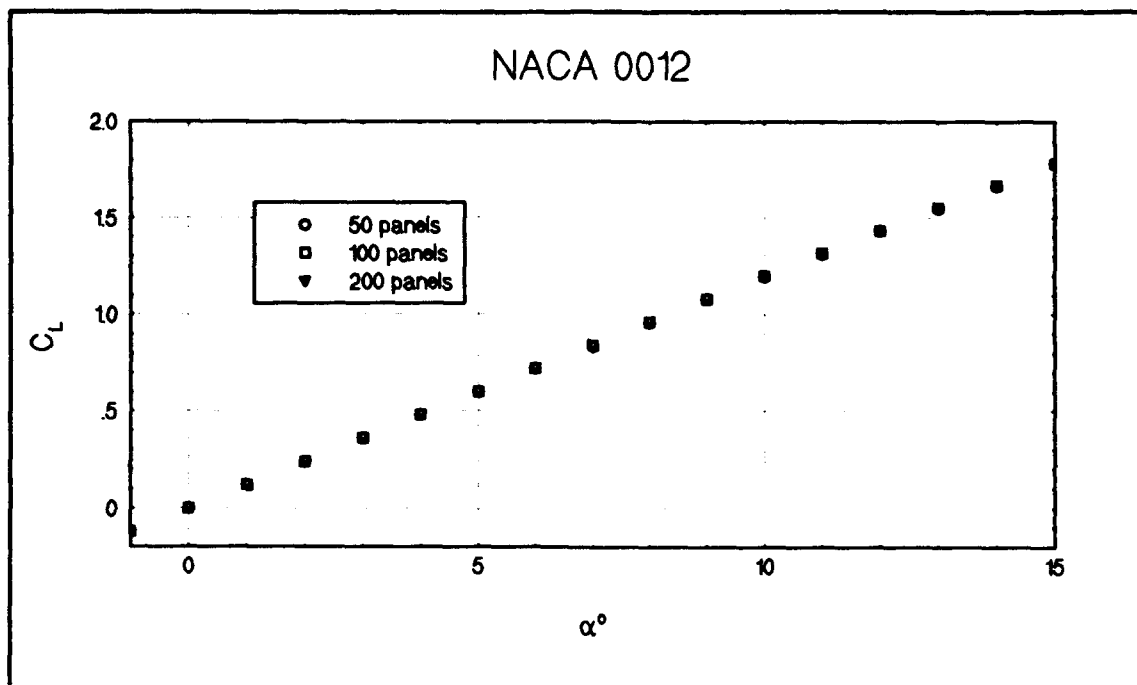


Figure 2.6

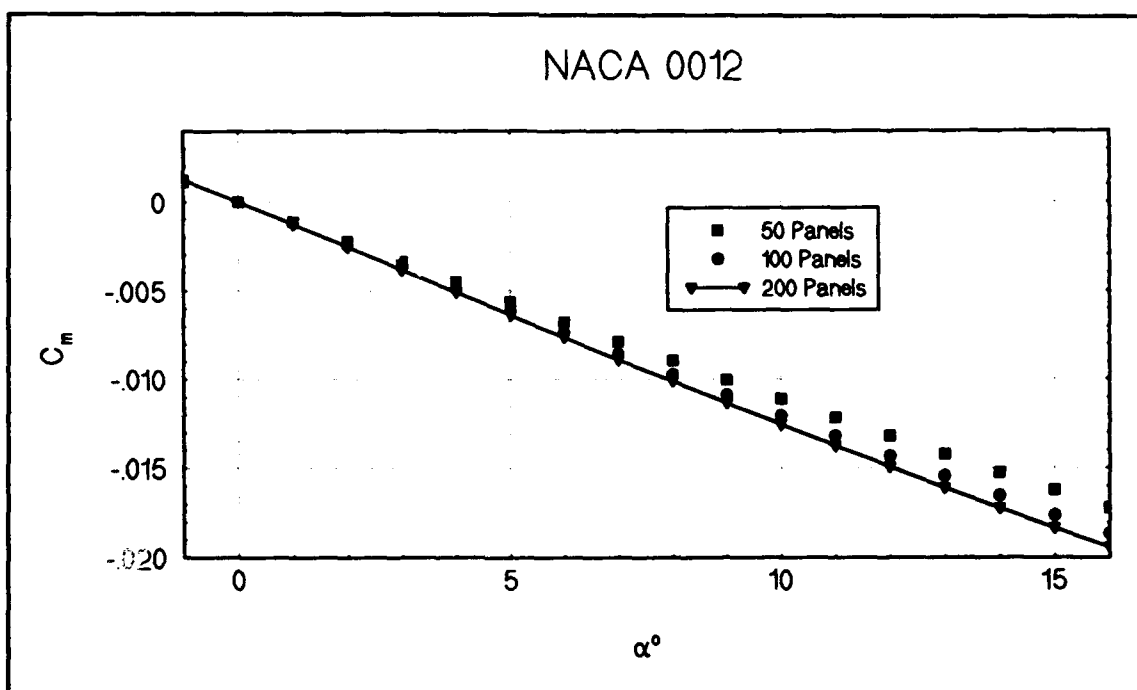


Figure 2.7

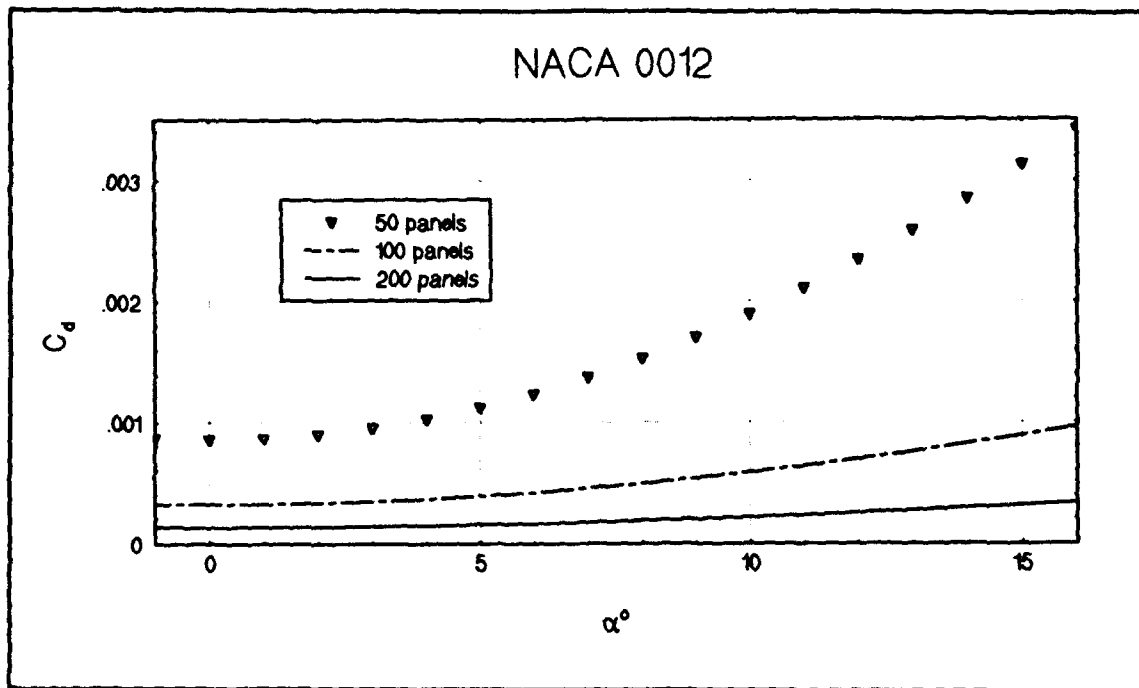


Figure 2.8

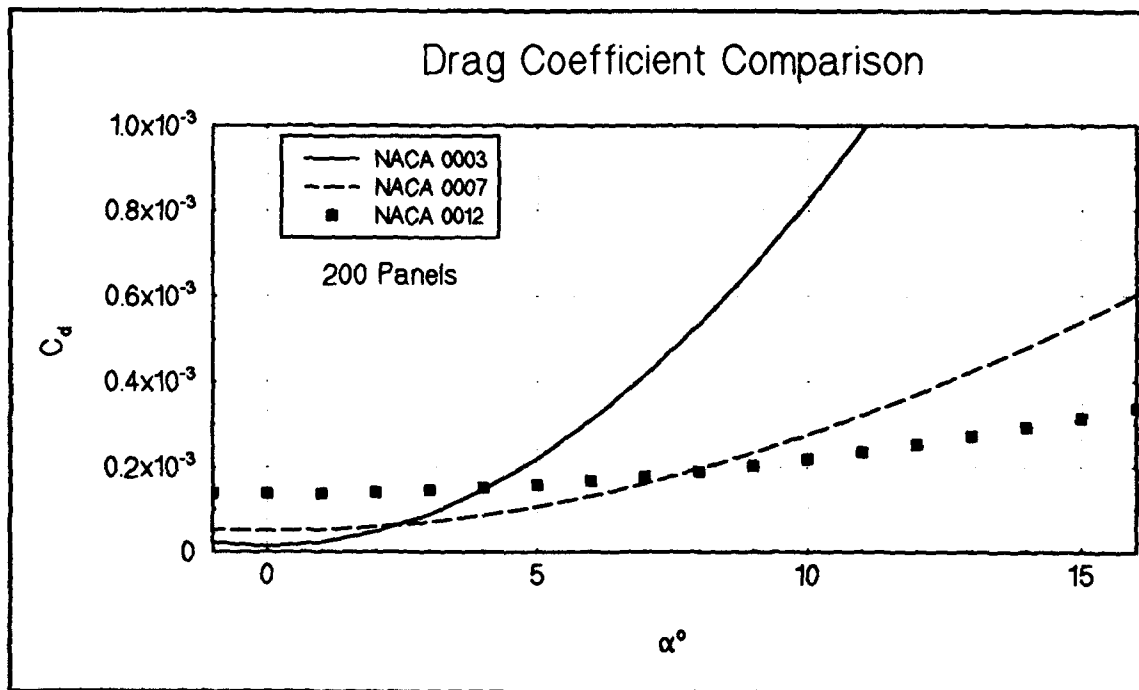


Figure 2.9

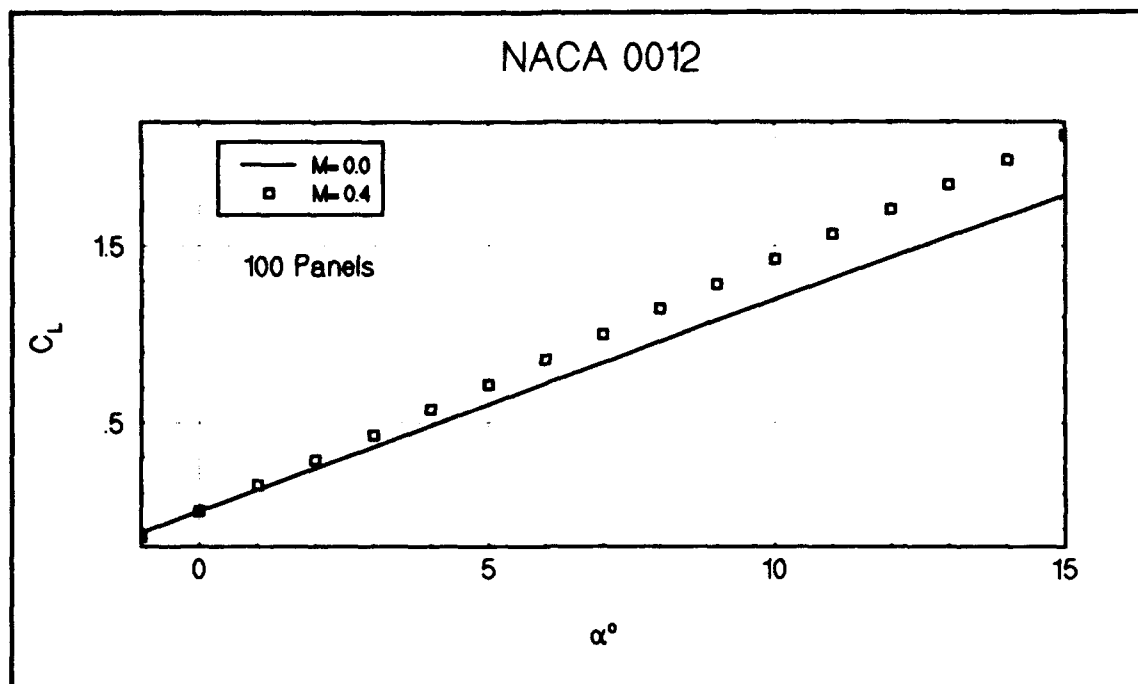


Figure 2.10

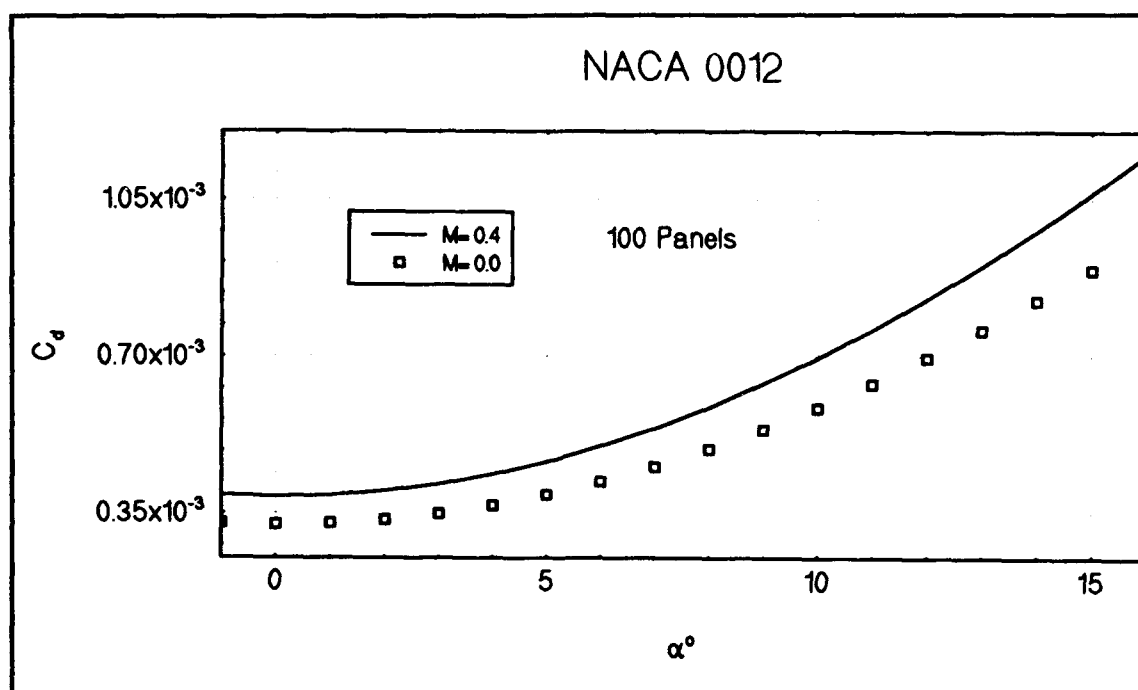


Figure 2.11

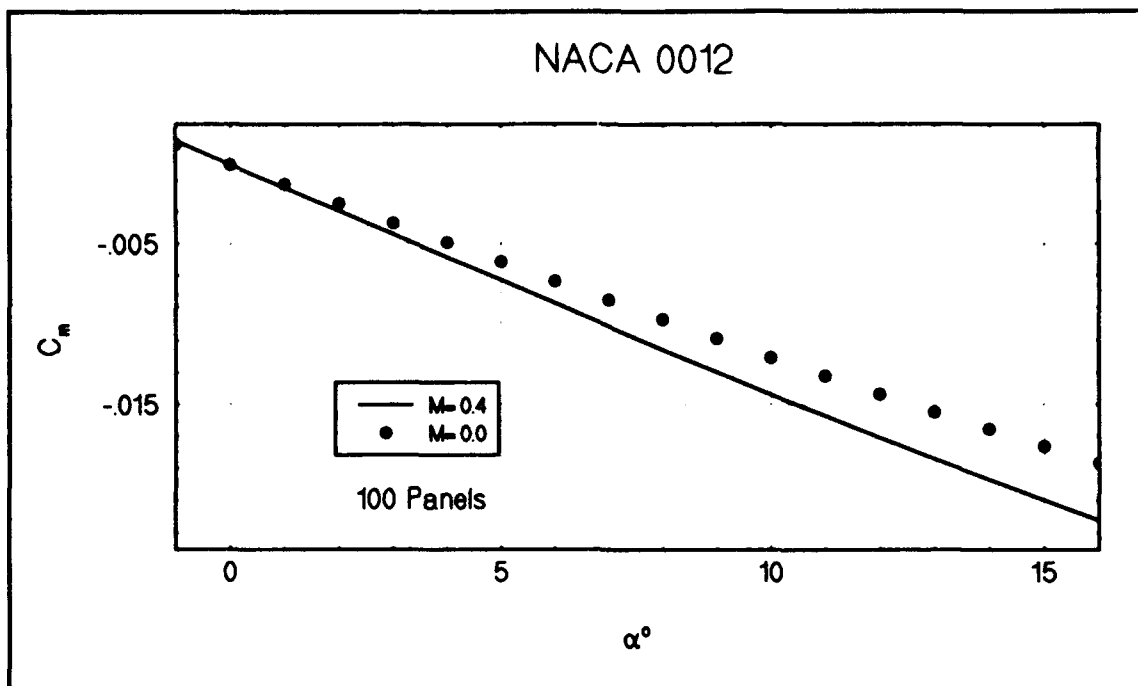


Figure 2.12

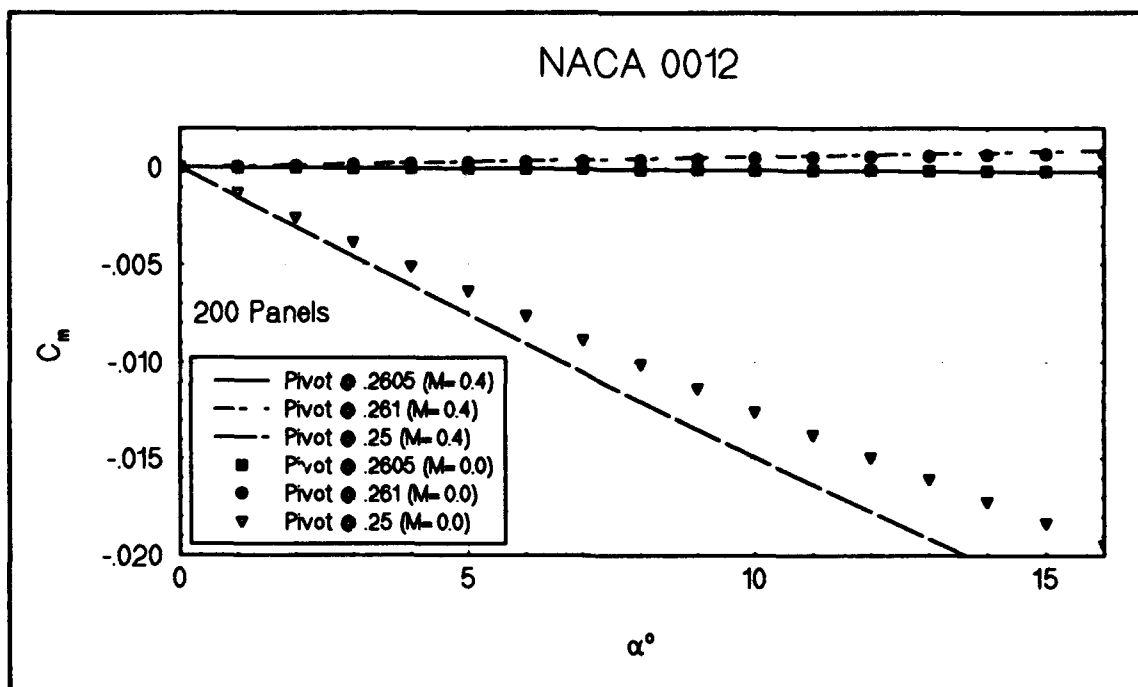


Figure 2.13

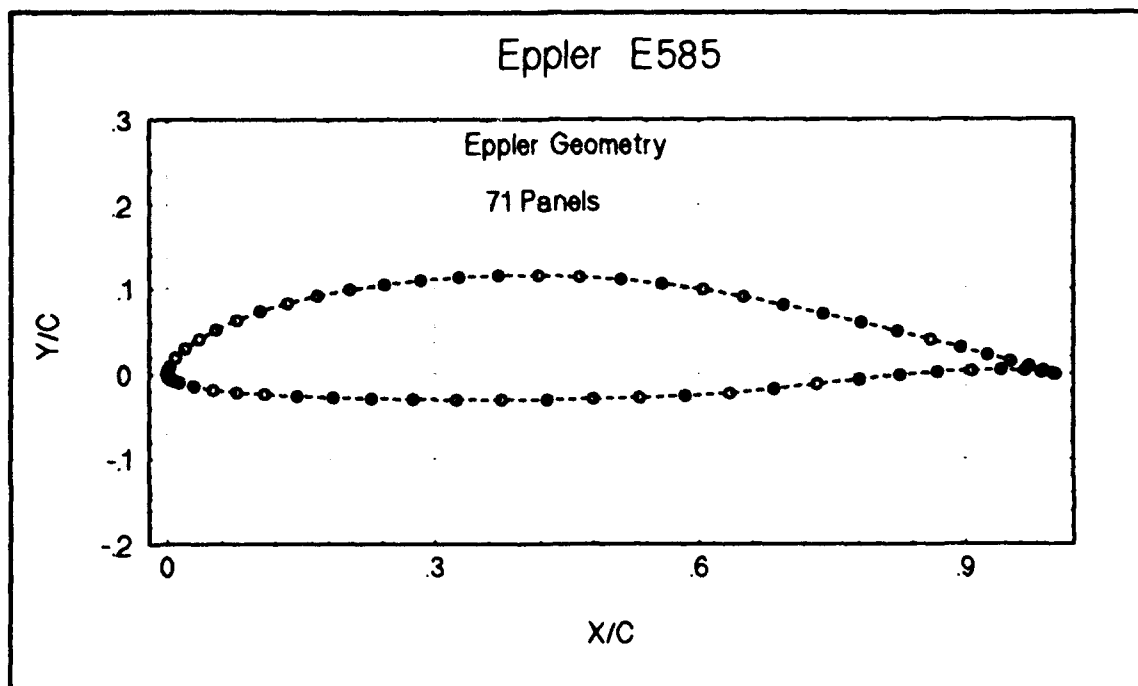


Figure 2.14

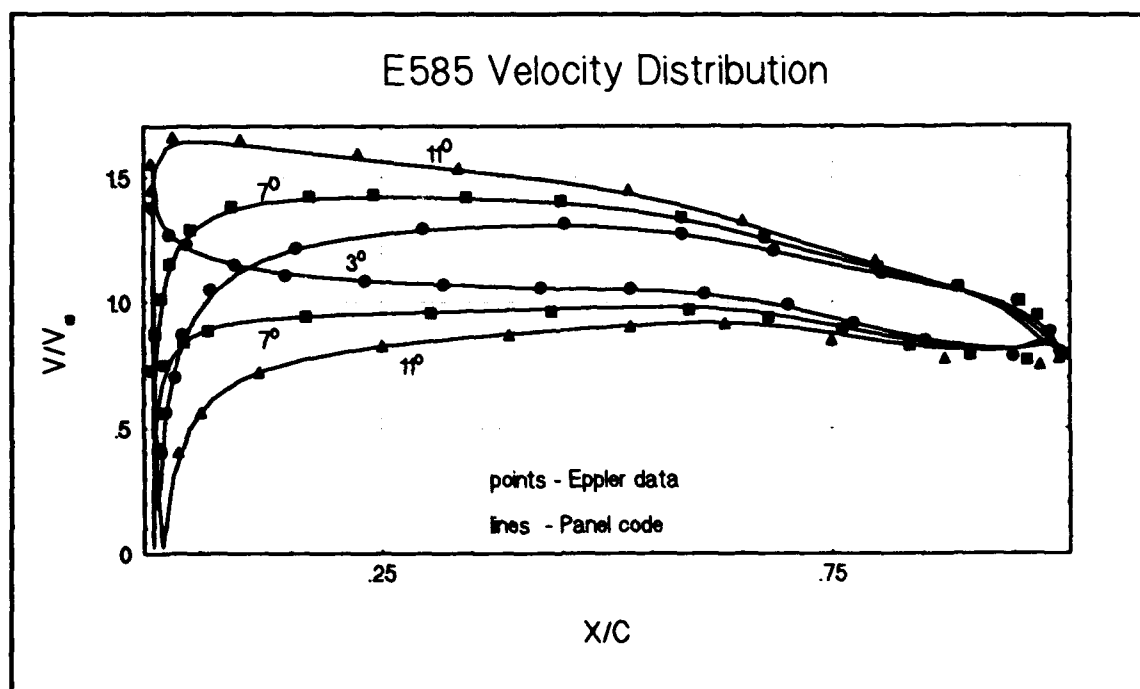


Figure 2.15

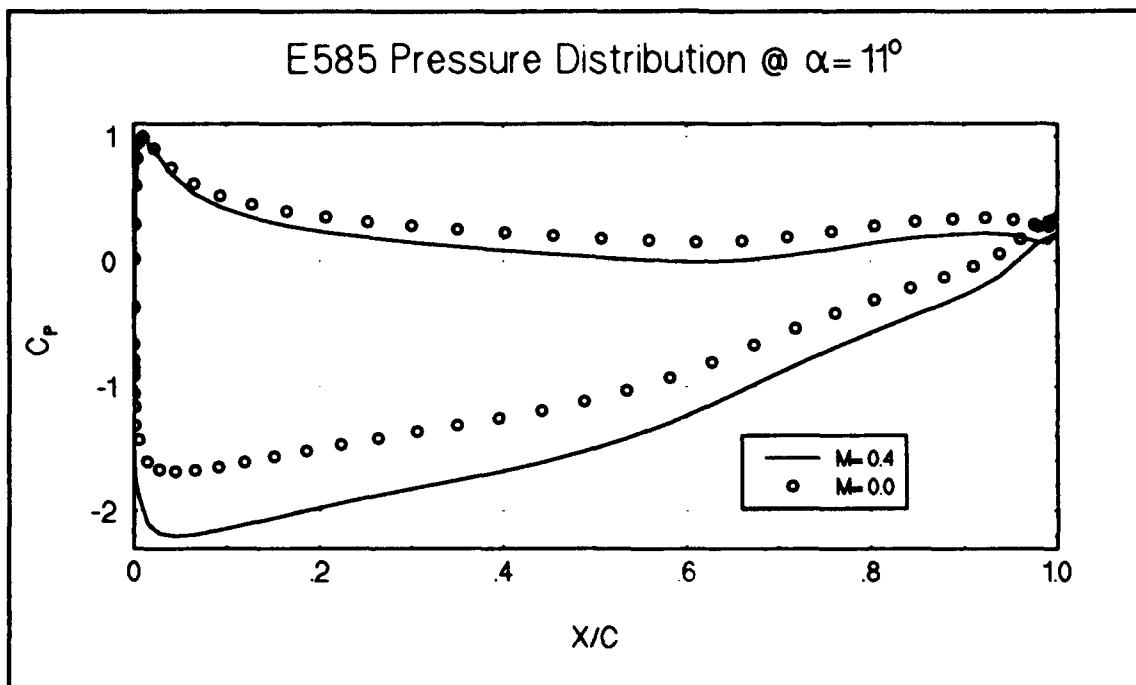


Figure 2.16

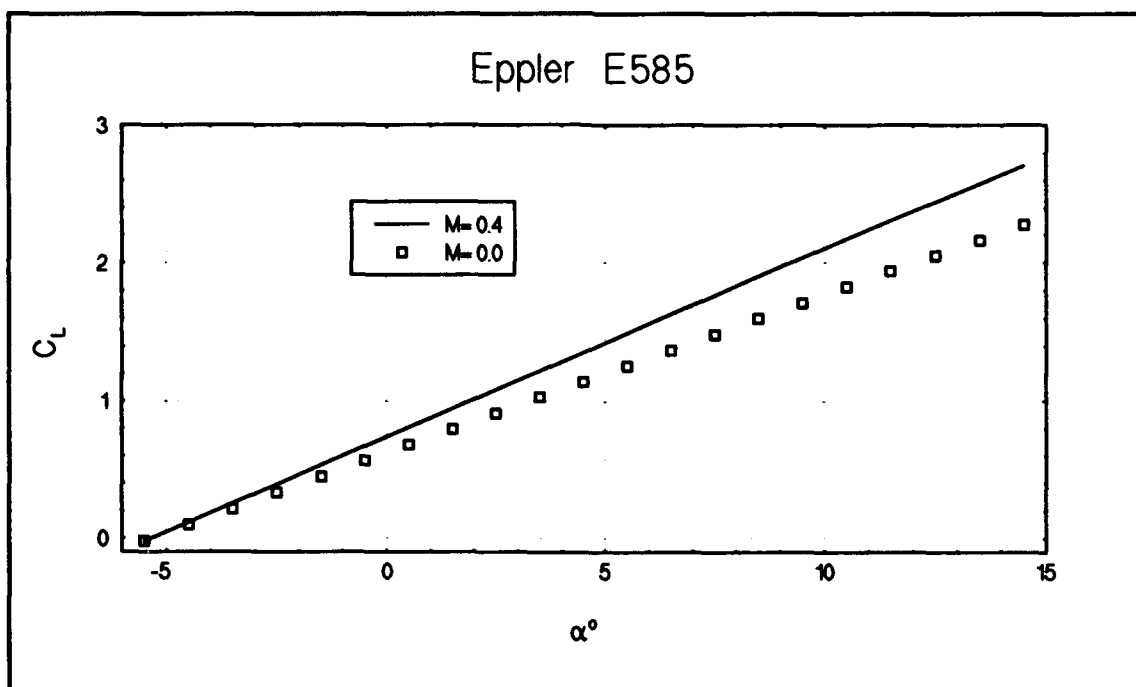


Figure 2.17

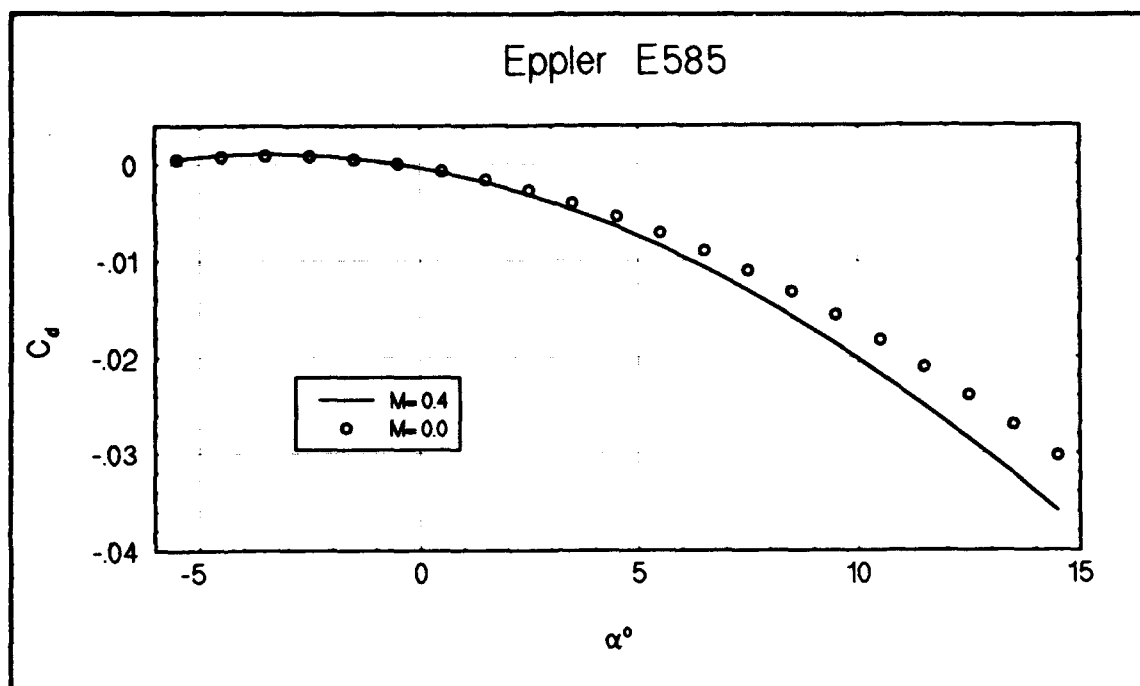


Figure 2.18

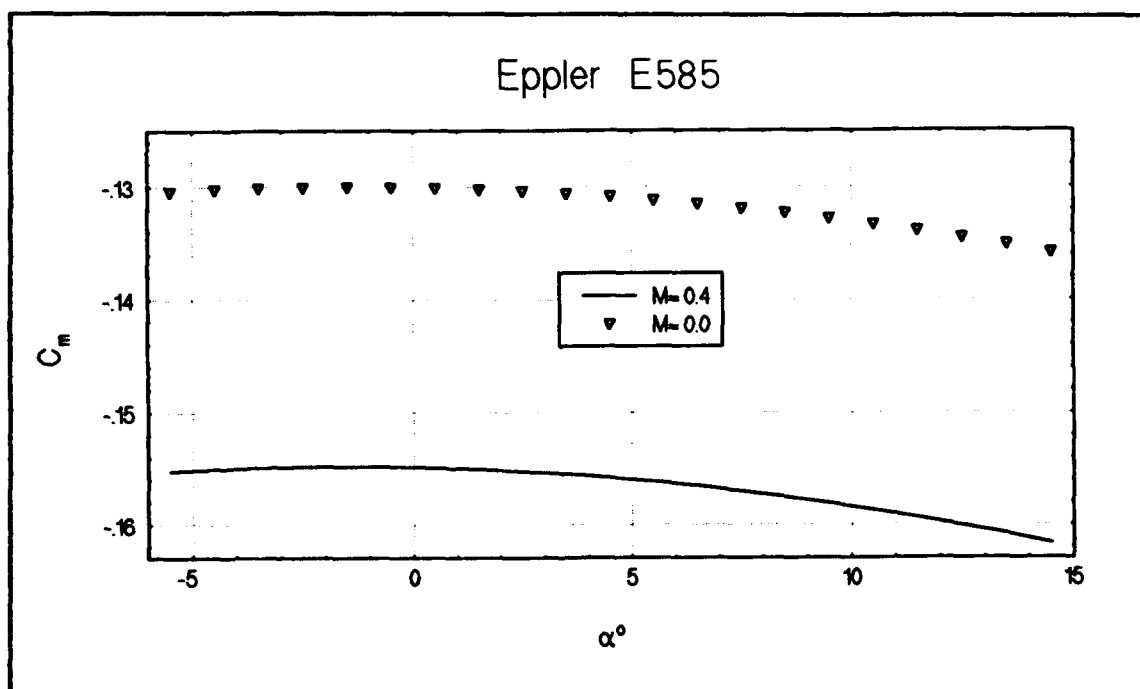


Figure 2.19

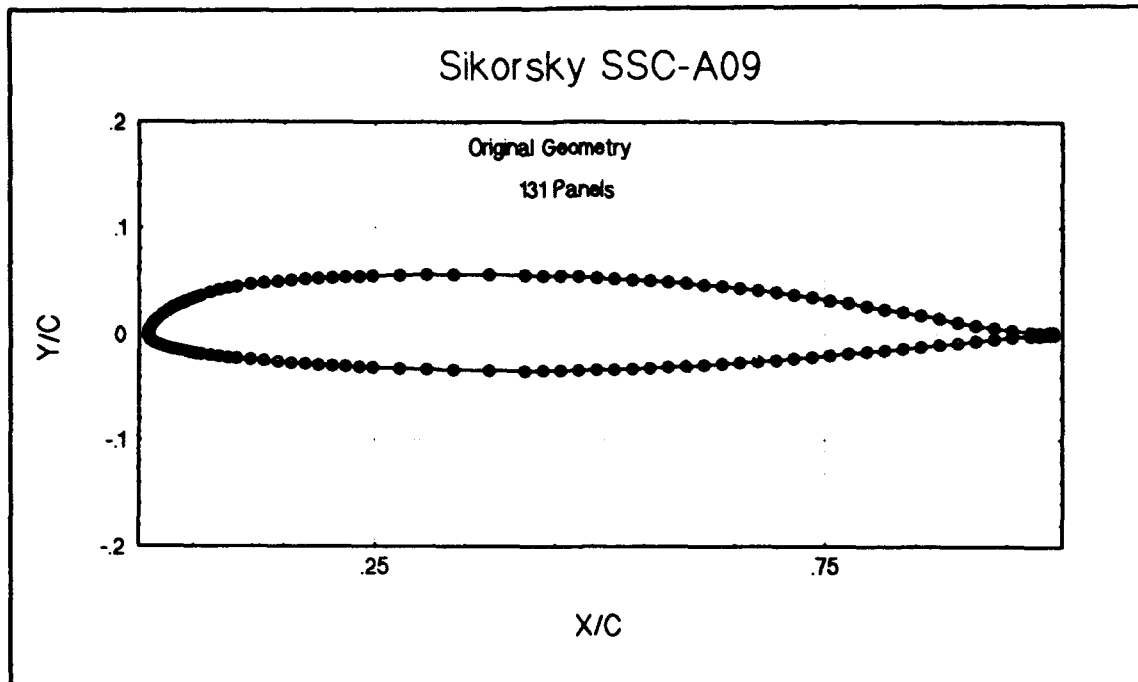


Figure 2.20

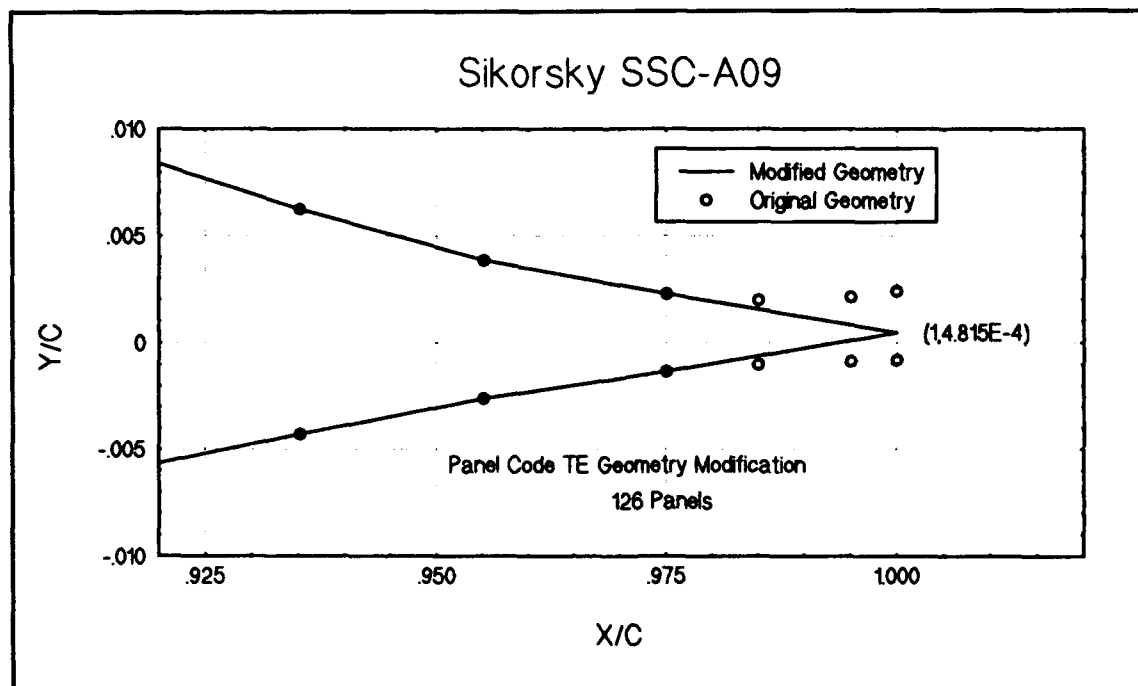


Figure 2.21

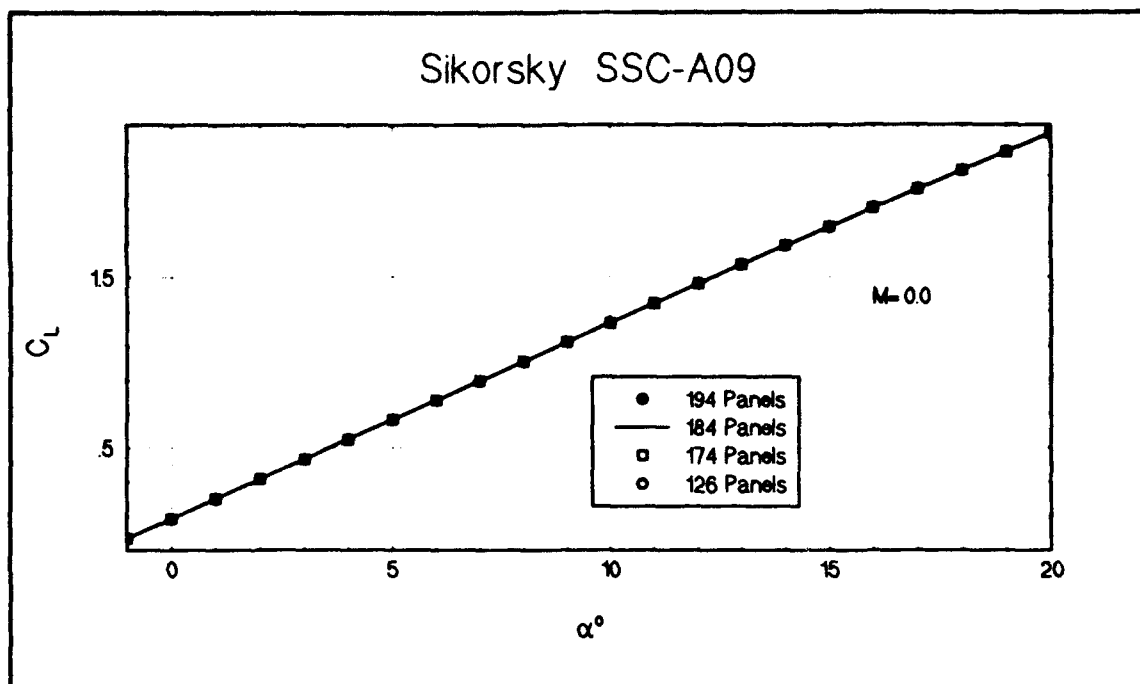


Figure 2.22

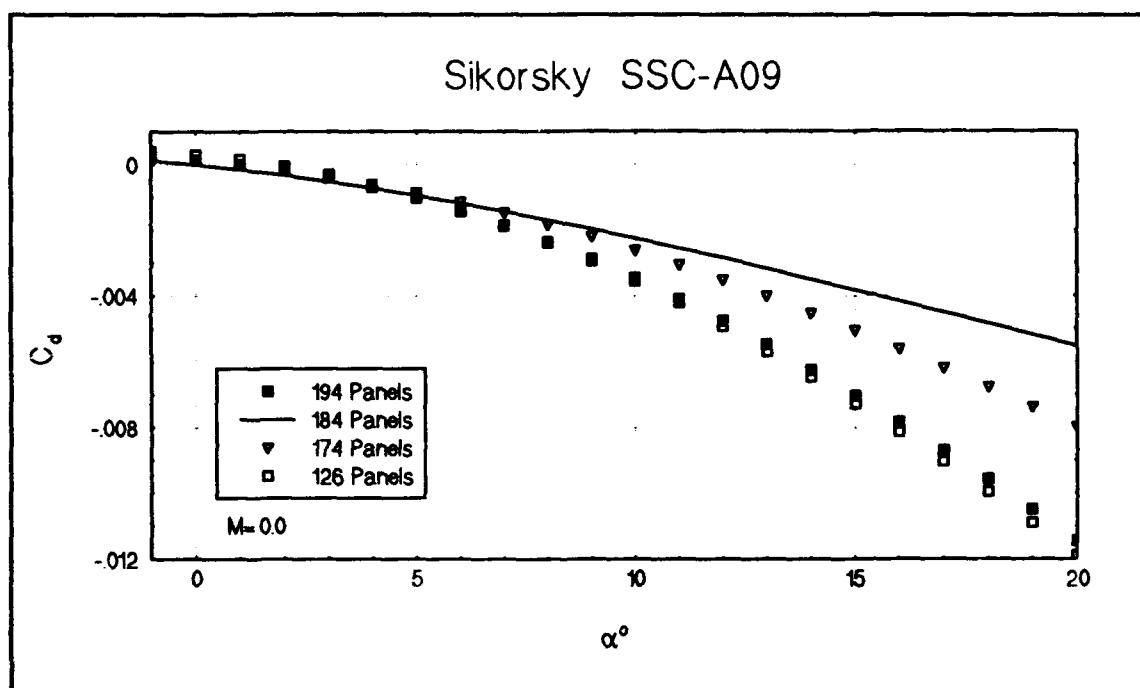


Figure 2.23

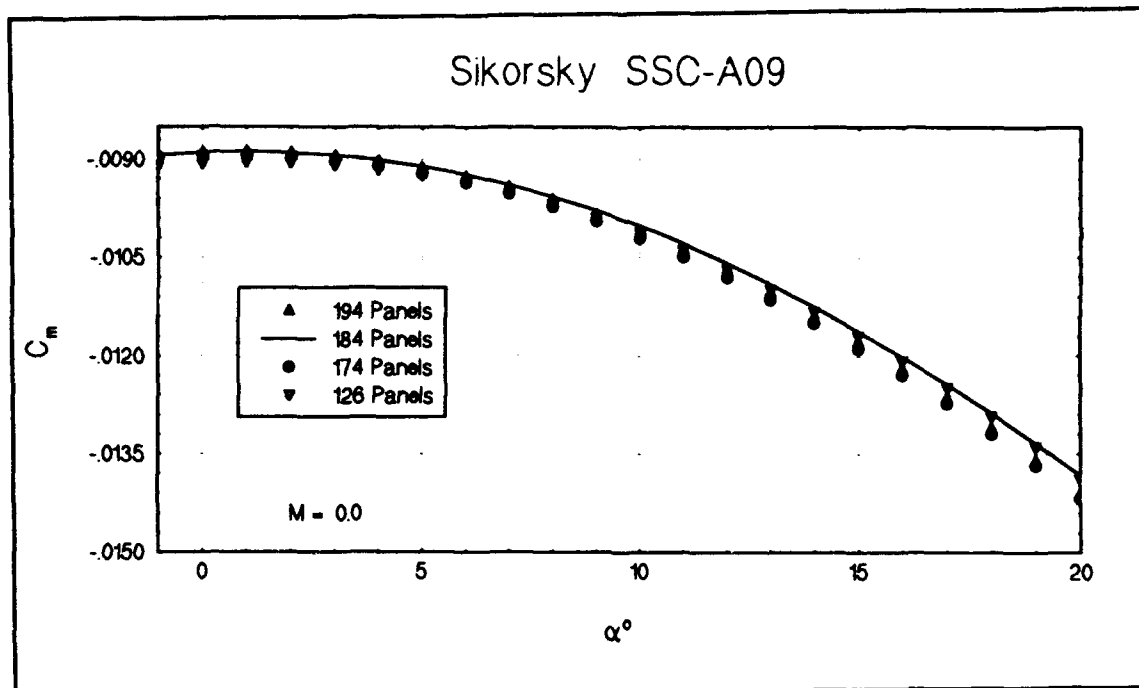


Figure 2.24

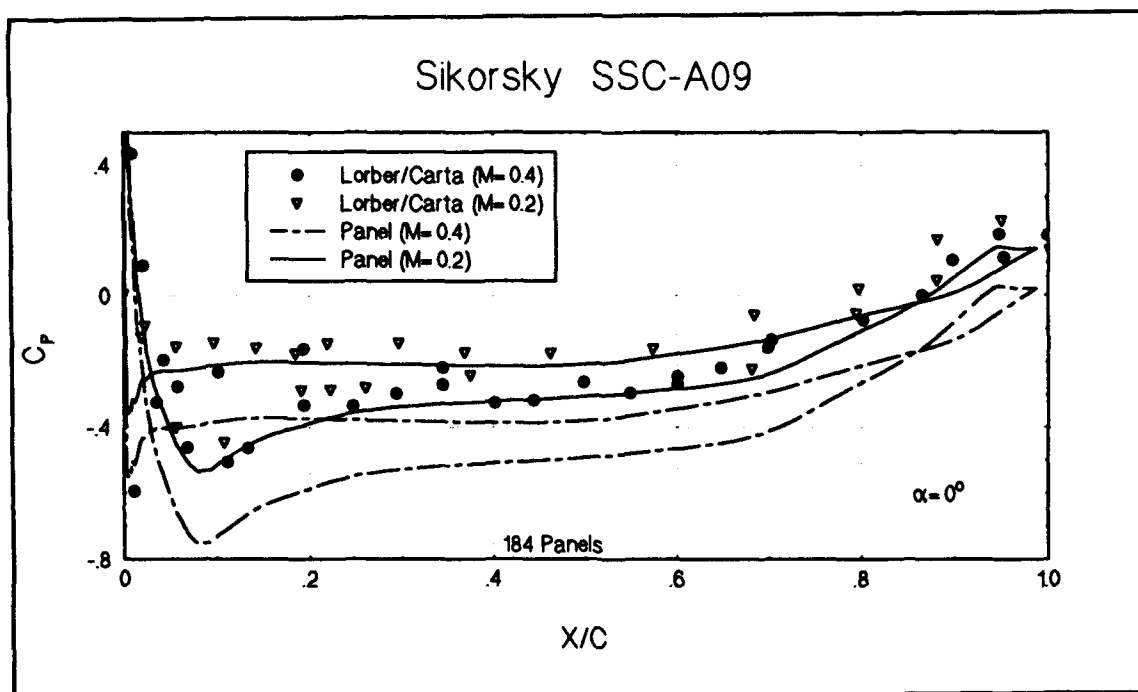


Figure 2.25

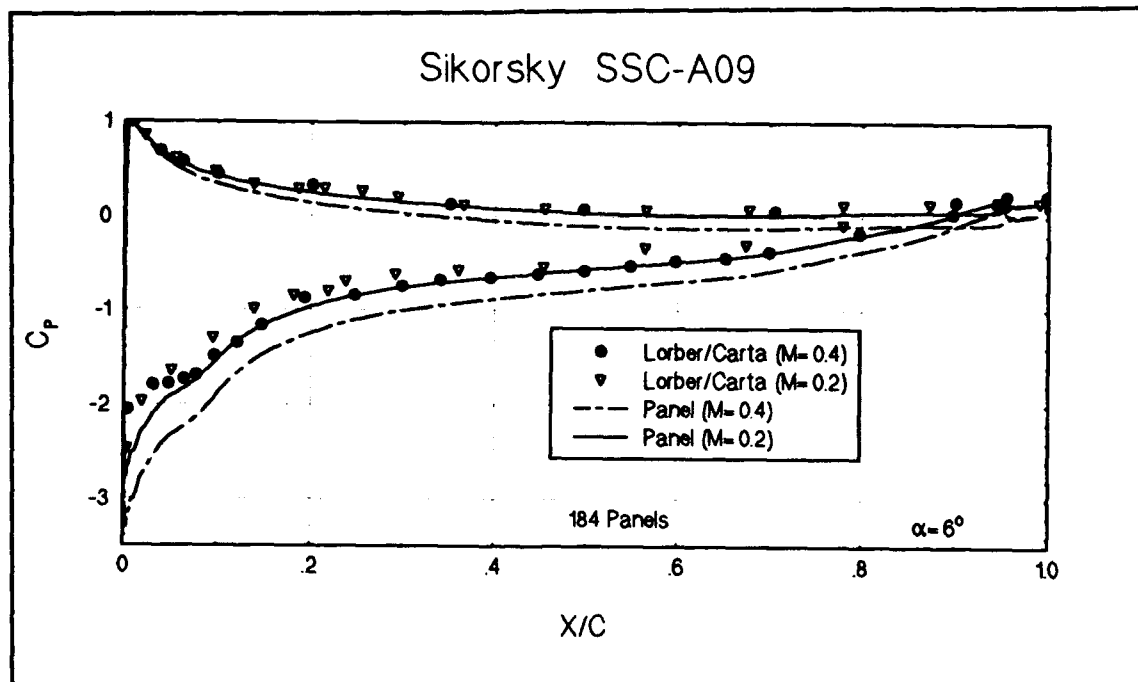


Figure 2.26

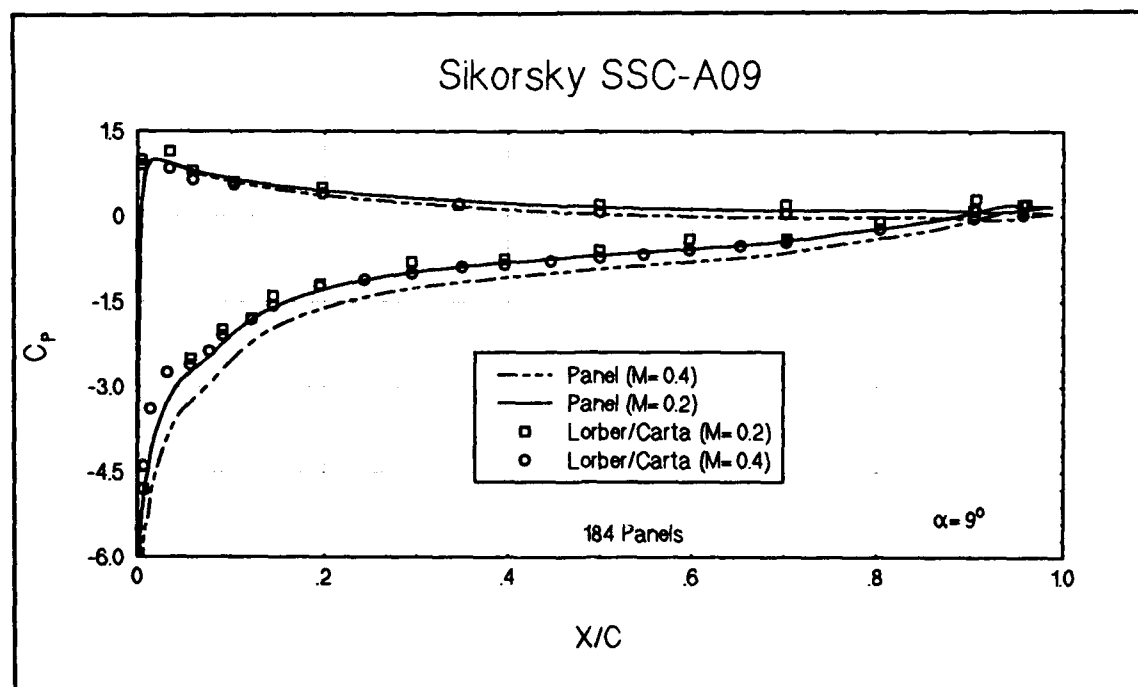


Figure 2.27

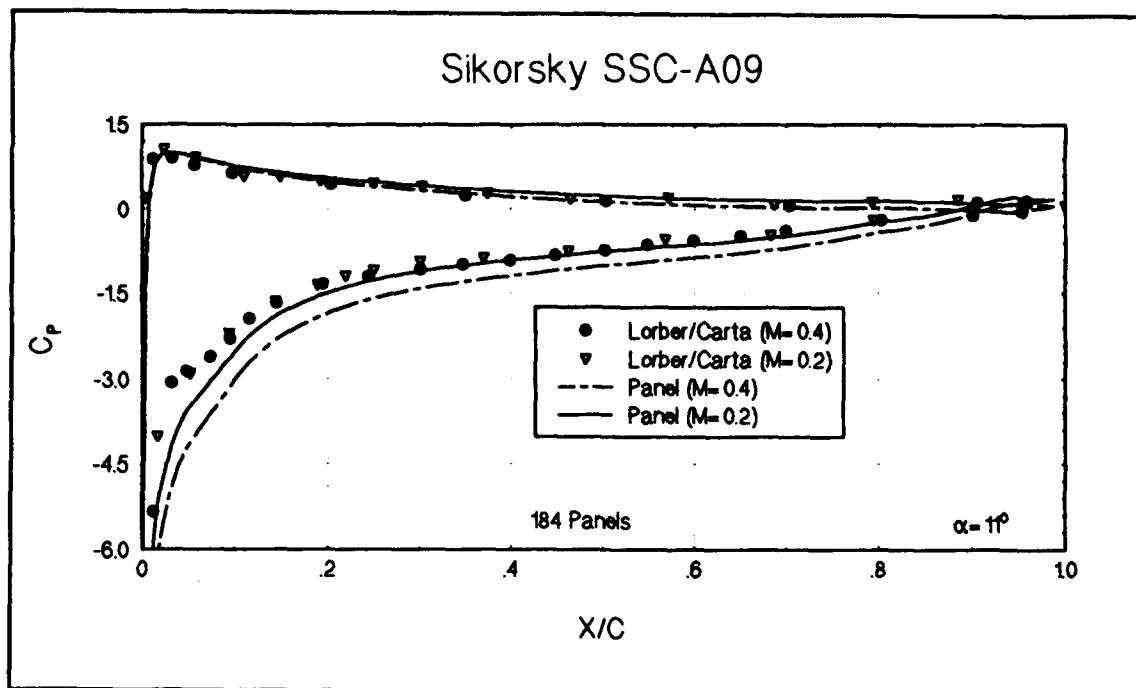


Figure 2.28

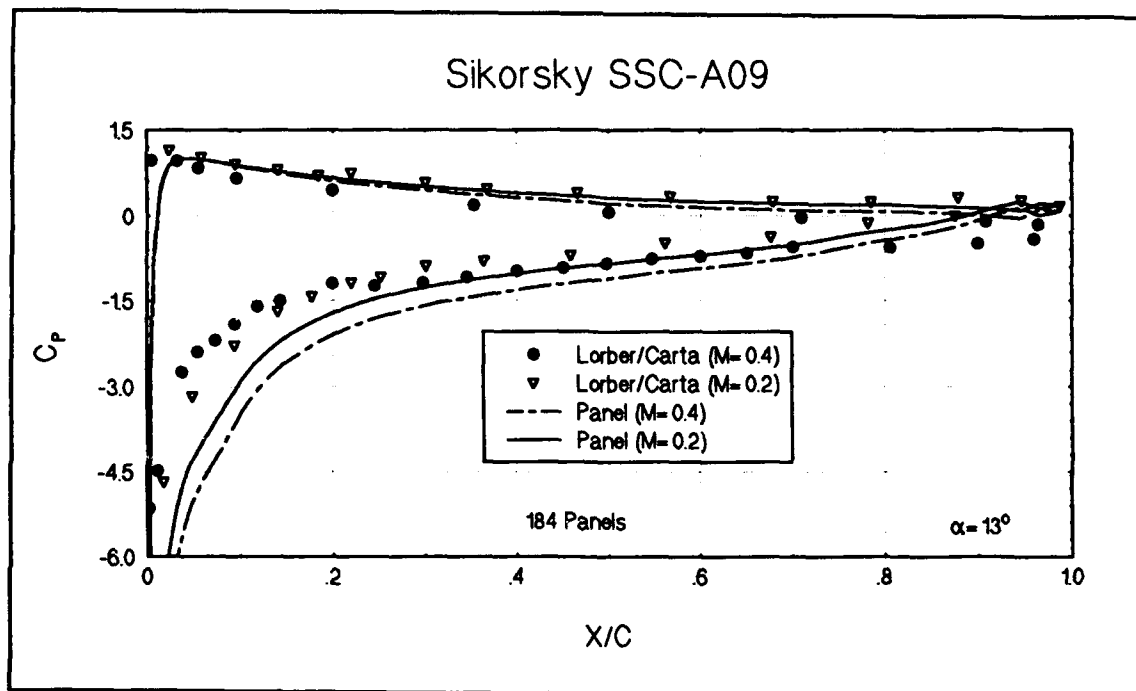


Figure 2.29

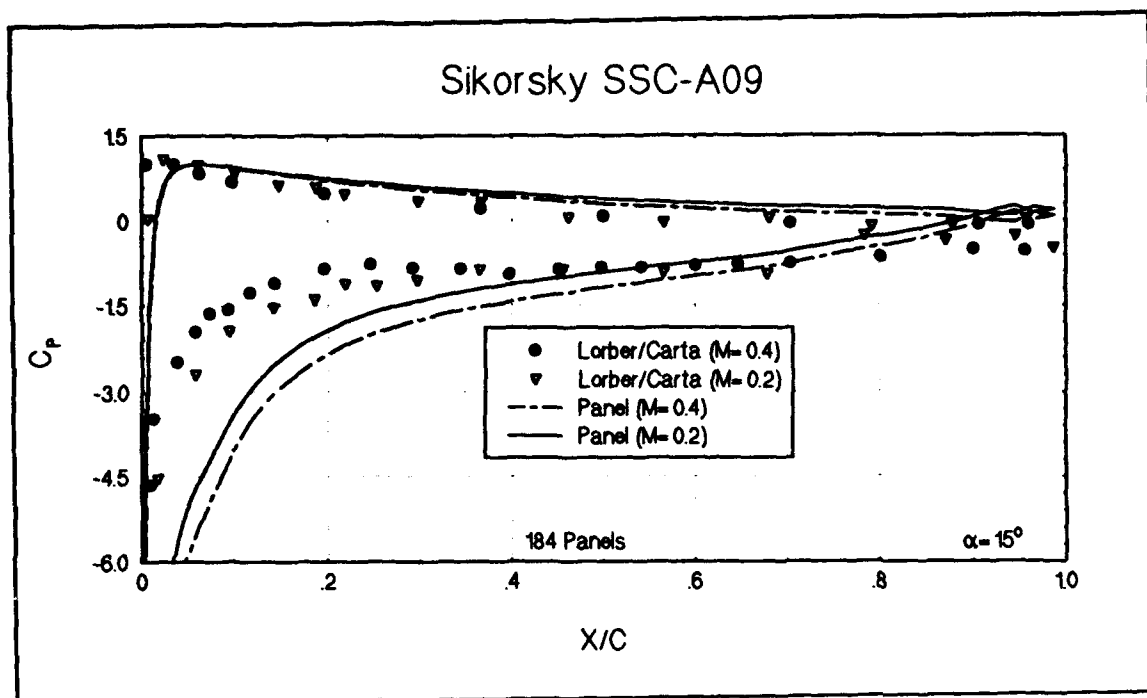


Figure 2.30

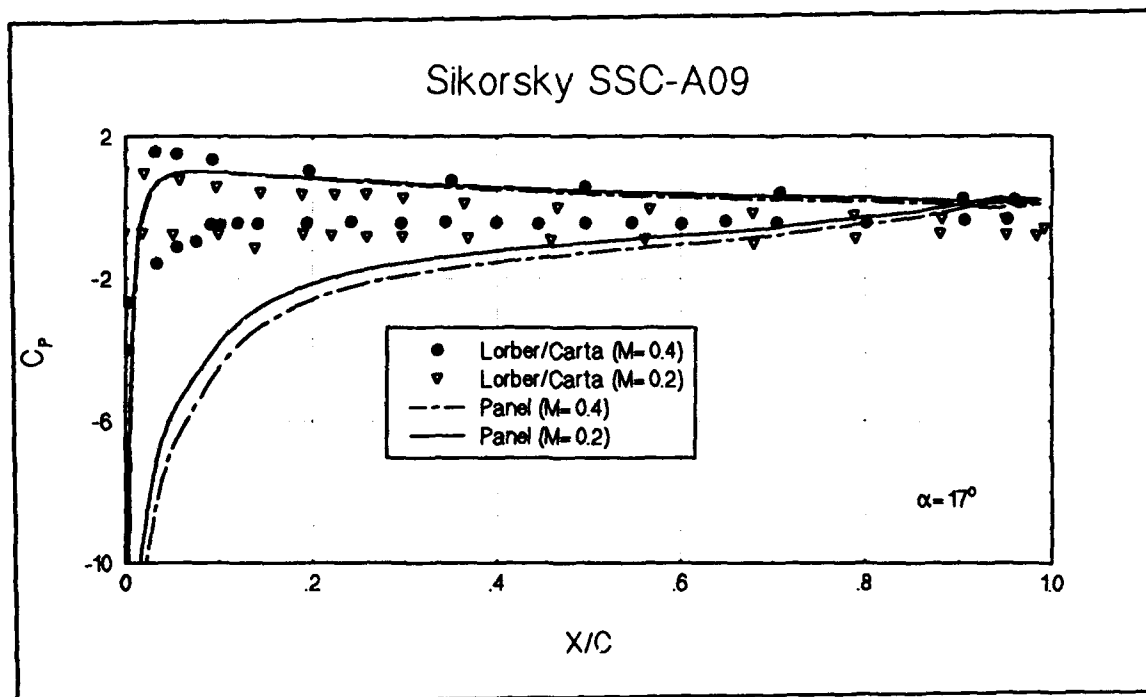


Figure 2.31

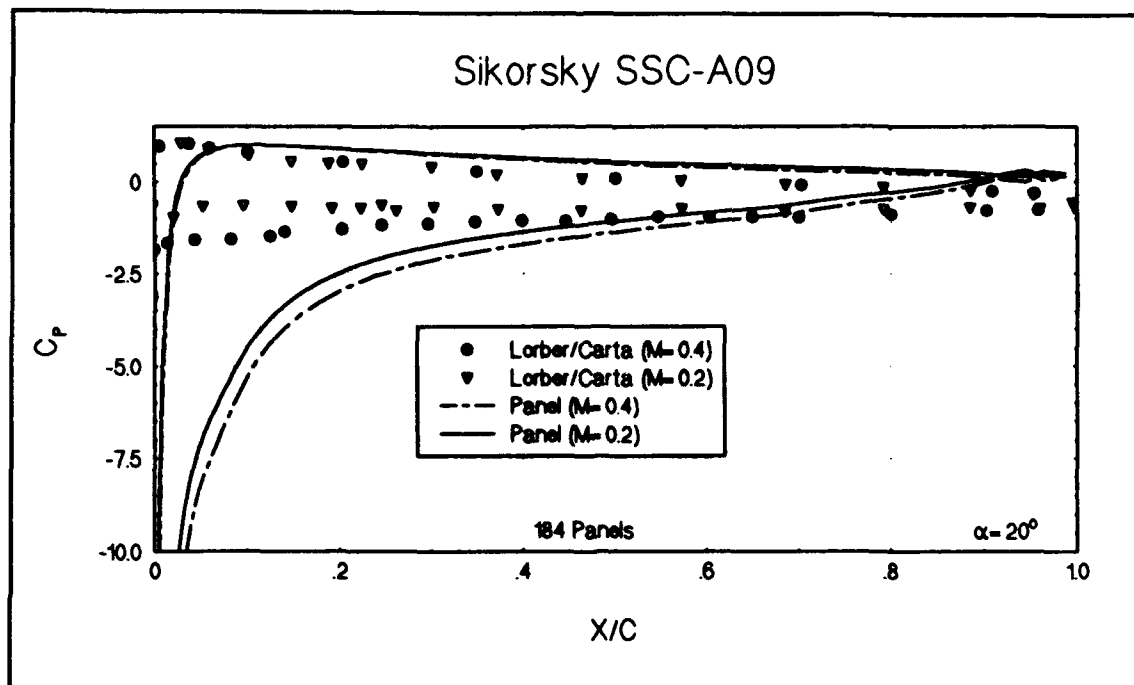


Figure 2.32

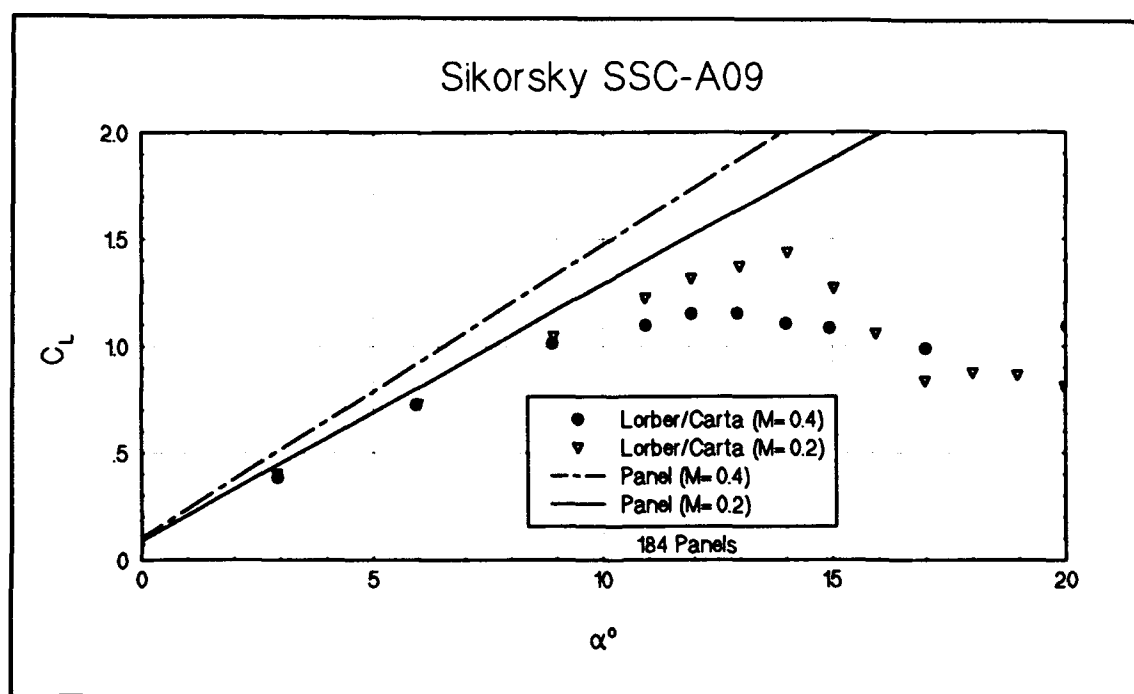


Figure 2.33

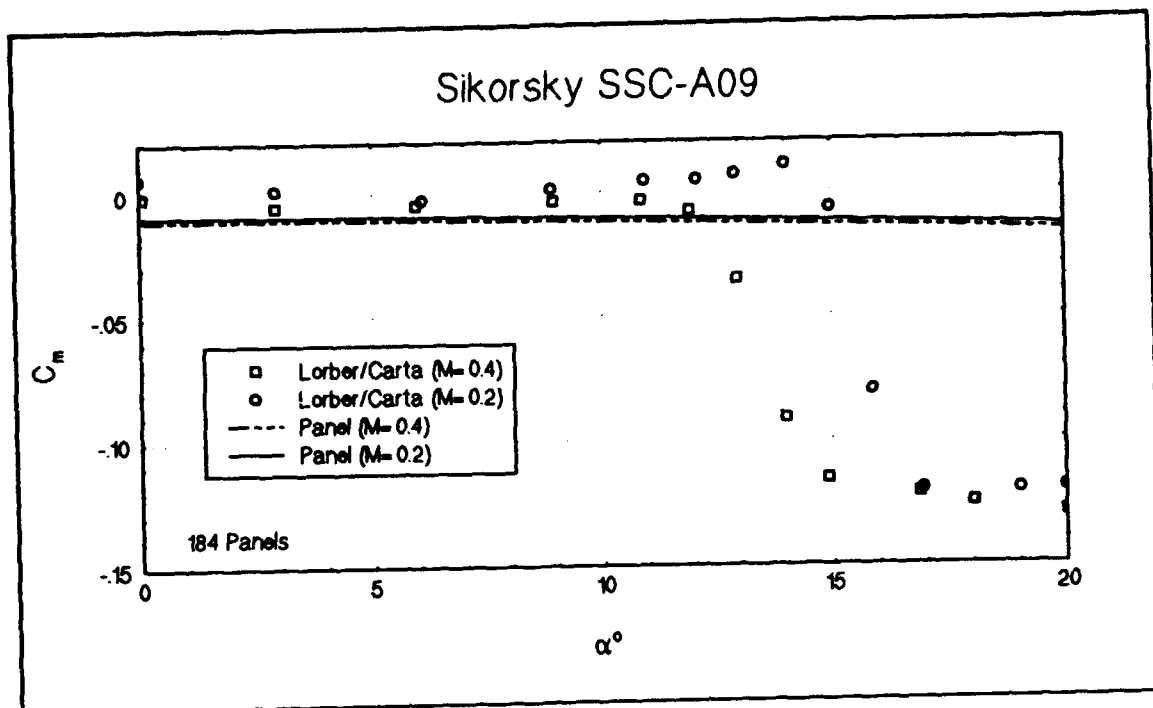


Figure 2.34

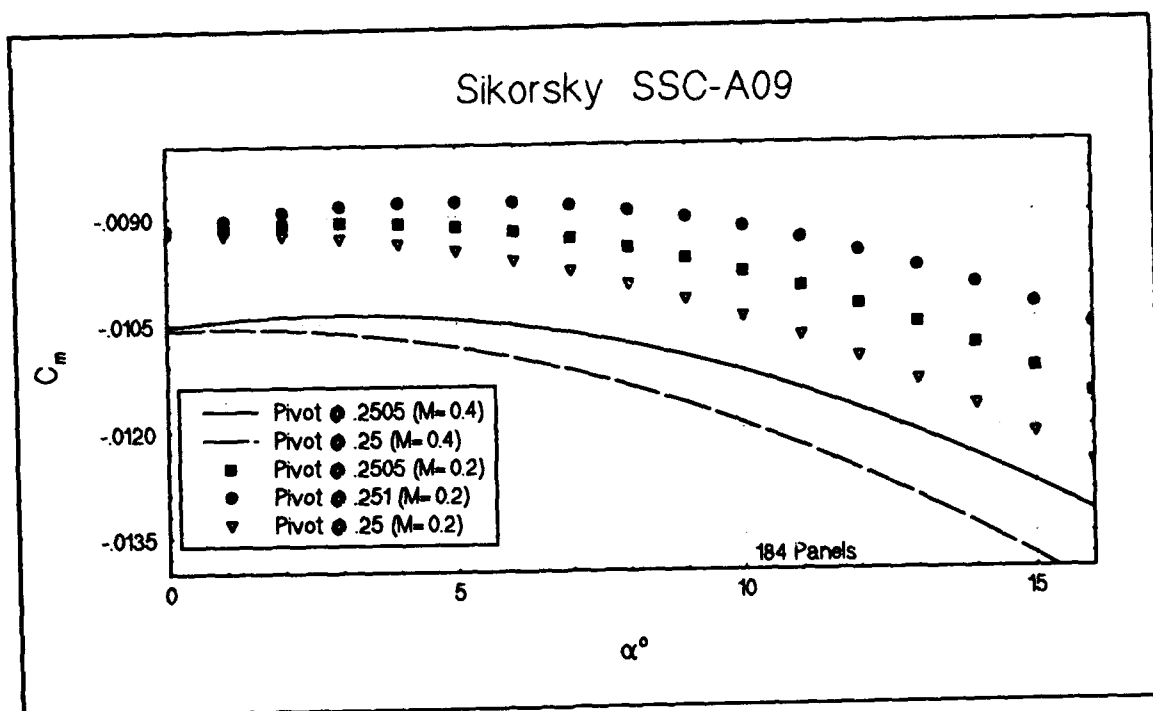


Figure 2.35

III. CEBECI 2-D LAMINAR & TURBULENT BOUNDARY LAYER CODE

A. SHEAR LAYER THEORY/BACKGROUND

Limiting the flow field to two-dimensional with no body forces, the generalized Navier-Stokes equations can be obtained when Newton's second law is applied to a finite control volume fixed in space or to an infinitesimally small moving fluid element. The resulting general unsteady, compressible, viscous flow Navier-Stokes equations become:

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} = - \frac{\partial p}{\partial x} + \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} \quad (3.1)$$

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho uv)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} = - \frac{\partial p}{\partial y} + \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} \quad (3.2)$$

The unsteady continuity equation results when the conservation of mass principle is applied to a finite control volume fixed in space and is shown in Equation 3.3:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \quad (3.3)$$

Since a complete flow field solution to these equations requires a vast amount of computer time and power, a dimensional/order-of-magnitude reduction of the Navier-Stokes equations results in the boundary layer equations. These equations allow a practical scheme to computationally solve the flow field.

When a steady, incompressible flow field is assumed, the energy equation will decouple from the momentum and continuity equations allowing ease of solution. Assuming that the fluid behaves as a Newtonian fluid, where viscous stress is proportional to the rate of fluid strain, assuming constant flow properties, and subtracting out the continuity equation the x-component of the momentum equation becomes:

$$u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = - \frac{1}{\rho} \frac{\partial p}{\partial x} + \nu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right] \quad (3.4)$$

where ν is the kinematic viscosity ($\nu = \mu/\rho$).

There are three types of fluid momentum transfer:

- Transport by fluid mean motion.
- Transfer of random molecular motion (viscous stresses).
- Transfer by turbulent eddies (mean turbulent stresses).

Except at very low Reynolds numbers, viscous stresses are small compared to the rate of momentum transfer by the mean fluid element motion. Instantaneous flow quantities are replaced by a mean and fluctuating term to incorporate turbulent flow effects. This results in extra stress terms often called Reynolds stresses.

$$u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = - \frac{1}{\rho} \frac{\partial p}{\partial x} + \nu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right] - \frac{\partial \overline{u'^2}}{\partial x} - \frac{\partial \overline{u'v'}}{\partial y} \quad (3.5)$$

The basic boundary layer theory assumptions are that the boundary layer is very thin when compared to the body length scale (airfoil chord), and the flow Reynolds number is large. An order-of-magnitude analysis of the x and y turbulent flow momentum equations result in the steady, two-dimensional, incompressible **Boundary Layer Equations** for laminar and turbulent flows (Cebeci and Bradshaw [Ref. 5]):

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \quad (3.6)$$

$$u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = - \frac{1}{\rho} \frac{\partial p_{ext}}{\partial x} + \nu \frac{\partial^2 u}{\partial y^2} - \frac{\partial \overline{u'v'}}{\partial y} \quad (3.7)$$

$$\frac{\partial p}{\partial y} = 0 \quad (3.8)$$

The y-component of the momentum equation implies that pressure is constant through the boundary layer in the direction normal to the surface. This means that the pressure distribution at the boundary layer outer edge is impressed directly onto the surface without change. This assumption is generally true as long as one stays away from large curvatures (Anderson [Ref. 2]). This allows division of the flow field into an inner, the viscous boundary layer region, and an outer region where viscous stresses are negligible and thus can be treated and solved using incompressible, inviscid numerical methods. Two boundary conditions are applied. The no slip,

airfoil surface boundary condition is represented by $u=v=0$. The outer boundary layer edge condition is $y=\delta$ at $U=U_e(x)$.

1. Turbulence Model

Turbulence modeling is used to relate the Reynolds shear stress term of Equation 3.7 to the local mean-velocity gradient allowing numerical flow field calculation. This modeling is based on local equilibrium - the assumption that the transport terms are small. Prandtl proposed a mixing length model, Equation 3.9, similar to the kinetic theory of gases where turbulent eddies are assumed to be discrete and to collide and exchange momentum at distinct/discrete intervals. Here, l is a characteristic length related to the fluid turbulence intensity (Cebeci and Bradshaw [Ref. 5]).

$$-\rho \overline{u'v'} = \rho l^2 \left| \frac{\partial u}{\partial y} \right| \frac{\partial u}{\partial y} \quad (3.9)$$

Boussinesq proposed a mean flow, eddy-viscosity model, Equation 3.10, where ϵ_m is termed the turbulent eddy-viscosity and is assumed to vary less rapidly than the shear stress term. It is important to note that eddy-viscosity is not a flow property and depends greatly on the mean-velocity gradient and mixing length (Cebeci and Bradshaw [Ref. 5]).

$$-\rho \overline{u'v'} = \rho \epsilon_m \frac{\partial u}{\partial y} \quad (3.10)$$

The Cebeci-Smith eddy-viscosity model, Equations 3.11 and 3.12, is used for separated flow computation and treats

the boundary layer as a composite layer having an inner, ϵ_{ni} , and an outer, ϵ_{no} , region with separate empirical formulations. The inner eddy-viscosity defines the region from the airfoil surface outward until $\epsilon_{ni} = \epsilon_{no}$, where the outer eddy-viscosity takes over to the edge of the boundary layer (Cebeci and Bradshaw [Ref. 5]).

$$\left(\frac{\epsilon_m}{\nu}\right)_i = .16 R_{e_x}^{\frac{1}{2}} \left[1 - e^{-\left(\frac{y}{\lambda}\right)^2}\right] \eta^2 \nu \gamma_{tr} \quad (3.11)$$

$$\left(\frac{\epsilon_m}{\nu}\right)_o = .0168 R_{e_x}^{\frac{1}{2}} [\eta_{\infty} - f_{\infty}] \gamma_{tr} \quad (3.12)$$

$$\begin{aligned} R_{e_x} &= \frac{U_e}{U_{\infty}} \xi R_L \\ \frac{y}{\lambda} &= \frac{1}{26} R_{e_x}^{\frac{1}{4}} \nu^{\frac{1}{2}} \eta \\ \gamma_{tr} &= 1 - \exp \left[-G (x - x_{tr}) \int_{x_{tr}}^x \frac{dx}{U_e} \right] \\ G &= \frac{1}{1200} \left[\frac{U_e}{U_{\infty}} \right]^3 R_L^2 R_{e_{x_{tr}}}^{-1.34} \\ f(x, \eta) &= \frac{\Psi(x, y)}{\sqrt{U_e \nu x}} \end{aligned} \quad (3.13)$$

2. Transition Model

Laminar to turbulent flow transition presents a stability problem where vortical interaction is very non-linear. The Chen-Tyson transition model utilizes a region of intermittency that is controlled by the intermittency factor,

γ_{tr} , which allows turbulence to gradually build in the streamwise direction creating a transition zone instead of a laminar to turbulent transition point.

Michel's empirical correlation curve for transition, Equation 3.14, is used as an initial estimate for transition location. It is based on incompressible and constant property flow. See Appendix D for further discussion.

$$R_{\theta_{tr}} = 1.174 \left[1 + \frac{22,400}{R_{e_{x_{tr}}}} \right] R_{e_{x_{tr}}}^{.46}$$

$$R_{\theta_{tr}} = \frac{U_e \theta}{\nu} \quad (3.14)$$

$$R_{e_{x_{tr}}} = \frac{U_e x}{\nu}$$

B. BL2D.F OVERVIEW

A two-dimensional, steady, incompressible, viscous flow program was developed by Cebeci and Bradshaw [Ref. 5] to provide solutions to the (Thin Shear Layer) boundary layer equations using the Cebeci-Smith eddy-viscosity turbulence model and the Chen-Tyson transition model. Required inputs for operation are:

- An external velocity distribution.
- Airfoil surface coordinates.
- Flow Reynolds number.
- A natural transition point estimate (upper and lower).

- The forward stagnation point location.

The program unwraps the surface coordinate onto the x-axis. A Falkner-Skan variable transformation is made to analyze laminar boundary layers and to reduce the turbulent boundary layer growth. The transformed coordinates are nearly independent in the streamwise direction. The Keller-Cebeci box, Newton's, and block tridiagonal methods are used to solve the second order partial differential equations. The program generates output files for graphical visualization and interpretation:

- Skin friction coefficient
- Displacement thickness
- Boundary layer velocity profiles

The laminar, transitional, and turbulent boundary layers are calculated starting from the forward stagnation point. A complete users guide for BL2D.F is located in Appendix A.

1. Program Hints

Convergence is critically dependent on the upper surface transition point input and to a lesser degree on the forward stagnation point input. If the laminar flow calculations indicate flow separation (a separation bubble) before the transition point can be calculated, the wall shear becomes negative causing solution divergence and meaningless results.

Transition from laminar to turbulent flow is indicated where C_f reaches a minimum and then dramatically increases. Separation is indicated when C_f reaches zero or a negative value. Nowak [Ref. 13] discovered that the program can handle mild amounts of separation with a symmetric airfoil at angle-of-attack. She also found that increasing Reynolds number decreases the probability of separation.

C. THE NACA 0012 AIRFOIL

The boundary layer code, bl2d.f, was validated using previously documented results. A test case was completed at a Reynolds number of one million and a transition location of .38 X/C and presented in Figure 3.1. Skin Friction Coefficient, C_f , and Displacement Thickness, δ^* , results agree well with those presented in Cebeci and Bradshaw [Ref. 5].

Nowak [Ref. 13] presented results for this airfoil at a Reynolds number of 540,000. BL2D.F results are shown in Figures 3.2 through 3.7. Also presented are the variations in C_f and δ^* outputs as a function of computer type: Indigo and Stardent. Table 3.1 presents the various input parameters. Variation in C_f and δ^* outputs appear dependent on how the specific computer handles C_f as it approaches zero or becomes negative. Michel's empirical estimate also varied between computers, depending on input (See Appendix D). As expected, the transition point moves forward on the upper surface as angle-of-attack (α°) increases. The Stardent computed a

slight separation zone at $4^\circ \alpha$ located at $0.25 X/C$ and at $6^\circ \alpha$ located at $0.05 X/C$ when the Indigo did not. Both computers calculated leading edge suction separation bubbles at $10^\circ \alpha$ but at slightly different positions. Neither computer could arrive at a converged solution at greater than $10^\circ \alpha$.

TABLE 3.1
NACA 0012 (100 PANELS)
INCOMPRESSIBLE FLOW AT $R_e=540,000$
BOUNDARY LAYER TRANSITION INPUTS

		Indigo			Stardent		
α°	Calculated Stagnation Point	Input Stagnation Point	Michel's Estimate	Transition Point	Input Stagnation Point	Michel's Estimate	Transition Point
0	51	51	0.597	0.578	51	0.597	0.545
2	50	51	0.374	0.390	50	0.379	0.380
4	49	48	0.277	0.219	50	0.212	0.301
6	48	48	0.156	0.054	47	0.092	0.070
8	47	47	0.285	0.027	47	0.045	0.044
10	46	46	0.055	0.075	46	0.055	0.041

D. THE SIKORSKY SSC-A09 AIRFOIL

The boundary layer code was run using the SSC-A09 airfoil and calculated C_f and δ^* results as a function of Mach number are illustrated in Figures 3.8 through 3.13. Table 3.2 presents the various input parameters. Again, the general upper surface transition point trend is to move forward as angle-of-attack increases. Compressibility causes C_f and δ^* to be thinner over the angle-of-attack range; the difference

increasing with increasing X/C . Small separation bubbles can be observed at $4^\circ \alpha$ located at $0.1 X/C$, $8^\circ \alpha$ located at $0.03 X/C$, and $9^\circ \alpha$ located at $0.02 X/C$.

TABLE 3.2
SIKORSKY SSC-A09
184 PANELS
BOUNDARY LAYER TRANSITION INPUTS

		$R_e=2E6 \quad (M=0.2)$			$R_e=4E6 \quad (M=0.4)$		
α°	Calculated Stagnation Point	Input Stagnation Point	Michel's Estimate	Transition Point	Input Stagnation Point	Michel's Estimate	Transition Point
0	94	94	0.438	0.65	94	0.027	0.068
2	92	92	0.184	0.63	92	0.146	0.064
4	90	89	0.121	0.49	89	0.115	0.060
6	86	86	0.105	0.075	86	0.079	0.075
8	82	81	0.0163	0.023	82	0.010	0.070
9	81	82	0.0160	0.022	81	0.011	0.022

Figures 3.14 through 3.25 present the upper surface boundary layer velocity profiles as a function of Mach number and angle-of-attack. Increasing Reynolds number appears to thicken the boundary layer profile placing more fluid energy closer to the airfoil surface and resulting in a decrease in the separation bubble size. An example of this can be seen in Figures 3.18 and 3.19 at $4^\circ \alpha$. A large separation bubble appears in the 2,000,000 Reynolds number flow but is greatly reduced in the 4,000,000 Reynolds number flow. The effect can also be observed at $8^\circ \alpha$ in Figures 3.22 and 3.23.

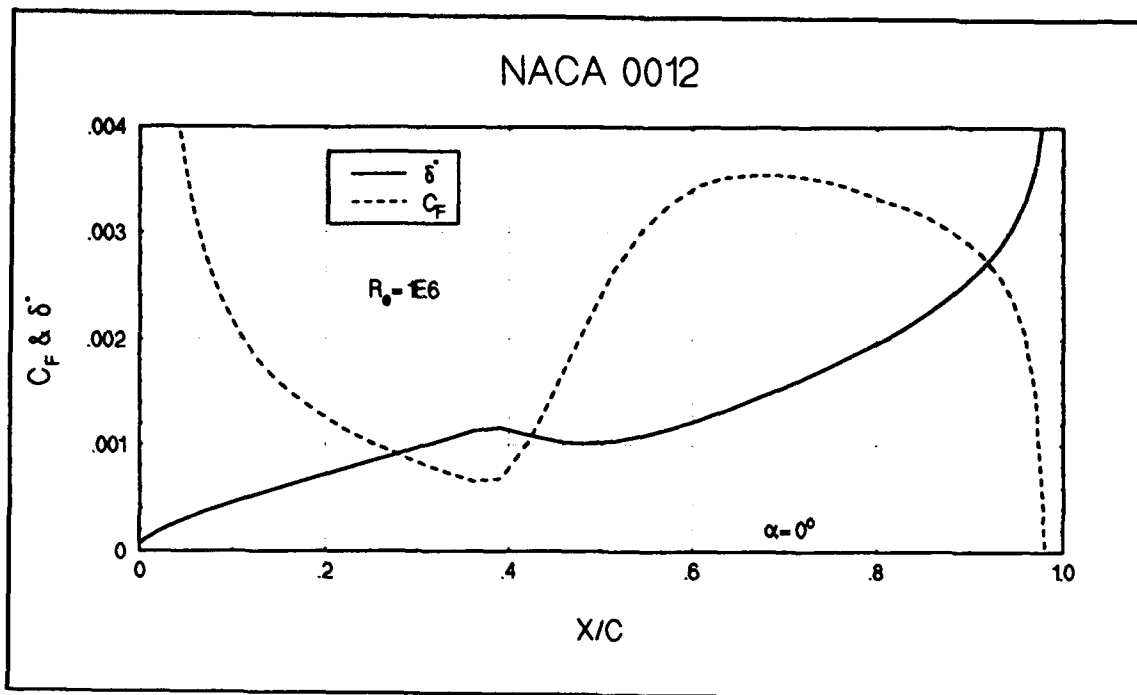


Figure 3.1

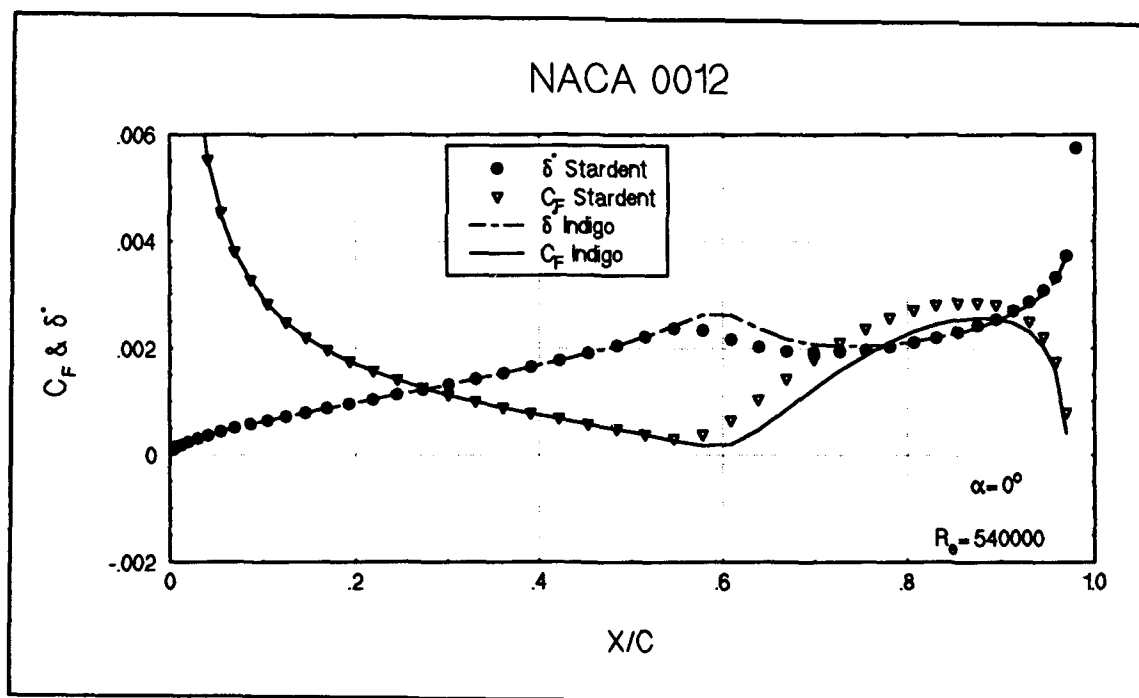


Figure 3.2

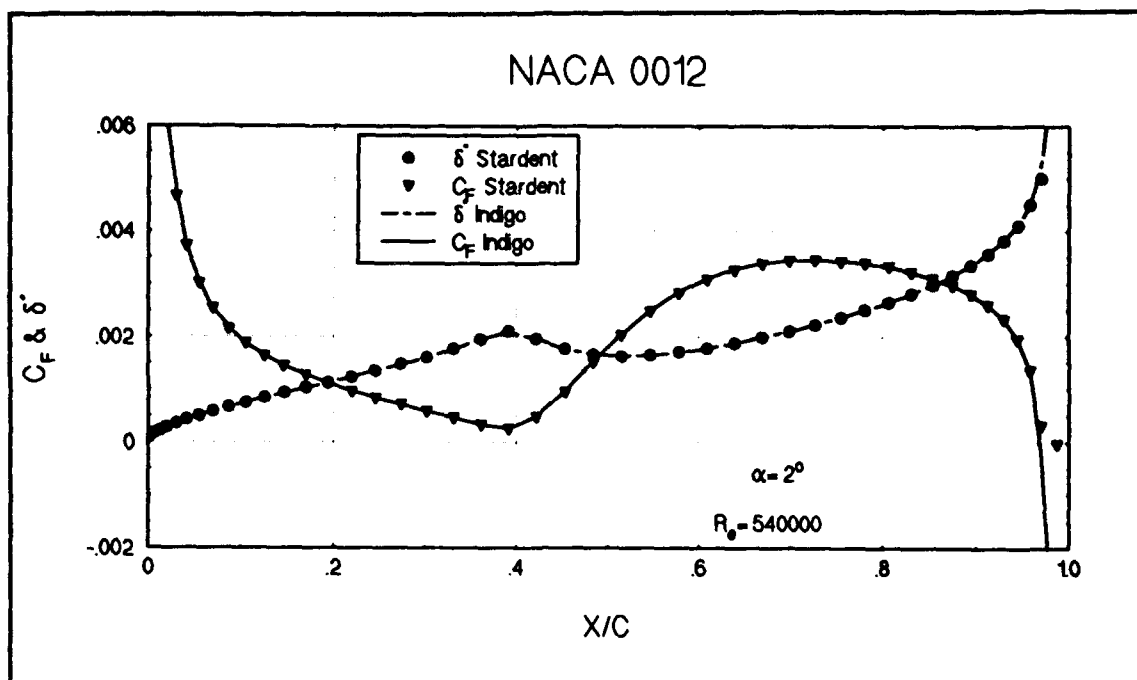


Figure 3.3

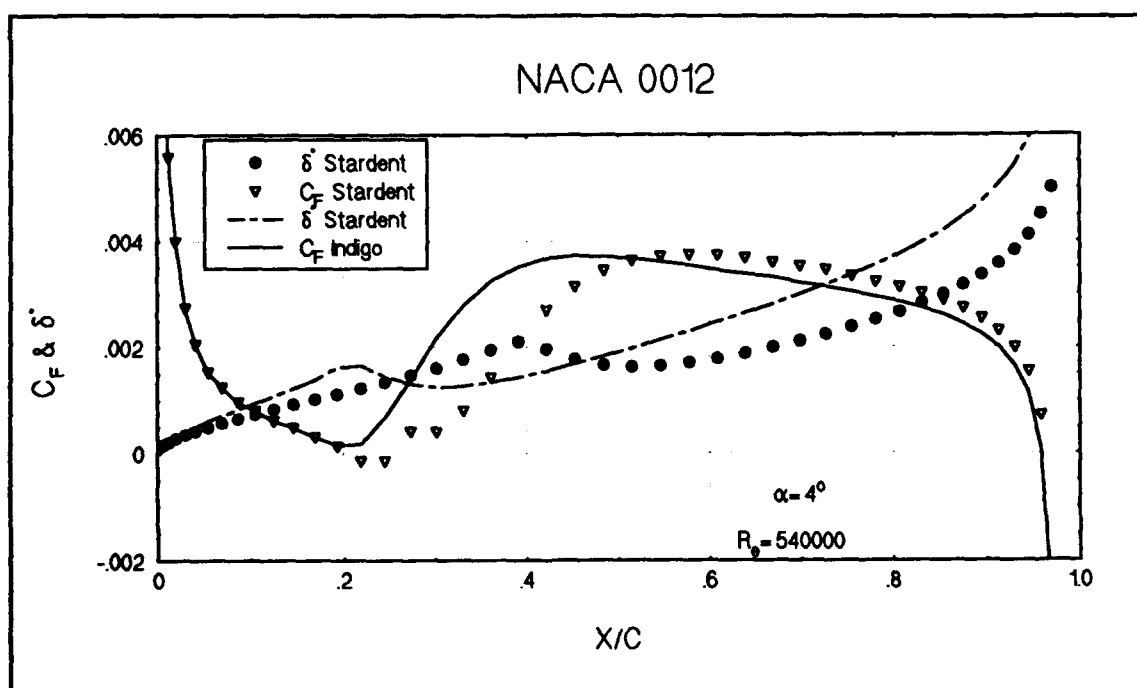


Figure 3.4

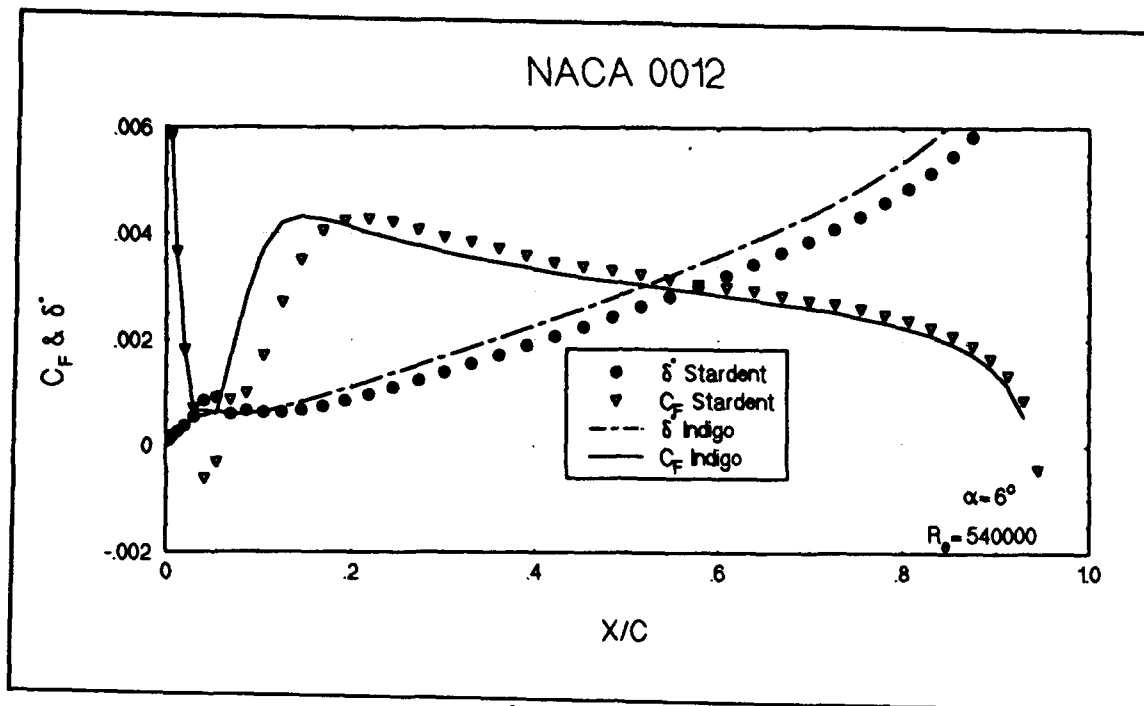


Figure 3.5

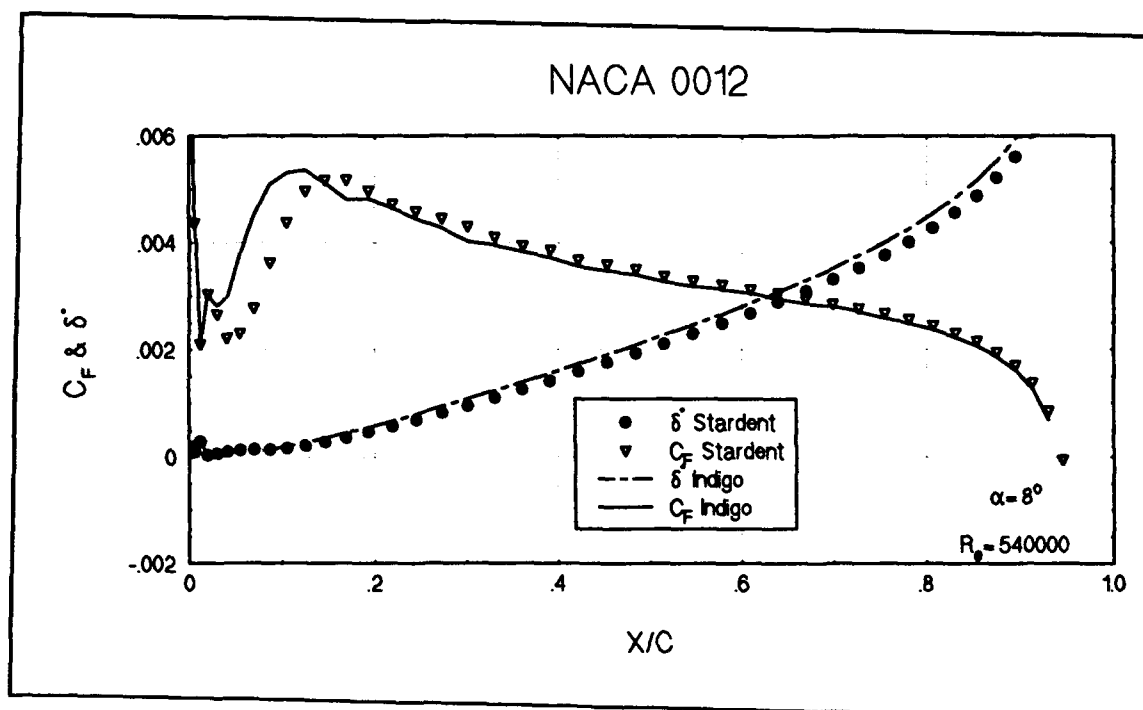


Figure 3.6

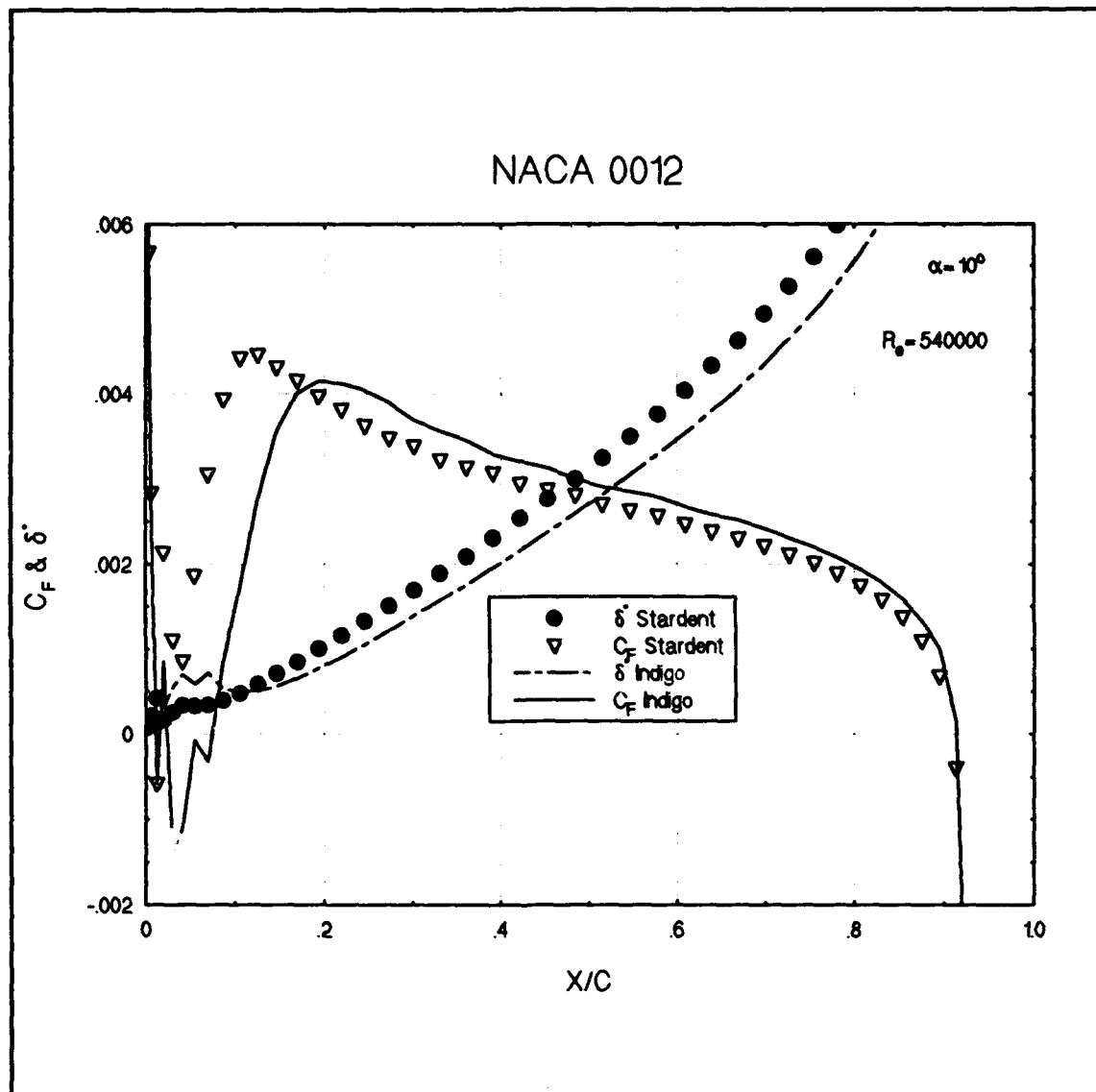


Figure 3.7

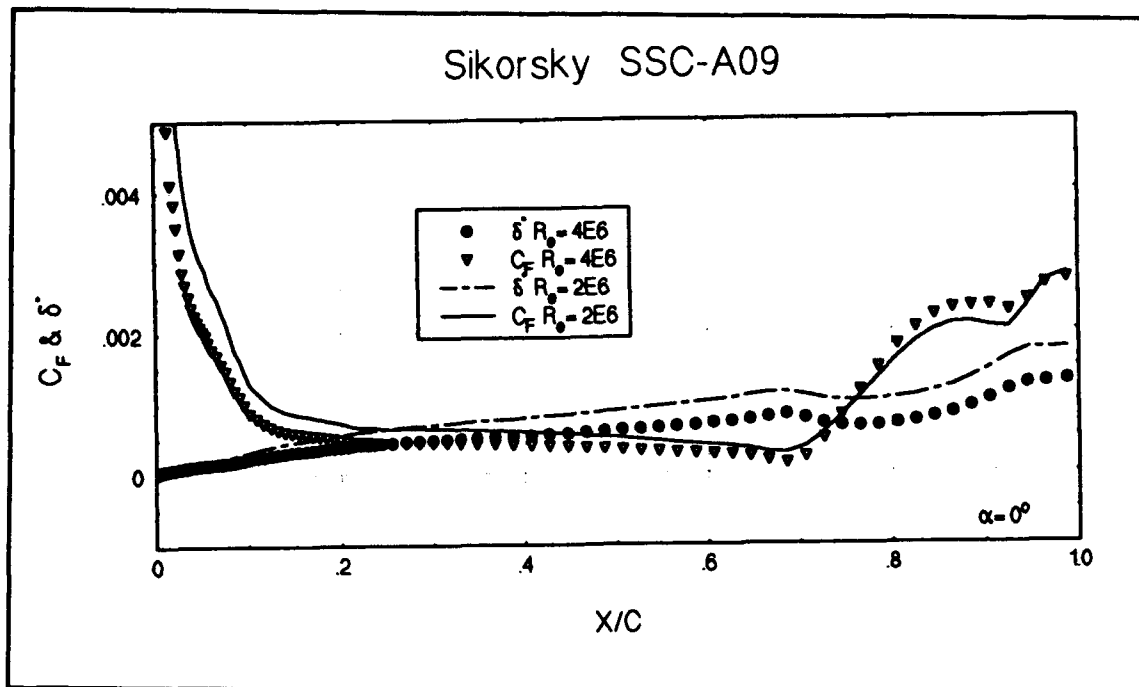


Figure 3.8

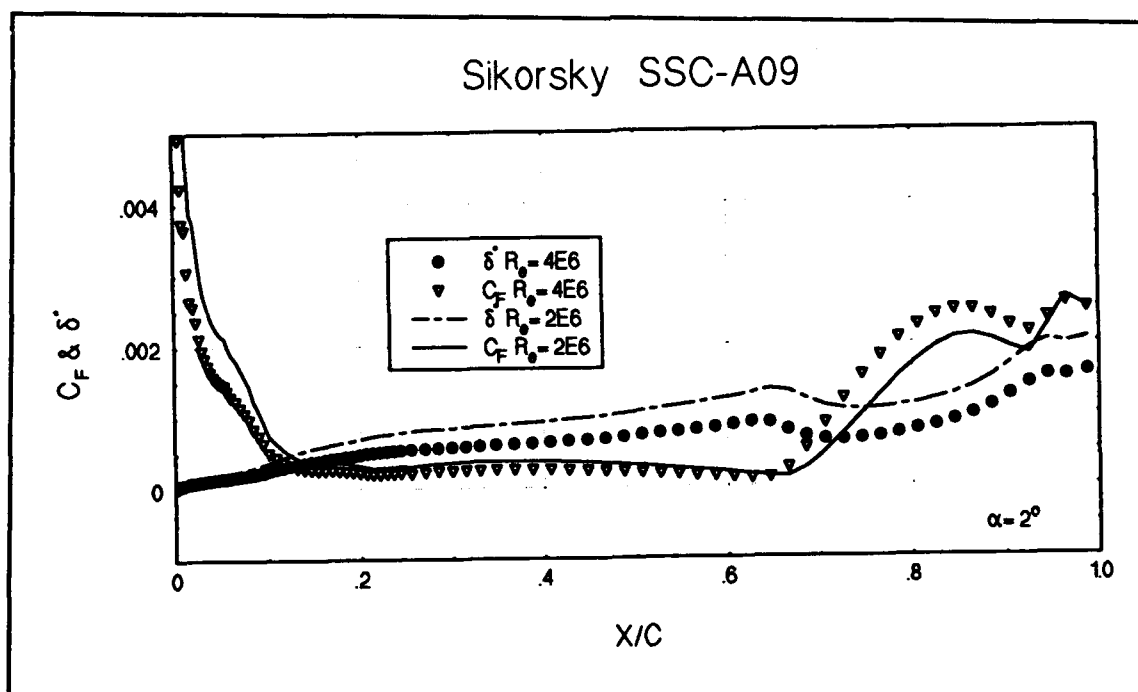


Figure 3.9

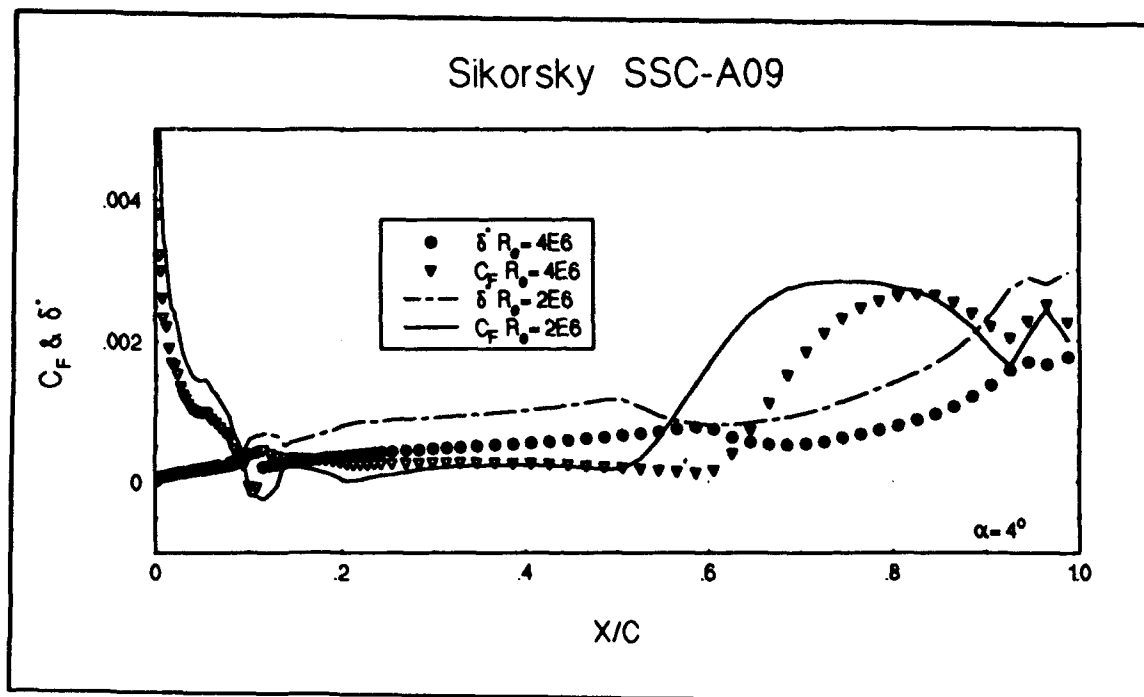


Figure 3.10

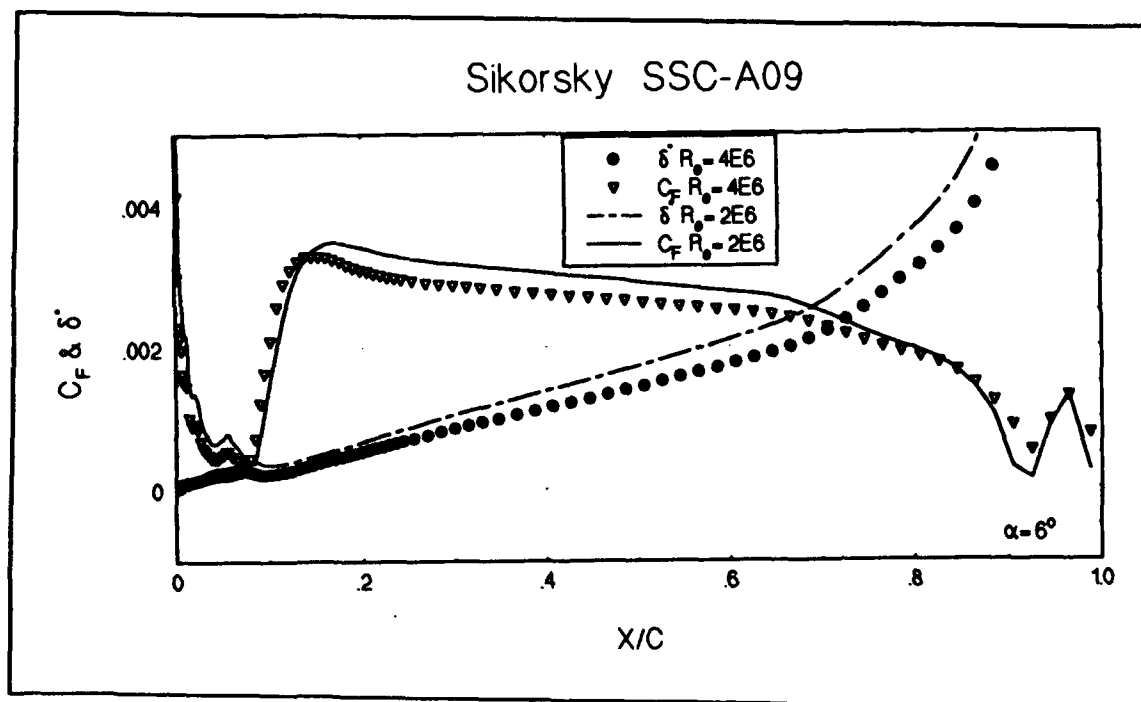


Figure 3.11

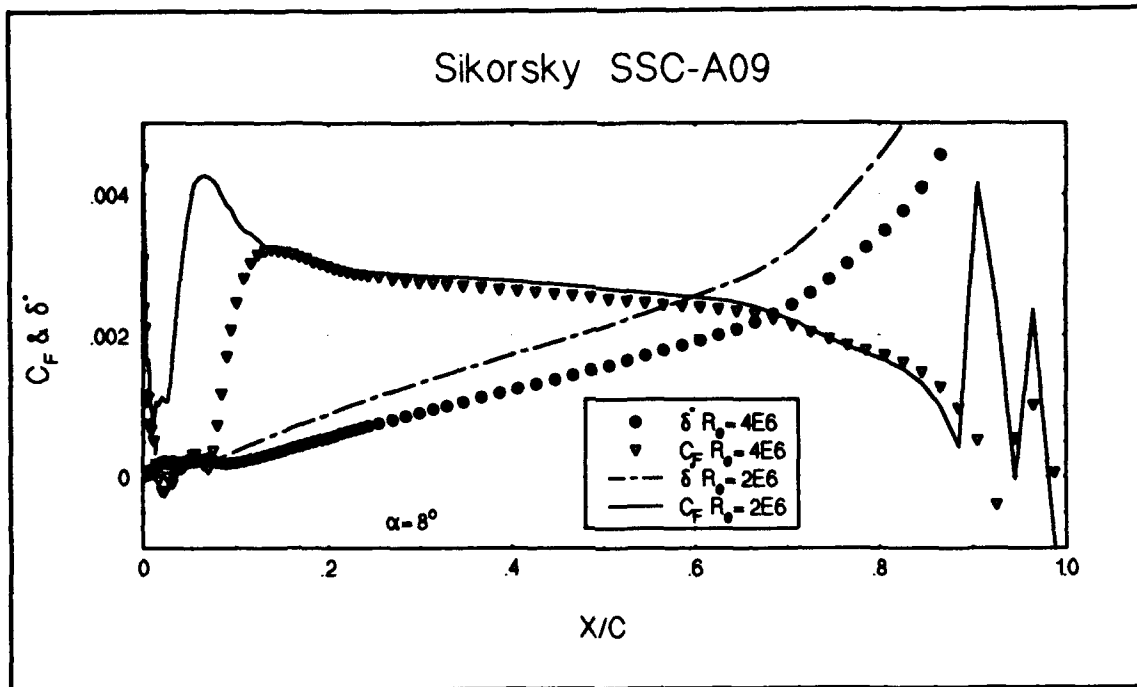


Figure 3.12

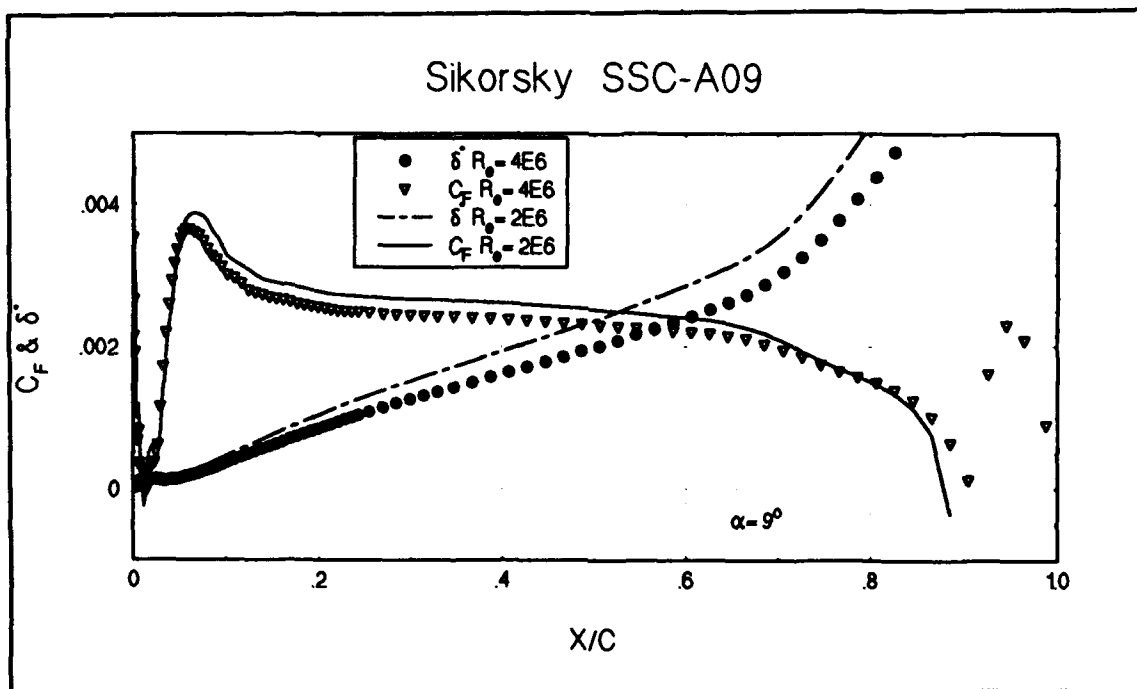


Figure 3.13

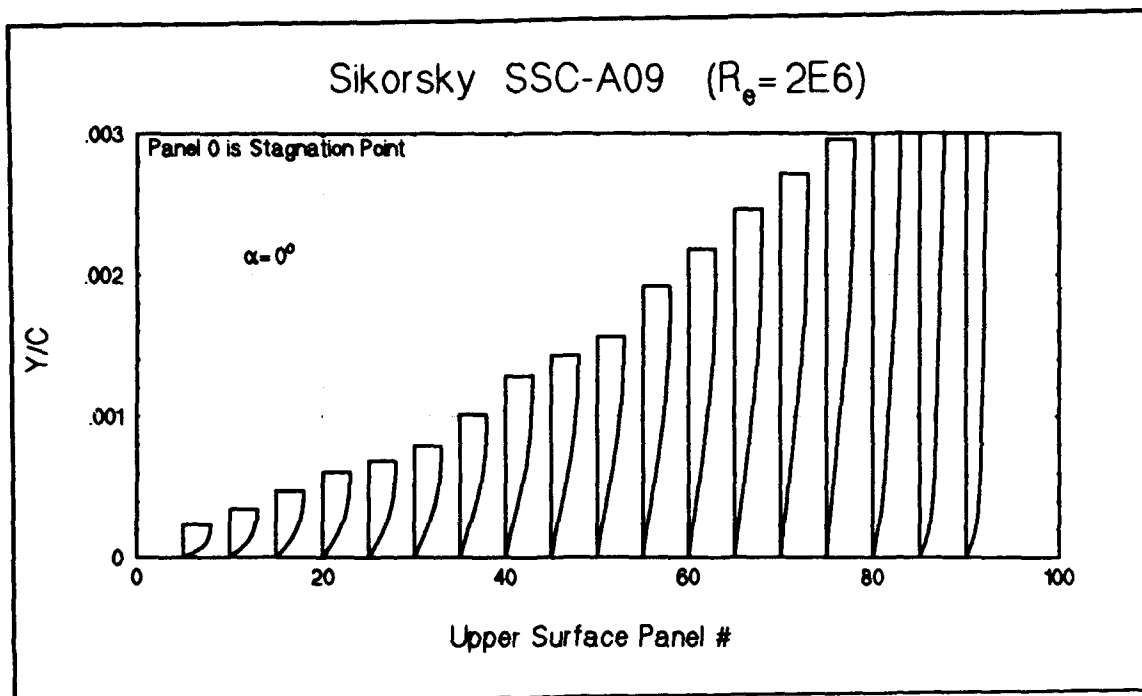


Figure 3.14

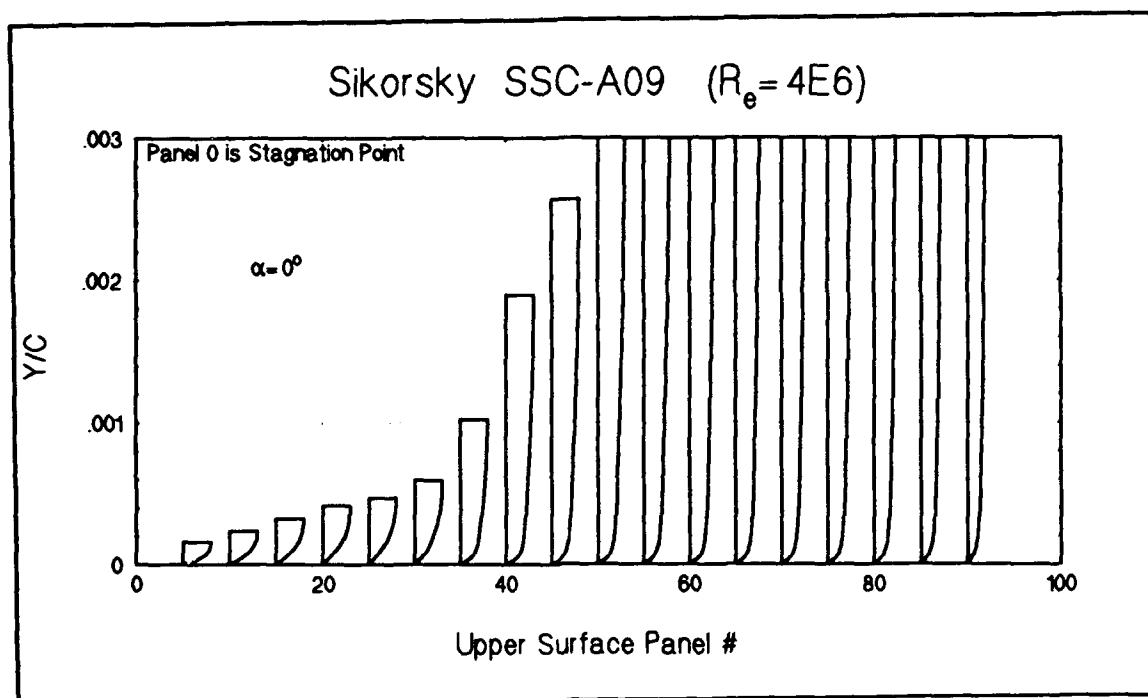


Figure 3.15

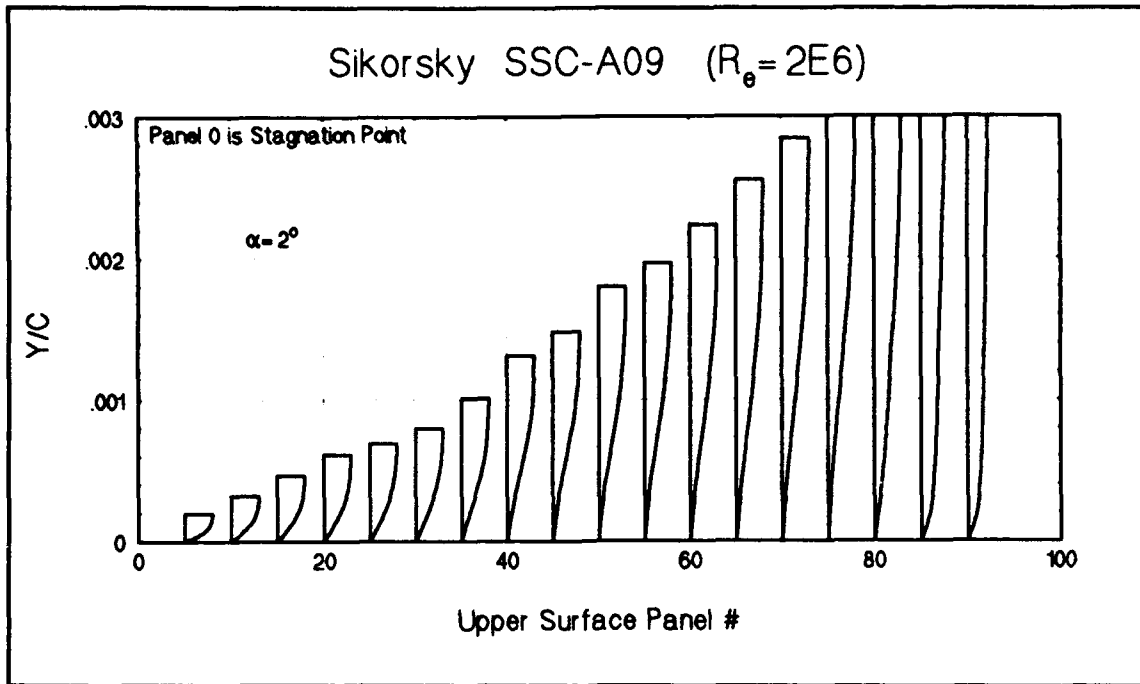


Figure 3.16

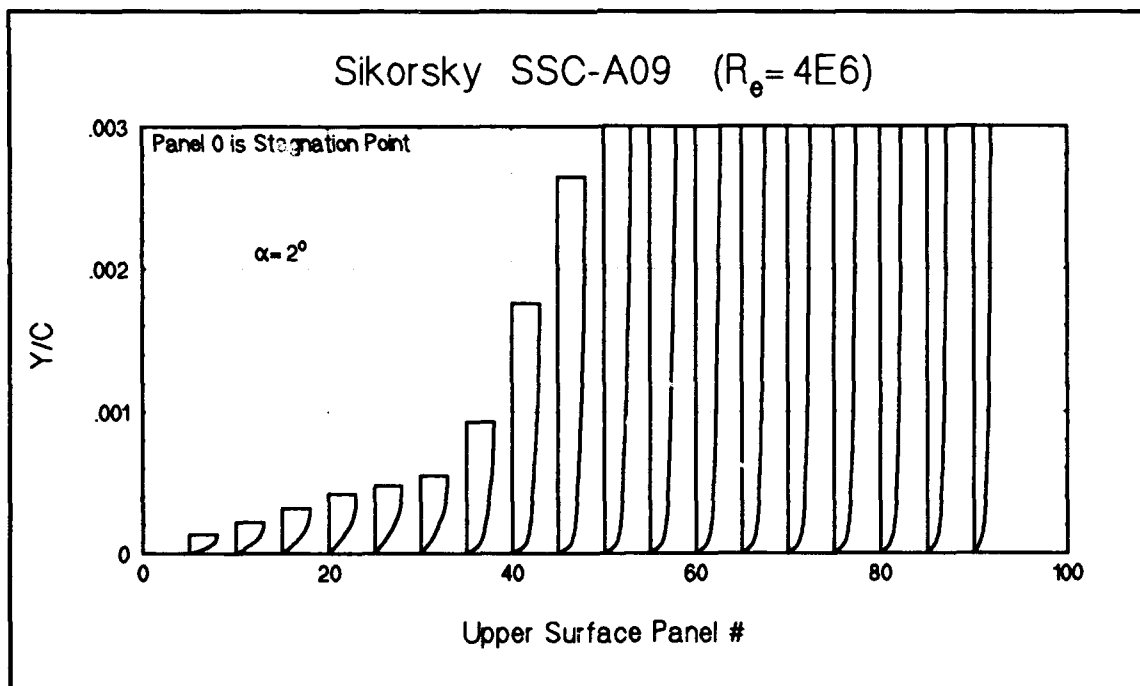


Figure 3.17

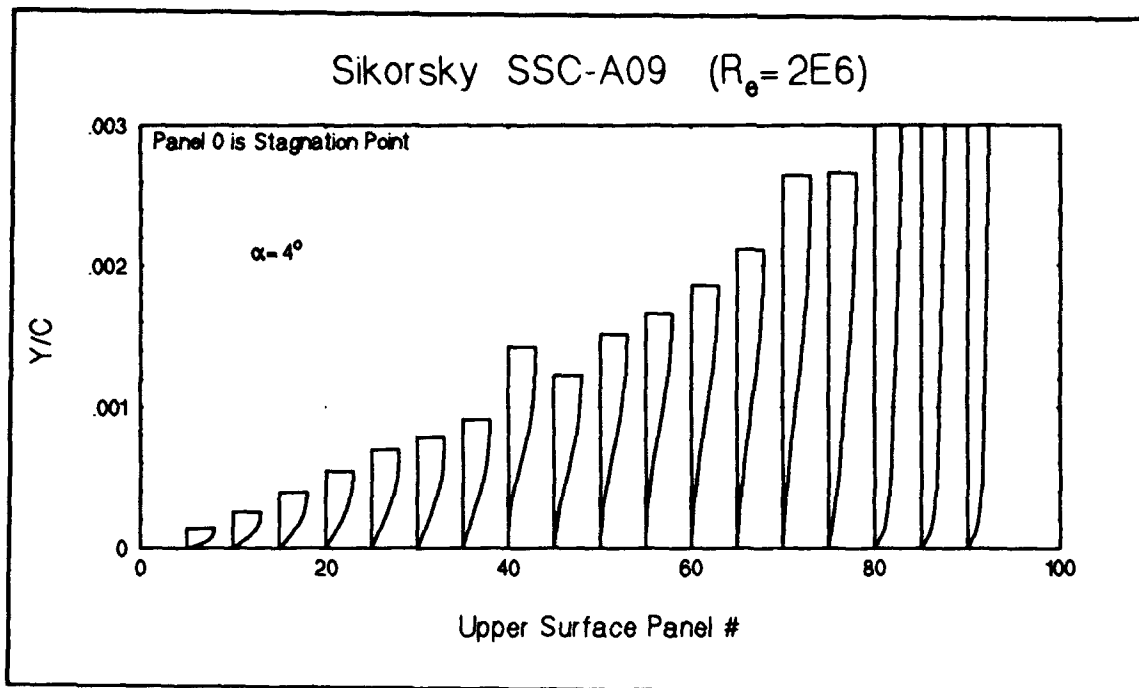


Figure 3.18

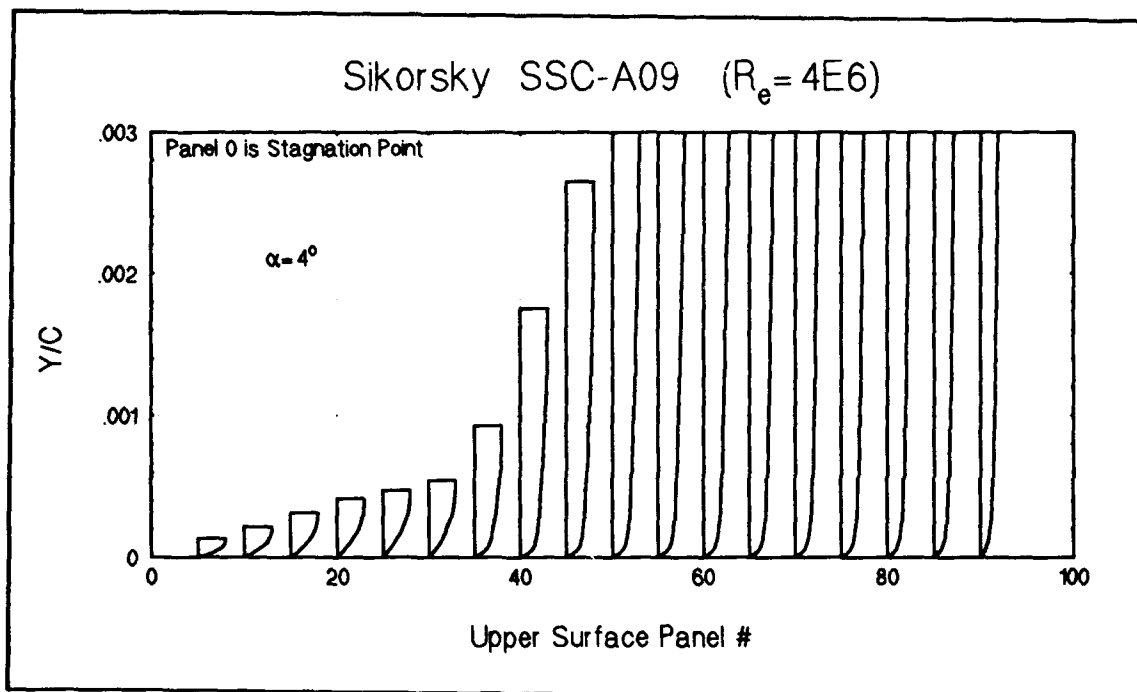


Figure 3.19

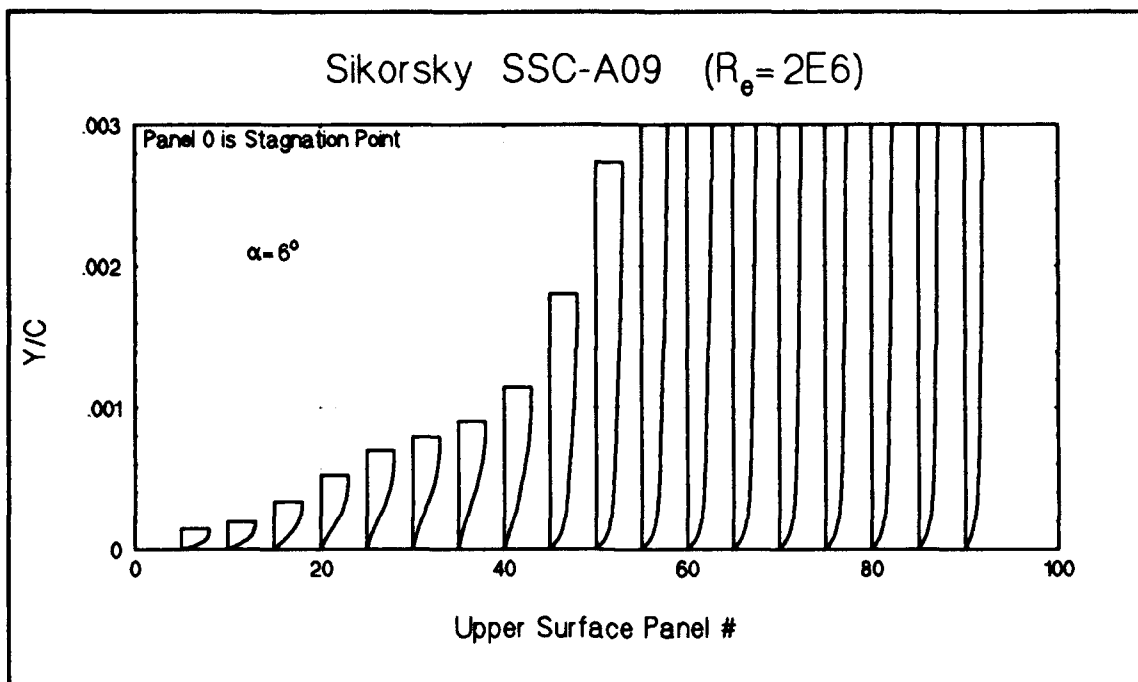


Figure 3.20

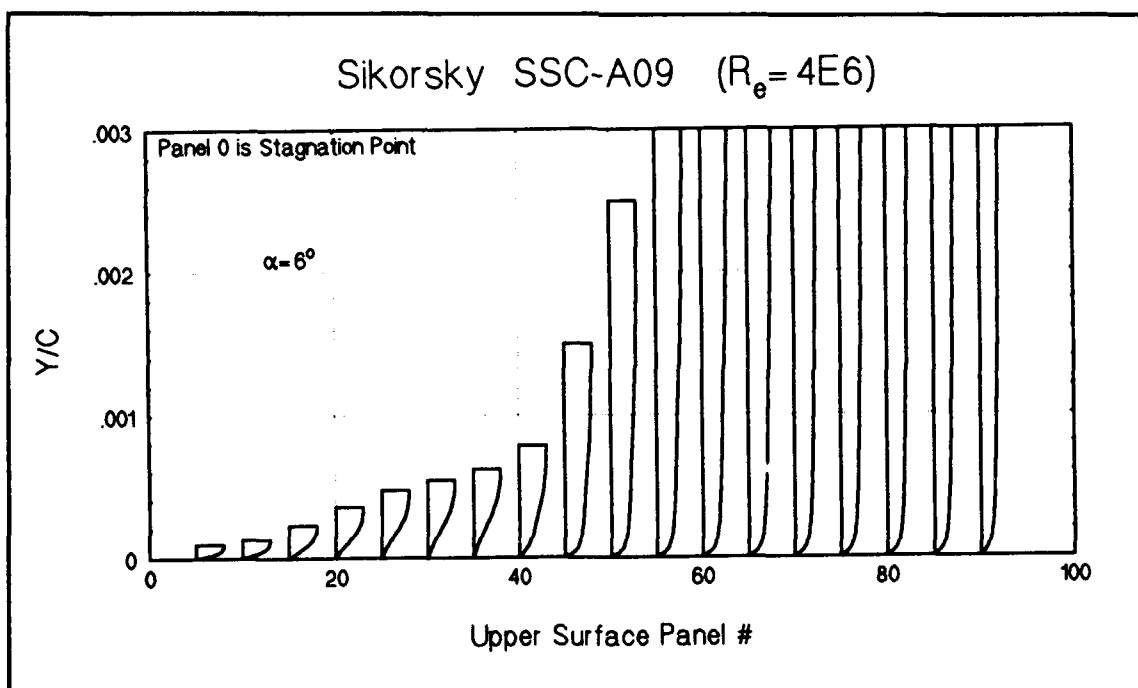


Figure 3.21

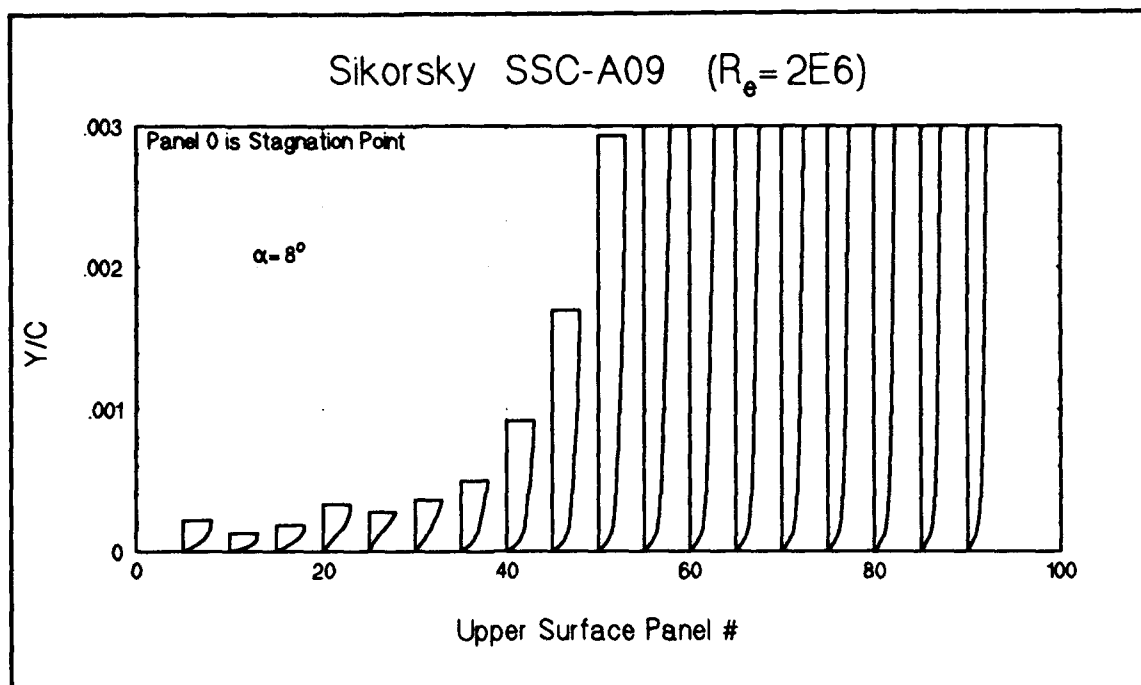


Figure 3.22

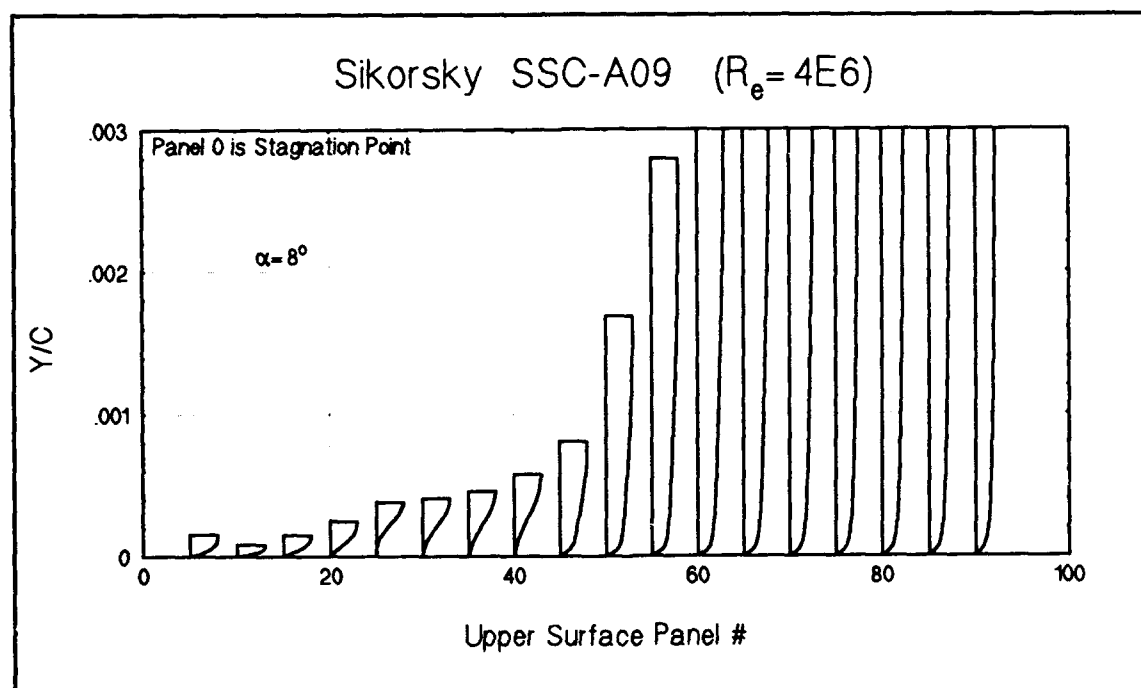


Figure 3.23

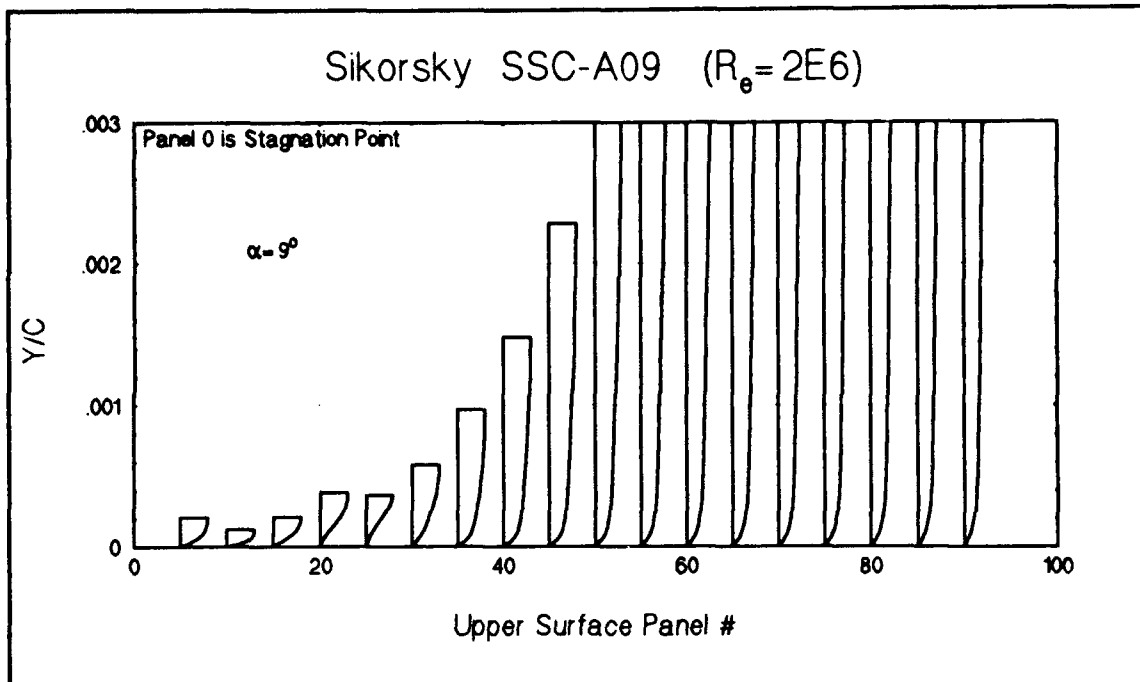


Figure 3.24

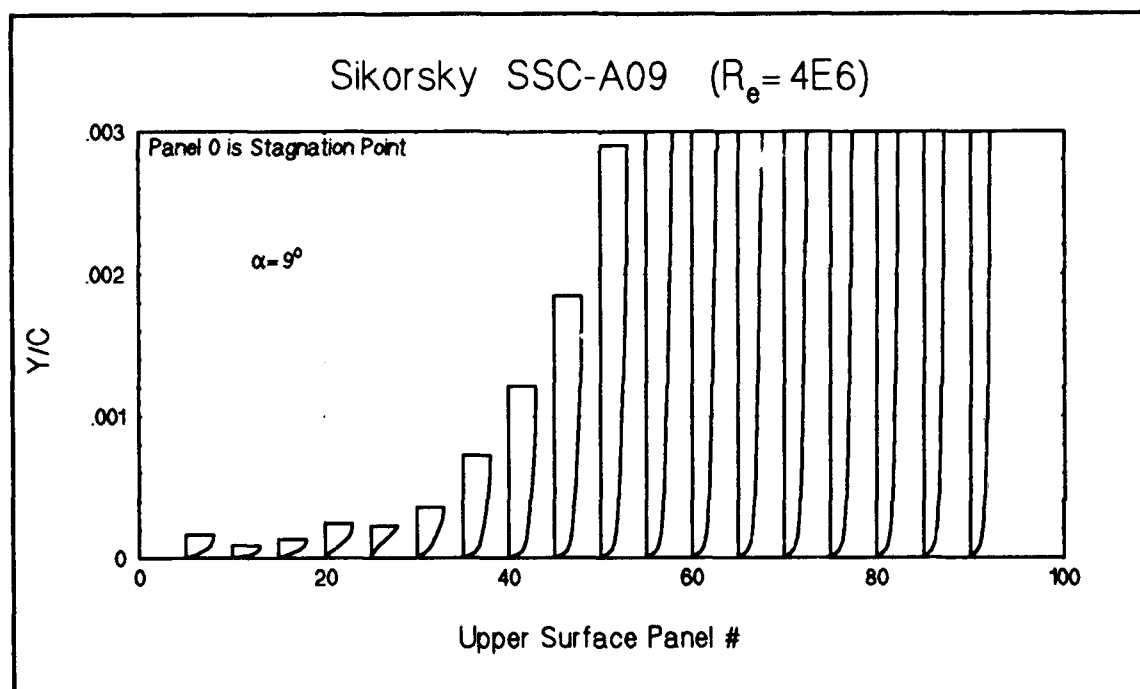


Figure 3.25

IV. UNSTEADY, LINEAR PANEL CODE

A. THEORY/BACKGROUND

The steady, linear panel code used in chapter II is adapted here to unsteady flow by building in a time dependency and modeling the vortex shedding process. Two further assumptions that are required are:

- The viscous flow effect must be negligible.
- The flow must stay attached on the airfoil surface.

Teng [Ref. 15] adapted such a formulation and is used here. This panel method was originally developed by Hess and Smith [Ref. 9] for steady flow. Its extension to unsteady motion was achieved by continuously shedding vorticity into a trailing wake using an interactive solution.

1. Flow Model

Complicating the unsteady flow solution are the now time dependent $N+1$ unknown singularity distributions (sources and vortices). These singularities are given a time index. As before, the source strengths are allowed to vary from panel to panel per time step; and the vorticity is a constant at each time step. The vortex shedding process can be defined through the basic definition of circulation, Equation 4.1, and the Helmholtz theorem of vortex continuity - that potential

flow total circulation must be preserved (Anderson [Ref. 2]). The airfoil perimeter is identified as p .

$$\Gamma_k = - \oint \{ V_k \cdot dS \} = \gamma_k \times p \quad (4.1)$$

Therefore, circulation changes on the airfoil surface must be equal and opposite to the wake vorticity. Thus the shed vorticity model allows a mechanism for communication between time steps.

2. Boundary Conditions

The flow tangency and Kutta conditions are no longer linear. This requires an iterative numerical solution scheme. The flow tangency condition remains the same. The Kutta condition must now include the trailing edge panel's potential rate of change.

3. Solution Scheme

The disturbance potential, Equation 4.2, is complicated by adding in potential contributions from the shed vorticity panels and the wake core vortices. The disturbance potential must be calculated at every control point at each time step taking great care to only include velocity contributions due to disturbances. Complete modeling, numerical solution scheme, and disturbance potential details can be found in Teng [Ref. 15].

$$\Phi = \{ \Phi_{\infty} + \Phi_{source} + \Phi_{vortex} + \Phi_{shed\ vortex} + \Phi_{core\ vorticity} \}$$

A complete program (UPOT.F) user's guide with input and output file examples and the source code are located in Appendix B. A few non-dimensional parameters must first be clarified. The Reduced Pitch Rate (A) used to classify ramp motion and the Reduced Frequency (k) used in sinusoidal motion can be based on full or half-chord. Program UPOT.F uses full chord, but the Lorber and Carta experimental results [Ref. 11] use half-chord as shown in equations 4.3 and 4.4.

$$A = \left\{ \frac{\dot{\alpha} C}{2 U} \right\} \quad (4.3)$$

$$k = \left\{ \frac{\omega C}{2 U} \right\}, \quad \omega = \text{Osillation Frequency} \quad (4.4)$$

B. THE NACA 0012 AIRFOIL

Calculated force and moment coefficients as a function of angle-of-attack during a 0.005 Reduced Pitch Rate ramp motion are displayed in Figures 4.1 through 4.3. Steady state results from chapter two are also displayed for comparison. Very little noticeable difference is noted in lift or moment coefficient.

Results for a sinusoidal motion with a Reduced Frequency of 0.025 and a pitch magnitude of 12° are shown in Figures 4.4 through 4.6, again with steady state results previously obtained. Results for a sinusoidal motion with a Reduced Frequency of 0.05 and a pitch magnitude of 20° are shown in

Figures 4.7 through 4.9. Here, lift is augmented throughout the down cycle and lost during the up cycle, contrary to what would be expected in experiment. Similar results are shown for the drag and the moment coefficients.

C. THE SIKORSKY SSC-A09 AIRFOIL

To validate the Lorber and Carta experimental data, a ramp motion with a Reduced Pitch Rate of 0.005 and sinusoidal motions with Reduced Frequencies of 0.025 and 0.05 were completed. UPOT.F unsteady, PANEL.F steady, and Lorber and Carta experimental data are presented in Figures 4.10 through 4.18.

1. Ramp Motion, $A=0.005$ (0° to 20°)

The calculated steady state and unsteady lift coefficient varied little from the experimental results throughout the linear range (0° to 14°). Once nonlinear, viscous effects dominated the real flowfield, calculated results diverged as expected. Steady and unsteady calculated drag results follow the general direction of experimentally measured values but differ widely in magnitude due to the basic inviscid flow assumptions. Pitching-moment coefficient agrees well with experimental results through approximately $11^\circ \alpha$.

2. Sinusoidal Motion

a. $\alpha(t) = 6 - 6\cos(\omega t)$, $k=0.025$, $0^\circ \sim 12^\circ \sim 0^\circ$

Computed unsteady, inviscid, incompressible results indicate a net loss of lift on the up cycle and augmented lift on the down cycle when compared to PANEL.F computed steady state values. UPOT.F overpredicts lift on both up and down cycles when compared to experiment. Pitching-moment coefficient results show a similar overall trend but are also displaced by a constant magnitude from experimentally measured results.

b. $\alpha(t) = 10 - 10\cos(\omega t)$, $k=0.05$, $0^\circ \sim 20^\circ \sim 0^\circ$

Similar results were obtained when the oscillation magnitude and Reduced Pitch rate were increased. Computed unsteady, inviscid, incompressible results indicate a net loss of lift on the up cycle and augmented lift on the down cycle when compared to PANEL.F computed steady state values. No reasonable correlation could be drawn from the drag results due to the code's inviscid flow assumption. The general pitching-moment coefficient trend is similar within the linear region, but varies wildly within the nonlinear region.

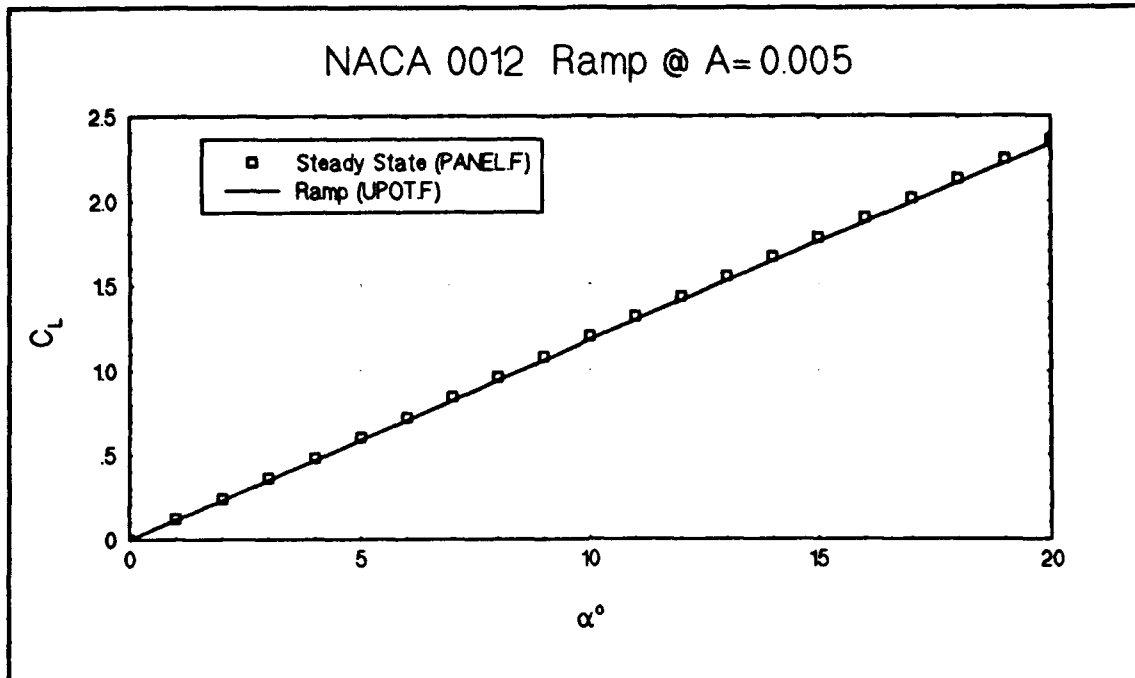


Figure 4.1

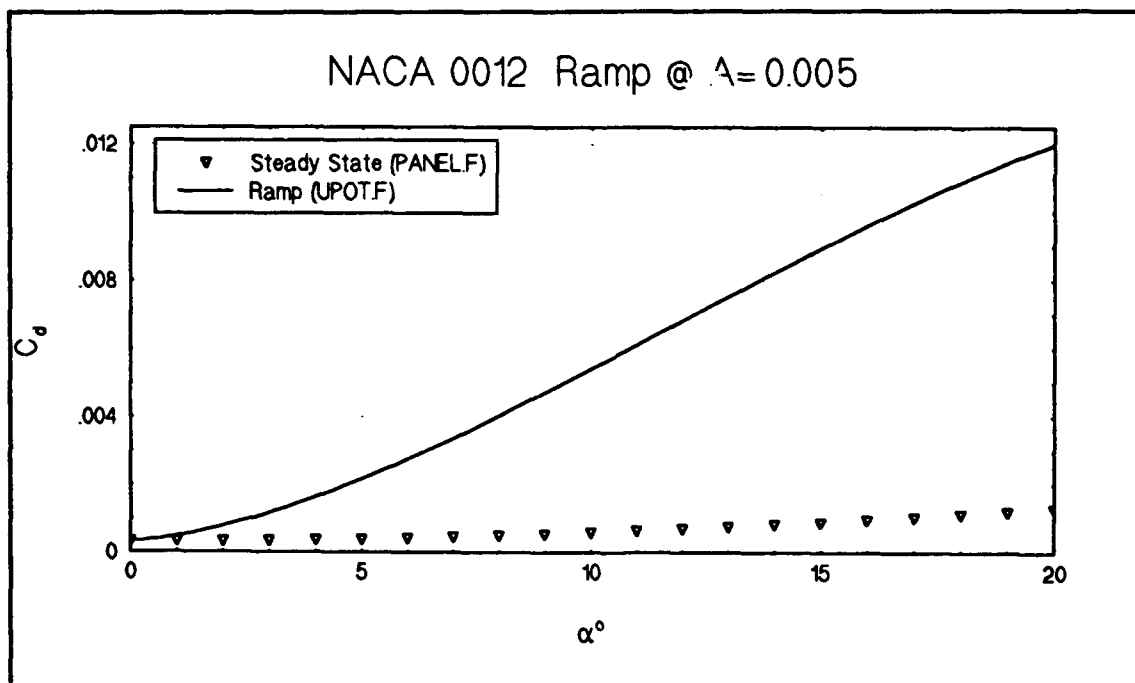


Figure 4.2

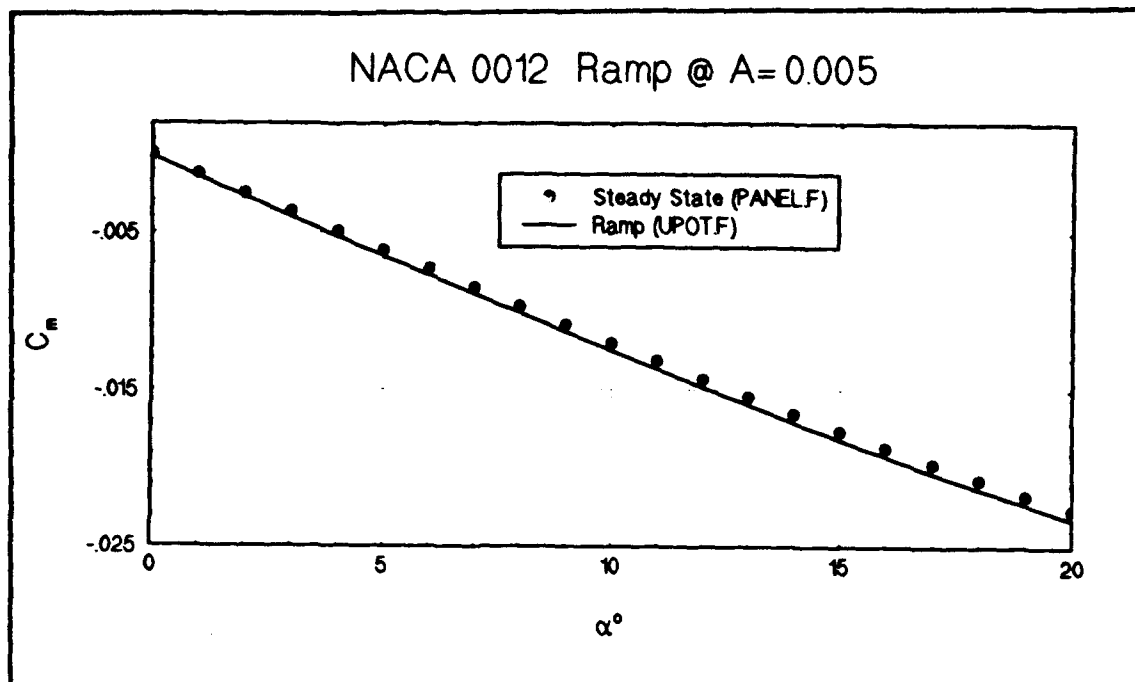


Figure 4.3

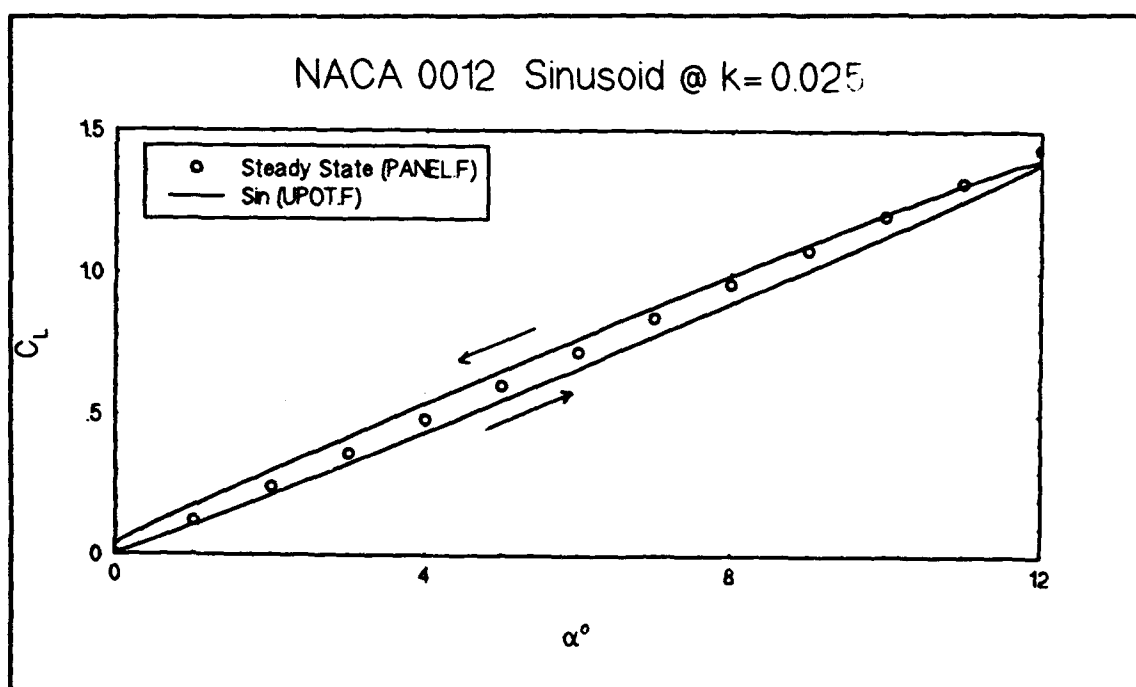


Figure 4.4

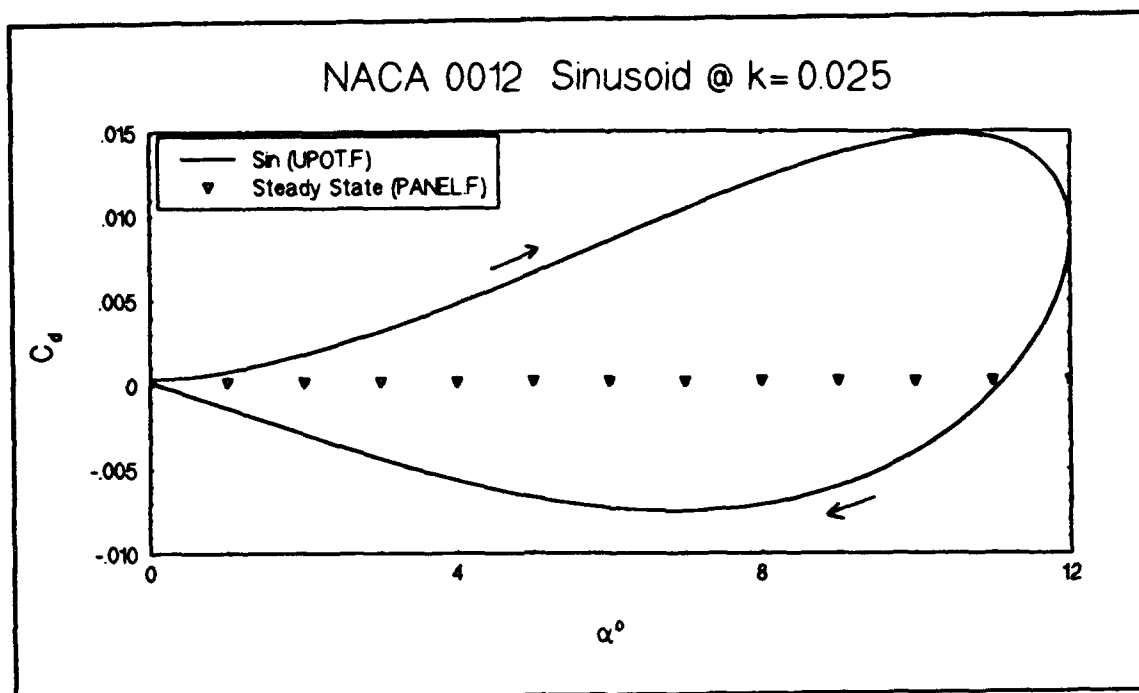


Figure 4.5

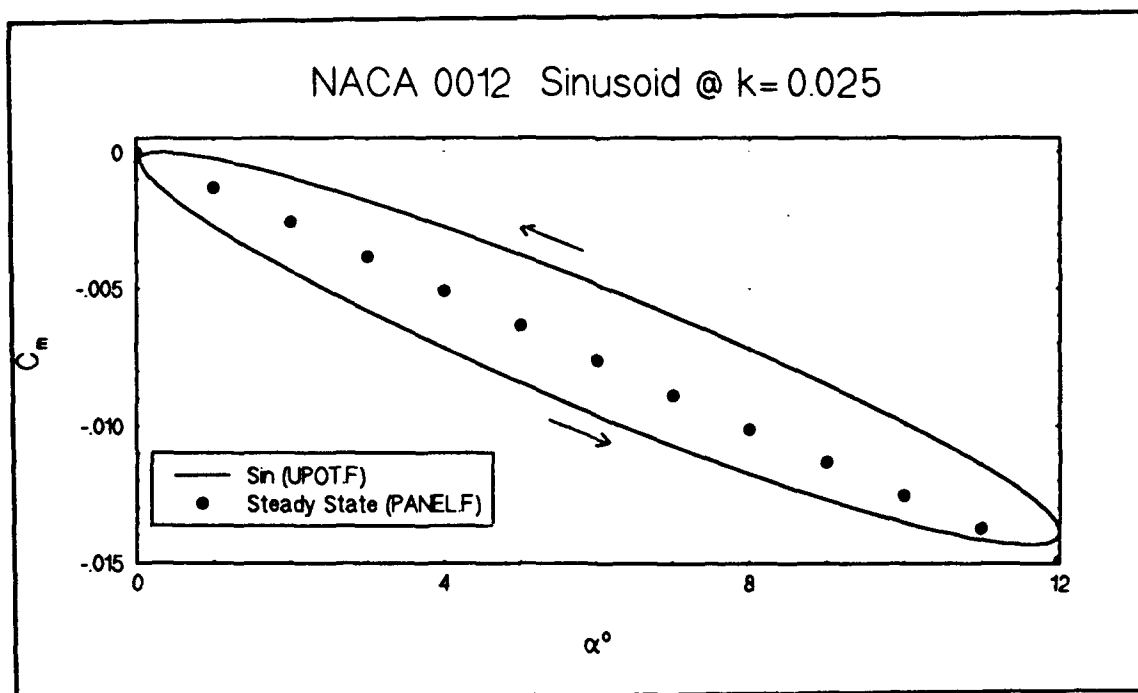


Figure 4.6

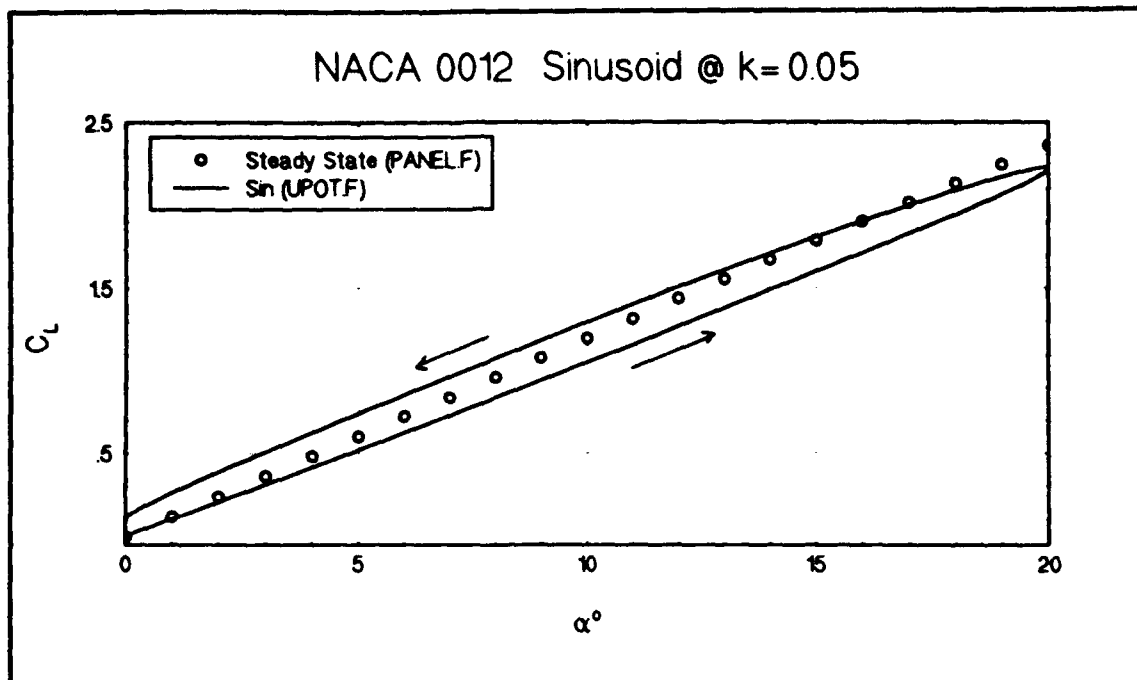


Figure 4.7

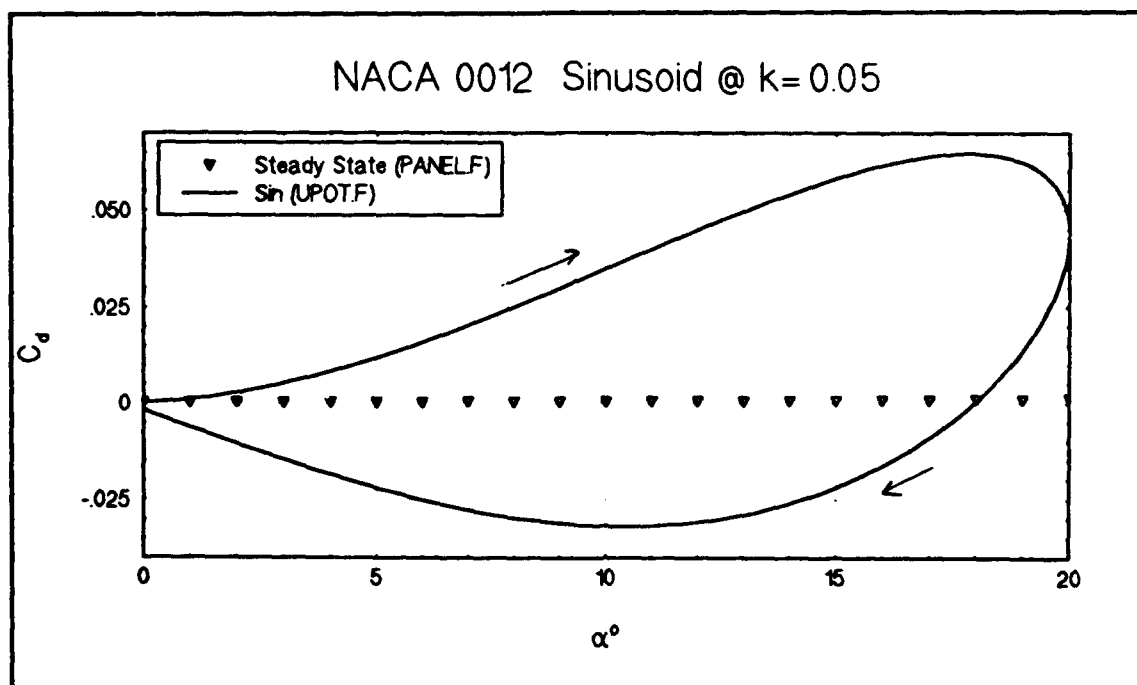


Figure 4.8

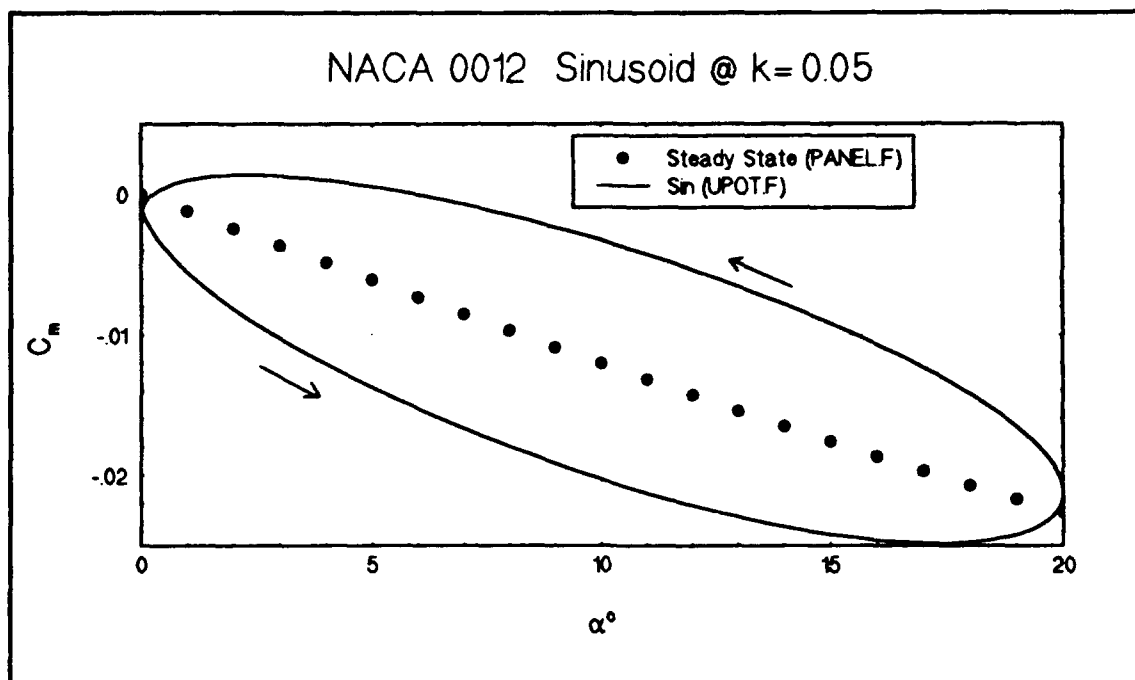


Figure 4.9

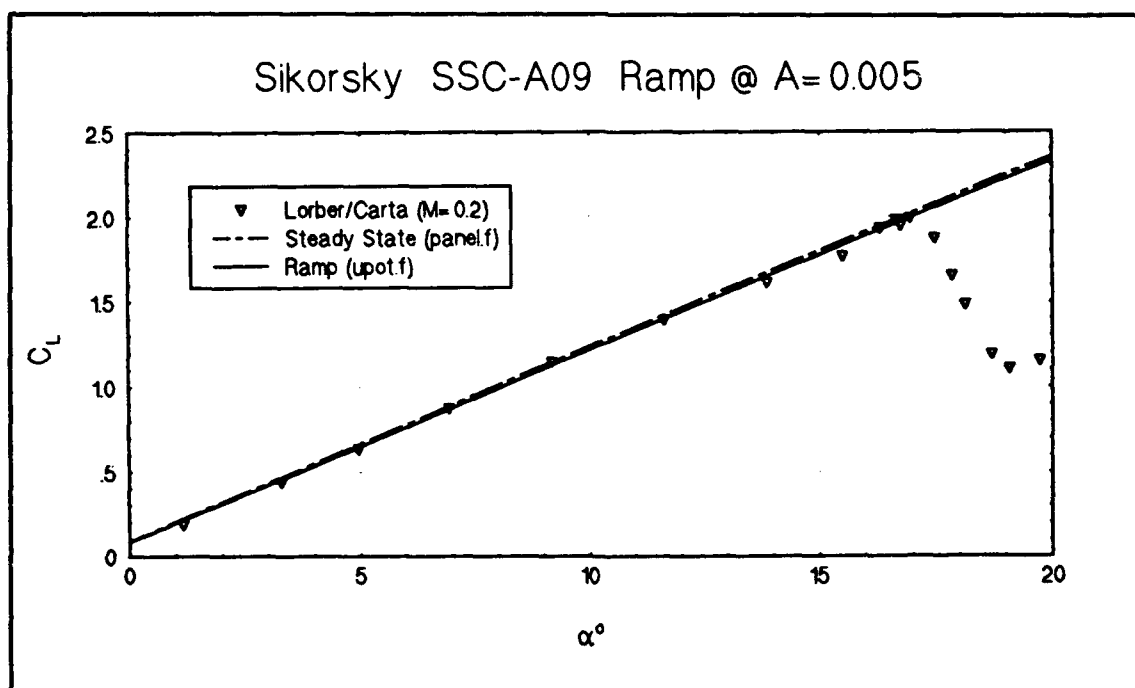


Figure 4.10

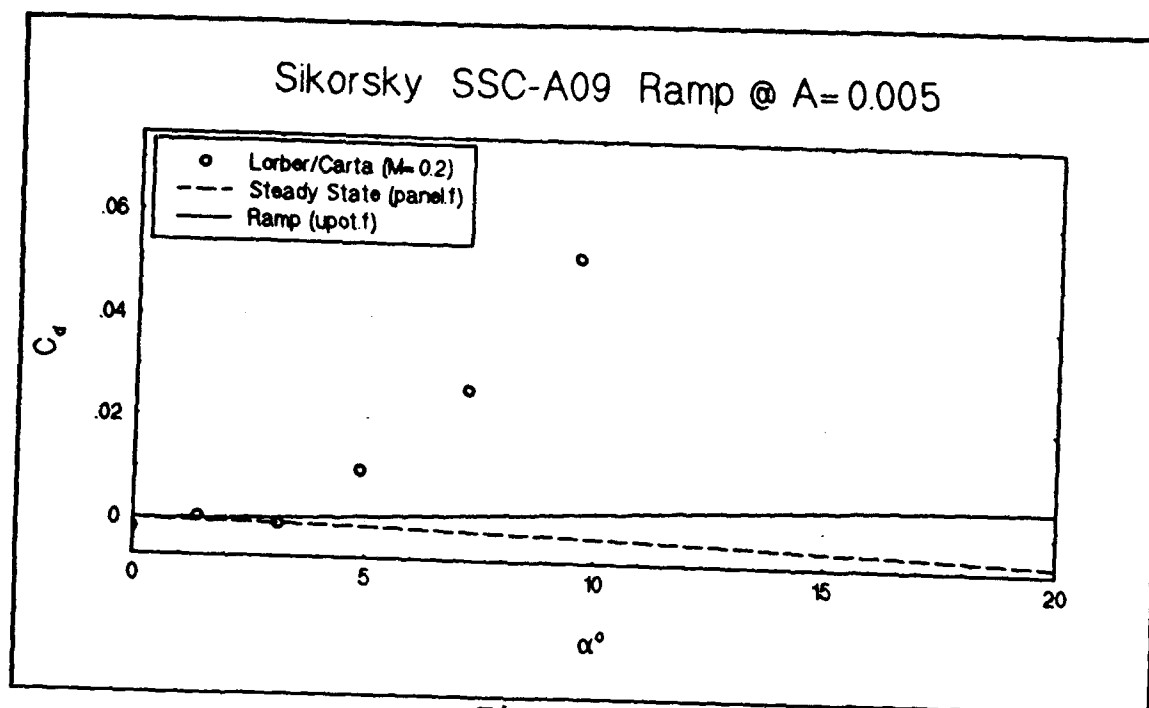


Figure 4.11

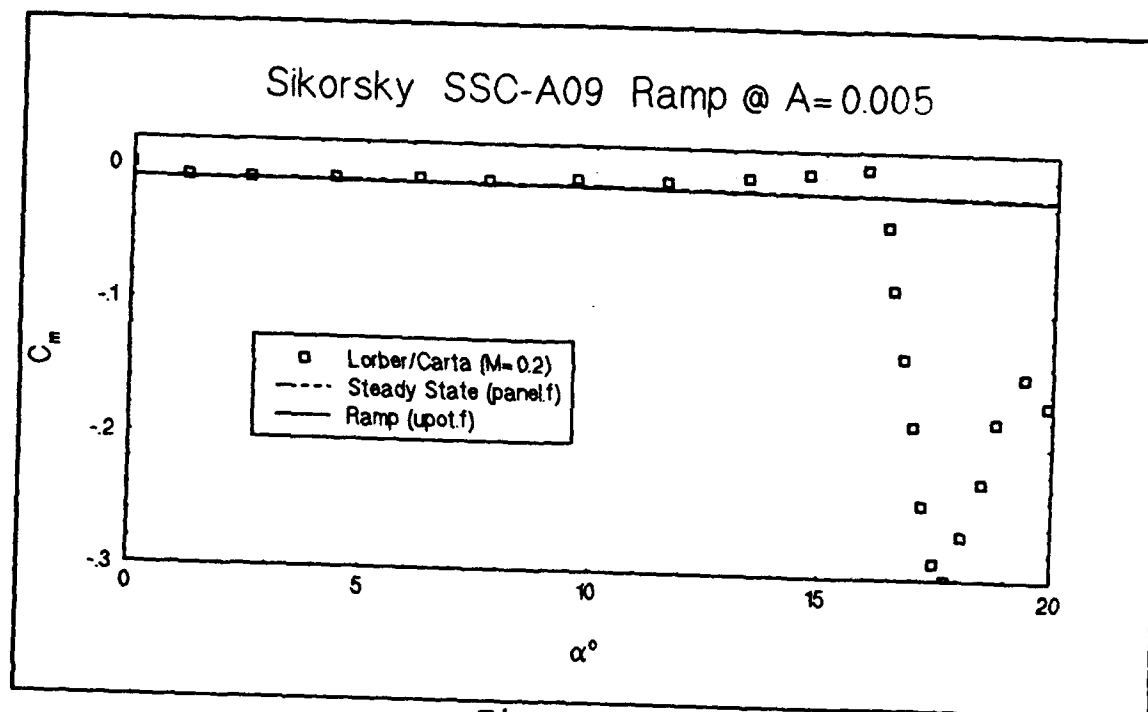


Figure 4.12

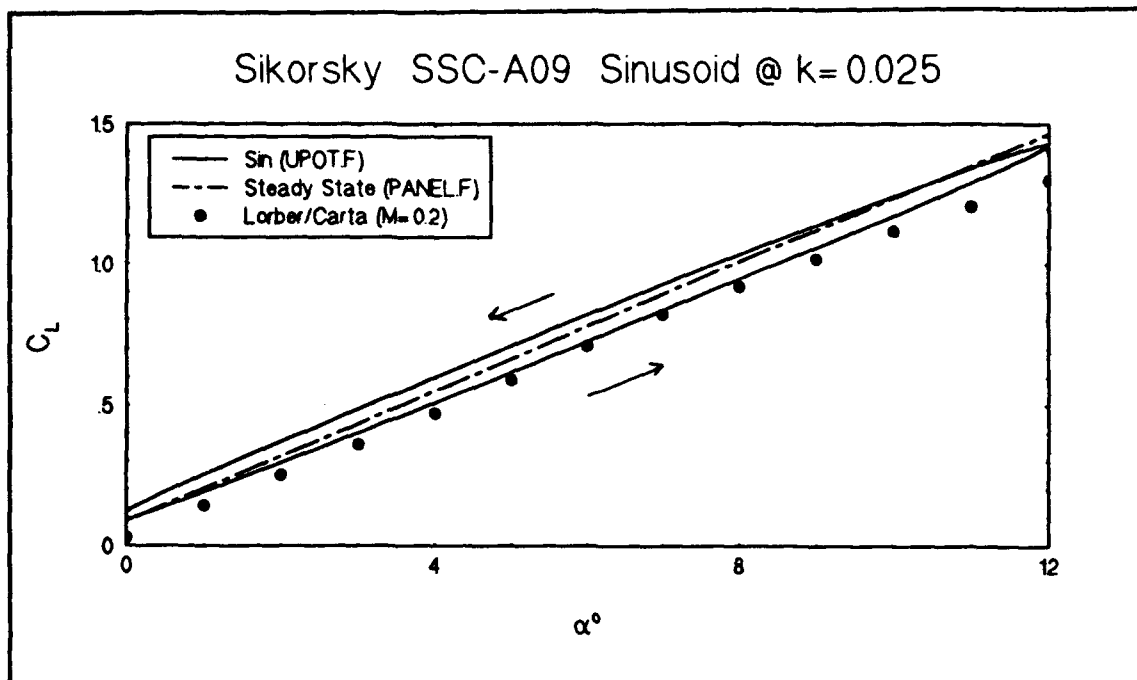


Figure 4.13

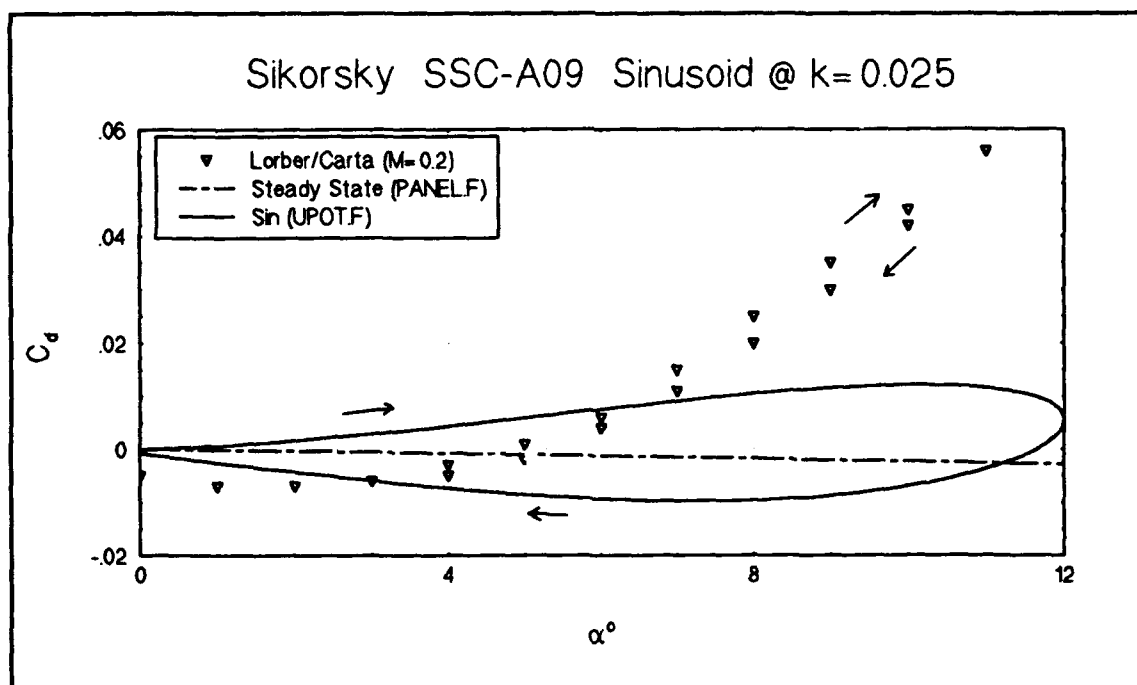


Figure 4.14

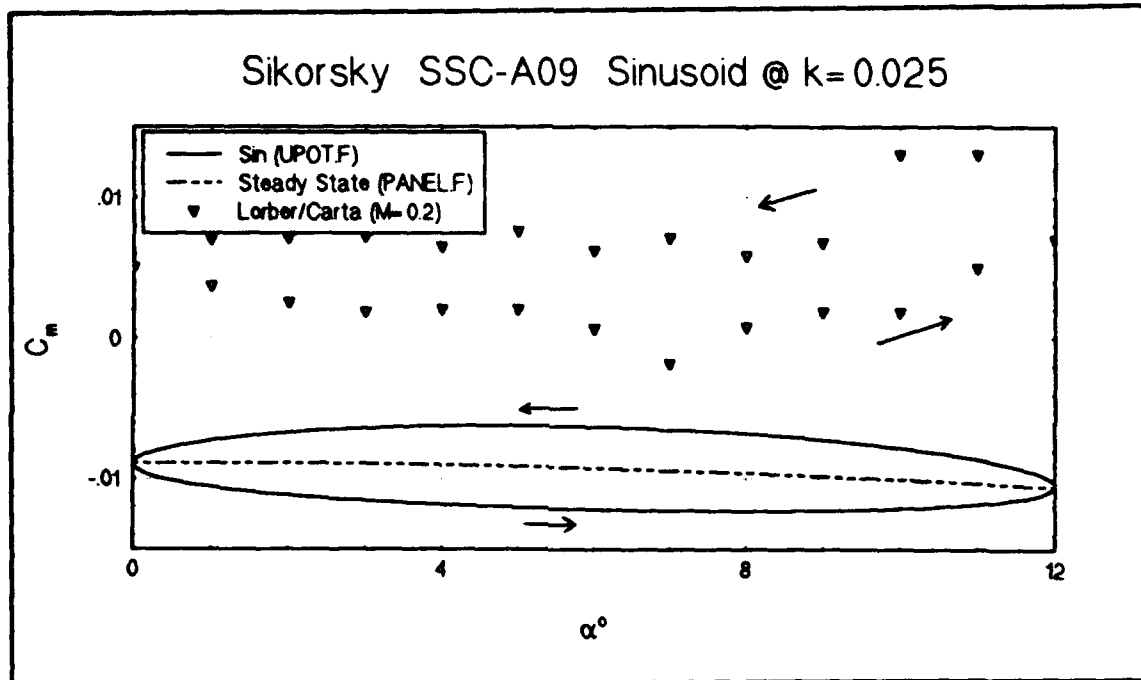


Figure 4.15

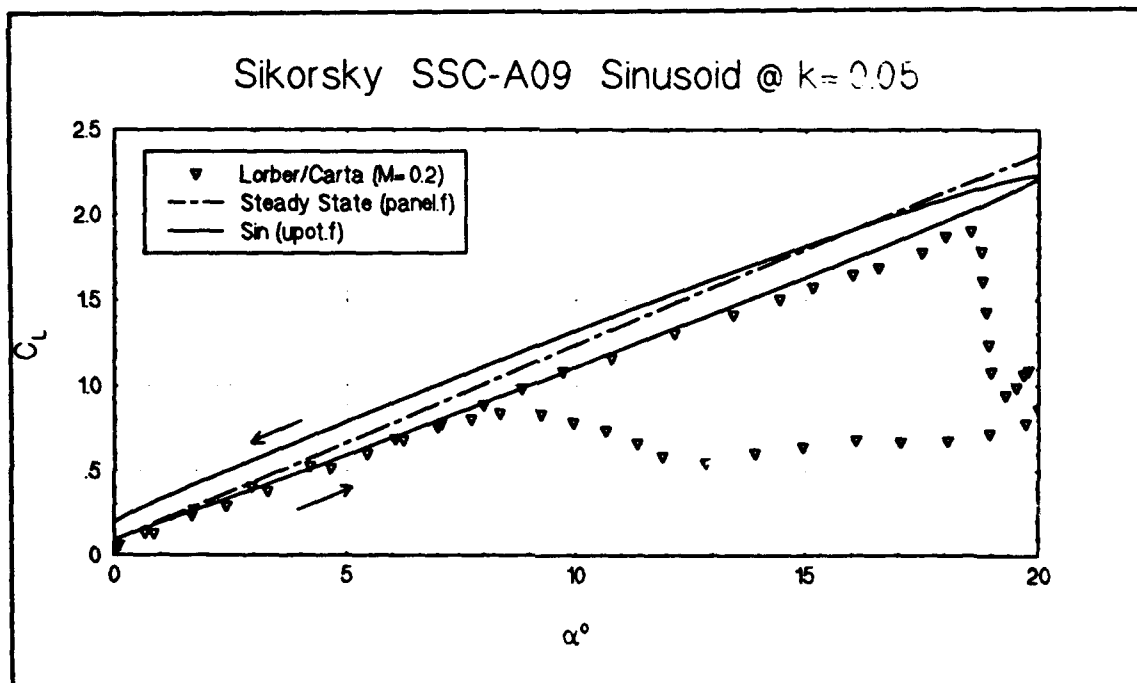


Figure 4.16

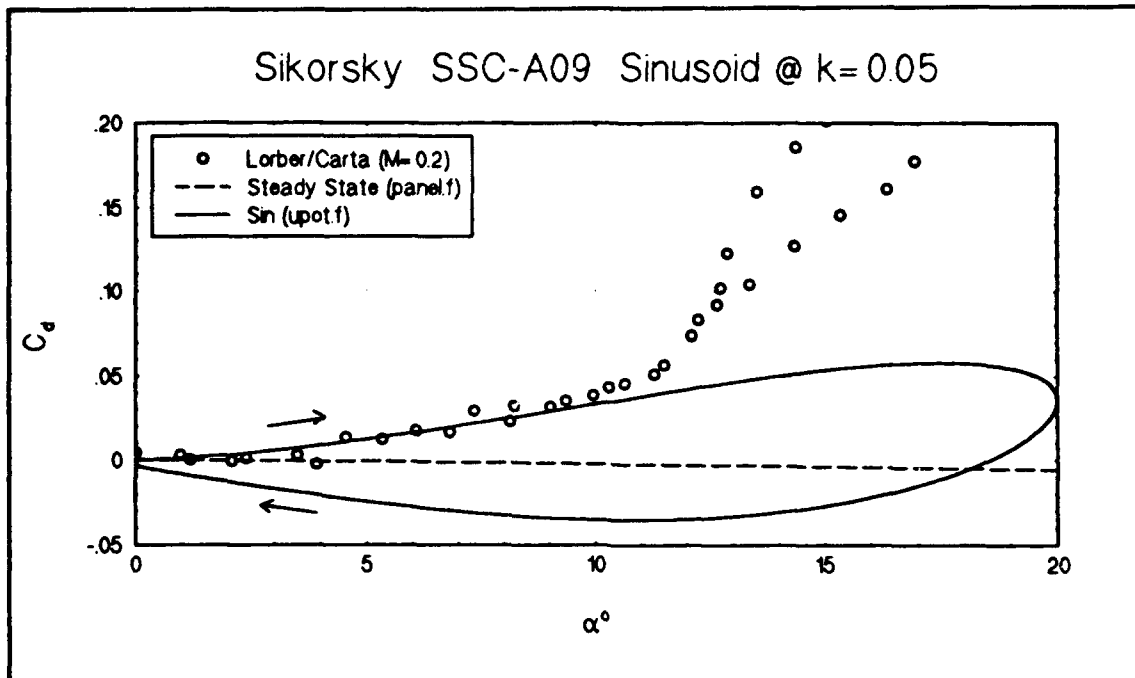


Figure 4.17

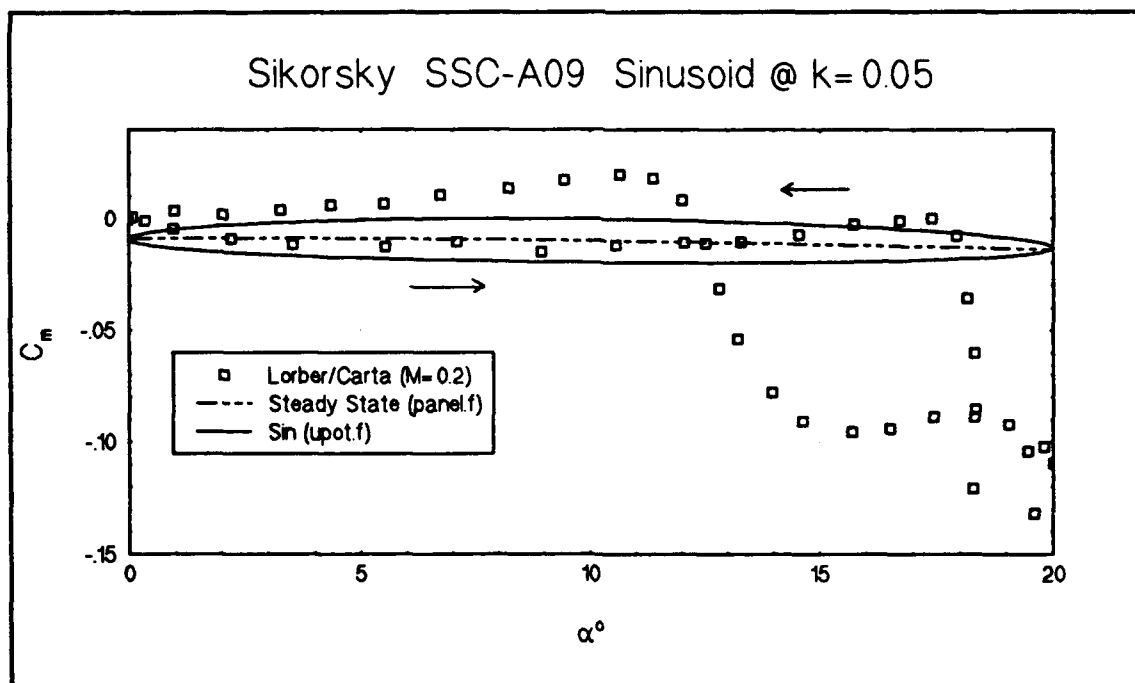


Figure 4.18

V. NAVIER-STOKES CODE

A. THEORY/BACKGROUND

To fully understand and visualize the viscous and compressibility effects on the dynamic stall phenomenon, a numerical solution of the unsteady Navier-Stokes equations is required. A thorough understanding of the flow physics at the leading edge region is most important due to the presence of significant compressibility effects and boundary layer transition. Compressibility effects appear at 0.2 to 0.3 Mach numbers on the NACA 0012; and shocks can form on the airfoil upper surface at 0.45 Mach number. Interaction of the shock and the local boundary layer then directly affects the flow separation process (VanDyken and Chandrasekhara [Ref. 17]).

Dynamic stall is the fluid aerodynamic response to an airfoil executing a time-dependent pitch. Rapid pitch up generates a vortex near the leading edge that increases flow circulation and, therefore, lift. At angles-of-attack beyond the static stall angle-of-attack, massive unsteady separation and large-scale vortical structures characterize the unsteady flowfield (Srinivasan, Ekaterinaris, and McCroskey [Ref. 14]). Vortex convection aft along the airfoil creates large force and moment changes. Successive weaker vortices may be generated with continued pitching or oscillatory motion. The

flow completely reattaches only after the angle of incidence is significantly reduced. Experimental results of unsteady flows over pitching airfoils by Lorber and Carta [Ref. 11] produced supersonic speeds and generated shocks near the leading edge at $M=0.3$ and higher free stream Mach numbers. The dynamic stall flowfield dependence parameters are: airfoil shape, Mach number, reduced frequency or reduced pitch rate, oscillation amplitude motion type (ramp or sinusoid), Reynolds number, and wind tunnel wall effects (Srinivasan, Ekaterinaris, and McCroskey [Ref. 14]). Wind tunnel wall effects were not included in this investigation.

B. NUMERICAL SCHEME

The numerical scheme and implementation used in this paper were taken from one developed by Tuncer, Ekaterinaris, and Platzer [Ref. 16] and Cricelli, Ekaterinaris, and Platzer [Ref. 6]. The strong, conservation law form of the two-dimensional, thin-layer Navier-Stokes equations is used. For a curvilinear coordinate system, (ξ, ζ) , along streamwise and normal direction respectively, the governing equations take the following form:

$$\partial_t \hat{Q} + \partial_\xi \hat{F} + \partial_\zeta \hat{G} = R_\theta^{-1} \partial_\zeta \hat{S} \quad (5.1)$$

Here $\hat{Q}$, Equation 5.2, is the vector of the conservative variables. The inviscid flux vectors, $\hat{F}$ and $\hat{G}$, are shown in

Equation 5.3. U and W are contravariant velocity components given by Equation 5.4. For the thin-layer approximation of the viscous flux term $\mathbf{s}$, in the ζ direction normal to the airfoil surface is shown in Equation 5.5.

$$\hat{\mathbf{Q}} = \frac{1}{J} \begin{Bmatrix} \rho \\ \rho U \\ \rho W \\ e \end{Bmatrix} \quad (5.2)$$

$$\hat{\mathbf{F}} = \frac{1}{J} \begin{Bmatrix} \rho U \\ \rho uU + \xi_x p \\ \rho wU + \xi_z p \\ (e + p)U - \xi_t p \end{Bmatrix} \quad (5.3)$$

$$\hat{\mathbf{G}} = \frac{1}{J} \begin{Bmatrix} \rho W \\ \rho uW + \zeta_x p \\ \rho wW + \zeta_z p \\ (e + p)W - \zeta_t p \end{Bmatrix}$$

$$\begin{aligned} U &= u\xi_x + w\xi_z + \xi_t \\ W &= u\zeta_x + w\zeta_z + \zeta_t \end{aligned} \quad (5.4)$$

$$\hat{\mathbf{g}} = \frac{1}{J} \begin{Bmatrix} 0 \\ \mu m_1 u_\zeta + \left(\frac{\mu}{3}\right) m_2 \zeta_x \\ \mu m_1 w_\zeta + \left(\frac{\mu}{3}\right) m_2 \zeta_z \\ \mu m_1 m_3 + \left(\frac{\mu}{3}\right) m_2 + (\zeta_x u + \zeta_z w) \end{Bmatrix} \quad (5.5)$$

$$m_1 = \zeta_x^2 + \zeta_z^2$$

$$m_2 = \zeta_x u_\zeta + \zeta_z w_\zeta$$

$$m_3 = \frac{u^2 + w^2}{2} + \kappa P_r^{-1} \left(\frac{\partial a^2}{\partial \zeta} \right)$$

In Equations 5.1 through 5.5, all geometrical dimensions are normalized by the root-chord length; density is normalized by the free-stream density, ρ_∞ ; u and w are the Cartesian velocity components of the physical domain normalized by the free-stream speed of sound, a_∞ ; e is the total energy per unit volume normalized by $\rho_\infty a_\infty^2$; and P_r is the Prandtl number. The equation of state for an ideal gas relates pressure to density and total energy and is presented in Equation 5.6. The flow field is assumed to be fully turbulent (the laminar and transitional boundary layers are neglected) and the Baldwin-Lomax turbulence model, Section III.A.1, is used to evaluate eddy viscosity.

$$p = (\gamma - 1) \left[e - \frac{\rho(u^2 + w^2)}{2} \right] \quad (5.6)$$

1. Boundary Conditions

The computational domain includes the airfoil and the entire viscous flow field. The no-slip boundary condition is applied on the airfoil surface. The density and pressure values are obtained by extrapolation. If the flow is unsteady, the surface fluid velocity is set to the dictated airfoil velocity to satisfy the no-slip boundary condition (Tuncer et al, [Ref. 16]).

Flow variables are evaluated using the zero-order Riemann invariant extrapolation at the downstream outflow boundary. Only pressure, the incoming characteristic, is specified at a subsonic outflow boundary and three outgoing characteristics are extrapolated from the interior. A first order extrapolation is used for density and normal velocity. The zero-order outgoing Riemann invariant determines the axial flow velocity. The outflow boundary conditions are shown in Equation 5.7.

$$\begin{aligned} \rho_1 &= \rho_2 \\ u_1 &= R_1^* - \frac{2a_1}{\gamma - 1}, & a_1 &= \sqrt{\frac{\gamma p_1}{\rho_1}} \\ w_1 &= w_2 \\ p_1 &= p_\infty \end{aligned} \quad (5.7)$$

2. Numerical Implementation

The numerical integration is obtained using a high-order accurate upwind biased, factorized, iterative, implicit scheme developed by Tuncer et al [Ref. 16]. The inviscid fluxes are evaluated using Osher's third-order-accurate upwind scheme. Time accuracy of the implicit numerical solution is improved by Newton subiteration within each time step. Complete code details are located in Tuncer et al [Ref. 16] and Cricelli et al [Ref. 5].

a. NPS Cray Y-MP

All NS.F program runs were accomplished using the Naval Postgraduate School Cray Y-MP EL computer. It is a 133 MFLOP processor with 2GB of main memory and a 50 GB local disk running Unicos 7.

b. PLOT3D

The solutions generated by any CFD program consist of millions of numbers representing grid points and physical variable magnitudes. Visual techniques are relied upon to translate this vast numerical data base into comprehensible graphic representations. PLOT3D is a computer graphics program that allows interactive fluid dynamics examination. Physical phenomena are represented by color gradations and through individual particle traces. Program PLCON.F, Appendix C, is used to write the visualization output files that are displayed here.

3. Computational Grid

A 213 by 61 point body-fitted, C-type computational grid was used in all computations. The grid had 213 points around the airfoil (the trailing edge lower point at 31 and upper point at 183) and 61 points in the normal direction, all generated by a hyperbolic grid generator. The grid was clustered at the body surface in the normal direction, at the leading edge and at the trailing edge regions. The C-type grid provided a sufficiently high enough grid density at the airfoil surface boundary to resolve wall viscous and vortical flow field effects, as well as capturing the leading edge shock created during unsteady maneuvers and created at high angle-of-attack in steady flows at greater than 0.3 Mach. Cell orthogonality was emphasized throughout the grid and facilitated solution convergence.

4. Program NS.F User's Guide

A complete NS.F user's guide is located in Appendix C. Included are the NS.IN input name list; Indigo and Cray Y-MP Batch and graphical interface codes; and the NS.F source code.

C. SIKORSKY SSC-A09 RESULTS

1. Steady State Motion

a. Force and Pitching-Moment Coefficients

Force and pitching-moment coefficient steady state results as a function of angle-of-attack for the Navier-Stokes, Panel code, and Lorber and Carta data for 0.2 and 0.4

Mach number flows are displayed in Figures 5.1 through 5.5. The Navier-Stokes code more closely approximates the lift coefficient experimental data through 14° angle-of-attack than the Panel code with accuracy decreasing with increasing Mach number. Some improvement is observed in moment coefficient calculations over the Panel code through 13° angle-of-attack. The Navier-Stokes code did calculate the experimentally measured rapid change in pitching-moment coefficient beyond 15° angle-of-attack with accuracy improving with increasing Mach number. The Navier-Stokes code closely followed experimentally measured drag results through 6° angle-of-attack, and accuracy improved with increasing Mach number beyond this point. The qualitative change in drag with increasing Mach number was properly calculated by the Navier-Stokes code.

b. Skin Friction Coefficient

Skin friction coefficient as a function of airfoil position, angle-of-attack, and Mach number are displayed in Figures 5.6 and 5.7. A 'loads' subroutine error in the NS.F code required modification of the displayed coefficients by a constant factor of 200. A new code version incorporating the Chen-Tysen flow transition model in a new 'bltrans' subroutine appears to have corrected this magnitude error. A separation bubble is observed at 0.2 Mach number located at approximately ten percent chord at both 9° and 11° angle-of-attack. The

trailing edge separation region grows and moves forward with increasing angle-of-attack. Separation bubbles also appear at 0.4 Mach number at 9° , 11° , and 13° angle-of-attack. The trailing edge separation region behaves the same, but is quantitatively smaller in size with increasing Mach number.

c. Pressure Coefficient

Pressure coefficient as a function of airfoil position, angle-of-attack, and Mach number is displayed in Figures 5.8 through 5.19. The Navier-Stokes code closely approximates the experimental data through 13° angle-of-attack with a small discrepancy at the trailing edge at low Mach numbers. Navier-Stokes correlation appears to be better with decreasing Mach number. The Baldwin-Lomax turbulence model was not able to accurately predict the shock induced boundary layer separation and reduced suction peak as can be observed in Figures 5.15, and 5.17 through 5.19.

d. Plot3D Visualization

Steady State density, pressure, Mach number, and vorticity flowfield variations as a function of Mach number and angle-of-attack are shown in Figures 5.20 through 5.31. The solutions presented for 0° angle-of-attack are the ones later used to initiate unsteady motion. At $M=0.2$, the flow remains attached until $9^\circ \alpha$, at which point the trailing edge lightly separates and the separation region begins to move

forward with increasing angle-of-attack. At $15^\circ \alpha$, the separation region extends forward to approximately $0.1 x/c$.

At $M=0.4$, the flow again remains attached until $9^\circ \alpha$, at which point the trailing edge separates a little more and the separation region begins to move forward with increasing angle-of-attack. Vorticity appears to increase with increasing Mach number. Shock induced boundary layer separation appears first at $11^\circ \alpha$ and increases in magnitude through $15^\circ \alpha$ where massive flowfield separation is observed, Figure 5.31.

2. Unsteady Motion

The unsteady calculations were started from the steady state solution at the lowest angle-of-attack. For sinusoidal motion, the instantaneous angle-of-attack is shown in Equation 5.8. Definitions for Reduced Pitch Rate, A , and Reduced Frequency, k , are given in Equations 4.3 and 4.4, respectively.

$$\alpha(t) = \frac{(\alpha_{\max} - \alpha_{\min})}{2} \times [1 - \cos(\omega t)] \quad (5.8)$$

a. Ramp Motion, $M=0.2$, $\alpha=0^\circ \sim 30^\circ$, and $A=0.005$

Density residual history as a function of angle-of-attack is displayed in Figure 5.32 and indicates stable

results through 15° angle-of-attack. Force and pitching-moment coefficient results, unsteady panel, and Lorber and Carta experimental results are displayed in Figures 5.33 through 5.35. The Navier-Stokes code more closely calculates lift and pitching-moment coefficients through 17° angle-of-attack than the unsteady panel code. The Navier-Stokes code closely followed experimentally measured drag through 10° angle-of-attack, but was late in calculating the experimentally measured sudden rise in drag at 16° angle-of-attack.

(1) *Flowfield Visualization.* Flowfield density, pressure, Mach number, and vorticity are displayed in Figures 5.36 through 5.42. A smaller leading edge stagnation region and a little larger trailing edge separation region is evident when compared to steady state results. Ramp maximum leading edge Mach number is also slightly smaller at any given angle-of-attack. Massive flowfield separation is observed at 20° angle-of-attack, Figure 5.42.

b. Ramp Motion, $M=0.3$ & 0.4 , $\alpha=0^\circ$ - 20° , and $A=0.005$

Force and pitching-moment coefficient results as a function of Mach number and angle-of-attack and Lorber and Carta experimental results are displayed in Figures 5.43 through 5.48. The Navier-Stokes code somewhat overestimates force and pitching-moment coefficients for all Mach numbers throughout the attached flow region, but did capture the

experimentally measured changes at stall with accuracy improving with increasing Mach number. Drag coefficient followed experiment closely through the linear region with accuracy improving with increasing Mach number.

(1) *Flowfield Visualization.* Flowfield density, pressure, Mach number, and vorticity for the $M=0.4$ ramp are displayed in Figures 5.49 through 5.55. Mach number over the leading edge is higher than steady state values. Shock induced boundary layer separation is observed starting at 11° angle-of-attack and is more massive and moves more forward than observed in steady state results. At 15° angle-of-attack, the upper surface is mostly separated and convecting vortices are observed departing the trailing edge.

The leading edge flowfield region of the $M=0.4$ ramp motion is magnified in Figures 5.56 through 5.63. These figures graphically display the complex leading edge flow physics of shock induced boundary layer separation. A secondary shock on the leading edge can also be identified in Figures 5.56, 5.57, and 5.60. The time dependent vortex shedding process can be observed in Figures 5.62 and 5.63.

c. Sinusoid, $M=0.2$, and $k=0.025$ $[6 - 6\cos(\omega t)]$

Density residual history as a function of angle-of-attack is displayed in Figure 5.64 and indicates stable results throughout the angle-of-attack range. Force and pitching-moment coefficient results are displayed with

unsteady panel and Lorber and Carta results in Figures 5.65 through 5.67. Navier-Stokes lift results more closely follow the experimental results; but the character of the results, decreased lift on the up stroke and augmented lift on the down stroke, are opposite to the expected results. Drag computations improved over the unsteady panel code. The general trend is to follow experiment, but there is an error associated with angle-of-attack. Pitching-moment follows the experiment well, but seems to have a constant error. The experimental pitching-moment results are all positive which would not be expected with this cambered airfoil.

(1) *Flowfield Visualization.* Flowfield density, pressure, Mach number, and vorticity are displayed in Figures 5.68 through 5.74 for both the up and down stroke. Trailing edge separation is first observed at 9° angle-of-attack. The flow generally remains well behaved throughout the cycle and no shocks are indicated. Higher local Mach numbers are observed on the down stroke when compared to the up stroke for the same angle-of-attack.

d. Sinusoid, $M=0.2$, and $k=0.05$ $[10 - 10\cos(\omega t)]$

Density residual history as a function of angle-of-attack is displayed in Figure 5.75 and indicates stable results below 13.5° angle-of-attack. Force and pitching-moment coefficient results are displayed in Figures 5.76 through 5.78. The Navier-Stokes code followed experimentally

measured lift well through 19° angle-of-attack. A large difference in lift was calculated during the down cycle between 20° and 9° angle-of-attack. The Navier-Stokes code followed experimentally measured drag well through 12° angle-of-attack. There appears to be good moment coefficient agreement through 12° angle-of-attack.

(1) *Flowfield Visualization.* Flowfield density, pressure, Mach number, and vorticity are displayed in Figures 5.79 through 5.91 for both the up and down stroke. For angles-of-attack below 13° , local leading edge Mach number is higher and the trailing edge separation region is larger on the down stroke when compared to the up stroke at the same angle-of-attack. Beginning at 15° angle-of-attack, the local leading edge Mach number is higher on the up stroke. Massive trailing edge separation is observed on the down stroke at 17° angle-of-attack and disappears at 15° angle-of-attack. The flowfield is massively separated with complex vortical structures at the maximum angle-of-attack, 20° . The leading edge flowfield region is magnified in Figures 5.92 through 5.94. These figures graphically display the complex leading edge flow physics of shock induced boundary layer separation.

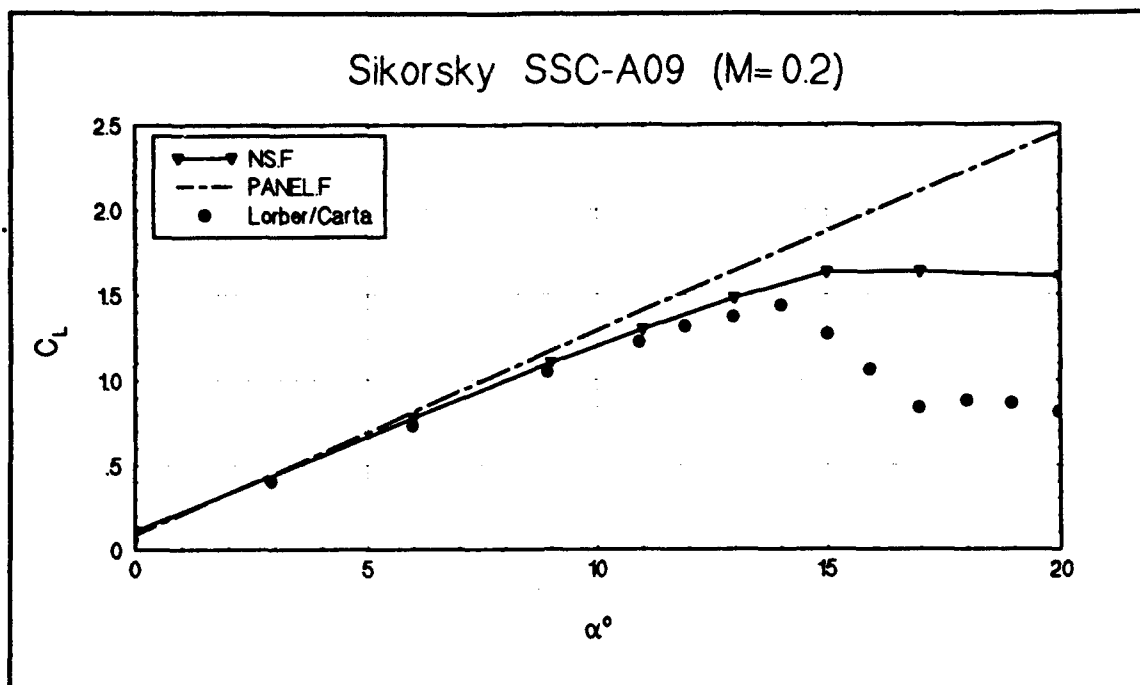


Figure 5.1
Steady State $C_{L\alpha}$ ($M=0.2$)

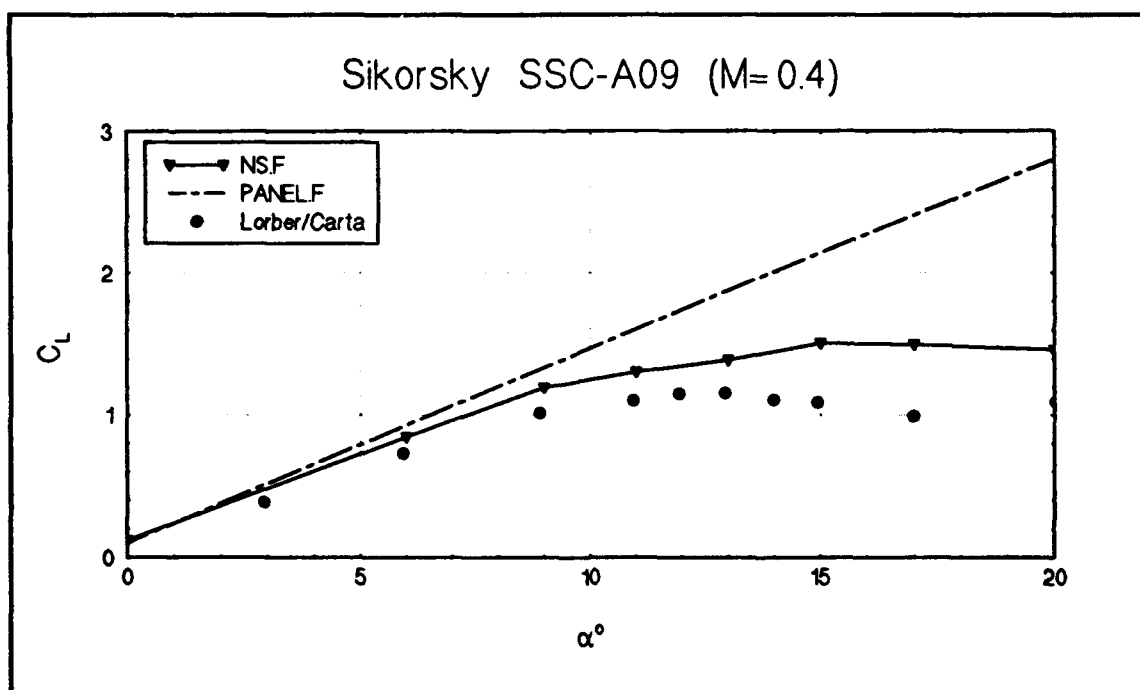


Figure 5.2
Steady State $C_{L\alpha}$ ($M=0.4$)

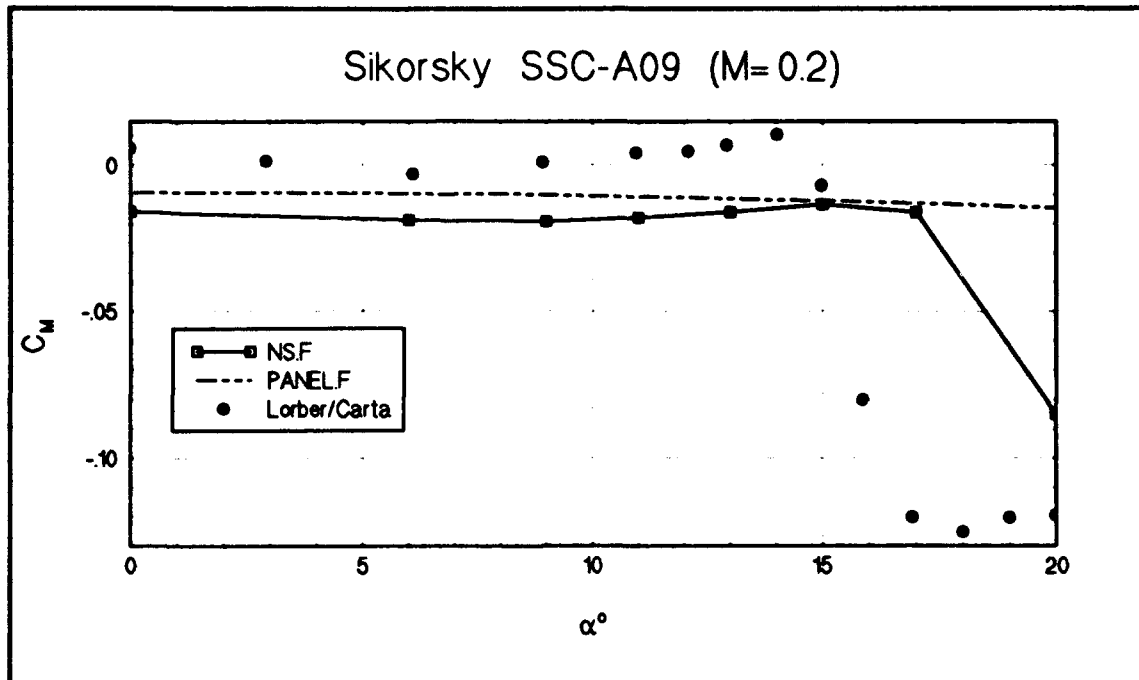


Figure 5.3
Steady State $C_{M\delta}$ ($M=0.2$)

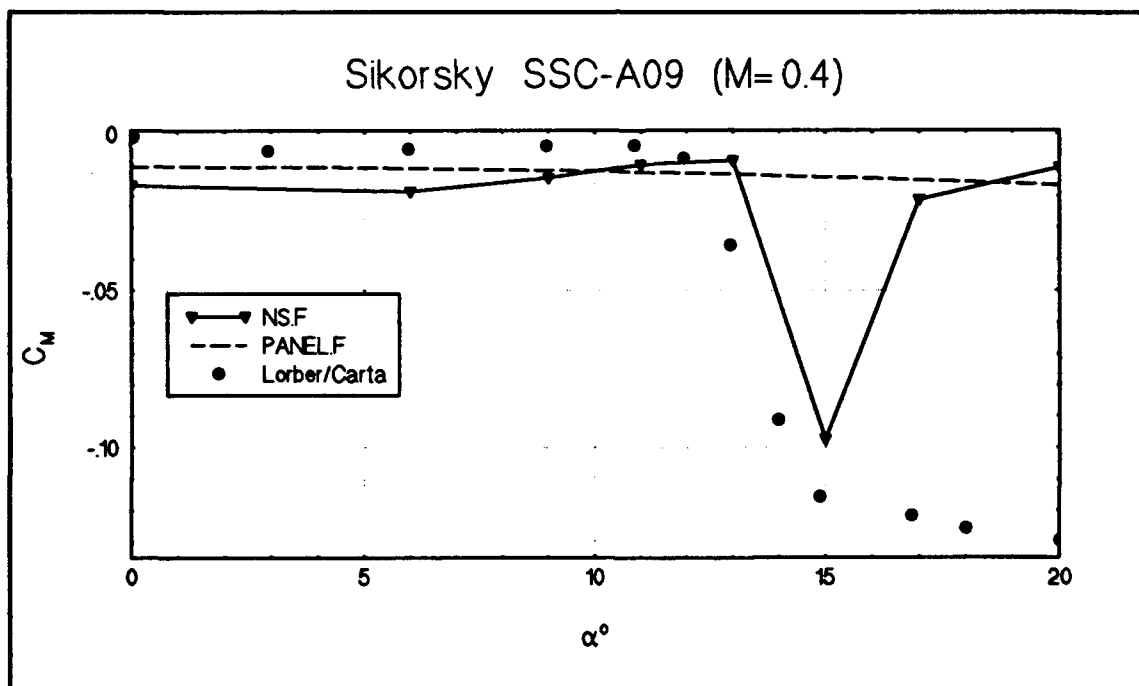


Figure 5.4
Steady State $C_{M\delta}$ ($M=0.4$)

Sikorsky SSC-A09

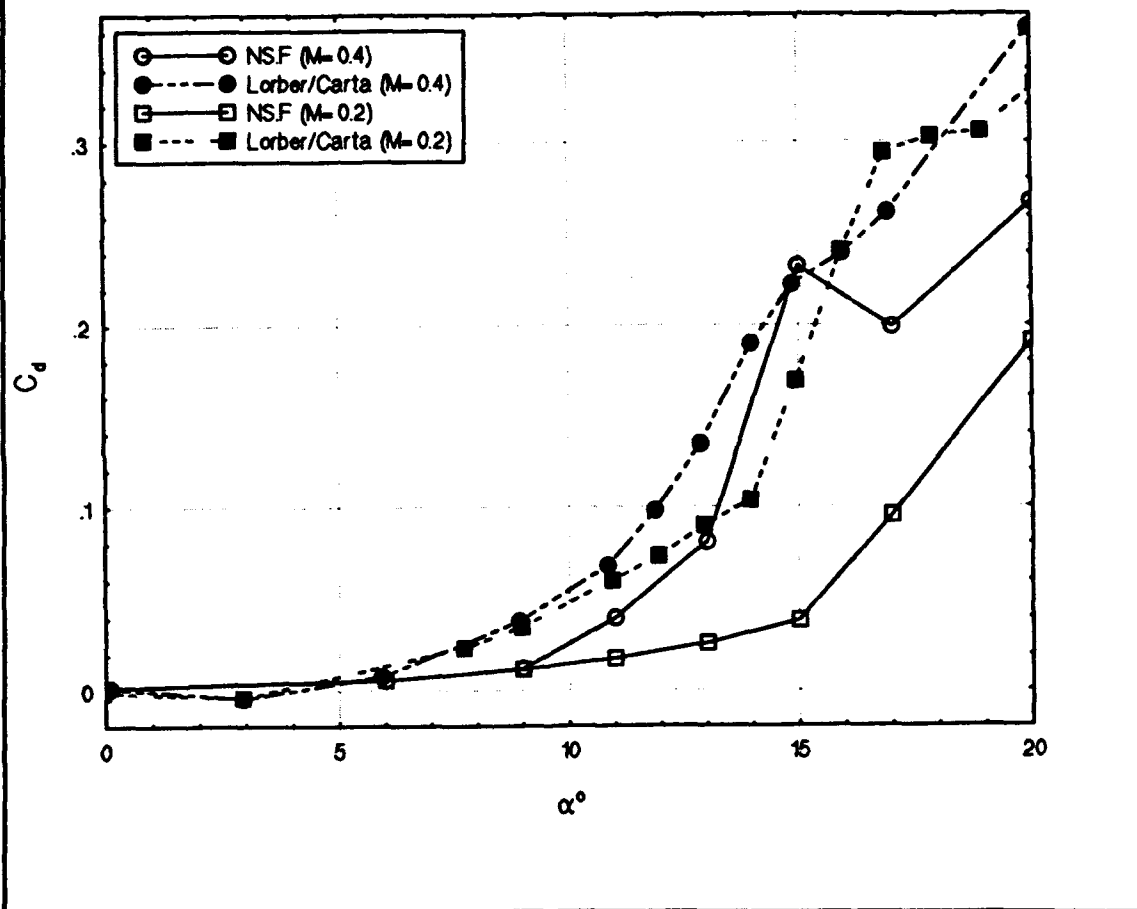


Figure 5.5
Steady State C_{ds}

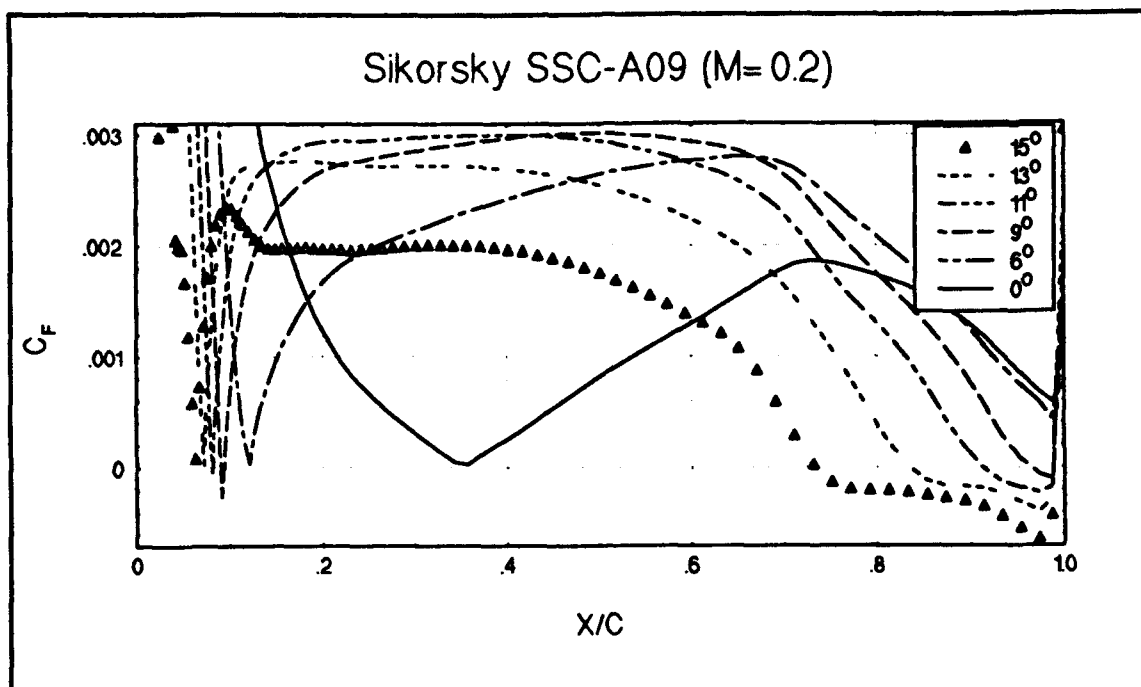


Figure 5.6
Steady State C_f (M=0.2)

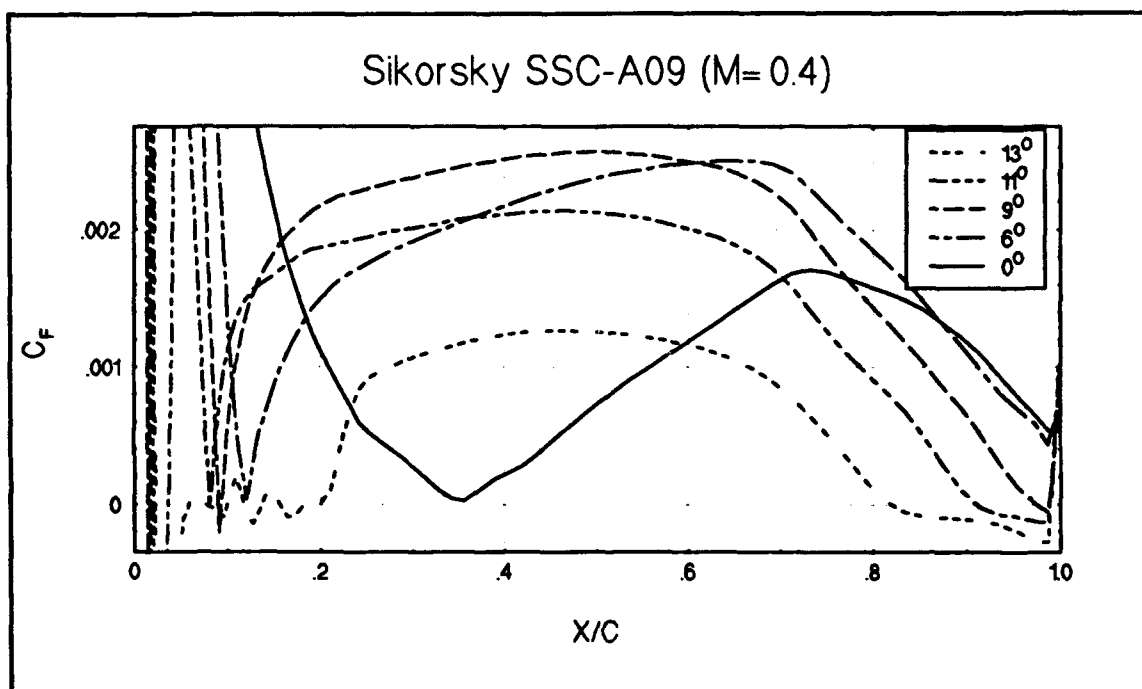


Figure 5.7
Steady State C_f (M=0.4)

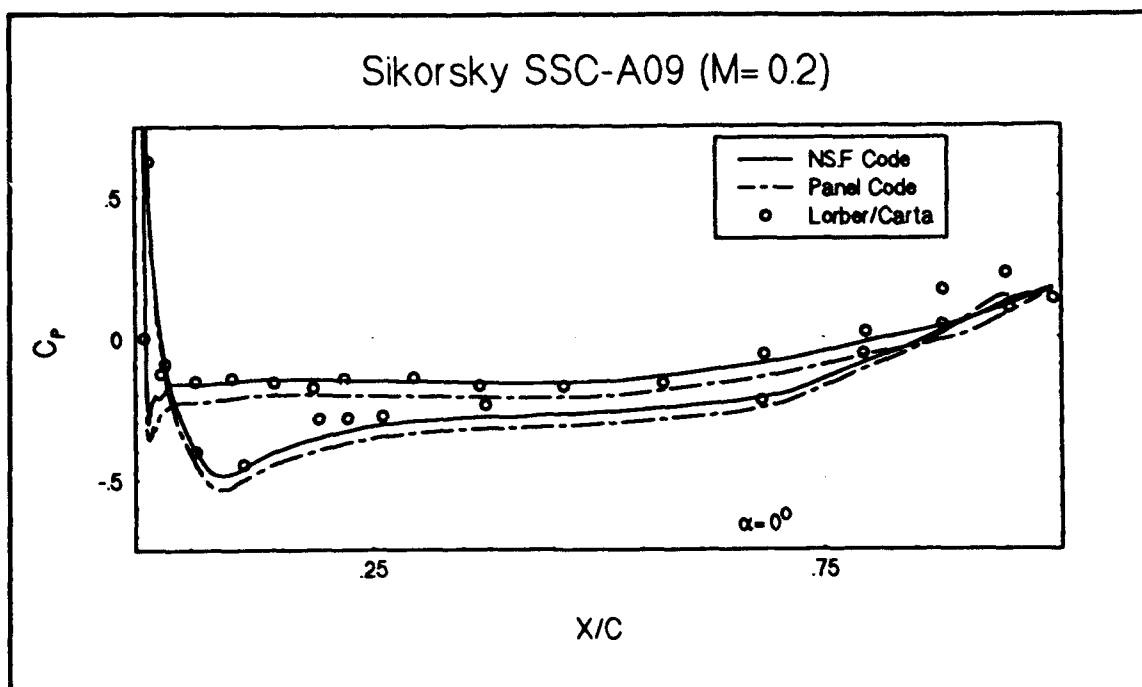


Figure 5.8
Steady State C_p ($M=0.2$)

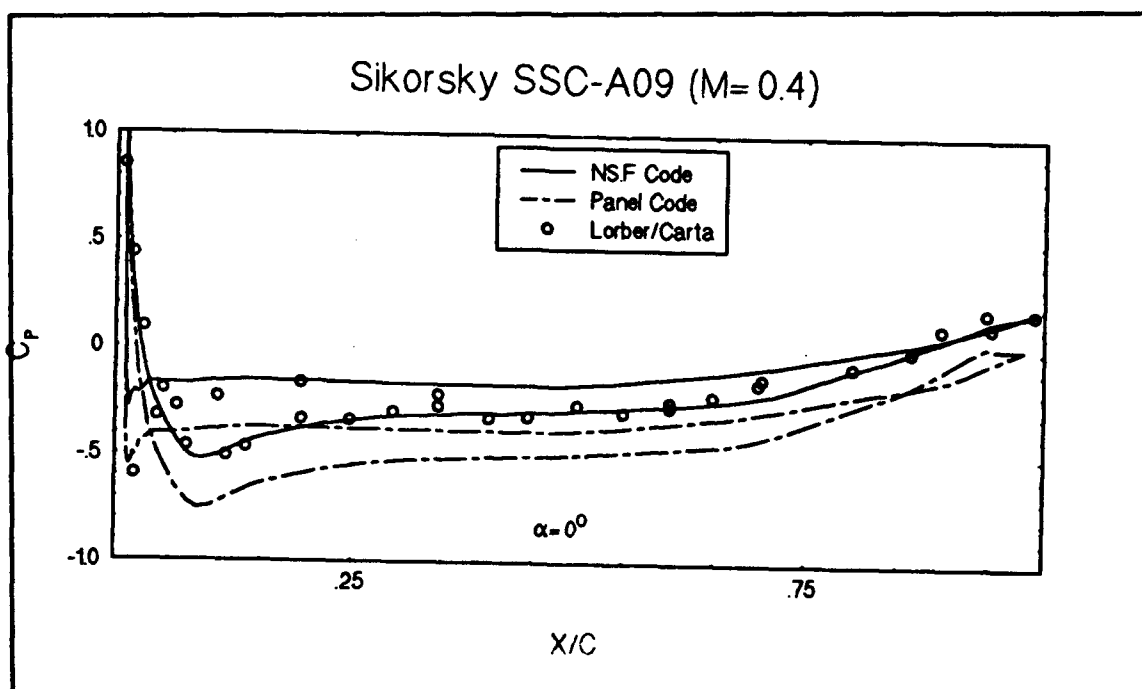


Figure 5.9
Steady State C_p ($M=0.4$)

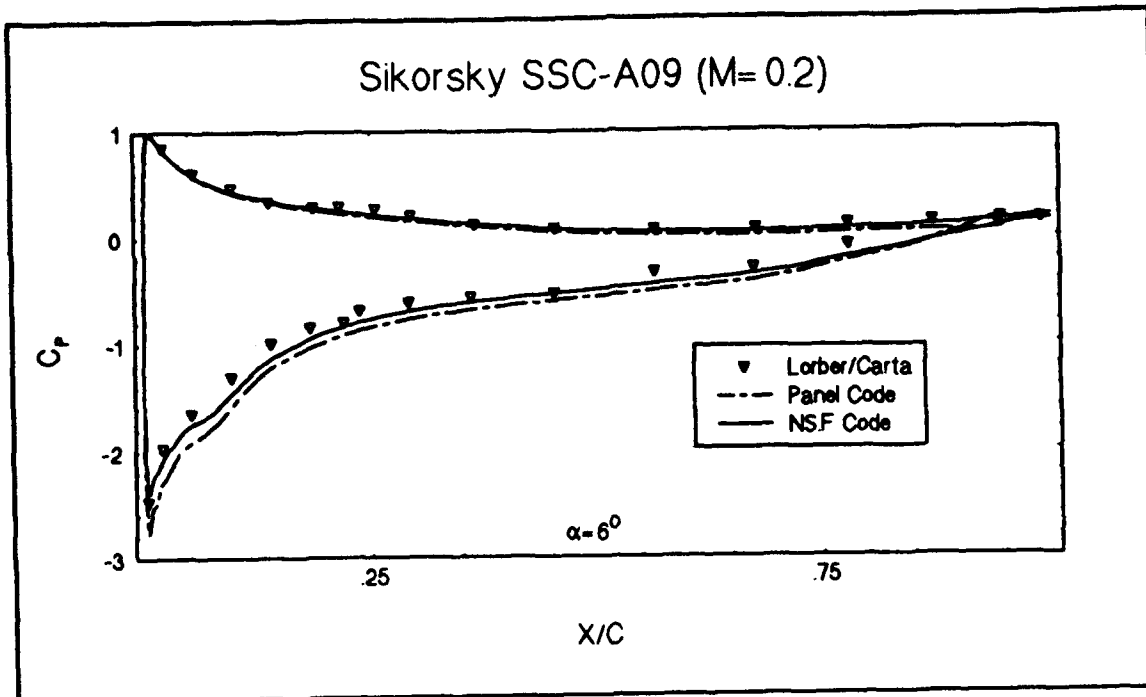


Figure 5.10
Steady State C_p ($M=0.2$)

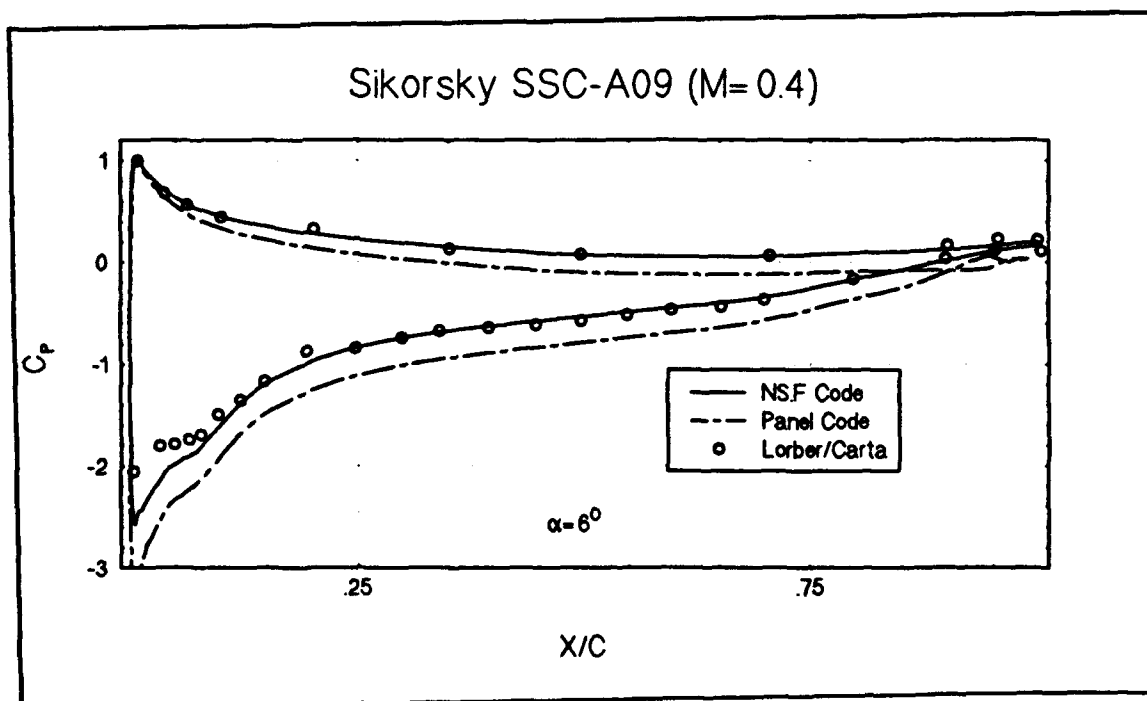


Figure 5.11
Steady State C_p ($M=0.4$)

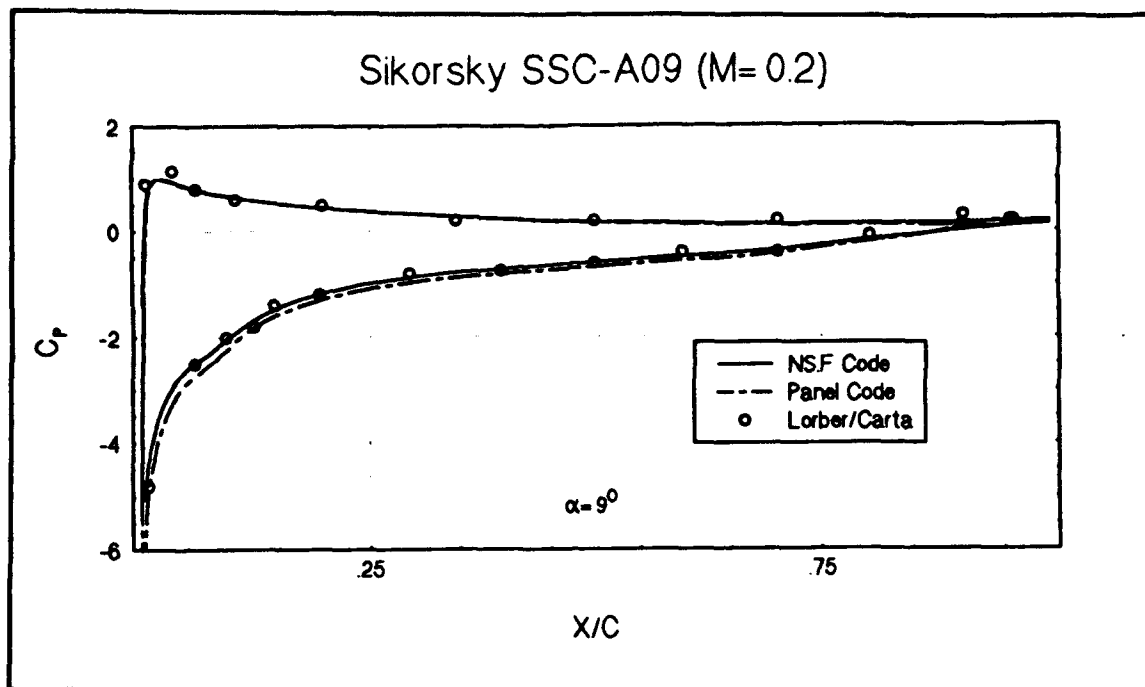


Figure 5.12
Steady State C_p ($M=0.2$)

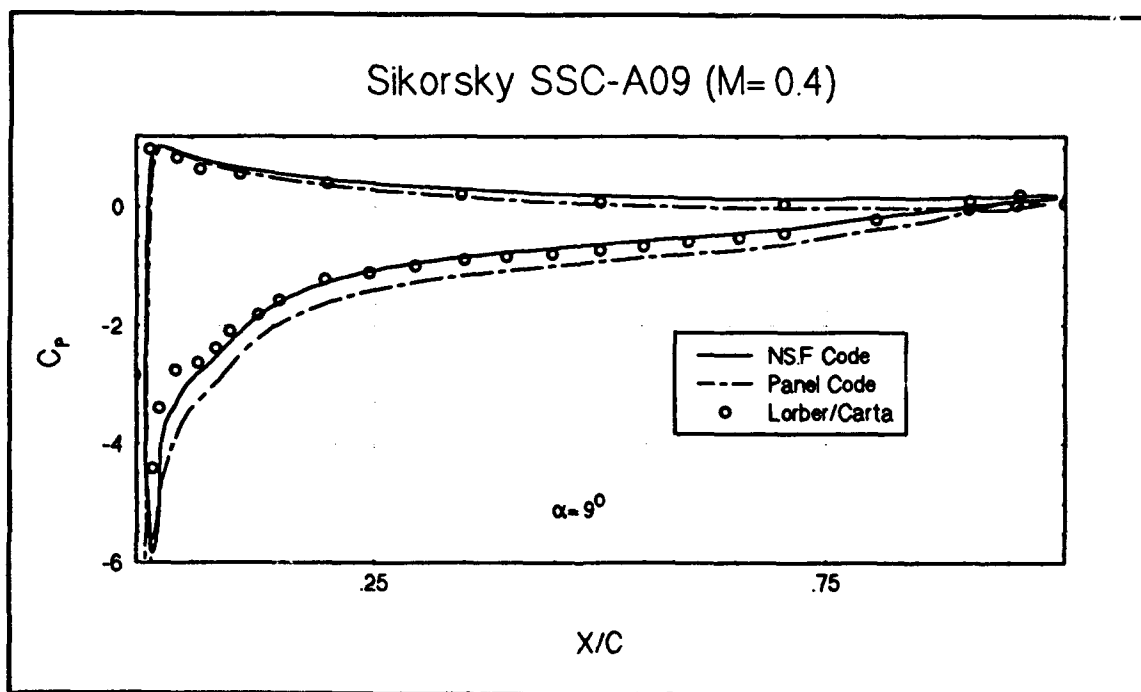


Figure 5.13
Steady State C_p ($M=0.4$)

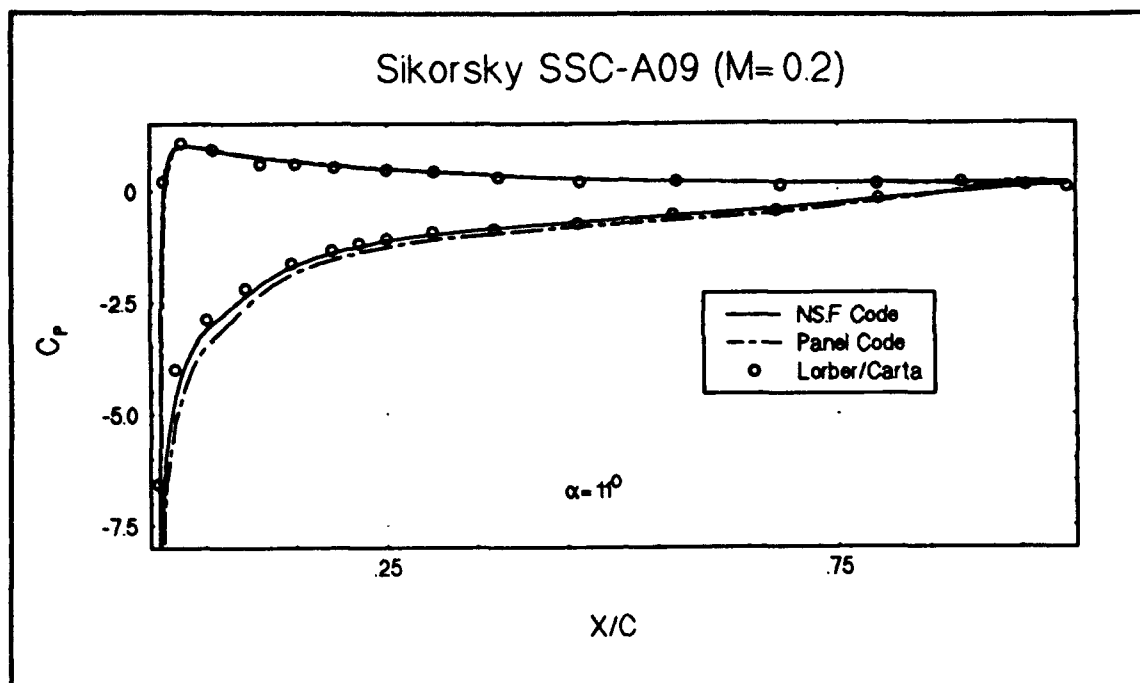


Figure 5.14
Steady State C_p ($M=0.2$)

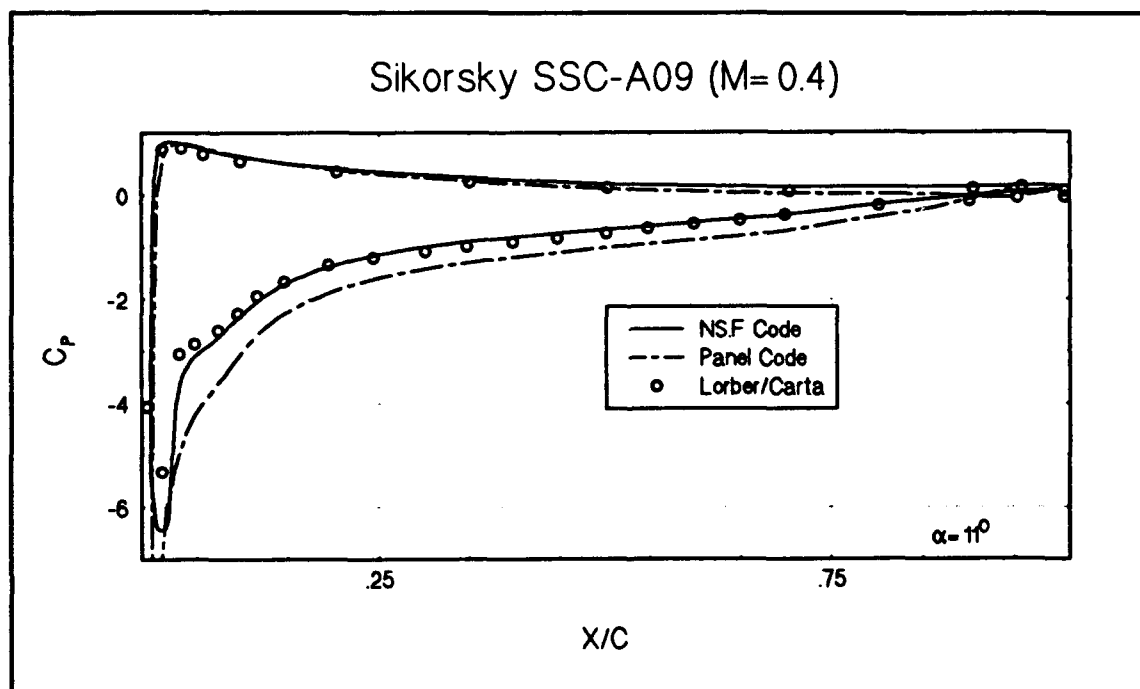


Figure 5.15
Steady State C_p ($M=0.4$)

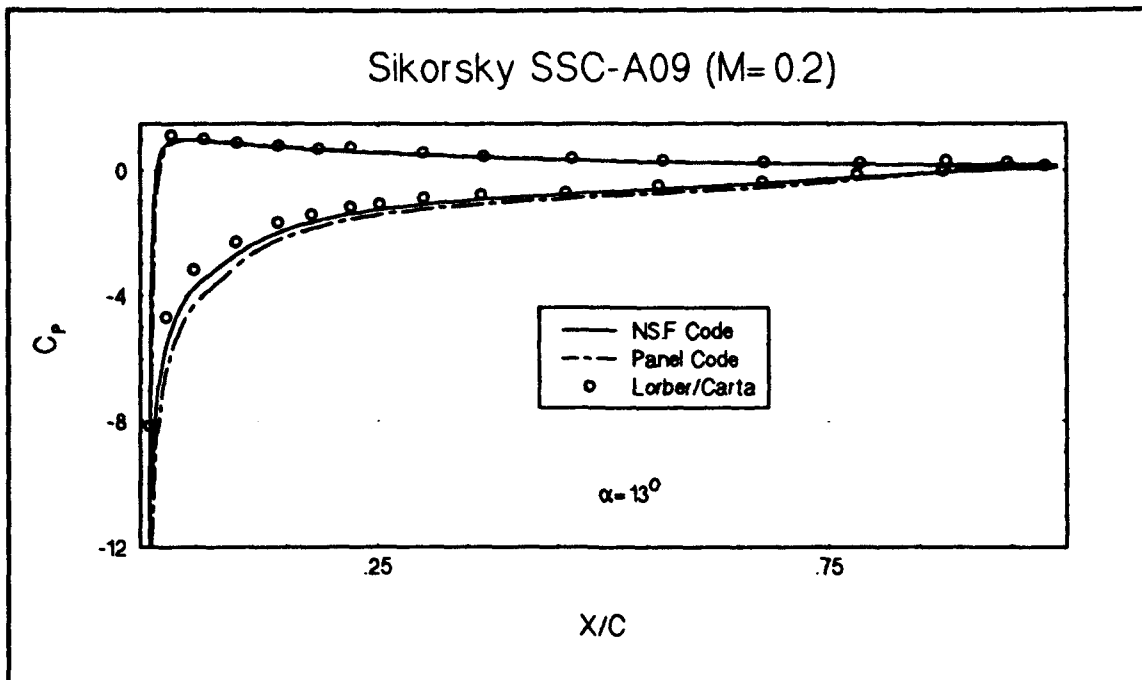


Figure 5.16
Steady State C_p ($M=0.2$)

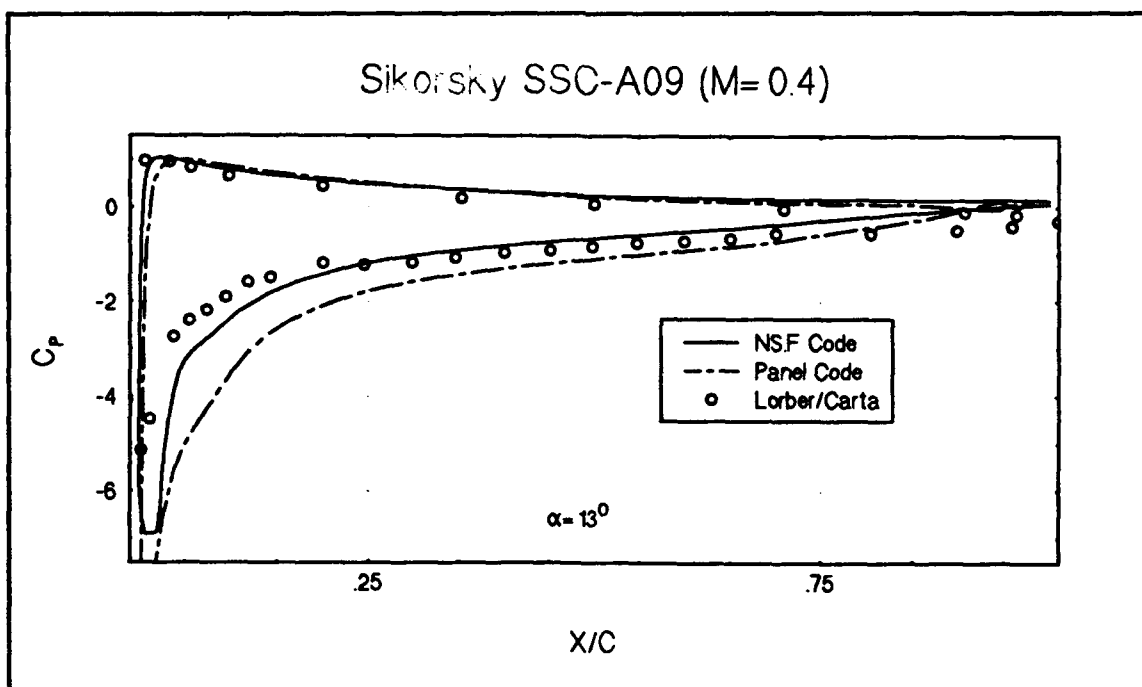


Figure 5.17
Steady State C_p ($M=0.4$)

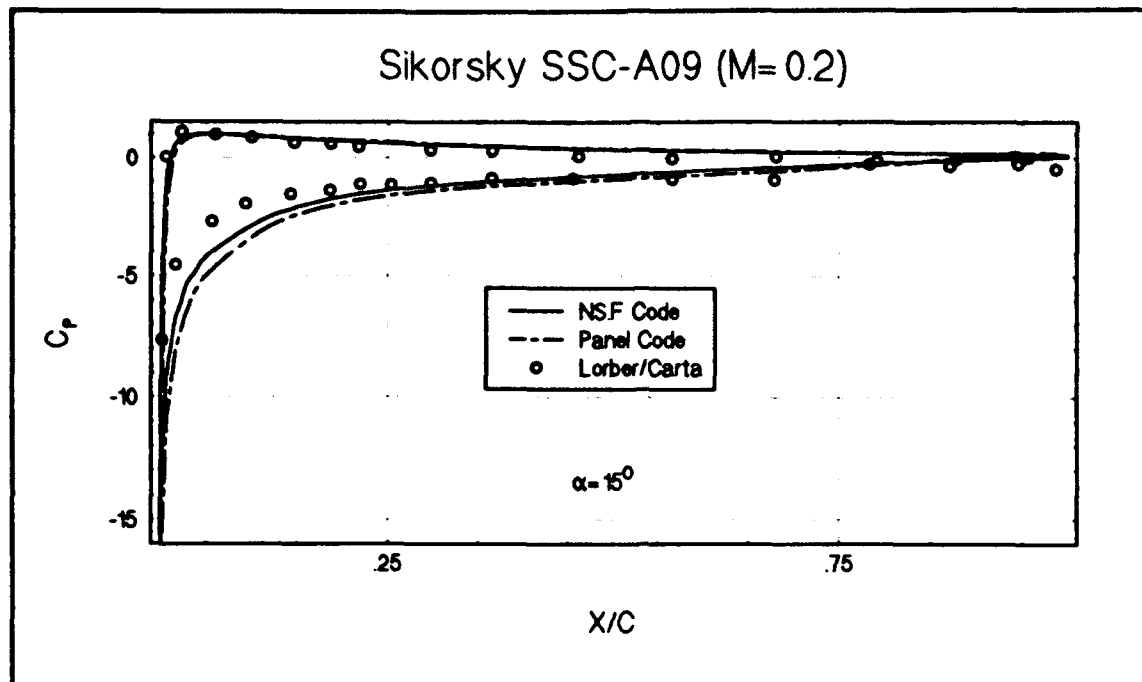


Figure 5.18
Steady State C_p ($M=0.2$)

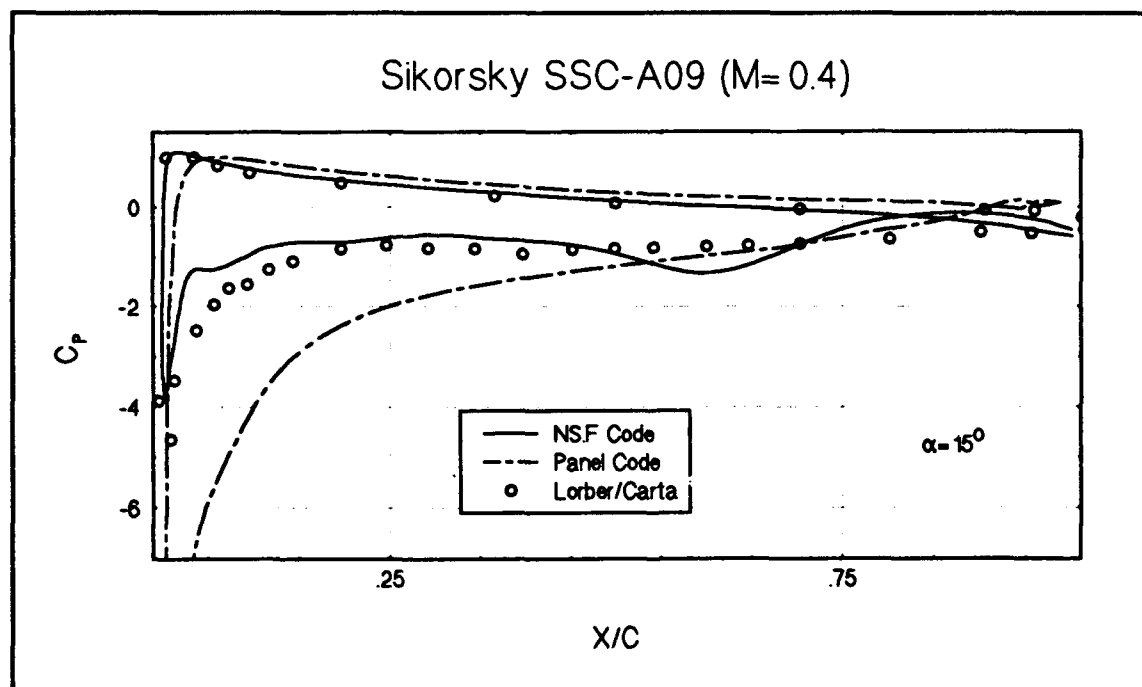


Figure 5.19
Steady State C_p ($M=0.4$)

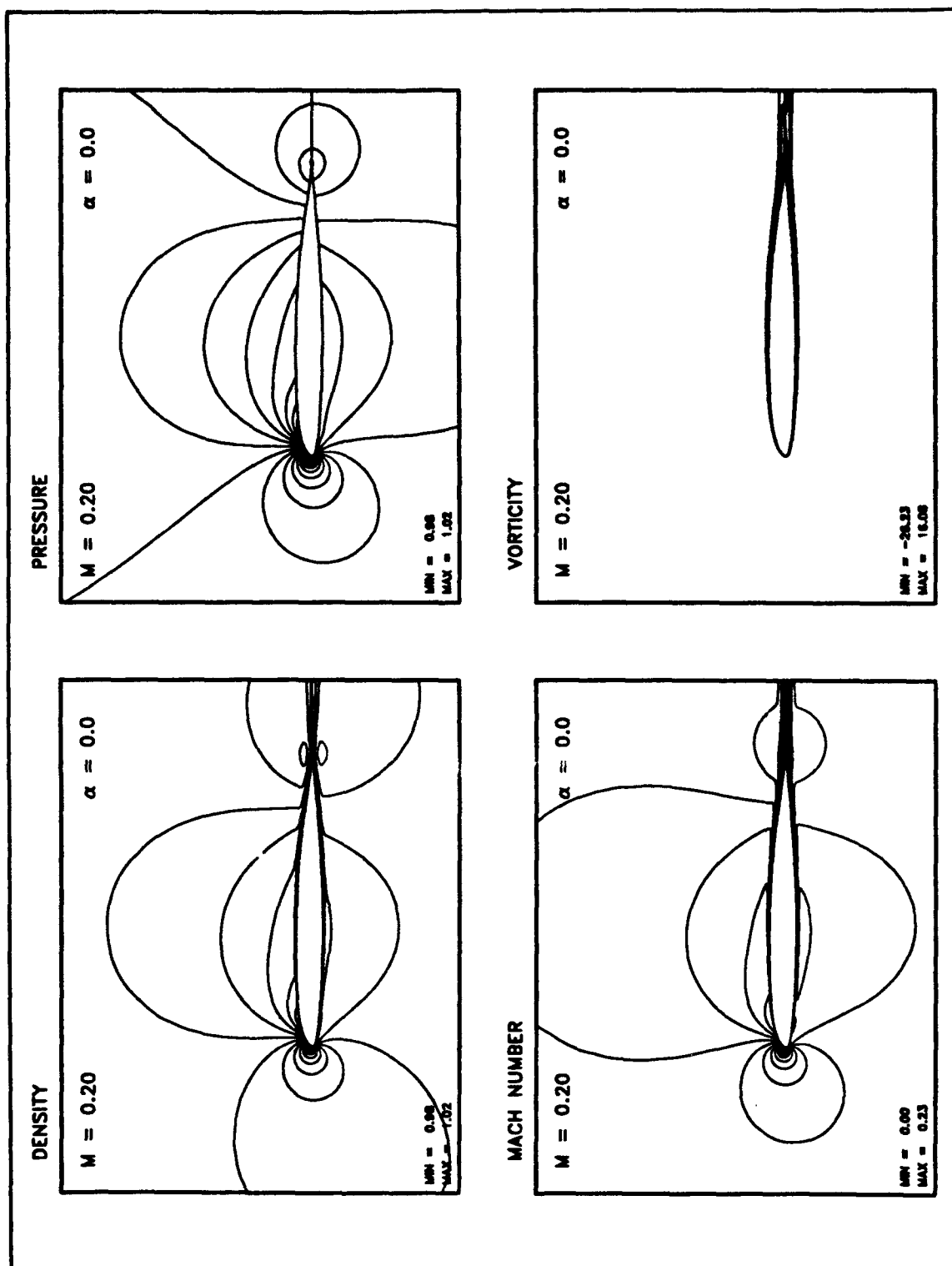


Figure 5.20
Sikorsky SSC-A09
Steady State ($M=0.2$)

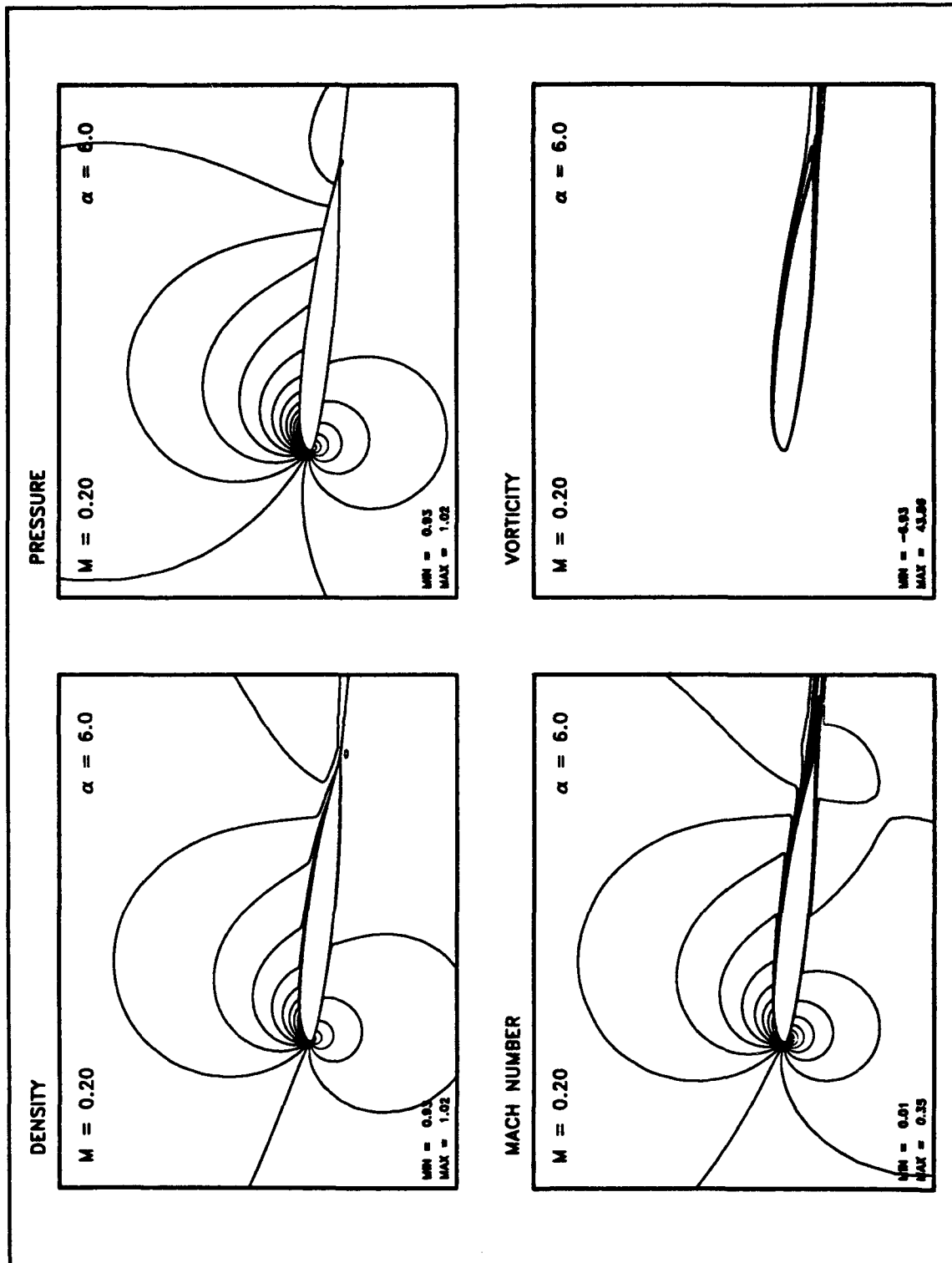


Figure 5.21
Sikorsky SSC-A09
Steady State ($M=0.2$)

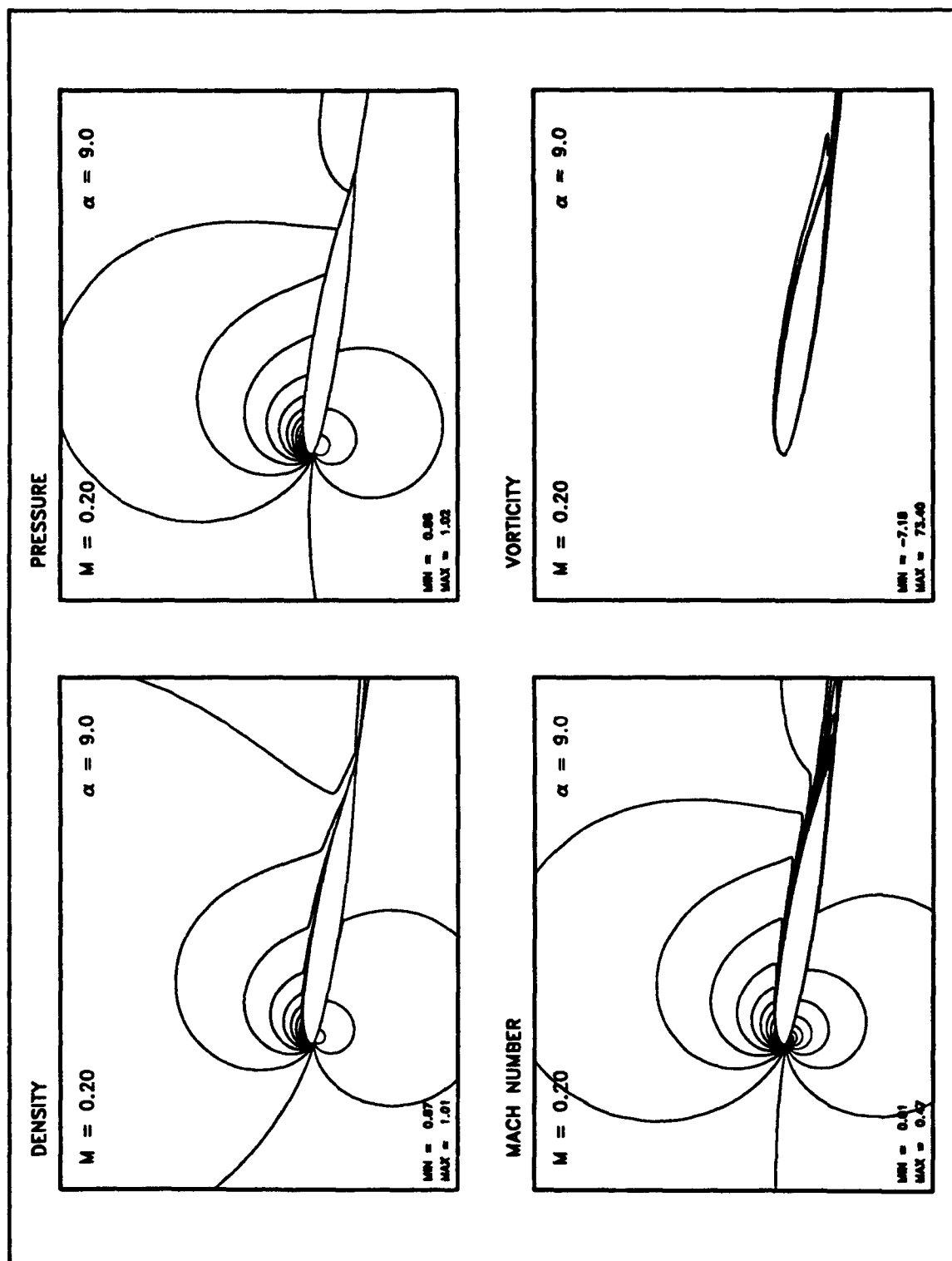


Figure 5.22
Sikorsky SSC-A09
Steady State ($M=0.2$)

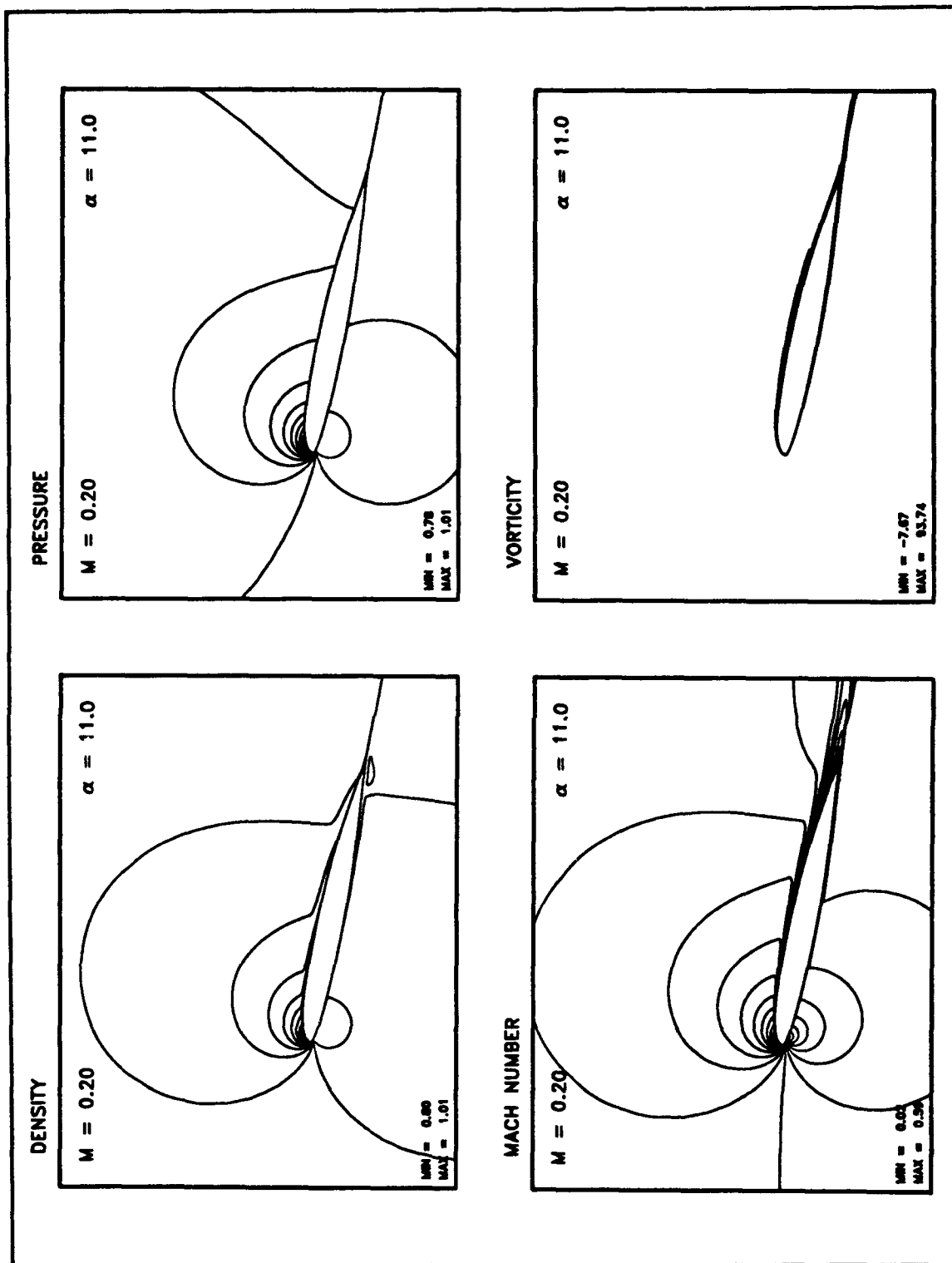


Figure 5.23
Sikorsky SSC-A09
Steady State ($M=0.2$)

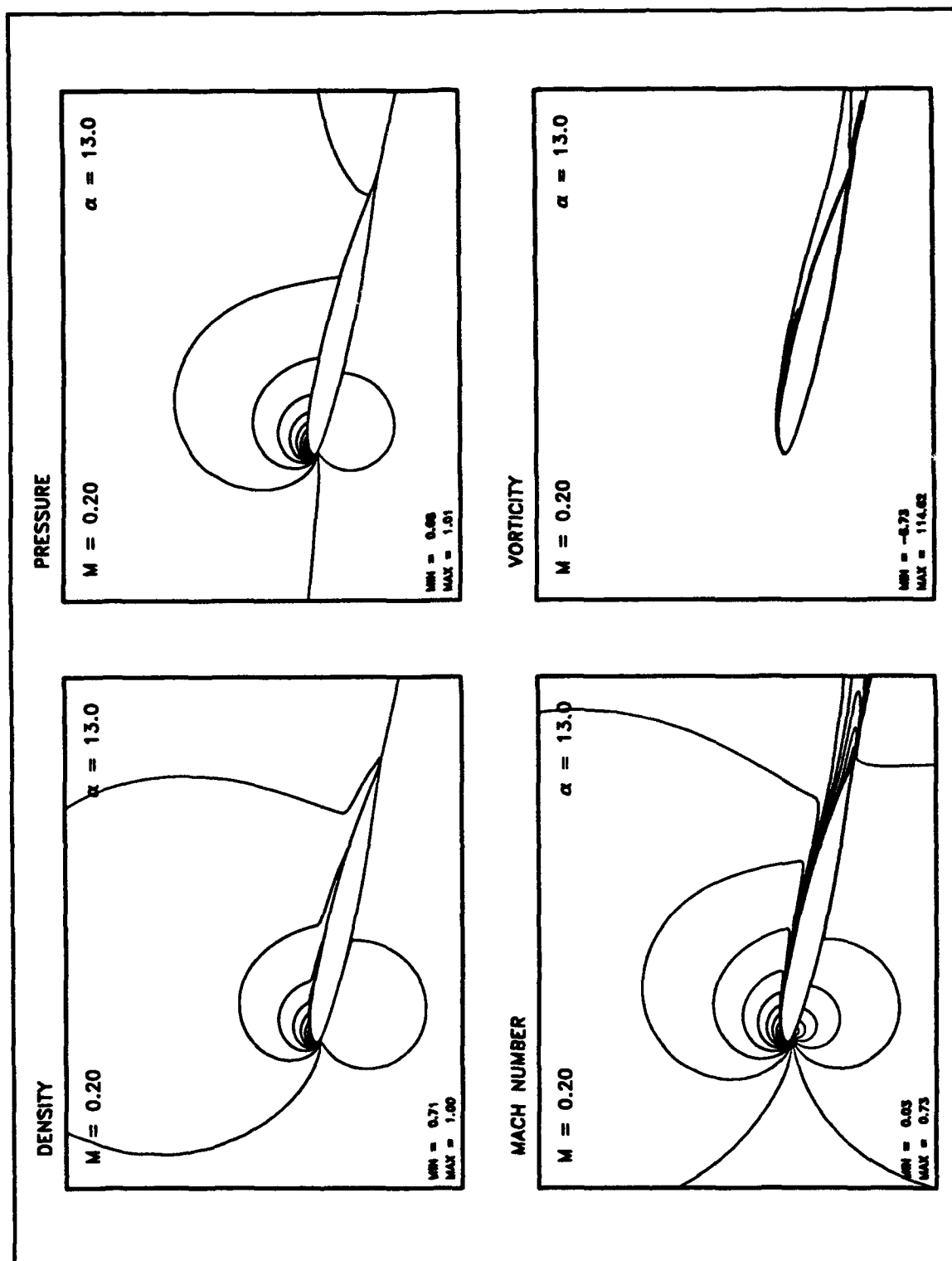


Figure 5.24
Sikorsky SSC-A09
Steady State ($M=0.2$)

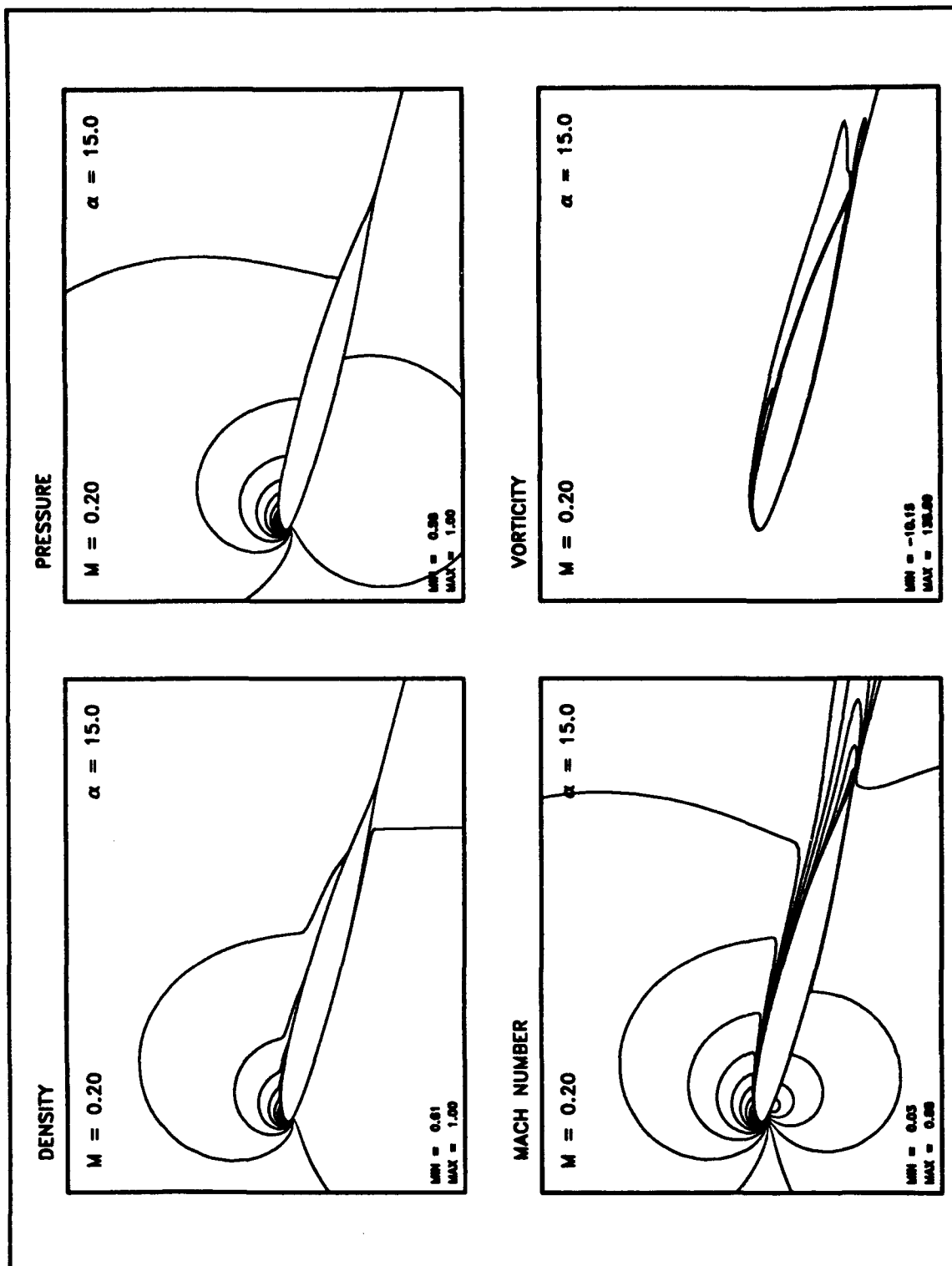


Figure 5.25
Sikorsky SSC-A09
Steady State ($M=0.2$)

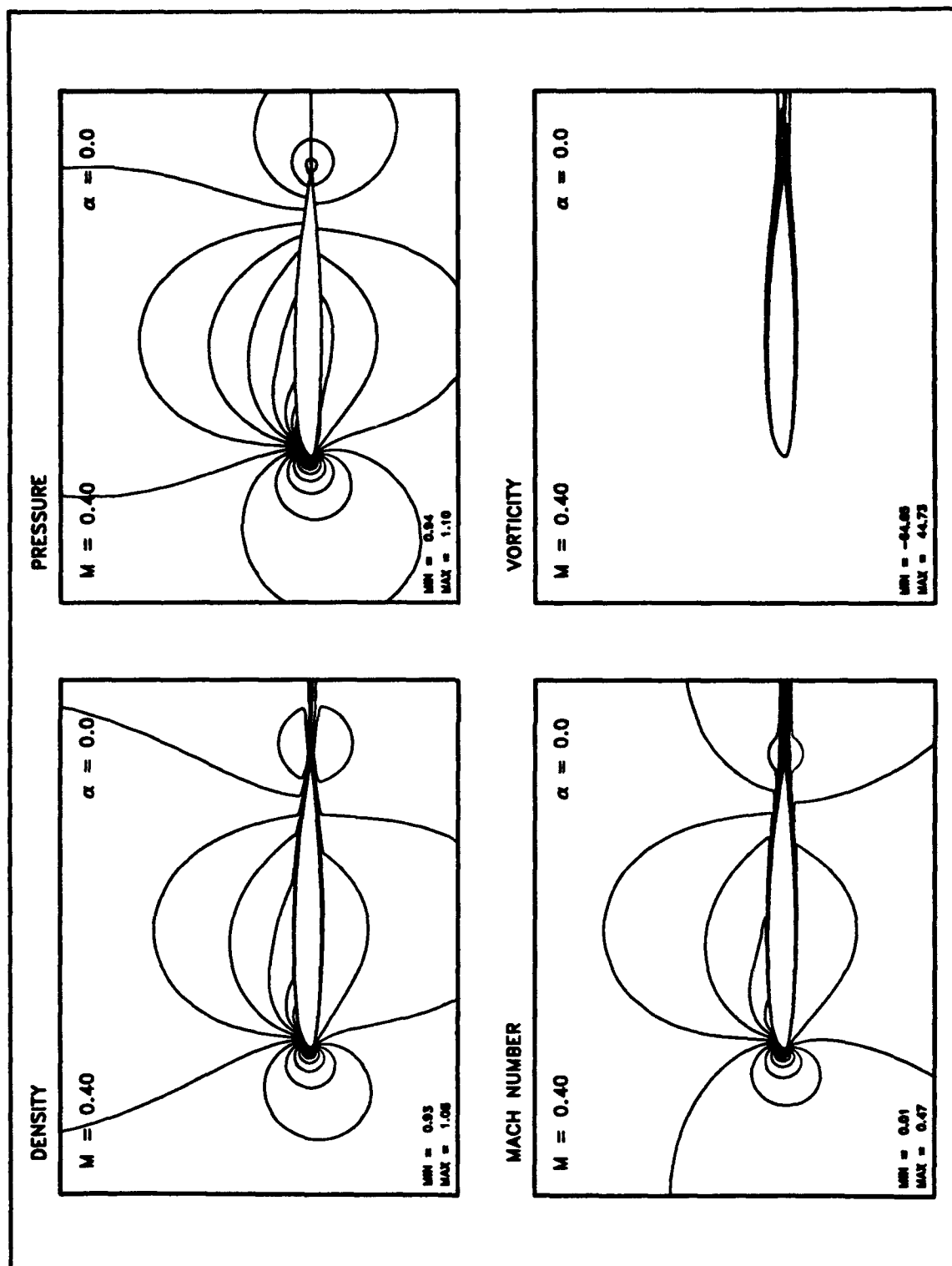


Figure 5.26
Sikorsky SSC-A09
Steady State ($M=0.4$)

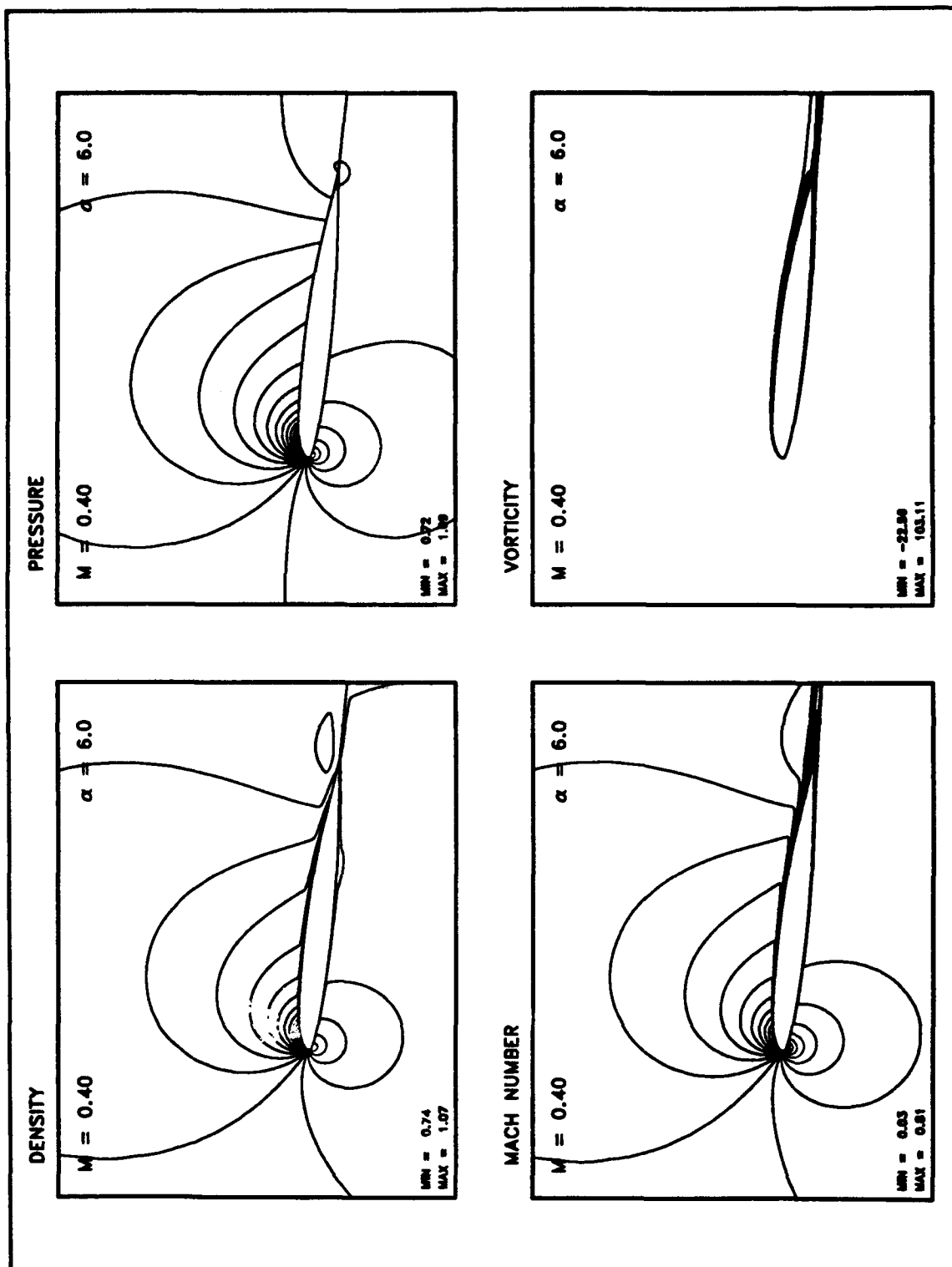


Figure 5.27
Sikorsky SSC-A09
Steady State ($M=0.4$)

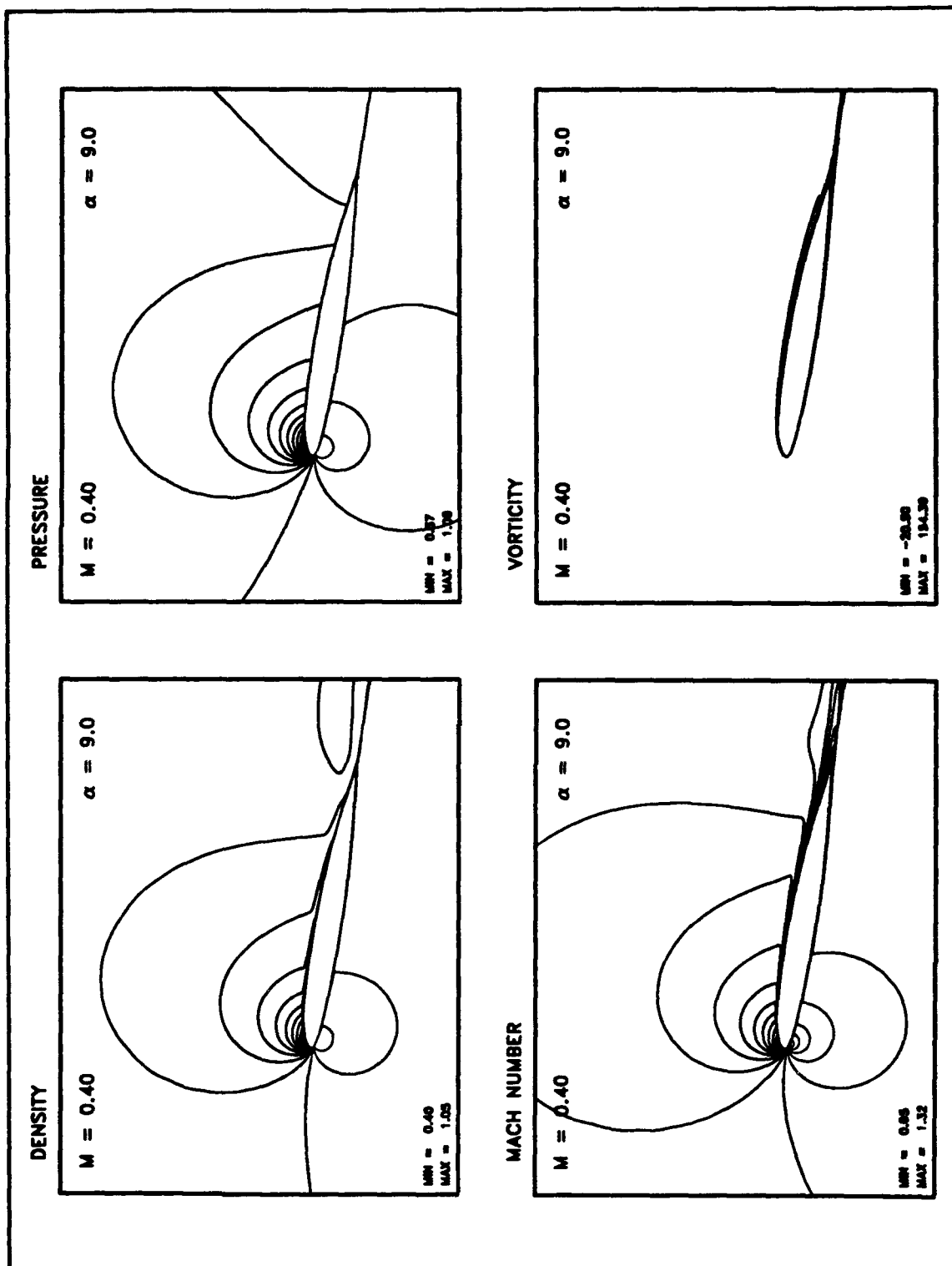


Figure 5.28
Sikorsky SSC-A09
Steady State ($M=0.4$)

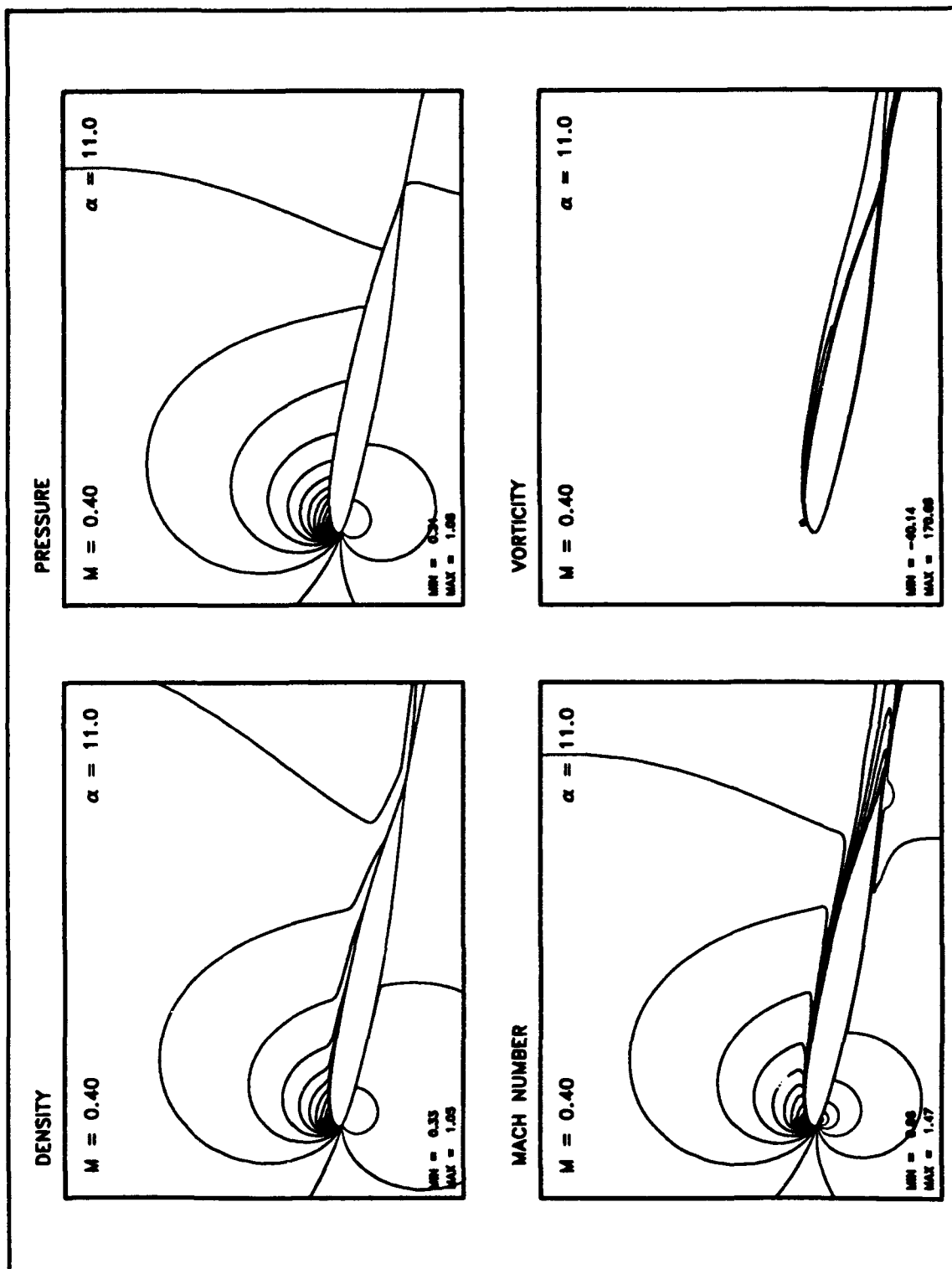


Figure 5.29
Sikorsky SSC-A09
Steady State ($M=0.4$)

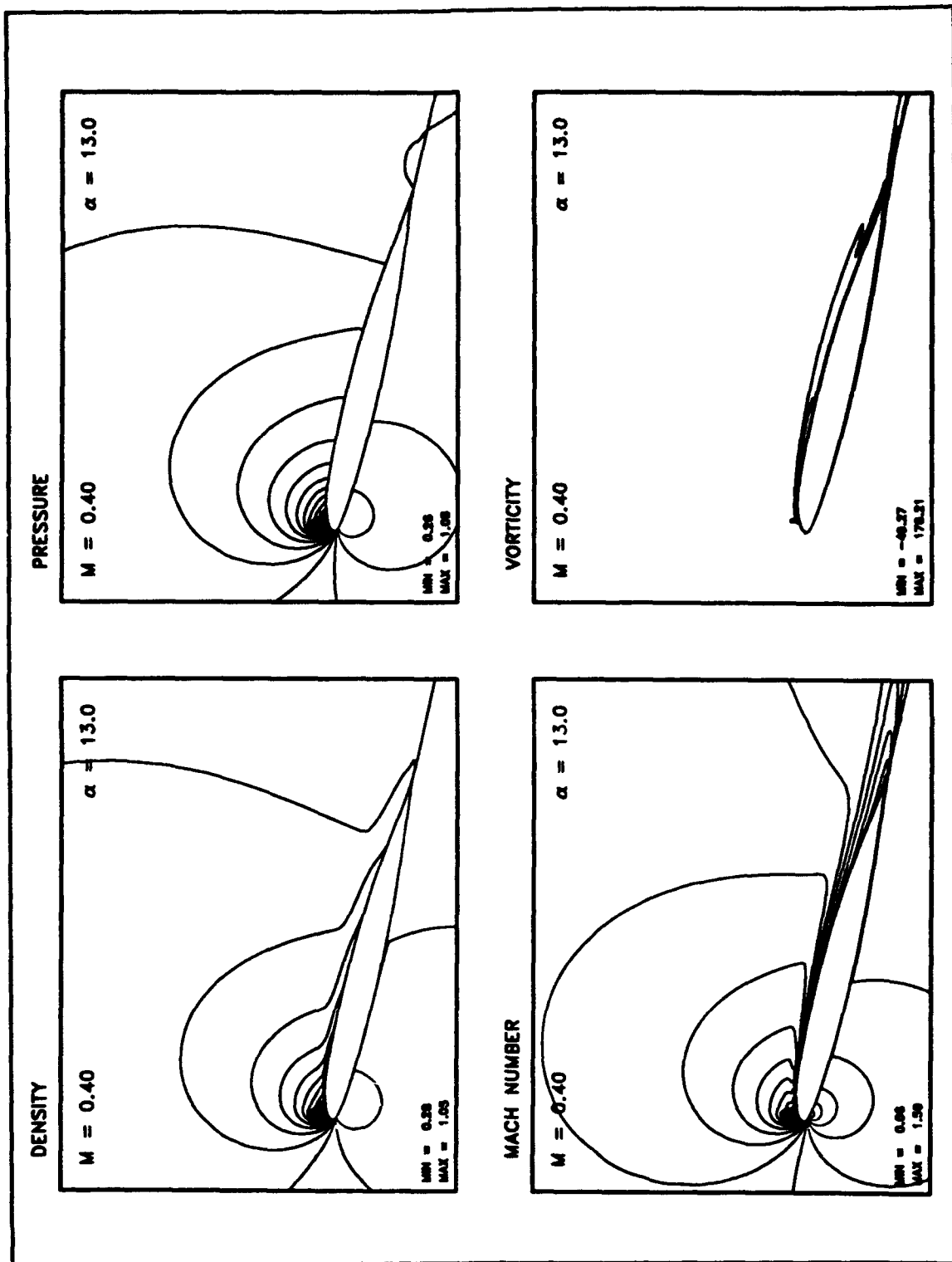


Figure 5.30
Sikorsky SSC-A09
Steady State ($M=0.4$)

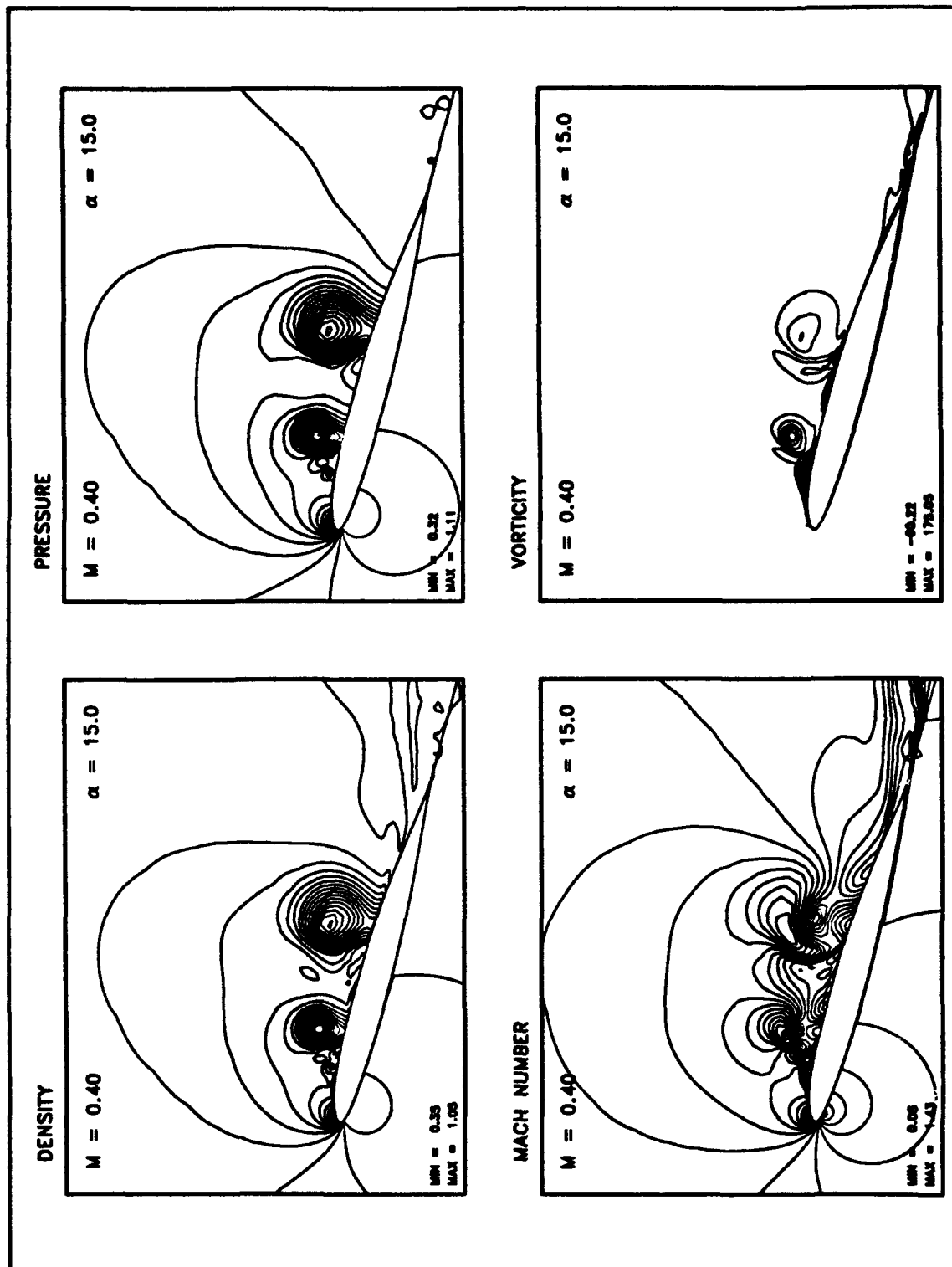


Figure 5.31
Sikorsky SSC-A09
Steady State ($M=0.4$)

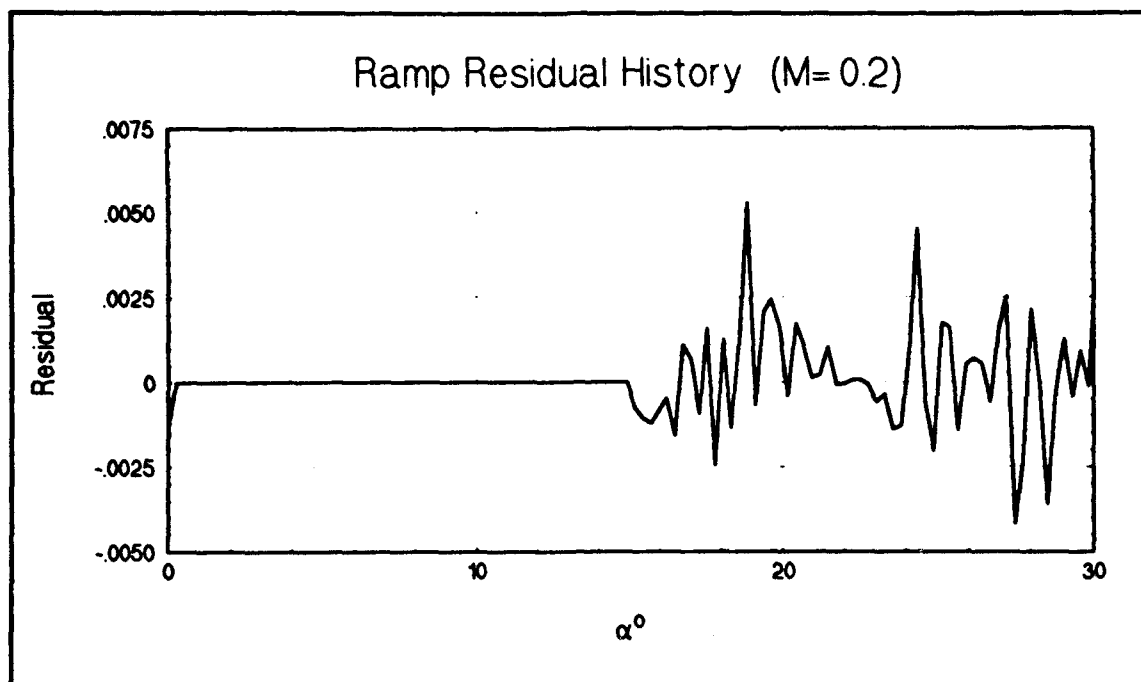


Figure 5.32
Ramp Residual History (M=0.2)

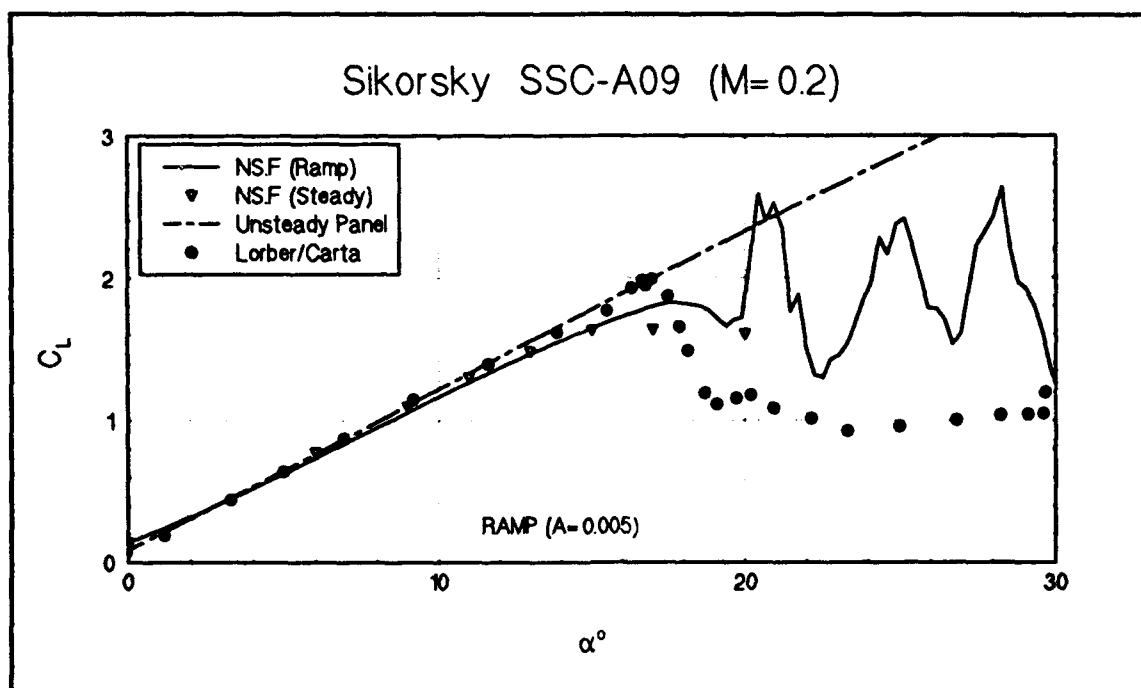


Figure 5.33
Ramp $C_{L\alpha}$ (M=0.2)

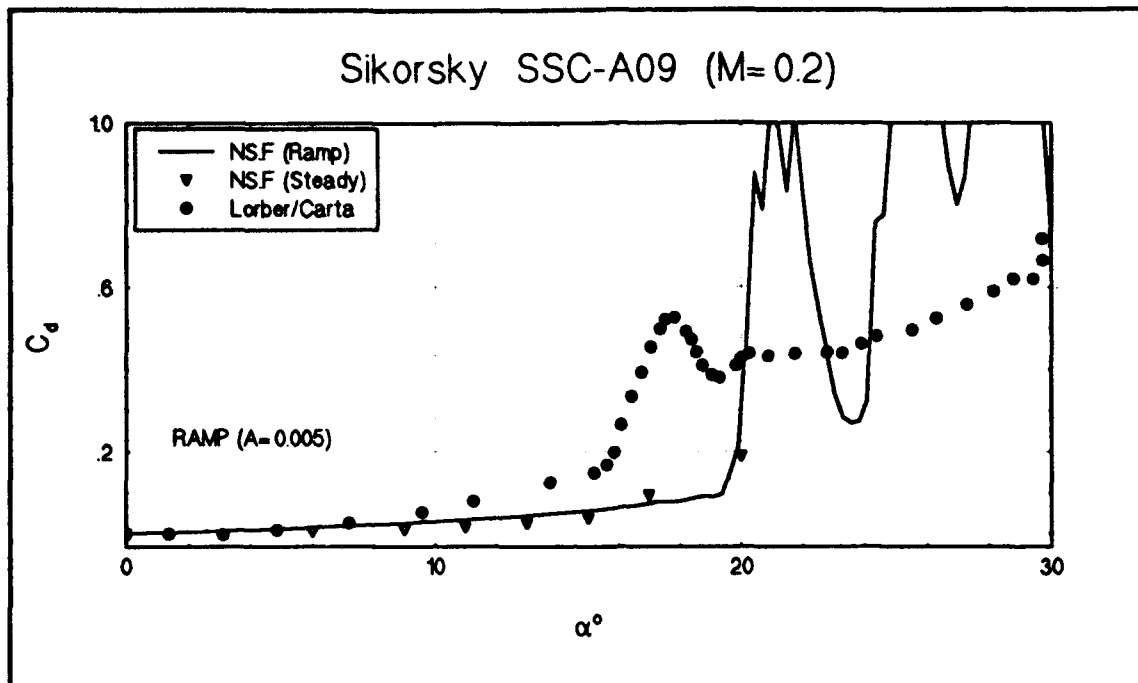


Figure 5.34
Ramp C_{da} ($M=0.2$)

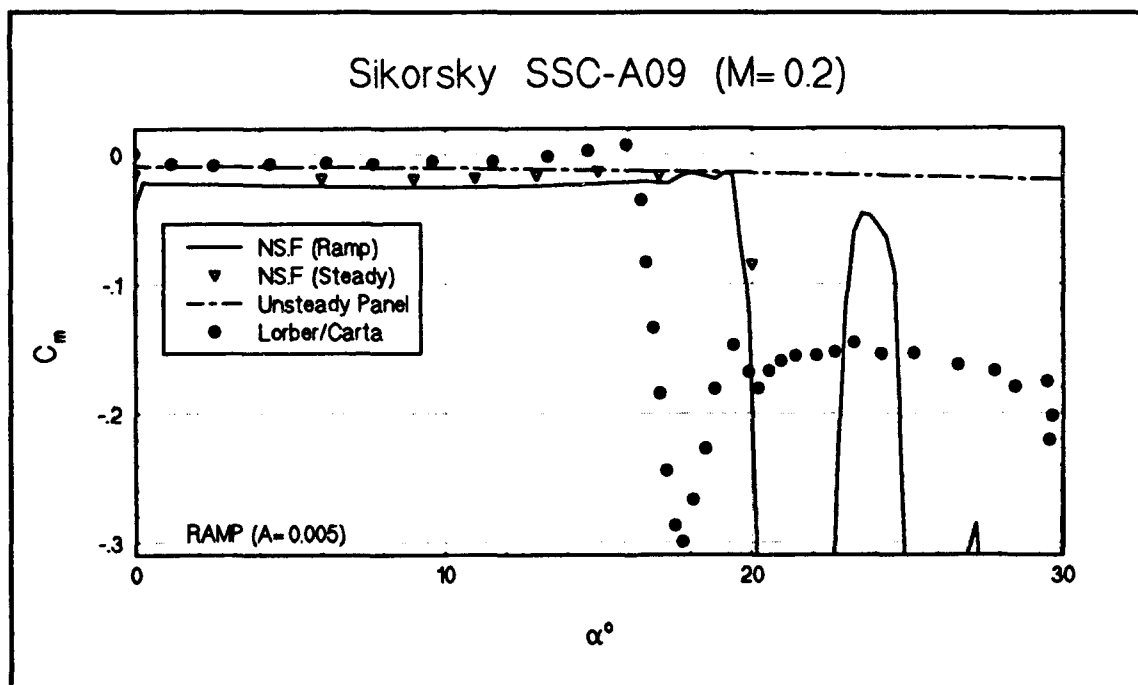


Figure 5.35
Ramp C_{Hs} ($M=0.2$)

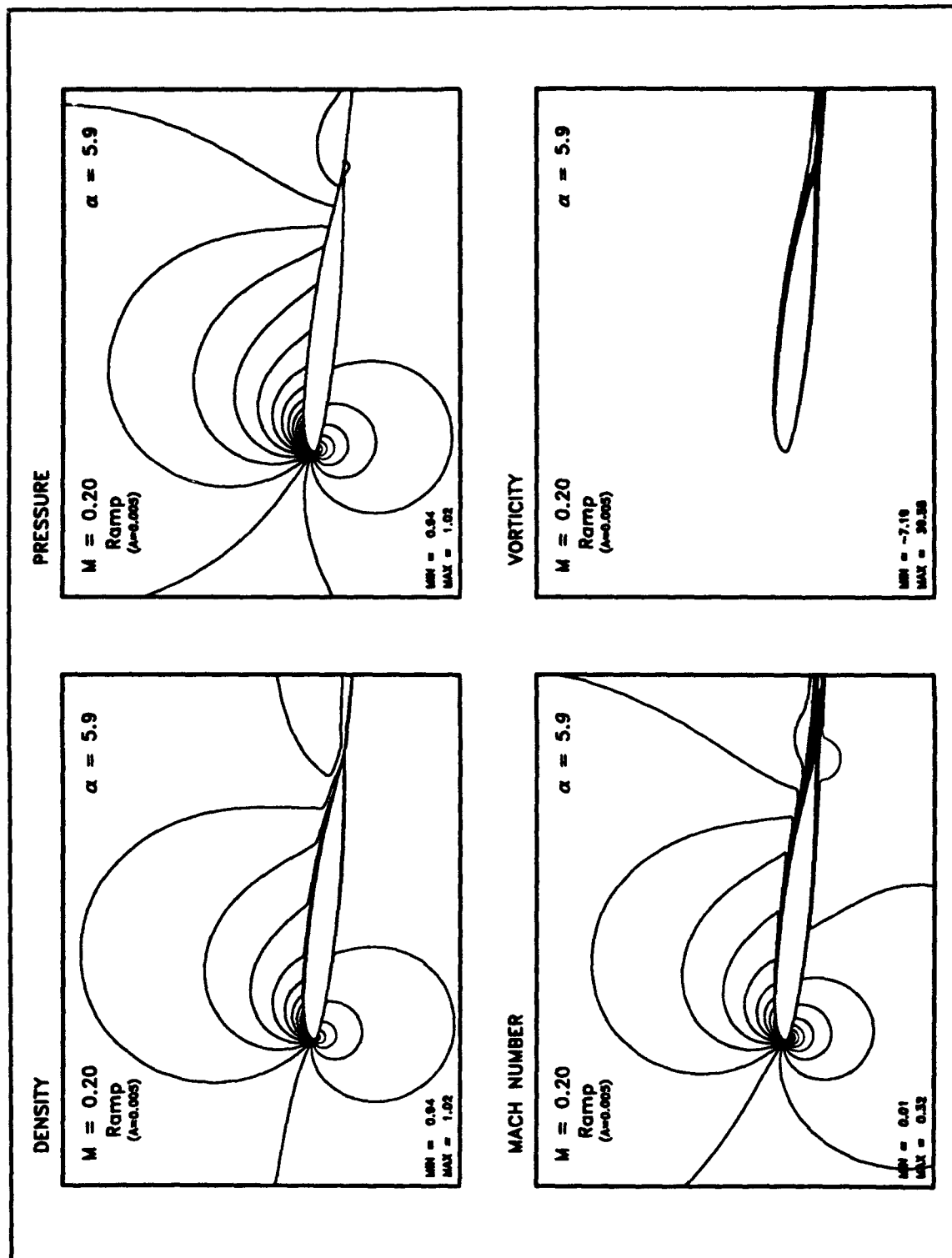


Figure 5.36
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.2$)

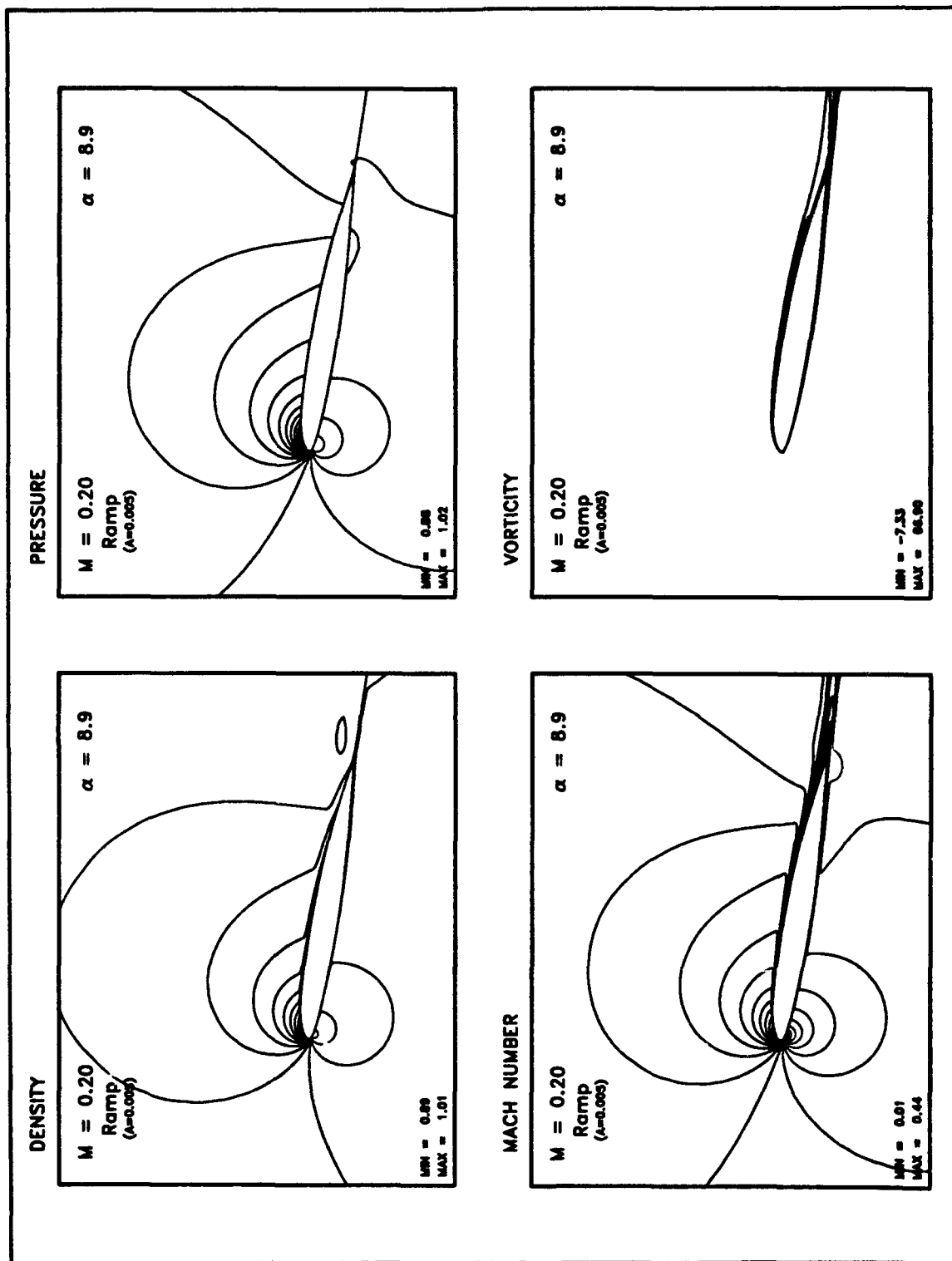


Figure 5.37
Sikorsky SSC-A09
Ramp ($\lambda = 0.005$, $M = 0.2$)

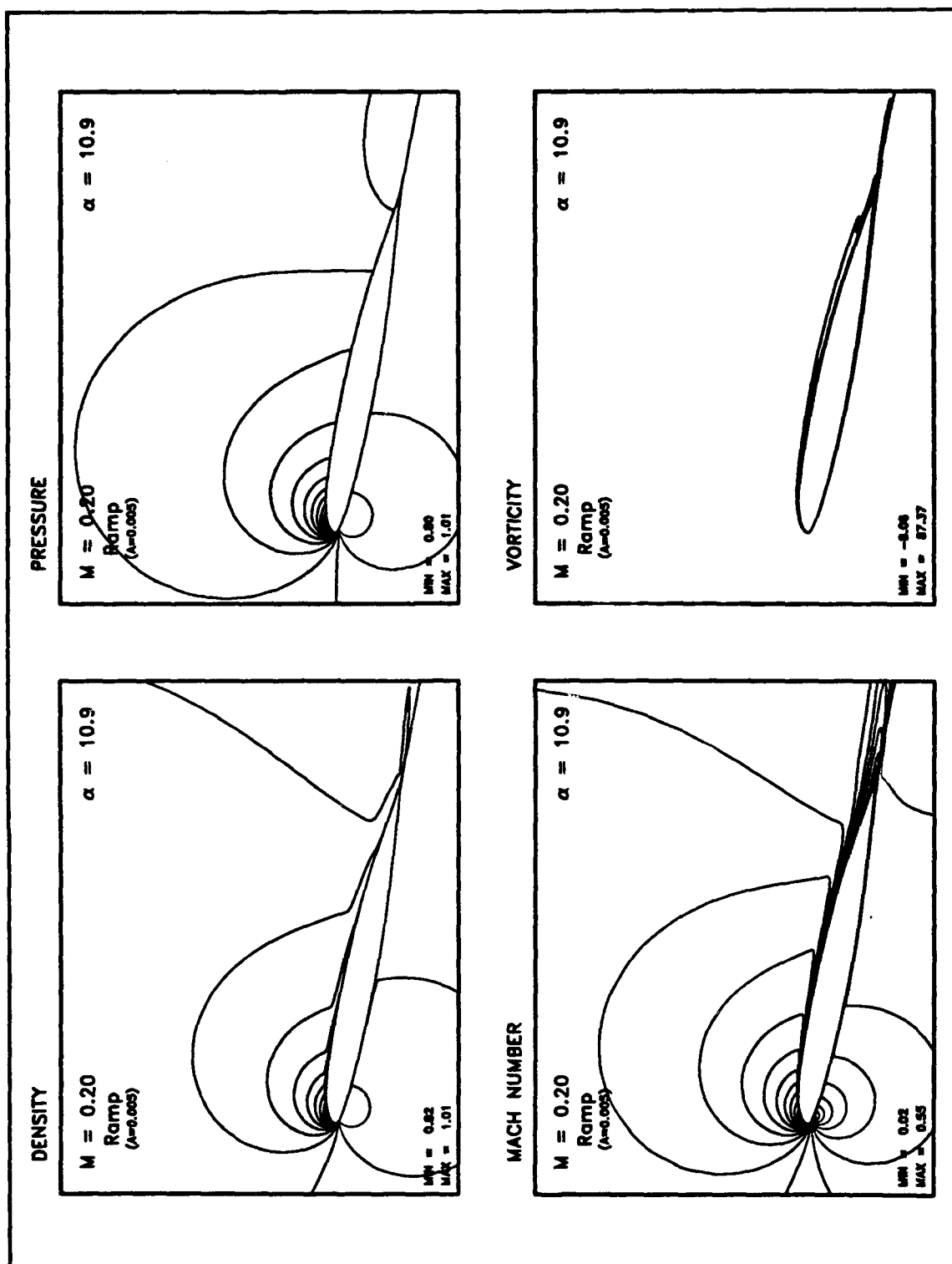


Figure 5.38
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.2$)

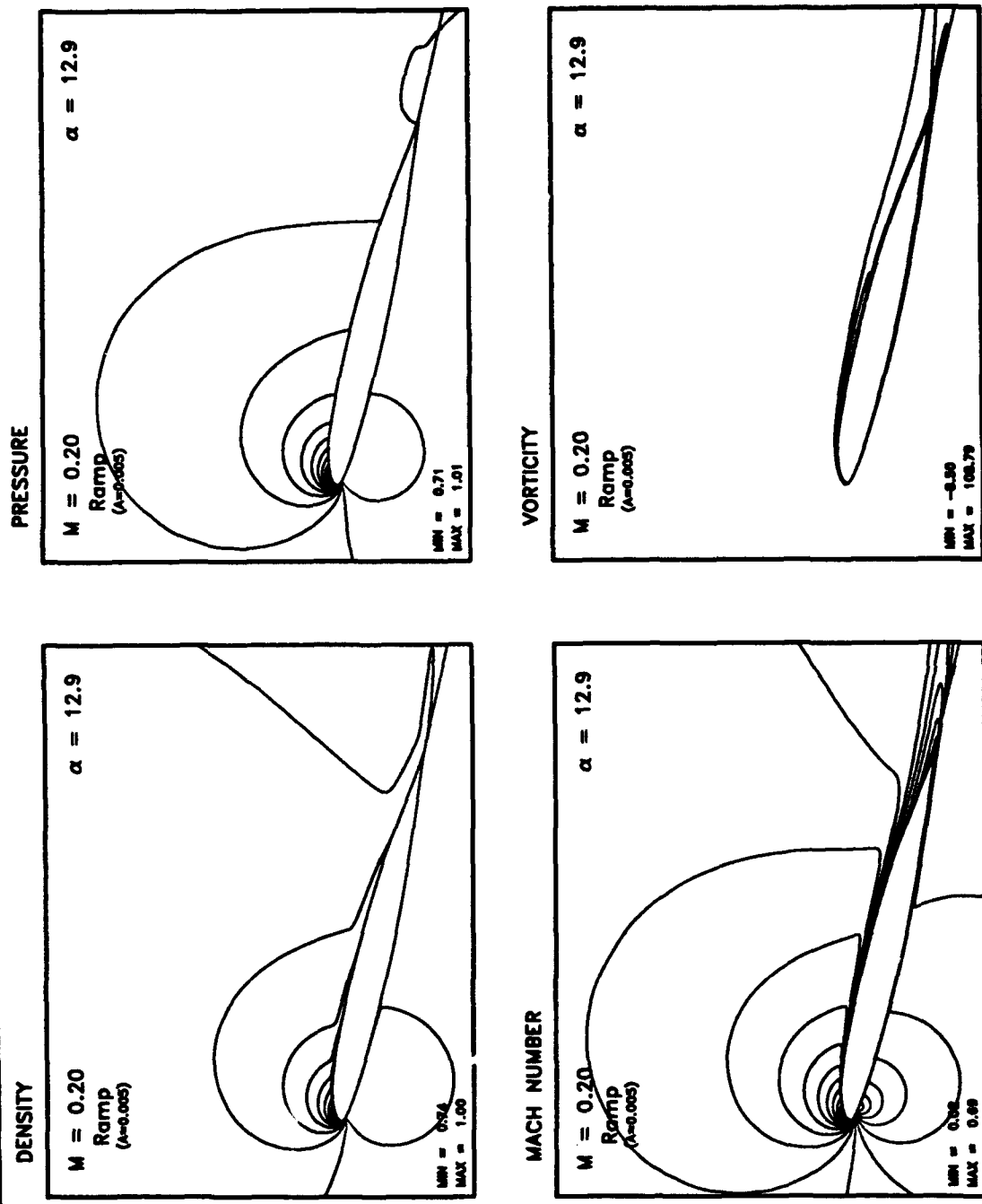


Figure 5.39
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.2$)

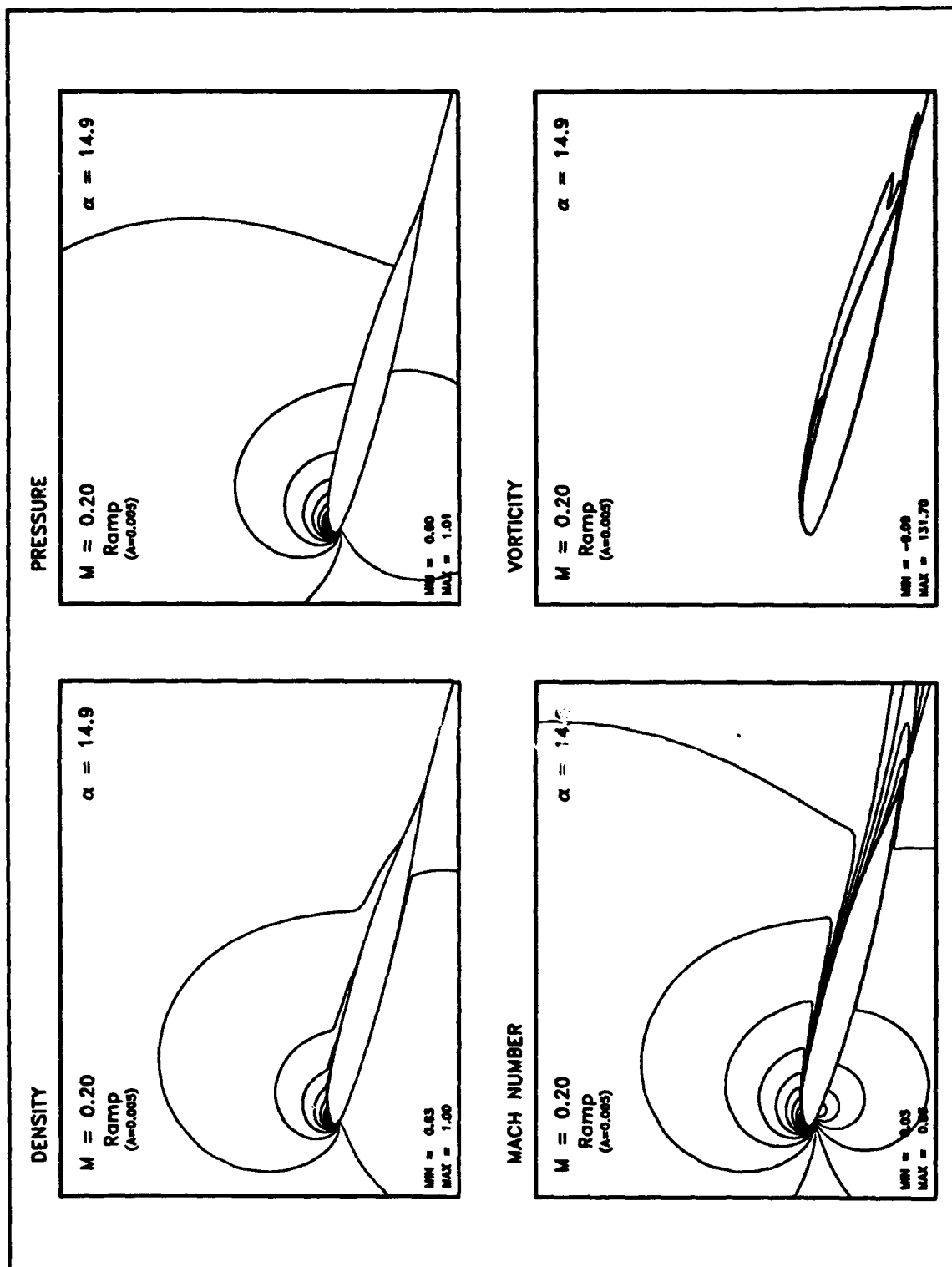


Figure 5.40
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.2$)

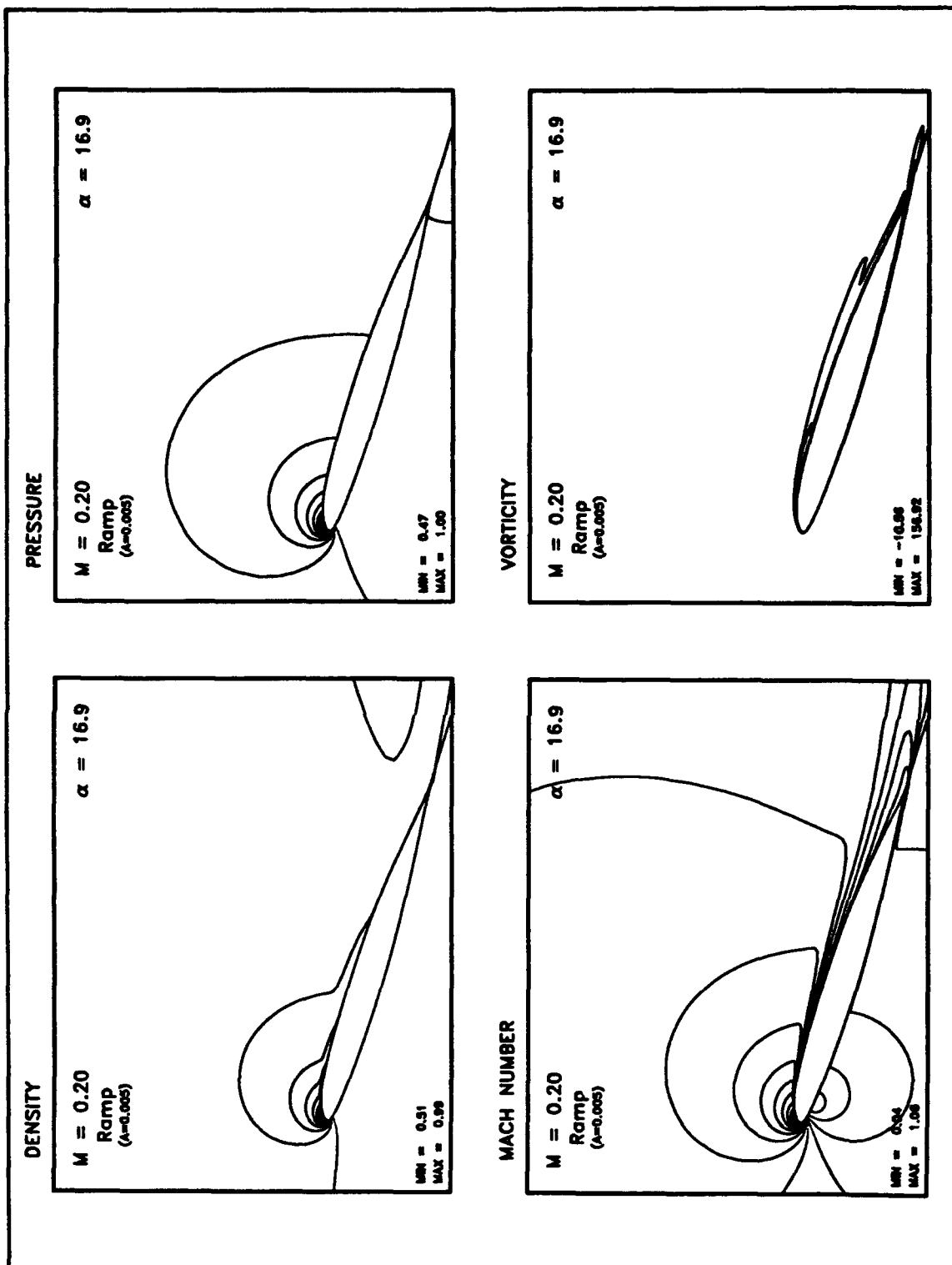


Figure 5.41
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.2$)

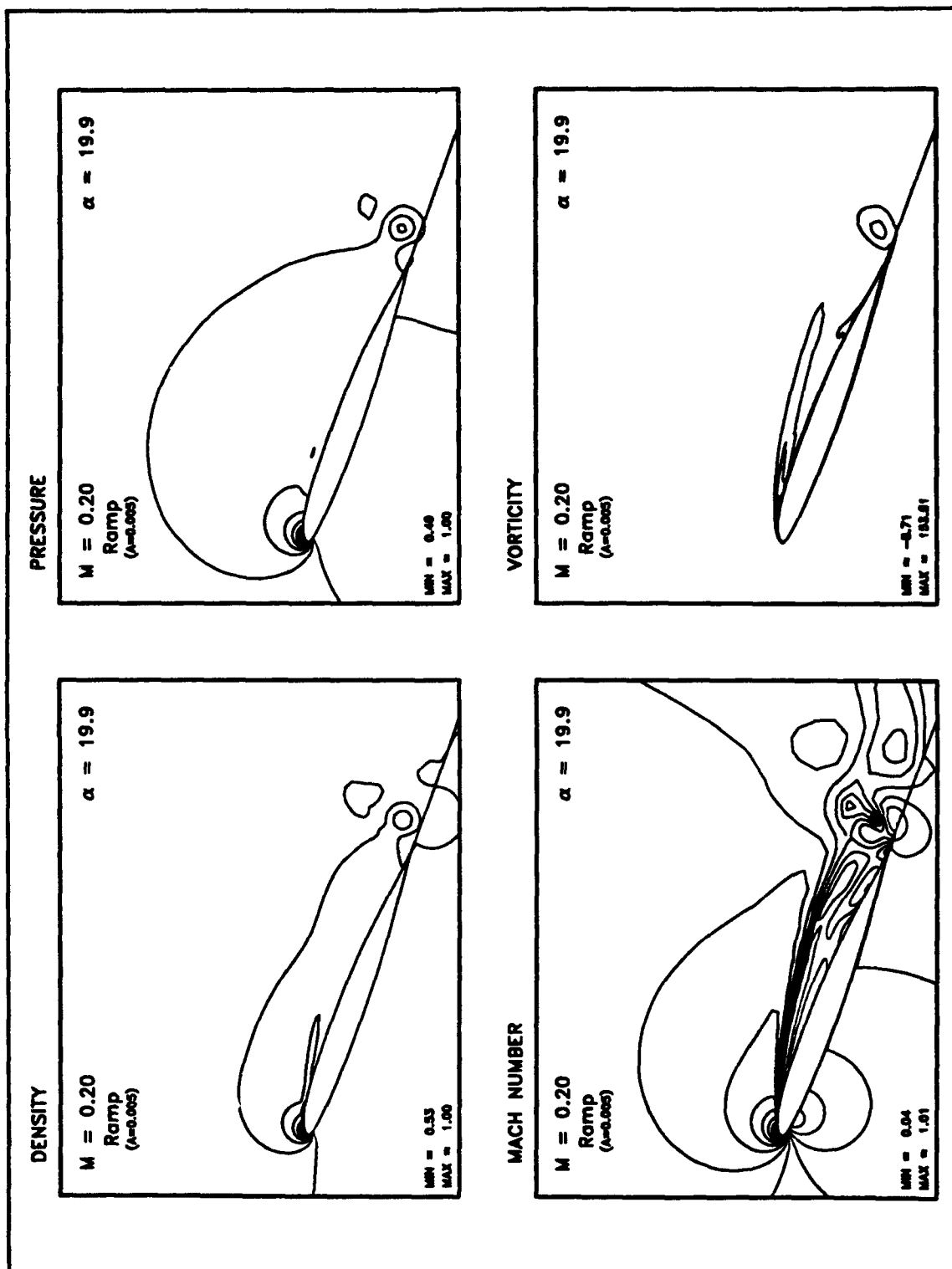


Figure 5.42
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.2$)

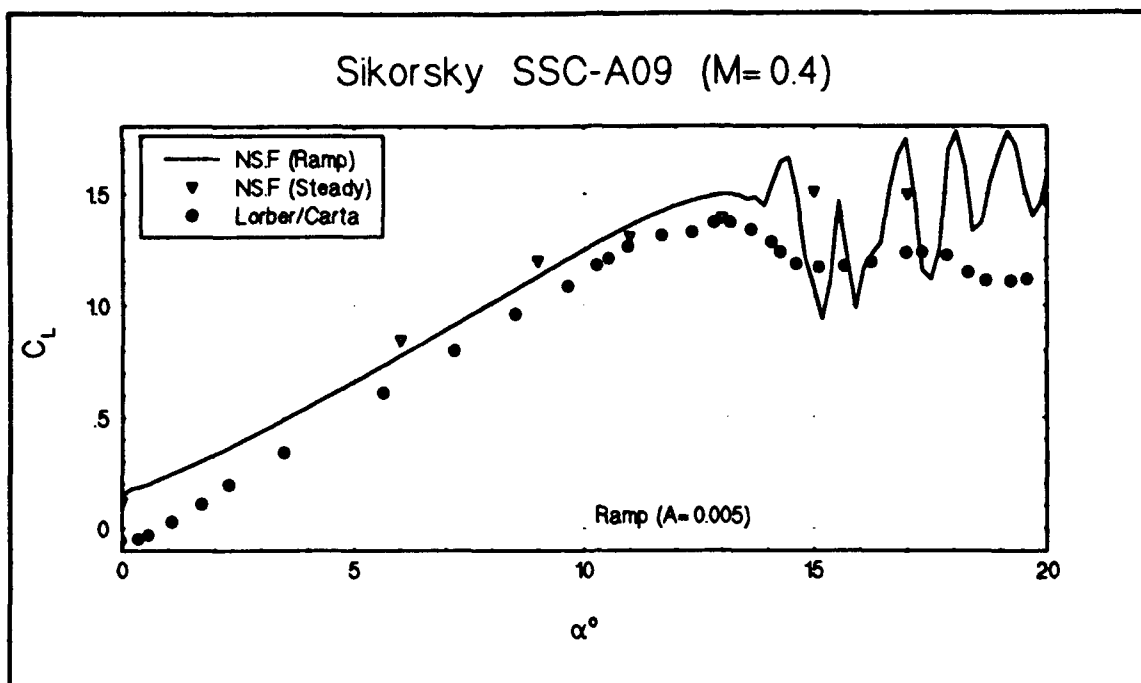


Figure 5.43
Ramp $C_{L\alpha}$ ($M=0.4$)

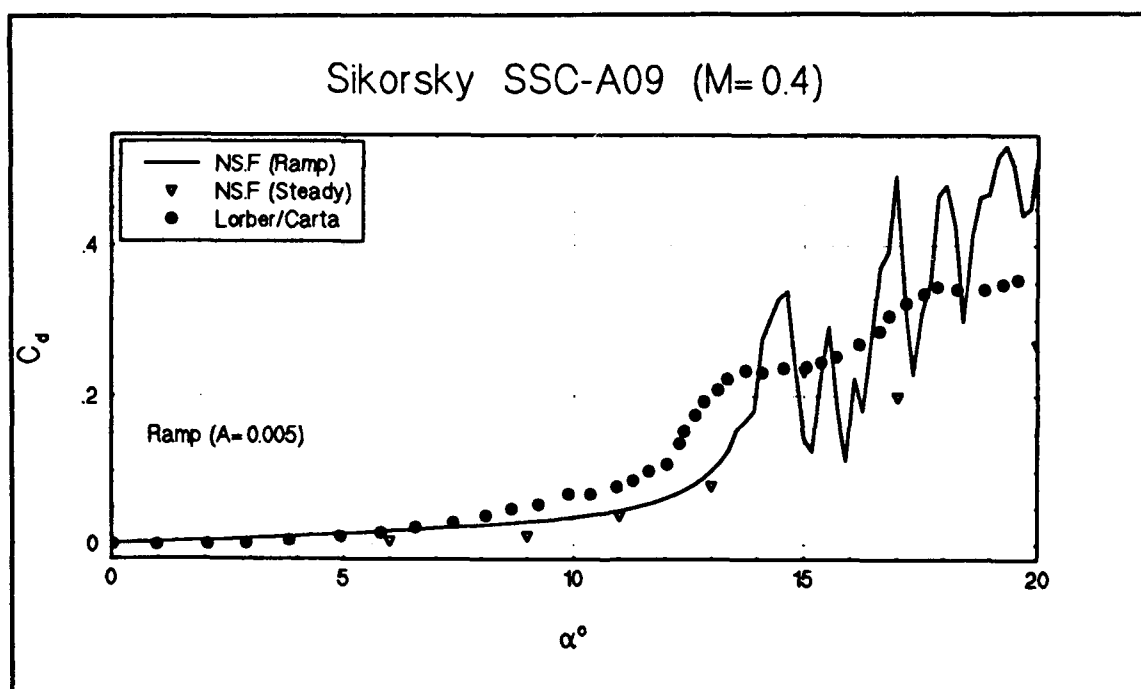


Figure 5.44
Ramp $C_{d\alpha}$ ($M=0.4$)

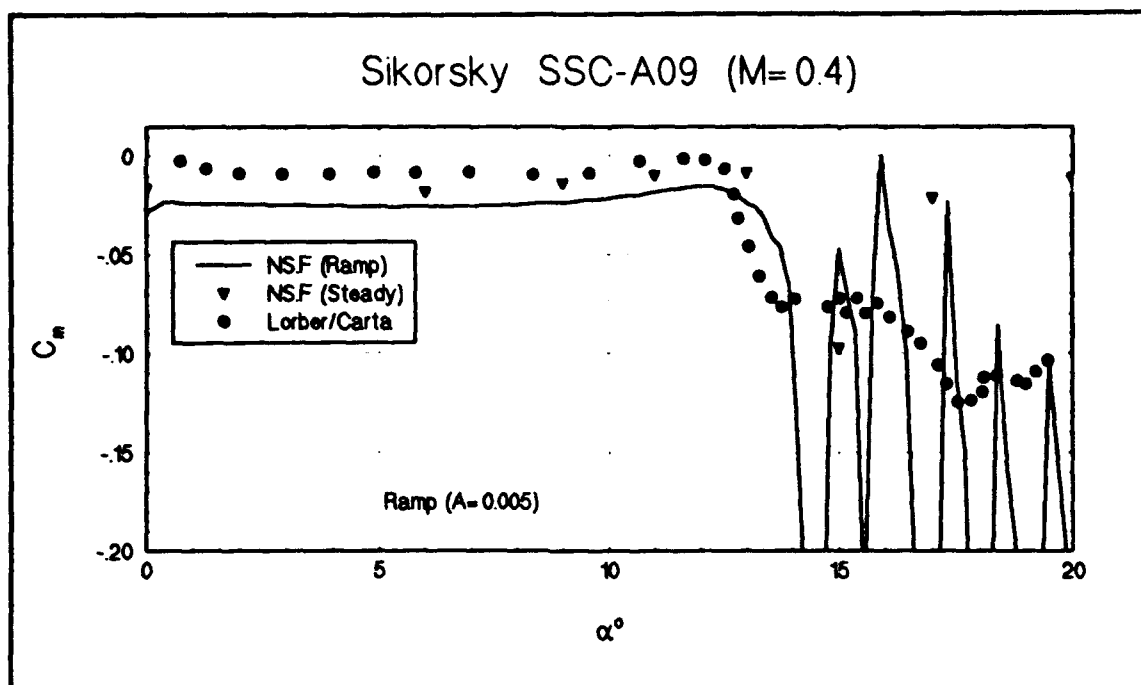


Figure 5.45
Ramp C_{Hs} ($M=0.4$)

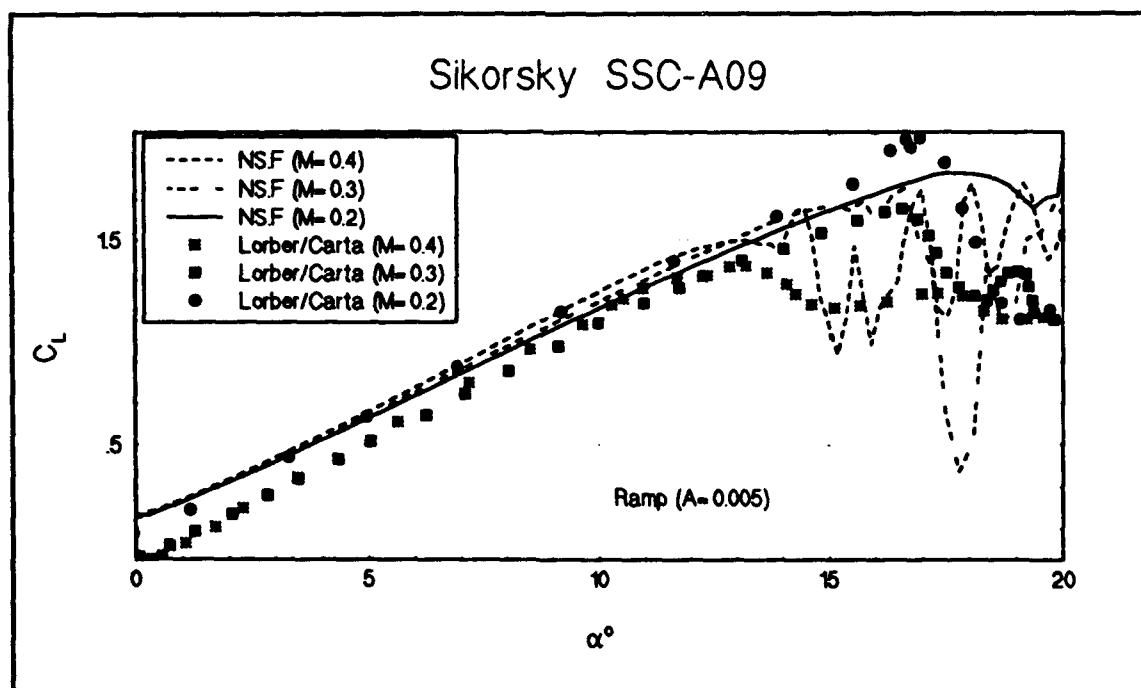


Figure 5.46
Ramp C_{La} ($M=0.2, 0.3, 0.4$)

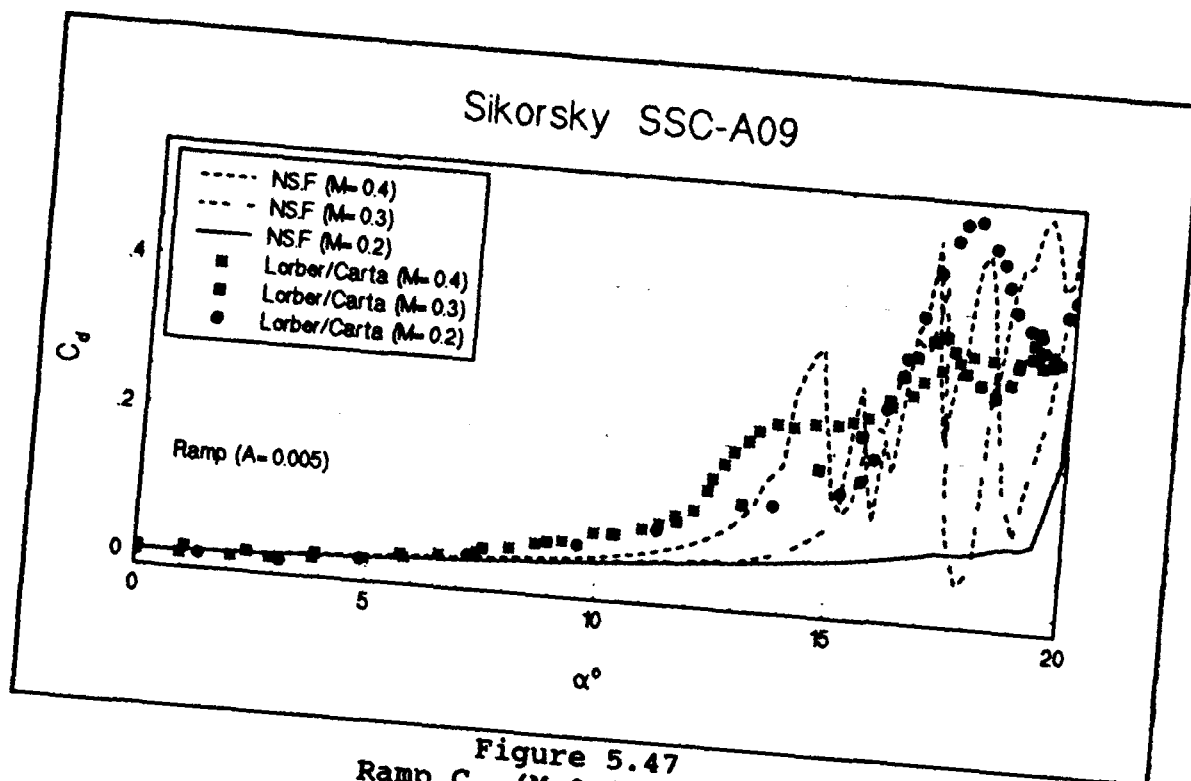


Figure 5.47
Ramp C_{ds} (M=0.2, 0.3, 0.4)

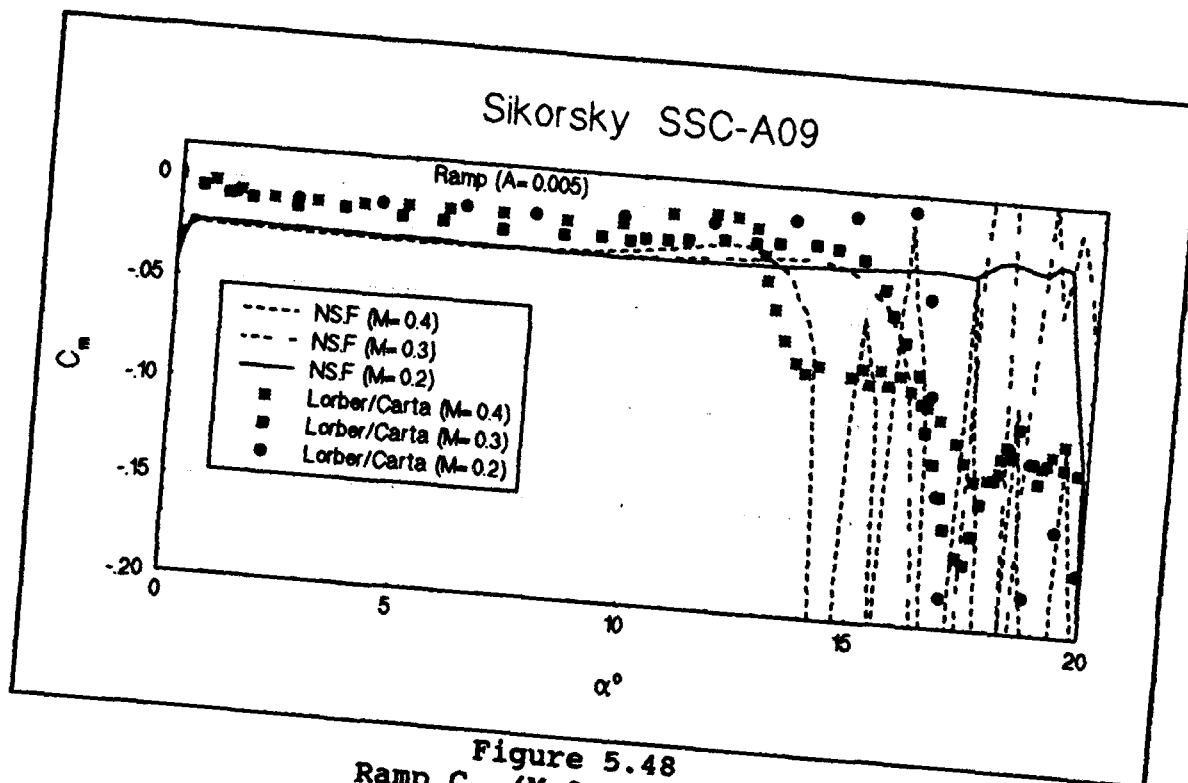


Figure 5.48
Ramp C_{ms} (M=0.2, 0.3, 0.4)

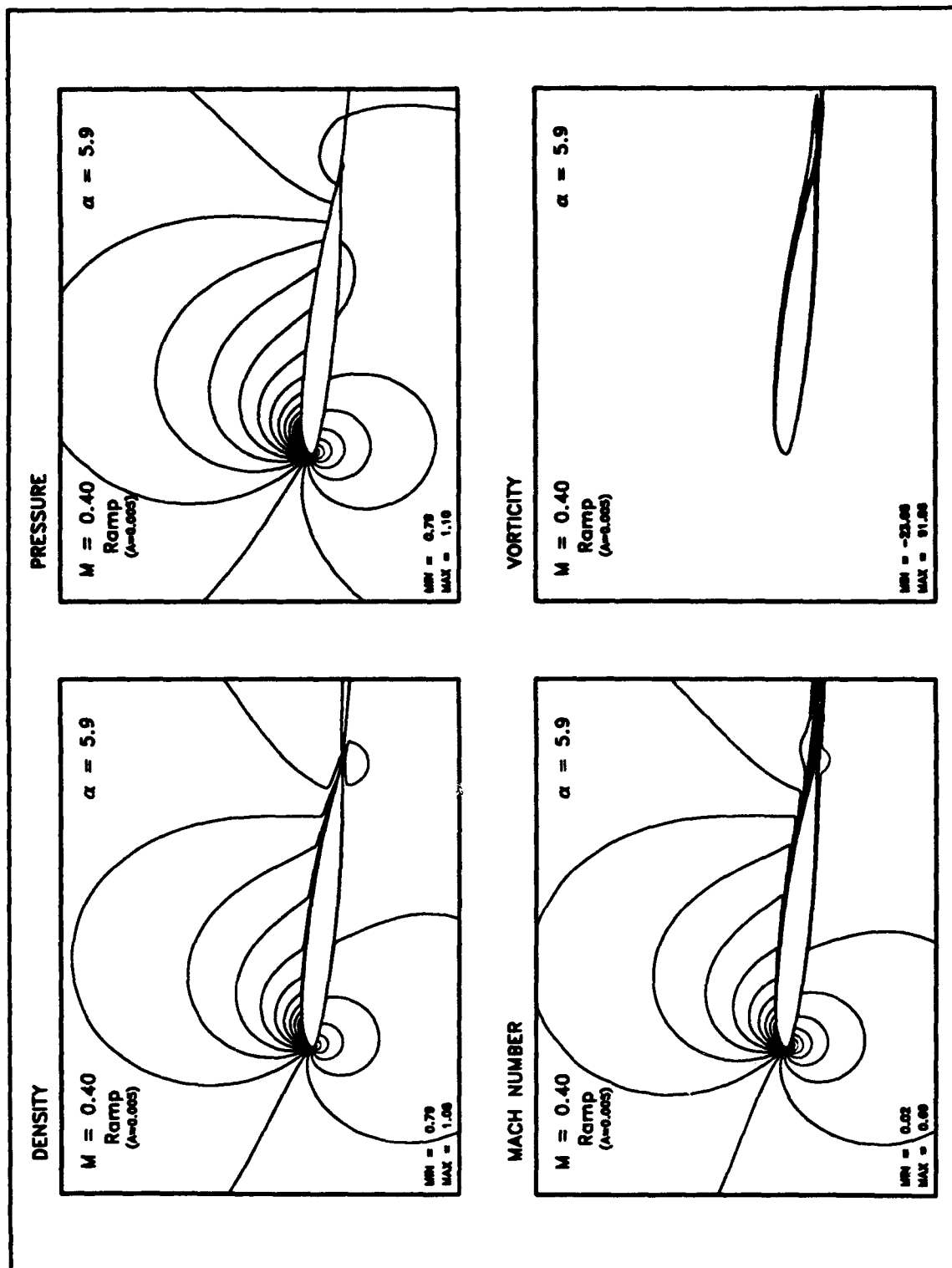


Figure 5.49
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

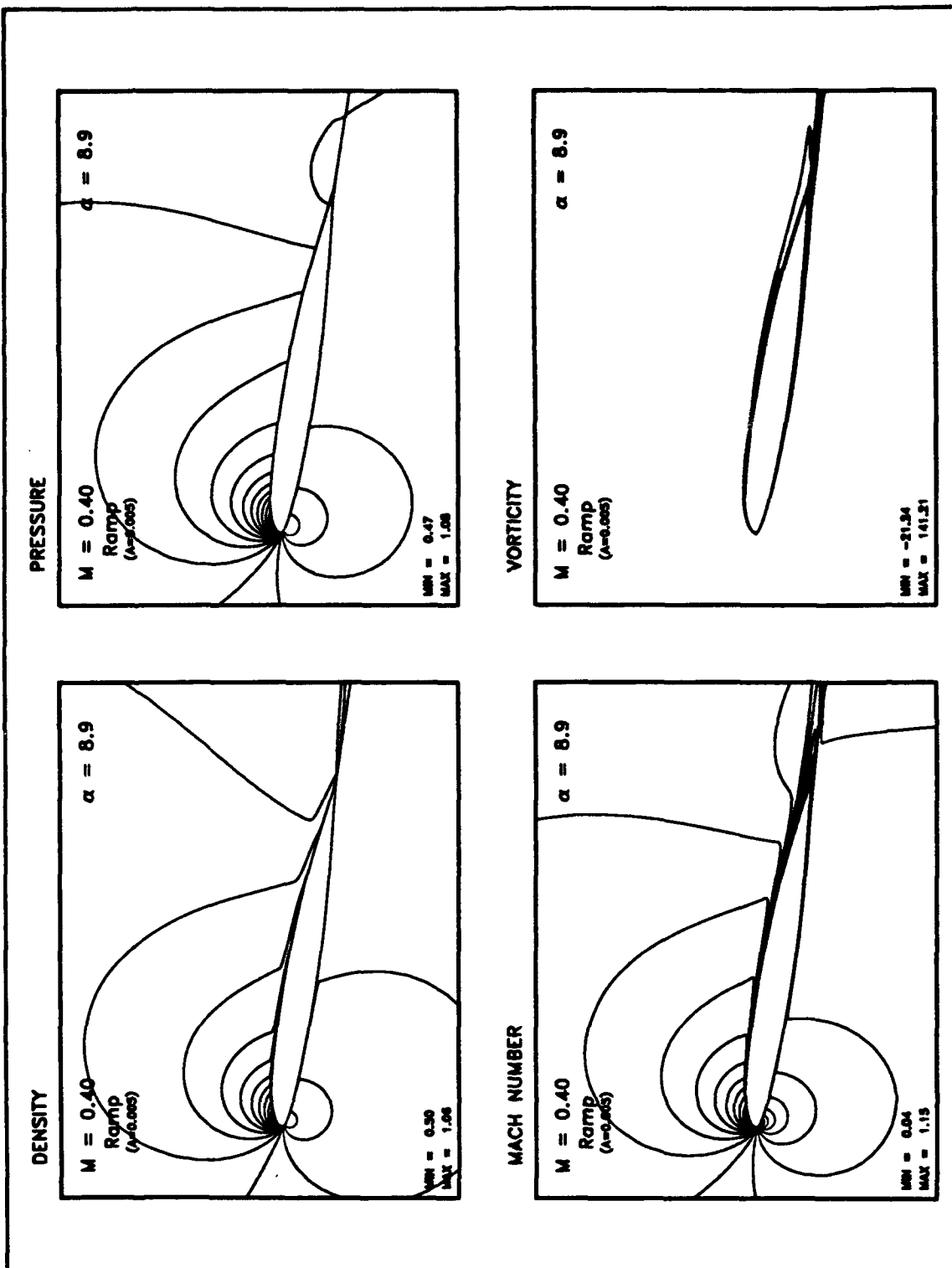


Figure 5.50
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

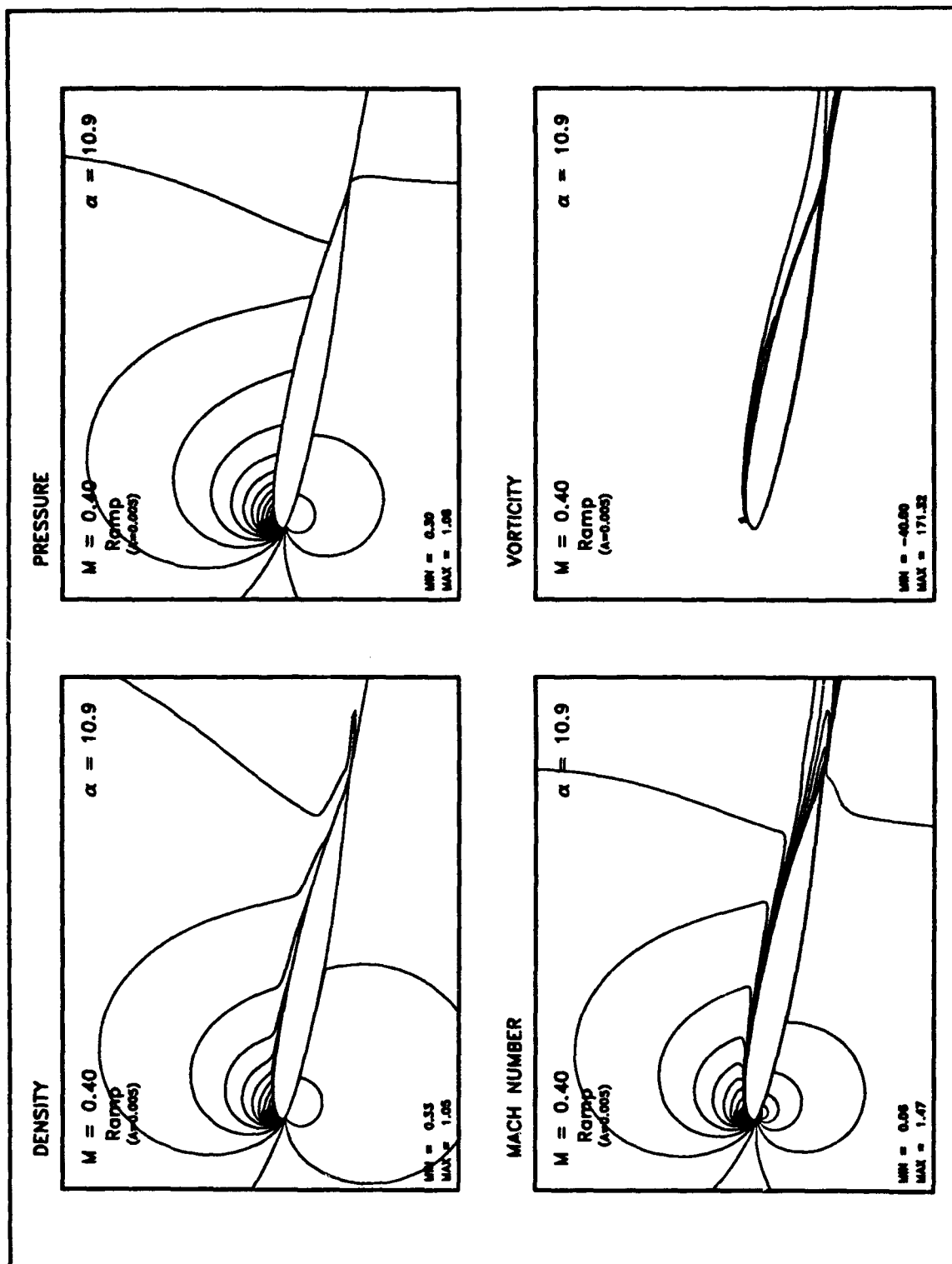


Figure 5.51
Sikorsky SSC-A09
Ramp ($\lambda=0.005$, $M=0.4$)

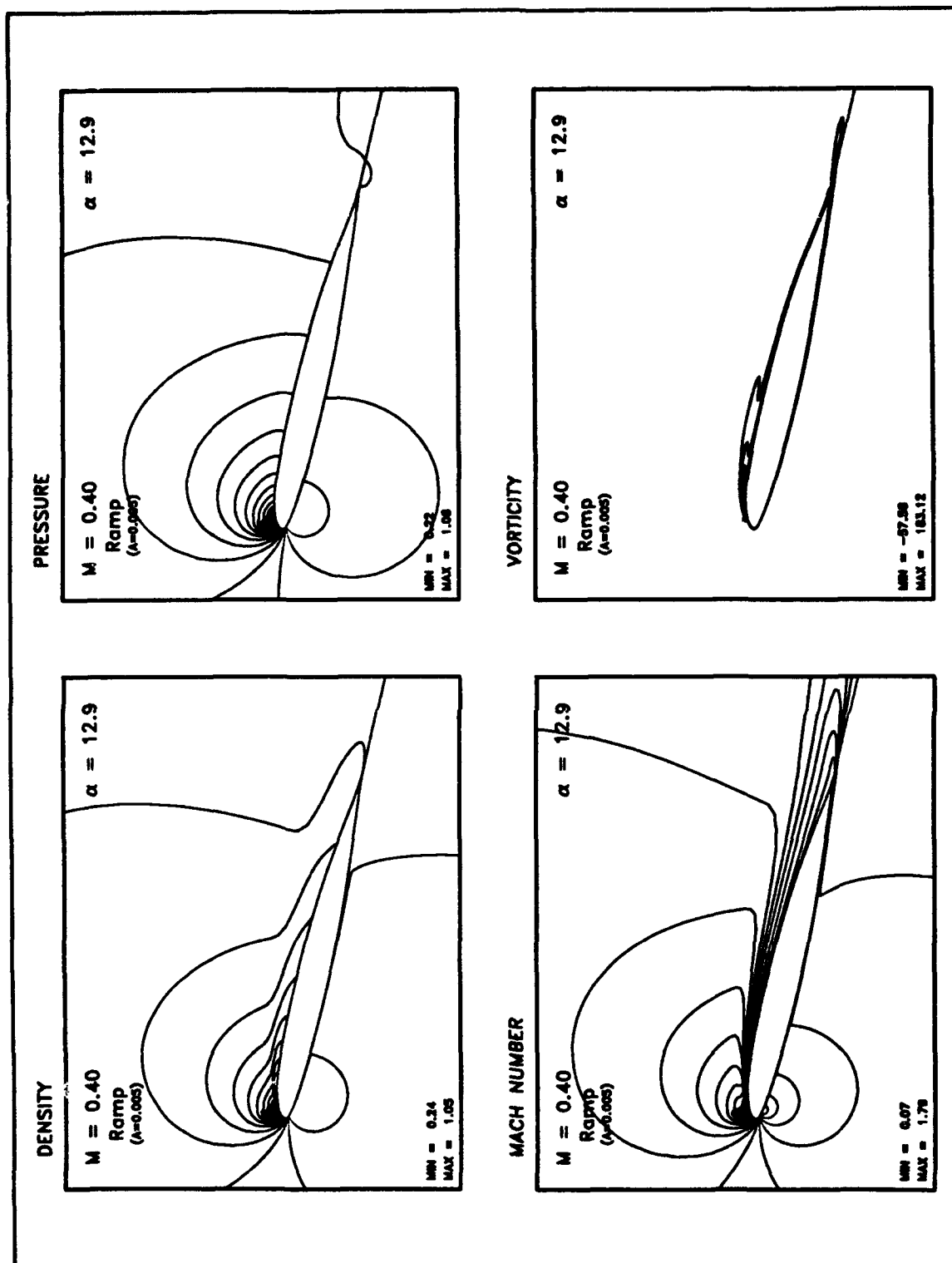


Figure 5.52
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

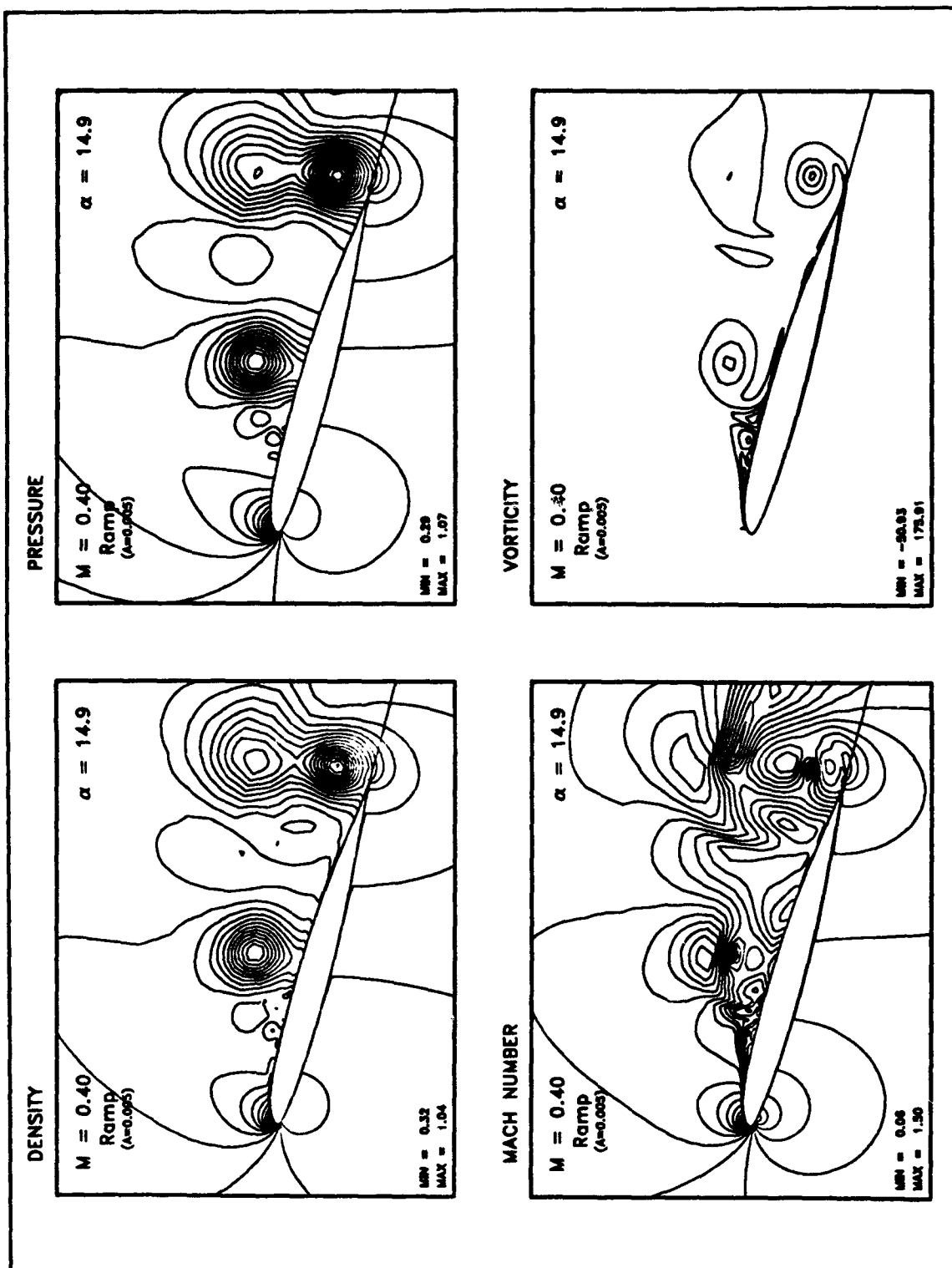


Figure 5.53
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

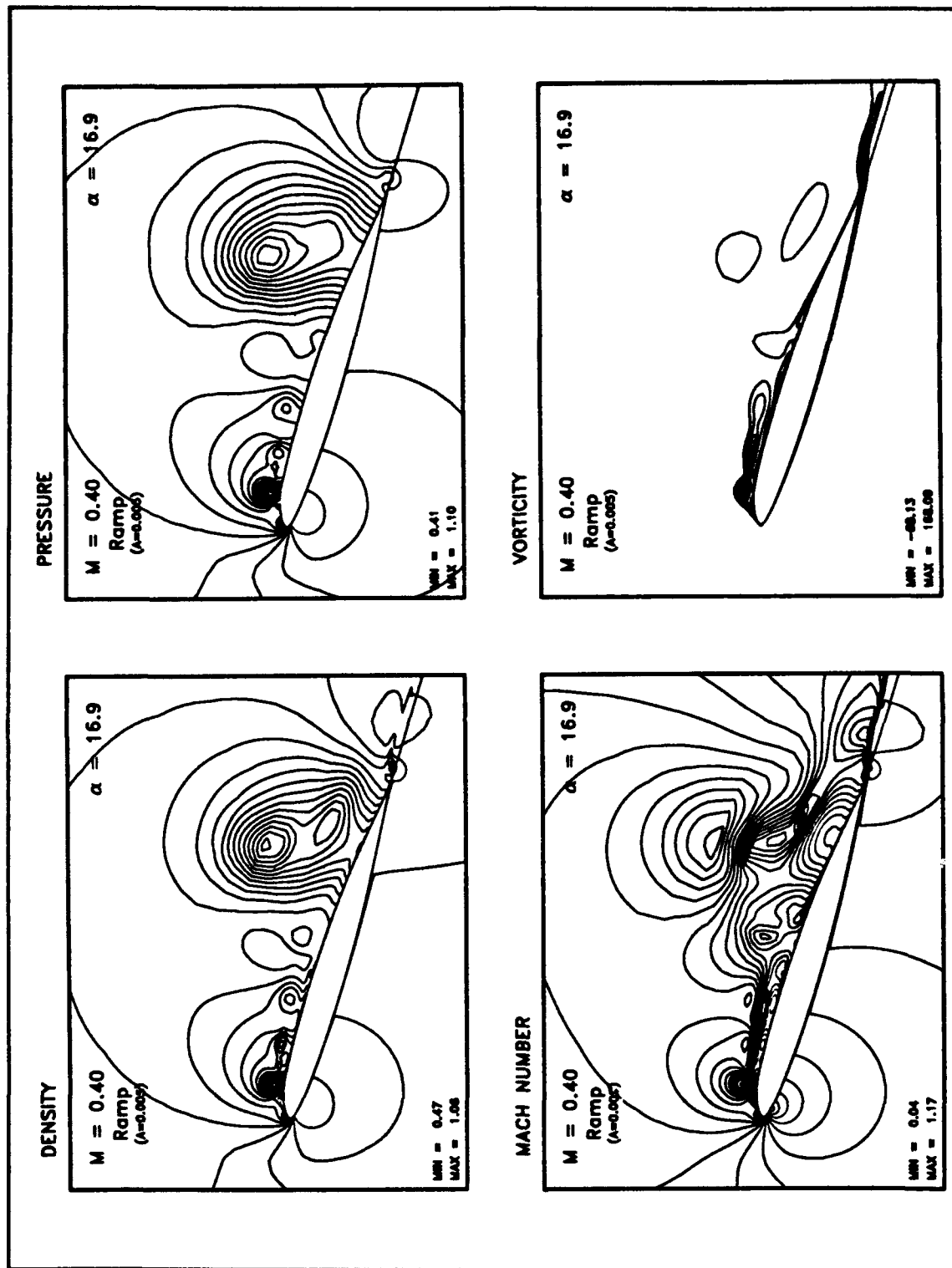


Figure 5.54
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

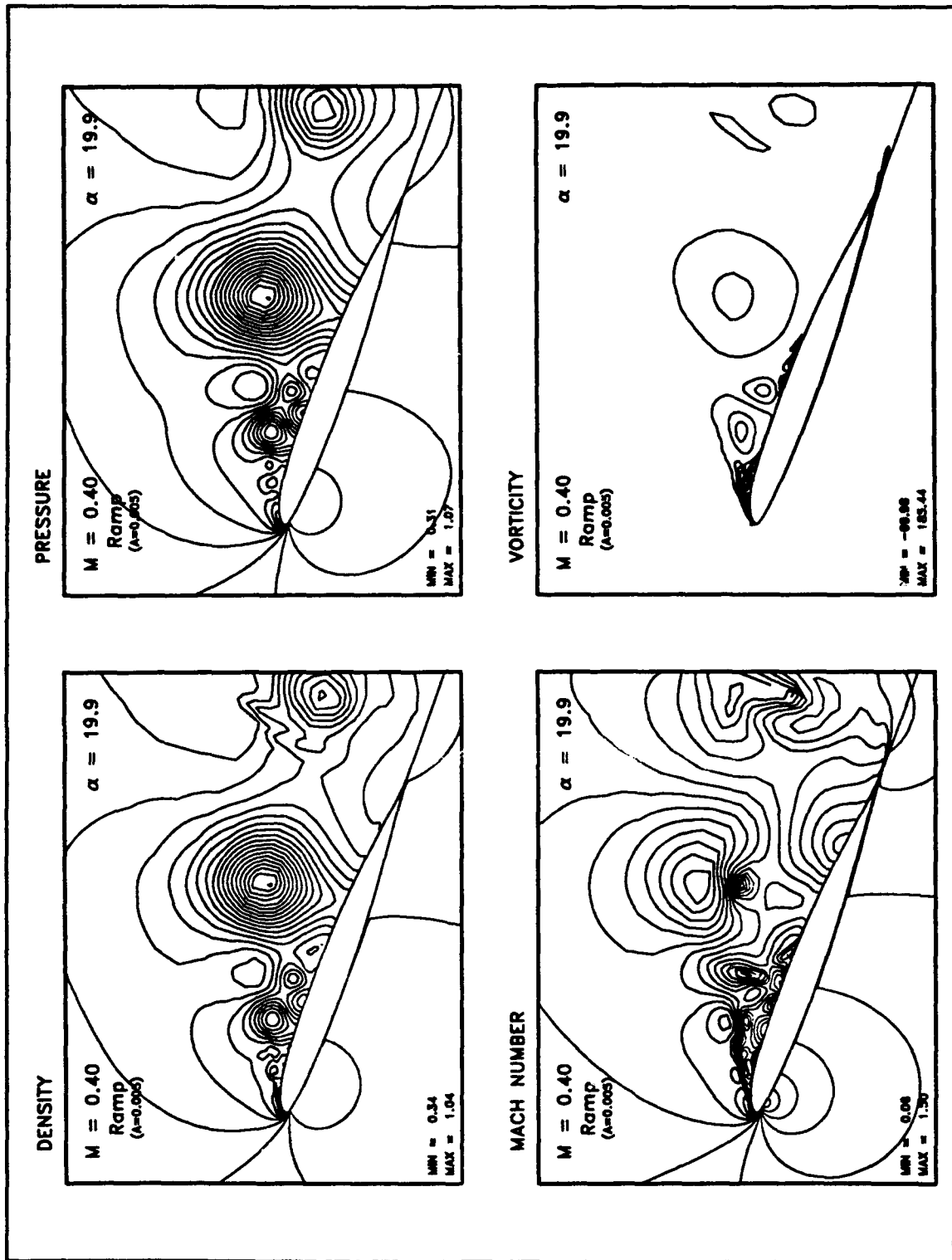


Figure 5.55
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

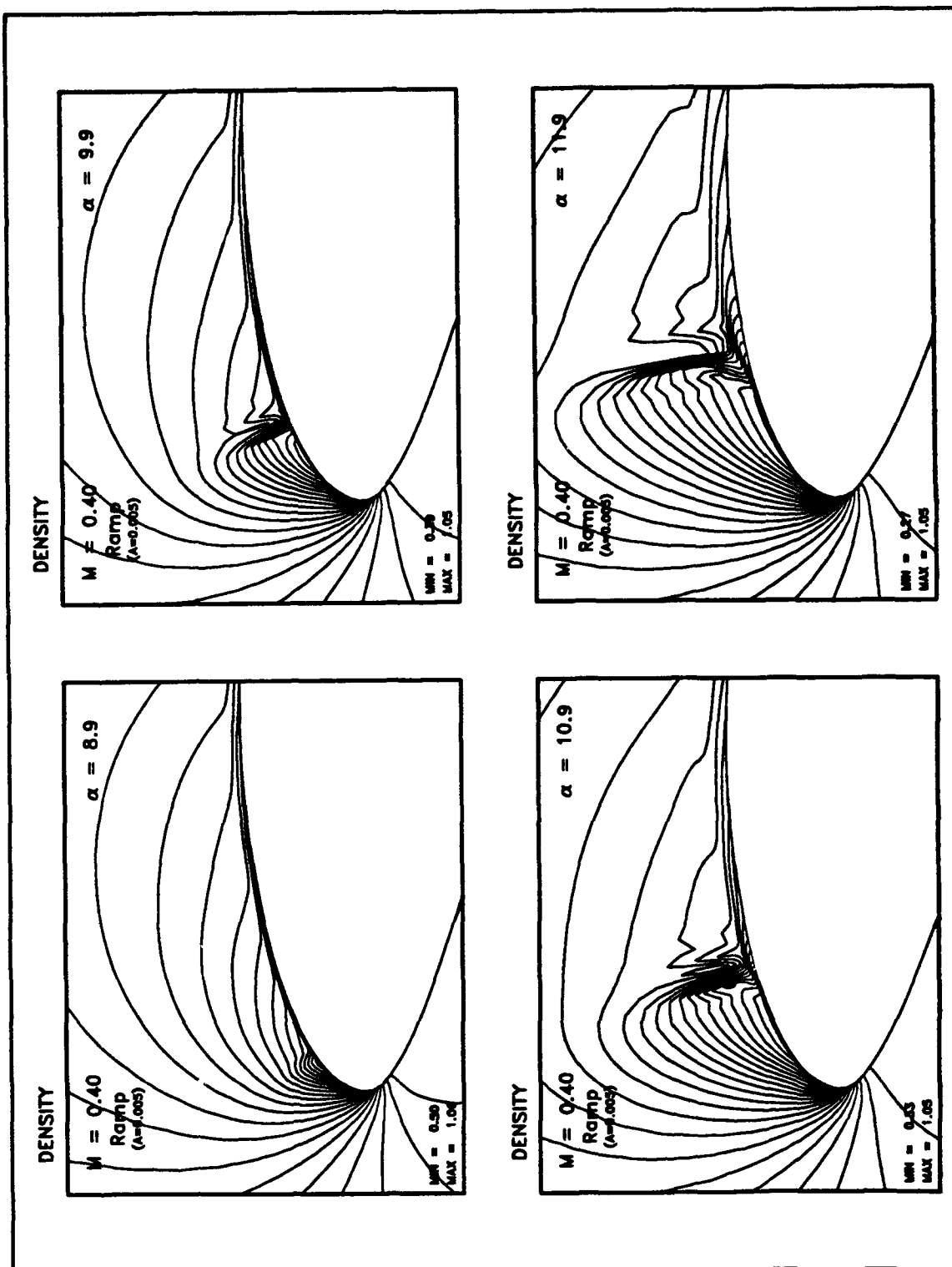


Figure 5.56
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

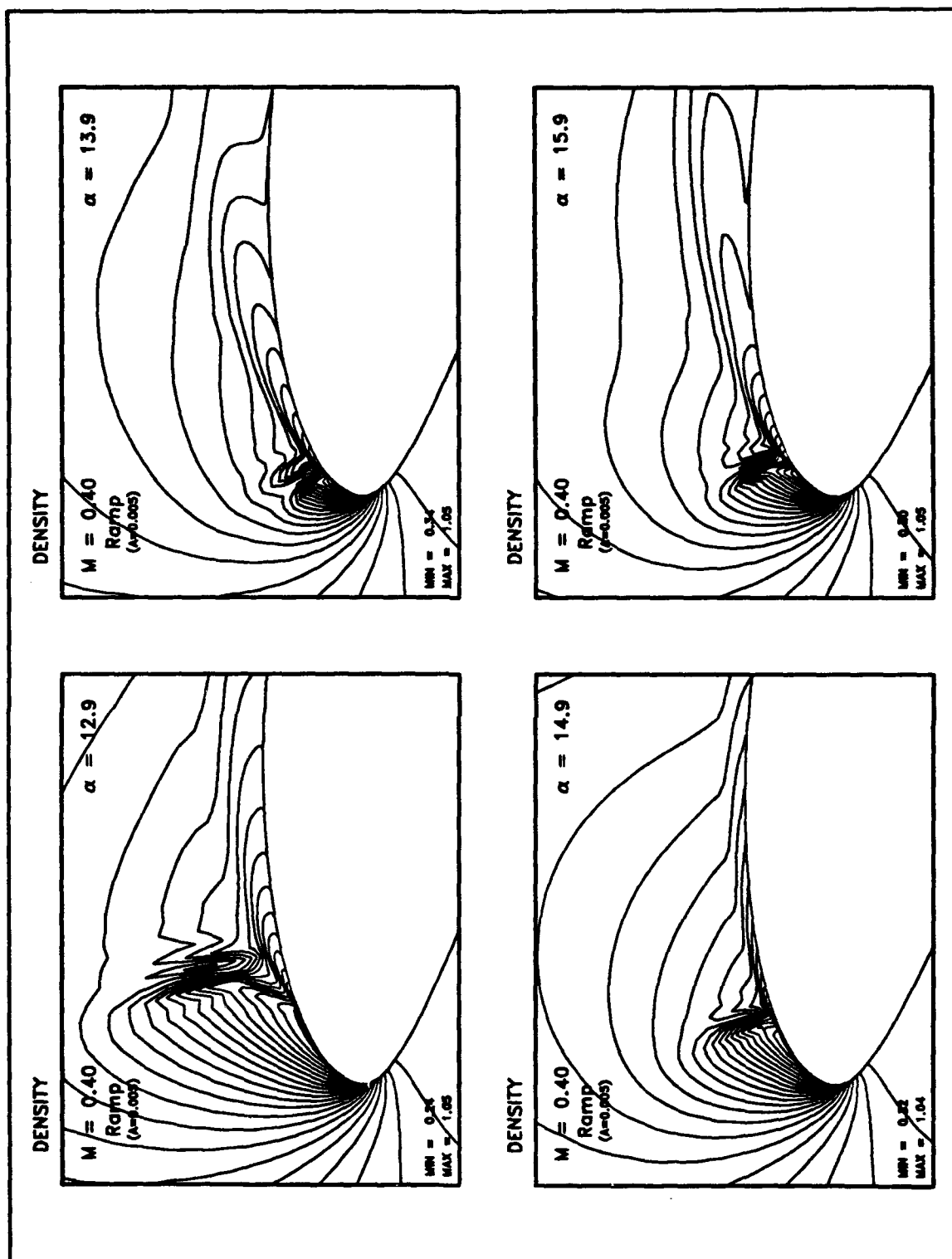


Figure 5.57
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

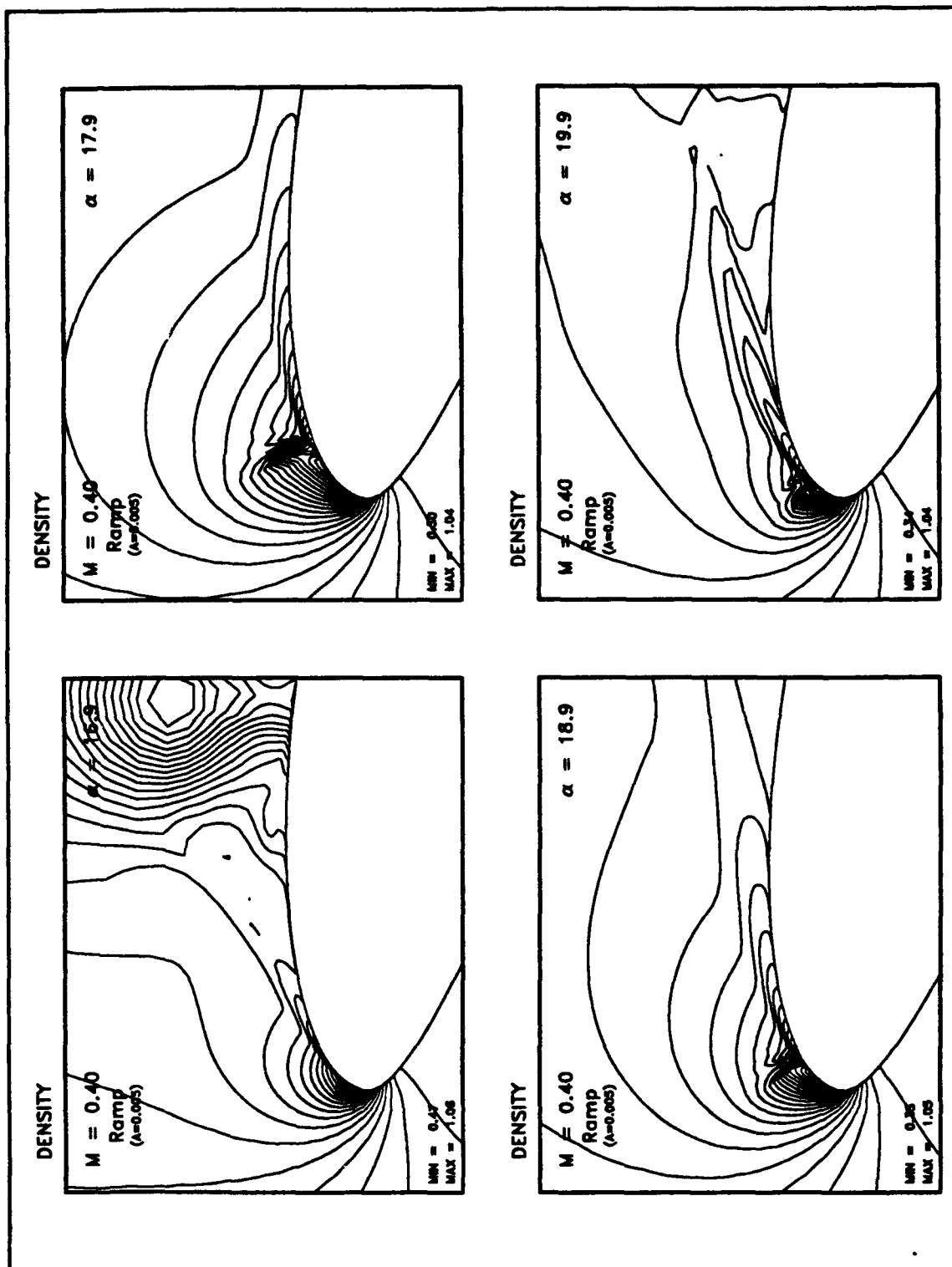


Figure 5.58
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

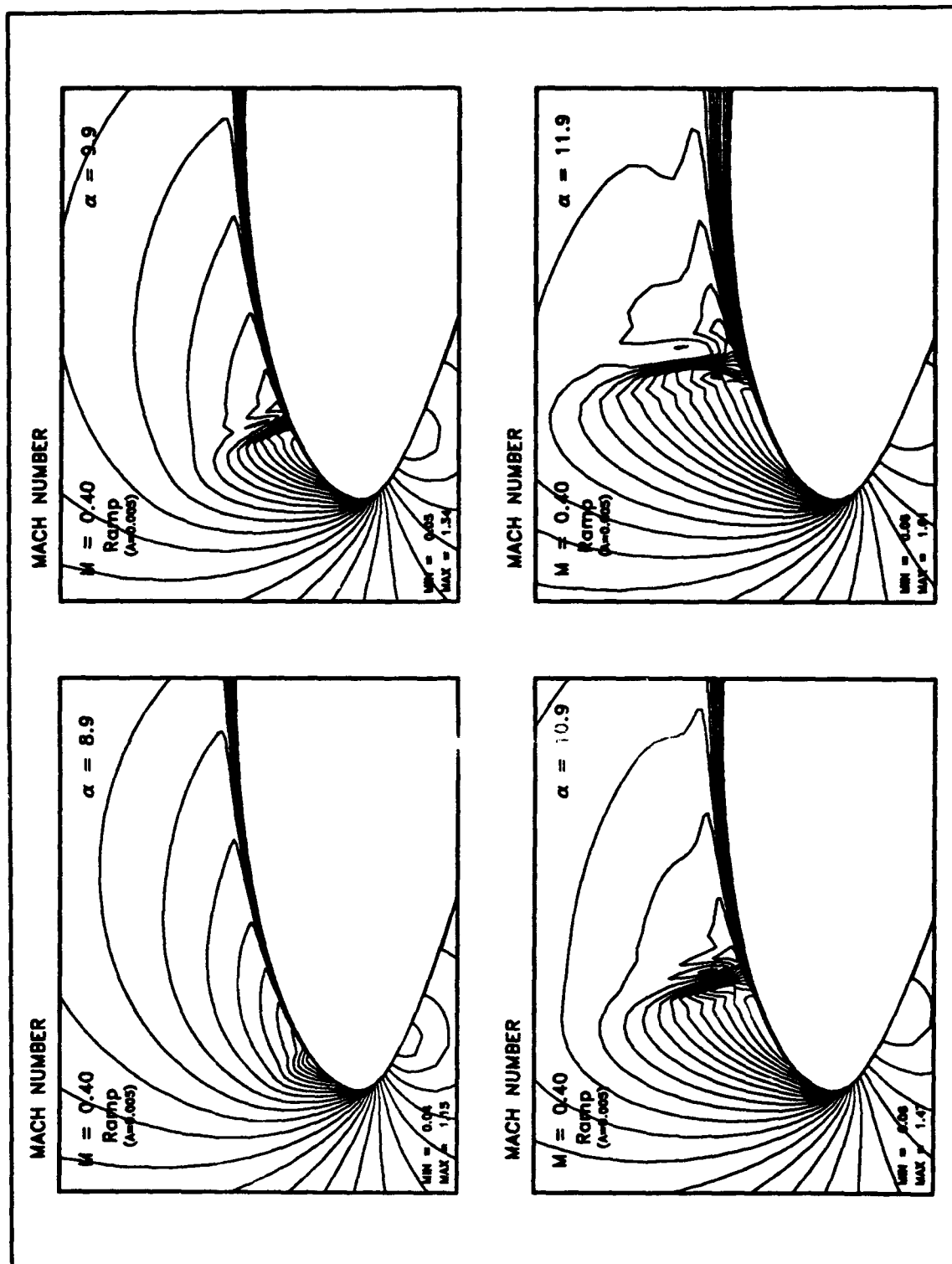


Figure 5.59
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

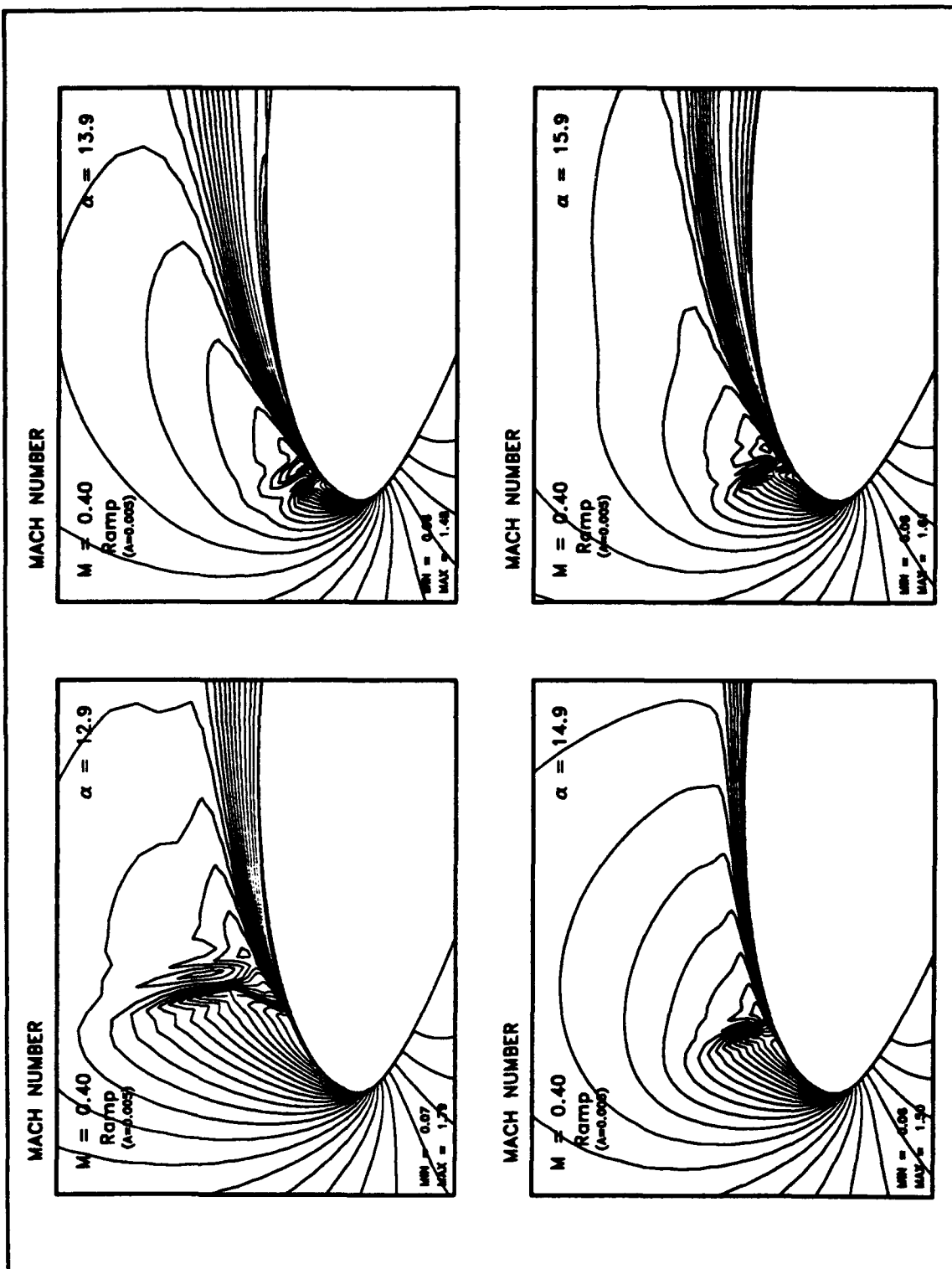


Figure 5.60
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

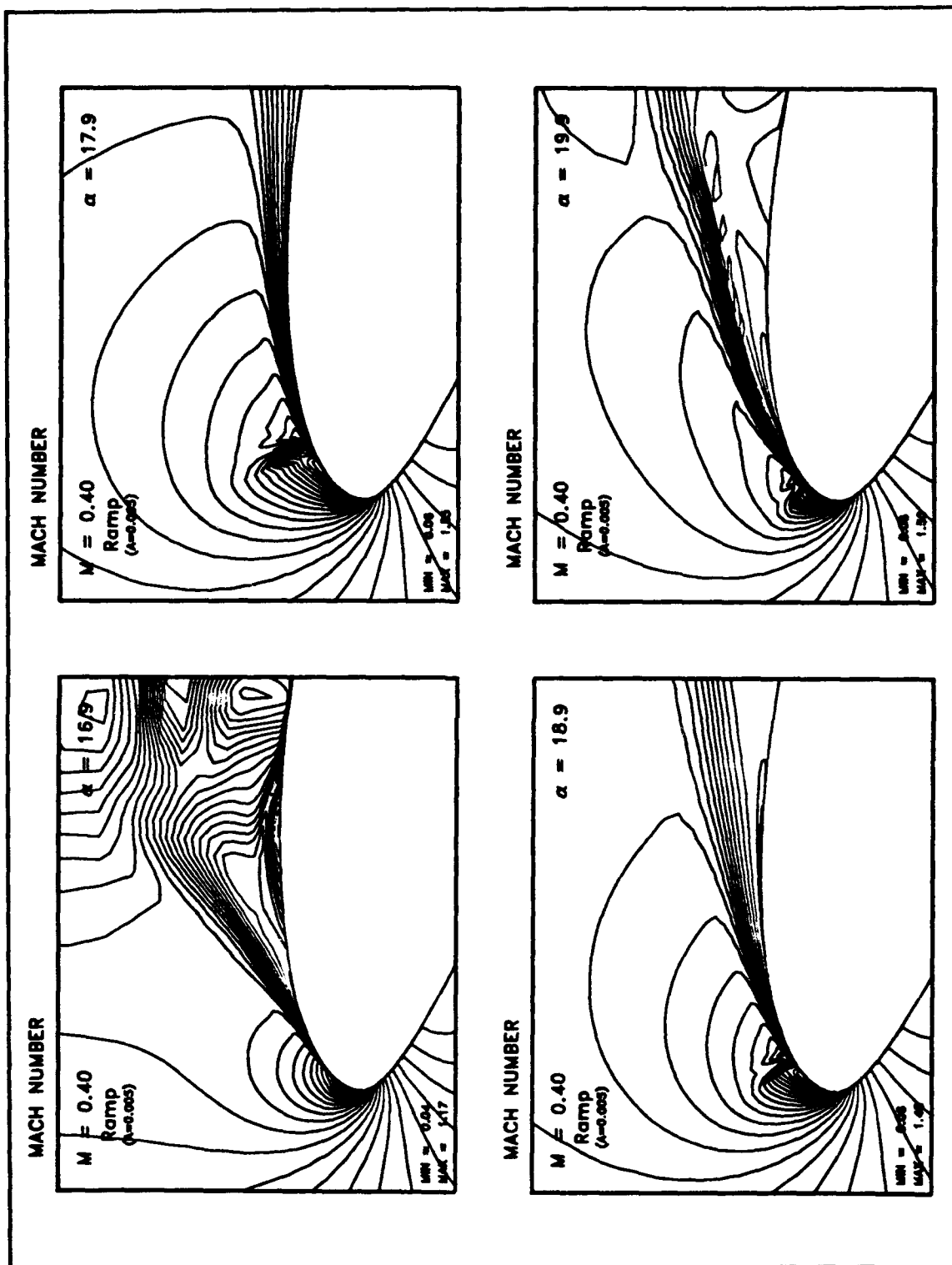


Figure 5.61
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

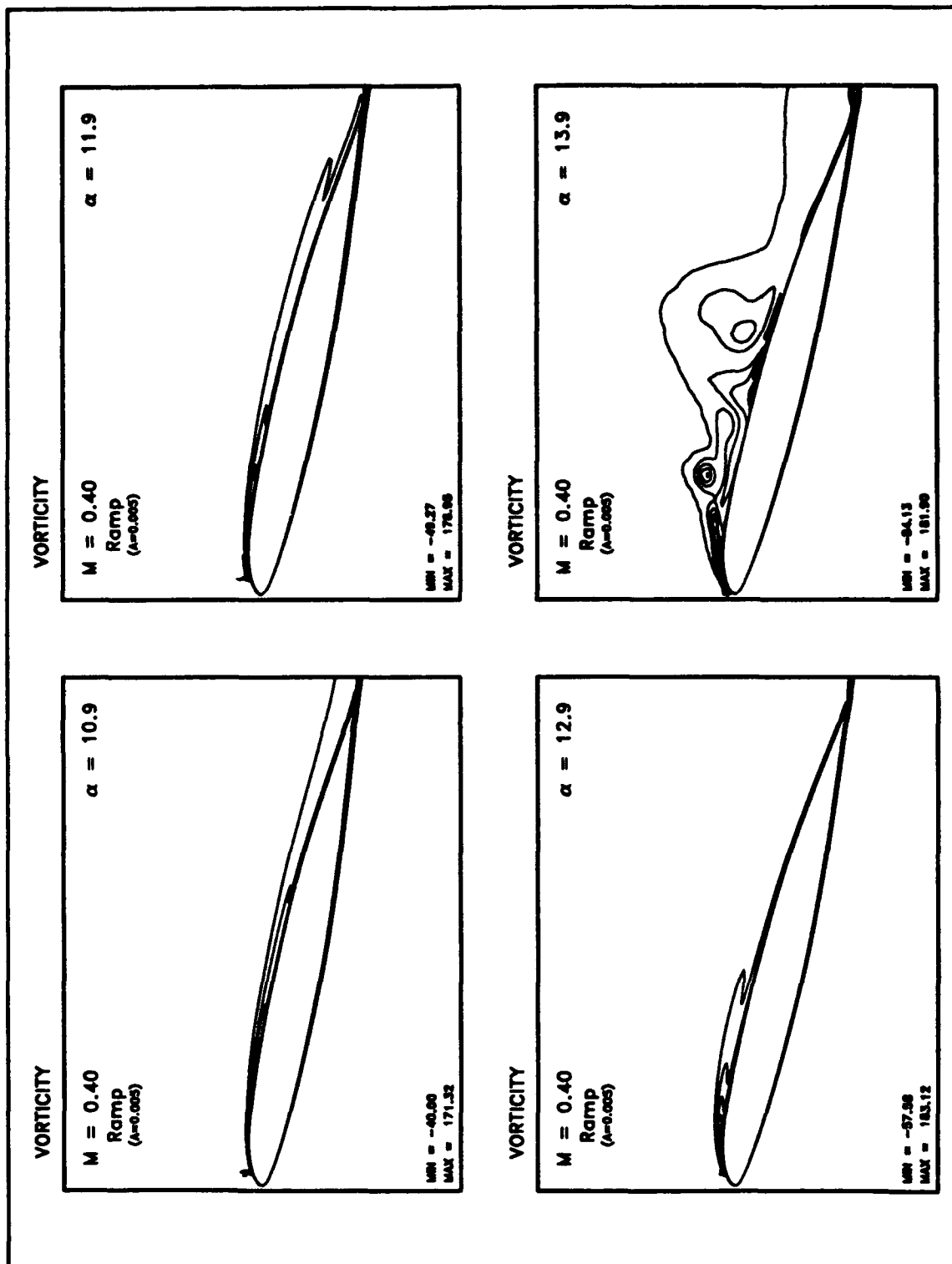


Figure 5.62
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

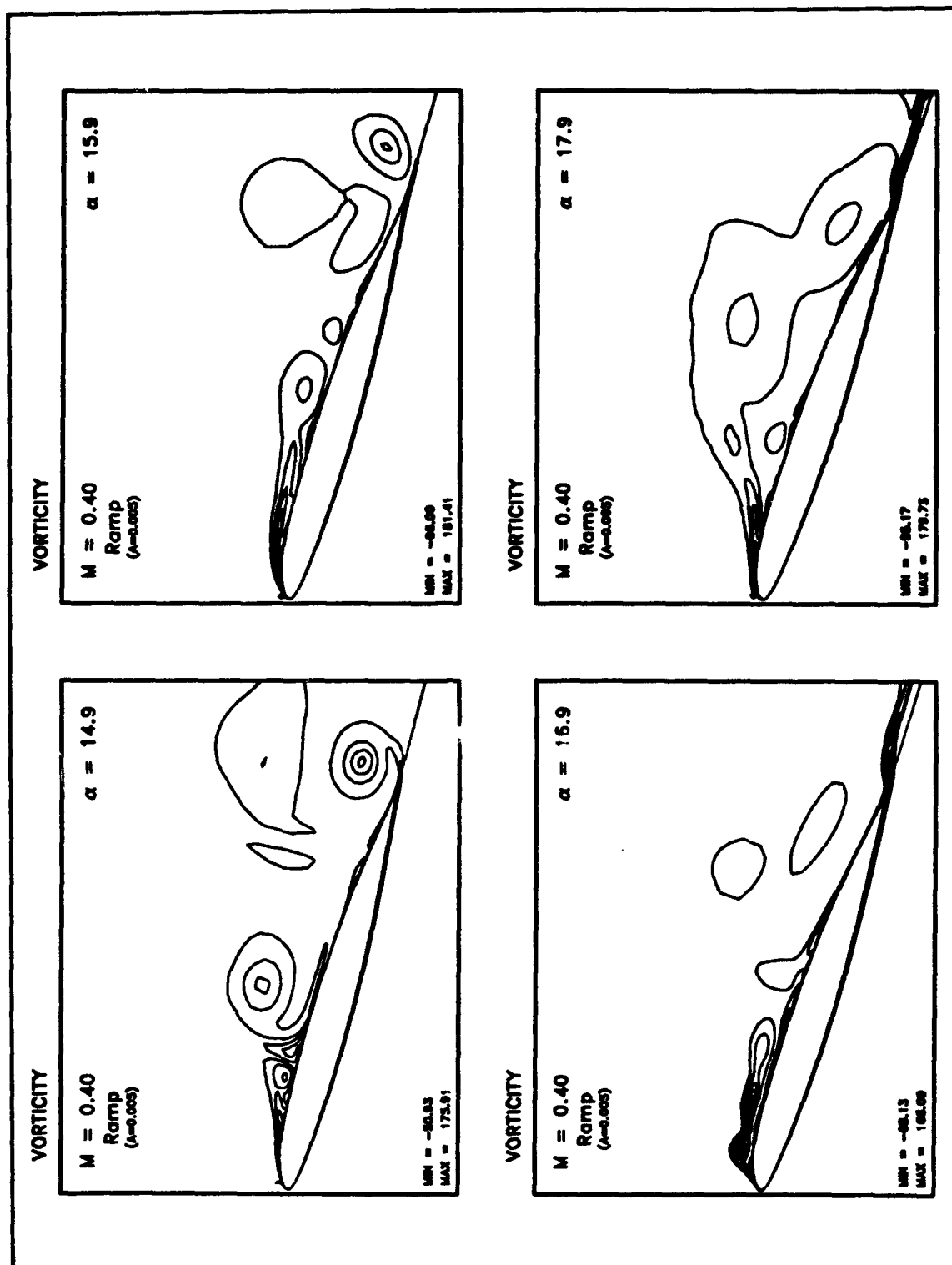


Figure 5.63
Sikorsky SSC-A09
Ramp ($A=0.005$, $M=0.4$)

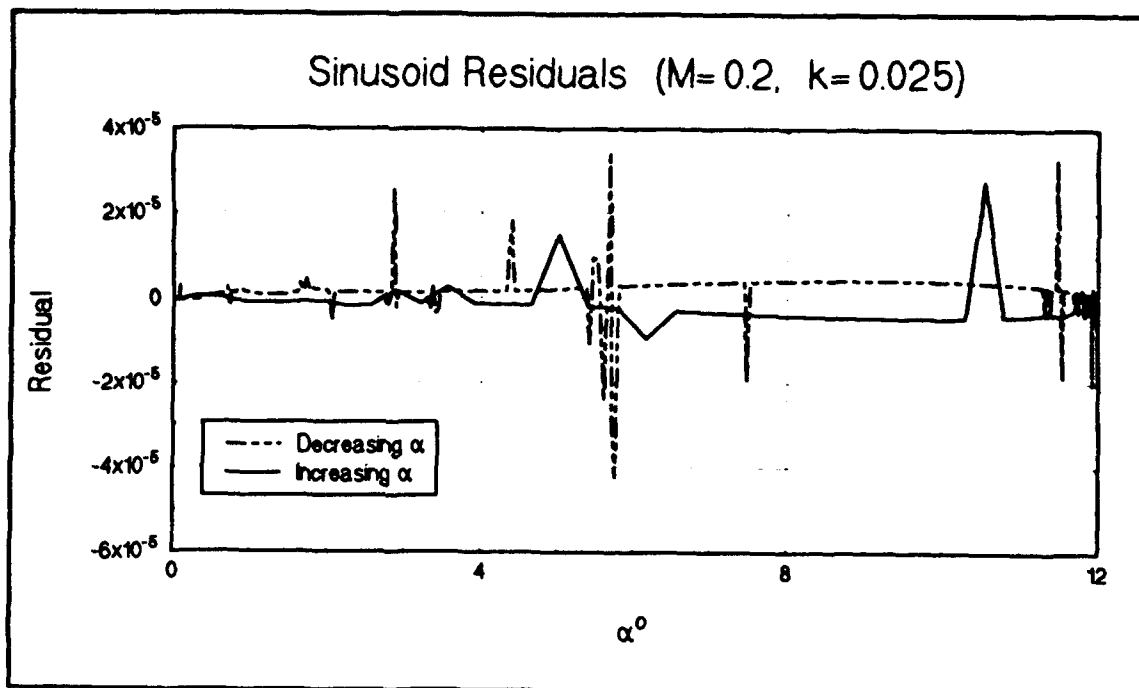


Figure 5.64
Sinusoid Residuals ($k=0.025$, $M=0.2$)

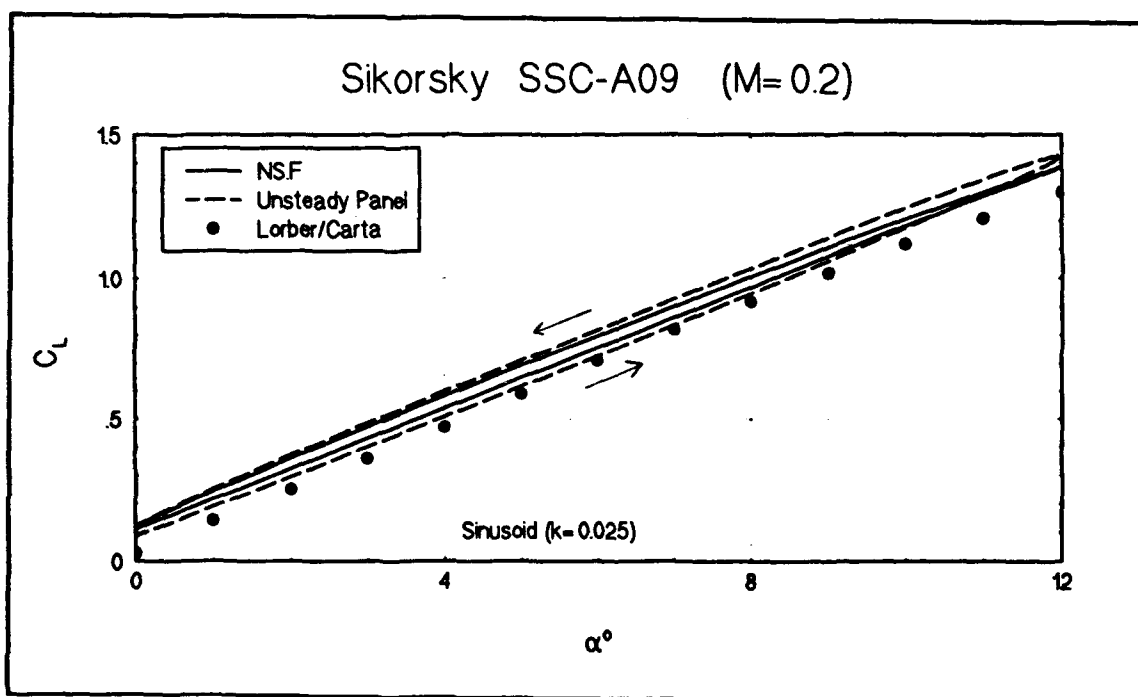


Figure 5.65
Sinusoid C_{Ls} ($k=0.025$, $M=0.2$)

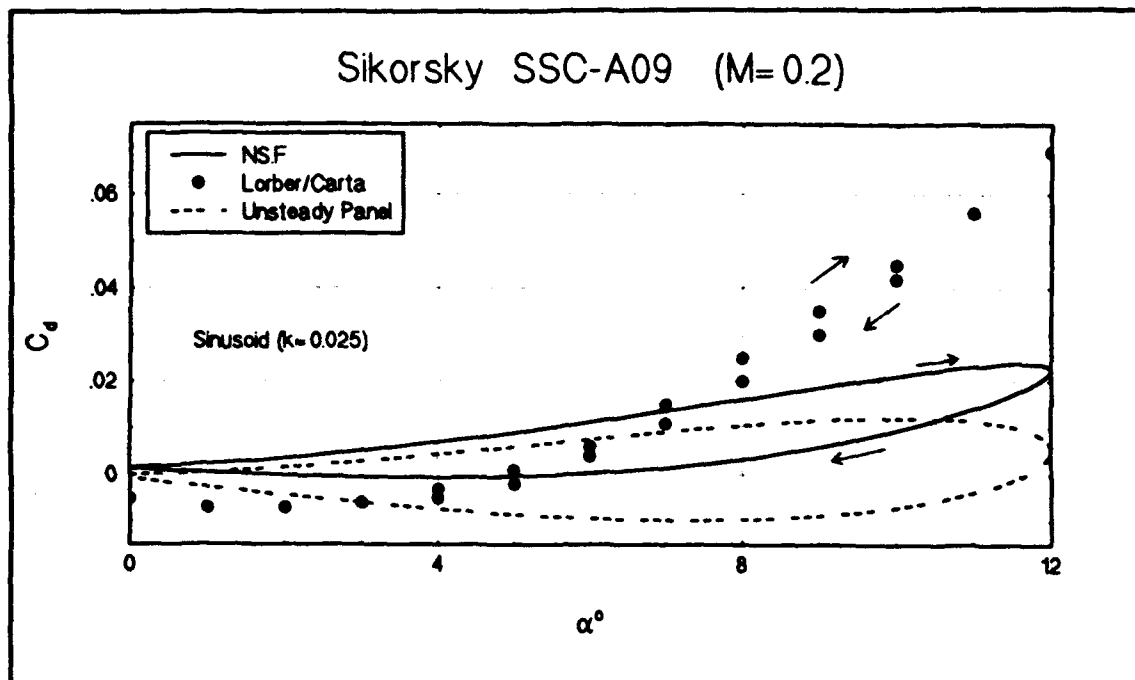


Figure 5.66
Sinusoid C_{da} ($k=0.025$, $M=0.2$)

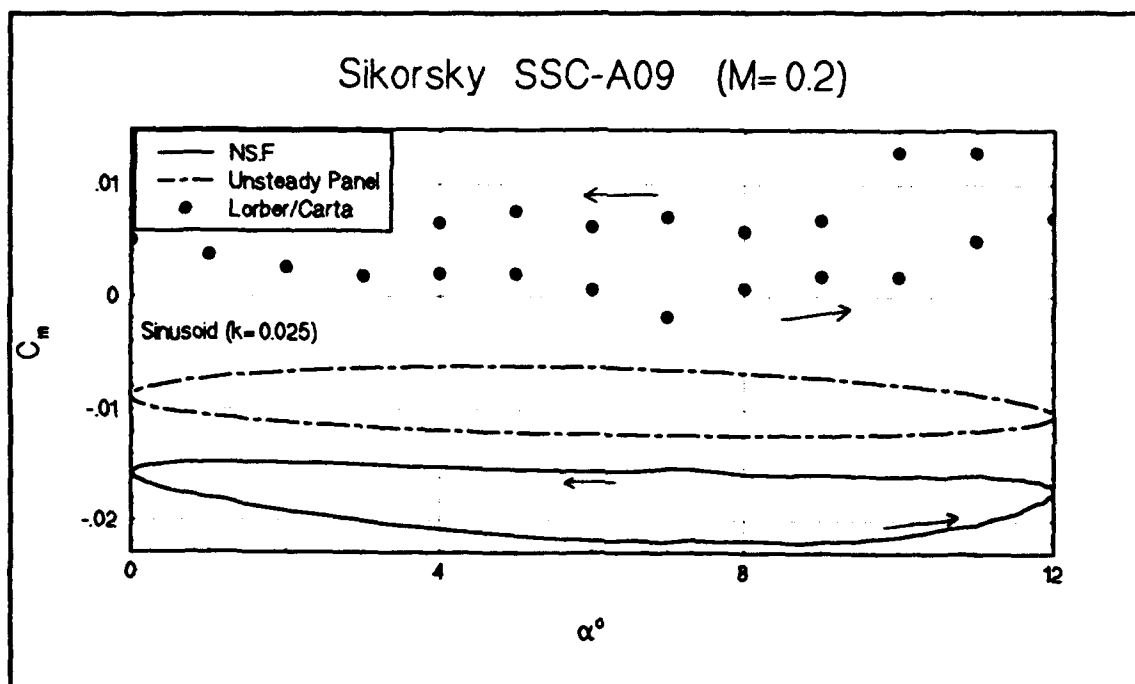


Figure 5.67
Sinusoid C_{He} ($k=0.025$, $M=0.2$)

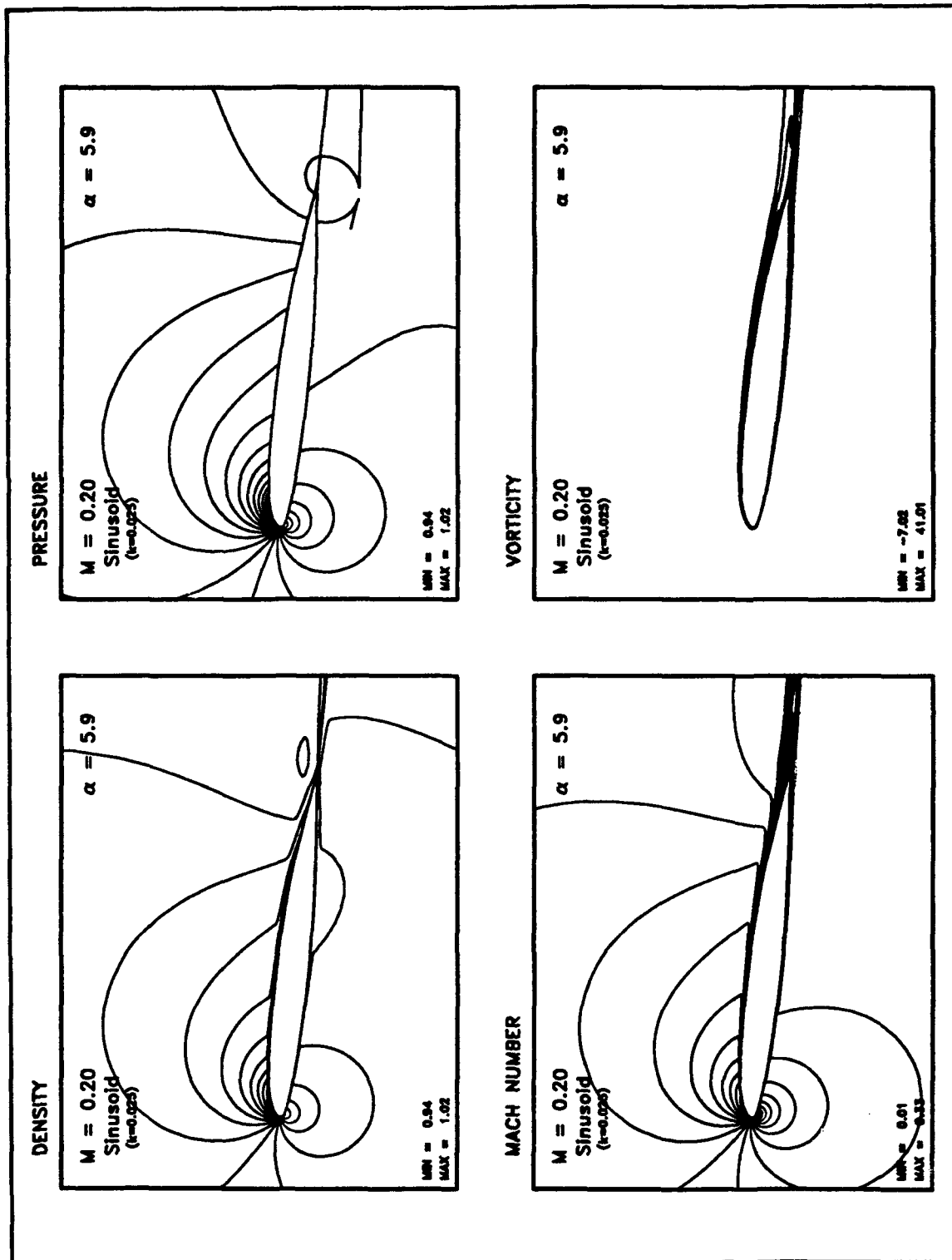


Figure 5.68
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.025$, $M=0.2$)

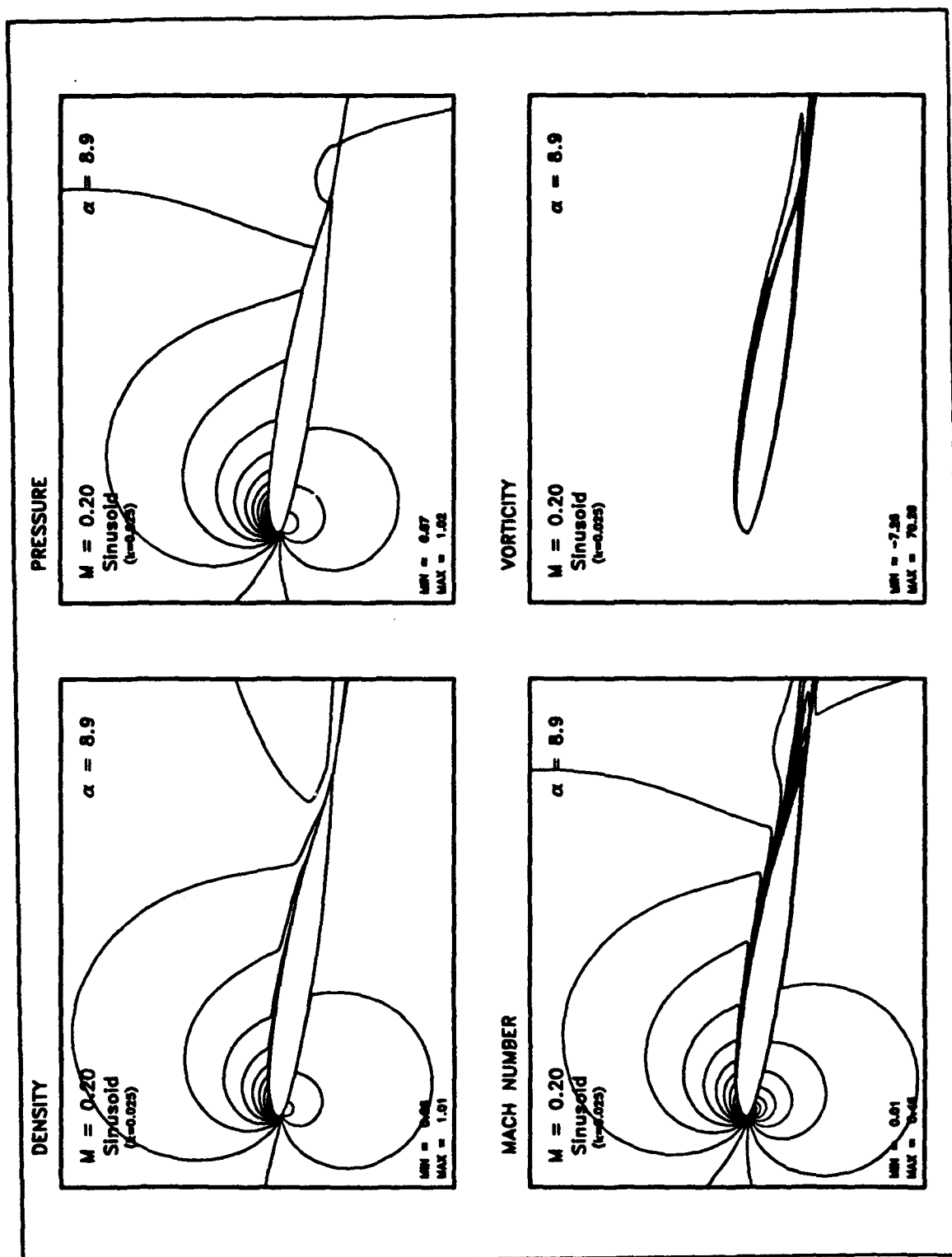


Figure 5.69
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.025$, $M=0.2$)

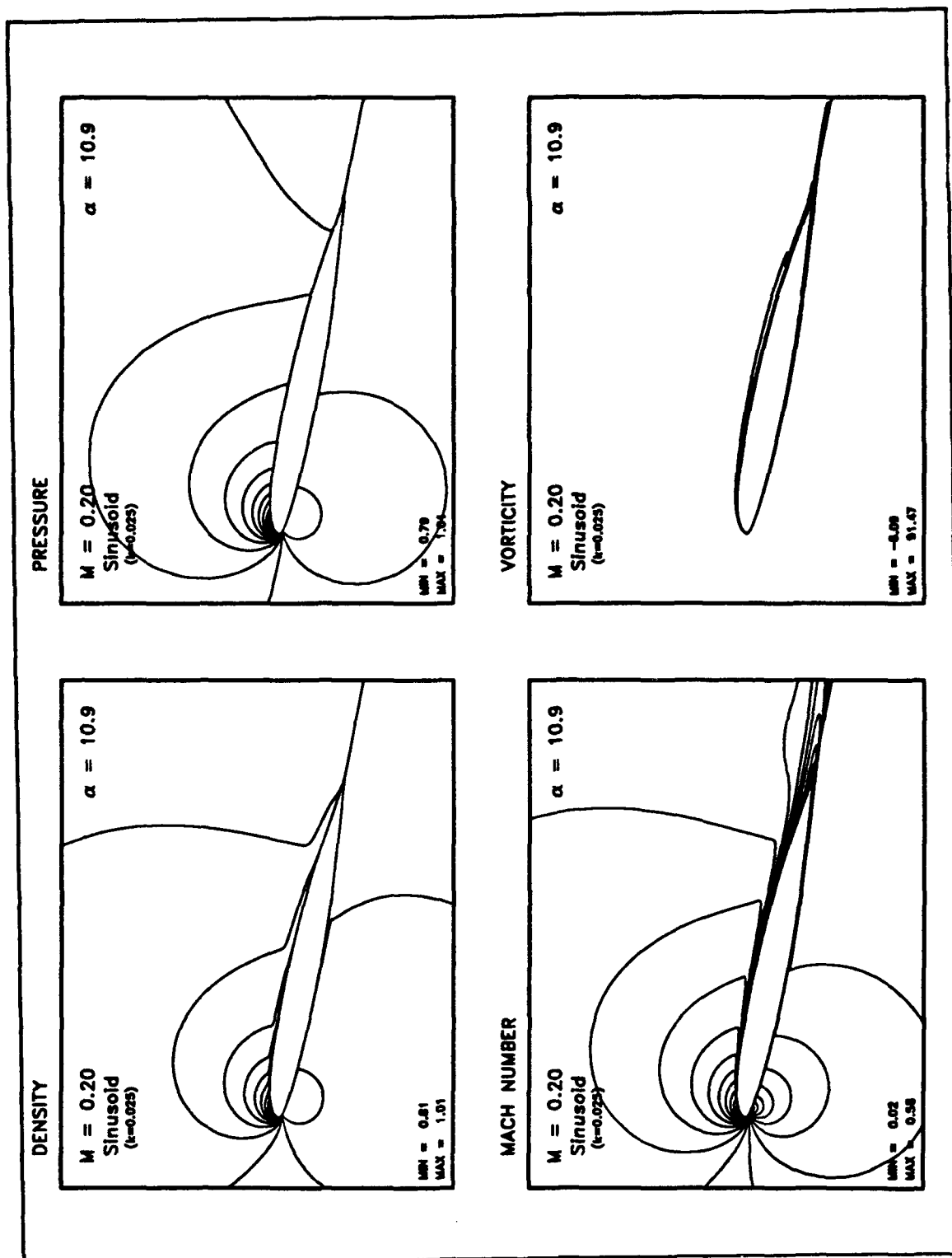


Figure 5.70
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.025$, $M=0.2$)

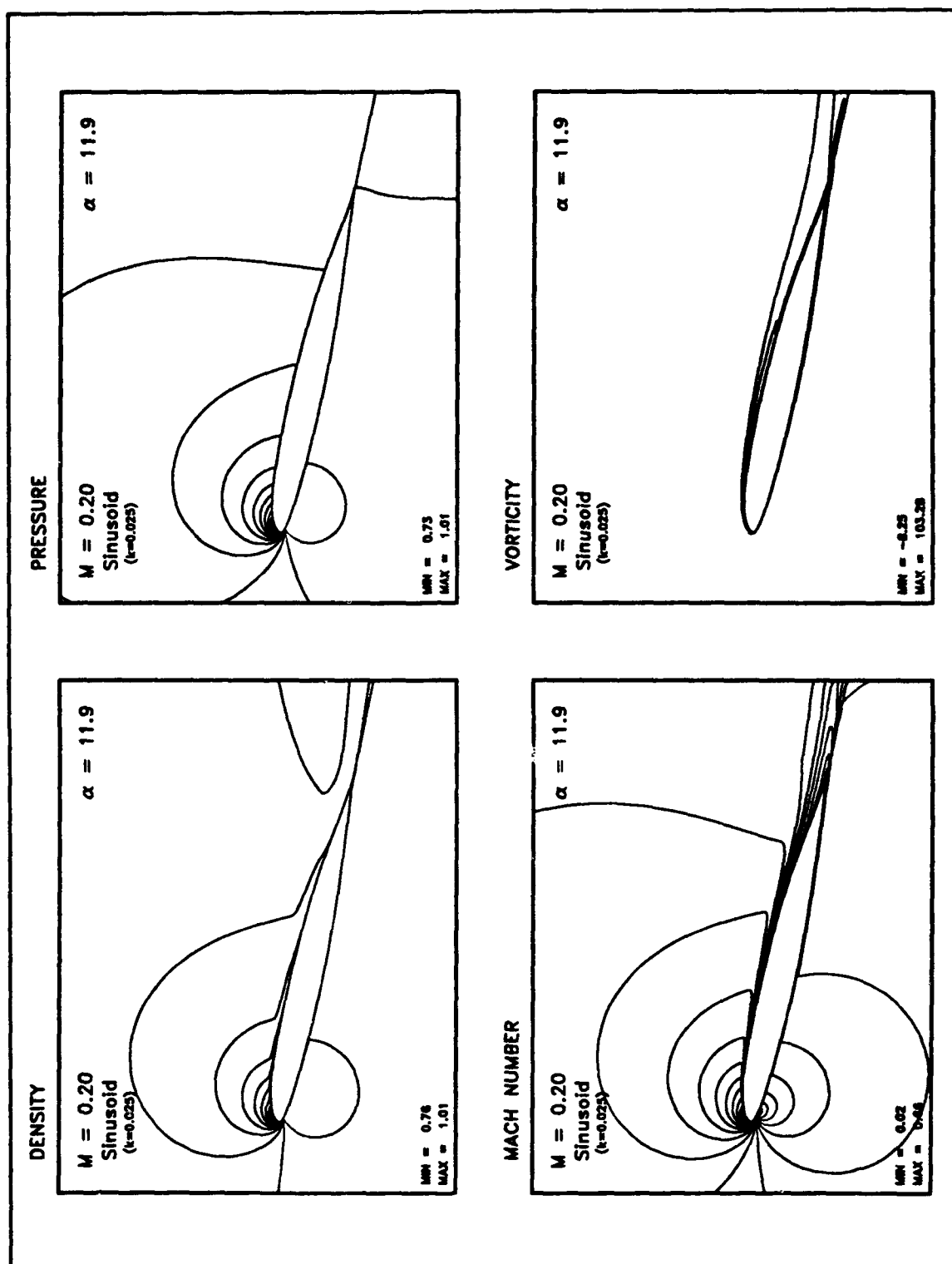


Figure 5.71
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.025$, $M=0.2$)

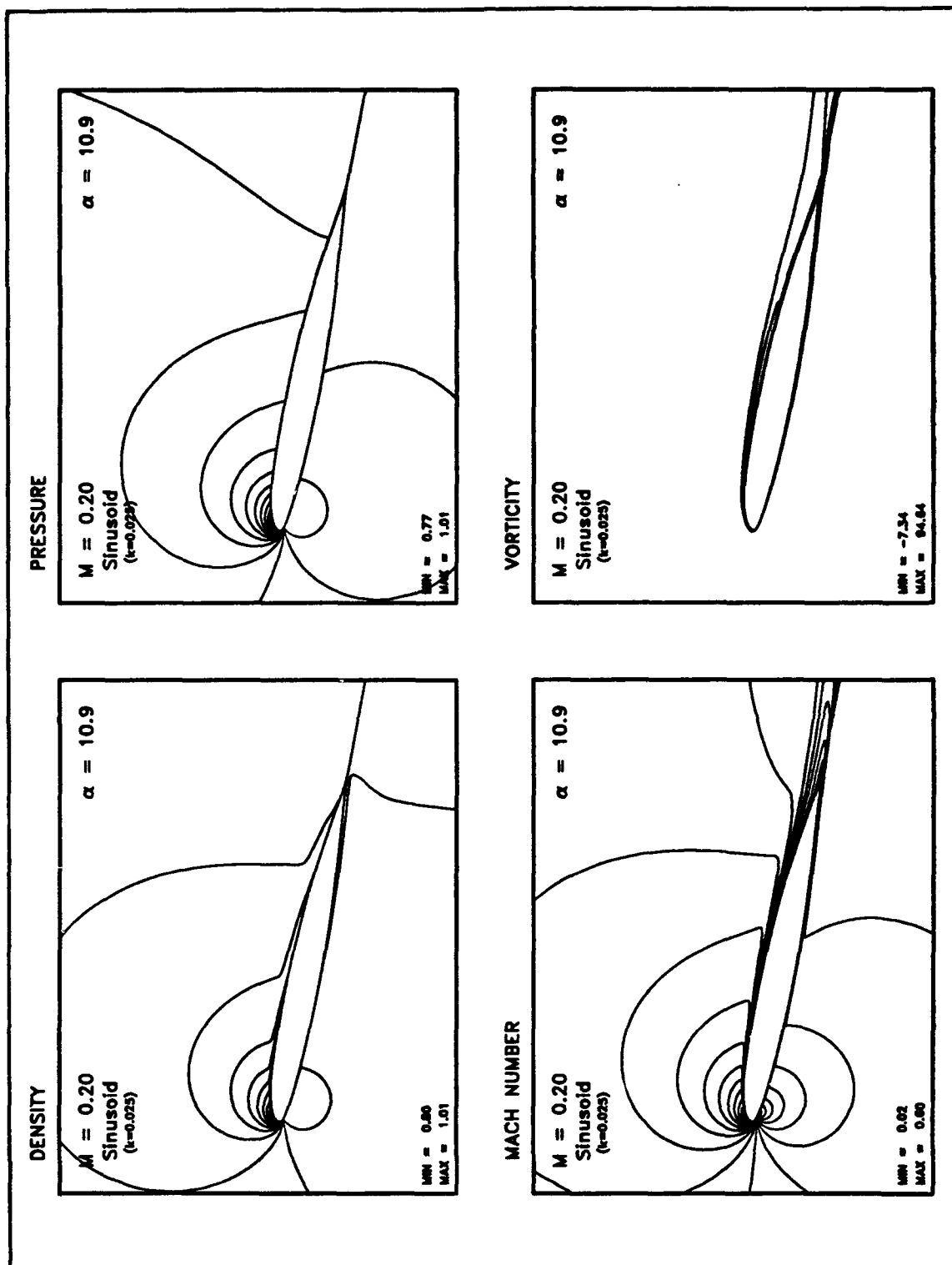


Figure 5.72
Sikorsky SSC-A09
Sinusoid (DOWN) ($k=0.025$, $M=0.2$)

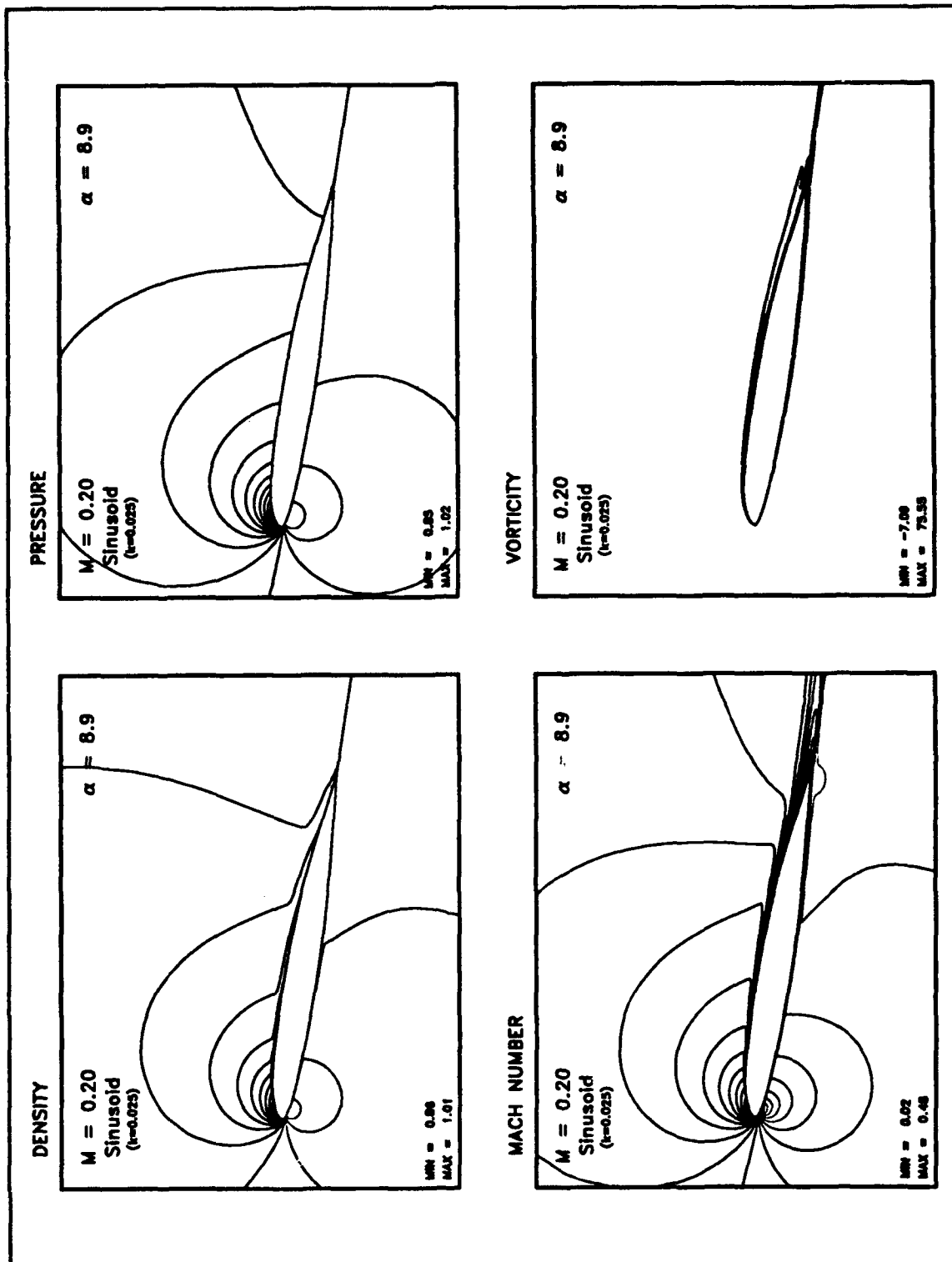


Figure 5.73
Sikorsky SSC-A09
Sinusoid (DOWN) ($k=0.025$, $M=0.2$)

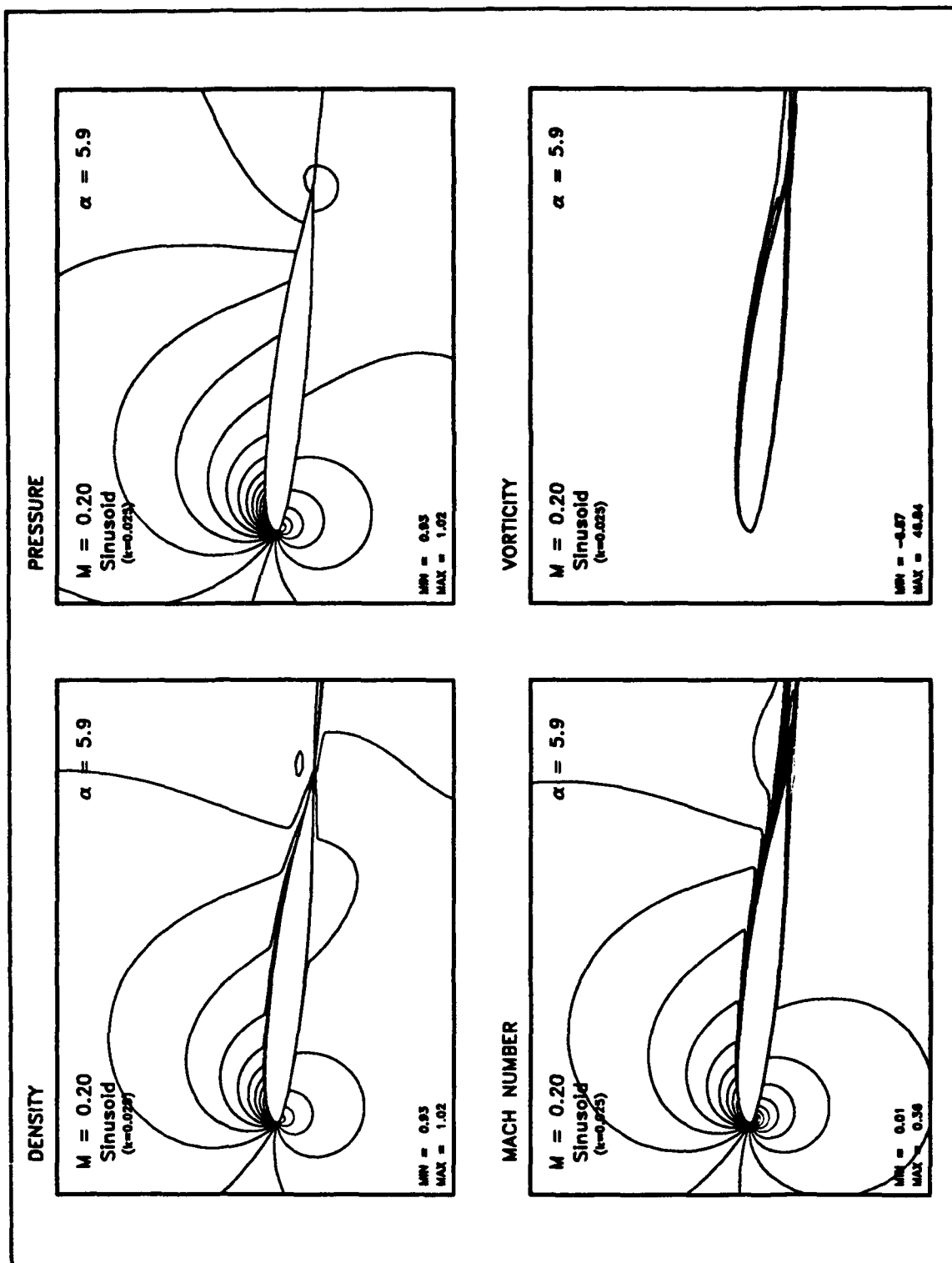


Figure 5.74
Sikorsky SSC-A09
Sinusoid (DOWN) ($k=0.025$, $M=0.2$)

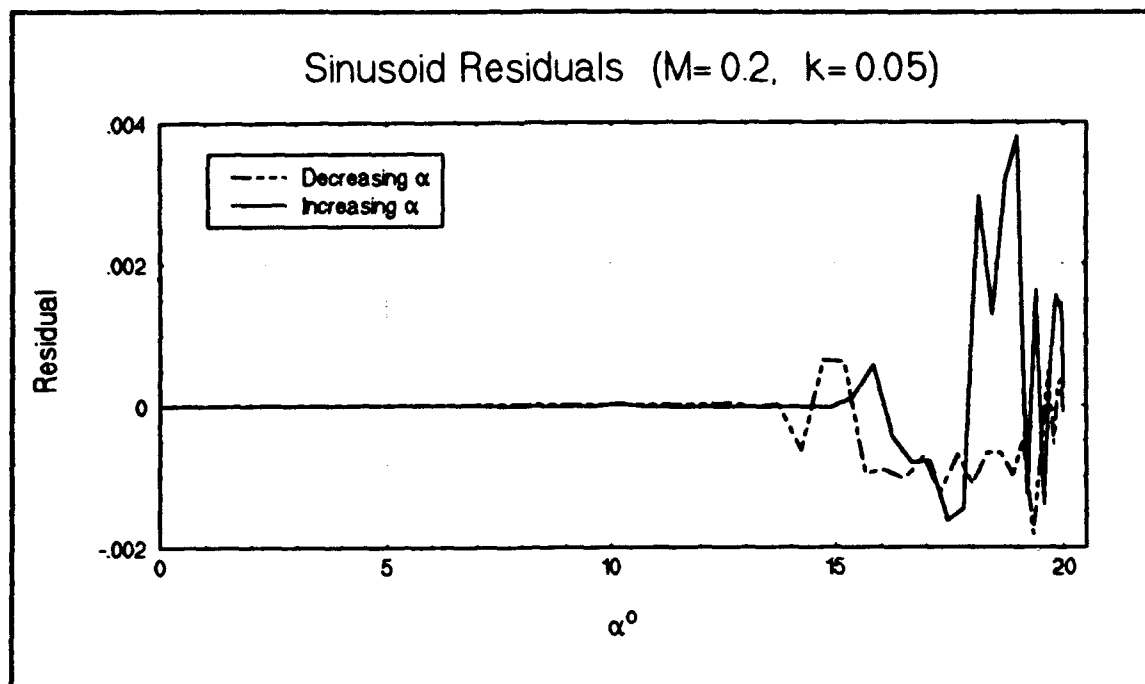


Figure 5.75
Sinusoid Residuals ($k=0.05$, $M=0.2$)

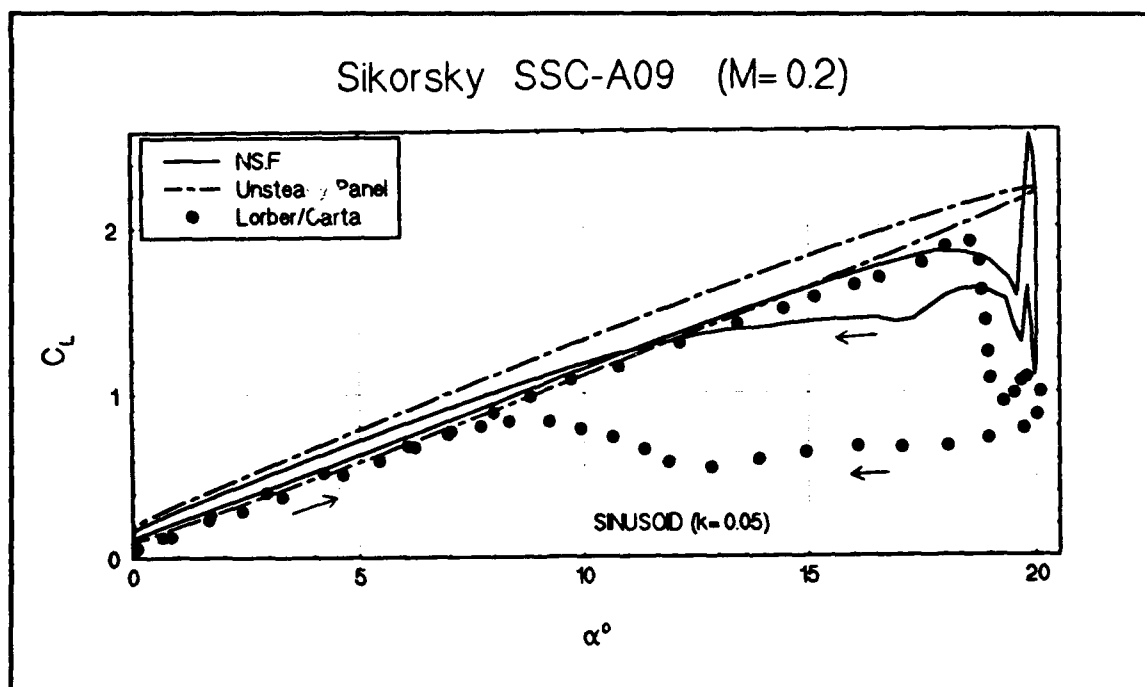


Figure 5.76
Sinusoid $C_{L\alpha}$ ($k=0.05$, $M=0.2$)

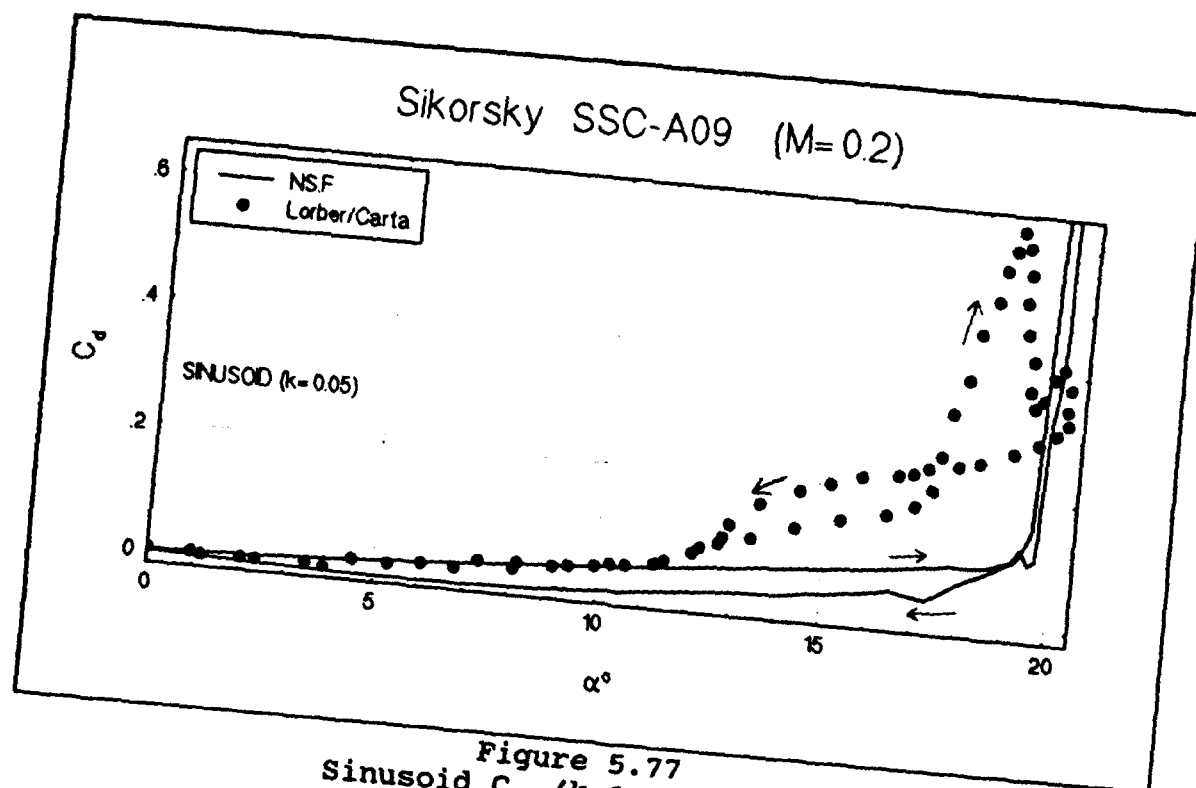


Figure 5.77
Sinusoid C_{da} ($k=0.05$, $M=0.2$)

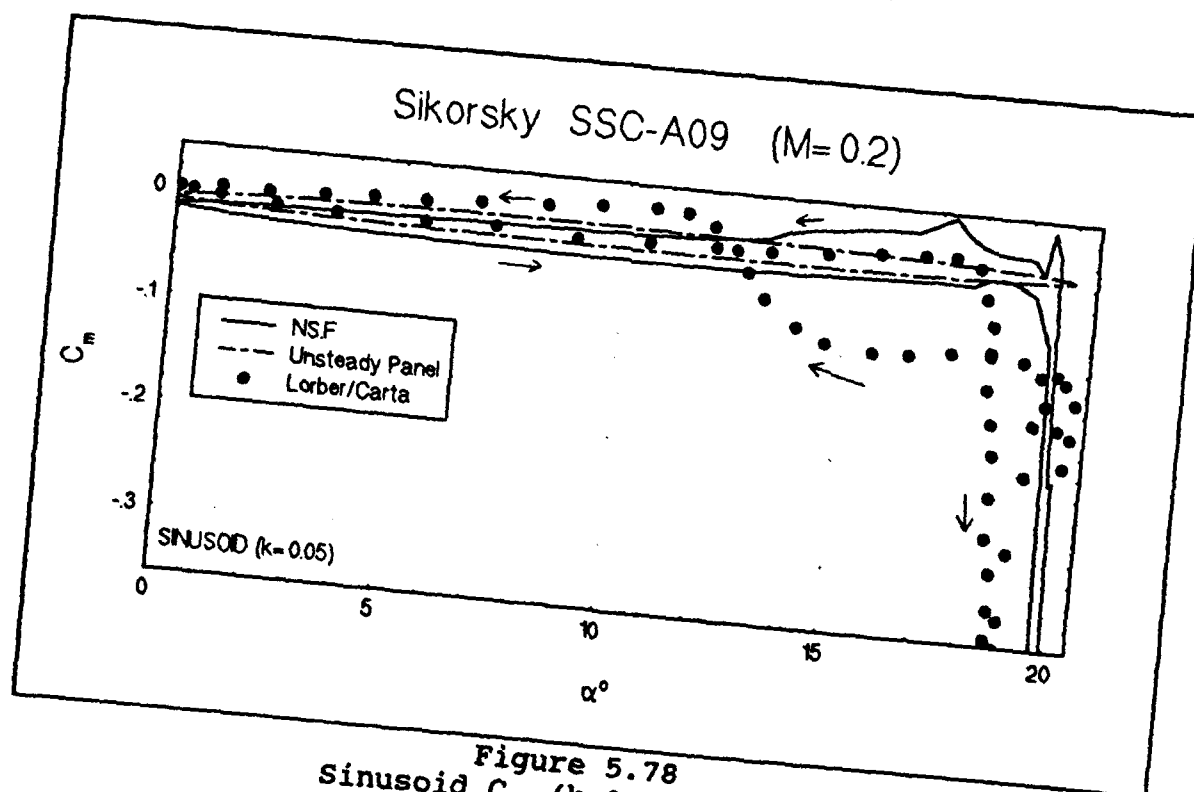


Figure 5.78
Sinusoid C_{Ma} ($k=0.05$, $M=0.2$)

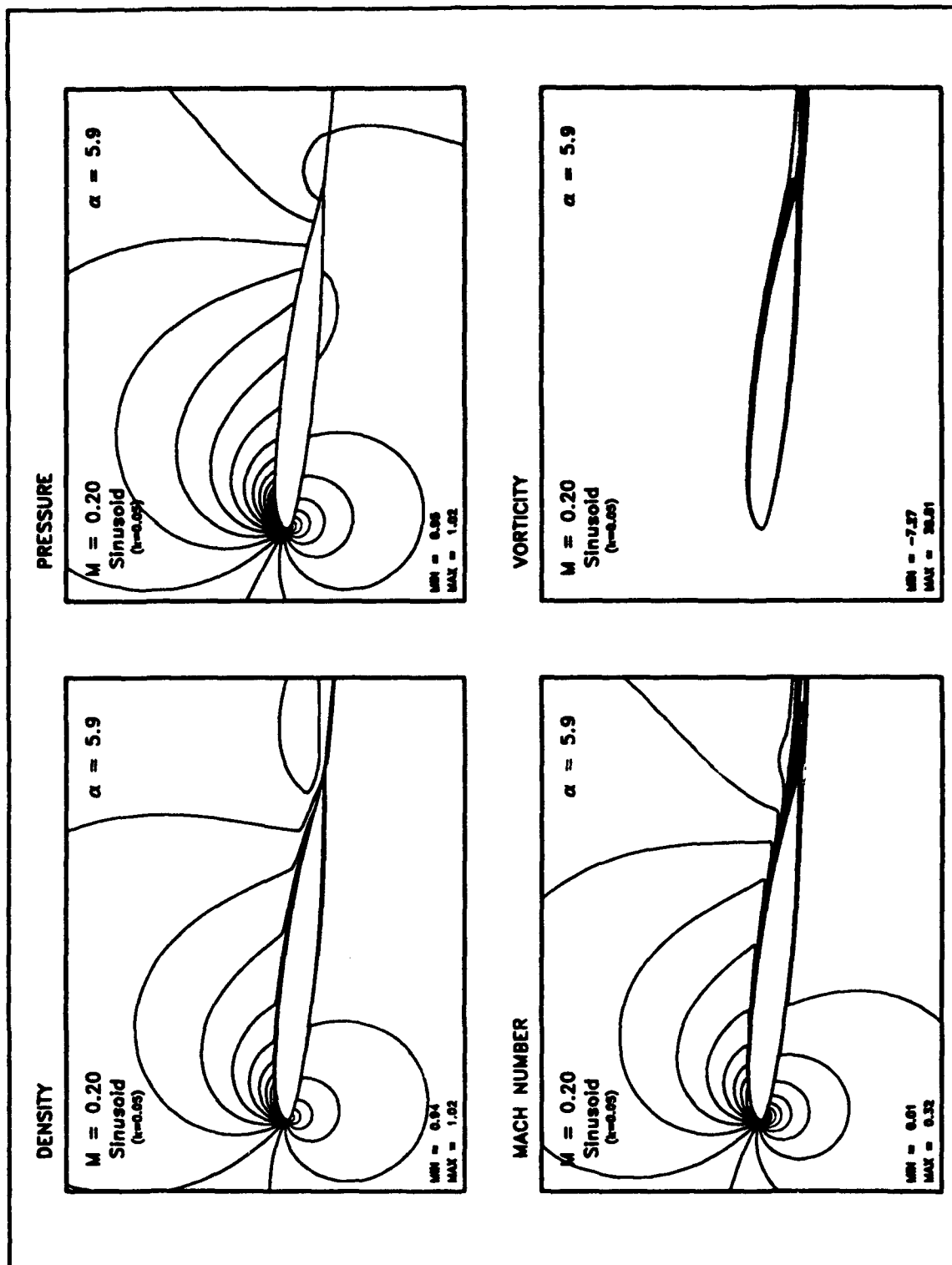


Figure 5.79
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.05$, $M=0.2$)

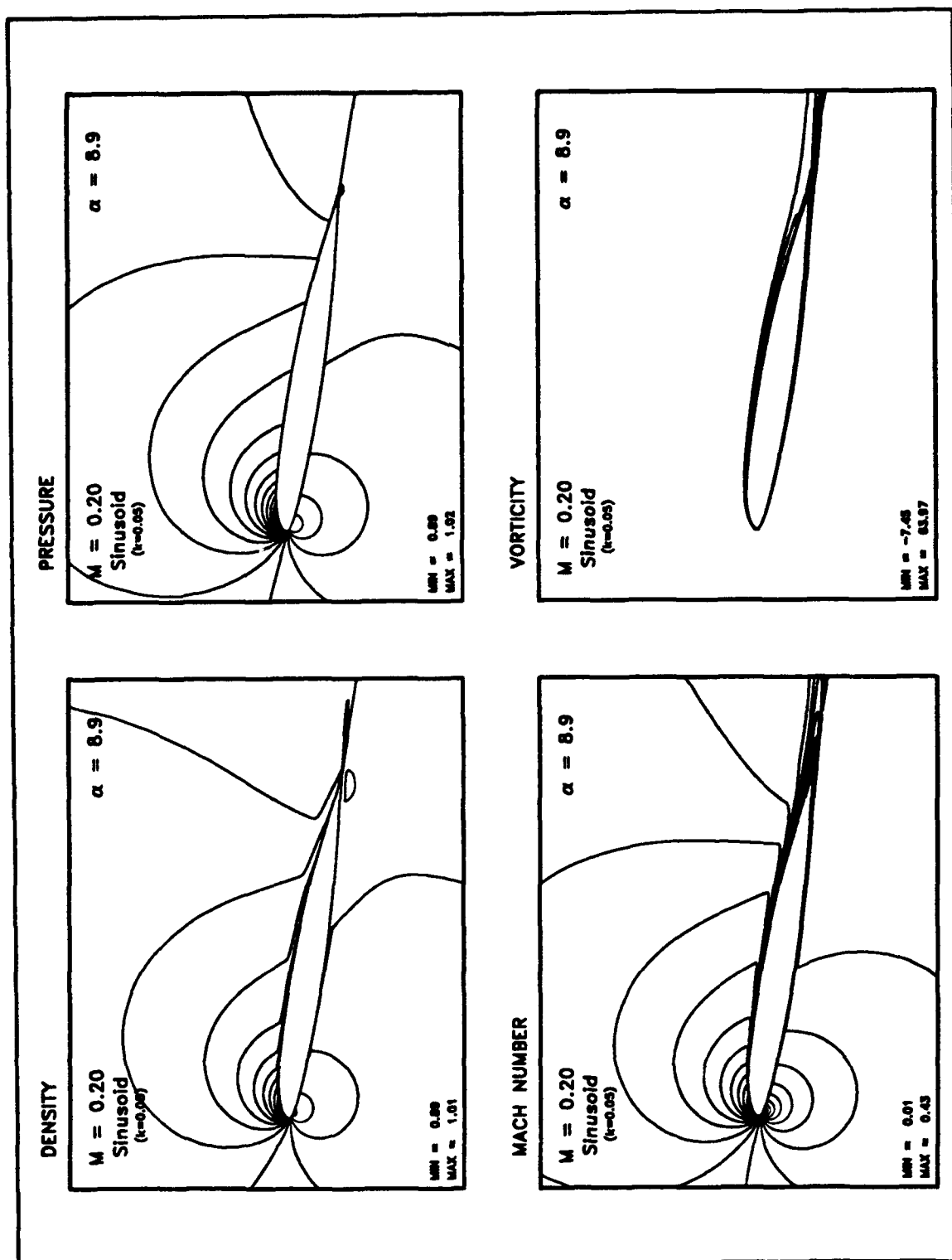


Figure 5.80
Sikorsky SSC-A09
sinusoid (UP) ($k=0.05$, $M=0.2$)

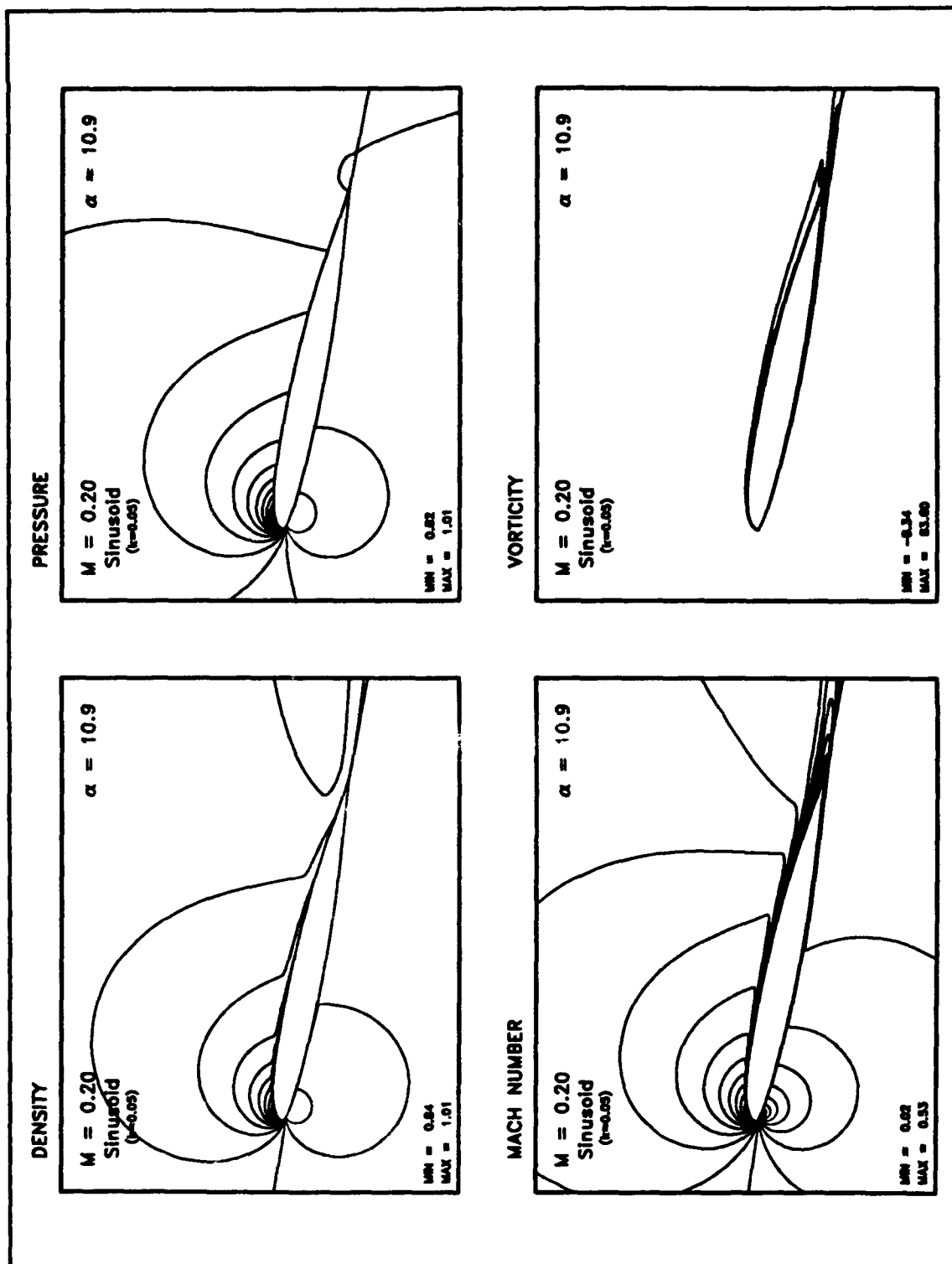


Figure 5.81
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.05$, $M=0.2$)

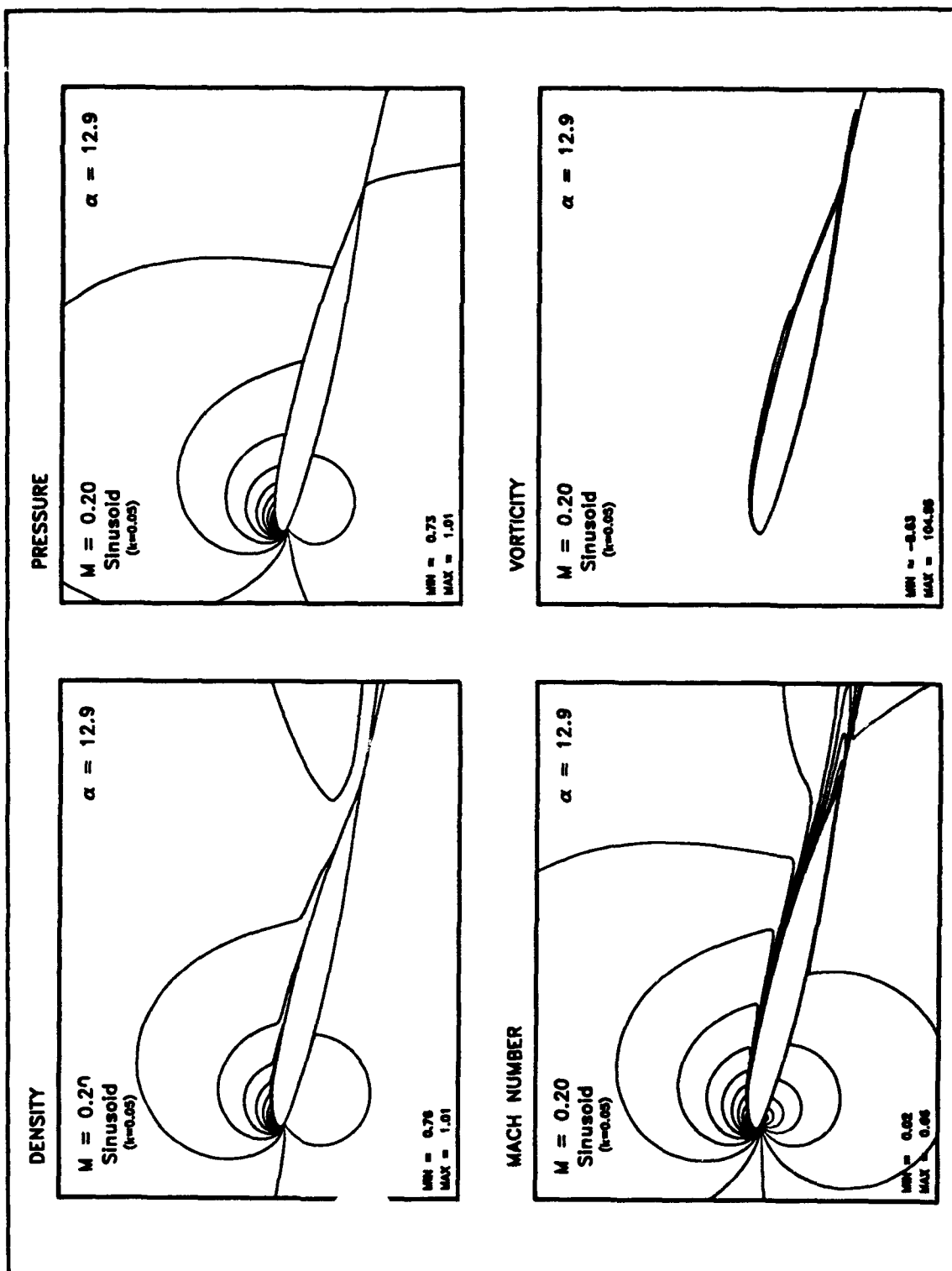


Figure 5.82
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.05$, $M=0.2$)

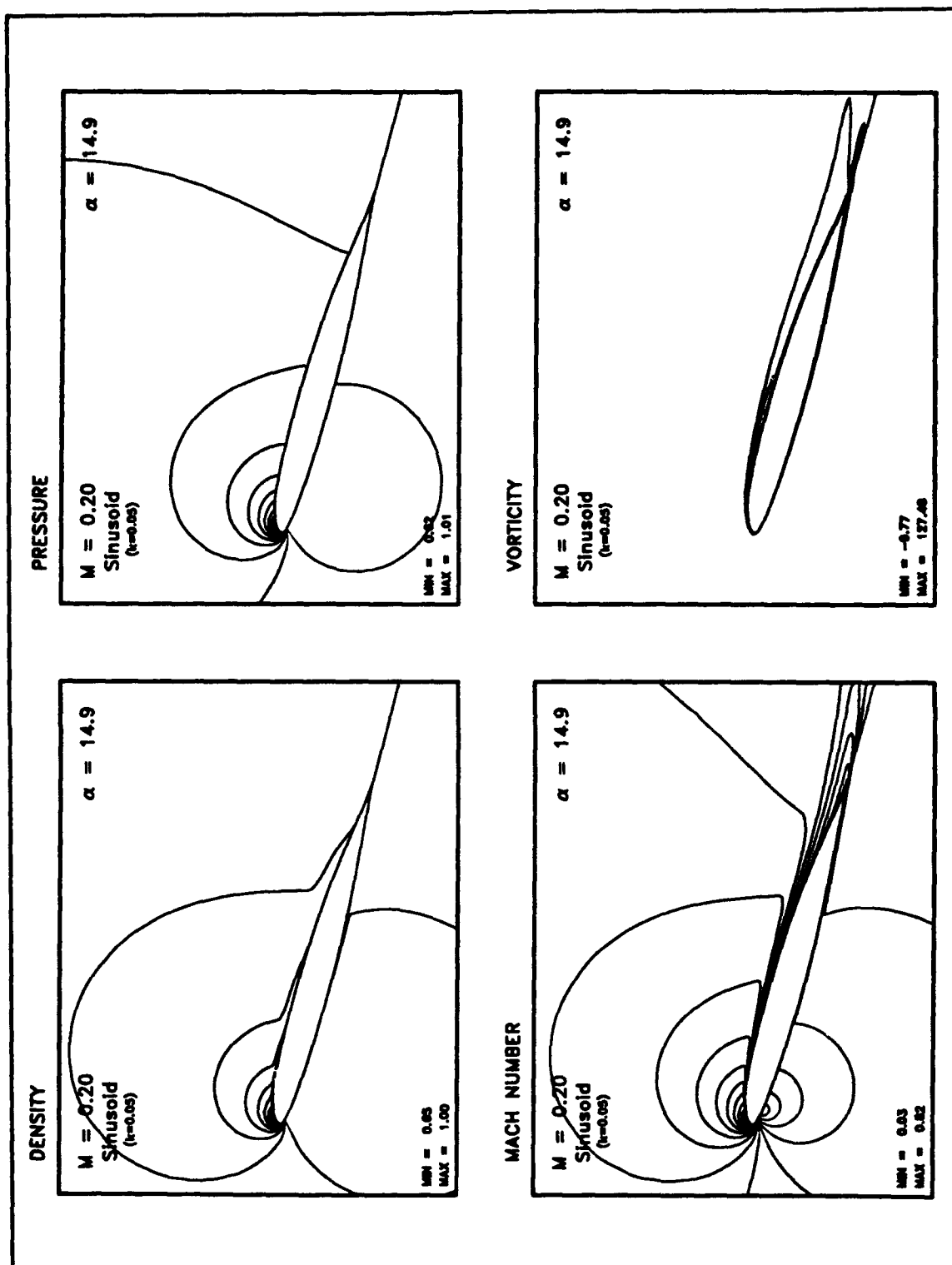


Figure 5.83
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.05$, $M=0.2$)

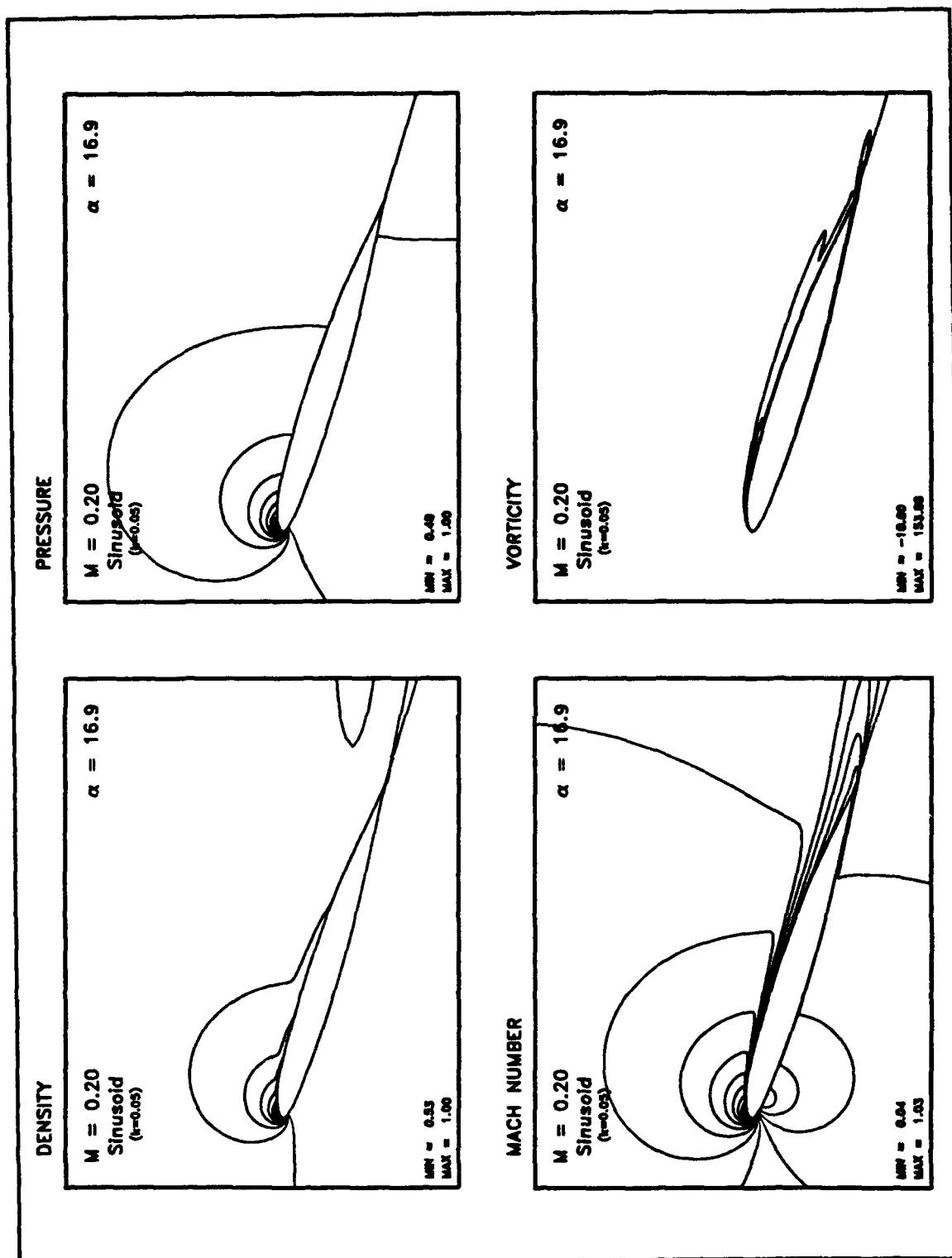


Figure 5.84
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.05$, $M=0.2$)

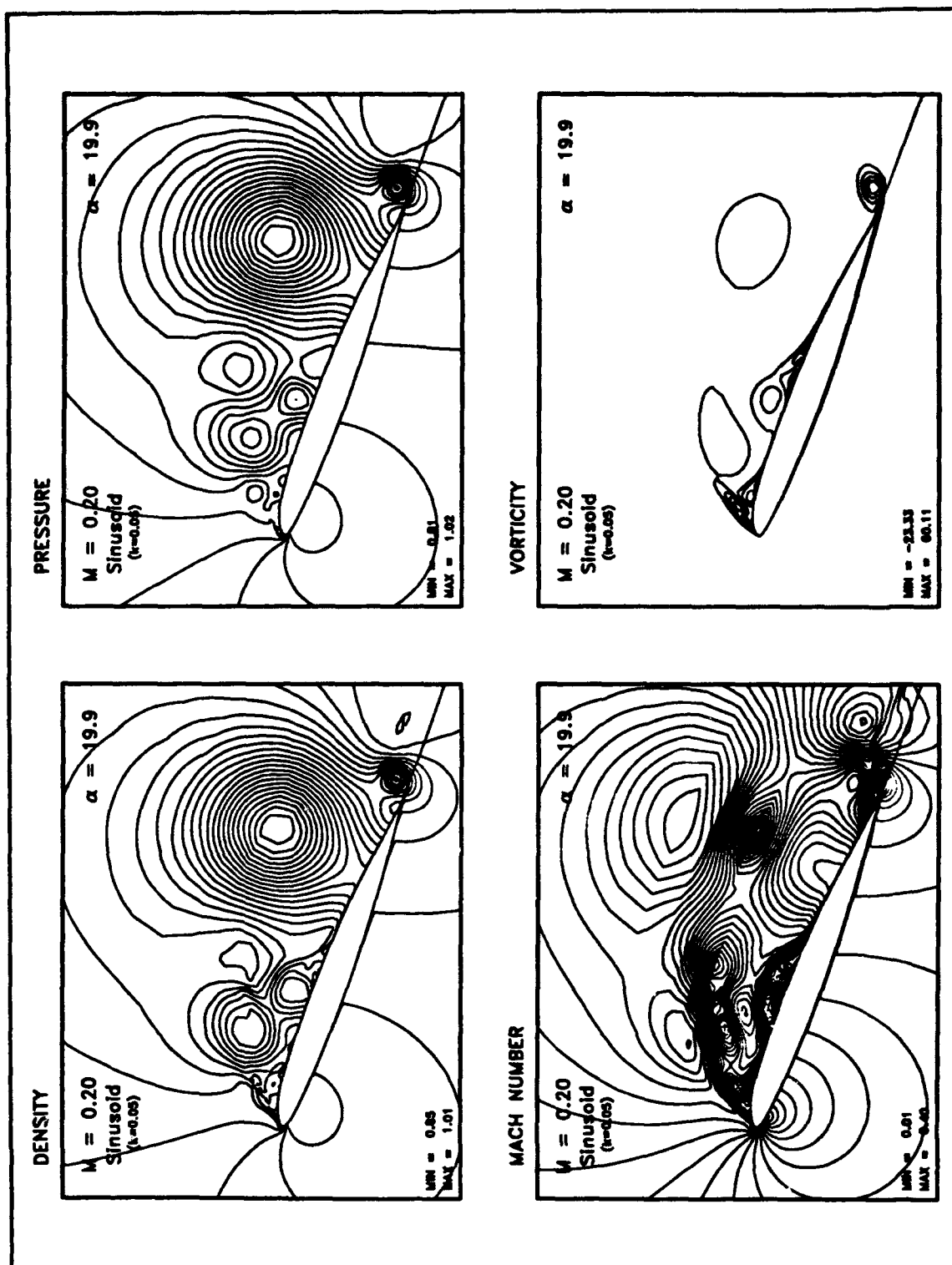


Figure 5.85
Sikorsky SSC-A09
Sinusoid (UP) ($k=0.05$, $M=0.2$)

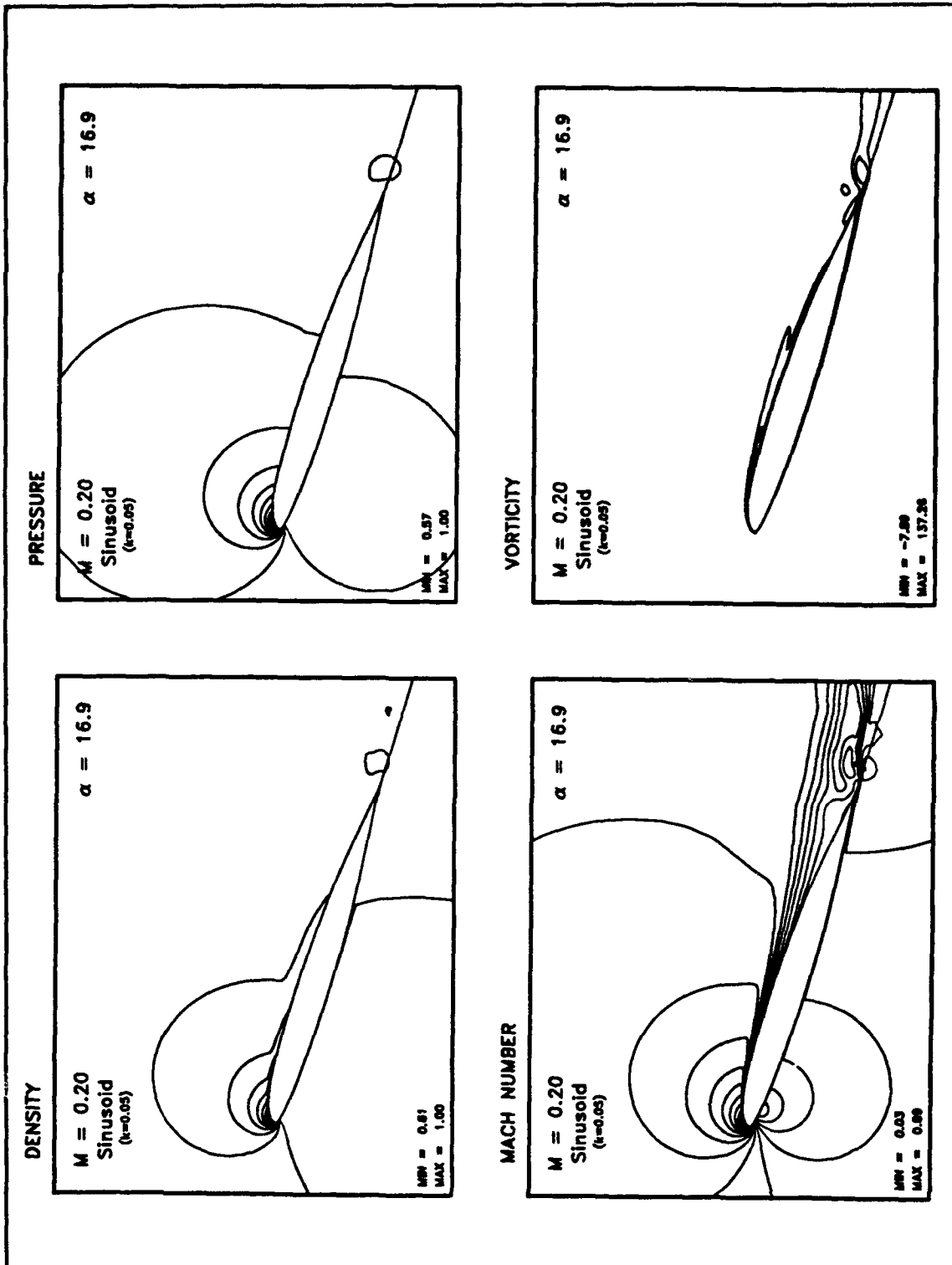


Figure 5.86
Sikorsky SSC-A09
Sinusoid (DOWN) ($k=0.05$, $M=0.2$)

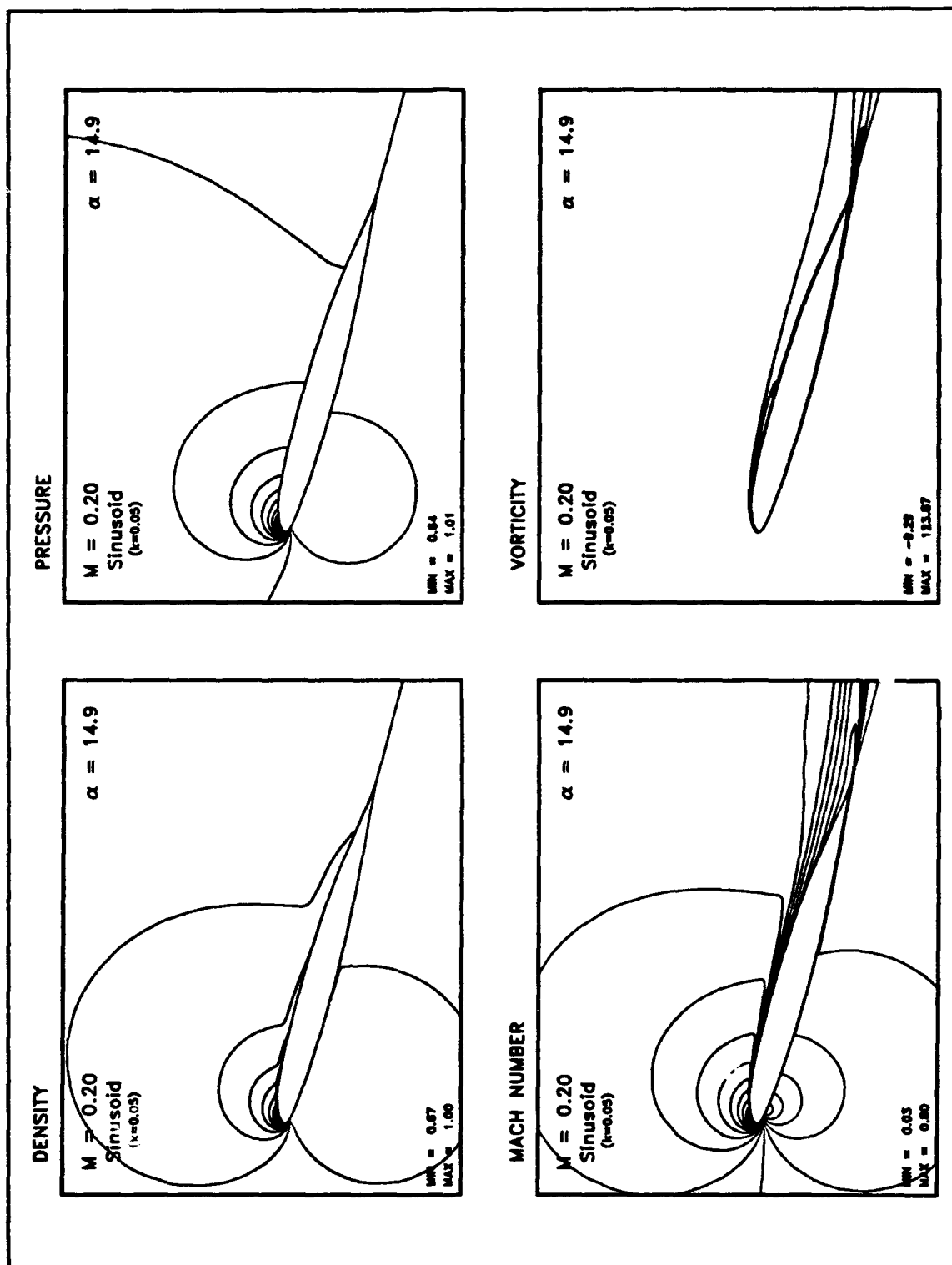


Figure 5.87
Sikorsky SSC-A09
Sinusoid (DOWN) ($k=0.05$, $M=0.2$)

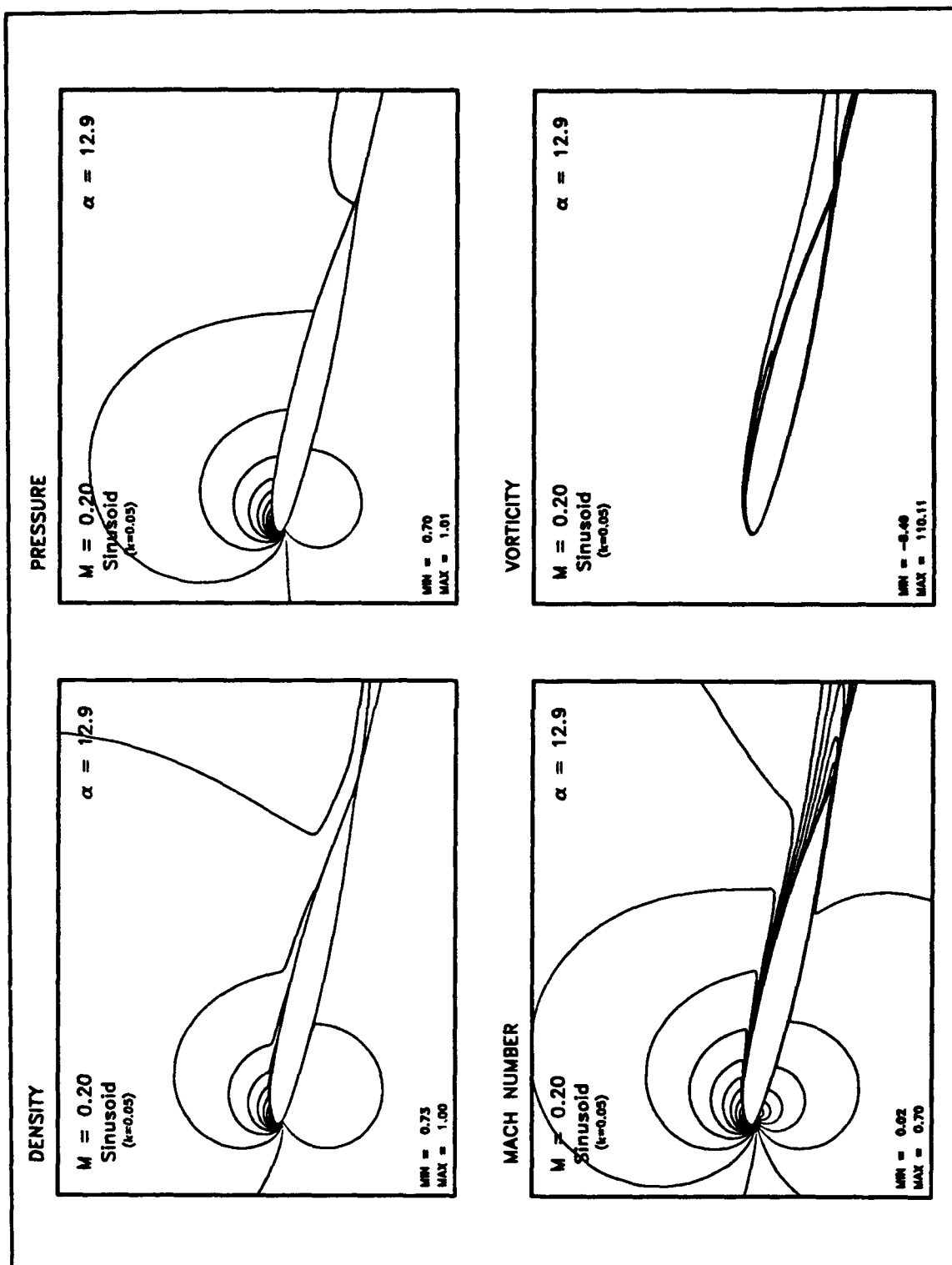


Figure 5.88
Sikorsky SSC-A09
Sinusoid (DOWN) ($k=0.05$, $M=0.2$)

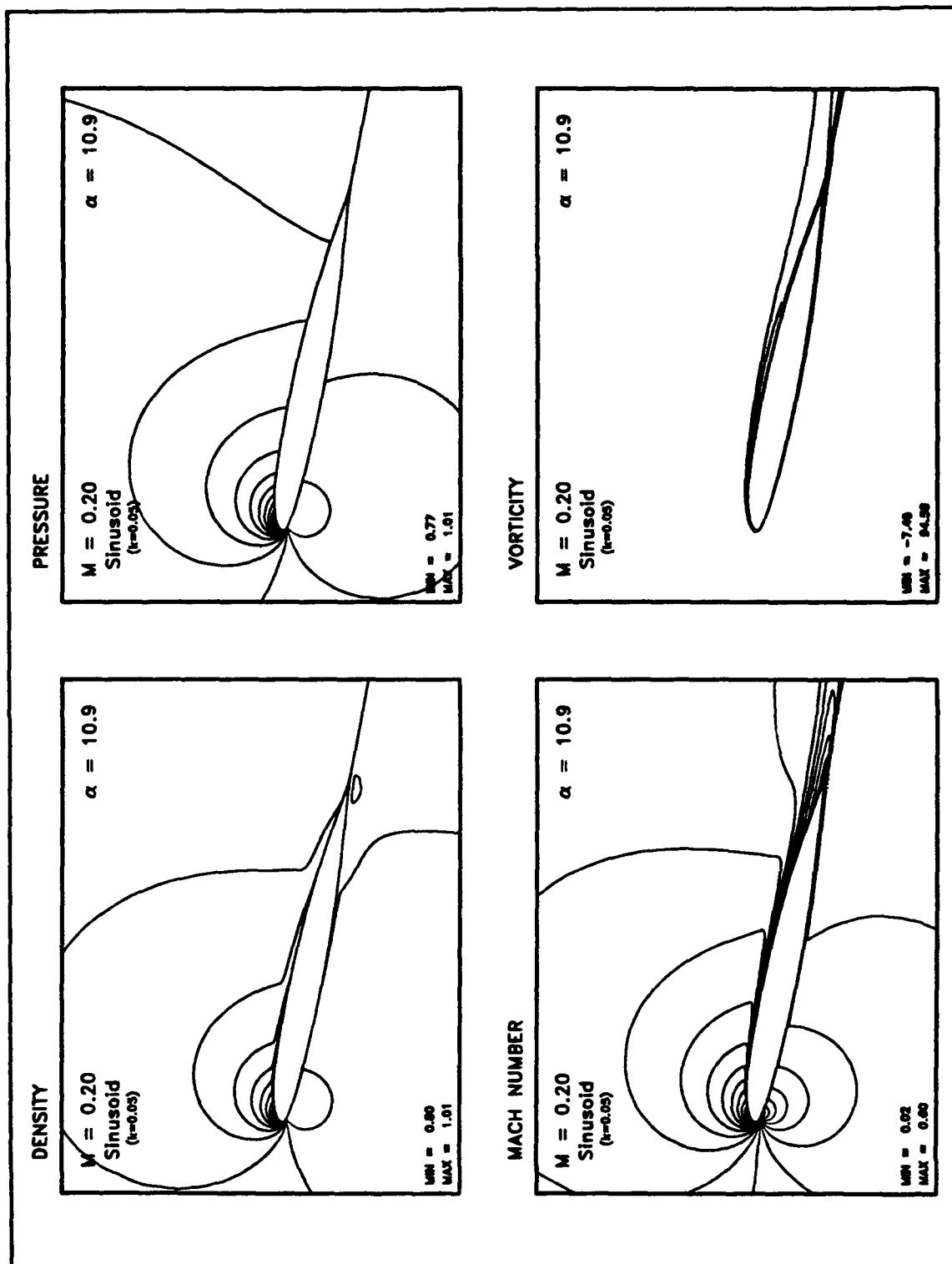


Figure 5.89
Sikorsky SSC-A09
Sinusoid (DOWN) ($k=0.05$, $M=0.2$)

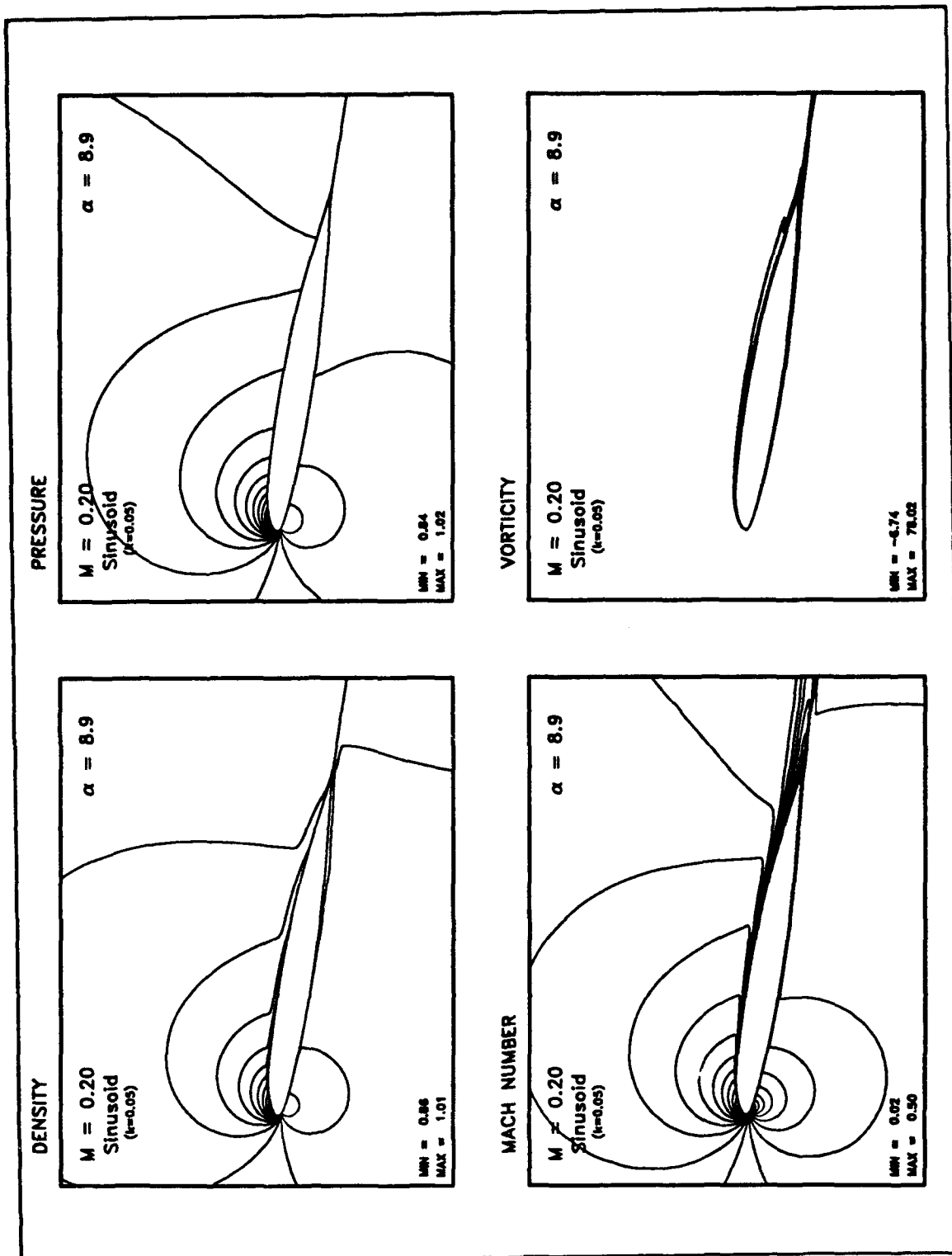


Figure 5.90
Sikorsky SSC-A09
Sinusoid (DOWN) ($k=0.05$, $M=0.2$)

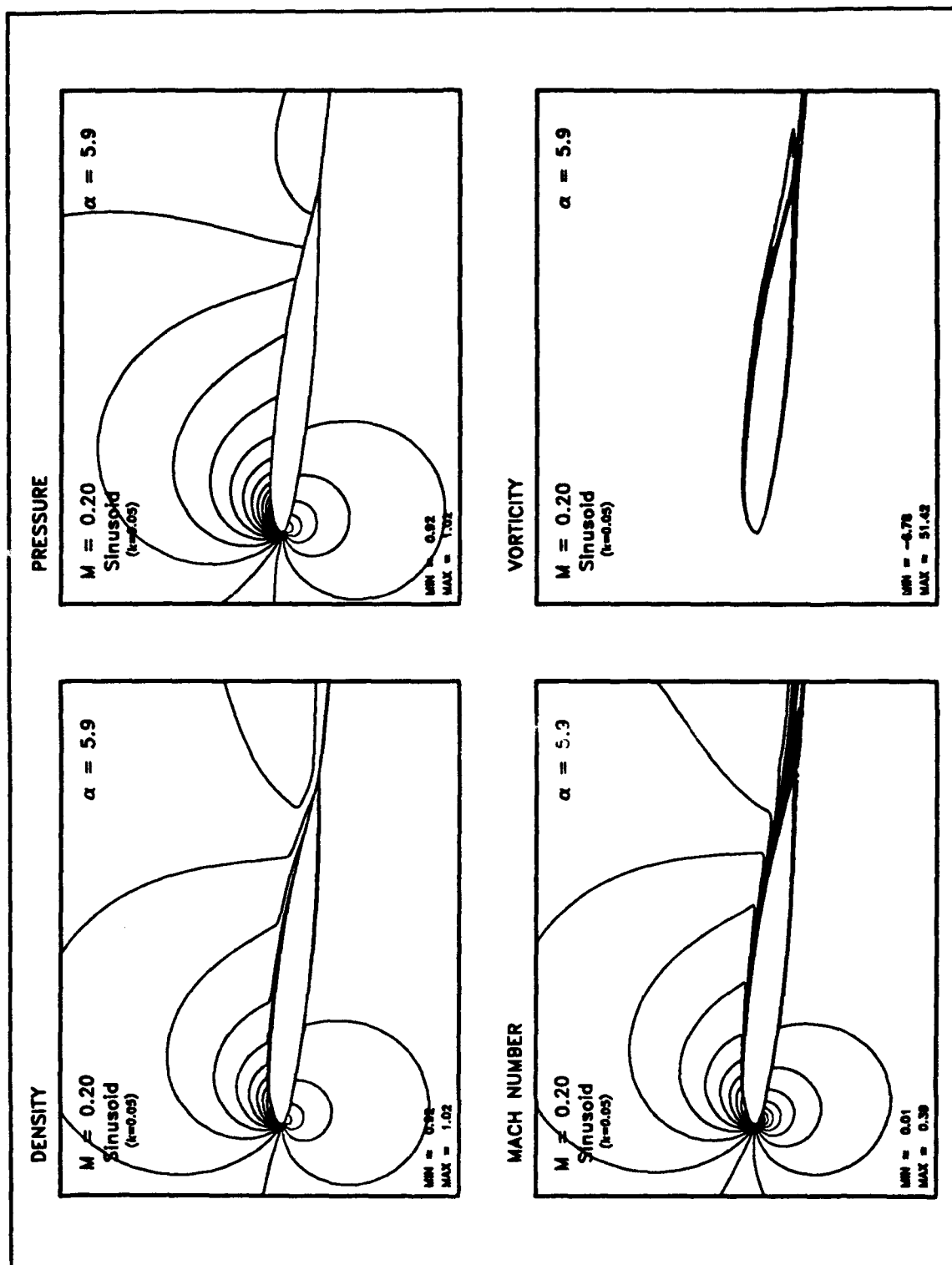


Figure 5.91
Sikorsky SSC-A09
Sinusoid (DOWN) (k=0.05, M=0.2)

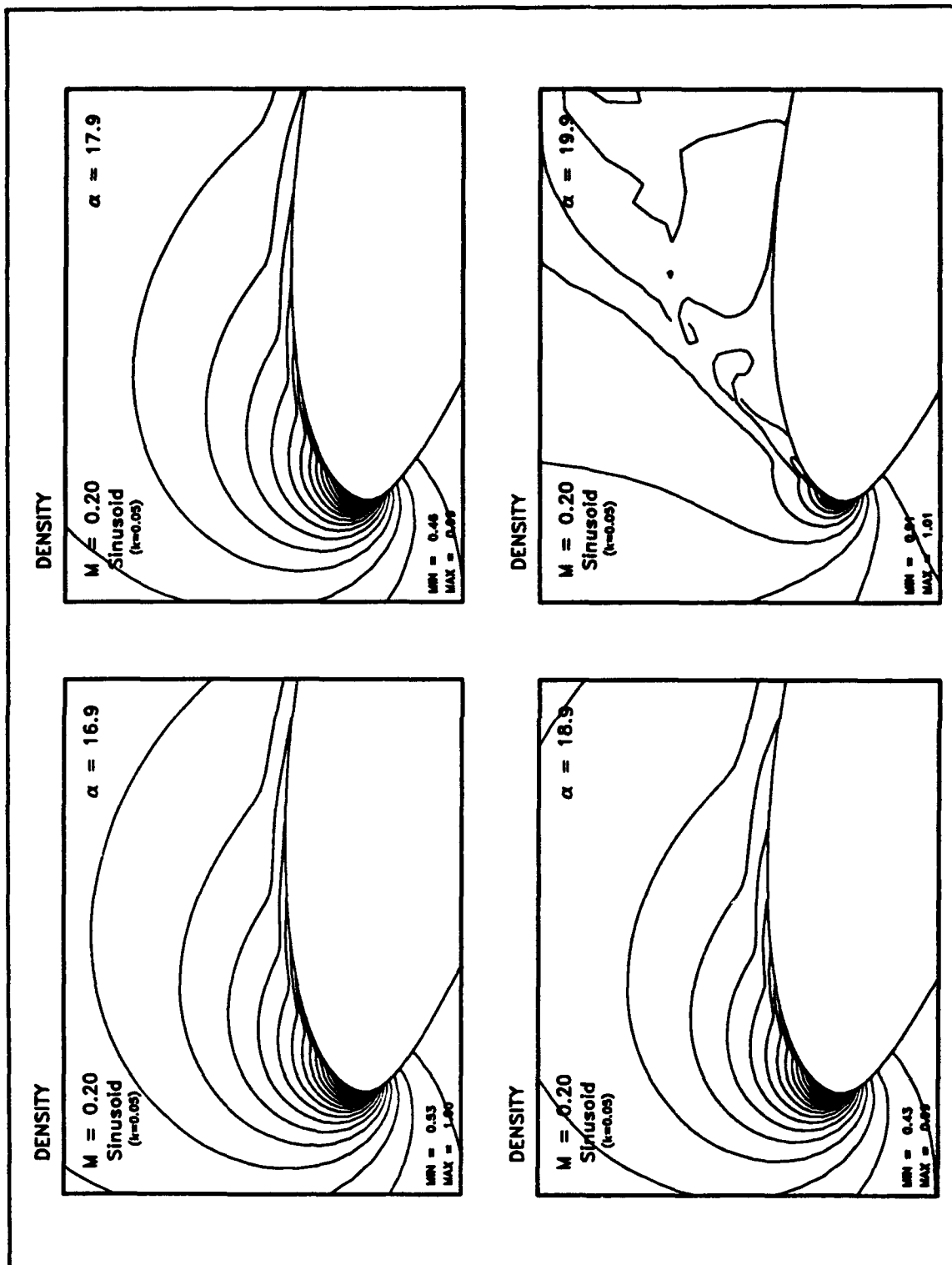


Figure 5.92
Sikorsky SSC-A09
Sinusoid ($k=0.05$, $M=0.2$)

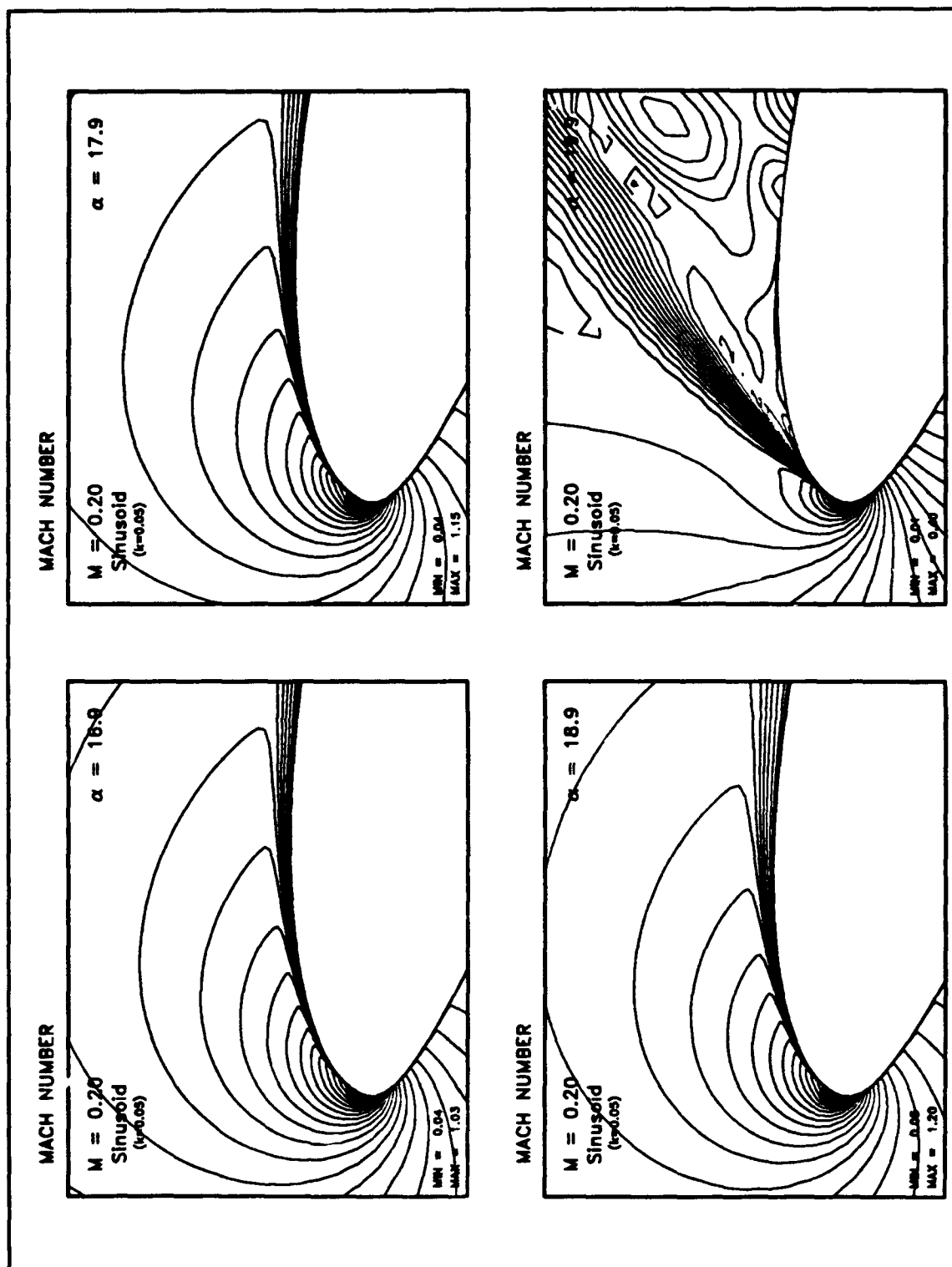


Figure 5.93
Sikorsky SSC-A09
Sinusoid ($k=0.05$, $M=0.2$)

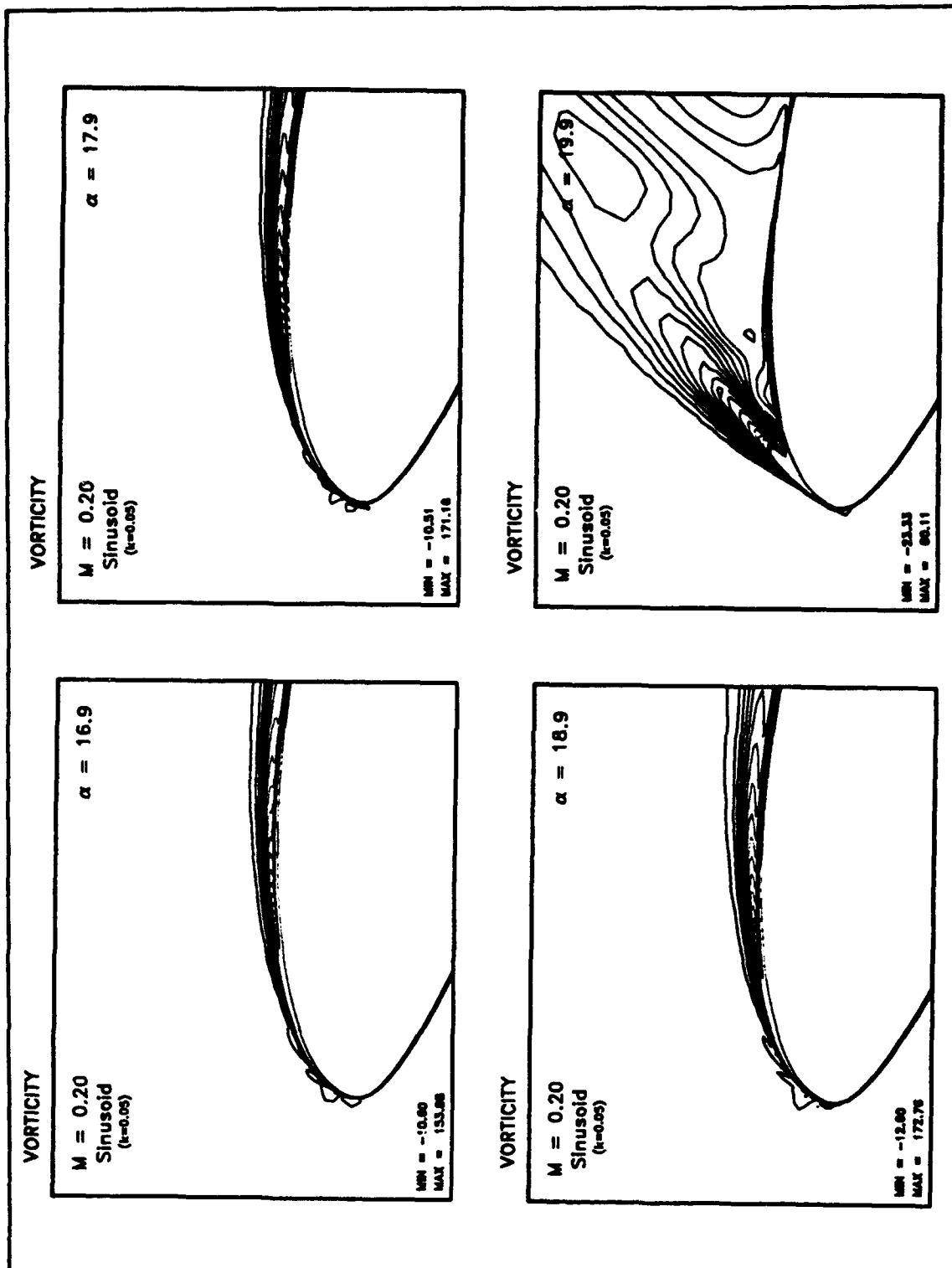


Figure 5.94
Sikorsky SSC-A09
Sinusoid ($k=0.05$, $M=0.2$)

VI. CONCLUSIONS AND RECOMMENDATIONS

The steady and unsteady two-dimensional flowfield analysis was conducted for a Sikorsky SSC-A09 airfoil in compressible, high Reynolds number flows. Computational methods included a steady panel method with compressibility corrections; a laminar and turbulent boundary layer method; an unsteady panel method; and a numerical solution method of the thin layer, compressible, Navier-Stokes equations. The Baldwin-Lomax, two-layer, zero-equation turbulence model was used. In steady flow with little or no separation, computed lift, drag, pitching moment, and skin friction coefficients, as well as displacement thickness and boundary layer velocity profiles at several angles-of-attack were generally found to be in good agreement with experimental data. The much simpler laminar and turbulent boundary layer method produced very good information, more than adequate for the preliminary design process, at a greatly reduced computational cost.

When any airfoil enters deep stall, the flowfield is seen to be dominated by massive flow separation and highly non-linear behavior characterized by the shedding of vortex-like structures. The Baldwin-Lomax, simple eddy viscosity turbulence model used here was found to be inadequate. It predicted steady flows accurately only when there was little or no flow separation. When the flow separated, it over

predicted the leading edge suction peak, over predicted experimentally shown separation, and consistently predicted higher lift and lower pitching-moment.

One major inaccuracy built into the assumptions was the transitional nature of the boundary layer being neglected and instead approximated by a fully turbulent flowfield. Here, the flowfield was shown to be dominated by leading edge separation, often induced by a shock. The small supersonic region and shock that form near the leading edge significantly reduced the peak suction pressure and airloads; thus negating the benefits from dynamic stall by reducing the stall vortex strength. The unsteady aerodynamic response near stall was shown to be strongly dependent on the leading edge stall vortex characteristics. Sinusoidal motions with higher reduced frequencies were shown to be qualitatively similar to ramp motion. Srinivasan, Ekaterinaris, and McCroskey [Ref. 14] concluded that no currently used turbulence model predicted all airloads consistently and in agreement with experiment for all flow conditions. They did conclude that the best improvement over the Baldwin-Lomax model was the Renormalization Group Theory of turbulence (RNG) model which also offered no additional computational costs. The RNG model is also an algebraic eddy viscosity, equilibrium model where the eddy viscosity is assumed to instantaneously adjust to the local flow without any history effects; and the specified integral length-scale is assumed proportional to the boundary

layer thickness. Srinivasan et al [Ref. 14] also concluded from a trade study that a denser C-type grid of 360 by 71 provided the best solution accuracy.

LIST OF REFERENCES

1. Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Corporation, 1984.
2. Anderson, J. D. Jr, *Fundamentals of Aerodynamics*, second edition, McGraw-Hill Publishing Company, 1991.
3. Anderson, J. D. Jr, *Modern Compressible Flow*, second edition, McGraw-Hill Publishing Company, 1990.
4. Baldwin, B. S., Lomax, H., *Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows*, AIAA Paper 78-257, 1978.
5. Cebeci, T. and Bradshaw, P., *Momentum Transfer in Boundary Layers*, Hemisphere Publishing Corporation, 1977.
6. Cricelli, A., Ekaterinaris, J. A., and M. F. Platzer, *Unsteady Airfoil Flow Solutions on Moving Zonal Grids*, AIAA-92-0543, 1992.
7. Eppler, R., *Airfoil Design and Data*, Springer-Verlag, New York, 1990.
8. Harris, Charles D., *NASA Supercritical Airfoils*, NASA Technical Paper 2969, Langley Research Center, Hampton, Virginia, 1990.
9. Hess, J. L. and Smith, A. M., *Calculations of Potential Flow About Arbitrary Bodies*, *Progress in Aeronautical Science*, Vol 8, Pergamon Press, Oxford, 1966.
10. Kuethe, Arnold M. and Chow, Chuen-Yen, *Foundations of Aerodynamics: Bases of Aerodynamic Design*, third edition, John Wiley & Sons, 1976.
11. Lorber, P. F. and Carta, F. O., *Unsteady Stall Penetration Experiments at High Reynolds Number*, United Technologies Research Center, East Hartford, CT, April 1987.

12. Merkle, C. L., *Computational Fluid Dynamics of Inviscid and High Reynolds Number Flows*, First Edition, ProCopy Inc, State College, PA, 1990.
13. Nowak, L. M., *Computational Investigations of a NACA 0012 Airfoil in Low Reynolds Number Flows*, Thesis, Naval Postgraduate School, Monterey, California, September 1992.
14. Srinivasan, G. R., Ekaterinaris, J. A., and McCroskey, W. J., *Dynamic Stall of an Oscillating Wing, Part 1: Evaluation of Turbulence Models*, AIAA Paper 93-3403, 1993.
15. Teng, N. H., June 1987, *The Development of a Computer Code (U2DIFF) for the Numerical Solution of Unsteady, Inviscid and Incompressible Flow over an Airfoil*, Thesis, Naval Postgraduate School, Monterey, California.
16. Tuncer, I. H., Ekaterinaris, J. A., and Platzler, M. F., 1993, *A Viscous-Inviscid Interaction Method for 2-D Unsteady, Compressible Flows*, AIAA Paper 93-3019.
17. VanDyken, R. and Chandrasekhara, M., 1992, *Leading Edge Velocity Field on an Oscillating Airfoil in Compressible Dynamic Stall*, AIAA paper 92-0193.

APPENDIX A

A. BASIC UNIX COMMANDS

```

man <topic> . . . . . obtain help info on any 'topic'
yppasswd . . . . . change your password
mkdir <name> . . . . . make directory
cd <name> . . . . . change directory to 'name'
ls . . . . . list directory contents
ls -l . . . . . w/permissions map and fn size (long option)
cp <fn1> <fn2> . . . . . copies 'fn1' to 'fn2' both in current
                                directory
cp ../<fn1> . . . . . copy 'fn1' from above dir.Period
                                specifies copy to same file name
mv <fn1> <fn2> . . . . . move/rename 'fn1' to 'fn2' & del fn1
rm <fn> . . . . . remove 'fn'
rm -r <dir> . . . . . removes all subfiles without prompting!
                                no way to recover
pwd . . . . . print working dir
more <fn> . . . . . displays 'fn1' one page at a time
head +xx <fn> . . . . . displays xx lines at top of fn
tail -xx <fn> . . . . . displays xx lines at end of fn
df . . . . . indicates disk free space
chmod +/- rwx <fn> . . . . . change 'fn' protection
ps -ef . . . . . show all sys processes (pid)
ps -ef |grep <name> . . . . . shows 'name' current processes
f77 -o <exe> <src.f> . . . . . compile fortran source code
                                'src.f' and use executable name
                                'exe'. -O2 or -O3 options are
                                for vectorization
ctrl c . . . . . kills a process
ctrl z . . . . . suspends a process
lpstat -t . . . . . shows all printing processes

```


3. Program Commands

airfoil

"Input Desired Number of Panels on the Upper, Lower
Surface." (max 100,100)

50,50

"Input NACA 4- or 5- Digit Airfoil Type."

XXXX or XXXXX

Note: Surface Coordinates are created and
stored into 'points.dat' where
{#panels = #points - 1}.

If desired, create or copy your own
surface coordinates to 'points.dat'

c. PROGRAM PANEL.F

This program reads in 'Points.dat', calculates Velocity
and Pressure distributions, and outputs BL2D.F required input
data to 'bl2d.dat'. It generates the files: 'vel.dat',
'cp.dat', 'bl2d.dat', and 'cldm.dat'. It then calculates and
displays force coefficient outputs c_l , c_d , and c_m .

1. Program Commands

panel

"Input the # Panels" (one less than in 'Points.dat')
(sum of two 'airfoil.f' panel entries)

100

"Input R_e (xE6)"(1 → 1,000,000)

1

"Enter Transition Location"

0 → Unknown

1 → Known

{If '1' chosen}

"Input X/C Transition Location Upper Surface"

(Note: The most CRITICAL input !)

0.XX

"Input X/C Transition Location Lower Surface"

0.XX

"Input Angle-of-Attack in Degrees"

X or XX.X

"Input Mach Number"

0 → Incompressible, or

.X → Compressible

D. PROGRAM BL2D.F

This program reads in 'bl2d.dat' and checks for proper stagnation point location. It then generates the files 'bl2d.out', 'cf.dat', 'dls.dat', 'pro1.dat', and 'pro2.dat'.

1. Program Commands

bl2d

"Reading in the data"

"I stagnation is = xx"

"Input new I stagnation ="

xx

"Boundary layer computation in progress"

"Boundary layer computation in progress"

"Michels Transition estimates"

"Estimate for upper transition = .xx"

"Estimate for lower transition = .xx"

2. Program optimization

The program is repeatedly run until convergence is obtained. Procedures:

- Run bl2d
- Run gnuplot
- Load 'dis' and check C_f for convergence
- If diverged solution, change the upper transition point (forward) or alter the stagnation point (± 1) and run bl2d again.
- Repeat the above until convergence is reached.
- Move the transition point progressively aft. Find the point where BL2D will just converge. This is the Transition Point.

E. VISUALIZATION ROUTINES

Data output is viewed using Gnuplot batch files. Gnuplot is a plotting routine available on the Indigos. Procedures to view or print your results:

g	Activates gnuplot plotting routine
l 'press'	View Pressure Distribution (l = load)
l 'vel'	View Velocity distribution
l 'cldm'	View C_{ls} , C_{ds} , or C_{ms}
l 'dis'	View C_F and Displacement Thickness
l 'profile'	View Boundary Layer Velocity Profiles
l 'pt'	Prints any graph displayed on your screen to the hp2p_ps printer

1. Gnuplot Commands

```

set autoscale . . . . . autoscale all axis
set title ' ' . . . . . graph title
set ylabel ' ' . . . . . y title
set xlabel ' ' . . . . . x title
set xrange [ : ]
set yrange [ : ]
set grid . . . . . turn on grid
set ngrid . . . . . turn off grid
set key x,y . . . . . moves legend to x,y
set nokey . . . . . no legend
replot . . . . . redisplay plot with new changes
set label '' at x,y displays a label at position x,y
set data style points . . . displays data as points
set data style lines . . . displays data as lines
set data style linespoints . displays data as lines
                                & points
p 'cp.dat' w lines . . . plots 'cp.dat' using lines
p 'cldm.dat' u 1:3 w lines . plots 'cldm.dat' using
                                1:3 ( $\alpha$  vs  $C_d$ ) using lines

```

2. Gnuplot Batch Files

PRESS

```
#C, distribution
set grid
set key
set label
set function style lines
set tics out
set ticslevel 0.5
set xtics
set ytics
set ztics
set title "Pressure Coefficient Distribution" 0,1
set xlabel "X/C" 0,-1
set xrange [0 : 1]
set ylabel "Cp" 0,.5
set yrange [1 : 4]
plot 'cp.dat' w lines
```

VEL

```
#Velocity distribution
set grid
set key
set label
set function style lines
set tics out
set ticslevel 0.5
set xtics
set ytics
set ztics
set title "Velocity Distribution" 0,1
set xlabel "X/C" 0,-1
set xrange [0 : 1]
set ylabel "Velocity" 0,.5
set yrange [0 : 4]
plot 'vel.dat' w lines
```

CLDM

```
#cldm distribution
set grid
set key
set label
set function style lines
set tics out
set ticslevel 0.5
set xtics
set ytics
set ztics
set autoscale
set title "Lift Coefficient Distribution" 0,1
set xlabel "AOA" 0,-1
#set xrange [0 : 1]
set ylabel "Cl" 0,.5
#set yrange [1 : 4]
#plot 'cldm0012.dat' u 1:3, 'cldm2412.dat' u 1:3
plot 'cldm.dat' u 1:2
```

DIS

```
# Cf and Del* Distribution
set grid
set key
set label
set border
set function style lines
set tics out
set ticslevel 0.5
set xtics
set ytics -.02,.001,.01
set ztics
set title "Cf and Delta-Star" 0,1
set xlabel "X/C" 0,-1
```


Appendix A

```

112      I      = NSTART + N                      U2D11620
113      CALL BODY(Z,SIGN,X(I),Y(I))              U2D11630
114      CONTINUE                                U2D11640
100     NPOINT = NUPPER                          U2D11650
115     SIGN   = 1.0                            U2D11660
116     NSTART = NLOWER                          U2D11670
117     CONTINUE                                U2D11680
118     NODTOT = NLOWER + NUPPER                 U2D11690
119     X(NODTOT+1) = X(1)                      U2D11700
120     Y(NODTOT+1) = Y(1)                      U2D11710
121     RETURN                                  U2D11930
122     END                                      U2D11940
123
*****
*****

```

PROGRAM PANEL

```

*****
* ***** AUGUST 1993 *****
*          +++++ CDR T. Johnston +++++
* PURPOSE: CALCULATE THE VELOCITIES ON AN AIRFOIL USING A PANEL METHOD.
* LIM:     Arrays currently dimensioned for maximum of N=200 panels
*          Input data file points.dat will have N+1 points
*          Output velocities are referenced to freestream, ie. V/Vinf
*
* METHOD:   FLOWFIELD CONSISTS OF THREE SIMPLER FLOWS: FREESTREAM, SOURCE,
*          AND VORTICITY. SOURCE DISTRIBUTIONS q(j) VARY FROM PANEL TO
*          PANEL. VORTICITY STRENGTH GAMMA IS THE SAME FOR ALL PANELS.
*          BOUNDARY CONDITIONS INCLUDE FLOW TANGENCY AT CONTROL POINTS AND
*          KUTTA CONDITION FOR FIRST AND LAST PANELS. INFLUENCE
*          COEFFICIENTS COMBINED TO FORM NEW COEFFICIENTS IN LINEAR SYSTEM
*          OF n+1 EQUATIONS, n+1 UNKNOWN (q(1)...q(n), GAMMA). VELOCITIES
*          AT CONTROL POINTS EVALUATED FROM q(j) AND GAMMA.
*****
* Variable Definitions
* N = # Panels
* RL = Reynolds Number
* IANS = USER Transition Location Flag
* TRANSUPPER = Upper Transition Point
* TRANSLOWER = Lower Transition Point
* ALPHA = Angle of Attack
* P = Mach Number
* BETA1 = SQRT ( 1-M^2 )
* X(I) = X Surface Coordinates
* Y(I) = Y Surface Coordinates
* XM(I) = X Control Points
* YM(I) = Y Control Points
* R(I,J) = Distance, Control Point to Panel with Unit Strength
*          Singularity Distribution
* AN(I,J) = Normal Velocity Component Induced at Ith Panel Control
*          Point by a Unit Source on Jth Panel
* AT(I,J) = Tangential Velocity Component Induced at Ith Panel Control
*          Point by a Unit Source on Jth Panel
* BN(I,J) = Normal Velocity Component Induced at Ith Panel Control
*          Point by a Unit Vorticity on Jth Panel
* BT(I,J) = Tangential Velocity Component Induced at Ith Panel Control
*          Point by a Unit Vorticity on Jth Panel
* q(I) = Source Strength
* GAMMA(I) = Vorticity Strength
* Vt(I) = Normalized Tangential Velocity @ I Control Point { V/Vinf }
* Vtc(I) = Compressible Normalized Tangential Velocity @ I Control pt
* CP = Pressure Coefficient { Cp = 1 - Vt^2 }
* ISTAG = Location of Stagnation Panel (Vt reversal)
*****
1      REAL X(1:202),Y(1:202),XM(1:202),YM(1:202),
2      : At(1:202,1:202),Bt(1:202,1:202),
3      : a(1:202,1:202),b(1:202),
4      : q(1:202),Vt(1:202),ALPHA,VtC(1:202),cP,cPc,CCPC(202),
5      : PI,GAMMA,THETA(1:202),NUM,DEN,
6      : R(1:202,1:202),BETA(1:202,1:202),NUM1,DEN1,NUM2,DEN2,
7      : AAUG(1:202,1:202),An(1:202,1:202),Bn(1:202,1:202)
8
9      * NUMBER OF PANELS ON AIRFOIL SURFACE:
10     PRINT*, 'INPUT NO. OF PANELS (1 less than #lines in points.dat):'
11     READ *,N
12     PI=ACOS(-1.)

```

```

13      OPEN (UNIT=88,FILE='points.dat',STATUS='UNKNOWN')
14      OPEN (UNIT=89,FILE='vel.dat',STATUS='UNKNOWN')
15      OPEN (UNIT=81,FILE='cp.dat',STATUS='UNKNOWN')
16      OPEN (UNIT=90,FILE='bl2d.dat',STATUS='UNKNOWN')
17      OPEN (UNIT=91,FILE='cldm.dat',STATUS='UNKNOWN')
18      print *, 'INPUT REYNOLDS NUMBER = ____ (X E+6)'
19      READ *, RL
20      RL = RL*1000000.
21
22      print *, 'ENTER 0 IF TRANSITION LOCATIONS UNKNOWN (.8 & .999 USED)'
23      PRINT *, '      1 IF You wish to enter Transition Locations'
24      READ *, IANS
25
26      IF (IANS.EQ.1) THEN
27          PRINT *, 'INPUT X/C TRANSITION LOCATION FOR UPPER SURFACE:'
28          READ *, TRANSUPPER
29          PRINT *, 'INPUT X/C TRANSITION LOCATION FOR LOWER SURFACE:'
30          READ *, TRANSLOWER
31      ELSE
32          ***These are arbitrary values intended to be downstream of the
33          *** actual transition points, for use with Michel's criterion in BL2D
34          TRANSUPPER=.8
35          TRANSLOWER=.999
36      ENDIF
37      WRITE (90,50) RL,TRANSUPPER,TRANSLOWER
38      50      FORMAT (F10.0,F10.6,F10.6)
39      PRINT *, 'INPUT ANGLE OF ATTACK IN DEGREES:'
40      READ *, ALPHA
41      ALPHA=ALPHA*PI/180.0
42      PRINT *, 'INPUT MACH NUMBER (0 FOR INCOMPRESSIBLE):'
43      READ *, P
44      p = rl/10000000.
45      BETA1=(1.0-P**2.0)**.5
46
47      **      Read in Points.dat - Surface Coordinates must be TE, Clockwise, TE
48      DO 30 I=1,N+1
49          READ (88,*) X(I),Y(I)
50      30      CONTINUE
51
52      *      This section defines the influence coefficients:
53      DO 110 I=1,N
54      *      Control Points
55          XM(I)=0.5*(X(I)+X(I+1))
56          YM(I)=0.5*(Y(I)+Y(I+1))
57          R(I,1)=(XM(I)-X(1))**2.+(YM(I)-Y(1))**2.
58          DO 100 J=1,N
59              NUM=Y(J+1)-Y(J)
60              DEN=X(J+1)-X(J)
61              THETA(J)=ATAN2(NUM,DEN)
62              NUM1=YM(I)-Y(J+1)
63              DEN1=XM(I)-X(J+1)
64              NUM2=YM(I)-Y(J)
65              DEN2=XM(I)-X(J)
66              BETA(I,J)=ATAN2((NUM1*DEN2-DEN1*NUM2),(DEN1*DEN2+NUM1*NUM2))
67              R(I,J+1)=(XM(I)-X(J+1))**2.+(YM(I)-Y(J+1))**2.
68              THETADIF=THETA(I)-THETA(J)
69              IF (I.EQ.J)
70                  THEN
71                  An(I,J)=0.5
72                  Bn(I,J)=0.0
73              ELSE
74                  An(I,J)=(1/(2*PI))*(SIN(THETADIF)*LOG(R(I,J+1)/R(I,J))
75                  * .5+COS(THETADIF)*BETA(I,J))
76                  Bn(I,J)=(1/(2*PI))*(COS(THETADIF)*LOG(R(I,J+1)/R(I,J))
77                  * .5-SIN(THETADIF)*BETA(I,J))
78              END IF
79              At(I,J)=-Bn(I,J)
80              Bt(I,J)=An(I,J)
81      100      CONTINUE
82      110      CONTINUE
83
84      * Matrix coefficients of linear system defined (a's and b's):
85          a(N+1,N+1)=0.0
86          DO 130 I=1,N
87              a(I,N+1)=0.0
88          DO 120 J=1,N
89              a(I,J)=An(I,J)
90              a(I,N+1)=a(I,N+1)+Bn(I,J)

```



```

168      SUBROUTINE FANDM(ALPHA,cl,cd,cm,CCPC,x,y,n)
169      REAL ALPHA,CL,CD,CM,CCPC(202),X(202),Y(202),CFX,CFY,XMID,YMID,
170      :      DX,DY
171      CFX = 0.0
172      CFY = 0.0
173      CM = 0.0
174      DO 100 I = 1,N
175      ***      moment coeff is computed around pivot point at 25% Chord
176      XMID = .5*(X(I) + X(I+1)) - 0.25
177      YMID = .5*(Y(I) + Y(I+1))
178      DX = X(I+1) - X(I)
179      DY = Y(I+1) - Y(I)
180      CFX = CFX + CCPC(I)*DY
181      CFY = CFY - CCPC(I)*DX
182      CM = CM + CCPC(I)*(DX*XMID + DY*YMID)
183      100 CONTINUE
184      CD = CFX*COS(ALPHA) + CFY*SIN(ALPHA)
185      CL = CFY*COS(ALPHA) - CFX*SIN(ALPHA)
186      RETURN
187      END
188
189      *****
190      * Gauss elimination procedure
191      SUBROUTINE GAUSS(N,Z)
192      INTEGER PV
193      REAL Z(1:202,1:203),E
194      E=1.0
195      10 IF (1.0+E.GT.1.0) THEN
196      E=E/2.0
197      GOTO 10
198      END IF
199      E=E*2
200      EPS2 = 2 * E
201      C PRINT *, ' MACHINE EPSILON=',E
202      DET= 1
203      1005 DO 1010 I=1,N-1
204      PV=I
205      DO 1020 J=I+1,N
206      IF (ABS(Z(PV,I)) .LT. ABS(Z(J,I))) PV=J
207      1020 CONTINUE
208      IF (PV.EQ.I) GOTO 1050
209      DO 1040 JC=1,N+1
210      TM=Z(I,JC)
211      Z(I,JC)=Z(PV,JC)
212      Z(PV,JC)=TM
213      1040 CONTINUE
214      DET=-1*DET
215      1050 IF (Z(I,I).EQ.0) THEN
216      GOTO 1200
217      END IF
218      DO 1060 JR=I+1, N
219      IF (Z(JR,I).NE.0) THEN
220      R=Z(JR,I)/Z(I,I)
221      DO 1075 KC=I+1,N+1
222      TEMP=Z(JR,KC)
223      Z(JR,KC)=Z(JR,KC)-R*Z(I,KC)
224      IF (ABS(Z(JR,KC)).LT.EPS2*TEMP) Z(JR,KC)=0.0
225
226      C      !-- if the result of subtraction is smaller than
227      C      !-- 2 times machine epsilon times the original
228      C      !-- value, it is set to zero.
229      1075 CONTINUE
230      END IF
231      1060 CONTINUE
232      1010 CONTINUE
233      DO 1084 I=1,N
234      DET=DET*Z(I,I)
235      1084 CONTINUE
236      IF (Z(N,N).EQ.0) GOTO 1200
237      Z(N,N+1)=Z(N,N+1)/Z(N,N)
238      DO 1130 NV=N-1,1,-1
239      VA=Z(NV,N+1)
240      DO 1120 K=NV+1,N
241      VA=VA-Z(NV,K)*Z(K,N+1)
242      1120 CONTINUE
243      Z(NV,N+1)=VA/Z(NV,NV)
244      1130 CONTINUE

```

```

245      RETURN
246 1200 PRINT *, 'MATRIX IS SINGULAR'
247      PRINT *, 'I=', I, 'Z(I,I)=', Z(I,I)
248      STOP
249      END

```

***** ***** PROGRAM BL2D *****

```

*          CDR T. Johnston (July 1993)
*****
*      XCTRI(1) = Input for upper transition pt from panel
*      XCTRI(2) = Input for Lower transition pt from panel
*      NI       = # Panels
*      IS       = Stagnation Pt
*      ITRANS   = Default/USER input for transition flag
*      XI(I)    = Control pts
*      YI(I)    = Control pts
*      VEI(I)   = Tangential Velocities
*      NXTSF(1) = # panels on upper surface
*      NXTSF(2) = # panels on lower surface
*      NXT      = # panels on a surface in 200 ISF loop
*      ISF      = Surface flag (1=upper, 2=Lower)
*      XC(I)    = Redefined Control pts
*      YC(I)    = Redefined Control pts
*      X(I)     = Original nodes
*****
1      COMMON /BLC0/ RL,TRANSNEW(2),NBL(2),XCTRI(2),ntflag,NI
2      COMMON /BLC1/ XCTR,XC(200),YC(200),itr
3      COMMON /BLC2/ NX,NXT,NP,NPT,NTR,IT,ISF
4      COMMON /BLC3/ X(200),UE(200),P1(200),P2(200),GMTR(200)
5      COMMON /BLC5/ DLS(200),VW(200),CF(200),THT(200)
6      integer ntflag,ni,nx,np,nxt,is,itrans,isf,it,npt,ntr
7      DIMENSION NXTSF(2),XI(200),YI(200),VEI(200)
8      CHARACTER VAR*25
9      REAL PGAMTR
10     OPEN (UNIT=9,FILE='bl2d.dat',STATUS='UNKNOWN')
11     OPEN (UNIT=8,FILE='bl2d.out',STATUS='UNKNOWN')
12     OPEN (UNIT=20,FILE='cf.dat',STATUS='UNKNOWN')
13     OPEN (UNIT=21,FILE='dls.dat',STATUS='UNKNOWN')
14     OPEN (UNIT=54,FILE='pro1.dat',STATUS='UNKNOWN')
15     OPEN (UNIT=55,FILE='pro2.dat',STATUS='UNKNOWN')
16 **      READ BL2D.DAT
17     WRITE(6,*) 'READING THE DATA...'
18     READ ( 9,* ) RL,XCTRI(1),XCTRI(2)
19     READ ( 9,* ) NI,IS,ITRANS
20     READ ( 9,15 ) (XI(I),YI(I),VEI(I),I=1,NI)
21     WRITE(6,*) 'INPUT OF DATA COMPLETE.'
22 ***      Check Stagnation Point for adjustment
23     print *
24     print *, ' I Stagnation is '
25     print *, IS
26     print *
27     print *, ' INPUT NEW I Stagnation = '
28     READ *, IS
29 **      Write to Bl2d.out
30     5 WRITE(8,90) RL,XCTRI(1),XCTRI(2)
31     NXTSF(1)= NI - IS + 1
32     NXTSF(2)= IS
33 ***      DATA FOR EACH SURFACE (UPPER,LOWER)
34     DO 200 ISF = 1,2
35         ntflag=0
36         NXT = NXTSF(ISF)
37         GO TO (201,202),ISF
38 ***      REDEFINE CONTROL POINTS
39     C      UPPER SURFACE
40     201 II = IS-1
41         DO 211 I=1,NXT
42             II = II+1
43             XC(I) = XI(II)
44             YC(I) = YI(II)
45             UE(I) = VEI(II)

```

```

46      211 CONTINUE
47      GO TO 300
48      C      LOWER SURFACE
49      202 II = IS+1
50      DO 212 I=1,NXT
51          II = II-1
52          XC(I) = XI(II)
53          YC(I) = YI(II)
54          UE(I) = VEI(II)
55      212 CONTINUE
56      300 X(1) = 0.0
57      ***      DEFINE NODES
58      DO 301 I=2,NXT
59      301 X(I) = X(I-1)+SQRT((XC(I)-XC(I-1))**2+(YC(I)-YC(I-1))**2)
60      C      TRANSITION LOCATION
61      DO 320 I=1,NXT
62          GMTR(I) = 0.0
63          IF (XC(I) .GE. XCTRI(ISF)) GO TO 321
64      320 CONTINUE
65      ***      NTR IS TRANSITION PANEL
66      321 NTR = I
67      ***      CEBECI-SMITH TURBULENCE MODEL
68      PGAMTR = 1200.
69      RXNTR = X(NTR-1) * UE(NTR-1) * RL
70      GGFT = RL**2/RXNTR**1.34*UE(NTR-1)**3
71      UEINTG = 0.0
72      U1 = 0.5/UE(NTR-1)/ PGAMTR
73      DO 322 I = NTR,NXT
74          U2 = 0.5/UE(I)/PGAMTR
75          UEINTG = UEINTG+(U1+U2)*(X(I)-X(I-1))
76          U1 = U2
77          GG = GGFT*UEINTG*(X(I)-X(NTR-1))
78          IF(GG .GT. 10.0) GO TO 323
79          GMTR(I) = 1.0-EXP(-GG)
80      322 CONTINUE
81      323 DO 324 II=I,NXT
82      324 GMTR(II) = 1.0
83      C      PRESSURE GRADIENT PARAMETERS
84      DX = X(2)-X(1)
85      DUE = UE(2)-UE(1)
86      ANG2 = ATAN2(DUE,DX)
87      DL2 = DX
88      DO 331 I = 2,NXT-1
89          ANG1 = ANG2
90          DL1 = DL2
91          DX = X(I+1)-X(I)
92          DUE = UE(I+1)-UE(I)
93          ANG2 = ATAN2(DUE,DX)
94          DL2 = DX
95          ANG = (DL2*ANG1+DL1*ANG2)/(DL1+DL2)
96          P2(I) = TAN(ANG)
97      331 CONTINUE
98      P2(NXT) = 2.*DUE/DL2 - P2(NXT-1)
99      DO 330 I = 2,NXT
100      P2(I) = X(I) * P2(I) /UE(I)
101      P1(I) = 0.5 * (1.0 + P2(I))
102      330 CONTINUE
103      P2(1) = 1.0
104      P1(1) = 0.5 * (1.0 + P2(1))
105      **      BOUNDARY LAYER CALCULATION
106      WRITE(6,*) 'BOUNDARY LAYER COMPUTATIONS IN PROGRESS..'
107      CALL BL
108      WRITE(8,910) ISF, (I,XC(I),X(I),VW(I),CF(I),DLS(I),THT(I),I=1,NXT)
109      if (ISF.EQ.1) then
110          write(20,905) (XC(I),CF(I),I=2,NXT)
111          write(21,905) (XC(I),DLS(I),I=2,NXT)
112      end if
113      905 FORMAT(F8.4,4X,E11.4)
114      200 CONTINUE
115      ***IF AOA is 0 deg., make trans. locs. equal:
116      if(vei(2).eq.vei(ni-1)) transnew(1)=transnew(2)
117      *** Display Michels transition estimate
118      if(ITRANS.eq.0) then
119      c      print *, 'MICHELS TRANSITION ESTIMATE ****'
120          print *, 'Estimate for upper transition:',transnew(1)
121          print *, 'Estimate for lower transition:',transnew(2)
122      c      END IF
123      CLOSE(UNIT=8)

```

```

124      CLOSE (UNIT=9)
125      CLOSE (UNIT=20)
126      CLOSE (UNIT=21)
127      CLOSE (UNIT=54)
128      CLOSE (UNIT=55)
129      10  FORMAT (3I5)
130      15  FORMAT (3F10.0)
131      90  FORMAT (//5X, 'RL=', E12.5, 5X, 'XCTRI(1) =', F8.3, 5X, 'XCTR(2) =', F8.3)
132      95  FORMAT (5X, A25, 2X, f8.7)
133      910 FORMAT (//2X, '*** SUMMARY OF BOUNDARY LAYER SOLUTIONS OF ISF =', I2
134      +//2X, 'NX', 4X, 'XC', 8X, 'S', 8X, 'VW', 8X, 'CF', 8X, 'DLS', 8X, 'THT'
135      +// (I5, 2F8.4, 4E11.4))
136      print*
137      print*, 'Output files: cf.dat, dls.dat, prol.dat, pro2.dat, bl2d.dat'
138      STOP
139      END
140      *****
141
142      SUBROUTINE BL
143      COMMON /BLC2/ NX, NXT, NP, NPT, NTR, IT, ISF
144      COMMON /BLC3/ X(200), UE(200), P1(200), P2(200), GMTR(200)
145      COMMON /BLC7/ ETA(201), DETA(201), A(201)
146      COMMON /BLC8/ F(201,2), U(201,2), V(201,2), B(201,2)
147      COMMON /BLC6/ DELF(201), DELU(201), DELV(201)
148      integer ntfalg, ni, nx, np, nxt, is, itrans, isf, it, npt, ntr
149      NX      = 0
150      ITMAX   = 10
151      IGROW   = 2
152      EPSL    = 0.0001
153      EPST    = 0.01
154      NPT     = 101
155      C      ETA-GRID
156      ETAE    = 8.0
157      VGP     = 1.10
158      DETA(1) = 0.01
159      NP = LOG((ETAE/DETA(1)) * (VGP-1.0) + 1.0) / LOG(VGP) + 1.001
160      ETA(1) = 0.0
161      DO 10 J=2, NPT
162          ETA(J) = ETA(J-1) + DETA(J-1)
163          DETA(J) = VGP*DETA(J-1)
164          A(J)   = 0.5*DETA(J-1)
165      10  CONTINUE
166      C      INITIAL LAMINAR VELOCITY PROFILE
167      DO 20 J=1, NP
168          ETAB = ETA(J)/ETA(NP)
169          ETAB2 = ETAB**2
170          F(J,2) = 0.25*ETA(NP)*ETAB2*(3.0 - 0.5*ETAB2)
171          U(J,2) = 0.5*ETAB*(3.0 - ETAB2)
172          V(J,2) = 1.5*(1.0 - ETAB2)/ETA(NP)
173          B(J,2) = 1.0
174      20  CONTINUE
175      ** Step thru all panels (NX to NXT)
176      1  NX      = NX+1
177      IT      = 0
178      IGROW   = 0
179      5  IT      = IT+1
180      IF (IT .GT. ITMAX) GO TO 101
181      IF (NX .GE. NTR) CALL EDDY
182      CALL COEF
183      CALL SOLV3
184      C      CHECK FOR CONVERGENCE
185      IF (NX .LT. NTR) THEN
186          IF (ABS(DELV(1)) .GT. EPSL) GO TO 5
187      ELSE
188          IF (ABS(DELV(1)/V(1,2)) .GT. EPST) GO TO 5
189      ENDIF
190      C      PROFILES FOR GROWTH
191      99  DO 30 J=NP+1, NPT
192          F(J,2) = F(J-1,2) + DETA(J-1)*U(J-1,2)
193          U(J,2) = U(J-1,2)
194          V(J,2) = 0.0
195          B(J,2) = B(J-1,2)
196      30  CONTINUE
197      C      CHECK FOR GROWTH
198      IF (ABS(V(NP,2)) .GT. 0.0005 .OR. ABS(1.0-U(NP-2,2)/U(NP,2))
199      + .GT. 0.005) THEN
200          NP = NP+2
201          IGROW = IGROW+1

```

```

202         IF (NP .LE. NPT .AND. IGROW .LE. IGROWT) THEN
203             IT = 0
204             GO TO 5
205         ENDIF
206     ENDIF
101 CALL OUTPUT
208     IF (NX .LT. NXT) GO TO 1
209     RETURN
210     END
211
212 *****
213     SUBROUTINE COEF
214     COMMON /BLC2/ NX,NXT,NP,NPT,NTR,IT,ISF
215     COMMON /BLC3/ X(200),UE(200),P1(200),P2(200),GMTR(200)
216     COMMON /BLC7/ ETA(201),DETA(201),A(201)
217     COMMON /BLC8/ F(201,2),U(201,2),V(201,2),B(201,2)
218     COMMON /BLC9/ S1(201),S2(201),S3(201),S4(201),S5(201),S6(201),
219     + S7(201),S8(201),R1(201),R2(201),R3(201),R4(201)
220     integer ntflag,ni,nx,np,nxt,is,itrans,isf,it,npt,ntr
221     P1H = 0.5 * P1(NX)
222     IF (NX .EQ. 1) THEN
223         CEL = 0.0
224         CELH = 0.0
225         DO 5 J=1,NP
226             F(J,1) = 0.0
227             U(J,1) = 0.0
228             V(J,1) = 0.0
229             B(J,1) = 0.0
230         5 CONTINUE
231     ELSE
232         CEL = 0.5 * (X(NX)+X(NX-1)) / (X(NX)-X(NX-1))
233         CELH = 0.5 * CEL
234     ENDIF
235     DO 100 J= 2,NP
236     C CURRENT STATION
237         FB = 0.5*(F(J,2) + F(J-1,2))
238         UB = 0.5*(U(J,2) + U(J-1,2))
239         FVB = 0.5*(F(J,2)*V(J,2)+F(J-1,2)*V(J-1,2))
240         VB = 0.5*(V(J,2) + V(J-1,2))
241         USB = 0.5*(U(J,2)**2 + U(J-1,2)**2)
242         DERBV = (B(J,2)*V(J,2) - B(J-1,2)*V(J-1,2))/DETA(J-1)
236     C PREVIOUS STATION
244         CFB = 0.5*(F(J,1) + F(J-1,1))
245         CUB = 0.5*(U(J,1) + U(J-1,1))
246         CVB = 0.5*(V(J,1) + V(J-1,1))
247         CUSB = 0.5*(U(J,1)**2 + U(J-1,1)**2)
248         CFVB = 0.5*(F(J,1)*V(J,1)+F(J-1,1)*V(J-1,1))
249         CDERBV = (B(J,1)*V(J,1) - B(J-1,1)*V(J-1,1))/DETA(J-1)
250     C S- COEFFICIENTS
251         S1(J) = CELH*(F(J,2) - CFB) + P1H*F(J,2) + B(J,2)/DETA(J-1)
252         S2(J) = CELH*(F(J-1,2)-CFB) + P1H*F(J-1,2)-B(J-1,2)/DETA(J-1)
253         S3(J) = CELH*(V(J,2) + CVB) + P1H*V(J,2)
254         S4(J) = CELH*(V(J-1,2) + CVB) + P1H*V(J-1,2)
255         S5(J) = -(CEL+P2(NX))*U(J,2)
256         S6(J) = -(CEL+P2(NX))*U(J-1,2)
257     C R- COEFFICIENTS
258         IF (NX .EQ. 1) THEN
259             CRB = -P2(NX)
260             R2(J) = CRB - (DERBV + P1(NX)*FVB - P2(NX)*USB)
261         ELSE
262             CLB = CDERBV + P1(NX-1)*CFVB - P2(NX-1)*CUSB + P2(NX-1)
263             CRB = -CLB - CEL*CUSB - P2(NX)
264             R2(J) = CRB - (DERBV + P1(NX)*FVB - (CEL+P2(NX))*USB + CEL*
265     + (FVB + CVB*FB - VB*CFB - CFVB))
266         ENDIF
267         R1(J) = F(J-1,2) - F(J,2) + DETA(J-1)*UB
268         R3(J-1) = U(J-1,2) - U(J,2) + DETA(J-1)*VB
269     100 CONTINUE
270     C BOUNDARY CONDITIONS
271         R1(1) = 0.0
272         R2(1) = 0.0
273         R3(NP) = 0.0
274     RETURN
275     END
276
277 *****
278 *****
279     SUBROUTINE EDDY

```

```

280      COMMON /BLC0/ RL,transnew(2),NBL(2),XCTRI(2),ntflag,NI
281      COMMON /BLC2/ NX,NXT,NP,NPT,NTR,IT,ISF
282      COMMON /BLC3/ X(200),UE(200),P1(200),P2(200),GMTR(200)
283      COMMON /BLC7/ ETA(201),DETA(201),A(201)
284      COMMON /BLC8/ F(201,2),U(201,2),V(201,2),B(201,2)
285      integer ntflag,ni,nx,np,nxt,is,itrans,isf,it,npt,ntr
286      DIMENSION EDVI(201)
287      RL2 = SQRT(RL*UE(NX)*X(NX))
288      RL4 = SQRT(RL2)
289      RL216 = 0.16 * RL2
290      ALFA = 0.0168
291      EDVO = ALFA*RL2*GMTR(NX)*(U(NP,2)*ETA(NP)-F(NP,2))
292      EDVI(1)= 0.0
293      YBAJ = RL4*SQRT(ABS(V(1,2)))/26.0
294      DO 70 J=2,NP
295          JJ = J
296          YBA = YBAJ*ETA(J)
297          EL = 1.0
298          IF(YBA .LT. 10.0) EL = 1.0 - EXP(-YBA)
299          EDVI(J) = RL216*GMTR(NX)*(EL*ETA(J))**2 * ABS(V(J,2))
300          IF(EDVI(J) .GT. EDVO) GO TO 90
301          IF (EDVI(J) .LE. EDVI(J-1)) EDVI(J)= EDVI(J-1)
302          B(J,2) = 1.0 + EDVI(J)
303      70 CONTINUE
304      90 DO 100 J=J,NPT
305      100 B(JJ,2) = 1.0 + EDVO
306      B(1,2) = 1.0
307      RETURN
308      END
309
310      *****
311      SUBROUTINE OUTPUT
312      COMMON /BLC0/ RL,transnew(2),NBL(2),XCTRI(2),ntflag,NI
313      COMMON /BLC2/ NX,NXT,NP,NPT,NTR,IT,ISF
314      COMMON /BLC3/ X(200),UE(200),P1(200),P2(200),GMTR(200)
315      COMMON /BLC7/ ETA(201),DETA(201),A(201)
316      COMMON /BLC8/ F(201,2),U(201,2),V(201,2),B(201,2)
317      COMMON /BLC9/ DLS(200),VW(200),CF(200),THT(200)
318      integer ni,nx,np,nxt,is,itrans,isf,it,npt,ntr,ntflag,nstop
319      dimension rdiffflow(201),rdlow(201)
320      IF(NX.EQ.1 ) THEN
321          DLS(NX)= 0.0
322          THT(NX)= 0.0
323          CF(NX) = 0.0
324          VW(NX) = V(1,2)
325          rdiffflow=1000
326          nstop=0
327      ELSE
328          SQRX = SQRT(UE(NX)*X(NX)*RL)
329          CF(NX) = 2.0 * V(1,2) * B(1,2) /SQRX
330          VW(NX) = V(1,2)
331          DLS(NX)= X(NX)/SQRX * (ETA(NP)-F(NP,2))
332
333          U1 = U(1,2) * (1.0 -U(1,2))
334          SUM = 0.0
335          DO 20 J=2,NP
336              U2 = U(J,2) * (1.0 -U(J,2))
337              SUM = SUM + A(J) * (U1 + U2)
338              U1 = U2
339      20 CONTINUE
340          THT(NX)= X(NX)/SQRX * SUM
341          rex=UE(NX)*X(NX)*RL
342          rtheta=UE(NX)*THT(NX)*RL
343      *** Michels's Criterion (JUST CALCULATED **NEVER** USED)
344          rtrans=1.174*(1.0+22400.0/rex)*rex**0.46
345          rdiffflow=abs(rtheta-rtrans)
346          if ((NX.gt.2) .and. (rdlow(nx-1).eq.rdlow(nx-2))) then
347              if (rdlow(nx-2).eq.rdlow(nx-3)) nstop=1
348          endif
349          if (ISF.eq.2) then
350              if ((ntflag.eq.1) .and. (nstop.eq.0)) then
351                  transnew(ISF)=rex/(RL*UE(NX))
352                  ntflag=0
353              endif
354          endif
355          if((rdiffflow.LT.rdlow(nx) .and. (nstop.eq.0)) then
356              transnew(ISF)=rex/(RL*UE(NX))
357              rdiffflow=rdiffflow(nx)

```

```

358         ntflag=1
359     endif
360     rdlow(nx)=rdifflow
361 ENDIF
362 C     SHIFT PROFILES FOR THE NEXT STATION
363     ymark=.0005
364     DO 175 J=1,NPT
365         if (ISF.EQ.1) then
366             if (U(J,1).LT.(0.995)) then
367                 lasty=1
368                 yplot=ETA(J)*SQRT(X(NX)/(RL*UE(NX)))
369                 do nxloop=5,NI/2-1,5
370                     if (NX.EQ.nxloop) then
371                         markx=NX/5
372                         numw=markx+30
373                     c     write (numw,*) U(J,1)+markx,yplot
374                         write (54,*) U(J,1)+markx,yplot
375                         if (yplot.gt.ymark) then
376                             write (55,*) markx,ymark
377                             ydiff=yplot-yplotold
378                             udiff=U(J,1)-U(J-1,1)
379                             xvalue=U(J-1,1)+udiff*(ymark-yplotold)/ydiff
380                             write (55,*) xvalue+markx,ymark
381                             write (55,92)
382                         92     format (//)
383                             ymark=ymark+.0005
384                             if (yplot.GT.ymark) goto 91
385                         endif
386                     endif
387                 end do
388             else
389                 if (lasty.EQ.1) then
390                     lasty=0
391                     do m=1,2
392                         do nxloop=5,NI/2-1,5
393                             if (NX.EQ.nxloop) then
394                                 markx=NX/5
395                                 numw=markx+30
396                             c     write (numw,*) markx,yplot
397                                 write (54,*) markx,yplot
398                                 endif
399                             end do
400                             yplot=0.0
401                         end do
402                     endif
403                 endif
404             endif
405             yplotold=yplot
406             F(J,1) = F(J,2)
407             U(J,1) = U(J,2)
408             V(J,1) = V(J,2)
409             B(J,1) = B(J,2)
410 175     CONTINUE
411     RETURN
412     END
413
414 *****
415     SUBROUTINE SOLV3
416     COMMON /BLC2/ NX,NXT,NP,NPT,NTR,IT,ISF
417     COMMON /BLC7/ ETA(201),DETA(201),A(201)
418     COMMON /BLC8/ F(201,2),U(201,2),V(201,2),B(201,2)
419     COMMON /BLC9/ S1(201),S2(201),S3(201),S4(201),S5(201),S6(201),
420 +     S7(201),S8(201),R1(201),R2(201),R3(201),R4(201)
421     COMMON /BLC6/ DELF(201),DELU(201),DELV(201)
422     integer ntflag,ni,nx,np,nxt,is,itrans,isf,it,npt,ntr
423     DIMENSION A11(201),A12(201),A13(201),A14(201),
424 +     A21(201),A22(201),A23(201),A24(201)
425     A11(1)= 1.0
426     A12(1)= 0.0
427     A13(1)= 0.0
428     A21(1)= 0.0
429     A22(1)= 1.0
430     A23(1)= 0.0
431     G11 =-1.0
432     G12 =-A(2)
433     G13 = 0.0
434     G21 = S4(2)
435     G23 =-S2(2)/A(2)

```

```

436      G22 = G23+S6(2)
437      A11(2)= 1.0
438      A12(2)=-A(2)-G13
439      A13(2)= A(2)*G13
440      A21(2)= S3(2)
441      A22(2)= S5(2)-G23
442      A23(2)= S1(2)+A(2)*G23
443      R1(2)= R1(2)-(G11*R1(1)+G12*R2(1)+G13*R3(1))
444      R2(2)= R2(2)-(G21*R1(1)+G22*R2(1)+G23*R3(1))
445  C FORWARD SWEEP
446      DO 500 J=2,NP
447  C      IF(DEN .GT. 1E12) GO TO 11
448      DEN = (A13(J-1)*A21(J-1)-A23(J-1)*A11(J-1)-A(J)*
449      +      (A12(J-1)*A21(J-1)-A22(J-1)*A11(J-1)))
450  C 11      IF(DEN1 .GT. 1E12) GO TO 12
451      DEN1 = A22(J-1)*A(J)-A23(J-1)
452  C 12      PRINT*, 'DEN, DEN1=', DEN, DEN1
453      G11 = (A23(J-1)+A(J)*(A(J)*A21(J-1)-A22(J-1)))/DEN
454      G12 = -(A(J)*A(J)+G11*(A12(J-1)*A(J)-A13(J-1)))/DEN1
455      G13 = (G11*A13(J-1)+G12*A23(J-1))/A(J)
456      G21 = (S2(J)*A21(J-1)-S4(J)*A23(J-1)+A(J)*(S4(J)*
457      +      A22(J-1)-S6(J)*A21(J-1)))/DEN
458      G22 = (-S2(J)+S6(J)*A(J)-G21*(A(J)*A12(J-1)-A13(J-1)))/DEN1
459      G23 = G21*A12(J-1)+G22*A22(J-1)-S6(J)
460      A11(J)= 1.0
461      A12(J)=-A(J)-G13
462      A13(J)= A(J)*G13
463      A21(J)= S3(J)
464      A22(J)= S5(J)-G23
465      A23(J)= S1(J)+A(J)*G23
466      R1(J)= R1(J)-(G11*R1(J-1)+G12*R2(J-1)+G13*R3(J-1))
467      R2(J)= R2(J)-(G21*R1(J-1)+G22*R2(J-1)+G23*R3(J-1))
468      IF(R2(J) .GT. 1E20) THEN
469          RCHK = R2(J)
470          GO TO 99
471      END IF
472 500 CONTINUE
473  C BACKWARD SWEEP
474      DELU(NP) = R3(NP)
475      E1 = R1(NP)-A12(NP)*DELU(NP)
476      E2 = R2(NP)-A22(NP)*DELU(NP)
477      DELV(NP) = (E2*A11(NP)-E1*A21(NP))/(A23(NP)*A11(NP)-A13(NP)*
478      +      A21(NP))
479      DELF(NP) = (E1-A13(NP)*DELV(NP))/A11(NP)
480      DO 600 J = NP-1, 1, -1
481          E3 = R3(J)-DELU(J+1)+A(J+1)*DELV(J+1)
482          DEN2 = A21(J)*A12(J)*A(J+1)-A21(J)*A13(J)-A(J+1)*A22(J)*
483      +      A11(J)+A23(J)*A11(J)
484          DELV(J) = (A11(J)*(R2(J)+E3*A22(J))-A21(J)*R1(J)-E3*A21(J)*
485      +      A12(J))/DEN2
486          DELU(J) = -A(J+1)*DELV(J)-E3
487          DELF(J) = (R1(J)-A12(J)*DELU(J)-A13(J)*DELV(J))/A11(J)
488 600 CONTINUE
489 99      DO 700 J=1,NP
490          F(J,2)= F(J,2)+DELF(J)
491          U(J,2)= U(J,2)+DELU(J)
492          V(J,2)= V(J,2)+DELV(J)
493 700 CONTINUE
494      U(1,2)= 0.0
495
496      RETURN
497      END
498
499
500

```

APPENDIX B

A. UPOT.IN NAME LIST

```

3
*****
*      AIRFOIL TYPE : NACA 0012  (RAMP)
*****
IFLAG  NLOWER  NUPPER
  0      50      50
  12
IRAMP  IOSCIL  ALPI  ALPMAX  RFREQ  PIVOT
  1      0      0.0    30.0    0.01    0.25
IGUST  UGUST  VGUST
  0      0.      0.
ITRANS  DELHX  DELHY  PHASE
  0      0.00  0.00  0.00
CYCLE  NTCYCLE  TOL
  1.5    125    0.0001
naot & naot X aoa values multiplied by 10 (integer)
  8      1,60,90,110,130,150,170,200
*****
Comments...

IFLAG  0: NACA 4 or 5 digit airfoil (program computes coordinates)
       1: User supplies surface coordinates
           NLOWER: # panels upper surface
           NUPPER: # panels lower surface
           (NOTE: Next line entry is either a NACA 4,5 digit
                airfoil or user supplied coordinates (no blanks))

IRAMP  0: n/a          *** RFREQ is based on full chord
       2: Straight ramp *** RFREQ = A = Reduced pitch rate
       1: Modified ramp ***      =  $\alpha c/U$ 

IOSCIL 0: n/a          *** RFREQ is based on full chord
       1: Sinusoidal pitch, motion starts at min Aoa
           *** RFREQ = k = Reduced Frequency
           =  $\omega c/U$ 

ITRANS 0: n/a
       1: Translational harmonic oscillatio

CYCLE  : # of cycles for oscillatory motions
        in case of ramp, cycle=1.5 denotes airfoil is held
        at max aoa for the duration of .5 cycle

NTCYCLE: # of time steps for each cycle
        CYCLE*NTCYCLE is limited to 200 currently.

NAOT: # of input aoa for cp output
      - angles should be increasing order,
      - for oscillatory motions angles should increase
        first then decrease, decreasing angles are for
        return cycle..

```

```

3
*****
*      AIRFOIL TYPE : SSC-A09  (SINUSOID)
*****
IFLAG  NLOWER  NUPPER
  1      92      92
1.000000  .0004815
.975084  -.001325
.955144  -.002610
.935204  -.004290
.915264  -.006081
*
*
*
.915264  .009046
.935204  .006229

```



```

9      COMMON /POT/ PHI(nwmx),PHIK(nwmx)
10     COMMON /GUST/ UG(nwmx),VG(nwmx),XGF,UGUST,VGUST
11     COMMON /EXTV/ UE(nwmx)
12     COMMON /MAINout/ ialfao(20), naot, nao
13     COMMON /CPD/ CP(200),pivot
14     common /phase/ ta(nwmx),alphaa(nwmx),cla(nwmx),cda(nwmx),
15     > cma(nwmx),hya(nwmx)
16     DIMENSION XXC(nwmx),YYC(nwmx)
17     PI = 3.1415926585
18     open (unit=1,file='u.in',form='formatted')
19     open (unit=8,file='cl.d',form='formatted')
20     open (unit=9,file='cm.d',form='formatted')
21     open (unit=91,file='v.d',form='formatted')
22     open (unit=92,file='uout.d',form='formatted')
23 C..INPUT FROM FILE CODE 5 AND SET UP PANEL NODES AND SLOPES
24 CALL INDATA
25 CALL SETUP
26 read(1,*)
27 READ(1,*) IRAMP, IOSCIL, ALPI,ALPMAX,FREQ,PIVOT
28 read(1,*)
29 READ(1,*) IGUST, UGUST, VGUST
30 read(1,*)
31 READ(1,*) ITRANS,DELHX,DELHY,PHASE
32 read(1,*)
33 READ(1,*) cycle,ntcycle,TOL
34 read(1,*)
35 READ(1,*) naot, (ialfao(i), i=1,naot)
36 nao = 1
37 if(iramp.gt. 0 .or. ioscil .gt. 0) then
38   if(iramp.gt. 0) print*, ' RAMP MOTION, IRAMP      = ',iramp
39   if(ioscil .gt. 0) print*,
40   > ' OSCILLATORY MOTION, IOSCIL      = ',ioscil
41   WRITE (6,555) ALPI,alpmx,FREQ,PIVOT
42 555 FORMAT (2X, 'INITIAL ANGLE OF ATTACK      =',F10.4,/,
43  > 2X, 'FINAL ANGLE OF ATTACK      =',F10.4,/,
44  > 2X, 'REDUCED FREQ. FOR OSCIL      =',F10.4,/,
45  > 2X, 'PIVOT POINT      =',F10.4,/,
46  > 2X, '-----')
47   dalp = alpmx-alpi
48   ** tcon = 2(pi)/k
49   ** k = (w c) / 2 Vinf = (w A) / alfadot
50   ** alfadot = w = (2 u A) / c
51   tcon = 2.*pi/freq
52   endif
53   if(igust .eq. 1) then
54     WRITE (6,558) ugust, vgust
55 558 FORMAT (2X, 'STREAMWISE GUST VELOCITY      =',F10.4,/,
56  > 2X, 'PERPENDICULAR GUST VELOCITY      =',F10.4,/,
57  > 2X, '-----')
58     ANGLE = ALPI*PI/180. + ATAN(VGUST/(1.+UGUST))
59     COSAng = COS(ANGLE)
60     SINAng = SIN(ANGLE)
61   endif
62   if(itrans .eq. 1) then
63     WRITE (6,556) DELHX,DELHY,PHASE
64 556 FORMAT (2X, 'X AMPLITUDE OF TRANS OSCILL. =',F10.4,/,
65  > 2X, 'Y AMPLITUDE OF TRANS OSCILL. =',F10.4,/,
66  > 2X, 'PHASE OF TRANS OSCILL. =',F10.4,/,
67  > 2X, '-----')
68     tcon = 2.*pi/freq
69   endif
70   WRITE (6,557) cycle,ntcycle,TOL
71 557 FORMAT (2X, 'TOTAL # OF CYCLES      =',F10.4,/,
72  > 2X, '# of TIME STEPS PER CYCLE      =',i6,/,
73  > 2X, 'TOLERANCE FOR CONVERGENCE      =',F10.4,/,
74  > 2X, '-----')
75   dts = tcon/float(ntcycle)
76   ** float() >> Numeric to real
77 C.STEADY FLOW CALCULATION AT ALPI
78   ALPHA = ALPI
79   WRITE (6,1030) ALPHA
80 1030 FORMAT (//,' STEADY FLOW SOLUTION AT ALPHA = ',F10.6,/)
81   WRITE (6,1032)
82 1032 format (//,4x,'I X MID Q(I) GAMMA CP(I) UE(I) ',/)
83   IF (ALPHA .GT. 90.) stop "Alpha .gt. 90 degrees"
84   COSALF = COS(ALPHA*PI/180.)
85   SINALF = SIN(ALPHA*PI/180.)
86   CALL COFISH(SINALF,COSALF)

```

```

87      CALL INFL (0)
88      CALL GAUSS(1,0,0)
89      CALL VELDIS(SINALF,COSALF)
90      CALL PINTEG (GAMMA,Q)
91      CALL PRESSs
92      CALL FANDM(SINALF,COSALF,cl,cd,cm
93      write(6,1031)cl,cd,cm
94 1031 format (/,
95      + 5x,'Cl = ',f10.6,/,
96      + 5x,'Cd = ',f10.6,/,
97      + 5x,'Cm = ',f10.6,/)
98      if(cycle.le. 0.) stop " Steady solution only"
99 C..INITIALIZATION FOR UNSTEADY FLOW CALCULATION
100      HX      = 0.0
101      HY      = 0.0
102      HXO     = 0.0
103      HYO     = 0.0
104      DHX     = 0.0
105      DHY     = 0.0
106      UX      = 0.0
107      UY      = 0.0
108      ALP = ALPI
109      DA      = 0.0
110      COSDA   = 1.0
111      SINDA   = 0.0
112      OMEGA   = 0.0
113      XGF     = 0.0
114      PHA     = PHASE*PI/180.
115      VXW     = COSALF
116      VYW     = SINALF
117      GAMK    = GAMMA
118      T       = 0.0
119      TOLD    = 0.0
120      DT      = DTS
121      TD      = DTS
122      WRITE (6,1051)
123 1051 FORMAT (////, ' *****',/,
124      + ' *** BEGIN UNSTEADY FLOW SOLUTION ****',/,
125      + ' *****',/,
126      > ' istep alpha time nitr cl cd cm'//)
127      ntmx = min(nwmx, int(cycle*ntcycle) )
128      Tmax for Ramp motion
129      **      Tmax = tcon/ntcycle*(ntmx-1.)
130      DO NT = 1,ntmx
131      ta(nt) = t
132 C.. STORE CORE VORTEX COORDINATES FOR TIME STEP ADJUSTMENTS
133      if (nt .ne. 1) then
134      DO 51 I = 1,nt-1
135      XXC(I) = XC(I)
136      51 YYC(I) = YC(I)
137      endif
138      IF (IRAMP .eq. 1) then
139 C. modified ramp change in aoa
140      if (t .le. tcon) then
141      DAL = DALP * (3.-2.*T/TCON)*(T/TCON)**2
142      ALPHA = ALPI + DAL
143      COSALF = COS(ALPHA*PI/180.)
144      SINALF = SIN(ALPHA*PI/180.)
145      DA = ALPHA - ALP
146      COSDA = COS(DA*PI/180.)
147      SINDA = SIN(DA*PI/180.)
148      OMEGA = - (DALP*PI/180.) * (6.*T/(TCON*TCON)) * (1.-T/TCON)
149      DHX = PIVOT * (1.-COSDA)
150      DHY = - PIVOT * SINDA
151      UY = PIVOT * OMEGA
152      else
153      DAL = 0.0
154      ALPHA = ALPmax
155      COSALF = COS(ALPHA*PI/180.)
156      SINALF = SIN(ALPHA*PI/180.)
157      DA = 0.0
158      COSDA = 1.0
159      SINDA = 0.0
160      OMEGA = 0.0
161      DHX = 0.0
162      DHY = 0.0
163      UY = 0.0
164      endif

```

```

165      ELSEIF (IRAMP .eq. 2) then
166      C.straight ramp change in aoa
167          if (t .le. tcon) then
168              alpha = alpi + dalp/tcon*t
169              COSALF = COS(ALPHA*PI/180.)
170              SINALF = SIN(ALPHA*PI/180.)
171              DA = ALPHA - ALP
172              COSDA = COS(DA*PI/180.)
173              SINDA = SIN(DA*PI/180.)
174              OMEGA = - dalp/tcon*(pi/180.)
175              DHX = PIVOT * (1.-COSDA)
176              DHY = - PIVOT * SINDA
177              UY = PIVOT * OMEGA
178          else
179              DAL = 0.0
180              ALPHA = ALPmax
181              COSALF = COS(ALPHA*PI/180.)
182              SINALF = SIN(ALPHA*PI/180.)
183              DA = 0.0
184              COSDA = 1.0
185              SINDA = 0.0
186              OMEGA = 0.0
187              DHX = 0.0
188              DHY = 0.0
189              UY = 0.0
190          endif
191      ELSEIF (Ioscil .eq. 1) then
192      C..rotational harmonic oscillation
193      c      DAL = DALP*SIN(FREQ*T)
194      c      OMEGA = - (DALP*PI/180.) * FREQ * COS(FREQ*T)
195      c      ALPHA = ALPI + DAL
196      alpha = alpi + 0.5*dalp*(1.- cos(freq*t) )
197      omega = - 0.5*(dalp*pi/180.)*freq*sin(freq*t)
198      COSALF = COS(ALPHA*PI/180.)
199      SINALF = SIN(ALPHA*PI/180.)
200      DA = ALPHA - ALP
201      COSDA = COS(DA*PI/180.)
202      SINDA = SIN(DA*PI/180.)
203      UY = PIVOT * OMEGA
204      DHX = PIVOT * (1.-COSDA)
205      DHY = - PIVOT * SINDA
206      ELSEIF (Igust .eq. 1) then
207      C..sharp edge gust (ugust and/or vgust)
208      XGF = T
209      DO 110 IG = 1,NODTOT
210      UG(IG) = 0.0
211      VG(IG) = 0.0
212      XG = X(IG)*COSALF + Y(IG)*SINALF
213      XGP1 = X(IG+1)*COSALF + Y(IG+1)*SINALF
214      IF (IG .LT. NLOWER+1) GO TO 120
215      IF (XGF .LE. XG) GO TO 110
216      IF (XGF .GE. XGP1) GO TO 111
217      FAC = (XGF - XG)/(XGP1 - XG)
218      UG(IG) = UGUST*FAC
219      VG(IG) = VGUST*FAC
220      GO TO 110
221      111 UG(IG) = UGUST
222      VG(IG) = VGUST
223      GO TO 110
224      120 IF (XGF .LE. XGP1) GO TO 110
225      IF (XGF .GE. XG) GO TO 121
226      FAC = (XGF - XGP1)/(XG - XGP1)
227      UG(IG) = UGUST*FAC
228      VG(IG) = VGUST*FAC
229      GO TO 110
230      121 UG(IG) = UGUST
231      VG(IG) = VGUST
232      110 CONTINUE
233      IF (XGF .LE. COSALF) MGUST = nt
234      ENDIF
235      alphaa(nt) = alpha
236      if (Itrans .eq. 1) then
237      C..translation harmonic oscillation
238      HX = DELHX * SIN(FREQ*T + PHA)
239      HY = DELHY * SIN(FREQ*T)
240      c      HX = --DELHX * COS(FREQ*T + PHA)
241      c      HY = --DELHY * COS(FREQ*T)
242      DHX = HX - HXO

```

```

243      DHY      = HY - HYO
244      UX      = DELHX*FREQ*COS(FREQ*T+PHA)
245      UY      = DELHY*FREQ*COS(FREQ*T)
246      c      UX      = DELHX*FREQ*SIN(FREQ*T+PHA)
247      c      UY      = DELHY*FREQ*SIN(FREQ*T)
248      hya(nt) = hy
249      endif
250      C..TRANSFORM CORE VORTEX COORDINATES W. R. T. NEW AIRFOIL POSITION
251      IF (nt .ne. 1) then
252      DO 90 I = 1,nt-1
253      XC(I) = XKC(I) + CVVX(I) * DT
254      YC(I) = YKC(I) + CVVY(I) * DT
255      XCO = XC(I)
256      YCO = YC(I)
257      XC(I) = XCO*COSDA - YCO*SINDA + DHX
258      90 YC(I) = XCO*SINDA + YCO*COSDA + DHY
259      endif
260      C..CALCULATE THE TRAILING EDGE WAKE ELEMENT
261      NITR = 0
262      10 WAKE = SQRT(VYW*VYW+VXW*VXW)*DT
263      THENP1 = ATAN2(VYW,VXW)
264      COSTHE(NP1) = COS(THENP1)
265      SINTHE(NP1) = SIN(THENP1)
266      X(NP2) = X(NP1) + WAKE*COSTHE(NP1)
267      Y(NP2) = Y(NP1) + WAKE*SINTHE(NP1)
268      CALL INFL(NITR)
269      CALL COEF(SINALF,COSALF,OMEGA,UX,UY,NITR)
270      CALL GAUSS(2,nt,NITR)
271      CALL KUTTA(ALPHA,SINALF,COSALF,OMEGA,UX,UY)
272      CALL TEWAK(SINALF,COSALF)
273      TOL1 = ABS(VYW - VYWK)
274      TOL2 = ABS(VXW - VXWK)
275      IF ((TOL1 .LT. TOL) .AND. (TOL2 .LT. TOL)) GO TO 20
276      VYW = VYWK
277      VXW = VXWK
278      NITR = NITR + 1
279      GO TO 10
280      20 continue
281      CALL PRESS(SINALF,COSALF,OMEGA,UX,UY,ALPHA)
282      CALL PINTEG(GAMK,QK)
283      if( igust .eq. 1) then
284      CALL FANDM(SINAng,COSAng,cl,cd,cm)
285      else
286      CALL FANDM(SINALF,COSALF,cl,cd,cm)
287      endif
288      cla(nt) = cl
289      cda(nt) = cd
290      cma(nt) = cm
291      ttt = ((nt-1.)*dts)/((ntmax-1.)*dts)
292      IF (IRAMP .GT. 0) THEN
293      write(6,1011) nt, alpha, ttt+.125, nitr, cl,cd,cm
294      write(92,1011) nt, alpha, ttt+.125, nitr, cl,cd,cm
295      ELSE
296      write(6,1011) nt, alpha, ttt, nitr, cl,cd,cm
297      write(92,1011) nt, alpha, ttt, nitr, cl,cd,cm
298      ENDIF
299      1011 FORMAT( i4, 2x, f7.4, 2x, f9.6, 3x, i2, 3x, 3f10.6 )
300      write(8,'(2f10.5)') alpha, cl
301      write(9,'(2f10.5)') alpha, cm
302      C..WAKE ELEMENT LEAVES TRAILING EDGE AS A CORE-VORTEX
303      CV(nt) = SS*(GAMMA-GAMK)
304      XC(nt) = X(NP1) + 0.5*WAKE*COSTHE(NP1)
305      YC(nt) = Y(NP1) + 0.5*WAKE*SINTHE(NP1)
306      CVVX(nt) = VXW
307      CVVY(nt) = VYW
308      CALL CORVOR(SINALF,COSALF)
309      C..RE-INITIALISE PARAMETERS FOR NEXT TIME STEP CALCULATION
310      DO 30 I = 1,NODTOT
311      Q(I) = QK(I)
312      PHI(I) = PHIK(I)
313      30 CONTINUE
314      GAMMA = GAMK
315      ALP = ALPHA
316      HXO = HX
317      HYO = HY
318      TOLD = T
319      DT = TD
320      T = T + TD

```

```

321      ENDDO
322      if(ioscil .eq. 1 .or. itrans .eq. 1 )
323      > call PHAZ(ntmax,ntcycle,freq,itrans,alpl)
324      STOP " normal stop "
325      END
326      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
327      C      SUBROUTINE BODY(Z,SIGN,X,Y)
328      C      RETURN COORDINATES OF POINT ON THE BODY SURFACE
329      C
330      C      Z = NODE-SPACING PARAMETER
331      C      X,Y = CARTESIAN COORDINATES
332      C      SIGN = +1. FOR UPPER SURFACE
333      C      -1. FOR LOWER SURFACE
334      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
335      C      SUBROUTINE BODY(Z,SIGN,X,Y)
336      C      COMMON /PAR/ NACA,TAU,EPSSMAX,PTMAX
337      C      IF (SIGN .LT. 0.0) Z = 1. - Z
338      C      CALL NACA45(Z,THICK,CAMBER,BETA)
339      C      X = Z - SIGN*THICK*SIN(BETA)
340      C      Y = CAMBER + SIGN*THICK*COS(BETA)
341      C      RETURN
342      C      END
343      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
344      C      SUBROUTINE COEF (SINALF,COSALF,OMEGA,UX,UY,NITR)
345      C
346      C      SET COEFFICIENTS OF N EQS ARISING FROM FLOW
347      C      TANGENCY CONDITIONS AT MID POINTS OF PANELS
348      C      SOLVING THE N-SOURCE STRENGTHS IN TERMS OF THE
349      C      VORTICITY STRENGTH (RESULTING IN 2 RHS)
350      C      KUTTA CONDITION IS SATISFIED SEPARATELY TO OBTAIN
351      C      THE VORTICITY STRENGTH
352      C      THIS SOLUTION METHOD IS DESIRED FOR UNSTEADY FLOW
353      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
354      C      SUBROUTINE COEF (SINALF,COSALF,OMEGA,UX,UY,NITR)
355      C      COMMON /BOD/ IFLAG,NLOWER,NUPPER,NODTOT,X(202),Y(202),
356      C      + COSTHE(201),SINTHE(201),SS,NP1,NP2
357      C      COMMON /COF/ A(201,211),NEQS
358      C      COMMON /SING/ Q(200),GAMMA,QK(200),GAMK
359      C      COMMON /WAK/ VYW,VXW,WAKE,DT
360      C      COMMON /CORV/ CV(200),XC(200),YC(200),M,TD,CCVX(200),CCVY(200)
361      C      COMMON /INF1/ AAN(201,201),BBN(201,201),AYNP1(201),BYNP1(201)
362      C      COMMON /INF2/ SUMCCN(201),SUMCCT(201),CYNP1(200),CXNP1(200)
363      C      COMMON /GUST/ UG(200),VG(200),XGF,UGUST,VGUST
364      C      NEQS = NODTOT
365      C      NP1 = NODTOT + 1
366      C      NP2 = NODTOT + 2
367      C      INITIALISE COEFFICIENTS
368      C      IF ((M .GT. 1) .OR. (NITR .GT. 0)) GO TO 91
369      C      DO 90 I = 1,NODTOT
370      C      DO 90 J = 1,NP2
371      C      90 A(I,J) = 0.0
372      C      91 CONTINUE
373      C      SET LHS MATRIX A(I,J)
374      C      DO 120 I = 1,NODTOT
375      C      XMID = 0.5 * (X(I) + X(I+1))
376      C      YMID = 0.5 * (Y(I) + Y(I+1))
377      C      B = 0.0
378      C      DO 110 J = 1,NODTOT
379      C      IF ((M .EQ. 1) .AND. (NITR .EQ. 0)) A(I,J) = AAN(I,J)
380      C      B = B + BBN(I,J)
381      C      110 CONTINUE
382      C      110 CONTINUE
383      C      FILL IN THE RIGHT HAND SIDE
384      C      A(I,NP1) = -B + BBN(I,NP1)*SS/WAKE
385      C      A(I,NP2) = -BBN(I,NP1)*GAMMA*SS/WAKE
386      C      + + SINTHE(I) * ((1.+UG(I))*COSALF-VG(I)*SINALF+UX)
387      C      + - COSTHE(I) * ((1.+UG(I))*SINALF+VG(I)*COSALF+UY)
388      C      + + OMEGA*(YMID*SINTHE(I) + XMID*COSTHE(I))
389      C      ADD CORE VORTEX CONTRIBUTION
390      C      IF (M .EQ. 1) GOTO 140
391      C      A(I,NP2) = A(I,NP2) - SUMCCN(I)
392      C      140 CONTINUE
393      C      120 CONTINUE
394      C      RETURN
395      C      END
396
397      C      SUBROUTINE COFISH(SINALF,COSALF)
398

```

```

399      C          SET COEFFICIENTS OF LINEAR SYSTEM - N+1 EQUATIONS      C
400      C          N EQUS - FLOW TANGENCY AT MID POINTS OF PANELS      C
401      C          1 EQU - KATTA CONDITION AT TRAILING EDGE PANELS      C
402      C          THIS SOLUTION METHOD IS EFFECTIVE FOR STEADY FLOW, NO      C
403      C          ITERATION IS REQUIRED, N-SOURCE STRENGTHS AND 1      C
404      C          VORTICITY STRENGTH ARE SOLVED SIMULTANEOUSLY      C
405      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
406      SUBROUTINE COFISH(SINALF,COSALF)
407      COMMON /BOD/ IFLAG,NLOWER,NUPPER,NODTOT,X(202),Y(202),
408      + COSTHE(201),SINTHE(201),SS,NP1,NP2
409      COMMON /COF/ A(201,211),KUTTA
410      COMMON /NUM/ PI,PI2INV
411      KUTTA = NODTOT + 1
412      C          INITIALISE COEFFICIENTS
413      DO 90 J = 1,KUTTA
414      90 A(KUTTA,J) = 0.0
415      C          SET VN = 0 AT MID-POINT OF I-TH PANEL
416      DO 120 I = 1,NODTOT
417      XMID = .5*(X(I) + X(I+1))
418      YMID = .5*(Y(I) + Y(I+1))
419      A(I,KUTTA) = 0.0
420      C          -- FIND CONTRIBUTION OF J-TH PANEL
421      DO 110 J = 1,NODTOT
422      FLOG = 0.0
423      FTAN = PI
424      IF (J.EQ. I) GO TO 100
425      DXJ = XMID - X(J)
426      DXJP = XMID - X(J+1)
427      DYJ = YMID - Y(J)
428      DYJP = YMID - Y(J+1)
429      FLOG = .5*ALOG((DXJP*DXJP+DYJP*DYJP)/(DXJ*DXJ+DYJ*DYJ))
430      FTAN = ATAN2(DYJP*DXJ-DXJP*DYJ,DXJP*DXJ+DYJP*DYJ)
431      100 CTIMTJ = COSTHE(I)*COSTHE(J) + SINTHE(I)*SINTHE(J)
432      STIMTJ = SINTHE(I)*COSTHE(J) - COSTHE(I)*SINTHE(J)
433      A(I,J) = PI2INV*(FTAN*CTIMTJ + FLOG*STIMTJ)
434      B = PI2INV*(FLOG*CTIMTJ - FTAN*STIMTJ)
435      A(I,KUTTA) = A(I,KUTTA) + B
436      IF ((I.GT. 1).AND. (I.LT. NODTOT))GO TO 110
437      C          -- IF I-TH PANEL TOUCHES TRAILING EDGE,
438      C          ADD CONTRIBUTION TO KUTTA CONDITION
439      A(KUTTA,J) = A(KUTTA,J) - B
440      A(KUTTA,KUTTA) = A(KUTTA,KUTTA) + A(I,J)
441      110 CONTINUE
442      C          FILL IN KNOWN SIDES
443      A(I,KUTTA+1) = SINTHE(I)*COSALF - COSTHE(I)*SINALF
444      120 CONTINUE
445      A(KUTTA,KUTTA+1) = - (COSTHE(1) + COSTHE(NODTOT))*COSALF
446      + - (SINTHE(1) + SINTHE(NODTOT))*SINALF
447      RETURN
448      END
449      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
450      C          SUBROUTINE CORVOR (SINALF,COSALF)      C
451      C          COMPUTE THE LOCAL VELOCITIES OF CORE VORTICES      C
452      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
453      SUBROUTINE CORVOR (SINALF,COSALF)
454      COMMON /BOD/ IFLAG,NLOWER,NUPPER,NODTOT,X(202),Y(202),
455      + COSTHE(201),SINTHE(201),SS,NP1,NP2
456      COMMON /SING/ Q(200),GAMMA,QK(200),GAMK
457      COMMON /WAK/ VYW,VXW,WAKE,DT
458      COMMON /CORV/ CV(200),XC(200),YC(200),M,TD,CCVX(200),CCVY(200)
459      COMMON /POT/ PHI(200),PHIK(200)
460      COMMON /GUST/ UG(200),VG(200),XGF,UGUST,VGUST
461      COMMON /NUM/ PI,PI2INV
462      IF (M.EQ.1) GOTO 40
463      MM1 = M - 1
464      C          VELOCITY COMPONENTS OF CORE VORTICES AT CURRENT TIME STEP
465      UGC = 0.0
466      VGC = 0.0
467      DO 10 N = 1,MM1
468      XG = XC(N)*COSALF + YC(N)*SINALF
469      IF (XG.GT. XGF) GO TO 5
470      UGC = UGUST
471      VGC = VGUST
472      5 CONTINUE
473      VY = (1.+UGC)*SINALF+VGC*COSALF
474      VX = (1.+UGC)*COSALF-VGC*SINALF
475      XMID = XC(N)
476      YMID = YC(N)

```

Appendix B

```

555 C          NRHS      = NUMBER OF RIGHT HAND SIDES                      C
556 C          RIGHT-HAND SIDES AND SOLUTIONS STORED IN                    C
557 C          COLUMNS NEQNS+1 THRU NEQNS+NRHS OF (A)                      C
558 C          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
559 C          SUBROUTINE GAUSS(NRHS,M,NITR)
560 C          COMMON /COF/ A(201,211),NEQNS
561 C          NP      = NEQNS + 1
562 C          NTOT    = NEQNS + NRHS
563 C          IF ((M .GT. 1) .OR. (NITR .GT. 0)) GO TO 160
564 C          GAUSS REDUCTION
565 C          DO 150 I = 2,NEQNS
566 C          IM      = I - 1
567 C          ELIMINATE (I-1)TH UNKNOWN FROM
568 C          ITH THRU (NEQNS)TH EQUATIONS
569 C          DO 150 J = I,NEQNS
570 C          R      = A(J,IM)/A(IM,IM)
571 C          DO 150 K = I,NTOT
572 C          150 A(J,K) = A(J,K) - R*A(IM,K)
573 C          GO TO 170
574 C          GAUSSIAN ELIMINATION ON ONLY THE RIGHT-HAND-SIDES
575 C          160 DO 180 I = 2,NEQNS
576 C          IM      = I - 1
577 C          DO 180 J = I,NEQNS
578 C          R      = A(J,IM)/A(IM,IM)
579 C          DO 180 K = NP,NTOT
580 C          180 A(J,K) = A(J,K) - R*A(IM,K)
581 C          170 CONTINUE
582 C          BACK SUBSTITUTION
583 C          DO 220 K = NP,NTOT
584 C          A(NEQNS,K) = A(NEQNS,K)/A(NEQNS,NEQNS)
585 C          DO 210 L = 2,NEQNS
586 C          I      = NEQNS + 1 - L
587 C          IP      = I + 1
588 C          DO 200 J = IP,NEQNS
589 C          200 A(I,K) = A(I,K) - A(I,J)*A(J,K)
590 C          210 A(I,K) = A(I,K)/A(I,I)
591 C          220 CONTINUE
592 C          RETURN
593 C          END
594 C          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
595 C          SUBROUTINE INDATA                                          C
596 C          SET PARAMETERS OF BODY SHAPE                              C
597 C          FLOW SITUATION, AND NODE DISTRIBUTION                      C
598 C          USER MUST INPUT                                            C
599 C          NLOWER = NUMBER OF NODES ON LOWER SURFACE                C
600 C          NUPPER = NUMBER OF NODES ON UPPER SURFACE                 C
601 C          PLUS DATA ON BODY AND SUBROUTINE BODY                    C
602 C          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
603 C          SUBROUTINE INDATA
604 C          DIMENSION TITLE(20)
605 C          COMMON /BOD/ IFLAG,NLOWER,NUPPER,NODTOT,X(202),Y(202),
606 C          + COSTHE(201),SINTHE(201),SS,NP1,NP2
607 C          COMMON /PAR/ NACA,TAU,EPSMAX,PTMAX
608 C          READ (1,*) ITITLE
609 C          WRITE (6,*) ITITLE
610 C          DO 10 I = 1,ITITLE
611 C          READ (1,502) TITLE
612 C          10 WRITE (6,503) TITLE
613 C          501 FORMAT(3I5)
614 C          502 FORMAT(20A4)
615 C          503 FORMAT(1X,20A4)
616 C          read(1,*)
617 C          READ (1,*) IFLAG,NLOWER,NUPPER
618 C          WRITE (6,558) IFLAG,NLOWER,NUPPER
619 C          558 FORMAT (///2X, '-----',/,
620 C          1 2X, 'IFLAG (0:NACA, 1:INPUT) =', ,I5,/,
621 C          2 2X, 'NO. PANELS UPPER SURFACE =', ,I5,/,
622 C          3 2X, 'NO. PANELS LOWER SURFACE =', ,I5,/,
623 C          4 2X, '-----')
624 C          IF (IFLAG .NE. 0) RETURN
625 C          read(1,*)
626 C          READ (1,*) NACA
627 C          WRITE (6,501) NACA
628 C          IEPS = NACA/1000
629 C          IPTMAX = NACA/100 - 10*IFPS
630 C          ITAU = NACA - 1000*IEPS - 100*IPTMAX
631 C          EPSMAX = IEPS*0.01
632 C          PTMAX = IPTMAX*0.1

```

Appendix B

Appendix B

Appendix B

```

945      XLP      = -10.0 * (1.0 - COS(0.5*PI*FRACT))
946      DELX     = XL - XLP
947      XMID     = 0.5*(XL+XLP)*COSALF
948      YMID     = 0.5*(XL+XLP)*SINALF
949      XL       = XLP
950      VELX     = UGUST
951      C        ADD CONTRIBUTION OF J-TH PANEL
952      DO 20    J = 1,NP1
953      DXJ      = XMID - X(J)
954      DXJP     = XMID - X(J+1)
955      DYJ      = YMID - Y(J)
956      DYJP     = YMID - Y(J+1)
957      FLOG     = .5*ALOG((DXJP*DXJP+DYJP*DYJP)/(DXJ*DXJ+DYJ*DYJ))
958      FTAN     = ATAN2(DYJP*DXJ-DXJP*DYJ,DXJP*DXJ+DYJP*DYJ)
959      CALMTJ   = -COSALF*COSTHE(J) - SINALF*SINTHE(J)
960      SALMTJ   = -SINALF*COSTHE(J) + COSALF*SINTHE(J)
961      APY      = PI2INV*(FTAN*CALMTJ + FLOG*SALMTJ)
962      BPY      = PI2INV*(FLOG*CALMTJ - FTAN*SALMTJ)
963      IF (J.EQ. NP1) GO TO 40
964      VELX     = VELX - BPY*QK(J) + GAMK*APY
965      GO TO 20
966      40      VELX = VELX + SS*APY*(GAMMA-GAMK)/WAKE
967      20      CONTINUE
968      C        ADD CORE VORTEX CONTRIBUTION
969      IF (M.EQ. 1) GO TO 50
970      MM1      = M - 1
971      DO 60    N = 1,MM1
972      DX       = XMID - XC(N)
973      DY       = YMID - YC(N)
974      DIST     = SQRT(DX*DX+DY*DY)
975      COSTHN   = DX/DIST
976      SINTHN   = DY/DIST
977      SALMTN   = -SINALF*COSTHN + COSALF*SINTHN
978      CPT      = -PI2INV*SALMTN/DIST
979      60      VELX = VELX + CPT*CV(N)
980      50      CONTINUE
981      PINK     = PINK + VELX * DELX
982      30      CONTINUE
983      C        COMPUTE DISTURBANCE POTENTIAL AT MID-POINT OF I-TH PANEL
984      C        LOWER SURFACE
985      DO 230   I = 1,NLOWER
986      PH       = -PINK
987      DO 240   J = 1,NLOWER
988      240     PH = PH - PHIK(J)
989      PHIK(I) = PH
990      230     CONTINUE
991      8850     PHILOW = PHIK(1)
992      DO 270   I = 1,NLOWER-1
993      PHIK(I) = 0.5*(PHIK(I) + PHIK(I+1))
994      270     CONTINUE
995      PHIK(NLOWER) = 0.5*(PHIK(NLOWER) - PINK)
996      C        UPPER SURFACE
997      DO 250   I = NODTOT,NLOWER+1,-1
998      PH       = -PINK
999      DO 260   J = NLOWER+1,I
1000     260    PH = PH + PHIK(J)
1001     PHIK(I) = PH
1002     250    CONTINUE
1003     8851    PHIUPP = PHIK(NODTOT)
1004     DO 280   I = NODTOT,NLOWER+2,-1
1005     PHIK(I) = 0.5*(PHIK(I) + PHIK(I-1))
1006     280     CONTINUE
1007     PHIK(NLOWER+1) = 0.5*(PHIK(NLOWER+1) - PINK)
1008     8871    DPHITE = (PHIUPP-PHILOW)/SS
1009     8872    DPHIMP = (PHIK(NODTOT)-PHIK(1))/SS
1010     DO 290   I = 1,NODTOT
1011     290     CP(I) = CP(I) - 2.*(PHIK(I)-PHI(I))/DT
1012     IF ( ( ialfao(nao) .gt. ialfao(nao-1) .and.
1013     >    alpha .ge. float(ialfao(nao))/10.) .OR.
1014     >    ( ialfao(nao) .lt. ialfao(nao-1) .and.
1015     >    alpha .le. float(ialfao(nao))/10.) ) then
1016     itn = ialfao(nao)
1017     i3 = itn/100 + 1
1018     i2 = (itn - (i3-1)*100)/10 + 1
1019     i1 = (itn - (i3-1)*100 - (i2-1)*10) + 1
1020     IF ( ialfao(nao) .lt. ialfao(nao-1) ) then
1021     filnq = 'cpd'//alpn(i3:i3)//alpn(i2:i2)//alpn(i1:i1)///'.d'
1022     else

```

Appendix B

[illegible]

```

1257 C *** PRECISE CONTOUR INTEGRATION ****
1258 SUMC = 0.0
1259 DO 8000 I = 1,NODTOT
1260 DX = (X(I+1) - X(I))
1261 DY = (Y(I+1) - Y(I))
1262 DIST = SQRT(DX*DX+DY*DY)
1263 DO 8000 K = 1,6
1264 VTANG = 0.0
1265 DO 8100 J = 1,NODTOT
1266 8100 VTANG = VTANG - BBNP(I,J,K)*Q(J) + AANP(I,J,K)*GAMMA
1267 8000 SUMC = SUMC + VTANG*DIST*WGHT(K)
1268 C *** DUMMY INTEGRATION ****
1269 SUM = 0.0
1270 C INTEGRATION FROM TRAILING EDGE TO (1.0,-0.1)
1271 XMID = 1.0
1272 Y1 = 0.0
1273 DO 9100 K=1,10
1274 Y2 = -FLOAT(K)/100.
1275 YMID = 0.5*(Y1+Y2)
1276 DELY = Y2-Y1
1277 SUM1 = 0.0
1278 DO 9000 J = 1,NODTOT
1279 DXJ = XMID - X(J)
1280 DXJP = XMID - X(J+1)
1281 DYJ = YMID - Y(J)
1282 DYJP = YMID - Y(J+1)
1283 FLOG = .5*ALOG((DXJP*DXJP+DYJP*DYJP)/(DXJ*DXJ+DYJ*DYJ))
1284 FTAN = ATAN2(DYJP*DXJ-DXJP*DYJ,DXJP*DXJ+DYJP*DYJ)
1285 APY = PI2INV*(FTAN*COSTHE(J) - FLOG*SINTHE(J))
1286 BPY = PI2INV*(FLOG*COSTHE(J) + FTAN*SINTHE(J))
1287 9000 SUM1 = SUM1 + APY*Q(J) + BPY*GAMMA
1288 SUM = SUM + SUM1*DELY
1289 Y1 = Y2
1290 9100 CONTINUE
1291 C INTEGRATION FROM (1.0,-0.1) TO (-0.1,-0.1)
1292 YMID = -0.1
1293 X1 = 1.0
1294 DO 9200 K=1,100
1295 X2 = 1.0-1.1*FLOAT(K)/100.
1296 XMID = 0.5*(X1+X2)
1297 DELX = X2-X1
1298 SUM1 = 0.0
1299 DO 9250 J = 1,NODTOT
1300 DXJ = XMID - X(J)
1301 DXJP = XMID - X(J+1)
1302 DYJ = YMID - Y(J)
1303 DYJP = YMID - Y(J+1)
1304 FLOG = .5*ALOG((DXJP*DXJP+DYJP*DYJP)/(DXJ*DXJ+DYJ*DYJ))
1305 FTAN = ATAN2(DYJP*DXJ-DXJP*DYJ,DXJP*DXJ+DYJP*DYJ)
1306 APY = PI2INV*(FTAN*COSTHE(J) - FLOG*SINTHE(J))
1307 BPY = PI2INV*(FLOG*COSTHE(J) + FTAN*SINTHE(J))
1308 9250 SUM1 = SUM1 - BPY*Q(J) + APY*GAMMA
1309 X1 = X2
1310 SUM = SUM + SUM1*DELX
1311 9200 CONTINUE
1312 C INTEGRATION FROM (-0.1,-0.1) TO (-0.1,0.1)
1313 XMID = -0.1
1314 Y1 = -0.1
1315 DO 9300 K=1,20
1316 Y2 = -0.1+FLOAT(K)/100.
1317 YMID = 0.5*(Y1+Y2)
1318 DELY = Y2-Y1
1319 SUM1 = 0.0
1320 DO 9350 J = 1,NODTOT
1321 DXJ = XMID - X(J)
1322 DXJP = XMID - X(J+1)
1323 DYJ = YMID - Y(J)
1324 DYJP = YMID - Y(J+1)
1325 FLOG = .5*ALOG((DXJP*DXJP+DYJP*DYJP)/(DXJ*DXJ+DYJ*DYJ))
1326 FTAN = ATAN2(DYJP*DXJ-DXJP*DYJ,DXJP*DXJ+DYJP*DYJ)
1327 APY = PI2INV*(FTAN*COSTHE(J) - FLOG*SINTHE(J))
1328 BPY = PI2INV*(FLOG*COSTHE(J) + FTAN*SINTHE(J))
1329 9350 SUM1 = SUM1 + APY*Q(J) + BPY*GAMMA
1330 SUM = SUM + SUM1*DELY
1331 Y1 = Y2
1332 9300 CONTINUE
1333 C INTEGRATION FROM (-0.1,0.1) TO (1.0,0.1)
1334 YMID = 0.1

```



```

1413 C READ POSITION DATA
1414 IF(itrans.EQ. 1) THEN
1415 DO I=1,NPTS
1416 alpha(I) = HY(I)
1417 END DO
1418 zero = .00001
1419 ELSE
1420 zero = .01
1421 END IF
1422 CALL AMPLITUDE(DAT,AMP,NPTS,J)
1423 C..DETERMINE PHASE SHIFT
1424 PHI = 0.
1425 ERR = 10000.
1426 CN = -2.0
1427 itter = 500
1428 icount = 0
1429 phi = cn*pi/180.0
1430 nts = npts - 3*ntcycle/4
1431 nte = npts - ntcycle/4
1432 C..BEGIN ITERATION TO CONVERGENCE
1433 30 icount = icount + 1
1434 SUM = 0
1435 DO I = nts, nte
1436 FN(I) = -AMP(J)*cos(W*T(I) + PHI )
1437 R(I) = ABS(FN(I) - DAT(I))
1438 SUM = SUM + R(I)
1439 END DO
1440 c print*, 'icount, phi, cn err :',icount,phi*180./pi,cn,err
1441 IF(sum.gt. err) THEN
1442 CN = -0.5*CN
1443 endif
1444 PHI = PHI + CN*PI/180.0
1445 ERR = SUM
1446 IF( abs(cn) .gt. 0.001 .and. icount .lt. itter ) GO TO 30
1447 PHASE(J) = PHI*180.0/PI
1448
1449 c do i = 4,npts
1450 c FNT(J,I) = AMP(J)*SIN(W*T(I) + PHASE(J)*pi/180.0)
1451 c end do
1452 200 CONTINUE
1453 open (unit=15,file='phase.d',form='formatted')
1454 write(15,'(4f12.5)')
1455 c > ( t(i),alpha(i), fnt(1,i),fnt(2,i), I=1,npts)
1456 c > ( t(i),alpha(i), cl(i),cm(i), I=1,npts)
1457 close(15)
1458 print*, ' '
1459 print*, 'AMPLITUDE; clamp, cmamp :',amp(1),amp(2)
1460 print*, 'PHASE; clp, cmp :',
1461 c > phase(1)+180, phase(2)
1462 c DETERMINE THE PROPULSIVE EFFICIENCY
1463 PHASE(1) = PHASE(1)*pi/180.0
1464 PHASE(2) = PHASE(2)*pi/180.0
1465 CDTOT = 0
1466 k = 0
1467 DO I =npts-ntcycle, npts
1468 CDTOT = CDTOT + CD(I) - CD(1)
1469 k = k+1
1470 END DO
1471 DBAR = CDTOT/K
1472 DBAR = DBAR
1473 PRINT*, 'AVERAGE DRAG, TOTAL DRAG : ', DBAR,CDTOT
1474 IF(itrans.EQ. 1) THEN
1475 WBAR = -.5*w*SIN(PHASE(1))*AMP(1)
1476 ETA = 2*DBAR/WBAR
1477 ELSE
1478 WBAR = .5*w*SIN(PHASE(2))*AMP(2)
1479 ETAS = DBAR/WBAR
1480 END IF
1481 PRINT*, 'ETAS, WBAR : ',ETAS,WBAR
1482 C DETERMINE AERODYNAMIC FORCES
1483 PHASE(1) = PHASE(1) + PI
1484 AMP(1) = AMP(1)/2.0
1485 IF(itrans.EQ. 1) THEN
1486 L1=AMP(1)*cos(PHASE(1))/(pi*(w/2.0)**2*alp1)
1487 L2=AMP(1)*sin(PHASE(1))/(pi*(w/2.0)**2*alp1)
1488 M1= .5
1489 M2= 0
1490 c print*, 'L1,L2 = ',L1,L2,amp(1)

```

```
c      print*, 'M1,M2 = ', M1,M2,amp(2)
1492     ELSE
1493       print*, 'INPUT L1, L2 ='
c       read(*,*) L1,L2
1494
c       L3=2*AMP(1)*cos(PHASE(1))/(pi*(w/2.0)**2*alp1) + .5*L1
1495
c       L4=2*AMP(1)*sin(PHASE(1))/(pi*(w/2.0)**2*alp1) + .5*L2
1496
1497       M1 = .5
1498       M2 = 0
1499       M3 = .375
1500       M4 = -(2/W)
1501     END IF
1502     return
1503   end
1504 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
1505 SUBROUTINE AMPLITUDE(DAT,AMP,NPTS,J)
1506 DIMENSION DAT(200),AMP(3),AMP1(10),AMP2(10)
1507 n2 = 0
1508 m2 = 0
1509 do i= 1,10
1510   amp1(i) = 0
1511   amp2(i) = 0
1512 end do
1513 DO I = 2,NPTS-1
1514   IF(DAT(I+1).LT. 0 .AND. DAT(I).LT. 0 ) THEN
1515     IF(ABS(DAT(I+1)).GT.ABS(DAT(I))) then
1516       if((n2+1)/2.0.EQ. int((n2+1)/2.0)) n2 = n2+1
1517       TMP = ABS(DAT(I+1))
1518     else
1519       goto 10
1520     end if
1521     if(tmp.gt. amp1(n2)) then
1522       amp1(n2) = tmp
1523       tmp =0
1524     end if
1525   ELSE
1526     if((n2+1)/2.0.ne. int((n2+1)/2.0)) n2 = n2+1
1527     IF(ABS(DAT(I+1)).GT.ABS(DAT(I))) THEN
1528       TMP = ABS(DAT(I+1))
1529     else
1530       goto 10
1531     end if
1532     if(tmp.gt. amp2(n2)) then
1533       amp2(n2) = tmp
1534       tmp =0
1535     end if
1536   end if
1537 END DO
1538 if (amp1(2).gt. amp2(2)) then
1539   if(amp1(2).lt. amp2(3)) then
1540     comp = amp1(2)
1541   else
1542     comp = amp2(3)
1543   end if
1544 Else
1545   if(amp2(2).lt. amp1(3)) then
1546     comp = amp2(2)
1547   else
1548     comp = amp1(3)
1549   end if
1550 end if
1551 do i = 2,n2
1552   if(abs(amp1(i)-comp).GT. .1*comp) go to 20
1553   m2 = m2 + 1
1554   amp(j) = amp(j) +amp1(i)
1555   go to 30
1556 20   if(abs(amp2(i)-comp).GT. .1*comp) go to 30
1557   m2 = m2 +1
1558   amp(j) = amp(j) + amp2(i)
1559 30 continue
1560 end do
1561 if (j.lt. 3) then
1562   AMP(J) = AMP(J)/(M2)
1563 else
1564   amp(j) = amp(j)/(2*m2)
1565 end if
1566 return
1567 END
```

APPENDIX C

A. NS.IN NAME LIST

```
*****
No dphi/dt, circulation is applied
213x61 new expanding smooth grid....ITEU=183...ITEL=31
-----
```

C.. IREAD,	ITER	NPRINT,	NLOAD	ODALFA	
0	2000	100	500	1.0	
C.. POTEN,	NTPOT,	MPOT,	MDF	KSISO,	SO DIST ~ (Line not used)
false	1	1	1	4	0.15
C.. ALPHA	OSCIL	RAMP	REDFRE	ALFAMND	ALFAMXD
2.00	false	false	0.01	0.001	20.0
C.. MACH	REYNOLD	VISC	TURBL		
0.200	2000000.	true	true		
C.. TIMEACC	COUR	NEWTIT			
false	100	2			

```
-----

..comments..

IREAD:      0: no initial solution
             1: initial solution binary ** (cp ends.d  strs.d)
             2: initial solution formatted (plot3d form)

ITER:       # of iterations..
             Ramp:       $((a_{max} - a_{min}) / (2 A M dtau))$ 
             Sinusoid:   $(\sin / (k M dtau))$ 

NPRINT:     prints residuals at every nprint timesteps

NLOAD:      prints loads at every nload timesteps

OALFA:      prints out q file at every oalpha degree

poten:      false: no interactive solution
             true:  potential flow ns interavtive solution
                   (Inner and Outer Grid)

ntpot:      timestep where interavtive solution starts

mpot:       interactive boundary conditons are updated at
             every mpot timestep

mdf:        dphi/dt is computed at every mdf timestep

ksiso:      inner boundary is located ksiso grid points
             inside the outer boundary

so_dist:    where the outer boundary is located (in terms of
             chord lengths

ALPHA:      steady state aoa

OSCIL:      false: no sinuzaidal oscillations
             true:  sinuzaidal oscillations

RAMP:       false: no straight ramp motion
             true:  straight ramp motion

REDFRE:     Reduced frequency based on HALF CHORD, chord length is
             assumed to be 1.
```



```

7 echo " " > $NO ; date >> $NO ; echo " " >> $NO ; \
8 cat ns/ns.in >> $NO ; \
9 ( $SRC/$NS < ns/ns.in >> $NO ) >> $NO.t ; echo " " >> $NO ; \
10 cat $NO.t >> $NO ; date >> $NO ; \
11 exit 0
*****

```

WP3D.F

```

PROGRAM WP3D
C..Reads binary last iteration file and writes it in plot3d format with Rotated
C... Grid
3 parameter (imax=250,kmax=100)
4 dimension q(4,imax,kmax),x(imax,kmax),z(imax,kmax)
5 > ,xr(imax,kmax),zr(imax,kmax)
6 character filnq*15,alpn*10,filngr*15
7 alpn = '0123456789'
8 pi = 3.14159
9 c..read the grid
10 open (7,file='/d3/johnston/ns/grid.in',status='old',
11 + form='formatted')
12 read (7,*) imx, kmx, iws, iwe
13 read (7,*) ((x(i,k), i = 1, imx), k = 1, kmx ),
14 > ((z(i,k), i = 1, imx), k = 1, kmx )
15 close(7)
16 open (8,file='inf',status='old',form='unformatted')
17 DO 100 II = 9,100,2
18 read (8,end=101) imx, kmx
19 read (8) amach,alfad,reynph,time,itn
20 read (8) ((( q(1,i,k), i=1,imx), k=1,kmx), l=1,4)
21 C..extract character string for iteration count
22 i6 = itn/100000 + 1
23 i5 = (itn - (i6-1)*100000)/10000 + 1
24 i4 = (itn - (i6-1)*100000 - (i5-1)*10000)/1000 + 1
25 i3 = (itn - (i6-1)*100000 - (i5-1)*10000
26 > - (i4-1)*1000)/100 + 1
27 i2 = (itn - (i6-1)*100000 - (i5-1)*10000 - (i4-1)*1000
28 > - (i3-1)*100 )/10 + 1
29 i1 = (itn - (i6-1)*100000 - (i5-1)*10000 - (i4-1)*1000
30 > - (i3-1)*100 - (i2-1)*10 ) + 1
31
32 filnq = 'q'//alpn(i5:i5)//alpn(i4:i4)//alpn(i3:i3)//
33 > alpn(i2:i2)//alpn(i1:i1)//'.fmt'
34 filngr = 'gr'//alpn(i5:i5)//alpn(i4:i4)//alpn(i3:i3)//
35 > alpn(i2:i2)//alpn(i1:i1)//'.fmt'
36 c..write the qfile
37 open (ii,file=filnq,form='formatted')
38 write(ii,*) imx,kmx
39 write(ii,'(5e15.7)') amach,alfad,reynph,time
40 write(ii,'(5e15.7)')
41 > ((( q(is,i,k),i=1,imx ),k=1,kmx ), is=1,4)
42 close(ii)
43 c..write the rotated grid
44 dalfa = alfad * pi/180.
45 ca = cos( dalfa )
46 sa =-sin( dalfa )
47 do 10 i=1,imx
48 do 10 k=1,kmx
49 xr(i,k) = x(i,k) * ca - z(i,k) * sa
50 zr(i,k) = z(i,k) * ca + x(i,k) * sa
51 10 continue
52 ig = ii + 1
53 open (ig,file=filngr,form='formatted')
54 write(ig,*) imx,kmx, iws, iwe
55 write(ig,'(5e15.7)')
56 > ((xr(i,k), i = 1, imx), k = 1, kmx ),
57 > ((zr(i,k), i = 1, imx), k = 1, kmx )
58 close(ig)
59 100 Continue
60 101 close(8)
61 STOP
62 END

```

WRCL.F

```

PROGRAM wrcl
*** auto-writes all files given dtau and niter
*** includes Cf data

```

```

*** Corrects NS.F grid/load error
C..write alpha and cl from loads.d file
1  parameter (idim=215 )
2  dimension cp(idim,idim), cl(idim),cd(idim),cm(idim),time(idim),
3  > alpha(idim),clv(idim),cdv(idim),cmv(idim),clw(idim),cdw(idim),
4  > cmw(idim), x(idim,idim),z(idim,idim),cf(idim,idim)
5  character fname*80
6  * Print*, 'input niter = '
7  * read*, niter
8  * print*, 'input dtau = '
9  * read*, dtau
10 print*, '
11 print*, ' Enter LOAD file name :>'
12 read(*, '(a80)') fname
13 open(2, file=fname, form='formatted', status='old')
14 do it = 1, 1000
15 read(2, *, end=21) iter, alpha(it), time(it), fsmach, re, itel, iteu,
16 > cl(it), cd(it), cm(it), clv(it), cdv(it), cmv(it),
17 > (cp(i), i=1, iteu-itel+1),
18 > (cf(i), i=1, iteu-itel+1)
19 enddo
20 21 close(2)
21 open(11, file='/d3/johnston/ns/grid.in', form='formatted',
22 + status='old')
23 read (11,*) imx, kmx, iwks, iwke
24 read (11,*) ((x(i,k), i = 1, imx), k = 1, kmx ),
25 > ((z(i,k), i = 1, imx), k = 1, kmx )
26 close(11)
27 -----
28 do 30 itt = 1, it-1
29 cn=0.
30 ch=0.
31 cmp=0.
32 ralfa = alpha(itt)*pi/180.
33 do 25 i=itel+1, iteu
34 dx = x(i+1) - x(i-1,1)
35 dz = z(i,1) - z(i-1,1)
36 avcp = .5*( cp(i,itt) + cp(i-1,itt) )
37 cn = cn - avcp*dx
38 ch = ch + avcp*dz
39 zave = .5*(z(i,1) + z(i-1,1) )
40 xave = .5*(x(i,1) + x(i-1,1) )
41 C.. Cm about .25 x/c
42 cmp = cmp + avcp*( dz*zave + dx*(xave-.25) )
43 25 continue
44 clw(itt) = cn*cos(ralfa) - ch*sin(ralfa)
45 cdw(itt) = cn*sin(ralfa) + ch*cos(ralfa)
46 cmw(itt) = cmp
47 30 continue
48 -----
49 open(3, file='cla.d', form='formatted')
50 write(3, '(2f10.5)') ( alpha(i), -clw(i), i = 1, it-1)
51
52 * open(4, file='clt.d', form='formatted')
53 * write(4, '(2f10.5)') ( time(i)/niter/dtau, -clw(i), i = 1, it-1)
54
55 open(5, file='cda.d', form='formatted')
56 write(5, '(2f10.5)') ( alpha(i), -cdw(i), i = 1, it-1)
57
58 * open(6, file='cdt.d', form='formatted')
59 * write(6, '(2f10.5)') ( time(i)/niter/dtau, -cdw(i), i = 1, it-1)
60
61 open(7, file='cma.d', form='formatted')
62 write(7, '(2f10.5)') ( alpha(i), -cmw(i), i = 1, it-1)
63
64 * open(8, file='cmt.d', form='formatted')
65 * write(8, '(2f10.5)') ( time(i)/niter/dtau, -cmw(i), i = 1, it-1)
66
67 open(9, file='cp.d', form='formatted')
68 write(9, '(2f10.5)') ( x(i,1), -cp(i, it-1), i = itel, iteu)
69
70 open(10, file='cf.d', form='formatted')
71 write(10, '(2f10.5)') ( x(i,1), cf(i, it-1), i = itel, iteu)
72
73 STOP
74 END
*****

```

PLCON.F

```

PROGRAM PLCON
1 C..reads gr.. and q .. solution files and plots contours
2 C..MUST COMPILE AND LINK THIS PROGRAM
3 ** f77 -c plcon.f
4 ** f77 plcon.o /usr/local/bin/nasadig.a -o plcon
5 parameter (idim=213,kdim=61)
6 dimension q(idim,kdim,4), xy(idim,kdim,2), ax(idim),ay(idim)
7 > ,ybl(kdim), vbl(kdim), func(idim,kdim)
8 character yn*1,title*80,fname*80, text*80
9 integer*4 ititle(20)
10 equivalence (title, ititle(1))
11 data xumin,xumax,yumin /-0.75, 1.50, -0.75 /
12 c data xumin,xumax,yumin /-0.1, 1.25, -0.35 /
13 > ,xlen,ylen/3.25, 2.5 /
14 > ,gam /1.4/
15
16 C..nasadig calls
17 print*, ' Enter Out.PS file name :>'
18 read(*,102) FNAME
19 open(10, file = fname,form='formatted',status='unknown')
20 CALL postsc(10)
21 c call device(ktype,xpage,ypage)
22 CALL PAGE(11., 8.5)
23 c CALL HRDrot('COMIC')
24 c CALL HRDrot('MOVIE')
25 CALL HRDSCL('NONE')
26 CALL NOBord
27 CALL NOCHEK
28 CALL DUPlex
29 CALL HEIGHT(0.08)
30 CALL frmwid(0.012)
31 CALL crvwid(0.001)
32 CALL MARGIN(0.)
33 print*, ' Input Ramp=1 or Sinusoid=2 or SS=9'
34 read*,jj
35 CC View area of interest definition
36 WRITE(*,105) XUMIN,XUMAX,yumin
37 105 FORMAT(/' xumin, xumax, yumin : ',
38 > '(', 3F7.2, ' )', ' Redefine ? (n):>' )
39 READ(*,101) YN
40 IF (YN.EQ.'Y' .OR. YN.EQ.'y') then
41 print*, 'Input xumin, xumax, yumin '
42 READ(*,*) XUMIN,XUMAX,yumin
43 endif
44 C..read caption (Placed at bottom of page in Landscape mode)
45 c print*, ' Any caption for the plots ?. (caption) :>'
46 c read(*,102,END=10) TITLE
47 102 format(A80)
48 c TITLE= ' '
49 C..read grid
50 alphao = 0.
51 alphao = 0.
52 c FNAME = 'grid.in'
53 print*, ' '
54 c print*, ' Default grid file is grid.in Is this o.k.? (y or n) '
55 c read(*,101) yn
56 101 format(a1)
57 c if( yn .eq. 'n' .or. yn .eq. 'N') then
58 13 print*, ' Enter GRID file name :>'
59 read(*,102) FNAME
60 c endif
61 OPEN(1,FILE=FNAME,FORM='FORMATTED',STATUS='OLD')
62 read(1,*) imx,kmx,iws,iwe
63 read(1,*) ((xy(i,k,1),i=1,imx ),k=1,kmx ),
64 > ((xy(i,k,2),i=1,imx ),k=1,kmx )
65 close(1)
66 ile = imx/2 + 1
67 c fix the leading edge and trainling edge points.
68 c ile = 31
69 ite = 183
70 C..read "q" file
71 PRINT*, ' Enter Q file name :>'
72 READ(*,102) FNAME
73 OPEN(2,FILE=FNAME,FORM='FORMATTED',STATUS='OLD')
74 10 read(2,*,end=1000) imx,kmx

```

```

75      read(2,*) fsmach,alpha,re,time
76      read(2,*)
77      > ((( q(i,k,is),i=1,imx ),k=1,kmx ), is=1,4)
78      C..ROTATE GRID WRT ALPHA (NOT required for ns.f)
79      c      CALL ROTXY(ALPHA-alpha0,IMX,KMX,XY)
80      alpha0 = alpha
81      c..extract airfoil coordinates
82      ii = 0
83      do i = 1,imx
84      ii = ii+1
85      ax(ii) = xy(I,1,1)
86      ay(ii) = xy(I,1,2)
87      enddo
88      PRINT*, ' '
89      PRINT*, ' Angle of ATTACK = ', ALPHA
90      30 PRINT*, ' '
91      c      PRINT*, ' Choose the contour function :>'
92      c      PRINT*, ' '
93      c      PRINT*, ' 1 : Density'
94      c      PRINT*, ' 2 : Pressure'
95      c      PRINT*, ' 3 : Mach Number'
96      c      PRINT*, ' 4 : Vorticity'
97      c      PRINT*, ' 5 : Mass-flux'
98      c      PRINT*, ' 6 : NEXT Time step'
99      c      PRINT*, ' 7 : EXIT'
100     c      PRINT*, ' 8 : Next q-file'
101     c      PRINT*, ' '
102     c      READ(*,*) IFUN
103     do 999 ifun = 1,4
104     fmax = -10000.
105     fmin = 10000.
106     NCON = 25
107     IF(IFUN.EQ. 8) THEN
108     goto 13
109     endif
110     IF(IFUN.EQ. 1) THEN
111     C..ASSIGN DENSITY
112     print*, 'Density'
113     DO 91 k = 1,kMX
114     DO 91 I = 1,IMX
115     FUNC(I,k) = Q(I,k,1)
116     fmin = min(func(i,k), fmin)
117     fmax = max(func(i,k), fmax)
118     91 continue
119     coninc = (fmax-fmin)/ncon
120     ELSEIF(IFUN.EQ. 2) THEN
121     C..EVALUATE Pressure
122     print*, 'Pressure'
123     DO 92 k = 1,kMX
124     DO 92 I = 1,IMX
125     VEL2 = ( q(i,k,2)**2 + q(i,k,3)**2 )/Q(I,k,1)
126     FUNC(I,k) = gam*(gam-1.)*( Q(I,k,4)-.5*VEL2 )
127     fmin = min(func(i,k), fmin)
128     fmax = max(func(i,k), fmax)
129     92 continue
130     coninc = (fmax-fmin)/ncon
131     ELSEIF(IFUN.EQ. 3) THEN
132     C..EVALUATE MACH NUMBER
133     print*, 'Mach'
134     DO 93 k = 1,kMX
135     DO 93 I = 1,IMX
136     VEL2 = ( q(i,k,2)**2 + q(i,k,3)**2 )/Q(I,k,1)**2
137     PP = (GAM-1.)*( Q(I,k,4)-.5*VEL2**2(I,k,1) )
138     AL2 = GAM*PP/Q(I,k,1)
139     Func(i,k) = SQRT(VEL2/AL2)
140     fmin = min(func(i,k), fmin)
141     fmax = max(func(i,k), fmax)
142     93 continue
143     coninc = (fmax-fmin)/ncon
144     ELSEIF(IFUN.EQ. 4) THEN
145     C..EVALUATE VORTICITY FIELD
146     print*, 'Vorticity'
147     do 41 k=2,kmx-1
148     km=k-1
149     kp=k+1
150     do 41 i=2,imx-1
151     ip=i+1
152     im=i-1

```

```

153      xxi = 0.5 * ( xy(ip,k,1) - xy(im,k,1) )
154      zxi = 0.5 * ( xy(ip,k,2) - xy(im,k,2) )
155      xze = 0.5 * ( xy(i,kp,1) - xy(i,km,1) )
156      zze = 0.5 * ( xy(i,kp,2) - xy(i,km,2) )
157      rjacob = 1./ ( xxi*zze - xze*zxi )
158      xix = rjacob * zze
159      xiz = -rjacob * xze
160      zex = -rjacob * zxi
161      zez = rjacob * xxi
162      u_ze = 0.5*( q(i,kp,2)/q(i,kp,1)-q(i,km,2)/q(i,km,1) )
163      w_ze = 0.5*( q(i,kp,3)/q(i,kp,1)-q(i,km,3)/q(i,km,1) )
164      u_xi = 0.5*( q(ip,k,2)/q(ip,k,1)-q(im,k,2)/q(im,k,1) )
165      w_xi = 0.5*( q(ip,k,3)/q(ip,k,1)-q(im,k,3)/q(im,k,1) )
166      dudz = u_xi*xiz + u_ze*zez
167      dwdx = w_xi*xix + w_ze*zex
168      func(i,k) = dudz-dwdx
169      fmin = min(func(i,k), fmin)
170      fmax = max(func(i,k), fmax)
171  41 continue
172  do 42 i = 1,imx
173  42 func(i,1) = func(i,2)
174      fmax = fmax/100.
175      fmin = fmin/100.
176      coninc = (fmax-fmin)/ncon
177      ELSEIF(IFUN.EQ. 5) THEN
178  C..EVALUATE MASS-FLUX FUNCTION
179      c1 = 0.5
180      k = 1
181      func(1,k) = 0.
182      DO I = 1,imx
183          FUNC(I,k) = FUNC(I-1,k) +
184      > C1*( (Q(I,k,2) + Q(I-1,k,2))*(XY(I,k,2)-XY(I-1,k,2)) -
185      > (Q(I,k,3) + Q(I-1,k,3))*(XY(I,k,1)-XY(I-1,k,1)) )
186      enddo
187      do i = iteu, imx
188      c      func(i,k) = func(imx-i+1,k)
189      c      enddo
190      DO I=1,IMX
191      DO k=2,KMX
192          FUNC(I,k) = FUNC(I,k-1) +
193      > C1*( (Q(I,k,2) + Q(I,k-1,2))*(XY(I,k,2) - XY(I,k-1,2)) -
194      > (Q(I,k,3) + Q(I,k-1,3))*(XY(I,k,1) - XY(I,k-1,1)) )
195          fmin = min(func(i,k), fmin)
196          fmax = max(func(i,k), fmax)
197      enddo
198      enddo
199      do k=1,kmx
200          func(imx,k) = func(1,k)
201      end do
202      coninc = fsmach/50.
203      ncon = (fmax-fmin)/coninc
204      ELSEIF(IFUN.EQ. 6) THEN
205      goto 10
206      ELSEIF(IFUN.EQ. 7) THEN
207      goto 1000
208      ELSE
209      PRINT*, ' WRONG Selection.....'
210      goto 30
211      ENDIF
212  31 PRINT*, ' '
213      PRINT*, ' Function Min and Max :> ',fmin,fmax
214      PRINT*, ' '
215      PRINT*, ' Enter NCON :>'
216  c      READ(*,*) NCON
217      conmin = fmin
218      conmax = fmax
219      fmax = -10000.
220      fmin = 10000.
221      NP = NP+1
222      MNP = MOD(NP,4)
223      IF(MNP.EQ.1) THEN
224          X0 = 2.37
225          Y0 = 4.31
226          CALL frmwid(0.012)
227          CALL crvwid(0.001)
228      ELSEIF(MNP.EQ.2) THEN
229          X0 = 6.12
230          Y0 = 4.31

```

```

231     ELSEIF(MNP.EQ.3) THEN
232         X0 = 2.37
233         Y0 = 1.31
234     ELSEIF(MNP.EQ.0) THEN
235         X0 = 6.12
236         Y0 = 1.31
237     ENDIF
238     DUY = YLEN*(XUMAX-XUMIN)/XLEN
239     YUMAX = YUMIN + DUY
240     YS = YLEN/(YUMAX-YUMIN)
241     XS = XLEN/(XUMAX-XUMIN)
242     CALL ORIGIN(X0,Y0)
243     CALL SETSUB(XLEN,YLEN)
244     C    CALL XLABEL(' ',1)
245     CALL AXES2D(XUMIN,XUMAX-XUMIN,XUMAX, YUMIN,YUMAX-YUMIN,YUMAX)
246     CALL FRAME
247     CALL MARGIN(0.)
248     DO 50 N = 1,NCON+1
249         CONV = CONMIN + coninc*(N-1)
250         if( ifun .eq. 4 .and. abs(conv) .lt. .50) goto 50
251         CALL CONTOUR(FUNC,XY,IMX,kMX,CONV,
252             >          XUMIN - ABS(XUMIN)*0.5 ,XUMAX*1.5,
253             >          YUMIN - ABS(YUMIN)*0.5 ,YUMAX*1.5, NCELL)
254         if(ncell .ne. 0) then
255             fmin = min(fmin,conv)
256             fmax = max(fmax,conv)
257         c    print*, 'n, conv, ncell :', n,conv,ncell
258         endif
259     50 CONTINUE
260     c..nomenclature
261     CALL HEIGHT(0.08)
262     if( ifun .eq. 1) CALL TXTMSG('DENSITY$',100,0.2, YLEN+0.1)
263     if( ifun .eq. 2) CALL TXTMSG('PRESSURE$',100,0.2, YLEN+0.1)
264     if( ifun .eq. 3) CALL TXTMSG('MACH NUMBERS$',100,0.2, YLEN+0.1)
265     if( ifun .eq. 4) CALL TXTMSG('VORTICITY$',100,0.2, YLEN+0.1)
266     if( ifun .eq. 5) CALL TXTMSG('MASS-FLUX$',100,0.2, YLEN+0.1)
267     call defalf('L/CGREEK')
268     call txtmsg('a =$',100, xlen-0.75, ylen-0.2)
269     call reset('DEFALF')
270     call realno (alpha,1, 'ABUT','ABUT')
271     call height(0.05)
272     if ( ifun .ne. 5 ) then
273     c    was 0.24
274         call txtmsg ('MIN = $',100, 0.05,0.16)
275         call realno (fmin, 2, 'ABUT','ABUT')
276         call txtmsg ('MAX = $',100, 0.05,0.05)
277         call realno (fmax, 2, 'ABUT','ABUT')
278         CALL HEIGHT(0.08)
279         call txtmsg('M =$',100,xlen-3.1,ylen-.2 )
280         call realno (fsmach,2, 'ABUT','ABUT')
281         if (jj .eq. 1) then
282             CALL HEIGHT(0.08)
283             call txtmsg ('Ramp$',100,xlen-2.95,ylen-.35)
284             CALL HEIGHT(0.05)
285             call txtmsg ('(A=0.005)$',100,xlen-2.97,ylen-.46)
286             elseif (jj .eq. 2) then
287                 CALL HEIGHT(0.08)
288                 call txtmsg ('Sinusoid$',100,xlen-3.05,ylen-.35)
289                 CALL HEIGHT(0.05)
290                 call txtmsg ('(k=0.05)$',100,xlen-3.05,ylen-.46)
291             endif
292         c    call txtmsg ('TIME = $',100, 0.6,0.15)
293         c    call realno (TIME, 2, 'ABUT','ABUT')
294         end if
295     C..draw airfoil..
296     call curve (ax,ay, ii, 0)
297     IF(MNP .EQ. 1) then
298         call height(0.08)
299     c    call txtmsg('Mach =$',100, 0., -4.1 )
300     c    call realno (fsmach,2, 'ABUT','ABUT')
301     c    call txtmsg(' Re =$',100, 'ABUT', 'ABUT' )
302     c    call realno (re,1, 'ABUT','ABUT')
303     c    call txtmsg(ititle,100, 0., -4.4 )
304     call endsub(0)
305     elseif( mnp .eq. 0) then
306         call stoplt(0)
307     else
308         call endsub(0)

```

```

309      endif
310      c PRINT*, ' Do you want to change the increament.? (n) :>'
311      c READ(*, '(a1)') YN
312      c IF(YN.EQ.'Y' .OR. YN.EQ.'y') GOTO 31
313      c GOTO 30
314
315      999 continue
316
317      1000 IF(MNP .ne. 0) call stoplt(0)
318      CALL FINPLT
319      CLOSE(2)
320      CLOSE(10)
321      STOP
322      END
323
324      C-----
325      > SUBROUTINE CONTOUR (F,XY,IMX,JMX,CONV,
326      > XMIN,XMAX,YMIN,YMAX,NCELL)
327      C..FINDS CONTOUR LINES AND PLOTS
328      parameter (idim=213 ,jdim=61 )
329      DIMENSION F(IDIM,JDIM),XY(IDIM,JDIM,2), X(2),Y(2)
330      NCELL = 0
331      DO 50 I = 1, IMX-1
332      IP = I+1
333      DO 50 J = 1, JMX-1
334      JP = J+1
335      X1 = XY(I,J,1)
336      Y1 = XY(I,J,2)
337      IF(X1.GT.XMAX.OR.X1.LT.XMIN.OR.Y1.GT.YMAX.OR.Y1.LT.YMIN)
338      > GOTO 50
339      X2 = XY(I,JP,1)
340      Y2 = XY(I,JP,2)
341      X3 = XY(IP,JP,1)
342      Y3 = XY(IP,JP,2)
343      X4 = XY(IP,J,1)
344      Y4 = XY(IP,J,2)
345      F1 = F(I,J)
346      F2 = F(I,JP)
347      F3 = F(IP,JP)
348      F4 = F(IP,J)
349      NP = 0
350      IF((CONV.GT.F1.AND.CONV.LT.F2) .OR.
351      > (CONV.LT.F1.AND.CONV.GT.F2) ) THEN
352      NP = NP+1
353      X(NP) = X2 - (F2-CONV)*(X2-X1)/(F2-F1)
354      Y(NP) = Y2 - (F2-CONV)*(Y2-Y1)/(F2-F1)
355      ENDIF
356      IF((CONV.GT.F4.AND.CONV.LT.F3) .OR.
357      > (CONV.LT.F4.AND.CONV.GT.F3) ) THEN
358      NP = NP+1
359      X(NP) = X3 - (F3-CONV)*(X3-X4)/(F3-F4)
360      Y(NP) = Y3 - (F3-CONV)*(Y3-Y4)/(F3-F4)
361      ENDIF
362      IF(NP.EQ.2) THEN
363      CALL RELVEC( X(1),Y(1),X(2),Y(2),0)
364      NCELL = NCELL+1
365      ELSE
366      IF((CONV.GT.F2.AND.CONV.LT.F3) .OR.
367      > (CONV.LT.F2.AND.CONV.GT.F3) ) THEN
368      NP = NP+1
369      X(NP) = X3 - (F3-CONV)*(X3-X2)/(F3-F2)
370      Y(NP) = Y3 - (F3-CONV)*(Y3-Y2)/(F3-F2)
371      IF(NP.EQ.2) THEN
372      CALL RELVEC( X(1),Y(1),X(2),Y(2),0)
373      NCELL = NCELL+1
374      ENDIF
375      ENDIF
376      IF(NP.EQ.1) THEN
377      NP = NP+1
378      X(NP) = X4 - (F4-CONV)*(X4-X1)/(F4-F1)
379      Y(NP) = Y4 - (F4-CONV)*(Y4-Y1)/(F4-F1)
380      CALL RELVEC( X(1),Y(1),X(2),Y(2),0)
381      NCELL = NCELL+1
382      ENDIF
383      ENDIF
384      50 CONTINUE
385      RETURN
386      END
387      C-----

```

```

387      SUBROUTINE ROTXY ( ANGLE,IMX,JMX,XY )
388      parameter (idim=213 ,jdim=61 )
389      DIMENSION XY(IDIM,JDIM,2)
390      ROTANG = ANGLE*3.14159/180.
391      CA = COS(ROTANG)
392      SA = -SIN(ROTANG)
393      DO 10 I = 1,IMX
394      DO 10 J = 1,JMX
395          XC = XY(I,J,1)
396          YC = XY(I,J,2)
397          XY(I,J,1) = XC*CA - YC*SA
398      10 XY(I,J,2) = YC*CA + XC*SA
399      RETURN
400      END

```

C. PROGRAM NS.F SOURCE CODE

```

      program ns_pot2d
      c*****
      c                                     by
      c                               John A. Ekaterinaris
      c                               nasa, ames research center
      c                               march, 1990
      c                               modified by
      c                               I.H. Tuncer
      c                               nps
      c                               april 1992
      c                               solution of the 2-D unsteady, thin-layer navier-stokes
      c                               equations in a time-accurate manner.
      c                               characteristics of the code:
      c                               1) factored, iterative, implicit algorithm
      c                               2) high-order accurate upwind difference scheme (third order)
      c                               3) baldwin-lomax turbulence model
      c                               4) patched and overlaid grids
      c                               5) the code is almost completely vectorized for the cray-ymp
      c*****
      c.. "coms.f"
      1      parameter (nia = 213, nka = 61)
      2      common /alpar /oscl, ramp, redfre, omega,
      3      >      alfa, alfad, alfamn, alfamx
      4      logical      oscl, ramp
      5      common /gamvl /gamma,      gmm,      gmp,
      6      >      rgamma,      rgmm,      rgmp,
      7      >      gmbygp
      8      common /iksri /imx(2),      kmx(2),
      9      >      imx1(2),      kmx1(2),
      10     >      imx2(2),      kmx2(2)
      11     common /ikwk /iwks(2),      iwke(2)
      12     common /infvl /rinf,      uinf,      vinf,
      13     >      winf,      pinf,      einf,
      14     >      tinf,      amach,      pratio
      15     common /load /cl, cd, cm, clv, cdv, cmv
      16     common /nparm /niter,      newtit,loop,      iter,itr,
      17     >      nprint,iread,      nload,      odalfa,oalfa
      18     common /tmval /timeacc, time,      dtau,dt(nia,nka),cour
      19     common /visci /vismu(nia,nka),      turmu(nia,nka)
      20     common /visvl /reynnu,reynph,      prkin,      prtur
      21     common /vispar/visc,      turbl
      22     common /poten / poten, ntpot,mpot, mdf, ksi, kso, ksiso, sodist
      23     logical      visc,      turbl,      poten, timeacc
      24     common /grid / x(nia,nka), z(nia,nka)
      25     common /flow / q(4,nia,nka)
      26     common /dflow / qd(4,nia,nka)
      27     common /jacob / aja(nia,nka)
      28     common /metrcs/ xix(nia,nka), xiz(nia,nka), xit(nia,nka),
      29     >      zex(nia,nka), zez(nia,nka), zet(nia,nka)
      30     pi = 4.*atan(1.)
      31     call data
      32     do 10 itr = 1, niter
      33     iter = iter + 1
      34     time = time + dtau
      35     alfaold = alfa
      36     if( oscl ) then
      37     freq = 2.*redfre*uinf
      38

```

```

39      alfa = alfamn + 0.5*(alfamx-alfamn)*(1.- cos(freq*time))
40      omega = freq * 0.5*(alfamx-alfamn)*sin(freq*time)
41      call grmove(alfa-alfaold)
42      elseif( ramp ) then
43      omega = 2.*redfre*uin
44      alfa = alfamn + omega*time
45      if( alfa .gt. alfamx) then
46      alfa = alfamx
47      omega = 0.
48      endif
49      call grmove(alfa-alfaold)
50      endif
51      alfad = alfa * (180./pi)
52      c..update outer bc
53      c      if (poten ) call nspot
54      call step
55      if( abs(alfad-oalfa) .gt. 0.999*odalfa ) then
56      call qio(10)
57      oalfa = alfad
58      endif
59      if(mod(iter,1000) .eq. 0 .or. itr .eq. niter) call qio(0)
60      10 continue
61      c      if( .not. poten) then
62      c      open(unit=33,file='pres.d',form='formatted')
63      c      kso = 47
64      c      sso = 0.
65      c      do i = 2, imx(1)
66      c      ssoo = sso + sqrt( (x(i,kso)-x(i-1,kso))**2 +
67      c      > (z(i,kso)-z(i-1,kso))**2 )
68      c      pres = gmm*( q(4,i,kso)
69      c      > - .5*(q(2,i,kso)**2 + q(3,i,kso)**2)/q(1,i,kso) )
70      c      sso = ssoo
71      c      write(33,'(2e15.7)') sso, pres
72      c      enddo
73      c      endif
74      close(8)
75      close(9)
76      STOP
77      END
78      c      include 'nspot.f'
79      C-----
80      subroutine data
81      include 'coms.f'
82      pi = 4.*atan(1.)
83      read (5,*)
84      read (5,*)
85      read (5,*)
86      c..read top comments
87      read (5,*)
88      read (5,*) iread, niter, nprint, nload, odalfa
89      read (5,*)
90      read (5,*) poten, ntpot, mpot, mdf, ksiso, sodist
91      read (5,*)
92      read (5,*) alfad, oscil, ramp, redfre, alfamnd, alfamxd
93      read (5,*)
94      read (5,*) amach, reynph, visc, turbl
95      read (5,*)
96      read (5,*) timeacc, cour, newtit
97      c.. read in grid
98      open(unit=11,file='grid.in',form='formatted',status='old')
99      nt = 1
100     ng = 1
101     read (11,*) imx(nt), kmx(nt), iwks(1), iwke(1)
102     read (11,*) ((x(i,k), i = 1, imx(1)), k = 1, kmx(1) ),
103     > ((z(i,k), i = 1, imx(1)), k = 1, kmx(1) )
104     print *, 'Grid dimensions are :', imx(nt), kmx(nt)
105     print *, 'TE is located at I =', iwks(1)
106     kmx(2) = kmx(1)
107     imx1(nt) = imx(nt) - 1
108     imx2(nt) = imx(nt) - 2
109     kmx1(nt) = kmx(nt) - 1
110     kmx2(nt) = kmx(nt) - 2
111     alfa = alfad*pi/180.
112     alfamn = alfamnd*pi/180.
113     alfamx = alfamxd*pi/180.
114     omega = 0.
115     c.. specify some parameters ***
116     tinf = 530.0

```

```

117      prkin = 0.72
118      prtur = 0.90
119      reynnu = reynph / amach
120      gamma = 1.4
121      gmm = gamma - 1.0
122      gmp = gamma + 1.0
123      rgamma = 1.0/gamma
124      rgmm = 1.0/gmm
125      rgmp = 1.0/gmp
126      gmbygp = gmm/gmp
127      rinf = 1.0
128      uinf = amach
129      winf = 0.
130      pinf = 1.0/gamma
131      einf = 0.5*rinf*(uinf**2 + winf**2) + pinf*rgmm
132      iter = 0
133      time = 0.
134      if (iread .ne. 0) then
135          call qio(iread)
136          kmx1(1) = kmx(1) - 1
137          kmx2(1) = kmx(1) - 2
138          alfa = alfad*pi/180.
139          call grmove(alfa)
140          if( (oscil .or. ramp) .and.
141              > abs(alfad - alfamnd) .lt. 1.e-05 ) then
142              iter = 0
143              time = 0.
144          endif
145      c elseif( poten ) then
146      c..initialize bl (k<kpots) and the potential flowfield..
147      c      kpots = 25
148      c      do i = 1,imx(1)
149      c          do k = 1,kmx(1)
150      c              q(1,i,k) = rinf
151      c      enddo
152      c      q(2,i,1) = 0.
153      c      if (i .lt. iwks(1) .or. i .gt. iwke(1)) q(2,i,1) = uinf
154      c      q(3,i,1) = 0.
155      c      q(4,i,1) = einf
156      cenddo
157      c      call grmove(alfa)
158      c      call potsv(1,kmx(1), kpots, iwks(1), uinf)
159      c      do k = 2, kpots-1
160      c          ratiok = float(k)/kpots
161      c          do l = 1,4
162      c              do i = 1,imx(1)
163      c                  if (i .ge. iwks(1) .and. i .le. iwke(1)) then
164      c                      q(1,i,k) = q(1,i,1) + ratiok*( q(1,i,kpots)-q(1,i,1) )
165      c                  else
166      c                      q(1,i,k) = q(1,i,kpots)
167      c                  endif
168      c              enddo
169      c          enddo
170      c      enddo
171      c      else
172      c.. initialize q to freestream values everywhere ***
173      c      do 60 k = 1, kmx(1)
174      c          fact = min(1., float(k)/15)
175      c          do 60 i = 1, imx(1)
176      c              q(1,i,k) = rinf
177      c              if (i .ge. iwks(1) .and. i .le. iwke(1)) then
178      c                  q(2,i,k) = fact * rinf*uinf
179      c              else
180      c                  q(2,i,k) = rinf*uinf
181      c              endif
182      c              q(3,i,k) = 0.
183      c              q(4,i,k) = einf
184      c          60 continue
185      c          call grmove(alfa)
186      c      endif
187      c      call eigen
188      c      oalfa = float(int(alfad))
189      c      open(unit=9,file='loads.d',form='formatted')
190      c      open(unit=8,file='qp.d',form='unformatted')
191      c      write(6,101)
192      101 format(// ' Iter Alpha Time Resid Density i k ',
193      > ' Cm Cd Cl')
194      return

```

```

195      end
196  C-----
197      subroutine bc
198      include 'coms.f'
199      logical doit
200  c    if ( poten .or. ramp .or. oscil .or. iter .gt. 50) then
201      decay = 0.
202  c    else
203  c      decay = 1.0 - float(iter)/50.
204  c    endif
205      ng = 1
206  c..for <k=1>, <i=itel,iteu> for airfoils ***
207      il = iwcs(ng)
208      i2 = iwke(ng)
209      k1 = 1
210      k2 = 2
211      k3 = 3
212      do 100 i=il,i2
213          rval2 = q(1,i,k2)
214          pval2 = gmm*(q(4,i,k2) -
215      >          0.5*( q(2,i,k2)**2 + q(3,i,k2)**2 ) / q(1,i,k2))
216          rval1 = rval2
217          pval1 = pval2
218          xtau = omega * z(i,k1)
219          ztau = -omega * x(i,k1)
220          if( visc ) then
221      c..enforce non-slip boundary condition on the surface ***
222              u1 = q(2,i,k1)
223              u2 = q(3,i,k1)
224          else
225      c..enforce slip boundary condition on the surface ***
226              decay = 1.
227              u2 = q(2,i,k2)/q(1,i,k2)
228              u3 = q(2,i,k3)/q(1,i,k3)
229              v2 = q(3,i,k2)/q(1,i,k2)
230              v3 = q(3,i,k3)/q(1,i,k3)
231              ucon2 = xtau + xix(i,k2)*u2 + xiz(i,k2)*v2
232              ucon3 = xtau + xix(i,k3)*u3 + xiz(i,k3)*v3
233              vcon2 = ztau + zex(i,k2)*u2 + zez(i,k2)*v2
234              vcon3 = ztau + zex(i,k3)*u3 + zez(i,k3)*v3
235              ucon1 = 2.*ucon2 - ucon3
236              vcon1 = 0.
237              u1 = ( (ucon1-xtau)*zez(i,k1) + xiz(i,k1)*ztau )
238      >              *aja(i,k1)
239              v1 = ( -(ucon1-xtau)*zex(i,k1) - xix(i,k1)*ztau )
240      >              *aja(i,k1)
241          endif
242              q(1,i,k1) = rval1
243              q(2,i,k1) = (decay * u1 + xtau) * rval1
244              q(3,i,k1) = (decay * v1 + ztau) * rval1
245              q(4,i,k1) = rgmm*pval1 + 0.5*(q(2,i,k1)**2+q(3,i,k1)**2 )
246      >              / rval1
247      100 continue
248      doit = .false.
249      if( .not. poten .and. doit ) then
250      c..set free-stream conditions..
251          q(1,i,k1) = rinf
252          q(2,i,k1) = rinf*uinf
253          q(3,i,k1) = 0.
254          q(4,i,k1) = einf
255      elseif( .not. poten ) then
256      c.. enforce boundary conditions at the inlet boundary
257      c      1) pt(kmax) = pt(inlet) (total pressure condition)
258      c      2) u(kmax) = uinf (inlet angle = a degs)
259      c      3) w(kmax) = winf (inlet angle = a degs)
260      c      4) reim1(kmax) = reim1(inlet)
261      c          (reimann variable reim1 = u + 2c/(gamma-1))
262      c      5) reim2(kmax) = reim2(kmax-1)
263      c          (reimann variable reim2 = u - 2c/(gamma-1)) ***
264      c
265      cinf = sqrt(gamma*pinf/rinf)
266      sinf = pinf/rinf**gamma
267      reim1 = uinf + 2.0*cinf*rgmm
268      ptinf = pinf*(1.0 + 0.5*gmm*amach**2)**(gamma*rgmm)
269      k1 = kmx(ng)
270      k2 = kmx1(ng)
271      do 200 i = 2, imx1(ng)
272          pval2 = gmm*(q(4,i,k2) - 0.5*( q(2,i,k2)**2 + q(3,i,k2)**2 )

```

```

273 > cval2 = sqrt(gamma*pval2/q(1,i,k2)) / q(1,i,k2) )
274 uval2 = q(2,i,k2)/q(1,i,k2)
275 reim2 = uval2 - 2.0*cval2*rgmm
276 uval1 = 0.5*(reim1 + reim2)
277 wval1 = winf
278 cval1 = 0.25*gmm*(reim1 - reim2)
280 c amsq = (uval1**2 + wval1**2)/cval1**2
281 c pval1 = ptinf*(1.0 + 0.5*gmm*amsq)**(-gamma*rgmm)
282 c rval1 = gamma*pval1/cval1**2
283 sval1 = sinf
284 rval1 = (rgamma*cval1**2/sval1)**rgmm
285 pval1 = rgamma*rval1*cval1**2
286 q(1,i,k1) = rval1
287 q(2,i,k1) = rval1*uval1
288 q(3,i,k1) = rval1*wval1
289 q(4,i,k1) = pval1*rgmm + 0.5*rval1*(uval1**2 + wval1**2)
290 200 continue
291 endif
292 c..enforce boundary conditions at the exit boundary
293 c 1) p = pstat (static pressure condition)
294 c 2) w(imax)= w(imax-1)
295 c 2) reim1(imax) = reim1(imax-1)
296 c (reimann variable reim1= u + 2c/(gamma-1))
297 c 4) s(imax) = s(imax-1) (entropy condition) ***
298 c
299 ng = 1
300 i1 = imx(ng)
301 i2 = imx1(ng)
302 do 300 k = 1, kmx(ng)
303 rval2 = q(1,i2,k)
304 pval2 = gmm*(q(4,i2,k) - 0.5*( q(2,i2,k)**2 + q(3,i2,k)**2 )
305 > / q(1,i2,k) )
306 cval2 = sqrt(gamma*pval2/q(1,i2,k))
307 uval2 = q(2,i2,k)/q(1,i2,k)
308 wval2 = q(3,i2,k)/q(1,i2,k)
309 reim1 = uval2 + 2.0*rgmm*cval2
310 sval2 = pval2/rval2**gamma
311 sval1 = sval2
312 pval1 = pinf
313 rval1 = (pval1/sval1)**rgamma
314 cval1 = sqrt(gamma*pval1/rval1)
315 uval1 = min( uinf, reim1 - 2.0*rgmm*cval1 )
316 wval1 = wval2
317 c rval1 = rval2
318 c uval1 = uval2
319 c wval1 = wval2
320 q(1,i1,k) = rval1
321 q(2,i1,k) = rval1*uval1
322 q(3,i1,k) = rval1*wval1
323 q(4,i1,k) = pval1*rgmm + 0.5*rval1*(uval1**2 + wval1**2)
324 300 continue
325 i1 = 1
326 i2 = 2
327 do 350 k = 1, kmx(ng)
328 rval2 = q(1,i2,k)
329 pval2 = gmm*(q(4,i2,k) - 0.5*( q(2,i2,k)**2 + q(3,i2,k)**2 )
330 > / q(1,i2,k) )
331 cval2 = sqrt(gamma*pval2/q(1,i2,k))
332 uval2 = q(2,i2,k)/q(1,i2,k)
333 wval2 = q(3,i2,k)/q(1,i2,k)
334 reim1 = uval2 + 2.0*rgmm*cval2
335 sval2 = pval2/rval2**gamma
336 sval1 = sval2
337 pval1 = pinf
338 rval1 = (pval1/sval1)**rgamma
339 cval1 = sqrt(gamma*pval1/rval1)
340 uval1 = reim1 - 2.0*rgmm*cval1
341 wval1 = wval2
342 q(1,i1,k) = rval1
343 q(2,i1,k) = rval1*uval1
344 q(3,i1,k) = rval1*wval1
345 q(4,i1,k) = pval1*rgmm + 0.5*rval1*(uval1**2 + wval1**2)
346 350 continue
347 c..outgoing bc along the C part of the grid..
348 doit = .false.
349 if ( doit) then
350 k1 = kmx(1)

```

```

351      k2 = kmx1(1)
352      do 360 i = 2, imx1(1)
353      vxs = (z(i,k1)-z(i-1,k1))*q(2,i,k1) -
354      > (x(i,k1)-x(i-1,k1))*q(3,i,k1)
355      if( vxs .lt. 0. ) then
356      c      print*, 'Outgoing bc at i:', i, pval1
357      c      rval2 = q(1,i,k2)
358      c      pval2 = gmm*(q(4,i,k2) - 0.5*( q(2,i,k2)**2 + q(3,i,k2)**2 )
359      c      > / q(1,i,k2))
360      c      cval2 = sqrt(gamma*pval2/rval2)
361      c      uval2 = q(2,i,k2)/q(1,i,k2)
362      c      wval2 = q(3,i,k2)/q(1,i,k2)
363      c      reiml = uval2 + 2.0*rgmm*cval2
364      c      sval2 = pval2/rval2**gamma
365      c      sval1 = sval2
366      c      pval1 = pinf
367      c      if (poten) pval1 = gmm*(q(4,i,k1) -
368      c      > 0.5*( q(2,i,k1)**2 + q(3,i,k1)**2 ) / q(1,i,k1))
369      c      rval1 = (pval1/sval1)**rgamma
370      c      cval1 = sqrt(gamma*pval1/rval1)
371      c      uval1 = reiml - 2.0*rgmm*cval1
372      c      wval1 = wval2
373      c      q(1,i,k1) = rval1
374      c      q(2,i,k1) = rval1*uval1
375      c      q(3,i,k1) = rval1*wval1
376      c      q(4,i,k1) = pval1*rgmm + 0.5*rval1*(uval1**2 + wval1**2)
377      c      pval1 = gmm*(q(4,i,k1) -
378      c      > 0.5*( q(2,i,k1)**2 + q(3,i,k1)**2 ) / q(1,i,k1))
379      c      q(1,i,k1) = (4.*q(1,i,k2) - q(1,i,k2-1))/3.
380      c      q(1,i,k1) = 2.*q(1,i,k2) - q(1,i,k2-1)
381      c      q(2,i,k1) = 2.*q(2,i,k2) - q(2,i,k2-1)
382      c      q(3,i,k1) = 2.*q(3,i,k2) - q(3,i,k2-1)
383      c      q(4,i,k1) = pval1*rgmm +
384      c      > 0.5*( q(2,i,k1)**2 + q(3,i,k1)**2 ) / q(1,i,k1)
385      c      else
386      c      q(1,i,k1) = 2.*q(1,i,k2) - q(1,i,k2-1)
387      c      endif
388      360 continue
389      endif
390      c..boundary condition treatment for the wake ***
391      ng = 1
392      k = 1
393      if = iwks(ng)-1
394      do 400 l = 1,4
395      do 400 i = 1,il
396      il = i
397      iu = imx(ng) - i + 1
398      c..average values on upper and lower surfaces of cut,
399      q(1,il,k) = 0.5*( q(1,iu,k+1) + q(1,il,k+1) )
400      q(1,iu,k) = 0.5*( q(1,iu,k+1) + q(1,il,k+1) )
401      400 continue
402      return
403      end
404      C-----
405      subroutine step
406      include 'coms.f'
407      dimension qold(4,nia,nka)
408      nt = 1
409      c.. store all the q values to facilitate an iterative update ***
410      do 1 l=1,4
411      do 1 i=1,imx(1)
412      do 1 k=1,kmx(1)
413      qold(l,i,k) = q(1,i,k)
414      1 continue
415      c.. update all the q values ***
416      DO 1000 loop = 1, newtit
417      c..update outer bc
418      c      if (poten .and. loop .eq. 1 ) call nspot
419      c      if (.not. poten .and. loop .eq. 1 ) then
420      c..write out dphi/dt related terms..
421      c      rho = q(1,61,39)
422      c      pres = gmm*(q(4,61,39) - 0.5*( q(2,61,39)**2 + q(3,61,39)**2 )
423      c      > / rho )
424      c      v2 = (q(2,61,39)/rho)**2 + (q(3,61,39)/rho)**2
425      c      dfdt = 0.5*(uinf**2-v2) + rgmm*(1.-(gamma*pres)**(gmm/gamma))
426      c write(6,'(3x,e14.4,14x,3e14.4)') dfdt, rho, pres, v2
427      c      endif
428      c..update all the qsi values

```

```

429      do 10 k = 1, kmx(nt)
430      do 10 l = 1, 4
431      do 10 i = 1, imx(nt)
432      qd(l,i,k) = -( q(l,i,k) - qold(l,i,k) )/aja(i,k)
433 10      continue
434      call rhsosh
435      call lhs
436      adromax = 0.0
437      do 20 k = 2, kmx1(nt)
438      do 20 i = 2, imx1(nt)
439      dro = aja(i,k)*qd(l,i,k)
440      adro = abs(dro)
441      if ( adromax .lt. adro) then
442      dromax = dro
443      adromax = adro
444      ires = i
445      kres = k
446      endif
447      if(.not.(adro .lt. 5.0 .and. adro .ge. 0.))then
448      write(6,101) iter, adro, i,k
449      format(// ' *** It has BLOWN UP *** @ ITER =', i5 /
101      >      , drho = ',e11.3, ' @ ', 2i4 )
450      stop
451      endif
452 20      continue
453      do 30 k = 2, kmx1(nt)
454      do 30 l = 1, 4
455      do 30 i = 2, imx1(nt)
456      q(l,i,k) = q(l,i,k) + aja(i,k)*qd(l,i,k)
457 30      continue
458      call bc
459 1000 CONTINUE
460      if( mod(iter,nload).eq.0 .or.
461      >      itr .eq. 1 .or. itr .eq. niter) then
462      call loads
463      write (6,60) iter,alfad,time,dromax,
464      q(1,ires,kres), ires,kres,cm,cd,cl
465      >      format(i5 ,f6.2, 1x,f8.4, e11.3,f9.4, 2i4, 2x,3f8.4)
466      60 ***** C1 loop
467      c      if ((.not.(oscl.or.ramp)) .and. (abs(clo-cl).lt..001))then
468      c      call gio(0)
469      c      stop
470      c      endif
471      c      clo=cl
472      *****
473      elseif( mod(iter,nprint).eq.0) then
474      write (6,60) iter,alfad,time,dromax,
475      q(1,ires,kres), ires,kres
476      >      endif
477      return
478      end
479
480 C-----
481      subroutine rhsosh
482      include 'coms.f'
483      common /dfdq / ap(nia,4,4), am(nia,4,4)
484      common /fmet /aktj(nia),aktnj(nia),
485      >      akxj(nia),akzj(nia),ajac(nia,2),adt(nia)
486      common /flux /qv(nia,2,4),fnum(nia,4),dfp(nia,4,3),dfm(nia,4,3)
487      >      ,dvq(nia,4,3)
488      common /osflxs/ dfpt(nia,4),dfmt(nia,4)
489      common /osvars/ r(nia,6),u(nia,6),v(nia,6),e(nia,6)
490      dimension akx(nia), akz(nia), akt(nia), aktn(nia)
491      nt=1
492      oneby6=1.0/6.0
493
494      c
495      do 1000 k=2,kmx1(nt)
496      c*****
497      c      compute the fluxes for all the segments in the psi direction
498      c*****
499      do 110 i=2,imx(nt)
500      il=i-1
501      i4=i
502      r(i,1)=q(1,il,k)
503      u(i,1)=q(2,il,k)/r(i,1)
504      v(i,1)=q(3,il,k)/r(i,1)
505      e(i,1)=q(4,il,k)
506      r(i,4)=q(1,i4,k)

```

```

507      u(i,4)=q(2,i4,k)/r(i,4)
508      v(i,4)=q(3,i4,k)/r(i,4)
509      e(i,4)=q(4,i4,k)
510      xi_x=0.5*(xix(i1,k)+xix(i4,k))
511      xi_z=0.5*(xiz(i1,k)+xiz(i4,k))
512      xi_t=0.5*(xit(i1,k)+xit(i4,k))
513      ze_t=0.5*(zet(i1,k)+zet(i4,k))
514      aktj(i) = xi_t
515      aktnj(i)= ze_t
516      akxj(i) = xi_x
517      akzj(i) = xi_z
518 110 continue
519      ilft=2
520      irgt=imx(nt)
521      call osflux(ilft,irgt)
522 c*****
523 c      add the fluxes in each subpath
524 c*****
525      do 120 n=1,4
526      do 120 i=ilft,irgt
527      dfpt(i,n)=dfp(i,n,1)+dfp(i,n,2)+dfp(i,n,3)
528      dfmt(i,n)=dfm(i,n,1)+dfm(i,n,2)+dfm(i,n,3)
529      continue
530 c*****
531 c      add the eta flux contribution for the second and last but one
532 c      points (second order accurate fluxes)
533 c*****
534      idif=imx(nt)-3
535      do 130 n=1,4
536      do 130 i=2,imx1(nt),idif
537      ip0=i
538      ip1=i+1
539      qd(n,i,k)=qd(n,i,k)-dt(i,k)*(fnum(ip1,n)-fnum(ip0,n)+
540      > 0.45*(dfpt(ip1,n)-dfpt(ip0,n)-dfmt(ip1,n)+dfmt(ip0,n)))
541 130 continue
542 c*****
543 c      add the eta flux contribution for the points in the interior
544 c      points (third order accurate fluxes)
545 c*****
546      do 140 n=1,4
547      do 140 i=3,imx2(nt)
548      im1=i-1
549      ip0=i
550      ip1=i+1
551      ip2=i+2
552      qd(n,i,k)=qd(n,i,k)-dt(i,k)*(fnum(ip1,n)-fnum(ip0,n)
553      > +oneby6*(2.0*dfpt(ip1,n)-dfpt(ip0,n)-dfpt(im1,n))
554      > +oneby6*(2.0*dfmt(ip0,n)-dfmt(ip1,n)-dfmt(ip2,n)))
555 140 continue
556 1000 continue
557      do 2000 i=2,imx1(nt)
558 c*****
559 c      compute the fluxes for all the segments in the zet direction
560 c*****
561      if( i .lt. iwks(nt) .or. i .gt. iwke(nt) ) then
562      kbot = 1
563      else
564      kbot = 2
565      endif
566      ktop=kmx(nt)
567      do 210 k=kbot,kmx(nt)
568      if(k .eq. 1) then
569      ii = imx(nt)-i+1
570      k1 = 2
571      sign=-1.
572      else
573      ii = i
574      k1 = k-1
575      sign=1.
576      endif
577      k4=k
578      r(k,1)=q(1,ii,k1)
579      u(k,1)=q(2,ii,k1)/r(k,1)
580      v(k,1)=q(3,ii,k1)/r(k,1)
581      e(k,1)=q(4,ii,k1)
582      r(k,4)=q(1,i,k4)
583      u(k,4)=q(2,i,k4)/r(k,4)
584      v(k,4)=q(3,i,k4)/r(k,4)

```

```

585      e(k,4)=q(4,i,k4)
586      ze_x=0.5*(sign*zex(ii,k1)+zex(i,k4))
587      ze_z=0.5*(sign*zez(ii,k1)+zez(i,k4))
588      ze_t=0.5*(sign*zet(ii,k1)+zet(i,k4))
589      xi_t=0.5*(sign*xit(ii,k1)+xit(i,k4))
590      aktj(k)=xi_t
591      aktj(k)=ze_x
592      akxj(k)=ze_x
593      akzj(k)=ze_z
594      210 continue
595      call osflux(kbot,ktop)
596      c*****
597      c      add the fluxes in each subpath
598      c*****
599      do 220 n=1,4
600      do 220 k=kbot,ktop
601      dfpt(k,n)=dfp(k,n,1)+dfp(k,n,2)+dfp(k,n,3)
602      dfmt(k,n)=dfm(k,n,1)+dfm(k,n,2)+dfm(k,n,3)
603      220 continue
604      c*****
605      c      add the eta flux contribution for the last
606      c      point (first, or second order accurate fluxes)
607      c*****
608      c      -- second order at the inner boundaries --
609      c      qd(n,i,k) = qd(n,i,k)-dt(i,k)*(fnum(kp1,n)-fnum(kp0,n)+
610      > 0.45*(dfpt(kp1,n)-dfpt(kp0,n)-dfmt(kp1,n)+dfmt(kp0,n)))
611      do n = 1,4
612      if(kbot .eq. 2 ) then
613      k=2
614      kp0=k
615      kp1=k+1
616      c      qd(n,i,k) = qd(n,i,k)-dt(i,k)*(fnum(kp1,n)-fnum(kp0,n)+
617      c      > 0.45*(dfpt(kp1,n)-dfpt(kp0,n)-dfmt(kp1,n)+dfmt(kp0,n)))
618      qd(n,i,k) = qd(n,i,k)-dt(i,k)*( fnum(kp1,n)-fnum(kp0,n) )
619      endif
620      k=kmx1(nt)
621      kp0=k
622      kp1=k+1
623      qd(n,i,k) = qd(n,i,k)-dt(i,k)*( fnum(kp1,n)-fnum(kp0,n) )
624      k=kmx2(nt)
625      kp0=k
626      kp1=k+1
627      qd(n,i,k) = qd(n,i,k)-dt(i,k)*( fnum(kp1,n)-fnum(kp0,n) )
628      enddo
629      c*****
630      c      add the eta flux contribution for the points in the interior
631      c      points (third order accurate fluxes)
632      c*****
633      do 240 n=1,4
634      do 240 k=kbot+1,kmx2(nt)-1
635      km1=k-1
636      kp0=k
637      kp1=k+1
638      kp2=k+2
639      qd(n,i,k)=qd(n,i,k)-dt(i,k)*(fnum(kp1,n)-fnum(kp0,n)
640      > +oneby6*(2.0*dfpt(kp1,n)-dfpt(kp0,n)-dfpt(km1,n))
641      > +oneby6*(2.0*dfmt(kp0,n)-dfmt(kp1,n)-dfmt(kp2,n)))
642      240 continue
643      2000 continue
644      if( visc ) call oshvrhs
645      return
646      end
647      C-----
648      subroutine lhs
649      parameter (nikp = 213, ninv=61)
650      include 'coms.f'
651      common /ctri /amat(ninv,nikp,4,4), bmat(ninv,nikp,4,4),
652      > cmat(ninv,nikp,4,4), fmat(ninv,nikp,4)
653      common /swvar / rsw(nia),usw(nia),vsw(nia),esw(nia)
654      common /dfdq / ap(nia,4,4), am(nia,4,4)
655      common /fmet /aktj(nia),aktj(nia),
656      > akxj(nia),akzj(nia),ajac(nia,2),adt(nia)
657      common /flux /qv(nia,2,4),fnum(nia,4),dfp(nia,4,3),dfm(nia,4,3)
658      > ,dvq(nia,4,3)
659      common /visdi /vmui(nka), tmui(nka)
660      dimension iline(nikp), kline(nikp)
661      nt=1
662      ng = 1

```

```

663 c..inversion in the psi direction ***
664 kift=2
665 krgt=kmx1(nt)
666 line=0
667 do 1000 k=kift,krgt
668   kpl=k+1
669   km1=k-1
670   line=line+1
671   if(line.gt.ninv) line=line-ninv
672   kline(line)=k
673 c..initialize the matrices ***
674   do 110 l=1,4
675     do 110 i=1,imx(nt)
676       fmat(line,i,l)=qd(1,i,k)
677 110   continue
678 c..calculate the matrices aplus and aminus ***
679   do 120 i=1,imx(nt)
680     yac = aja(i,k)
681     akxj(i) = xix(i,k) * yac
682     akzj(i) = xiz(i,k) * yac
683     aktj(i) = xit(i,k) * yac
684     adt(i) = dt(i,k)
685     ajac(i,1)= aja(i,k)
686     ajac(i,2)= aja(i,k)
687     rsw(i) = q(1,i,k)
688     usw(i) = q(2,i,k)/q(1,i,k)
689     vsw(i) = q(3,i,k)/q(1,i,k)
690     esw(i) = q(4,i,k)
691 120   continue
692     do 130 l=1,4
693       do 130 i=1,imx(nt)
694         qv(i,1,l)=q(1,i,k)
695         qv(i,2,l)=q(1,i,k)
696 130   continue
697     ilft=1
698     irgt=imx(nt)
699     call smatrix(ilft,irgt)
700 c.. calculate the matrices amat, bmat and cmat ***
701     do 140 l=1,4
702       do 140 m=1,4
703         do 140 i=2,imx1(nt)
704           ip = i+1
705           im = i-1
706           amat(line,i,1,m)=-ap(im,1,m)
707           bmat(line,i,1,m)= ap(i,1,m)-am(i,1,m)
708           cmat(line,i,1,m)= am(ip,1,m)
709 140   continue
710 c..add the contribution from the time term ***
711     ilft=1
712     irgt=imx(nt)
713     do 150 l=1,4
714       do 150 i=ilft,irgt
715         bmat(line,i,1,l)=bmat(line,i,1,l)+1.0
716 150   continue
717     i=1
718     do 160 l=1,4
719       fmat(line,i,l)=0.0
720     do 160 m=1,4
721       amat(line,i,1,m)=0.0
722       bmat(line,i,1,m)=0.0
723       cmat(line,i,1,m)=0.0
724 160   continue
725     do 161 l=1,4
726       bmat(line,i,1,l) = 1
727 161   continue
728     i=imx(nt)
729     do 165 l=1,4
730       fmat(line,i,1,l)=0.0
731     do 165 m=1,4
732       amat(line,i,1,m)=0.0
733       bmat(line,i,1,m)=0.0
734       cmat(line,i,1,m)=0.0
735 165   continue
736     do 166 l=1,4
737       bmat(line,i,1,l) = 1
738 166   continue
739     if( line .eq. ninv .or. k. eq. krgt ) then
740

```

```

741 c..solve the block tridiagonal system ***
742     call btri(1,imx(nt),line)
743 c..redefine the rhs vector ***
744     do 170 lcount=1,line
745         kd=kline(lcount)
746         do 170 l=1,4
747             do 170 id=1,imx(nt)
748                 qd(1,id,kd)=fmat(lcount,id,l)
749     170     continue
750     endif
751     1000 continue
752 c..inversion in the zeta direction ***
753     ilft=2
754     irgt=imx1(nt)
755     line=0
756     do 2000 i=ilft,irgt
757         ipl=i+1
758         iml=i-1
759         line=line+1
760         if(line.gt.ninv) line=line-ninv
761         iline(line)=i
762 c..initialize the matrices ***
763         do 210 l=1,4
764             do 210 k=1,kmx(nt)
765                 fmat(line,k,l)=qd(1,i,k)
766     210     continue
767 c..calculate the matrices aplus and aminus ***
768         do 220 k=1,kmx(nt)
769             yac = aja(i,k)
770             akxj(k) = zex(i,k) * yac
771             akzj(k) = zez(i,k) * yac
772             aktj(k) = zet(i,k) * yac
773             adt(k) = dt(i,k)
774             ajac(k,1)= aja(i,k)
775             ajac(k,2)= aja(i,k)
776             rsw(k) = q(1,i,k)
777             usw(k) = q(2,i,k)/q(1,i,k)
778             vsw(k) = q(3,i,k)/q(1,i,k)
779             esw(k) = q(4,i,k)
780     220     continue
781         do 230 l=1,4
782             do 230 k=1,kmx(nt)
783                 qv(k,1,l)=q(1,i,k)
784                 qv(k,2,l)=q(1,i,k)
785     230     continue
786         klft=1
787         krgt=kmx(nt)
788         call smatrix(klft,krgt)
789
790 c..calculate the matrices amat, bmat and cmat ***
791         do 240 l=1,4
792             do 240 m=1,4
793                 do 240 k=2,kmx1(nt)
794                     kp=k+1
795                     km=k-1
796                     amat(line,k,l,m)=ap(km,l,m)
797                     bmat(line,k,l,m)= ap(k,l,m)-am(k,l,m)
798                     cmat(line,k,l,m)= am(kp,l,m)
799     240     continue
800 c..add the viscous eta contribution to the lhs ***
801         if ( visc ) then
802             klft=1
803             krgt=kmx(nt)
804             do 300 k=klft,krgt
805                 vmui(k)=vismu(i,k)
806                 tmui(k)=turmu(i,k)
807                 akxj(k) = zex(i,k)
808                 akzj(k) = zez(i,k)
809                 aktj(k) = zet(i,k)
810     300     continue
811             call vmatrix(klft,krgt)
812             do 310 l=1,4
813                 do 310 m=1,4
814                     do 310 k=2,kmx1(nt)
815                         kp=k+1
816                         km=k-1
817                         amat(line,k,l,m)=amat(line,k,l,m)-ap(km,l,m)
818                         bmat(line,k,l,m)=bmat(line,k,l,m)+ap(k,l,m)+

```

```

819      >                                ap(k,l,m)
820      cmat(line,k,l,m)=cmat(line,k,l,m)-ap(kp,l,m)
821 310      continue
822      endif
823 c..add the contribution from the time term ***
824      klft=1
825      krgt=kmx(nt)
826      do 250 l=1,4
827          do 250 k=klft,krgt
828              bmat(line,k,l,l)=bmat(line,k,l,l)+1.0
829 250      continue
830      k=1
831      do 330 l=1,4
832          fmat(line,k,l)=0.0
833          do 330 m=1,4
834              amat(line,k,l,m)=0.0
835              bmat(line,k,l,m)=0.0
836              cmat(line,k,l,m)=0.0
837 330      continue
838          do 331 m=1,4
839              bmat(line,k,m,m)=1.0
840 331      continue
841      k=kmx(nt)
842      do 340 l=1,4
843          fmat(line,k,l)=0.0
844          do 340 m=1,4
845              amat(line,k,l,m)=0.0
846              bmat(line,k,l,m)=0.0
847              cmat(line,k,l,m)=0.0
848 340      continue
849          do 341 m=1,4
850              bmat(line,k,m,m)=1.0
851 341      continue
852      if( line .eq. ninv .or. i.eq.irgt ) then
853 c..solve the block tridiagonal system ***
854      call btri(1,kmx(nt),line)
855 c..redefine the rhs vector ***
856      do 270 lcount=1,line
857          id=iline(lcount)
858          do 270 l=1,4
859              do 270 kd=1,kmx(nt)
860                  qd(l,id,kd)=fmat(lcount,kd,l)
861 270      continue
862      endif
863 2000      continue
864      return
865      end
866
867 C-----
868      subroutine osflux(lbeg,lend)
869      parameter (nia = 213, nka = 61)
870      common /gamv1 /gamma, gmm, gmp,
871      > rgamma, rgmm, rgmp,
872      > gmbygp
873      common /fmet /aktj(nia),aktnj(nia),
874      > akxj(nia),akzj(nia),ajac(nia,2),adt(nia)
875      common /flux /qv(nia,2,4),fnum(nia,4),dfp(nia,4,3),dfm(nia,4,3)
876      > dvq(nia,4,3)
877      common /osflxs/ dfpt(nia,4),dfmt(nia,4)
878      common /osvars/ r(nia,6),u(nia,6),v(nia,6),e(nia,6)
879      dimension eig11(nia),eig12(nia),eig21(nia),eig22(nia),eig31(nia)
880      dimension eig32(nia),p(nia,6),c(nia,6),fact(nia)
881      dimension fvs(nia,4,6),u_b(6),v_b(6),vqs(nia,4,4)
882 c*****
883 c      define constants for the osher scheme
884 c*****
885      osher=-1.0
886      exp=0.5*gmm
887      rexp=1.0/exp
888 c*****
889 c      calculate the intermediate quantities
890 c*****
891      do 10 l=lbeg,lend
892          p(l,1)=gmm*(e(l,1)-0.5*r(l,1)*(u(l,1)**2+v(l,1)**2))
893          c(l,1)=sqrt(gamma*p(l,1)/r(l,1))
894          p(l,4)=gmm*(e(l,4)-0.5*r(l,4)*(u(l,4)**2+v(l,4)**2))
895          c(l,4)=sqrt(gamma*p(l,4)/r(l,4))
896          fact(l)=sqrt(akxj(l)**2+akzj(l)**2)

```

```

897      ofact =1.0/fact(l)
898      aktj(l) =ofact*aktj(l)
899      akxj(l) =ofact*akxj(l)
900      akzj(l) =ofact*akzj(l)
901      aktnj(l)=ofact*aktnj(l)
902 10      continue
903      do 20 l=lbeg,lend
904      u_b(l)=u(l,l)*akxj(l)+v(l,l)*akzj(l)
905      v_b(l)=v(l,l)*akxj(l)-u(l,l)*akzj(l)
906      u_b(4)=u(l,4)*akxj(l)+v(l,4)*akzj(l)
907      v_b(4)=v(l,4)*akxj(l)-u(l,4)*akzj(l)
908      az=(p(l,4)/p(l,1))**(0.5*rgamma)/sqrt(r(l,4)/r(l,1))
909      u_b(2)=(u_b(4)+osher*rexp*c(l,4)+az*(u_b(1)
910      > -osher*rexp*c(l,1)))/(1.0+az)
911      u_b(3)=u_b(2)
912      v_b(2)=v_b(1)
913      v_b(3)=v_b(4)
914      c(l,2)=c(l,1)+osher*exp*(u_b(2)-u_b(1))
915      c(l,3)=c(l,4)+osher*exp*(u_b(4)-u_b(3))
916      r(l,2)=r(l,1)*(c(l,2)/c(l,1))**rexp
917      r(l,3)=r(l,4)*(c(l,3)/c(l,4))**rexp
918      p(l,2)=rgamma*r(l,2)*c(l,2)**2
919      p(l,3)=p(l,2)
920
921      u(l,2)=akxj(l)*u_b(2)-akzj(l)*v_b(2)
922      v(l,2)=akzj(l)*u_b(2)+akxj(l)*v_b(2)
923      e(l,2)=0.5*r(l,2)*(u(l,2)**2+v(l,2)**2)+rgmm*p(l,2)
924
925      u(l,3)=akxj(l)*u_b(3)-akzj(l)*v_b(3)
926      v(l,3)=akzj(l)*u_b(3)+akxj(l)*v_b(3)
927      e(l,3)=0.5*r(l,3)*(u(l,3)**2+v(l,3)**2)+rgmm*p(l,3)
928      grvel = aktj(l)
929      eig11(l)= ( u_b(1) + grvel ) + osher*c(l,1)
930      eig12(l)= ( u_b(2) + grvel ) + osher*c(l,2)
931      eig21(l)= ( u_b(2) + grvel )
932      eig22(l)= ( u_b(3) + grvel )
933      eig31(l)= ( u_b(3) + grvel ) - osher*c(l,3)
934      eig32(l)= ( u_b(4) + grvel ) - osher*c(l,4)
935 20      continue
936 c*****
937 c      calculate fluxes at each of the nodes for all the segments
938 c*****
939      do 30 m=1,4
940      do 30 l=lbeg,lend
941      Ucon =fact(l)*(aktj(l)+u(l,m)*akxj(l)+v(l,m)*akzj(l))
942      rUcon =r(l,m)*Ucon
943      epp =e(l,m)+p(l,m)
944      pfact =fact(l)*p(l,m)
945      fvs(l,1,m)= rUcon
946      fvs(l,2,m)= rUcon*u(l,m)+akxj(l)*pfact
947      fvs(l,3,m)= rUcon*v(l,m)+akzj(l)*pfact
948      fvs(l,4,m)=epp*Ucon -aktj(l)*pfact
949 30      continue
950 c*****
951 c      calculate dfp for the first path
952 c*****
953      do 40 l=lbeg,lend
954      az=sign(1.0,eig11(l))
955      bz=sign(1.0,eig12(l))
956      cz=az*bz
957      dz=0.25*(cz+abs(cz))*(az+abs(az))
958      dfp(l,1,1)=dz*(fvs(l,1,2)-fvs(l,1,1))
959      dfp(l,2,1)=dz*(fvs(l,2,2)-fvs(l,2,1))
960      dfp(l,3,1)=dz*(fvs(l,3,2)-fvs(l,3,1))
961      dfp(l,4,1)=dz*(fvs(l,4,2)-fvs(l,4,1))
962 c*****
963 c      calculate dfp for the second path
964 c*****
965      az=sign(1.0,eig21(l))
966      dz=0.5*(az+abs(az))
967      dfp(l,1,2)=dz*(fvs(l,1,3)-fvs(l,1,2))
968      dfp(l,2,2)=dz*(fvs(l,2,3)-fvs(l,2,2))
969      dfp(l,3,2)=dz*(fvs(l,3,3)-fvs(l,3,2))
970      dfp(l,4,2)=dz*(fvs(l,4,3)-fvs(l,4,2))
971 c*****
972 c      calculate dfp for the third path
973 c*****
974      az=sign(1.0,eig31(l))

```

```

975      bz=sign(1.0,eig32(1))
976      cz=az*bz
977      dz=0.25*(cz+abs(cz))*(az+abs(az))
978      dfp(1,1,3)=dz*(fvs(1,1,4)-fvs(1,1,3))
979      dfp(1,2,3)=dz*(fvs(1,2,4)-fvs(1,2,3))
980      dfp(1,3,3)=dz*(fvs(1,3,4)-fvs(1,3,3))
981      dfp(1,4,3)=dz*(fvs(1,4,4)-fvs(1,4,3))
982 40      continue
983 c*****
984 c      correction for a sonic point in first path
985 c*****
986      do 50 l=lbeg,lend
987      az=sign(1.0,eig11(1))
988      bz=sign(1.0,eig12(1))
989      if((az*bz).gt.0.0) go to 50
990      u_b(1)=u(1,1)*akxj(1)+v(1,1)*akzj(1)
991      v_b(1)=v(1,1)*akxj(1)-u(1,1)*akzj(1)
992      u_b(5)=gmbygp*(u_b(1)-osher*rexp*c(1,1))
993      c(1,5)=-osher*u_b(5)
994      v_b(5)=v_b(1)
995  c
996      u(1,5)=akxj(1)*u_b(5)-akzj(1)*v_b(5)
997      v(1,5)=akzj(1)*u_b(5)+akxj(1)*v_b(5)
998      r(1,5)=r(1,1)*(c(1,5)/c(1,1))**fexp
999      p(1,5)=r(1,5)*c(1,5)**2/gamma
1000      e(1,5)=0.5*r(1,5)*(u(1,5)**2+v(1,5)**2)+p(1,5)*rgmm
1001      m=5
1002      Ucon =fact(1)*(aktj(1)+u(1,m)*akxj(1)+v(1,m)*akzj(1))
1003      rUcon=r(1,m)*Ucon
1004      epp =e(1,m)+p(1,m)
1005      pfact=fact(1)*p(1,m)
1006      fvs(1,1,m)=rUcon
1007      fvs(1,2,m)=rUcon*u(1,m)+akxj(1)*pfact
1008      fvs(1,3,m)=rUcon*v(1,m)+akzj(1)*pfact
1009      fvs(1,4,m)=epp*Ucon -aktj(1)*pfact
1010      cz=0.5*(az+abs(az))
1011      dz=0.5*(bz+abs(bz))
1012      ez=cz-dz
1013      dfp(1,1,1)=dfp(1,1,1)-cz*fvs(1,1,1)+ez*fvs(1,1,5)+dz*fvs(1,1,2)
1014      dfp(1,2,1)=dfp(1,2,1)-cz*fvs(1,2,1)+ez*fvs(1,2,5)+dz*fvs(1,2,2)
1015      dfp(1,3,1)=dfp(1,3,1)-cz*fvs(1,3,1)+ez*fvs(1,3,5)+dz*fvs(1,3,2)
1016      dfp(1,4,1)=dfp(1,4,1)-cz*fvs(1,4,1)+ez*fvs(1,4,5)+dz*fvs(1,4,2)
1017 50      continue
1018 c*****
1019 c      correction for a sonic point in third path
1020 c*****
1021      do 60 l=lbeg,lend
1022      az=sign(1.0,eig31(1))
1023      bz=sign(1.0,eig32(1))
1024      if((az*bz).gt.0.0) go to 60
1025      u_b(4)=u(1,4)*akxj(1)+v(1,4)*akzj(1)
1026      v_b(4)=v(1,4)*akxj(1)-u(1,4)*akzj(1)
1027      u_b(6)=gmbygp*(u_b(4)+osher*rexp*c(1,4))
1028      c(1,6)=osher*u_b(6)
1029      v_b(6)=v_b(4)
1030  c
1031      u(1,6)=akxj(1)*u_b(6)-akzj(1)*v_b(6)
1032      v(1,6)=akzj(1)*u_b(6)+akxj(1)*v_b(6)
1033      r(1,6)=r(1,4)*(c(1,6)/c(1,4))**fexp
1034      p(1,6)=r(1,6)*c(1,6)**2/gamma
1035      e(1,6)=0.5*r(1,6)*(u(1,6)**2+v(1,6)**2)+p(1,6)*rgmm
1036      m=6
1037      Ucon =fact(1)*(aktj(1)+u(1,m)*akxj(1)+v(1,m)*akzj(1))
1038      rUcon=r(1,m)*Ucon
1039      epp =e(1,m)+p(1,m)
1040      pfact=fact(1)*p(1,m)
1041      fvs(1,1,m)=rUcon
1042      fvs(1,2,m)=rUcon*u(1,m)+akxj(1)*pfact
1043      fvs(1,3,m)=rUcon*v(1,m)+akzj(1)*pfact
1044      fvs(1,4,m)=epp*Ucon -aktj(1)*pfact
1045      cz=0.5*(az+abs(az))
1046      dz=0.5*(bz+abs(bz))
1047      ez=cz-dz
1048      dfp(1,1,3)=dfp(1,1,3)-cz*fvs(1,1,3)+ez*fvs(1,1,6)+dz*fvs(1,1,4)
1049      dfp(1,2,3)=dfp(1,2,3)-cz*fvs(1,2,3)+ez*fvs(1,2,6)+dz*fvs(1,2,4)
1050      dfp(1,3,3)=dfp(1,3,3)-cz*fvs(1,3,3)+ez*fvs(1,3,6)+dz*fvs(1,3,4)
1051      dfp(1,4,3)=dfp(1,4,3)-cz*fvs(1,4,3)+ez*fvs(1,4,6)+dz*fvs(1,4,4)
1052 60      continue

```

```

1053 c*****
1054 c calculate dfm for all the paths and fnum for the segment
1055 c*****
1056 do 70 k=1,4
1057 do 70 l=lbeg,lend
1058 dfm(l,k,1)=fvs(l,k,2)-fvs(l,k,1)-dfp(l,k,1)
1059 dfm(l,k,2)=fvs(l,k,3)-fvs(l,k,2)-dfp(l,k,2)
1060 dfm(l,k,3)=fvs(l,k,4)-fvs(l,k,3)-dfp(l,k,3)
1061 dfpl=dfp(l,k,1)+dfp(l,k,2)+dfp(l,k,3)
1062 fnum(l,k)=fvs(l,k,4)-dfpl
1063 70 continue
1064 c*****
1065 c calculate dvq for all the paths
1066 c*****
1067 do 80 m=1,4
1068 do 80 l=lbeg,lend
1069 az=gmm/p(l,m)
1070 bz=az*r(l,m)
1071 vqs(l,1,m)=gmp-log(p(l,m)/r(l,m)**gamma)-az*e(l,m)
1072 vqs(l,2,m)=bz*u(l,m)
1073 vqs(l,3,m)=bz*v(l,m)
1074 vqs(l,4,m)=-bz
1075 80 continue
1076 do 90 k=1,4
1077 do 90 l=lbeg,lend
1078 dvq(l,k,1)=vqs(l,k,2)-vqs(l,k,1)
1079 dvq(l,k,2)=vqs(l,k,3)-vqs(l,k,2)
1080 dvq(l,k,3)=vqs(l,k,4)-vqs(l,k,3)
1081 90 continue
1082 return
1083 end
1084 C-----
1085 subroutine oshvrhs
1086 include 'coms.f'
1087 common /flux /qv(nia,2,4),fnum(nia,4),dfp(nia,4,3),dfm(nia,4,3)
1088 > ,dvq(nia,4,3)
1089 nt = 1
1090 obyre=1.0/reynnu
1091 call mulam
1092 if ( turbl ) call eddybl
1093 c*****
1094 c compute the eta direction viscous terms
1095 c*****
1096 do 1000 i=1,imx(nt)
1097 ip=i+1
1098 im=i-1
1099 do 20 k=2,kmx(nt)
1100 km=k-1
1101 u_xi=0.0
1102 w_xi=0.0
1103 a_xi=0.0
1104 u0 = q(2,i,k)/q(1,i,k)
1105 u1 = q(2,i,km)/q(1,i,km)
1106 u_ze = u0 - u1
1107 w0 = q(3,i,k)/q(1,i,k)
1108 w1 = q(3,i,km)/q(1,i,km)
1109 w_ze = w0 - w1
1110 a0 = ( q(4,i,k)/q(1,i,k) - 0.5*( u0**2 + w0**2 ) )
1111 a1 = ( q(4,i,km)/q(1,i,km) - 0.5*( u1**2 + w1**2 ) )
1112 a_ze = a0-a1
1113 c*****
1114 c compute the necessary metrics
1115 c*****
1116 xi_x = 0.5*( xix(i,km)+xix(i,k) )
1117 xi_z = 0.5*( xiz(i,km)+xiz(i,k) )
1118 ze_x = 0.5*( zex(i,km)+zex(i,k) )
1119 ze_z = 0.5*( zez(i,km)+zez(i,k) )
1120 ajac = 0.5*( aja(i,km)+aja(i,k) )
1121 c ajac = 1.
1122 c*****
1123 c compute the velocity derivatives w.r.t. x and z
1124 c*****
1125 Ux = ajac*( u_xi*xi_x + u_ze*ze_x )
1126 Wx = ajac*( w_xi*xi_x + w_ze*ze_x )
1127 Ax = ajac*( a_xi*xi_x + a_ze*ze_x )
1128 Uz = ajac*( u_xi*xi_z + u_ze*ze_z )
1129 Wz = ajac*( w_xi*xi_z + w_ze*ze_z )
1130 Az = ajac*( a_xi*xi_z + a_ze*ze_z )

```

```

1131 c*****
1132 c   compute the stress tensors
1133 c*****
1134 Vmu = 0.5*obyre*( vismu(i,km) + vismu(i,k) )
1135 Tmu = 0.5*obyre*( turmu(i,km) + turmu(i,k) )
1136 Cmu = Vmu+Tmu
1137 T_xx = Cmu*( 2.0*Ux-2.0*(Ux+Wz)/3.0 )
1138 T_zz = Cmu*( 2.0*Wz-2.0*(Ux+Wz)/3.0 )
1139 T_xz = Cmu*( Uz+Wx )
1140 Uvel = 0.5*( u0 + u1 )
1141 Wvel = 0.5*( w0 + w1 )
1142 akbycp = Vmu/prkin+Tmu/prtur
1143 gkbycp = gamma * akbycp
1144 Rx      = Uvel*T_xx + Wvel*T_xz + gkbycp*Ax
1145 Sz      = Uvel*T_xz + Wvel*T_zz + gkbycp*Az
1146 c*****
1147 c   compute the numerical fluxes
1148 c*****
1149 fnum(k,1) = 0.0
1150 fnum(k,2) = ze_x*T_xx + ze_z*T_xz
1151 fnum(k,3) = ze_x*T_xz + ze_z*T_zz
1152 fnum(k,4) = ze_x*Rx   + ze_z*Sz
1153 20  continue
1154 do 30 n=1,4
1155 do 30 k=2,kmx1(nt)
1156 kp0=k
1157 kp1=k+1
1158 qd(n,i,k)=qd(n,i,k)+dt(i,k)*( fnum(kp1,n)-fnum(kp0,n) )
1159 30  continue
1160 1000 continue
1161 return
1162 end
1163 C-----
1164 subroutine vmatrix(jkbeg,jkend)
1165 parameter (nia = 213, nka = 61)
1166 common /dfdq / ap(nia,4,4), am(nia,4,4)
1167 common /fmet /aktj(nia),aktj(nia),
1168 > akxj(nia),akzj(nia),ajac(nia,2),adt(nia)
1169 common /flux /qv(nia,2,4),fnum(nia,4),dfp(nia,4,3),dfm(nia,4,3)
1170 > ,dvq(nia,4,3)
1171 common /gamvl /gamma, gmm, gmp,
1172 > rgamma, rgmm, rgmp,
1173 > gmbygp
1174 common /tmval /timeacc, time, dtau,dt(nia,nka),cour
1175 logical timeacc
1176 common /visdi /vmui(nka), tmui(nka)
1177 common /visvl /reynnu,reynph, prkin, prtur
1178 const=1./reynnu
1179 rat1b3=1.0/3.0
1180 rat4b3=4.0/3.0
1181 c
1182 c   *** logic for zeta direction matrices ***
1183 c
1184 do 10 jk=jkbeg,jkend
1185 adm = ajac(jk,1)
1186 zetax = adm*akxj(jk)
1187 zetaz = adm*akzj(jk)
1188 zetaxsq= zetax**2
1189 zetazsq= zetaz**2
1190 c
1191 c   *** compute the viscous parameters ***
1192 c
1193 amu = const*adt(jk)*vmui(jk)
1194 bmu = const*adt(jk)*tmui(jk)
1195 fmu = amu+bmu
1196 akbycp= amu/prkin + bmu/prtur
1197 c
1198 c   *** compute often used terms ***
1199 c
1200 alf0 = gamma*akbycp*(zetaxsq+zetazsq)
1201 alf1 = fmu*(rat4b3*zetaxsq+zetazsq)
1202 alf3 = fmu*rat1b3*zetax*zetaz
1203 alf4 = fmu*(zetaxsq + zetazsq)
1204 alf5 = fmu*rat1b3*zetay*zetaz
1205 alf6 = fmu*(zetaxsq + rat4b3*zetazsq)
1206 rval = 0.5*(qv(jk,1,1)+qv(jk,2,1))
1207 obyr = 1.0/rval
1208 obyr1 = 1.0/qv(jk,1,1)

```

```

1209      obyr2 = 1.0/qv(jk,2,1)
1210      uval = 0.5*(qv(jk,1,2)*obyr1+qv(jk,2,2)*obyr2)
1211      wval = 0.5*(qv(jk,1,3)*obyr1+qv(jk,2,3)*obyr2)
1212      eval = 0.5*(qv(jk,1,4)+qv(jk,2,4))
1213      ubyro = uval*obyr
1214      wbyro = wval*obyr
1215      usqbyr = obyr*uval**2
1216      uwbyr = obyr*uval*wval
1217      wsqbyr = obyr*wval**2
1218      ebyrsq = eval*obyr**2
1219
1220      c      *** compute the viscous matrix ***
1221      c
1222      ap(jk,1,1) = 0.0
1223      ap(jk,1,2) = 0.0
1224      ap(jk,1,3) = 0.0
1225      ap(jk,1,4) = 0.0
1226      ap(jk,2,1) = -alf1*ubyro-alf3*wbyro
1227      ap(jk,2,2) = alf1*obyr
1228      ap(jk,2,3) = alf3*obyr
1229      ap(jk,2,4) = 0.0
1230      ap(jk,3,1) = -alf3*ubyro-alf6*wbyro
1231      ap(jk,3,2) = alf3*obyr
1232      ap(jk,3,3) = alf6*obyr
1233      ap(jk,3,4) = 0.0
1234      bz = -alf1*usqbyr-alf6*wsqbyr
1235      cz = alf0*(usqbyr+wsqbyr-ebyrsq)
1236      ap(jk,4,1) = bz+cz
1237      ap(jk,4,2) = -ap(jk,2,1)-alf0*ubyro
1238      ap(jk,4,3) = -ap(jk,3,1)-alf0*wbyro
1239      ap(jk,4,4) = alf0*obyr
1240      10      continue
1241      return
1242      end
1243      C-----
1244      subroutine smatrix(ikbeg,ikend)
1245      parameter (nia = 213, nka = 61)
1246      common /dfdq / ap(nia,4,4), am(nia,4,4)
1247      common /swvar / rsw(nia),usw(nia),vsw(nia),esw(nia)
1248      common /fmet / aktj(nia),aktnj(nia),
1249      >      akxj(nia),akzj(nia),ajac(nia,2),adt(nia)
1250      common /flux /qv(nia,2,4),fnum(nia,4),dfp(nia,4,3),dfm(nia,4,3)
1251      >      ,dvq(nia,4,3)
1252      common /gamvl /gamma, gmn, gmp,
1253      >      rgamma, rgmn, rgmp,
1254      >      gmbygp
1255      common /tmval /timeacc, time, dtau,dt(nia,nka),cour
1256      logical timeacc
1257
1258      c      dimension akxsq(nia),akzsq(nia),ofact(nia)
1259      dimension eig1(nia),eig2(nia),eig3(nia),eig4(nia),eig5(nia)
1260      dimension eig6(nia),eig7(nia),eig8(nia),eig9(nia),eig10(nia)
1261      dimension eigmd(nia,4),eigmdpl(nia,4),p(nia),c(nia),qsqby2(nia)
1262      eps=0.02
1263
1264      c
1265      do 10 ik=ikbeg,ikend
1266      qsqby2(ik)=0.5*(usw(ik)**2+vsw(ik)**2)
1267      p(ik) =gmn*(esw(ik)-0.5*rsw(ik)*qsqby2(ik))
1268      c(ik) =sqrt(gamma*p(ik)/rsw(ik))
1269      ofact(ik) =sqrt(akxj(ik)**2+akzj(ik)**2)
1270      fact=1.0/ofact(ik)
1271      aktj(ik)=fact*aktj(ik)
1272      akxj(ik)=fact*akxj(ik)
1273      akzj(ik)=fact*akzj(ik)
1274      akxsq(ik)=akxj(ik)**2
1275      akzsq(ik)=akzj(ik)**2
1276      tconst=0.25*adt(ik)/sqrt(1.5)
1277      az=tconst*ofact(ik)
1278      eigmd(ik,1)=az*(aktj(ik)+usw(ik)*akxj(ik)+vsw(ik)*akzj(ik))
1279      eigmd(ik,2)=eigmd(ik,1)
1280      bz=az*c(ik)
1281      eigmd(ik,3)=eigmd(ik,1)+bz
1282      eigmd(ik,4)=eigmd(ik,1)-bz
1283      add=(az*eps)**2
1284      eigmdpl(ik,1)=sqrt(eigmd(ik,1)**2+add)
1285      eigmdpl(ik,2)=sqrt(eigmd(ik,2)**2+add)
1286      eigmdpl(ik,3)=sqrt(eigmd(ik,3)**2+add)
1287      eigmdpl(ik,4)=sqrt(eigmd(ik,4)**2+add)

```

```

1287 10 continue
1288 do 20 ik=ikbeg,ikend
1289 eigmd1=eigmd(ik,1)+eigmdpl(ik,1)
1290 eigmd2=eigmd(ik,2)+eigmdpl(ik,2)
1291 eigmd3=eigmd(ik,3)+eigmdpl(ik,3)
1292 eigmd4=eigmd(ik,4)+eigmdpl(ik,4)
1293 eig1(ik)=eigmd1+eigmd2
1294 eig2(ik)=eigmd3+eigmd4
1295 eig3(ik)=eigmd4-eigmd3
1296 eig4(ik)=eig2(ik)-eig1(ik)
1297 gbar=gmm/c(ik)**2
1298 eig5(ik)=gbar*eig4(ik)
1299 eig6(ik)=gbar*eig3(ik)*c(ik)
1300 eig7(ik)=eig1(ik)*akzsq(ik)+eig2(ik)*akxsq(ik)
1301 eig8(ik)=eig1(ik)*akxsq(ik)+eig2(ik)*akzsq(ik)
1302 eig9(ik)=akxj(ik)*akzj(ik)*eig4(ik)
1303 eig10(ik)=c(ik)*rgmm*eig3(ik)
1304 20 continue
1305 do 30 ik=ikbeg,ikend
1306 az=eig3(ik)/c(ik)
1307 ap(ik,1,1)=eig1(ik)+qsqby2(ik)*eig5(ik)
1308 > + (akxj(ik)*usw(ik)+akzj(ik)*vsw(ik))*az
1309 ap(ik,1,2)=-usw(ik)*eig5(ik)-akxj(ik)*az
1310 ap(ik,1,3)=-vsw(ik)*eig5(ik)-akzj(ik)*az
1311 ap(ik,1,4)=eig5(ik)
1312 az=akxj(ik)*eig6(ik)
1313 ap(ik,2,1)=usw(ik)*ap(ik,1,1)-vsw(ik)*eig9(ik)
1314 > -usw(ik)*eig7(ik)-qsqby2(ik)*az
1315 ap(ik,2,2)=usw(ik)*ap(ik,1,2)+eig7(ik)+usw(ik)*az
1316 ap(ik,2,3)=usw(ik)*ap(ik,1,3)+eig9(ik)+vsw(ik)*az
1317 ap(ik,2,4)=usw(ik)*ap(ik,1,4)-az
1318 az=akzj(ik)*eig6(ik)
1319 ap(ik,3,1)=vsw(ik)*ap(ik,1,1)-usw(ik)*eig9(ik)
1320 > -vsw(ik)*eig8(ik)-qsqby2(ik)*az
1321 ap(ik,3,2)=vsw(ik)*ap(ik,1,2)+eig9(ik)+usw(ik)*az
1322 ap(ik,3,3)=vsw(ik)*ap(ik,1,3)+eig8(ik)+vsw(ik)*az
1323 ap(ik,3,4)=vsw(ik)*ap(ik,1,4)-az
1324 ap(ik,4,1)=-qsqby2(ik)*ap(ik,1,1)+usw(ik)*ap(ik,2,1)
1325 > +vsw(ik)*ap(ik,3,1)+qsqby2(ik)*eig2(ik)
1326 > +eig10(ik)*(usw(ik)*akxj(ik)+vsw(ik)*akzj(ik))
1327 ap(ik,4,2)=-qsqby2(ik)*ap(ik,1,2)+usw(ik)*ap(ik,2,2)
1328 > +vsw(ik)*ap(ik,3,2)-usw(ik)*eig2(ik)-eig10(ik)*akxj(ik)
1329 ap(ik,4,3)=-qsqby2(ik)*ap(ik,1,3)+usw(ik)*ap(ik,2,3)
1330 > +vsw(ik)*ap(ik,3,3)-vsw(ik)*eig2(ik)-eig10(ik)*akzj(ik)
1331 ap(ik,4,4)=-qsqby2(ik)*ap(ik,1,4)+usw(ik)*ap(ik,2,4)
1332 > +vsw(ik)*ap(ik,3,4)+eig2(ik)
1333 30 continue
1334 do 40 ik=ikbeg,ikend
1335 eigmd1=eigmd(ik,1)-eigmdpl(ik,1)
1336 eigmd2=eigmd(ik,2)-eigmdpl(ik,2)
1337 eigmd3=eigmd(ik,3)-eigmdpl(ik,3)
1338 eigmd4=eigmd(ik,4)-eigmdpl(ik,4)
1339 eig1(ik)=eigmd1+eigmd2
1340 eig2(ik)=eigmd3+eigmd4
1341 eig3(ik)=eigmd4-eigmd3
1342 eig4(ik)=eig2(ik)-eig1(ik)
1343 gbar=gmm/c(ik)**2
1344 eig5(ik)=gbar*eig4(ik)
1345 eig6(ik)=gbar*eig3(ik)*c(ik)
1346 eig7(ik)=eig1(ik)*akzsq(ik)+eig2(ik)*akxsq(ik)
1347 eig8(ik)=eig1(ik)*akxsq(ik)+eig2(ik)*akzsq(ik)
1348 eig9(ik)=akxj(ik)*akzj(ik)*eig4(ik)
1349 eig10(ik)=c(ik)*rgmm*eig3(ik)
1350 40 continue
1351 do 50 ik=ikbeg,ikend
1352 az=eig3(ik)/c(ik)
1353 am(ik,1,1)=eig1(ik)+qsqby2(ik)*eig5(ik)
1354 > + (akxj(ik)*usw(ik)+akzj(ik)*vsw(ik))*az
1355 am(ik,1,2)=-usw(ik)*eig5(ik)-akxj(ik)*az
1356 am(ik,1,3)=-vsw(ik)*eig5(ik)-akzj(ik)*az
1357 am(ik,1,4)=eig5(ik)
1358 az=akxj(ik)*eig6(ik)
1359 am(ik,2,1)=usw(ik)*am(ik,1,1)-vsw(ik)*eig9(ik)
1360 > -usw(ik)*eig7(ik)-qsqby2(ik)*az
1361 am(ik,2,2)=usw(ik)*am(ik,1,2)+eig7(ik)+usw(ik)*az
1362 am(ik,2,3)=usw(ik)*am(ik,1,3)+eig9(ik)+vsw(ik)*az
1363 am(ik,2,4)=usw(ik)*am(ik,1,4)-az
1364 az=akzj(ik)*eig6(ik)

```

```

1365      am(ik,3,1)=vsw(ik)*am(ik,1,1)-usw(ik)*eig9(ik)
1366      >      -vsw(ik)*eig8(ik)-qsqby2(ik)*az
1367      am(ik,3,2)=vsw(ik)*am(ik,1,2)+eig9(ik)+usw(ik)*az
1368      am(ik,3,3)=vsw(ik)*am(ik,1,3)+eig8(ik)+vsw(ik)*az
1369      am(ik,3,4)=vsw(ik)*am(ik,1,4)-az
1370      am(ik,4,1)=-qsqby2(ik)*am(ik,1,1)+usw(ik)*am(ik,2,1)
1371      >      +vsw(ik)*am(ik,3,1)+qsqby2(ik)*eig2(ik)
1372      >      +eig10(ik)*(usw(ik)*akxj(ik)+vsw(ik)*akzj(ik))
1373      am(ik,4,2)=-qsqby2(ik)*am(ik,1,2)+usw(ik)*am(ik,2,2)
1374      >      +vsw(ik)*am(ik,3,2)-usw(ik)*eig2(ik)-eig10(ik)*akxj(ik)
1375      am(ik,4,3)=-qsqby2(ik)*am(ik,1,3)+usw(ik)*am(ik,2,3)
1376      >      +vsw(ik)*am(ik,3,3)-vsw(ik)*eig2(ik)-eig10(ik)*akzj(ik)
1377      am(ik,4,4)=-qsqby2(ik)*am(ik,1,4)
1378      >      +usw(ik)*am(ik,2,4)+vsw(ik)*am(ik,3,4)+eig2(ik)
1379 50      continue
1380      return
1381      end
1382  C-----
1383      subroutine btri(lmin,lmax,itrmax)
1384      parameter (nikp = 213, ninv=61)
1385      common /ctri /amat(ninv,nikp,4,4), bmat(ninv,nikp,4,4),
1386      >      cmat(ninv,nikp,4,4), fmat(ninv,nikp,4)
1387      dimension      dum(nikp,4)
1388      lmax=lmax-1
1389  C*****
1390  c      lu decompose the first b block and put the elements back in this
1391  c      b block (the diagonals contain the reciprocals of the
1392  c      diagonals of the lower triangular matrix)
1393  C*****
1394      l=1
1395      do 10 i=1,itrmax
1396      bmat(i,1,1,1)=1.0/bmat(i,1,1,1)
1397      bmat(i,1,1,2)=bmat(i,1,1,1)*bmat(i,1,1,2)
1398      bmat(i,1,1,3)=bmat(i,1,1,1)*bmat(i,1,1,3)
1399      bmat(i,1,1,4)=bmat(i,1,1,1)*bmat(i,1,1,4)
1400      bmat(i,1,2,1)=bmat(i,1,2,1)
1401      bmat(i,1,2,2)=1.0/(bmat(i,1,2,2)-bmat(i,1,2,1)*bmat(i,1,1,2))
1402      bmat(i,1,2,3)=bmat(i,1,2,3)*(bmat(i,1,2,3)-bmat(i,1,2,1)*
1403      >      bmat(i,1,1,3))
1404      bmat(i,1,2,4)=bmat(i,1,2,4)*(bmat(i,1,2,4)-bmat(i,1,2,1)*
1405      >      bmat(i,1,1,4))
1406  10      continue
1407      do 15 i=1,itrmax
1408      bmat(i,1,3,1)=bmat(i,1,3,1)
1409      bmat(i,1,3,2)=bmat(i,1,3,2)-bmat(i,1,3,1)*bmat(i,1,1,2)
1410      bmat(i,1,3,3)=1.0/(bmat(i,1,3,3)-bmat(i,1,3,1)*bmat(i,1,1,3)-
1411      >      bmat(i,1,3,2)*bmat(i,1,2,3))
1412      bmat(i,1,3,4)=bmat(i,1,3,4)*(bmat(i,1,3,4)-bmat(i,1,3,1)*
1413      >      bmat(i,1,1,4)-bmat(i,1,3,2)*bmat(i,1,2,4))
1414      bmat(i,1,4,1)=bmat(i,1,4,1)
1415      bmat(i,1,4,2)=bmat(i,1,4,2)-bmat(i,1,4,1)*bmat(i,1,1,2)
1416      bmat(i,1,4,3)=bmat(i,1,4,3)-bmat(i,1,4,1)*bmat(i,1,1,3)-
1417      >      bmat(i,1,4,2)*bmat(i,1,2,3)
1418      bmat(i,1,4,4)=1.0/(bmat(i,1,4,4)-bmat(i,1,4,1)*bmat(i,1,1,4)-
1419      >      bmat(i,1,4,2)*bmat(i,1,2,4) -
1420      >      bmat(i,1,4,3)*bmat(i,1,3,4))
1421  15      continue
1422  C*****
1423  c      unitize the first b block
1424  C*****
1425      do 20 i=1,itrmax
1426      fmat(i,1,1)=bmat(i,1,1,1)*fmat(i,1,1)
1427      fmat(i,1,2)=bmat(i,1,2,2)*(fmat(i,1,2)-bmat(i,1,2,1)*fmat(i,1,1))
1428      fmat(i,1,3)=bmat(i,1,3,3)*(fmat(i,1,3)-bmat(i,1,3,1)*fmat(i,1,1)-
1429      >      bmat(i,1,3,2)*fmat(i,1,2))
1430      fmat(i,1,4)=bmat(i,1,4,4)*(fmat(i,1,4)-bmat(i,1,4,1)*fmat(i,1,1)-
1431      >      bmat(i,1,4,2)*fmat(i,1,2)-bmat(i,1,4,3)*fmat(i,1,3))
1432      fmat(i,1,3)=fmat(i,1,3)-bmat(i,1,3,4)*fmat(i,1,4)
1433      fmat(i,1,2)=fmat(i,1,2)-bmat(i,1,2,3)*fmat(i,1,3)-bmat(i,1,2,4)*
1434      >      fmat(i,1,4)
1435      fmat(i,1,1)=fmat(i,1,1)-bmat(i,1,1,2)*fmat(i,1,2)-bmat(i,1,1,3)*
1436      >      fmat(i,1,3)-bmat(i,1,1,4)*fmat(i,1,4)
1437  20      continue
1438      do 30 m=1,4
1439      do 30 i=1,itrmax
1440      cmat(i,1,1,m)=bmat(i,1,1,1)*cmat(i,1,1,m)
1441      cmat(i,1,2,m)=bmat(i,1,2,2)*(cmat(i,1,2,m)-bmat(i,1,2,1)*
1442      >      cmat(i,1,1,m))

```

```

1443      cmat(i,1,3,m)=bmat(i,1,3,3)*(cmat(i,1,3,m)-bmat(i,1,3,1)*
1444      >      cmat(i,1,1,m)-bmat(i,1,3,2)*cmat(i,1,2,m))
1445      cmat(i,1,4,m)=bmat(i,1,4,4)*(cmat(i,1,4,m)-bmat(i,1,4,1)*
1446      >      cmat(i,1,1,m)-bmat(i,1,4,2)*cmat(i,1,2,m)-
1447      >      bmat(i,1,4,3)*cmat(i,1,3,m))
1448      cmat(i,1,3,m)=cmat(i,1,3,m)-bmat(i,1,3,4)*cmat(i,1,4,m)
1449      cmat(i,1,2,m)=cmat(i,1,2,m)-bmat(i,1,2,3)*cmat(i,1,3,m)-
1450      >      bmat(i,1,2,4)*cmat(i,1,4,m)
1451      cmat(i,1,1,m)=cmat(i,1,1,m)-bmat(i,1,1,2)*cmat(i,1,2,m)-
1452      >      bmat(i,1,1,3)*cmat(i,1,3,m)-
1453      >      bmat(i,1,1,4)*cmat(i,1,4,m)
1454 30      continue
1455 c*****
1456 c      upper triangularize the block tridiagonal matrix
1457 c*****
1458      do 40 l=2,lmax
1459      lm=l-1
1460 c*****
1461 c      add -a(l)*f(l-1) to f(l) and -a(l)*c(l-1) to b(l)
1462 c*****
1463      do 50 k=1,4
1464      do 50 i=1,itrmax
1465      dum(i,k)=fmat(i,lm,k)
1466 50      continue
1467      do 60 k=1,4
1468      do 60 i=1,itrmax
1469      fmat(i,1,k)=fmat(i,1,k)-
1470      >      amat(i,1,k,1)*dum(i,1)-amat(i,1,k,2)*dum(i,2)-
1471      >      amat(i,1,k,3)*dum(i,3)-amat(i,1,k,4)*dum(i,4)
1472 60      continue
1473      do 70 k=1,4
1474      do 70 m=1,4
1475      do 70 i=1,itrmax
1476      bmat(i,1,k,m)=bmat(i,1,k,m)-amat(i,1,k,1)*cmat(i,lm,1,m)-
1477      >      amat(i,1,k,2)*cmat(i,lm,2,m)-
1478      >      amat(i,1,k,3)*cmat(i,lm,3,m)-
1479      >      amat(i,1,k,4)*cmat(i,lm,4,m)
1480 70      continue
1481 c*****
1482 c      lu decompose the b(l) block and put the elements back in this
1483 c      b block (the diagonals contain the reciprocals of the
1484 c      diagonals of the lower triangular matrix)
1485 c*****
1486      do 80 i=1,itrmax
1487      bmat(i,1,1,1)=1.0/bmat(i,1,1,1)
1488      bmat(i,1,1,2)=bmat(i,1,1,1)*bmat(i,1,1,2)
1489      bmat(i,1,1,3)=bmat(i,1,1,1)*bmat(i,1,1,3)
1490      bmat(i,1,1,4)=bmat(i,1,1,1)*bmat(i,1,1,4)
1491      bmat(i,1,2,1)=bmat(i,1,2,1)
1492      bmat(i,1,2,2)=1.0/(bmat(i,1,2,2)-bmat(i,1,2,1)*bmat(i,1,1,2))
1493      bmat(i,1,2,3)=bmat(i,1,2,2)*(bmat(i,1,2,3)-bmat(i,1,2,1)*
1494      >      bmat(i,1,1,3))
1495      bmat(i,1,2,4)=bmat(i,1,2,2)*(bmat(i,1,2,4)-bmat(i,1,2,1)*
1496      >      bmat(i,1,1,4))
1497 80      continue
1498      do 85 i=1,itrmax
1499      bmat(i,1,3,1)=bmat(i,1,3,1)
1500      bmat(i,1,3,2)=bmat(i,1,3,2)-bmat(i,1,3,1)*bmat(i,1,1,2)
1501      bmat(i,1,3,3)=1.0/(bmat(i,1,3,3)-bmat(i,1,3,1)*bmat(i,1,1,3)-
1502      >      bmat(i,1,3,2)*bmat(i,1,2,3))
1503      bmat(i,1,3,4)=bmat(i,1,3,3)*(bmat(i,1,3,4)-bmat(i,1,3,1)*
1504      >      bmat(i,1,1,4)-bmat(i,1,3,2)*bmat(i,1,2,4))
1505      bmat(i,1,4,1)=bmat(i,1,4,1)
1506      bmat(i,1,4,2)=bmat(i,1,4,2)-bmat(i,1,4,1)*bmat(i,1,1,2)
1507      bmat(i,1,4,3)=bmat(i,1,4,3)-bmat(i,1,4,1)*bmat(i,1,1,3)-
1508      >      bmat(i,1,4,2)*bmat(i,1,2,3)
1509      bmat(i,1,4,4)=1.0/(bmat(i,1,4,4)-bmat(i,1,4,1)*bmat(i,1,1,4)-
1510      >      bmat(i,1,4,2)*bmat(i,1,2,4)-
1511      >      bmat(i,1,4,3)*bmat(i,1,3,4))
1512 85      continue
1513 c*****
1514 c      unitize the b(l) block
1515 c*****
1516      do 90 i=1,itrmax
1517      fmat(i,1,1)=bmat(i,1,1,1)*fmat(i,1,1)
1518      fmat(i,1,2)=bmat(i,1,2,2)*(fmat(i,1,2)-bmat(i,1,2,1)*fmat(i,1,1))
1519      fmat(i,1,3)=bmat(i,1,3,3)*(fmat(i,1,3)-bmat(i,1,3,1)*fmat(i,1,1)-
1520      >      bmat(i,1,3,2)*fmat(i,1,2))

```

```

1521      fmat(i,1,4)=bmat(i,1,4,4)*(fmat(i,1,4)-bmat(i,1,4,1)*fmat(i,1,1)-
1522      >      bmat(i,1,4,2)*fmat(i,1,2)-bmat(i,1,4,3)*fmat(i,1,3))
1523      fmat(i,1,3)=fmat(i,1,3)-bmat(i,1,3,4)*fmat(i,1,4)
1524      fmat(i,1,2)=fmat(i,1,2)-bmat(i,1,2,3)*fmat(i,1,3)-bmat(i,1,2,4)*
1525      >      fmat(i,1,4)
1526      fmat(i,1,1)=fmat(i,1,1)-bmat(i,1,1,2)*fmat(i,1,2)-bmat(i,1,1,3)*
1527      >      fmat(i,1,3)-bmat(i,1,1,4)*fmat(i,1,4)
1528  90      continue
1529      do 100 m=1,4
1530      do 100 i=1,itrmax
1531      cmat(i,1,1,m)=bmat(i,1,1,1)*cmat(i,1,1,m)
1532      cmat(i,1,2,m)=bmat(i,1,2,2)*(cmat(i,1,2,m)-bmat(i,1,2,1)*
1533      >      cmat(i,1,1,m))
1534      cmat(i,1,3,m)=bmat(i,1,3,3)*(cmat(i,1,3,m)-bmat(i,1,3,1)*
1535      >      cmat(i,1,1,m)-bmat(i,1,3,2)*cmat(i,1,2,m))
1536      cmat(i,1,4,m)=bmat(i,1,4,4)*(cmat(i,1,4,m)-bmat(i,1,4,1)*
1537      >      cmat(i,1,1,m)-bmat(i,1,4,2)*cmat(i,1,2,m)-
1538      >      bmat(i,1,4,3)*cmat(i,1,3,m))
1539      cmat(i,1,3,m)=cmat(i,1,3,m)-bmat(i,1,3,4)*cmat(i,1,4,m)
1540      cmat(i,1,2,m)=cmat(i,1,2,m)-bmat(i,1,2,3)*cmat(i,1,3,m)-
1541      >      bmat(i,1,2,4)*cmat(i,1,4,m)
1542      cmat(i,1,1,m)=cmat(i,1,1,m)-bmat(i,1,1,2)*cmat(i,1,2,m)-
1543      >      bmat(i,1,1,3)*cmat(i,1,3,m)-
1544      >      bmat(i,1,1,4)*cmat(i,1,4,m)
1545  100      continue
1546  40      continue
1547  c*****
1548  c      perform the back substitution
1549  c*****
1550      do 110 l=lmxm,1,-1
1551      lp=l+1
1552      do 120 m=1,4
1553      do 120 i=1,itrmax
1554      dum(i,m)=fmat(i,lp,m)
1555  120      continue
1556      do 110 m=1,4
1557      do 110 i=1,itrmax
1558      fmat(i,1,m)=fmat(i,1,m)-
1559      >      cmat(i,1,m,1)*dum(i,1)-cmat(i,1,m,2)*dum(i,2)-
1560      >      cmat(i,1,m,3)*dum(i,3)-cmat(i,1,m,4)*dum(i,4)
1561  110      continue
1562      return
1563      end
1564  C-----
1565      subroutine mulam
1566      include 'coms.f'
1567      nt=1
1568      cinfsg=gamma*pinf/rinf
1569
1570      do 10 i=1,imx(nt)
1571      do 10 k=1,kmx(nt)
1572      rsqqsq = q(2,i,k)**2+q(3,i,k)**2
1573      pval = gmm*(q(4,i,k)-0.5*rsqqsq/q(1,i,k))
1574      cvalsq = gamma*pval/q(1,i,k)
1575      tval = tinf*cvalsq/cinfsg
1576      az = (tinf+198.6)/(tval+198.6)
1577      vismu(i,k) = az*(tval/tinf)**1.5
1578      turmu(i,k) = 0.
1579  10      continue
1580      return
1581      end
1582  C-----
1583      subroutine eddybl
1584      include 'coms.f'
1585
1586      dimension turmui(nka),turmuo(nka),fval(nka),snor(nka),vort(nka)
1587      dimension u_xi(nka),u_ze(nka), w_xi(nka),w_ze(nka), qtot(nka)
1588  c      constants for the turbulence model
1589      aplus=26.0
1590      ccp=1.6
1591      ckleb=0.3
1592      cwk=0.25
1593      smallk=0.4
1594      capk=0.0168
1595      cmutm=14.0
1596      imax = imx(1)
1597      itel = iwks(1)
1598      iteu = iwke(1)

```

```

1599      kmax = kmx(1)
1600      c calculate the eddy viscosity
1601      c..eddy is scaled down in wake
1602      do 10 i=itel,iteu
1603      c do 10 i=2,imax-1
1604      c calculate the magnitude of the vorticity and total velocity
1605      u_ze(1)=-1.5*(q(2,i,1)/q(1,i,1))+2.0*(q(2,i,2)/q(1,i,2))
1606      > -0.5*(q(2,i,3)/q(1,i,3))
1607      w_ze(1)=-1.5*(q(3,i,1)/q(1,i,1))+2.0*(q(3,i,2)/q(1,i,2))
1608      > -0.5*(q(3,i,3)/q(1,i,3))
1609      do 20 k=2,kmax-1
1610      km=k-1
1611      kp=k+1
1612      u_ze(k)=0.5*(q(2,i,kp)/q(1,i,kp)-q(2,i,km)/q(1,i,km))
1613      w_ze(k)=0.5*(q(3,i,kp)/q(1,i,kp)-q(3,i,km)/q(1,i,km))
1614      20 continue
1615      if(i.eq. 1) then
1616      do 40 k=1,kmax
1617      ip=i+1
1618      u_xi(k)=( q(2,ip,k)/q(1,ip,k)-q(2,i,k)/q(1,i,k) )
1619      w_xi(k)=( q(3,ip,k)/q(1,ip,k)-q(3,i,k)/q(1,i,k) )
1620      40 continue
1621      else
1622      do 60 k=1,kmax-1
1623      ip=i+1
1624      im=i-1
1625      u_xi(k)=0.5*( q(2,ip,k)/q(1,ip,k)-q(2,im,k)/q(1,im,k) )
1626      w_xi(k)=0.5*( q(3,ip,k)/q(1,ip,k)-q(3,im,k)/q(1,im,k) )
1627      60 continue
1628      endif
1629      do 70 k=1,kmax-1
1630      dudz=( u_xi(k)*xiz(i,k)+u_ze(k)*zez(i,k) )*aja(i,k)
1631      dwdx=( w_xi(k)*xix(i,k)+w_ze(k)*zex(i,k) )*aja(i,k)
1632      vort(k)=abs(dudz-dwdx)
1633      qtot(k)=sqrt(q(2,i,k)**2+q(3,i,k)**2)/q(1,i,k)
1634      70 continue
1635      c
1636      c calculate the distance normal to the body
1637      c
1638      snor(1)=0.0
1639      do 80 k=2,kmax
1640      km=k-1
1641      az=x(i,k)-x(i,km)
1642      bz=z(i,k)-z(i,km)
1643      snor(k)=snor(km)+sqrt(az**2+bz**2)
1644      80 continue
1645      c calculate the exponent for the exponential term
1646      k=1
1647      c..by vorticity...
1648      c yac = aja(i,k)
1649      c ux=( u_xi(k)*xix(i,k)+u_ze(k)*zex(i,k) ) * yac
1650      c wx=( w_xi(k)*xix(i,k)+w_ze(k)*zex(i,k) ) * yac
1651      c uz=( u_xi(k)*xiz(i,k)+u_ze(k)*zez(i,k) ) * yac
1652      c wz=( w_xi(k)*xiz(i,k)+w_ze(k)*zez(i,k) ) * yac
1653      c fmu=vismu(i,k)
1654      c tauxx=fmu*(2.0*ux-2.0*(ux+wz)/3.0)
1655      c tauxz=fmu*(uz+wx)
1656      c tauzz=fmu*(2.0*wz-2.0*(ux+wz)/3.0)
1657      c ze_x = zex(i,k)
1658      c ze_z = zez(i,k)
1659      c fact= 1.0/sqrt( ze_x**2+ze_z**2)
1660      c ak1 = fact*ze_z
1661      c ak2 =-fact*ze_x
1662      c tauwal=abs( (tauxx-tauzz)*ak1*ak2+tauxz*(ak2**2-ak1**2))
1663      c expnnt=sqrt(q(1,i,k)*tauwal)/(vismu(i,k)*aplus)
1664      c expnnt=expnnt*sqrt(reynnu)
1665      c EXPNNT = SQRT( REYNNU*q(1,I,K)*VORT(K) ) / (VISMU(I,K)*APLUS)
1666      c calculate the eddy viscosity for the inner region
1667      c mt_inner = rho * (1**2) * vort
1668      do 90 k=1,kmax-1
1669      alen=smallk*snor(k)*(1.0-exp(-expnnt*snor(k)))
1670      turmui(k)=reynnu*q(1,i,k)*vort(k)*alen**2
1671      90 continue
1672      c calculate the eddy viscosity for the outer region
1673      do k=1,kmax-1
1674      fval(k)=snor(k)*vort(k)*(1.0-exp(-expnnt*snor(k)))
1675      enddo
1676      fmax=0.0

```

```

1677      do 110 k=3,kmax-1
1678      if(fmax.le.fval(k) .or. fmax.lt.fval(k+1)) go to 115
1679      ypl=expnnt*aplns*snor(k)
1680      if(ypl.gt.30) go to 120
1681 115      continue
1682      fmax=fval(k)
1683      ks=k
1684      continue
1685 120      continue
1686      ksm=ks-1
1687      ksp=ks+1
1688      az=(fval(ks)-fval(ksm))/(snor(ks)-snor(ksm))
1689      bz=(fval(ksp)-fval(ks))/(snor(ksp)-snor(ks))
1690      aval=(bz-az)/(snor(ksp)-snor(ksm))
1691      bval=az-aval*(snor(ks)+snor(ksm))
1692      aval=aval+1.0e-08*sign(1.0,aval)
1693      snormx=-0.5*bval/aval
1694      snormx=max(snormx,snor(ksm))
1695      snormx=min(snormx,snor(ksp))
1696      klft=ksm
1697      if(snormx.gt.snor(ks)) klft=ks
1698      krgt=klft+1
1699      frc=(snormx-snor(klft))/(snor(krgt)-snor(klft))
1700      fmax=fval(klft)+frc*(fval(krgt)-fval(klft))
1701      qmax=-100000.0
1702      qmin=100000.0
1703      do 130 k=1,kmax-1
1704      qmax=max(qmax,qtot(k))
1705      qmin=min(qmin,qtot(k))
1706 130      continue
1707      qdif=qmax-qmin
1708      az=snormx*fmax
1709      bz=cwk*snormx*qdif*qdif/(fmax+1.0e-08*sign(1.0,fmax))
1710  c      fwake=min(az,bz)
1711      if( i .gt. itel .and. i .lt. iteu ) then
1712          fwake=az
1713      else
1714          fwake=bz
1715      endif
1716      const=capk*ccp*fwake*reynnu
1717      do k=1,kmax-1
1718          fkleb=1.0/(1.0+5.5*(ckleb*snor(k)/snormx)**6)
1719          turmuo(k)=const*fkleb*q(1,i,k)
1720      enddo
1721  c..choose from the inner and outer eddy viscosity values
1722      ainner=1.0
1723      do k=1,kmax-1
1724      if(turmui(k).gt.turmuo(k)) ainner=0.0
1725      turmu(i,k)=ainner*turmui(k)+(1.0-ainner)*turmuo(k)
1726      enddo
1727 10      continue
1728  c..eddy is scaled down in wake
1729      do i = 2,itel
1730          iu = imax - i + 1
1731          fac = 1./(1.+ (x(i,1)-x(itel,1))**3)
1732          do k = 1,kmax-1
1733              turmu(i,k) = turmu(itel+1,k)*fac
1734              turmu(iu,k) = turmu(iteu-1,k)*fac
1735          enddo
1736      enddo
1737  c      if(itr .eq. niter) then
1738  c          open(unit=30,file='turmu.d',form='formatted')
1739  c          ip1 = 71
1740  c          ip2 = 100
1741  c          ip3 = 101
1742  c          ip4 = 102
1743  c          ip5 = 115
1744  c          write(30,'(f5.2,5e14.6)') ( float(k),
1745  c              > turmu(ip1,k), turmu(ip2,k), turmu(ip3,k),
1746  c              > turmu(ip4,k), turmu(ip5,k),
1747  c              > k=2, kmx(1))
1748  c          endif
1749      return
1750      end
1751  C-----
1752      subroutine metric
1753      include 'coms.f'
1754      dimension ajamax(nia),ajamin(nia)

```

```

1755
1756      data eps /1.e-26/
1757 c*****
1758      nt = 1
1759      ng = 1
1760      do 100 i=1,imx(nt)
1761          iml = i - 1
1762          ip1 = i + 1
1763          do 100 k = 1,kmx(nt)
1764              if( i .eq. 1 ) then
1765                  xxi = x(2,k) - x(1,k)
1766                  zxi = z(2,k) - z(1,k)
1767              elseif( i .eq. imx(nt) ) then
1768                  xxi = x(imx(nt),k) - x(imx1(nt),k)
1769                  zxi = z(imx(nt),k) - z(imx1(nt),k)
1770              else
1771                  xxi = 0.5 * ( x(ip1,k) - x(iml,k) )
1772                  zxi = 0.5 * ( z(ip1,k) - z(iml,k) )
1773              endif
1774              if( k .eq. 1 ) then
1775                  xze = 2.*x(i,2) - 1.5*x(i,1) - 0.5*x(i,3)
1776                  zze = 2.*z(i,2) - 1.5*z(i,1) - 0.5*z(i,3)
1777              elseif( k .eq. kmx(nt) ) then
1778                  xze = 1.5*x(i,kmx(nt)) - 2.*x(i,kmx1(nt)) + 0.5*x(i,kmx2(nt))
1779                  zze = 1.5*z(i,kmx(nt)) - 2.*z(i,kmx1(nt)) + 0.5*z(i,kmx2(nt))
1780              else
1781                  kml = k - 1
1782                  kpl = k + 1
1783                  xze = 0.5 * ( x(i,kpl) - x(i,kml) )
1784                  zze = 0.5 * ( z(i,kpl) - z(i,kml) )
1785              endif
1786              xix(i,k) = zze
1787              xiz(i,k) = -xze
1788              zex(i,k) = -zxi
1789              zez(i,k) = xxi
1790 c          xix(i,k) = bjac * zze
1791 c          xiz(i,k) = -bjac * xze
1792 c          zex(i,k) = -bjac * zxi
1793 c          zez(i,k) = bjac * xxi
1794          xdot = omega * z(i,k)
1795          zdot = -omega * x(i,k)
1796          xit(i,k) = -xdot*xix(i,k) - zdot*xiz(i,k)
1797          zet(i,k) = -xdot*zex(i,k) - zdot*zez(i,k)
1798          yacob = ( xxi*zze - xze*zxi )
1799          if ( yacob .eq. 0. ) then
1800              print *, 'zero jac at ', i,k
1801              yacob = eps
1802          endif
1803          aja(i,k) = 1.0 / yacob
1804      100 continue
1805      if( oscil .or. ramp ) return
1806 c*****
1807 c          compute max and min values of jacobian and check for
1808 c          negative values
1809 c*****
1810          ajmax = -1.0e35
1811          ajmin = 1.0e35
1812          do 63 k = 1,kmx(ng)
1813              do 63 i = 1,imx(ng)
1814                  ajmax = max( ajmax,aja(i,k) )
1815                  ajmin = min( ajmin,aja(i,k) )
1816          63 continue
1817          write(6,602) ajmax,ajmin
1818 c..write negative jacobians and stop
1819          if( ajmin.lt.0.0 ) then
1820              do 64 k = 1,kmx(ng)
1821                  do 64 i = 1,imx(ng)
1822                      if( aja(i,k).lt.0.0 ) then
1823                          write(6,603) aja(i,k), i, k
1824                          stop
1825                      end if
1826          64 continue
1827          end if
1828          602 format( ' The range of the jacobian is: ',
1829                  > ' jmax = ',e10.3,5x,'jmin = ',e10.3,/ )
1830          603 format( ' ',10x,'negative jacobian = ',e10.3,1x,'at i,k = ',
1831                  > ' 2i5 )
1832          return

```

```

1833      end
1834 C-----
1835      subroutine eigen
1836      include 'coms.f'
1837      almax=0.0
1838      c..compute the maximum eigenvalue
1839      nt = 1
1840      ng = 1
1841      do 10 i = 2, imx1(nt)
1842      ip = i + 1
1843      im = i - 1
1844      c..evaluate the derivatives of x and z for the line ***
1845      do 20 k = 2, kmx1(nt)
1846      kp = k + 1
1847      km = k - 1
1848      xta = 0.0
1849      zta = 0.0
1850      xps = 0.5*(x(ip,k) - x(im,k))
1851      zps = 0.5*(z(ip,k) - z(im,k))
1852      xze = 0.5*(x(i,kp) - x(i,km))
1853      zze = 0.5*(z(i,kp) - z(i,km))
1854      c..compute the maximum eigenvalue ***
1855      bjac = abs(1.0/( xps*zze - xze*zps ) )
1856      ccc      bjac = aja(i,k)
1857      ccc      psixj = bjac * zze
1858      ccc      psizj = -bjac * xze
1859      ccc      zetxj = -bjac * zps
1860      ccc      zetzj = bjac * xps
1861      psixj = xps
1862      psizj = xze
1863      zetxj = zps
1864      zetzj = zze
1865      obyr = 1.0/q(1,i,k)
1866      rqsq = obyr*(q(2,i,k)**2 + q(3,i,k)**2 )
1867      pval = gmm*(q(4,i,k) - 0.5*rqsq)
1868      cval = sqrt(gamma*pval*obyr)
1869      uvel = obyr*q(2,i,k) - xta
1870      wvel = obyr*q(3,i,k) - zta
1871      ucon = abs(uvel*psixj + wvel*psizj)
1872      bz = cval*sqrt(psixj**2 + psizj**2)
1873      alpsi = bjac*(ucon + bz)
1874      vcon = abs(uvel*zetxj + wvel*zetzj)
1875      bz = cval*sqrt(zetxj**2 + zetzj**2)
1876      alzet = bjac*(vcon + bz)
1877      almaxn = sqrt(alpsi**2 + alzet**2)
1878      almax = max(almaxn, almax)
1879      20      continue
1880      10      continue
1881      dtau=cour/almax
1882      if( timeacc ) then
1883      do k=1,kmx(1)
1884      do i=1,imx(1)
1885      dt(i,k) = dtau
1886      enddo
1887      enddo
1888      else
1889      c..evaluate variable dtau scaling based on Jacobian..
1890      do k=1,kmx(1)
1891      do i=1,imx(1)
1892      sqjac = sqrt( aja(i,k) )
1893      dt(i,k) = ( 1.0 + dtau*sqjac )/( 1.0+sqjac )
1894      enddo
1895      enddo
1896      endif
1897      print *, 'L max = ', almax
1898      write (6,61) dtau
1899      61      format ( ' dtau = ', f12.8)
1900      return
1901      end
1902 C-----
1903      subroutine loads
1904      Comment !!!
1905      ** THIS SUBROUTINE IS INCORRECT !!!
1906      ** Must use non-rotated grid x(i,k), z(i,k)
1907      ** See wrcl.f
1908      include 'coms.f'
1909      dimension cp(nia) , cf(nia), txzs(nia), yplus(nia)
1910      itel = iwks(1)

```

```

1911      iteu = iwke(1)
1912      c..compute pressure loads
1913      cpc = 1. / ( 0.5*rinf*uinf**2)
1914      do 100 i = itel, iteu
1915          p = gmm*( q(4,i,1)
1916          > - .5*(q(2,i,1)**2 + q(3,i,1)**2)/q(1,i,1) )
1917      cp(i) = - (p - pinf) * cpc
1918 100      continue
1919      cn = 0.0
1920      ch = 0.0
1921      cm = 0.0
1922      do 25 i=itel,iteu-1
1923          dx = x(i+1,1) - x(i,1)
1924          dz = z(i+1,1) - z(i,1)
1925          avcp = 0.5*( cp(i+1)+cp(i) )
1926          cn = cn + avcp*dx
1927          ch = ch - avcp*dz
1928      c.. cm about 25% chord
1929          cm = cm - avcp * ( dz*z(i,1) + dx*(x(i,1)-.25) )
1930      25      continue
1931      cl = cn*cos(alfa) - ch*sin(alfa)
1932      cd = cn*sin(alfa) + ch*cos(alfa)
1933      c..compute viscous loads
1934      do 10 i =itel, iteu
1935          u_xi = 0.0
1936          u_ze = q(2,i,2)/q(1,i,2) - q(2,i,1)/q(1,i,1)
1937          w_xi = 0.0
1938          w_ze = q(3,i,2)/q(1,i,2) - q(3,i,1)/q(1,i,1)
1939          xi_x = xix(i,1)*aja(i,1)
1940          ze_x = zex(i,1)*aja(i,1)
1941          xi_z = xiz(i,1)*aja(i,1)
1942          ze_z = zex(i,1)*aja(i,1)
1943          u_x = u_xi * xi_x + u_ze * ze_x
1944          w_x = w_xi * xi_x + w_ze * ze_x
1945          u_z = u_xi * xi_z + u_ze * ze_z
1946          w_z = w_xi * xi_z + w_ze * ze_z
1947          viscl = 0.5*( vismu(i,1) + vismu(i,2) )
1948          visct = 0.5*( turmu(i,1) + turmu(i,2) )
1949          viscto = (viscl + visct) / reynnu
1950          txzs(i) = ( (viscto*(u_z + w_x)) / (0.5 * amach**2) )
1951      c..skin friction
1952          sn = sqrt( (x(i,2)-x(i,3))**2 + (z(i,2)-z(i,3))**2 )
1953          rho = q(i,1,1)
1954          yplus(i) = sqrt( abs(txzs(i)) * rho ) * sn / viscto
1955          dx = (x(i+1,1) - x(i,1))
1956          dz = (z(i+1,1) - z(i,1))
1957          cf(i) = -txzs(i)*(dz/abs(dz)) * 1000
1958      10      continue
1959      cnv = 0.
1960      chv = 0.
1961      cmv = 0.
1962      do 20 i = itel, iteu-1
1963          dx = (x(i+1,1) - x(i,1))
1964          dz = (z(i+1,1) - z(i,1))
1965          avtxzs = 0.5*( txzs(i+1)+txzs(i) )
1966          cnv = cnv + avtxzs*dz
1967          chv = chv + avtxzs*dx
1968          cmv = cmv + avtxzs * (dx * z(i,1) - dz * x(i,1) )
1969      20      continue
1970      clv = cnv*cos(alfa) - chv*sin(alfa)
1971      cdv = cnv*sin(alfa) + chv*cos(alfa)
1972      write(9,101) iter, alfad, time, amach, reynph,
1973      > itel, iteu, cl,cd,cm, clv,cdv,cmv,
1974      > (cp(i), i=itel,iteu),(cf(i), i=itel,iteu)
1975      101 format(i10,4e12.4, 2i5/ 6e12.4 / (6e12.3) )
1976      return
1977      end
1978      C-----
1979      subroutine grmove(dalfa)
1980      include 'coms.f'
1981      if( dalfa .eq. 0.) return
1982      ca = cos( dalfa )
1983      sa =-sin( dalfa )
1984      do 10 i=1,imx(1)
1985      do 10 k=1,kmx(1)
1986          xold = x(i,k)
1987          zold = z(i,k)
1988          x(i,k) = xold * ca - zold * sa

```

```

1989      z(i,k) = zold * ca + xold * sa
1990 10      continue
1991      call metric
1992      return
1993      end
1994 C-----
1995      subroutine qio(io)
1996      include 'coms.f'
1997      IF ( IO .eq. 0) THEN
1998      open(unit=32,file='ends.d',form='unformatted')
1999      write (32) imx(1), kmx(1), ksi
2000      write (32) amach,alfad,reynph,time,iter
2001      write (32) ((( q(1,i,k), i=1,imx(1) ), k=1,kmx(1) ), l=1,4)
2002      close(32)
2003      ELSEIF (IO .eq. 10) THEN
2004      write (8) imx(1), kmx(1), ksi
2005      write (8) amach,alfad,reynph,time,iter
2006      write (8) ((( q(1,i,k), i=1,imx(1) ), k=1,kmx(1) ), l=1,4)
2007      ELSEIF (IO .eq. 1) THEN
2008      open(unit=31,file='strs.d',form='unformatted',status='old')
2009      read (31) imx(1), kmx(1), ksi
2010      read (31) amachr,alfad,reynphr,time,iter
2011      read (31) ((( q(1,i,k), i=1,imx(1) ), k=1,kmx(1) ), l=1,4)
2012      close(31)
2013      kso = kmx(1)
2014      ELSEIF (IO .eq. 2) THEN
2015      open(unit=31,file='strs.d',form='formatted',status='old')
2016      read (31,*) imx(1), kmx(1), ksi
2017      read (31,*) amachr,alfad,reynphr,time,iter
2018      read (31,*) ((( q(1,i,k), i=1,imx(1) ), k=1,kmx(1) ), l=1,4)
2019      close(31)
2020      kso = kmx(1)
2021      ENDIF
2022      return
2023      end
2024
2025
2026
2027
2028
2029
2030

```

APPENDIX D

A. MICHEL'S EMPIRICAL CORRELATION METHOD

A problem was discovered with the subroutine 'output' in BL2D.F that calculates the transition location using the Michel's empirical correlation. The computed transition location was found to be computer and input transition point dependent. This apparent computational error did not affect any other computational results.

The Michel empirical correlation is based on incompressible, constant property flow over a flat plate, and is presented in Equation D.1 with chord (c) assumed to be one (Cebeci and Bradshaw [Ref. 5]).

$$R_{\theta_{tr}} = 1.174 \left[1 + \frac{22,400}{R_{\theta_{tr}}} \right] R_{\theta_{tr}}^{.46} \quad (D.1)$$

$$\begin{aligned} R_{\theta_{tr}} &= \frac{U_{\theta} X_{tr}}{\nu} = \frac{U_{\theta}}{U_{\infty}} X_{tr} R_{\theta} \\ R_{\theta} &= \frac{\rho U_{\infty} c}{\mu} = \frac{U_{\infty} c}{\nu} = \frac{U_{\infty}}{\nu} \end{aligned} \quad (D.2)$$

$$\frac{U_{\theta}}{U_{\infty}} = \text{Normalized Velocity on } i^{th} \text{ Panel}$$

The functional relationship between momentum thickness and transition Reynolds number is presented in Equation D.3. Solving Equations D.1 and D.3 simultaneously yields the results shown in Figure D.1.

$$R_{\theta_{tr}} = 0.664 \sqrt{R_{x_{tr}}} \quad (D.3)$$

An alternate approach for the solution of $R_{\theta_{tr}}$ is shown in Equation D.4 and D.5 with $\rho/\rho_e=1$ for incompressible flow.

$$R_{\theta_{tr}} = \frac{U_e \theta_{tr}}{\nu} = \frac{U_e}{U_\infty} \theta_{tr} R_\theta \quad (D.4)$$

$$\begin{aligned} \theta_{tr} &= \int_0^\delta \left[\frac{\rho u}{\rho_e u_e} \left(1 - \frac{u}{u_e} \right) \right] dy \\ \delta^* &= \int_0^\delta \left(1 - \frac{\rho u}{\rho_e u_e} \right) dy \\ \delta^* &\approx 0.3 \delta \end{aligned} \quad (D.5)$$

The BL2D.F program uses Equations D.1 and D.4 to find the transition location. Each panel on a surface is checked by computing the transition Reynolds number, momentum thickness, and Equations D.1 and D.4. The panel where Equation D.1 is

approximately equal to Equation D.4 is identified as the transition location (surface distance from the input stagnation point). Both the Indigo and Stardent computers compute Equation D.1 exactly the same as can be observed in Figures D.2 and D.3. However, the summing routines used to calculate Equations D.4 and D.5 are computed differently depending on the machine used due to precision differences, thus producing different transition locations with the same input parameters (Figure D.3).

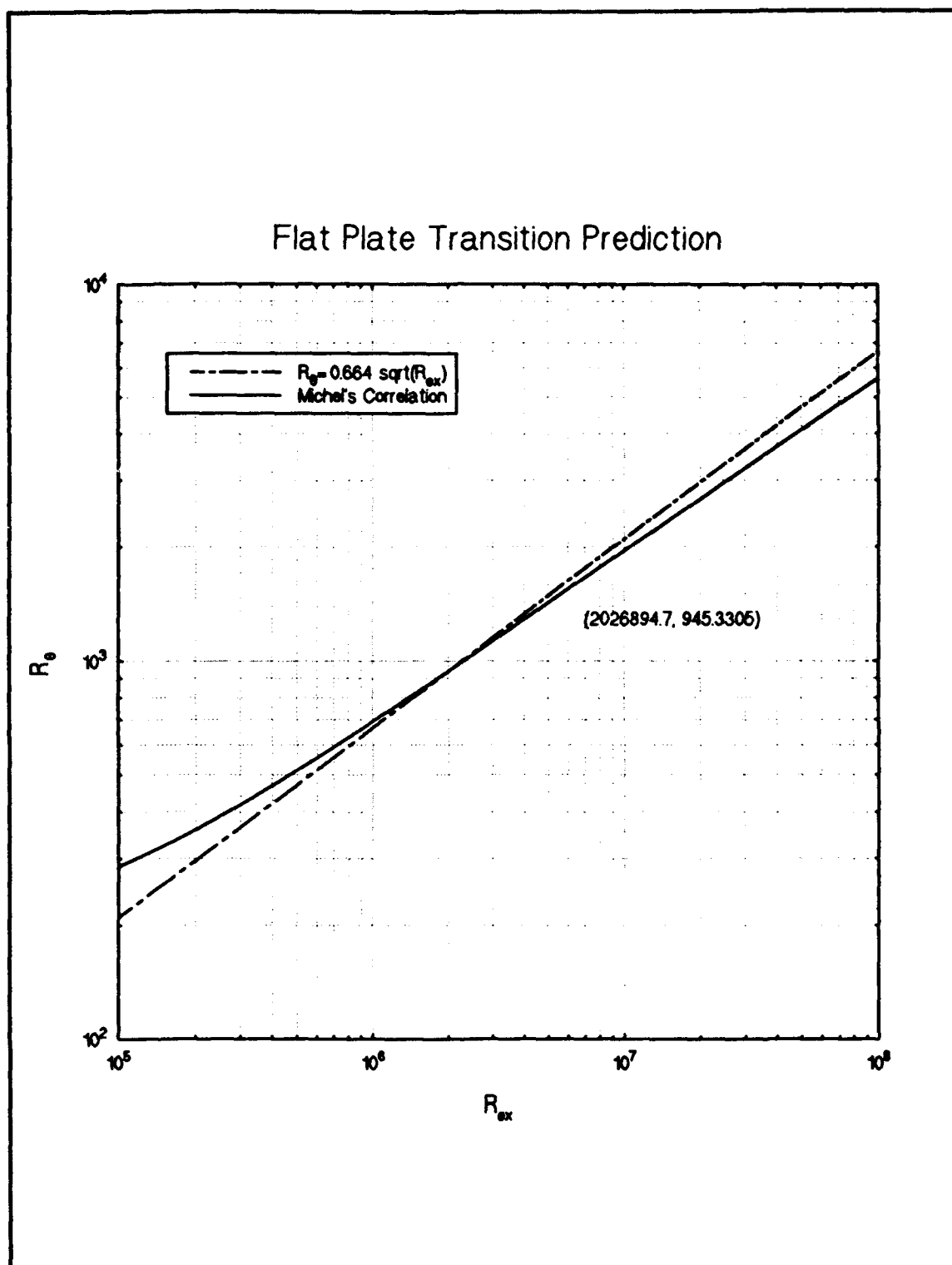


Figure D.1

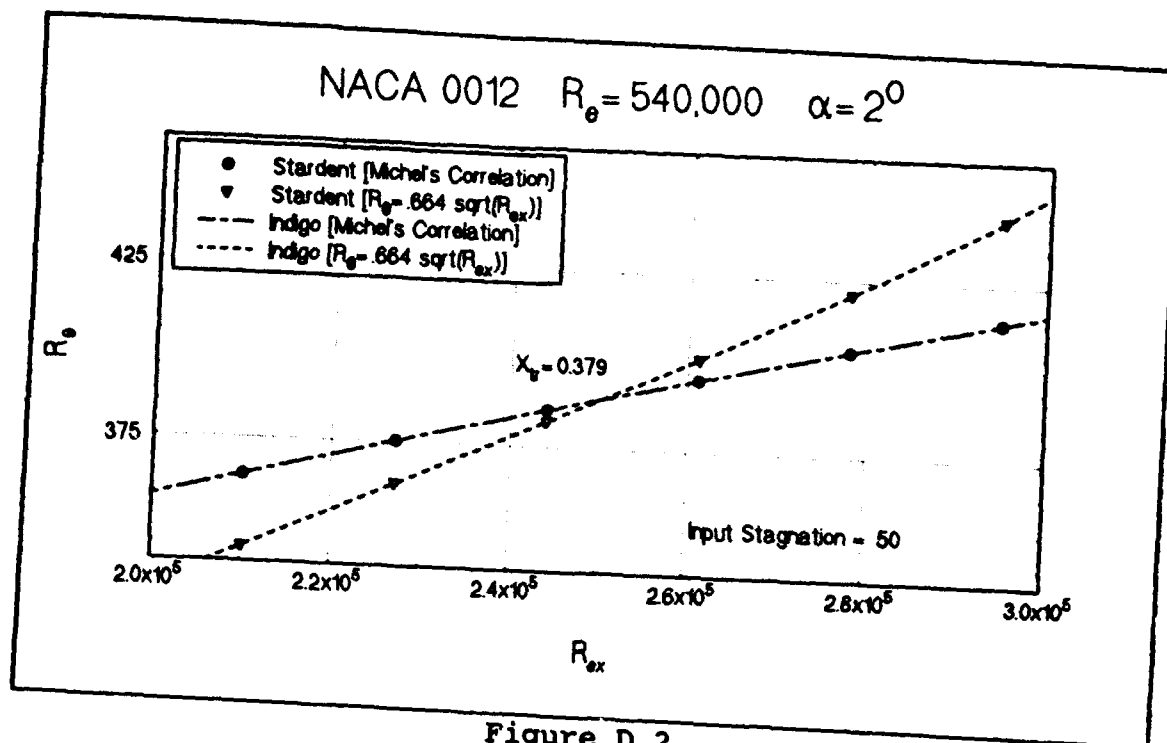


Figure D.2

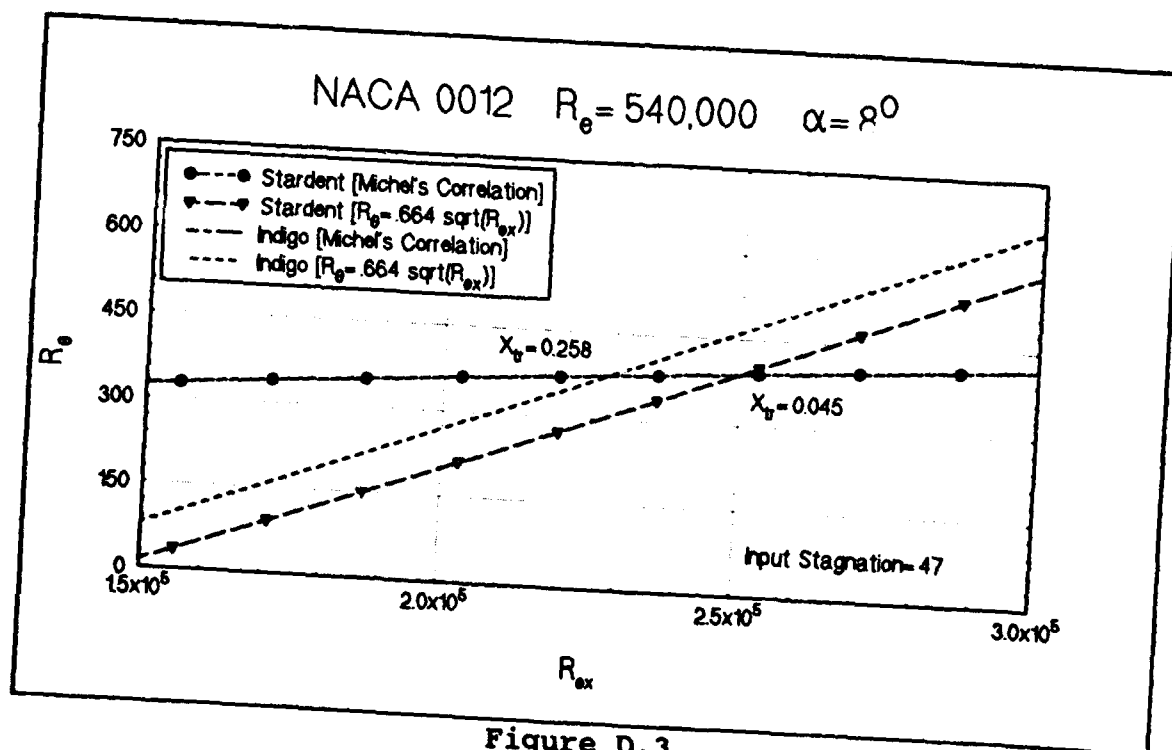


Figure D.3

INITIAL DISTRIBUTION LIST

- | | | |
|----|--|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 052
Naval Postgraduate School
Monterey, California 93943-5002 | 2 |
| 3. | Dr. M. F. Plazer
Dept. of Aeronautics and Astronautics,
Code AA/Pl
Naval Postgraduate School
Monterey, California 93943-5002 | 7 |
| 4. | Dr. J. A. Ekaterinaris
M. S. 260-1
NASA Ames Research Center
Moffett Field, California 94035 | 1 |
| 5. | CDR T. A. Johnston, USN
PMA 273
Room 852 JP1
1421 Jefferson Davis Highway
Arlington, Va 22243-1273 | 3 |