7-1-1995

A Numerical optimization technique for the design of airfoils in viscous flows

Robert MacNeill

"A Numerical Optimization Technique for the Design of Airfoils in Viscous Flows"

by

Robert MacNeill

A Thesis Submitted
in
Partial Fulfillment
of the
Requirements for the

MASTER OF SCIENCE

in

Mechanical Engineering

Approved by:

Professor _____
Dr. P. Venketaraman
Thesis Advisor

Professor _____
Dr. Kevin Kochersberger

Professor _____
Dr. Ali Ogut

Professor _____
Dr. Charles Haines
Department Head

DEPARTMENT OF MECHANICAL ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
JULY 1995

PERMISSION TO REPRODUCE:

Thesis Title: "A Numerical Optimization Technique for the Design of Airfoils in Viscous Flows"

I, Robert MacNeill, hereby grant permission to the Wallace Memorial Library, of the Rochester Institute of Technology to reproduce my thesis in whole or part. Any reproduction can not be used for commercial use or profit.

July 3, 1995

ACKNOWLEDGMENTS:

I would like to express my appreciation and thanks to the following people: Dr. Venkataraman, the Dean's Office, and the Graduate Studies Office for helping me get support for this project, Dr. Ogut for letting me use his lab, Ali for her patience over the past couple years, my dog, Max, for being my best friend, and my mother for being my inspiration to persevere through the past five years, especially the last few months.

This work is dedicated to the memory of my late step-father, Henry Ernst Kertgens, Jr.

ABSTRACT

The present work involves the implementation of an efficient optimization procedure for the design of airfoils in viscous flows. The scope of the work is limited to low Reynolds number, incompressible, and unstalled fluid flow. Cubic Bezier curves with corresponding polygons are employed to define the airfoil, the vertices of which are used as design variables in the optimization process. Inviscid conditions about the airfoil are determined using a traditional Hess-Smith-Douglas panel method. Boundary layer calculations are subsequently made based on the inviscid results and the solution is updated, thereby accounting for viscous effects. A hybrid Generalized Reduced Gradient/Sequential Quadratic Programming method is used in conjunction with the aerodynamic model, to optimize the airfoils. Results were obtained for maximum lift and minimum drag problems with and without constraints. The results of the optimization were validated using CFD.

TABLE OF CONTENTS

LIST OF FIGURES:

NOMENCLATURE

$\mathbf{0}$	the null vector
$\mathbf{A}$	(m+l) x (n-l) matrix of equality constraint gradients with respect to Z
	panel method coefficient matrix
A_{ij}	the $(i, j)^{th}$ component of the coefficients matrix
a_i	quadratic function coefficients; i = 1, 2, 3
$\mathbf{B}$	(m+l) x (m+l) matrix of equality constraint gradients with respect to Y
b_i	i^{th} element of **b** vector
b(y)	binomial distribution as a function of y
c_D	drag coefficient
c_L	lift coefficient
c_M	moment coefficient
$\mathbf{c}$	panel method freestream velocity vector
c_f	skin friction coefficient
c_p	pressure coefficient
E	entrainment velocity
$F(\mathbf{X})$	objective function
f_i	arbitrary i^{th} function
$\mathbf{G_R}$	the generalized reduced gradient
$g_j(\mathbf{X})$	j^{th} inequality constraint
$\mathbf{H}$	Hessian matrix

H	shape factor δ^*/θ
H_1	shape function parameter
$h_k(\mathbf{X})$	k^{th} equality constraint
$\mathbf{I}$	the identity matrix
$J_{3,i}$	3^{rd} order Bernstein basis
m	slope
n	number of design variables
$\hat{\mathbf{n}}$	unit normal vector
L	Lagrangian
l	number of equality constraints,
	parameter for c_f independent of Reynolds number
l_i	length of i^{th} panel
m	number of inequality constraints
N	number of nodes describing the airfoil panels
n	order of binomial distribution
P	(x,y) pair for the coordinate of the Bezier curve
P_i	i^{th} product of $\delta^*_i V_{e,i}$
$\mathbf{Q}$	(m+l) x (n+m) matrix of all equality constraints
Q	volume flow rate
q	source strength per unit length
$R_{e,x}$	Reynolds number with respect to x
$R_{e,\theta}$	Reynolds number with respect to θ

r	polar length coordinate
S	search direction
s	curvilinear distance
$\hat{\mathbf{t}}$	unit tangent vector
t	dummy variable for the distance along a panel
u	x velocity component
V	velocity
v	y velocity component
X	design variable vector
$\mathbf{X}^*$	optimal solution
X_i	i^{th} design variable
x	x-coordinate
x_j	j^{th} variable of function f_i
$\bar{x}_i$	i^{th} panel midpoint x-coordinate
Y	dependent variable vector of size m + 1
$\tilde{\mathbf{Y}}$	derivative matrix of matrix **Y** with respect to α
Y_i	i^{th} dependent design variable
y	index for binomial distribution; 0, 1, 2, ..., n, y-coordinate
$\bar{y}_i$	i^{th} panel midpoint y-coordinate
Z	independent variable vector of size n - *l*

Z_i i^{th} independent design variable

α step size parameter,

angle of attack

α^* optimal step size parameter

δ_j scalar parameter for j^{th} inequality constraint

δ^* displacement thickness

ε gradient perturbation parameter

ϕ flow potential

γ vortex strength per unit length

λ Lagrange multiplier,

dimensionless pressure gradient parameter

λ_j Lagrange multiplier for a corresponding j^{th} inequality constraint

λ_{k+m} Lagrange multiplier for a corresponding $(m+k)^{th}$ equality constraint

μ viscosity

ν parameter for determining airfoil coordinates, P(x,y)

θ angular polar coordinate,

momentum thickness

θ_i inclination of i^{th} panel

τ shear stress

∇ gradient

Subscripts

i	index for design variables; 1, 2, ..., n,
	Bezier vertex index,
	arbitrary function index,
	panel index; 1, 2, ..., N
j	index for inequality constraints; 1, 2, ..., m,
	arbitrary variable index,
	panel index; 1, 2, ..., N
k	index for equality constraints; 1, 2, ..., *l*
n	normal direction
S	indicates source singularity
t	tangential direction
V	indicates vortex singularity
w	wall (y = 0)
Y	indicates gradient with respect to Y
Z	indicates gradient with respect to Z
∞	freestream conditions

Superscripts

-1	inverse of matrix
I	Bezier vertex index
l	lower limit

n	order of binomial distribution
o	0^{th} iteration
q	iteration number
T	vector transpose
u	upper limit
y	binomial distribution index
*	optimum
	local coordinates

1. INTRODUCTION

Since the beginning of powered flight there has been intense interest in the area of airfoil design. In the early stages of aerodynamics, mostly ad hoc methods were employed to determine suitable wing sections. Over time, however, the design process has become more sophisticated.

In the past, direct analysis methods were widely used where a designer would define a geometry and then analyze the flow about the airfoil. Generally, panel methods were employed for these analyses. The original design would have to be modified based on the output and the process would be iterated until satisfactory aerodynamic performance was achieved. As one would expect, this can be a time consuming process that does not lend itself well to systematic updating.

Analyzing the flow conditions for a specified airfoil shape is the most direct approach but is by no means the most efficient method for optimization. For this reason, there has been considerable study into the inverse design problem. The inverse design problem may be defined as a technique that yields an airfoil shape as output while the pressure/velocity around the airfoil is specified. The inverse problem can be a very useful approach to airfoil design.

Conformal mapping techniques are mostly used in typical applications of inverse design problems. The famous Eppler method (Reference 1) employs a conformal mapping technique which requires a user specified velocity distribution. Based on this, an airfoil geometry is calculated. Incidentally, the Eppler code may also be used for direct design

since it also uses a panel method with an integral boundary layer solver. These solution schemes can be much quicker than direct methods because no guessing has to be done to achieve desired conditions. However, being restricted to a specified distribution can limit a solution from being potentially optimum. It would be ideal to use a method with optimization capabilities which could search the design space, updating based on previous solutions, to find optimum shapes. For this reason, the present work proposes an optimization method involving an objective function of maximum lift coefficient (c_L) or minimum drag coefficient (c_D) which takes into account pressure and velocity distributions. Unlike traditional inverse design problems, the present method is not restricted by initial flow conditions and lends itself to finding unconventional shapes.

Design optimization in general is extremely attractive and has been receiving a lot of attention lately. With the advancements in computers and optimization algorithms, once almost unsolvable problems are now very modest in run times. By virtue of their iterative nature, optimization procedures usually take longer than straightforward design techniques, but the end result is almost always superior. In response to the recent interest for design optimization, there are currently many easy to use software packages that make the optimization process much less burdensome. Some finite element packages even have optimization modules and it is foreseeable that CFD will soon incorporate similar ones. Until then, however, it is still desirable to have a method for the optimization of airfoils.

Over the past years, many methods for optimizing aerodynamic shapes have been developed. For example, Vanderplaats, Hicks, and Murman (Reference 2) advanced a method that optimizes airfoil shapes based on conjugate directions for locally

unconstrained problems and feasible directions for locally constrained problems. The airfoil geometry is defined by a polynomial approximation for different segments of the shape. For this geometry description method, the design variables become the segment endpoint coordinates and the polynomial coefficients. Problems for minimum drag, minimum pitching moment, and maximum lift were addressed with some success. Most of the solutions were, however, limited by local optima.

Another method developed by Vanderplaats and Hicks (Reference 3) allows the designer's intuition to play a role in the optimization. This is atypical of most optimization techniques. The method starts with many different shapes, each with desirable characteristics to the user. The approach is to define the airfoil as a linear combination of the shapes. The optimization problem becomes one of determining the influence of each shapes on the optimal solution. This method seemed more successful than the first but a major disadvantage is that solutions are limited by the initially defined shapes.

Liebeck and Ormsbee proposed yet another method that was very successful (Reference 4). For the technique, boundary layer theory and the calculus of variations were employed to determine the pressure distribution which provides the maximum lift without separation. This solution scheme produced the very well known Liebeck high lift profiles which were reported to have lift coefficients as high as 2.8 for Reynolds numbers between five and ten million. Drag coefficients for the corresponding airfoils were kept as low as 0.01. A single element airfoil definition was used to define the shape. The analysis is split into two subproblems which consists of first determining the pressure distribution for maximum lift without separation, and then determining an airfoil shape to determine

the specified pressure distribution (inverse problem). The first step employs a Stratford distribution to model separation. Besides the no-separation requirement, constraints are placed on the leading edge and trailing edge to assure design feasibility. Also, the satisfaction of the Kutta condition is required. The second step is accomplished by utilizing conformal mapping methods.

Low drag airfoils and fuselages are of special interest in the field of gliders. For this reason, Coiro and Nicolosi (Reference 5) developed a technique to determine low drag airfoils, high lift and low drag multi-component airfoils, and extended laminar region fuselages. A constrained-minimization method is used as the design optimizer. An integral boundary layer method is also used for the aerodynamic analysis.

Chang, et. al. (Reference 6) proposed a method for the optimization of wing-body configurations through the solution of the Euler equations for the flow surrounding the airfoil. A feasible direction method for constrained optimization is used as the optimizer. Unlike the other methods described, the geometry is controlled and analyzed through the use of a flow field grid generator and a set of shape functions for the definition of the shapes.

Although these optimization methods vary widely in their scope and solution strategy, they share some common characteristics. In the optimization process, there are three distinct components: (1) the design variables, (2) the optimizer, and (3) the mathematical model. Each component should be efficient individually to effectively streamline the optimization procedure.

To determine appropriate design variables for the problem, an efficient way to define an airfoil which would lend itself to design perturbation is necessary. There are many ways to describe airfoil geometries but the present work uses a Bezier polygon definition scheme which is outlined in Reference 7. This method provides an extremely efficient description technique that handles perturbation very nicely.

Again, it is very important that the optimizer be very efficient to solve the problem. Ideally, the optimization technique should determine search directions solely from gradient information. There are many efficient methods available. The class of optimization techniques referred to as direct methods are by far superior to others in dealing with problems with many design variables and constraints. The optimization procedure used for the present work is a hybrid method which combines the best characteristics of the Generalized Reduced Gradient method and Sequential Quadratic Programming. This algorithm is implemented through a software package called OptdesX (Reference 8).

Finally, an efficient method for calculating the conditions about the airfoil should be employed to solve the problem. An inviscid flow panel method is used with an integral method for the boundary layer effects because of the economy of its solution. This method may be somewhat primitive and, hence, less accurate compared to other more time consuming flow analyzing techniques but it is very important that the mathematical model be as efficient as possible in an already time consuming iterative process. Also, it could be advantageous to sacrifice some accuracy in the solution because if the objective is satisfactorily improved while the constraints are satisfied, then the optimization process may be considered successful, regardless of the precision of the results. Furthermore,

sophisticated CFD codes can be used to more accurately determine the flow conditions about the airfoil that is determined to be optimal. The present work first reviews optimization theory followed by an in depth discussion of the Generalized Reduced Gradient method. Then an airfoil design section is dedicated to the discussion of the three main components of optimization: the design variables, the optimizer, and the mathematical model. Program development is then discussed followed by CFD modeling of the solutions. For the current work, the NACA 0012 and the NACA 4412 airfoils were both used as starting points for the optimization. Also, two different Bezier definition schemes were used to describe the airfoils. Finally, six optimization cases were considered and are listed as follows:

A. Max c_L, unconstrained c_D and c_M

B. Max c_L, $c_D \leq 0.007$, unconstrained c_M

C. Max c_L, $C_D \leq 0.007$, $c_M \leq -0.15$

D. Min c_D, unconstrained c_L and c_M

E. Min c_D, $c_L \geq 0.7$, unconstrained c_M

F. Min c_D, $c_L \geq 0.7$, $c_M \leq -0.15$

Considering the two starting points, the two definition schemes, and the six cases, a total of twenty four optimization problems were considered. The presentation of the results is followed by a discussion and conclusions.

2. DESIGN OPTIMIZATION

2.1 Problem Statement

Design optimization is a systematic approach meant to improve upon existing designs. Many optimization algorithms have been developed but almost all of them share some common features. For example, most methods are meant to optimize a design by improving upon an objective function while satisfying some design constraints, where the objective and constraints are functions of the design variables relevant to the problem. The general statement of an optimization problem is as follows:

$$\text{Minimize:} \quad F(\mathbf{X}) \qquad \mathbf{X} = [X_1, X_2, \ldots, X_n] \tag{2.1a}$$

$$\text{Subject to:} \quad g_j(\mathbf{X}) \le 0 \qquad j = 1, m \tag{2.1b}$$

$$h_k(\mathbf{X}) = 0 \qquad k = 1, l \tag{2.1c}$$

$$X_i^l \le X_i \le X_i^u \qquad i = 1, n \tag{2.1d}$$

Where $F(\mathbf{X})$ is the objective function of the design variables, $\mathbf{X}$, $g_j(\mathbf{X})$ are the inequality constraints, and $h_k(\mathbf{X})$ are the equality constraints. Side constraints used to limit the design to the feasible, or realistic, domain are shown in Eq. (2.1d). The functions $F(\mathbf{X})$, $g_j(\mathbf{X})$, and $h_k(\mathbf{X})$ may be linear or nonlinear in $\mathbf{X}$, and may be evaluated by any analytical or numerical technique. Also, for most optimization techniques, the objective function and constraints have to be continuous and have continuous first derivatives (for gradient calculations).

2.2 Optimization Procedure

A fundamental similarity between different optimization techniques is the basic update and search method. Most methods require an initial set of design variables, $\mathbf{X}^{\circ}$. It is advantageous if the initial guess was in reasonable proximity to a realistic design. This is not always necessary for convergence, but it could expedite the design process.

From the start point the design is updated iteratively until an optimum is found. The common form of the update is

$$\mathbf{X}^{q} = \mathbf{X}^{q-1} + \alpha^{*}\mathbf{S}^{q} \tag{2.2}$$

where q is the iteration number, **S** is the vector search direction, and α^{*} is the optimal step size in direction **S**. As can be seen in Eq. (2.2), previous iterations are updated with the term $\alpha^{*}\mathbf{S}^{q}$. This update is a one-dimensional search which converts a multiple variable problem (n) into a one variable problem in α. The two main stages of the update consists of finding a suitable search direction, **S**, and determining the scalar α^{*}. The search direction would be found such that it improves the objective while maintaining the constraints. The scalar α^{*} is found to be the optimal step size in direction **S**. For a method to be successful, the update should be determined efficiently and reliably.

2.3 Existence and Uniqueness of an Optimal Solution

Because of the inherent nonlinearity of most optimization problems, it is difficult to ensure that an optimal solution is global. For nonlinear problems, there is a strong tendency to settle into local optima (also called relative optima). This means that there will assuredly be multiple solutions and it will be necessary to attack the problem from different starting points to try to find a global optimum. If the same solution is converged upon from different points, it is reasonable to say that the solution is a global optimum. Of course, this leads to an increase in design time to an already time consuming iterative process.

2.3.1 Unconstrained Problems

For unconstrained problems, there is a set of well defined criteria for determining whether a solution is a relative optimum. Also, though it is difficult to do, these conditions may be used to determine whether a solution is a global optimum. The first condition is that the gradient with respect to all variables (i = 1, n) be the null vector:

$$\nabla F(\mathbf{X}) = \begin{Bmatrix} \partial F(\mathbf{X})/\partial X_1 \\ \partial F(\mathbf{X})/\partial X_2 \\ \vdots \\ \partial F(\mathbf{X})/\partial X_n \end{Bmatrix} = \mathbf{0} \tag{2.3}$$

Furthermore, for a minimum, the second derivative matrix (Hessian) must be positive definite. Likewise, for a maximum, the Hessian must be negative definite. These are necessary, but not sufficient, conditions to prove that the solution is a relative minimum. To ensure that a solution is definitely a relative minimum, the Hessian of the objective

function must be positive definite for the solution $\mathbf{X}^*$. A Hessian matrix of a function is defined as the matrix of the second partial derivatives with respect to all variables. The Hessian of the objective function is

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 F(\mathbf{X})}{\partial X_1^2} & \frac{\partial^2 F(\mathbf{X})}{\partial X_1 \partial X_2} & \cdots & \frac{\partial^2 F(\mathbf{X})}{\partial X_1 \partial X_n} \\ \frac{\partial^2 F(\mathbf{X})}{\partial X_2 \partial X_1} & \frac{\partial^2 F(\mathbf{X})}{\partial X_2^2} & \cdots & \frac{\partial^2 F(\mathbf{X})}{\partial X_2 \partial X_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F(\mathbf{X})}{\partial X_n \partial X_1} & \frac{\partial^2 F(\mathbf{X})}{\partial X_n \partial X_2} & \cdots & \frac{\partial^2 F(\mathbf{X})}{\partial X_n^2} \end{bmatrix} \tag{2.4}$$

For a matrix to be positive definite, all its eigenvalues must be positive in sign. If both of the previously defined conditions are satisfied, then the solution $\mathbf{X}$ is at least a relative minimum. The only way to ensure that a design is a global minimum is if the condition defined by Eq. (2.3) is satisfied and the Hessian is found to be positive definite for all possible values of $\mathbf{X}$. Obviously, this can be difficult, if not unrealistic, especially for a design with many design variables. The best way to achieve a level of confidence that an optimum is global is to solve the same problem from many different starting points.

2.3.2 Constrained Problems and the Kuhn-Tucker Conditions

Unfortunately, determining whether a solution is a global optimum or even a relative optimum becomes more difficult with the addition of constraints to the optimization problem. Unlike the unconstrained problem, the gradient of the objective does not have to equal the null vector at the optimum. This can be the case if at least one constraint is active. An active constraint may be defined as one whose value is close to zero (from, $\mathbf{g}(\mathbf{X}) \leq 0$).

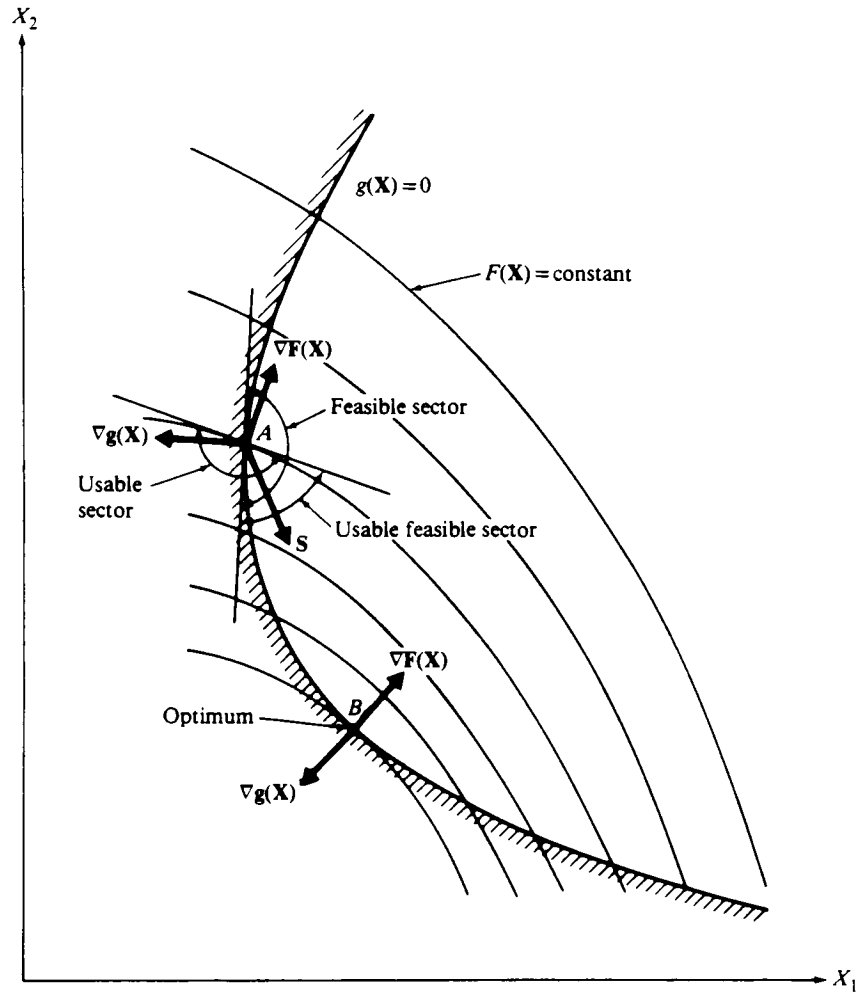


Figure 2.1:
Description of Usable-Feasible Sector

Figure 2.1 (from Reference 9) may be used to intuitively establish the necessary conditions for a constrained problem. For the optimum, point B, both the gradients of the constraints and the objective are perfectly opposite in direction. This means that, at the optimum, the search direction would have to be perpendicular to both $\nabla\mathbf{F}(\mathbf{X})$ and $\nabla\mathbf{g}(\mathbf{X})$. This is an important conclusion in that it establishes a necessary condition for a solution to be optimal, but, unfortunately, this is not a sufficient condition. This and two other

conditions make up what is known as the Kuhn-Tucker necessary conditions for constrained optimality. If $\mathbf{X}$ is an optimal solution, it must satisfy the following:

1. $\mathbf{X}^*$ is feasible (2.5)

2. $\lambda_j g_j(\mathbf{X}) = 0 \qquad j = 1, m, \quad \lambda_j \geq 0$ (2.6)

3. $$\nabla \mathbf{F}(\mathbf{X}^*) + \sum_{j=1}^{m} \lambda_j \nabla \mathbf{g}_j(\mathbf{X}^*) + \sum_{k=1}^{l} \lambda_{k+m} \nabla \mathbf{h}_k(\mathbf{X}^*) = \mathbf{0} \tag{2.7}$$

$\lambda_j \geq 0$ (2.8)

λ_{m+k} unrestricted in sign (2.9)

The first condition is obvious in that a design must be feasible (satisfy all constraints) to be optimal. Equation (2.6) states that if $g_j(\mathbf{X}) < 0$, and, hence, not active, then the Lagrange multiplier λ_j must be zero. The last set of equations is the mathematical statement that, at the optimum, the only possible direction vector $\mathbf{S}$ would be tangent to both the active constraint boundary and the line of constant objective function and would be perpendicular to both gradients. Also, Eq. (2.7) is called the Lagrangian $L(x,\lambda)$. At off optimal designs, the Lagrangian is defined as

$$L(\mathbf{X},\lambda) = F(\mathbf{X}) + \sum_{j=1}^{m} \lambda_j g_j(\mathbf{X}) + \sum_{k=1}^{l} \lambda_{m+k} h_k(\mathbf{X}) \tag{2.10}$$

Where the Lagrange multipliers λj and λ_{m+k} act as scalar multipliers to $g_j(\mathbf{X})$ and $h_k(\mathbf{X})$.

While necessary and sufficient for the determination of local optima, satisfaction of the Kuhn-Tucker conditions is not necessarily sufficient to ensure a global optimum. A solution $\mathbf{X}^*$ can, however, be said to be a global optimum if it satisfies the Kuhn-Tucker conditions and the surfaces of the problem's objective and constraints are convex. A convex surface can be defined as one where, if any point on the surface was connected to

the optimum, then the entire line connecting them would lie within the feasible region and the surface defined by $g(\mathbf{X}) = 0$ is convex. This implies that if the Kuhn-Tucker conditions were satisfied, then the solution would be a global optimum, as is the case with point B in Fig. 2.1. An example of a non-convex surface may be seen by referring to Fig. 2.2 (from Reference 9). While the line connecting points B and D lies within the feasible region, the lines connecting points A and C, and points A and D would be outside of the constraint bounds for some portion of the line. Therefore, the surface defined by $g(\mathbf{X}) = 0$ in Fig. 2.2 is non-convex and the Kuhn-Tucker conditions could not be used to determine a global optimum.

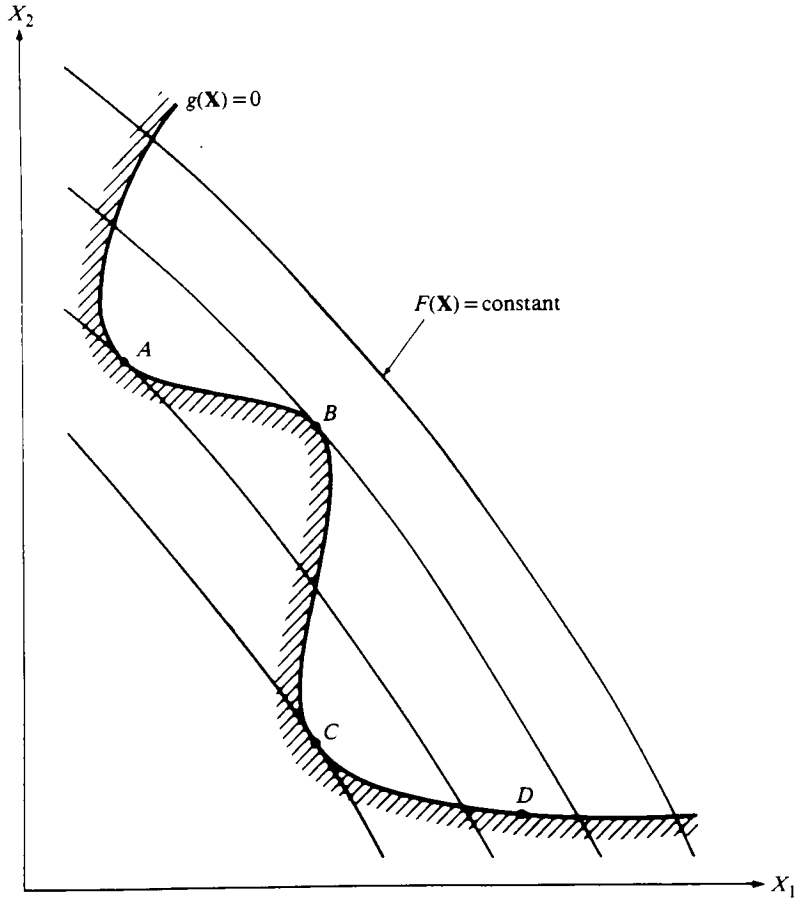


Figure 2.2:
Description of a Convex Curve

As previously mentioned, it is very difficult to determine whether the sufficiency requirements have been met for a particular problem. Therefore, it is again suggested that the problem be started from multiple points to ensure a global optimum.

2.4 Direct Optimization Methods

For the present work, a hybrid Generalized Reduced Gradient (GRG)/Sequential Quadratic Programming (SQP) method was used as the optimizing procedure. The algorithm was made available through the software package OptdesX. Since the procedure was not programmed by the author, its description is not included (see Reference 10). However, as an example of a typical optimization method, the GRG method and its implementation is described in detail.

This method belongs to the family of techniques known as direct methods. They are named as such since they all deal with constraints in a direct manner. This is opposed to former, less sophisticated methods that rely on constraint penalty functions to convert constrained into unconstrained optimization problems. The direct methods are typically very efficient and are in almost exclusive use with respect to optimization problems with multiple variables and constraints.

The Generalized Reduced Gradient method determines optimal solutions by relying heavily on active constraints. An active constraint is one which is at its bounds, hence, potentially limiting the solution. For the method, a search direction is found to keep any active constraints exactly active for small movements in that direction. If, for any reason such as nonlinearity, the currently active constraints become inactive upon movement in the search direction, then the constraint would be forced to its bounds by using Newton's method.

The present method solves equality constrained problems only by adding a slack variable to each inequality constraint. Subsequently, the general form of a GRG problem is

Minimize: $F(\mathbf{X})$ (2.11)

Subject to: $g_j(\mathbf{X}) + X_{j+n} = 0 \qquad j = 1, m$ (2.12)

$h_k(\mathbf{X}) = 0 \qquad k = 1, l$ (2.13)

$X_i^l \le X_i \le X_i^u \qquad i = 1, n$ (2.14)

$X_{j+n} \ge 0 \qquad j = 1, m$ (2.15)

Due to the requirement of slack variables in Eq. (2.12), the number of design variables increases to n + m. The extra design variables could make the problem difficult to solve because of extra storage requirements, however, this problem could be alleviated through careful storage of gradient information.

The basic idea behind the GRG method is that the total number of design variables may be reduced by defining one dependent design variable for each equality constraint.

Considering the problem stated in Eqs. (2.11-2.15), the **X** vector contains the original n design variables as well as the m slack variables. For clarity and convenience, **X** can be split into its components and written as follows:

$$\mathbf{X} = \{ \mathbf{Z}\ \mathbf{Y} \}^T \tag{2.16}$$

Where **Z** represents n - l independent variables, and **Y** represents m + l dependent variables. For this definition, no restrictions are assigned regarding which variables are to be contained in **Z** and **Y**. As will be discussed later, the dependent variables will be chosen from the original n variables as well as the m slack variables such that the problem is balanced. Because all constraints are now equal, it is convenient to combine Eqs. (2.12) and (2.13) into a single constraint definition:

$h_j(\mathbf{X}) = 0 \qquad j = 1, m + l$ (2.17)

and combine Eqs. (2.14) and (2.15), yielding

$$X_i^l \le X_i \le X_i^u \qquad i = 1, n + m \tag{2.18}$$

where the upper bounds on the slack variables are allowed to be very large.

Now the optimization problem has been reduced to

$$\text{Minimize:} \quad F(\mathbf{X}) \equiv F(\mathbf{Z}, \mathbf{Y}) \tag{2.19}$$

$$\text{Subject to:} \quad h_j(\mathbf{X}) = 0 \qquad j = 1, m + l \tag{2.20}$$

$$X_i^l \le X_i \le X_i^u \qquad i = 1, n + m \tag{2.21}$$

Differentiating the objective and the constraint function yields

$$dF(\mathbf{X}) = \nabla_Z \mathbf{F}(\mathbf{X}) \bullet \mathbf{dZ} + \nabla_Y \mathbf{F}(\mathbf{X}) \bullet \mathbf{dY} \tag{2.22}$$

$$dh_j(\mathbf{X}) = \nabla_Z \mathbf{h}_j(\mathbf{X}) \bullet \mathbf{dZ} + \nabla_Y \mathbf{h}_j(\mathbf{X}) \bullet \mathbf{dY} \tag{3.23}$$

$$\text{where } j = 1, m + l$$

The subscripts Z and Y indicate gradients with respect to the independent and dependent variables, respectfully.

For feasibility, the equality constraint must remain satisfied (assuming they are satisfied initially) for any change in the independent variables. This means that $dh_j(\mathbf{X}) = 0$, $j = 1, m + l$ in Eq. (2.23). Since **Y** contains $m + l$ dependent variables and since Eq. (2.23) is actually a system of $m + l$ equations, they can be written as

$$\mathbf{dh}(\mathbf{X}) = \begin{bmatrix} \nabla_Z^T \mathbf{h}_1(\mathbf{X}) \\ \nabla_Z^T \mathbf{h}_2(\mathbf{X}) \\ \vdots \\ \nabla_Z^T \mathbf{h}_{m+l}(\mathbf{X}) \end{bmatrix} \mathbf{dZ} + \begin{bmatrix} \nabla_Y^T \mathbf{h}_1(\mathbf{X}) \\ \nabla_Y^T \mathbf{h}_2(\mathbf{X}) \\ \vdots \\ \nabla_Y^T \mathbf{h}_{m+l}(\mathbf{X}) \end{bmatrix} \mathbf{dY} \tag{2.24}$$

or

$$\mathbf{dh}(\mathbf{X}) = \mathbf{A}\,\mathbf{dZ} + \mathbf{B}\,\mathbf{dY} \tag{2.25}$$

where the dimensions of **A** and **B** are $(m + l) \times (n - l)$ and $(m + l) \times (m + l)$, respectively. Since it was established that $\mathbf{dh(X)} = \mathbf{0}$, for any change of the independent variables **dZ**, Eq. (2.25) can be solved for the corresponding changes **dY** in the dependent variables to maintain feasibility:

$$\mathbf{dY} = -\mathbf{B}^{-1}\mathbf{A}\,\mathbf{dZ} \tag{2.26}$$

Now substituting Eq. (2.26) into Eq. (2.22) yields

$$\begin{aligned} dF &= \nabla_Z \mathbf{F(X)} \bullet \mathbf{dZ} - \nabla_Y \mathbf{F(X)}^T [\mathbf{B}^{-1}\mathbf{A}]\mathbf{dZ} \\ &= \{\nabla_Z{}^T \mathbf{F(X)} - \nabla_Y \mathbf{F(X)}^T [\mathbf{B}^{-1}\mathbf{A}]\} \bullet \mathbf{dZ} \end{aligned} \tag{2.27}$$

So

$$\mathbf{G_R} = dF(\mathbf{X})/dZ = \nabla_Z \mathbf{F(X)} - [\mathbf{B}^{-1}\mathbf{A}]^T \nabla_Y \mathbf{F(X)} \tag{2.28}$$

Equation (2.28) defines the *generalized reduced gradient* $\mathbf{G_R}$, and can be viewed as an unconstrained function. A search direction **S** can be found using $\mathbf{G_R}$ for use in the following equation:

$$\mathbf{X}^q = \mathbf{X}^{q-1} + \alpha^* \mathbf{S}^q \tag{2.29}$$

Also, the dependent variables **Y** are updated using Eq. (2.26) for every proposed step α. It must be remembered, however, that Eq. (2.26) is a linear approximation to a nonlinear problem. This means that when the constraints are evaluated for an α, they may not be exactly zero. In other words, the vector **dh(X)** from Eq. (2.25) may not be the null vector. In order to correct this, holding **Z** fixed, a new **dY** must be found to drive **h(X)** to zero. So

$$h_j(\mathbf{X}) + dh_j(\mathbf{X}) = 0 \qquad j = 1, m + l \tag{2.30}$$

and **dY** must be found so that

$$dh_j(\mathbf{X}) = -h_j(\mathbf{X}) \qquad j = 1, m + l \tag{2.31}$$

Substituting Eq. (2.31) into Eq. (2.25) yields a new estimate for **dY**:

$$\mathbf{dY} = \mathbf{B}^{-1} \{-\mathbf{h}(\mathbf{X}) - \mathbf{A}\, \mathbf{dZ}\} \tag{2.32}$$

Next, **dY** is added to the most recent vector **Y** of dependent variables and the constraints are evaluated again. This is repeated until $h_j(\mathbf{X}) = 0$, $j = 1, m + l$, within a specified tolerance.

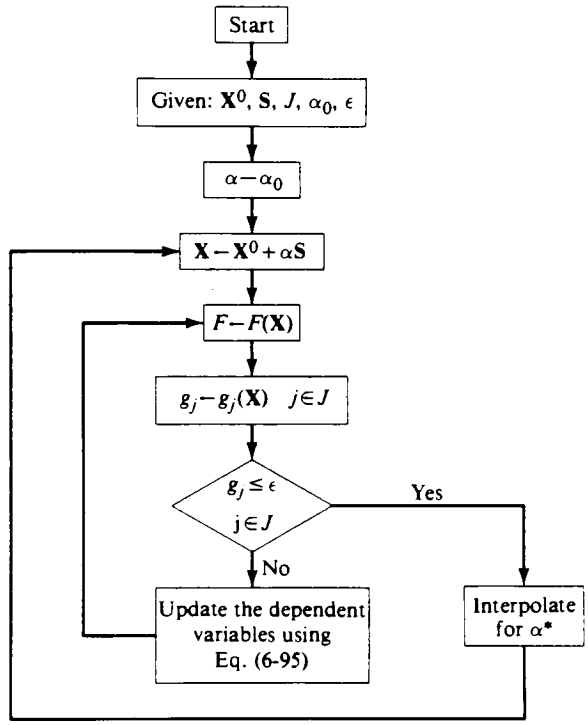


Figure 2.3:
Update Algorithm for the GRG Method

In summary, a reduced gradient is created from previously chosen dependent variables. The reduced gradient is then used to determine a search direction in the independent variables. While marching in this direction for each proposed α, the

dependent variable vector **Y** is updated as previously discussed. This updating method is quite similar to Newton's method for solving simultaneous linear equations for **dY**. However, the present method assumes the gradient information in **A** and **B** to be constant. After finding the minimum in the search direction, and, hence, α^*, the process is repeated until the convergence criteria is satisfied. Figure 2.3 (from Reference 9) shows the algorithm for this updating scheme.

The last requirement is to choose the dependent variables, **Y**, (1) such that the **B** matrix is not singular and (2) so that the solution can move some distance in a search direction without violating the side constraints. The second requirement is satisfied by picking dependent variables which are a sufficient distance from the limits of their respective side constraints. The first requirement, however, is a little less obvious; it is met by starting with the matrix of gradients:

$$\mathbf{Q} = \begin{bmatrix} \nabla^T \mathbf{h}_1(\mathbf{X}) \\ \nabla^T \mathbf{h}_2(\mathbf{X}) \\ \vdots \\ \nabla^T \mathbf{h}_{m+l}(\mathbf{X}) \end{bmatrix}_{(m+l)\times(n+m)} \tag{2.33}$$

To this point, the independent and dependent variables have not been divided. Since there are more columns than rows in matrix **Q**, it is desirable to find a nonsingular submatrix that would correspond to **B**, hence, defining the dependent variables. This can be done by employing Gaussian elimination with pivoting on matrix **Q**. To find the dependent variables, start with the following matrix equation:

$$\mathbf{QX} = \mathbf{I} \tag{2.34}$$

Where **I** is a (m + *l*) x (m + *l*) identity matrix. The first step is to search row one of **Q** for the element of largest magnitude, omitting those that are at the side constraint limits. Next, the matrix is then pivoted on that element and regular Gaussian elimination operations are performed. This process is repeated for the remaining rows. After this procedure is complete, the right hand side of the equation contains $\mathbf{B}^{-1}$. The remaining columns of **Q** will contain the product $\mathbf{B}^{-1}\mathbf{A}$ from Eq. (2.28).

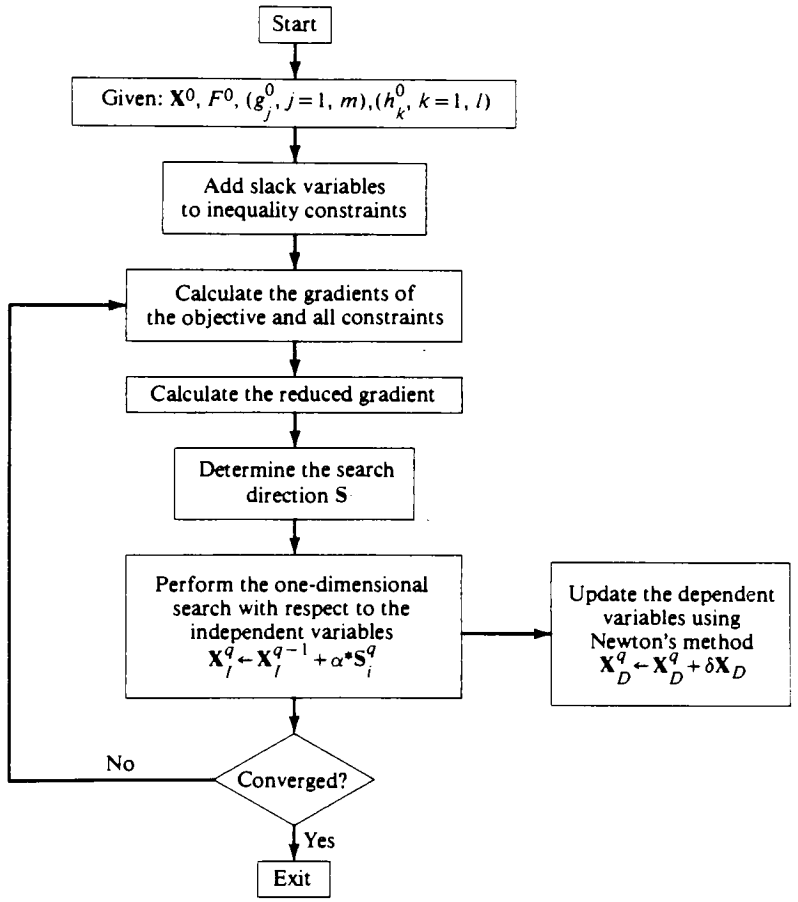


Figure 2.4:
Overall Algorithm for the GRG Method

The method just described for determining the dependent variables will satisfy the requirements of a nonsingular **B** matrix corresponding to variables that are not at their constraints. However, several potentially hazardous situations are not accounted for and need to be addressed separately. For instance, the method does not guarantee that a side constraint will not be violated during a one-dimensional search. If this was to occur, a smaller step size α^* would need to be chosen such that the constraint was not violated. Another situation that may arise is when a row or a column of **Q** contains all zeros. This could either mean that a redundant constraint exists which may be temporarily deleted from the set (temporarily because the constraint may become independent later in the optimization process) and the number of dependent variables reduced by one, or it could mean that no constraint is a function of the design variable. The latter is a degenerate case where no dependent variable should be used for the design variable in question. A final situation may occur where the only nonzero pivot element is at one of its side constraint limits. In this case, there is no choice but to include this variable in the dependent variable set. After calculating **dY** it may be found that the variable moves away from its limit. However, if the variable moves in a manner such that it violates its constraint, that variable is set to its limiting value. This could prevent Newton's method from converging which would indicate that the step size α should be reduced. Figure 2.4 shows the overall algorithm for the GRG method.

The simplest way of finding the search direction **S** is to take the negative of the generalized reduced gradient:

$$\mathbf{S} = -\mathbf{G}_R \tag{2.35}$$

For subsequent iterations, other more sophisticated methods such as a conjugate direction or a variable metric method may be used. See Reference (9) for a more in depth discussion of those methods. Also, a first estimate for α may be found as the distance to the nearest side constraint. For independent variables:

$$dZ_i/d\alpha = S_i \tag{2.36}$$

and for the dependent variables (using Eq. (2.32)):

$$\frac{\mathbf{dY}}{\mathbf{d}\alpha} = \tilde{\mathbf{Y}} = -\mathbf{B}^{-1}\mathbf{AS} \tag{2.37}$$

To find the α which will drive the i^{th} independent variable to its bound:

$$Z_i + \alpha \frac{dZ_i}{d\alpha} = Z_i + \alpha S_i = Z_i^l \text{ or } Z_i^u \tag{2.38}$$

yielding

$$\alpha = \frac{Z_i^l - Z_i}{S_i} \qquad \text{if } S_i < 0 \tag{2.39a}$$

$$\alpha = \frac{Z_i^u - Z_i}{S_i} \qquad \text{if } S_i > 0 \tag{2.39b}$$

Based on the above linear approximation, the α which drives an independent variable to either its upper or lower bound is the minimum α for all independent variables Z_i, i = 1, n - *l*. Similarly, for the dependent variables (using Eq. (2.37)):

$$Y_i + \alpha \tilde{Y}_i = Y_i^l \text{ or } Y_i^u \tag{2.40}$$

yielding

$$\alpha = \frac{Y_i^l - Y_i}{\tilde{Y}_i} \qquad \text{if } \tilde{Y}_i < 0 \tag{2.41a}$$

$$\alpha = \frac{Y_i^u - Y_i}{\tilde{Y}_i} \qquad \text{if} \quad \tilde{Y}_i > 0 \qquad (2.41b)$$

Again, the α which drives a dependent variable to its bound is the minimum α for all dependent variables Y_i, i = 1, m + *l*. The minimum of the values of α determined by Eqs. (2.39) and (2.41) is then taken as the step size for the one-dimensional search.

This one-dimensional search method is actually very efficient with quadratic polynomial interpolation usually being sufficient. Also, to hasten convergence, quadratic approximations can be used on the components of **dY** while using Newton's method on Eq. (2.32).

Up to this point, it has been assumed that the initial design was feasible. Sometimes, however, it is unavoidable to start from an infeasible design. If the initial design is not feasible, the first step is to obtain a feasible point from which feasibility can be maintained. This is given top priority because the GRG method is based on the requirement that the constraints are satisfied exactly and remain so throughout the optimization process. There are a few different methods for finding a feasible start point such as employing Newton's method, minimizing the sum of constraint violations, or minimizing the objective while heavily penalizing the sum of the constraint violations. See Reference (9) for a more detailed discussion of these methods as they apply to initially infeasible start points.

In general, the GRG method is a very efficient direct method. Its application is especially worthwhile if function and gradient evaluations are costly. However, the method does have difficulties in some areas. For instance, if the problem is highly

nonlinear, the use of Newton's method to maintain feasibility during the one dimensional search may become ill-conditioned and may not converge, requiring very small steps for convergence. Also, if there are many inequality constraints, storage requirements may become large as does the solution of the dependent variable subproblem. Furthermore, the method by which the GRG algorithm returns infeasible solutions to the constraint boundary (Newton's method) is not as efficient as some other algorithms but, on the other hand, a feasible solution is guaranteed at the end of every iteration, which has its virtues. Despite the mentioned problems, the Generalized Reduced Gradient method is an efficient direct method for solving constrained optimization problems.

3. AIRFOIL DESIGN

There are three main parts involved in the optimization process: (1) the design variables, (2) the optimizer, and (3) the mathematical model. It is very important that all of these components be robust to facilitate an efficient solution to an optimization problem. In the following sections, the individual facets to the process are discussed in detail.

3.1 Design Variables

In design optimization problems, it is highly important that design variables be chosen judiciously to facilitate a streamlined analysis. For the airfoil optimization problem, an efficient method for describing the airfoil shape is required. Trying to control individual points along the airfoil surface would become too cumbersome in the analysis because too many points would need to be used to achieve any degree of surface resolution.

To minimize the number of design variables, a Bezier parameterization scheme, as outlined in Reference 7, was used to define the airfoil. This method of definition can be easily used to describe airfoils, traditional or contemporary. In general, airfoil definition using Bezier curves lends itself extremely well to the optimization process.

Airfoil construction involves a total of four Bezier curves, two for the top surface and two for the bottom surface. Each parametric curve is completely defined by a Bezier polygon consisting of four vertices. The order of a Bezier curve is defined as the number of vertices minus one, hence, the curves are cubic. Adjacent curves share endpoints or polygon vertices. A typical airfoil as defined by Bezier parameterization may be seen in Fig. 3.1.

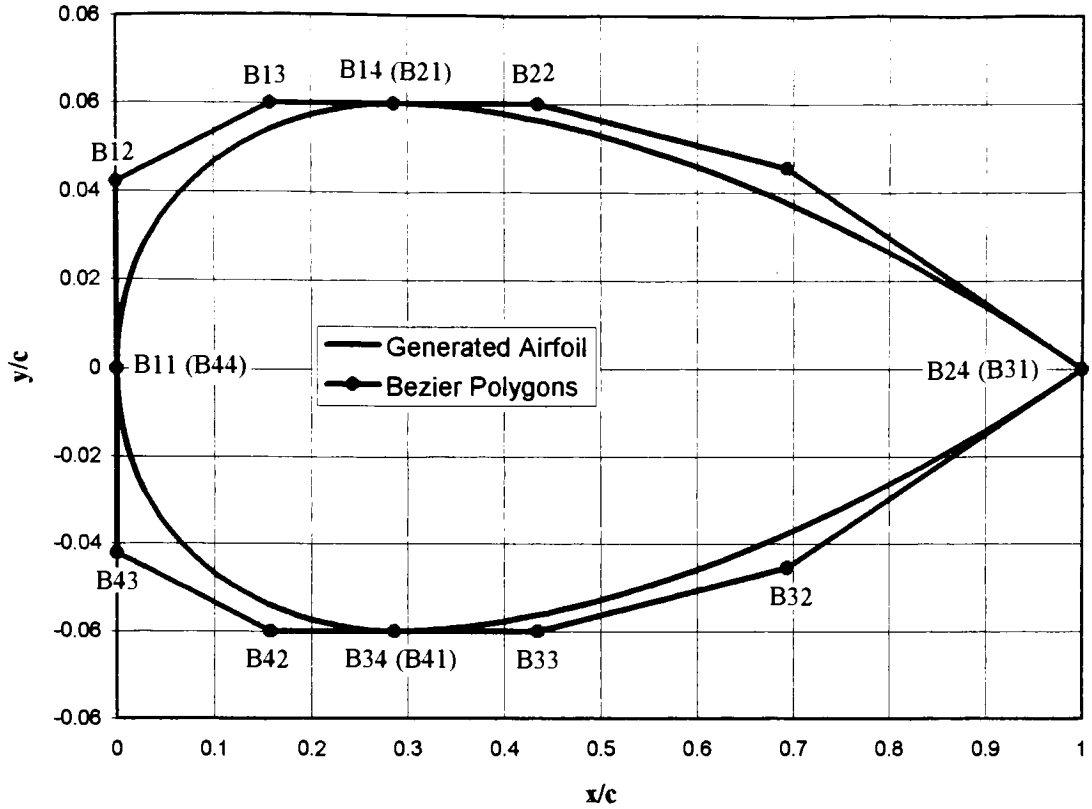
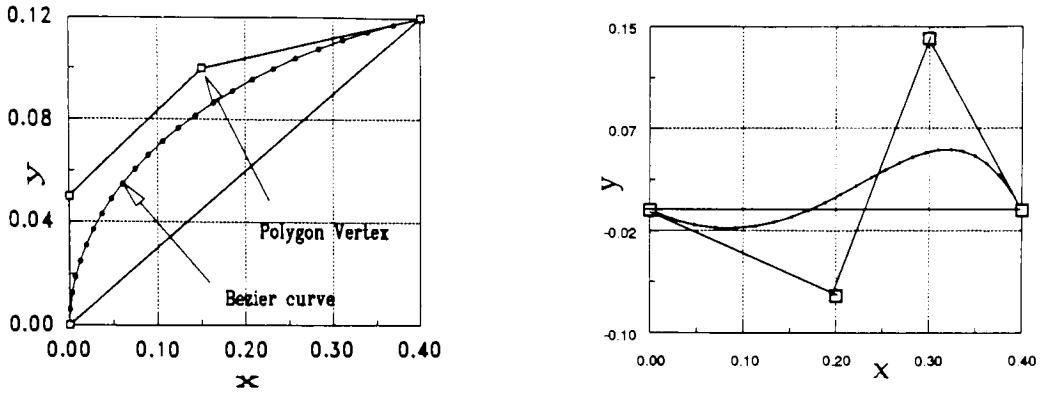


Figure 3.1:
Typical Airfoil Using Bezier Curves

As previously mentioned, Bezier parameterization lends itself very well to airfoil definition and several features of the description method enhance its usefulness. The Bernstein basis functions are used for the Bezier parameterization. Referring to Figs. 3.2a and 3.2b (from Reference 7), the following useful properties have been used to aid in defining design variables for the airfoil geometry: (1) the basis functions are real, (2) as previously mentioned, the order of the Bezier curve is defined as one less than the number of vertices of its corresponding polygon, (3) the first and last points of a Bezier curve share common points with the first and last points of its polygon, (4) at the endpoints, the

slopes of the polygon and the curve are equal, and (5) the curve is confined within the convex hull of the polygon.



(a) (b)

Figures 3.2a and 3.2b:
Examples of Bezier Curves

A cubic curve has four corresponding polygon vertices: B_0, B_1, B_2, and B_3, where each B_i represents a point in two-dimensional, x-y space. A parameter v, lying between one and zero, is used to characterize any point P(x,y) on the curve which may be found by the equation

$$P(v) = (x, y) = \sum_{i=0}^{3} B_i J_{3,i}(v) \tag{3.1}$$

where $J_{3,i}$ is the Bernstein basis, or blending function, and is defined as

$$J_{3,i}(v) = \binom{3}{i} v^i (1-v)^{3-i} \tag{3.2}$$

with

$$\binom{3}{i} = \frac{3!}{i!(3-i)!} \tag{3.3}$$

Notice that this is nothing more than the probability density function of the binomial distribution where parameters n and p equal 3 and ν, respectfully. The binomial distribution is defined as

$$g(y) = \binom{n}{y} p^y (1-p)^{n-y} \qquad y = 0, 1, 2, \ldots, n \qquad (3.3)$$

A more in depth discussion of the usefulness and applicability of Bezier parameterization of airfoils, as developed by Venkataraman, is considered in Reference (7). Also see Reference (11) for a more in depth discussion of the theory behind Bezier curves.

To construct an airfoil from four Bezier curves, some constraints need to be defined. First, the leading edge of the airfoil is shared by the leftmost top and bottom curves (refer to Fig. 3.1). To ensure a continuous and properly defined leading edge, the three vertices closest to it should be required to make a vertical line. This will make the leading edge the leftmost point. Similarly, the rear two segments share the trailing edge but no constraint is placed on the adjacent vertices. Next, a continuity restriction must be imposed on the shared vertices between the left and right segments for both the top and bottom surfaces. The three vertices in the vicinity of the shared vertex should all lie on a straight line to guarantee slope continuity. For the optimization problem at hand, two different definition schemes were used. Both used a horizontal line constraint for the upper surface but one used a horizontal line constraint for the lower surface while the other used an unrestricted sloped line. For convenience, the horizontal line definition is called scheme 1 and the sloped line definition is called scheme 2. The different schemes were used to investigate the effect of different definition methods on final solutions. Scheme 1 would normally lend itself to traditional airfoil definition while scheme 2 would normally be

better for more contemporary airfoils with high camber. Figure 3.1 shows a typical airfoil shape as defined by scheme 1 and Fig. 3.3 shows a typical airfoil shape as defined by scheme 2.

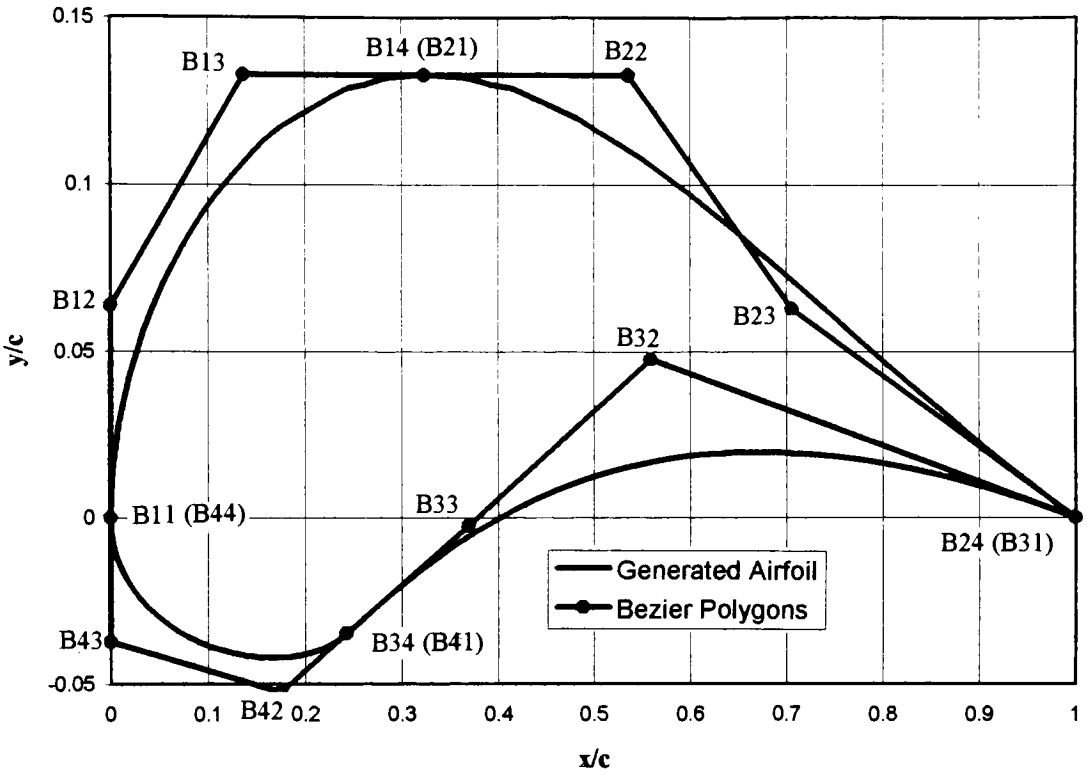


Figure 3.3:
Typical Airfoil Defined by Scheme 2

The actual design variables are described by some of the polygon vertices. Because of the definition constraints just outlined, it is unnecessary to use all of the vertices as design variables. Therefore, the constraints also help to reduce the number of design variables. Also, the leading edge coordinates as well as the trailing edge coordinates need not be design variables since their locations are fixed at (0,0) and (1,0), respectfully. Scheme 1, referring to Fig. 3.1, requires the determination of both the abscissa and

ordinate values for $B_{1,3}$, $B_{2,3}$, $B_{3,2}$, and $B_{3,3}$; the abscissa values for $B_{1,4}$, $B_{2,2}$, $B_{3,4}$, and $B_{4,2}$; and the ordinate values for $B_{1,2}$, and $B_{4,3}$ (the first number in the subscript refers to the segment number and the second number refers to the vertex number). Scheme 2, referring to Fig. 3.3 requires the determination of both the abscissa and ordinate values for $B_{1,3}$, $B_{2,3}$, $B_{3,2}$, $B_{3,3}$ and $B_{3,4}$; the abscissa values for $B_{1,4}$, $B_{2,2}$, and $B_{4,2}$; and the ordinate values for $B_{1,2}$, and $B_{4,3}$. This makes fourteen total design variables for scheme 1 and fifteen total design variables for scheme 2.

In addition to slope continuity constraints, some limitations are placed on vertices such that only realistic airfoils are allowed. This is done by constraining the magnitude of the abscissas and by constraining the distance between adjacent ordinates. For example, the difference between the locations of $B_{1,3}$ and $B_{1,2}$ may be constrained to be less than 30% of the chord. Likewise, the abscissa value of $B_{2,3}$ may be constrained to be less than 10% of the chord. These are simple linear constraints which are easy to analyze. There are 18 of these constraints for scheme 1 and 20 for scheme 2. Also, maximum and minimum values were set for all of the design variables (side constraints) such that the design was constrained to be realistic.

3.2 Optimizer: OptdesX

To run the optimization of the design variables, an interactive software package called OptdesX was used. This software allows a user created analysis model to be linked to it via program subroutines which are used to define the design variables, the constraints, and the mathematical model in general. OptdesX is extremely flexible in the way it allows adjustment of the design variables and design functions. This flexibility allows for quick redefinition of problems, making it easier to investigate the effect of different parameters on the optimal solution. Another major advantage of the software is that it allows the user to efficiently save intermediate results which could be quickly recalled for further study.

OptdesX works by updating design variables based on a user specified optimization algorithm and gradient calculation method. These updated variable values are then sent to the user supplied analysis model where the mathematical model for the problem is executed. After the calculations are completed, function values are relayed back to OptdesX. This process, illustrated in Fig 3.4, is repeated until a search direction is found. Then the chosen optimization algorithm determines a suitable step length, iterates the solution, and repeats the process until convergence is satisfied.

OptdesX has two optimization methods for continuous variables available within the software. The first is a hybrid GRG-SQP method (which is called GRG in OptdesX) and the second is a traditional SQP method. The first is said to be a more robust method than the plain SQP in that it is said to be able to solve a wider variety of problems. This may be due to the fact that, although solutions may go slightly infeasible during an iteration, feasible solutions are guaranteed by the GRG method at the end of each

iteration. For these and other reasons, the GRG method is suggested as the first method to try. On the other hand, the SQP method included in OptdesX is potentially the fastest and most efficient algorithm of the two. However, during iterations, the solution may go highly infeasible which may cause trouble in many analysis models. This ill conditioning can be expected in highly nonlinear mathematical models which is the case for the airfoil problem at hand. For this reason, the GRG method was used exclusively for the airfoil optimization problem.

As previously mentioned, the GRG method used in OptdesX is actually a GRG-SQP hybrid. This method was developed by the creators of OptdesX. For a more complete description of the method, see Reference 10. Some major points of the algorithm are addressed here. The algorithm is said to be robust and efficient as would be expected from a hybrid of two efficient and commonly used direct optimization methods. An iteration of the GRG algorithm has two parts: (1) a search direction is determined , and (2) a step size is then determined for the search direction. Gradients are calculated once for each iteration and the analysis model may be called and evaluated many times during an iteration. The first step in the algorithm (search direction determination) is borrowed from the common SQP method (see Reference 9). The step size determination is borrowed from the traditional GRG method which is described in section 2.5. The hybrid algorithm takes the best features from the respective methods and combines them into an efficient, robust optimization technique.

There are two methods of gradient calculation available in OptdesX: (1) the central difference method, and (2) the forward difference method. Gradient calculations,

whichever method is used, are required and are very important for a successful execution of the optimization algorithms. For a generic function of n variables, f(**X**), the gradient vector is expressed as

$$\nabla \mathbf{f}(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_{n1}} \right]^{\mathrm{T}} \tag{3.4}$$

Central difference derivatives of a function are calculated by the formula

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x_1, x_2, \ldots, x_j + \varepsilon, \ldots, x_n) - f_i(x_1, x_2, \ldots, x_j - \varepsilon, \ldots, x_n)}{2\varepsilon} \tag{3.5}$$

and forward difference derivatives are calculated using the formula

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x_1, x_2, \ldots, x_j + \varepsilon, \ldots, x_n) - f_i(x_1, x_2, \ldots, x_j, \ldots, x_n)}{\varepsilon} \tag{3.6}$$

Where ε is the derivative perturbation. This perturbation parameter may be set by the user in OptdesX.

Choosing an appropriate gradient method and perturbation can significantly influence the success of an optimization. When considering numerical derivatives, there are trade-offs between round-off error and truncation error. If the functions being evaluated are fairly smooth, the round-off error is usually negligible and a forward difference method may be used with a small perturbation to minimize truncation error. But for highly nonlinear functions, round-off error may become significant. To reduce this, a larger perturbation may be used but this tends to increase truncation error. To keep both round-off and truncation error to a minimum, a central difference method with an increased perturbation may be used since it has a smaller order of truncation error than the

forward difference method. However, the central difference method requires twice the number of analysis calls for the calculation of function gradients. Since, for the airfoil optimization problem, was by far the most time consuming part of iterating, the forward difference gradient calculator was chosen with a modest perturbation ($\varepsilon = 0.001$). It was also believed that since double position arithmetic was used in the analysis model, the gradient calculation error would be minimized.

It has been the author's experience that OptdesX is an extremely efficient optimizer. In fact, when running other, nonrelated analysis model that were less complicated, optimization times were as low as five seconds. In general, OptdesX served as a robust tool which aided in the success of the current work.

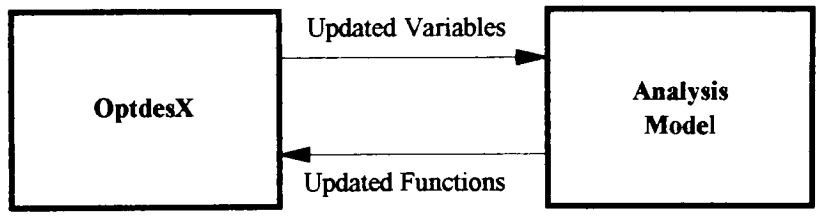


Figure 3.4:
Illustration of OptdesX/Analysis Model Interaction

3.3 Mathematical Model

For the optimization problem, an aerodynamic analysis model needed to be developed. The model was required to perform two main tasks: first, the model needed to calculate airfoil coordinates based on the Bezier vertices specified as design variables, and, second, the model needed to calculate the flow conditions and aerodynamic coefficients for the defined airfoil. The airfoil coordinates are generated by employing Eqs. (3.1) and (3.2) for the four Bezier segments necessary to define an airfoil. From this operation the, vertices are then converted into (x,y) coordinates for use in the aerodynamic section. To analyze the flow about an airfoil, the basic but adequate Hess-Smith-Douglas panel method is employed to determine inviscid conditions which are then used in an integral boundary layer method to approximate viscous effects. The development of both of these models are described in detail here.

3.3.1 Inviscid Flow Model: The Hess-Smith-Douglas Panel Method

Panel methods have been successfully utilized for many years for use in many different problems and are still used today despite the advances in CFD. These methods are appropriately named since the body surface is approximated as a collection of panels. There are many different panel methods where different types of singularities are used. These range from the very sophisticated with combinations of sources and doublets oriented normal to the surface, to the basic which makes use of a vortex acting at the trailing edge with sources distributed along the body surface. For the present model, the latter is used. It is a basic but efficient panel method which helps in reducing optimization

time. Recalling potential flow theory, the law of superposition allows the contribution of the individual singularities to be broken up into their respective components:

$$\phi = \phi_\infty + \phi_S + \phi_V \tag{3.8a}$$

Where ϕ_∞ represents the potential of the uniform flow, ϕ_S represents the potential of the source distribution, and ϕ_V represents the potential of the vortex distribution. These potentials are expressed as

$$\phi_\infty = V_\infty(x \cos \alpha + y \sin \alpha) \tag{3.8b}$$

$$\phi_S = \int \frac{q(s)}{2\pi} \ln r \, ds \tag{3.8c}$$

$$\phi_V = -\int \frac{\gamma(s)}{2\pi} \theta \, ds \tag{3.8d}$$

where for ϕ_S and ϕ_V, the integration is performed over the body surface. Figure 3.5 (from Reference 12) shows that s is the distance along the surface and (r,θ) are polar coordinates for a point (x, y) in the flow field. The source potential ϕ_S has a strength per unit length of q(s). Likewise, the vortex potential has a strength per unit length of $\gamma(s)$.

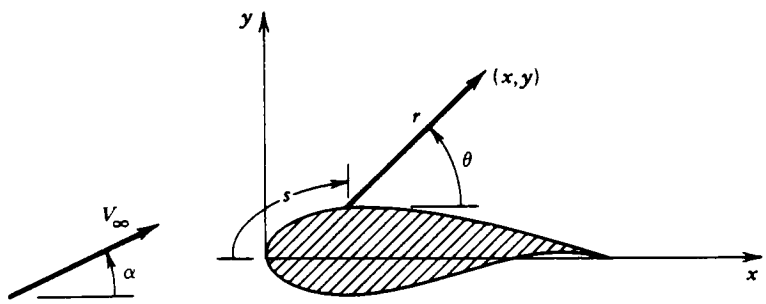


Figure 3.5:
Nomenclature for Panel Method

By virtue of the superposition principle, Eqs. (3.8) automatically satisfies the Laplace equation and the boundary conditions far from the body (infinity). However, for Eq. (3.8) to model the flow in the vicinity of the body, the boundary conditions of flow tangency and the Kutta condition must be satisfied. For inviscid flow, it is expected that flow be perfectly tangent to a body since there is no boundary layer showing the flow. Also, the Kutta condition must be satisfied which is stated as: the flow from a sharp-tailed airfoil must leave the trailing edge smoothly; that is, the velocity at the trailing edge must be finite. Some corollaries may be drawn form the Kutta condition and are as follows:

1. The trailing edge serves as a stagnation point from which a streamline emanates.
2. The stagnation streamline from a trailing edge follows the bisector of the trailing edge angle.
3. The flow velocities on the top and bottom surfaces are equal at equal distances near the trailing edge, hence, ensuring total pressure recovery.

Therefore, the problem becomes one to determine the source and vortex strengths such that these conditions are satisfied. The source strength may be thought of as governed by the flow tangency condition and, similarly, the vortex strength by the Kutta condition. This simplifies the problem in that a single, constant vortex strength may be applied over the whole airfoil while source strengths are allowed to vary to satisfy flow tangency. A single vortex strength may be used since the Kutta condition only applies the trailing edge.

Solution of Eq. (3.8) to determine the strengths, even with the established simplification, can be very difficult because of the complexity of the airfoil surface. For this reason, the surface is simplified by breaking it up into straight line segments called

panels, connected by the endpoints which are called nodes. These lines make evaluation of the integrals in Eqs. (3.8c) and (3.8d) much easier. Also, the airfoil coordinates generated by the Bezier definition can be used as the node locations. The described surface simplification is illustrated in Fig. 3.6 (from Reference 12). Distributing the sources and vortices along the panels makes Eq. (3.8) become

$$\phi = V_\infty (x \cos\alpha + y \sin\alpha) + \sum_{j=1}^{N} \int_{j^{th}\ \text{panel}} \left[\frac{q(s)}{2\pi} \ln r - \frac{\gamma}{2\pi}\theta \right] ds \tag{3.9}$$

Equation (3.9) can be used to closely approximate the exact solution with the accuracy of the solution increasing with the number of panels used (to a certain limit).

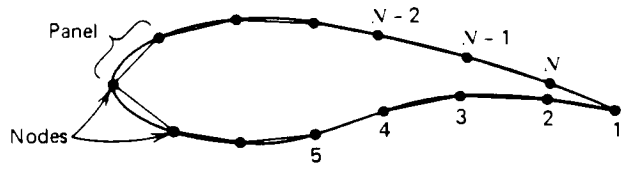


Figure 3.6:
An Airfoil Described by Panels

To further simplify the problem, the source distribution is approximated with constant strengths at each panel while, from panel to panel, the strengths are allowed to vary: $q(s) = q_i$ on panel i, i = 1, 2, ..., N. Again, the accuracy of the approximation increases with the number of panel used to model the airfoil surface.

Now, the parameters to determine are the N source strengths q_i and the vortex strength γ. These parameters must still be found such that they satisfy the flow tangency

condition and the Kutta condition. This may be done by imposing the Kutta condition and flow tangency at N control points. The midpoints of the panels are selected as the control points, since, as may be seen later, the velocity at the panel endpoints becomes infinite. Flow tangency will be imposed by requiring that the normal velocity at each i^{th} panel be zero. Also, the Kutta condition may be satisfied by equating the velocity components tangential to the first and last panels, as defined by Fig. 3.6.

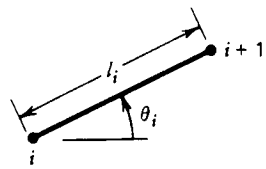


Figure 3.7:
The i^{th} Panel

For this method, the i^{th} panel is defined as the panel between the i^{th} and $(i+1)^{th}$ nodes and its inclination with the x-axis is θ_i, as shown in Fig. 3.7 (from Reference 12). So,

$$\sin \theta_i = (y_{i+1} - y_i)/l_i \tag{3.11a}$$

$$\cos \theta_i = (x_{i+1} - x_i)/l_i \tag{3.11b}$$

where l_i is the length of the i^{th} panel. It follows that the normal to the i^{th} panel is

$$\hat{\mathbf{n}}_i \equiv -\sin \theta_i \, \hat{\mathbf{i}} + \cos \theta_i \, \hat{\mathbf{j}} \tag{3.12}$$

Following the numbering scheme of Fig 3.6, the normal vector should point outward from the body. Similarly, a unit vector tangent to the i^{th} panel may be defined such that it is directed from node i to node i+1:

$$\hat{\mathbf{t}}_i \equiv \cos\theta_i \, \hat{\mathbf{i}} + \sin\theta_i \, \hat{\mathbf{j}} \tag{3.13}$$

The control point coordinates may be defined as

$$\bar{x}_i \equiv \frac{x_i + x_{i+1}}{2} \tag{3.14a}$$

$$\bar{y}_i \equiv \frac{y_i + y_{i+1}}{2} \tag{3.14b}$$

Where the velocity components at these points are defined as

$$u_i \equiv u\left(\bar{x}_i, \bar{y}_i\right) \tag{3.15a}$$

$$v_i \equiv v\left(\bar{x}_i, \bar{y}_i\right) \tag{3.15b}$$

Now the flow tangency condition may be written as

$$0 = -u_i \sin\theta_i + v_i \cos\theta_i \quad \text{for i} = 1, 2, \ldots, N \tag{3.16}$$

and the Kutta condition may be expressed as

$$u_1 \cos\theta_1 + v_1 \sin\theta_1 = -u_N \cos\theta_N - v_N \sin\theta_N \tag{3.17}$$

The velocity components u_i and v_i are made up of contributions from the onset flow, the sources on each panel, and the vortices on each panel. The velocities induced on a panel from all the sources and vortices are proportional to the strengths of the source and vortex on that panel. It follows that

$$u_i = V_\infty \cos\alpha + \sum_{j=1}^{N} q_j \, u_{Sij} + \gamma \sum_{j=1}^{N} u_{Vij} \tag{3.18a}$$

$$v_i = V_\infty \sin\alpha + \sum_{j=1}^{N} q_j\, v_{Sij} + \gamma \sum_{j=1}^{N} v_{Vij} \qquad (3.18b)$$

Where, for example, v_{Vij} is the y-component of velocity at the midpoint of the i^{th} panel due to the vortex distribution from the j^{th} panel.

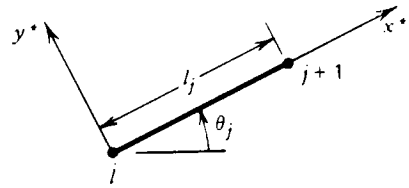


Figure 3.8:
Local Coordinate System

Now it becomes more convenient to work in local coordinate (x^*,y^*) for the evaluation of u_{Sij}, v_{Sij}, u_{Vij}, and v_{Vij}. This coordinate system is defined in Fig 3.8 (from Reference 12). Subsequently, after determining (u^*,v^*), the global velocity components can be evaluated from

$$u = u^* \cos\theta_j - v^* \sin\theta_j \qquad (3.19a)$$

$$v = u^* \sin\theta_j - v^* \cos\theta_j \qquad (3.19b)$$

From this, it can be shown that

$$\begin{aligned} u^*_{Sij} &= \frac{1}{2\pi}\int_0^{l_j} \frac{x^* - t}{(x^* - t)^2 + y^{*2}}\, dt \\ &= \frac{-1}{2\pi} \ln\left[(x^* - t)^2 + y^{*2}\right]^{1/2} \Big|_{t=0}^{t=l_j} \end{aligned} \qquad (3.20a)$$

$$v^*_{Sij} = \frac{1}{2\pi}\int_0^{l_j} \frac{y^*}{(x^* - t)^2 + y^{*2}}\,dt$$
$$= \frac{1}{2\pi}\tan^{-1}\frac{y^*}{x^* - t}\Bigg|_{t=0}^{t=l} \qquad (3.20b)$$

Where t is a dummy variable for the distance along the panel. Also, (x^*,y^*) are the local coordinates corresponding to (x_i,y_i). Figure 3.9 (from Reference 12) may be seen for a geometrical interpretation of these results:

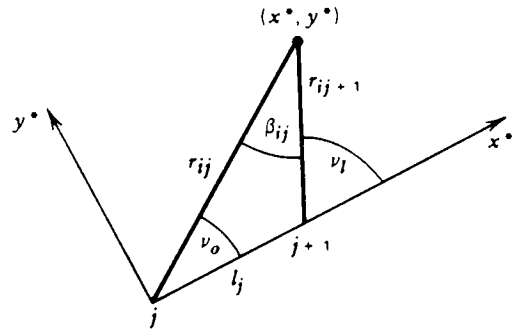


Figure 3.9:
Geometrical Interpretation of Eq. (3.20)

Now the velocity components can be written as

$$u^*_{Sij} = \frac{-1}{2\pi}\ln\frac{r_{ij+1}}{r_{ij}} \qquad (3.21a)$$

$$v^*_{Sij} = \frac{\nu_1 - \nu_0}{2\pi} = \frac{\beta_{ij}}{2\pi} \qquad (3.21b)$$

Where, as shown in Fig. 3.9, r_{ij} is the distance from the midpoint of the i^{th} panel to the j^{th} node, r_{ij+1} is the distance from the midpoint of the i^{th} panel to the $(j+1)^{th}$ node, and β_{ij} is the

angle defined by these lines. It can be shown, using a convenient FORTRAN expression, that β_{ij} is

$$\beta_{ij} = \pi \qquad \text{if } i = j \tag{3.22a}$$

$$\begin{aligned} &= \text{ATAN 2}\Big[(\bar{y}_i - y_{j+1})(\bar{x}_i - x_j) - (\bar{x}_i - x_{j+1})(\bar{y}_i - y_j), \\ &\quad (\bar{x}_i - x_{j+1})(\bar{x}_i - x_j) - (\bar{y}_i - y_{j+1})(\bar{y}_i - y_j) \Big] \quad \text{if } i \neq j \end{aligned} \tag{3.22b}$$

Similarly, the local velocity components due to the vortex distribution may be found as

$$u^*_{Vij} = \frac{-1}{2\pi} \int_0^{l_j} \frac{y^*}{(x^* - t)^2 + y^{*2}} dt = \frac{\beta_{ij}}{2\pi} \tag{3.23a}$$

$$v^*_{Vij} = \frac{-1}{2\pi} \int_0^{l_j} \frac{x^* - t}{(x^* - t)^2 + y^{*2}} dt = \frac{1}{2\pi} \ln \frac{r_{ij+1}}{r_{ij}} \tag{3.23b}$$

Now the flow tangency condition may be written as

$$\sum_{j=1}^{N} A_{ij}\, q_j + A_{iN+1}\gamma = b_i \tag{3.24a}$$

Where

$$\begin{aligned} A_{ij} &= -u_{Sij} \sin\theta_i + v_{Sij} \cos\theta_i \\ &= -u^*_{Sij} (\cos\theta_i \sin\theta_i - \sin\theta_i \cos\theta_i) \\ &\quad + v^*_{Sij} (\sin\theta_i \sin\theta_i - \cos\theta_i \cos\theta_i) \end{aligned} \tag{3.24b}$$

So,

$$2\pi A_{ij} = \sin(\theta_i - \theta_j) \ln \frac{r_{ij+1}}{r_{ij}} + \cos(\theta_i - \theta_j)\beta_{ij} \tag{3.25}$$

Similarly,

$$2\pi A_{iN+1} = \sum_{j=1}^{N} \cos(\theta_i - \theta_j) \ln \frac{r_{ij+1}}{r_{ij}} - \sin(\theta_i - \theta_j)\beta_{ij} \tag{3.26}$$

and

$$b_i = V_\infty \sin(\theta_i - \alpha) \tag{3.27}$$

Also, the Kutta condition from Eq. (3.17) can be rewritten as

$$\sum_{j=1}^{N} A_{N+1,j}\, q_j + A_{N+1,N+1}\gamma = b_{N+1} \tag{3.28}$$

and, in similar fashion to Eqs. (3.25) through (3.27), we find

$$2\pi A_{N+1,j} = \sum_{j=1}^{N} \sin(\theta_k - \theta_j)\beta_{kj} - \cos(\theta_{ki} - \theta_j) \ln \frac{r_{kj+1}}{r_{kj}} \tag{3.29}$$

$$2\pi A_{N+1,N+1} = \sum_{k=1}^{N}\sum_{j=1}^{N} \sin(\theta_k - \theta_j) \ln \frac{r_{kj+1}}{r_{kj}} + \cos(\theta_{ki} - \theta_j)\beta_{kj} \tag{3.30}$$

$$b_{N+1} = -V_\infty \cos(\theta_1 - \alpha) - V_\infty \cos(\theta_N - \alpha) \tag{3.31}$$

Now, the source strengths q_i, i = 1, 2, ..., N, and the vortex strength γ can be found by solving the N+1 system of equations defined in Eqs. (3.24a) and (3.28). After this, the tangential velocity at each control point can be calculated. Manipulating Eqs. (3.13), (3.18), (3.19), (3.21), and (3.23) yields

$$\begin{aligned} V_{ti} = {} & V_\infty \cos(\theta_i - \alpha) \\ & + \sum_{j=1}^{N} \frac{q_j}{2\pi}\left[\sin(\theta_i - \theta_j)\beta_{ij} - \cos(\theta_i - \theta_j) \ln \frac{r_{ij+1}}{r_{ij}}\right] \\ & + \frac{\gamma}{2\pi}\sum_{j=1}^{N}\left[\sin(\theta_i - \theta_j) \ln \frac{r_{ij+1}}{r_{ij}} + \cos(\theta_i - \theta_j)\beta_{ij}\right] \end{aligned} \tag{3.32}$$

and since $V_{n,i} = 0$, the pressure coefficient can be calculated by

$$c_p(\bar{x}_i,\ \bar{y}_i) = 1 - \frac{V_{ti}^2}{V_\infty^2} \tag{3.33}$$

Finally, the inviscid aerodynamic coefficients may be calculated from

$$c_L = -\left[\left(\sum_{i=1}^{N} c_{p_i} dx_i\right)\cos\alpha + \left(\sum_{i=1}^{N} c_{p_i} dy_i\right)\sin\alpha\right] \tag{3.34a}$$

$$c_D = \left(\sum_{i=1}^{N} c_{p_i} dx_i\right)\cos\alpha + \left(\sum_{i=1}^{N} c_{p_i} dy_i\right)\sin\alpha \tag{3.34b}$$

$$c_M = \sum_{i=1}^{N} c_{p_i}\left(dx_i \bar{x}_i + dy_i \bar{y}_i\right) \tag{3.34c}$$

Where $dx_i = x_{i+1} - x_i$ and $dy_i = y_{i+1} - y_i$.

This concludes the steps required to analyze the inviscid flow condition about an airfoil. The Hess-Smith-Douglas panel method presented is very efficient and cheap to calculate. The next step is to determine a suitable model for the viscous flow effects.

3.3.2 Viscous Flow Model

After the results are obtained form the previously described panel method, they can be used to help calculate a viscous correction to the inviscid solution. The correction for the present method has two forms: (1) skin friction is calculated as a correction to the updated c_D, and (2) the normal velocity at each panel is calculated and used to recalculate the pressure distribution, hence, updating c_D for form drag effects. In order to update the inviscid solution, an integral boundary layer calculator was employed. The viscous flow model utilized separate methods to calculate both the laminar and the turbulent flow regions. Also, two different transition criteria models were applied to the boundary layer approximator. Discussion of the specifics of the boundary layer model is preceded by a short review of the relevant theory.

As implied above, skin friction is an important parameter in the solution of the boundary layer. The calculation of the skin friction is based on shear stresses within the boundary layer:

$$\tau = \mu \frac{\partial u}{\partial y} \tag{3.35}$$

At the surface, or wall, of a body, the shear stress is

$$\tau_w = \mu \left. \frac{\partial u}{\partial y} \right|_{y=0} \tag{3.36}$$

This wall shear stress, τ_w, may be expressed in dimensionless terms:

$$c_f \equiv \frac{\tau_w}{\frac{1}{2}\rho V_e^2} \tag{3.37}$$

c_f is called the skin friction coefficient and, as will be seen later, it becomes important in correcting the drag coefficient, c_D, for skin friction effects.

The boundary layer displacement thickness is an important parameter for correcting the pressure distribution for the presence of a boundary layer. To do this, the normal component of the velocity needs to be calculated

$$V_n\big|_{y=0} = \lim_{y/\delta \to \infty} v(x,y) \tag{3.38}$$

To calculate the right hand side of Eq. (3.38), the continuity equation (div **V** = 0) may be used to get

$$\begin{aligned} v(x,y) &= \int_0^y \frac{\partial v}{\partial y}(x,y^*)dy^* \\ &= -\int_0^y \frac{\partial u}{\partial x}(x,y^*)dy^* \\ &= -\frac{\partial}{\partial x}\int_0^y u dy^* \end{aligned} \tag{3.39}$$

This can be written as

$$v(x,y) = \frac{\partial}{\partial x}\int_0^y \left[V_e(x) - u(x,y^*)\right]dy^* - y\frac{dV_e}{dx} \tag{3.40}$$

For large y^* compared to δ, $u(x,y^*) \approx V_e(x)$ ($u \to V_e$ as $y/\delta \to \infty$). Therefore,

$$v(x,y) = \frac{d}{dx}V_e\delta^* - y\frac{dV_e}{dx} \tag{3.41}$$

Where

$$\delta^*(x) \equiv \int_0^\infty \left[1 - \frac{u(x,y)}{V_e(x)}\right]dy \tag{3.42}$$

Equation 3.42 defines the displacement thickness which represents the distance the external flow streamlines are displaced by the boundary layer. That is, for a height h in the free stream, a height $h+\delta^*$ is required to allow the same volume rate of flow.

By evaluating Eq. (3.41) at y = 0, we get

$$V_n\big|_{y=0} = \frac{d}{dx}(V_e\delta^*) \tag{3.43}$$

This formula may be used as a correction to the right hand side of the system of equations described in Eq. (3.24a). Reevaluation of the system with the update would yield a pressure distribution which accounted for boundary layer effects.

Also, before proceeding with the discussion of the boundary layer model, a few more things should be introduced. The momentum thickness is another method for quantifying the boundary layer thickness. It plays a role in some empirical formulas for drag calculations. The momentum thickness θ is stated as

$$\theta \equiv \int_0^\infty \frac{u}{V_e}\left(1-\frac{u}{V_e}\right)dy \tag{3.44}$$

The momentum thickness and the displacement thickness may be related through a shape factor H:

$$H \equiv \delta^*/\theta \tag{3.45}$$

All of these major outputs of a boundary layer analysis (δ^*, θ, c_f) are all connected by the Karmen momentum integral equation:

$$\frac{d\theta}{dx}+\frac{\theta}{V_e}(2+H)\frac{dV_e}{dx} = \frac{1}{2}c_f \tag{3.46}$$

A derivation of Eq. 3.46 may be seen in Reference 12. Unfortunately, there are too many unknowns in the equation to make it useful by itself. However, other equations may be used in conjunction with it to solve the boundary layer problem. This is the focus of many integral methods.

To solve the laminar region of the boundary layer, Thwaites method is employed. This method makes use of the momentum integral equation without having to make any assumptions about the form of the velocity profile. This is one of the virtues of the method since velocity profile fitting is usually an inefficient and often unreliable approach to boundary layer calculation.

The basic idea behind the method is to supplement Eq. (3.46) with equations involving the unknowns θ, H, and c_f. First, to make things easier, θ and x are made dimensionless by forming

$$R_{e,\theta} = \frac{\rho V_e \theta}{\mu} \tag{3.47a}$$

$$R_{e,x} = \frac{\rho V_e x}{\mu} \tag{3.47b}$$

H and c_f are already dimensionless but c_f is a strong function of Reynolds number. To alleviate this, a parameter which is independent of Reynolds number is introduced.

$$l \equiv (R_{e,\theta}\, c_f)/2 \tag{3.48}$$

Next, the momentum integral equation is multiplied by $R_{e,\theta}$, yielding

$$\frac{\rho V_e \theta}{\mu}\frac{d\theta}{dx} + \frac{\rho \theta^2}{\mu}(2+H)\frac{dV_e}{dx} = l \tag{3.49}$$

Also, a dimensionless pressure gradient parameter λ is defined as:

$$\lambda \equiv \frac{\rho\theta^2}{\mu}\frac{dV_e}{dx} \tag{3.50}$$

The two previous equations can be combined and written as

$$\frac{\rho V_e}{\mu}\frac{d\theta^2}{dx} = 2[l-(2+H)\lambda] \tag{3.51}$$

From Thwaites, the right hand side can be accurately approximated by a linear equation:

$$2[l - (2+H)\lambda] \approx 0.45 - 6\lambda \tag{3.52}$$

This and the definition of λ may be substituted into Eq. (3.51), yielding

$$\frac{\rho V_e}{\mu}\frac{d\theta^2}{dx} = 0.45 - \frac{6\rho\theta^2}{\mu}\frac{dV_e}{dx} \tag{3.53}$$

After multiplying by V_e^5 and some rearrangement, the equation becomes

$$\frac{\rho}{\mu}\left(V_e^6\frac{d\theta^2}{dx} + \theta^2 6V_e^5\frac{dV_e}{dx}\right) = \frac{\rho}{\mu}\frac{d}{dx}(\theta^2 V^6) = 0.45V_e^5 \tag{3.54}$$

This equation, with any given $V_e(x)$ and an initial value of θ, can be solved for $\theta(x)$. All that is required is the integration of a first order ordinary differential equation (ODE). Once θ is known, λ can be calculated from Eq. (3.50), and using the empirical Thwaites correlation formulas suggested in Reference (12), $l(\lambda)$ and $H(x)$ can be calculated:

$$l(\lambda) = 0.22 + 1.57\lambda - 1.8\lambda^2 \qquad \text{for } 0 < \lambda < 0.1 \tag{3.55a}$$

$$= 0.22 + 1.402\lambda + \frac{0.018\lambda}{\lambda + 0.107} \qquad \text{for } -0.1 < \lambda < 0 \tag{3.55b}$$

$$H(\lambda) = 2.61 - 3.75\lambda + 5.24\lambda^2 \qquad \text{for } 0 < \lambda < 0.1 \tag{3.56a}$$

$$= 2.088 + \frac{0.0731}{\lambda + 0.14} \qquad \text{for } -0.1 < \lambda < 0 \tag{3.56b}$$

The initial condition for the solution of the ODE is θ at the stagnation point:

$$\theta(0) = \sqrt{\frac{0.075\mu}{\rho \dot{V}_o}} \tag{3.57}$$

Which is determined by solving Eq. (3.54) at the initial conditions ($V_e(0) \approx \dot{V}_o x$). Then Eq. (3.54) may be solved for $\theta(x)$ analytically.

For the present method, two different transition criteria are used. While evaluating the upper surface, Michel's criteria is used but while evaluating the lower surface, the point of minimum pressure is taken as the transition point.

Transition starts at a critical value of Reynolds number. For example, for a smooth flat plate, $R_{e,,critical} \approx 2.8 \times 10^6$ (Reference 12). Transition Reynolds number is a function of many different parameters. Most important are the imposed pressure gradients from the ideal solution and surface roughness. The critical Reynolds number is lowered for increased surface roughness and a positive value of dP/dx.

Michel's method predicts that transition should be expected when

$$R_{e,\theta} > 1.174\left(1+\frac{22{,}400}{R_{e,x}}\right)R_{e,x}^{0.46} \tag{3.58}$$

This method accounts for the effect of pressure gradients since θ grows more rapidly in positive pressure gradients. It does not, however, account for surface roughness but this method should still be good for airfoil analysis since published data doesn't show a strong dependency on surface roughness (see Reference (13)).

Michel's method is used for the top surface calculations but, due to instabilities in the method, transition is fixed to a point based on the point of minimum pressure for the lower surface.

These methods are used to fix transition to a specific point which isn't what actually occurs. As a matter of fact, transitional flow occurs over a segment of the airfoil not at a specific point. Within this region of transition, the flow oscillates between laminar and turbulent. Skin friction is typically high in this region and since the present method approximates transition as occurring at a point, the increased drag is not accounted for. The transition method for the boundary layer boundary layer program is basically used to toggle flow calculations from laminar to turbulent. Despite the deficiencies of the transition model, it is used since there are no other integral methods for the prediction of transition.

Finally, Head's method is used to predict the turbulent flow region of the boundary layer. The method is based on the concept of an entrainment velocity. The volume flowrate within the boundary layer at x is

$$Q(x) = \int_0^{\delta(x)} u dy \tag{3.59}$$

Where $\delta(x)$ is the boundary layer thickness. The rate at which Q increases with x is known as the entrainment velocity, E:

$$E = \frac{dQ}{dx} \tag{3.60}$$

The displacement thickness may be written as

$$\delta^* = \delta - \frac{Q}{V_e} \tag{3.61}$$

Which leads to

$$E = \frac{d}{dx} V_e (\delta - \delta^*) \tag{3.62}$$

Which can be written as

$$E = \frac{d}{dx}(V_e \theta H_1) \tag{3.63}$$

Where

$$H_1 \equiv \frac{\delta - \delta^*}{\theta} \tag{3.64}$$

It was assumed that a dimensionless entrainment velocity E/V_e was only dependent on H_1 which, in turn, is dependent on H. Curves have been fitted to many sets of experimental data, the equations of which were reported in Reference (12), and are as follows:

$$\frac{1}{V_e}\frac{d}{dx}(V_e \theta H_1) = 0.0306(H_1 - 3)^{-0.6169} \tag{3.65}$$

$$H_1 = 3.3 + 0.8234(H - 1.1)^{-1.287} \qquad \text{for } H \leq 1.6 \tag{3.66a}$$

$$= 3.3 + 1.5501(H - 0.6778)^{-3.064} \qquad \text{for } H > 1.6 \tag{3.66b}$$

Head's method employs another empirical equation known as the Ludwieg-Tillman skin friction law:

$$c_f = 0.246 \times 10^{-0.678H} R_{e,\theta}^{-0.268} \tag{3.67}$$

These three equations along with the momentum integral equation comprise the four equations required to solve for the four unknowns θ, H, H_1, and c_f.

The boundary layer method also handles flow separation, albeit primitive. The discussion of separation criteria and the model's reaction to separated flow is addressed in Section 4.1.

Both Thwaites' and Head's method produce the required data to perform the aforementioned inviscid solution update. The implementation of the analysis methods described here is discussed in depth in chapter 4.

4. PROGRAM DEVELOPMENT

The Bezier definition theory described in Section 3.1 and the mathematical model for the aerodynamic analysis needed to be encoded to interface with OptdesX. This chapter describes the main aspects of the developed program. Also, the code was validated by comparing the obtained results against published data for some well known airfoil sections.

4.1 Discussion of Program

As mentioned in Chapter 3, a program needed to be developed to execute an aerodynamic analysis model. FORTRAN was the programming language used to accomplish this and a copy of the source code may be seen in Appendix I. As seen in Fig. 4.1 (end of chapter), there were many subroutines used to execute the analysis. Following the numbering shown in Fig. 4.1, the main steps in the program may be described as follows:

(1) Updated variables are sent from OptdesX to subroutine anafun. Anafun serves as the main program for the model even though it is just a subroutine of OptdesX. OptdesX communicates with the analysis model through variable and function subroutine calls (avdsca and afdsca, respectively). Then from anafun, most of the other subroutines are called.

(2) Subroutine coord is then called from anafun. This subroutine serves one of the main functions of the analysis model - airfoil definition. The function of this subroutine is

to determine the airfoil coordinates based on the Bezier vertices which are specified as the design variables received from OptdesX. Based on the parameter 'npoints', 'nact' nodes and 'npanels' panels are defined where npanels = nact - 1. (3) Also, subroutine amult is called from coord to perform repetitive multiplications involved in calculating the airfoil coordinates.

(4) Next, subroutine ambient is called from anafun and is basically used to calculate sin α and cos α. The subroutine used to have the function of setting ambient conditions, but this has since been directly incorporated into the user interface of OptdesX.

(5) Psslope is the next subroutine called by anafun. the function of this routine is to calculate the slopes of the individual panels. Also, the actual panel lengths are calculated and stored.

(6) One of the other main analysis subroutines, coeffnt, is called next. It is in coeffnt that the inviscid panel method problem is set up. Coeffnt sets up the system of equations as described by Eqs. (3.24a) and (3.28). The coefficient matrix is stored in a 'nact x nact' matrix a(i, j), and the freestream velocity vector of length 'nact' is stored in c(i). Also, for reasons explained later, a copy of c(i) is stored in the vector tempo(i).

(7) Next, for the solution of the inviscid problem, $\mathbf{[A]\ x = c}$, subroutine solsys is called. The method of solution is Gaussian elimination with partial pivoting. In order to reduce space requirements, the solution is stored and returned in the vector c(i). Unfortunately the original vector is needed for the viscous update, hence, the vector is stored in tempo before it is sent to solsys. It should be mentioned that the present work is

based from an existing program that calculated optimum airfoils subjected to ideal flow (see References (14) and (12)), so the storage of c was not required.

(8) After returning from solsys, subroutine veldist is called. What started originally as a subroutine to calculate the inviscid pressure and velocity distributions turned into a gateway and setup subroutine for the boundary layer calculator. As can be seen in Fig. 4.1 veldist is very active in calling other subroutines. The major functions of veldist are: it still calculates velocity and pressure distribution for both the inviscid and viscous solutions; (9) it calls clcdcm for the inviscid flow coefficients which are calculated according to Eqs. (3.34a) to (3.34c); it determines the stagnation point for the airfoil and subsequently splits up the airfoil panels to define upper and lower surfaces; it finds the transition point for the minimum pressure transition model which is used for the lower surface; (10) it calls intgrl twice for the viscous solutions of both the upper and lower surfaces; it updates the c vector (or more appropriately, the tempo vector) with a normal velocity correction obtained from information calculated in intgrl; and (17) it calls solsys for the reevaluation of the $\mathbf{[A]\,x = c}$ problem with the incorporated viscous effects. Some of these features are addressed in more detail later.

In the description of veldist, steps 11 through 16 were skipped. Intgrl, too, has many routines which it calls. Subroutine thwats is used to calculate Eqs. (3.55) and (3.56) for the laminar region predicted by Thwaites' method. The rest of the functions and subroutines (12 through 16) are used to analyze the turbulent boundary layer region predicted by Head's method. The subroutine runge2 is used to perform a second-order

Runge-Kutta method on the system of two first-order ODE's defined by Eqs. (3.49) and (3.65).

Lastly, clcdcm is called again from anafun for the calculation of the aerodynamic coefficients which are corrected for the presence of a boundary layer.

Since much of the program was done by others, discussion of the specific aspects are limited to that which the author contributed. The discussion of these features are presented in the order that they are executed in the program.

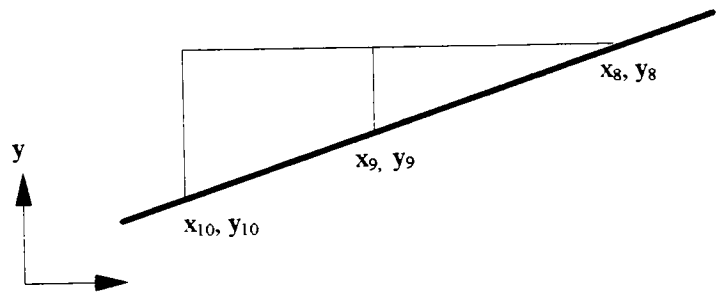


Figure 4.2:
Sloped Line Definition

First, a slight modification in the optimization constraints was made to accommodate a sloped line definition for the three Bezier vertices B_8, B_9, and B_{10}. This definition method was referred to as scheme 2 in Section 3.1. Before modification, constraints were placed on y_9, and y_{10} such that they were forced to be equal to y_8, thus ensuring a horizontal line definition scheme (scheme 1) for the bottom surface. To allow a free slope of the line defined by the points B_8, B_9, and B_{10}, y_9 was made to be a design

variable. Then y_{10} was constrained to lie on the line defined by y_8 and y_9. Referring to Fig. 4.2, the slope of the line may be found as

$$m = \frac{y_8 - y_9}{x_8 - x_9} \tag{4.1}$$

and knowing the equation of a line to be

$$y - y_1 = m(x - x_1) \tag{4.2}$$

Where (x_1, y_1) is a reference point. If B_8 is taken as the reference point, then the constraint on y_{10} may be expressed as

$$y_{10} = m(x_{10} - x_8) + y_8 \tag{4.3}$$

This constraint allows the line to be sloped (positive or negative) while maintaining the three points on the same line.

The next feature developed was the method for determination of the stagnation point. Because of the convention by which the tangent vector is defined (i^{th} to $(i+1)^{th}$ node, starting from the trailing edge in a clockwise direction), the velocity on the bottom surface is negative. This makes determination of the stagnation point simple. The program looks through the velocity data starting at the trailing edge until it finds a non-negative number. After finding the first non-negative velocity, say at the i^{th} panel, the panels may be split into an upper and a lower surface. To calculate the velocity gradient for the first panel after the stagnation point (this is required for the boundary layer calculation), the adjacent panel on the other side of the stagnation point is included in that surface and is made to be the first panel. After calculations are made, however, the data from that first panel is discarded. This is done because the data would be redundant with the data found by analyzing the same panel but on the other surface. Once the panels are split into sides, they

are renumbered such that the first panel is at the stagnation point, not the trailing edge. This is done because the boundary layer model calculates starting from the stagnation point.

Another feature developed for the program was the minimum pressure criteria to fix transition. The point of minimum pressure is found in similar fashion to the stagnation point. The pressure data is scanned starting at the trailing edge in a clockwise direction. The program looks for a value of pressure that is greater than the previous, and in satisfying this criteria, the point of minimum pressure is found. Recall that this method for fixing transition is only used for the lower surface.

To calculate the normal velocity correction as stated in equation (3.43), the displacement thickness needs to be calculated. This is straightforward because the momentum thickness θ and the shape factor H are already calculated in the solution of both Thwaites and Head's method. Recall these parameters are related by the equation $H = \delta^*/\theta$. So, to calculate the correction for each i^{th} panel, δ^*_i needs to be multiplied with its respective $V_{e,i}$. After doing this for every panel, the gradient for each product $\delta^*_i V_{e,i}$ needs to be computed. Before doing this, the data should be rearranged such that it starts at the trailing edge and is numbered in a clockwise fashion. The gradients for each panel are then calculated by a forward difference method which employs a three point quadratic approximation of the function. If the products are denoted as P_i, i = 1, npanels, then the function is

$$P_i(x) = a_0 + a_1 x_i + a_2 x_i^2 \qquad i = 1, 2, 3$$

$$\text{Given } (x_i, P_i) \tag{4.4}$$

Where x_i is the surface length for the panel. So the derivative of P_1 is

$$\frac{dP_1}{dx} = a_1 + 2a_2 x_1 \tag{4.5}$$

and it can be shown for a quadratic equation with three known points that the coefficients a_1 and a_2 are

$$a_1 = \frac{P_2 - P_1}{x_2 - x_1} - a_2(x_1 + x_2) \tag{4.6}$$

$$a_2 = \frac{\left(\frac{P_3 - P_1}{x_x - x_1}\right) - \left(\frac{P_2 - P_1}{x_2 - x_1}\right)}{x_3 - x_2} \tag{4.7}$$

Substituting Eqs. (4.6) and (4.7) into Eq. (4.5) and after some manipulation it may be shown that

$$\frac{dP_1}{dx} = \frac{\frac{(P_2 - P_1)(x_3 - x_1)}{x_2 - x_1} - \frac{(P_3 - P_1)(x_2 - x_1)}{x_3 - x_1}}{x_3 - x_2} \tag{4.8}$$

Then, after finding the gradients, every i^{th} component can be added to the respective component of the vector tempo(i), with the sum being renamed as c(i):

$$c(i) = tempo(i) + \frac{d}{dx}(V_{e,i}\delta_i^*) \tag{4.9}$$

This updated vector is all that is needed to reevaluate the **[A] x = c** problem because the coefficient matrix **A** will not change with viscous effects.

Skin friction is also calculated for both the upper and lower surfaces by the program and is done so that it could be added to the form drag found from the viscous update just outlined. It was established from Eq. (3.37) that

$$c_j \equiv \frac{\tau_w}{\frac{1}{2}\rho V_e^2}$$

and drag can be defined as

$$c_{D,f} = \frac{\int_S \tau_w ds}{\frac{1}{2}\rho V_e^2 S} \tag{4.10}$$

Rearranging yields

$$c_{D,j} = \frac{1}{S}\int_S \frac{\tau_w}{\frac{1}{2}\rho V_e^2} ds = \frac{1}{S}\int_S c_f ds \tag{4.11}$$

This can be approximated as

$$c_{D,f} \approx \frac{\sum_{j=1}^{nx} c_j S_j}{\sum_{j=1}^{nx} S_j} \tag{4.12}$$

Where nx is the number of points on a surface and s_j is the length of the j^{th} panel. The skin friction drag coefficients from both surfaces are added together and then the sum is added to the form drag, yielding a total drag coefficient:

$$c_D = c_{D,\text{form}} + c_{D,\text{skin friction}} \tag{4.13}$$

Lastly, subroutine intgrl has the ability to handle flow separation in a very limited capacity. For Thwaites method, laminar separation is predicted to occur when $l(\lambda)$ from Eq. (3.55) vanishes since the parameter is proportional to the wall shear stress (wall shear stress equals zero at separation). By the correlation formulas, $l(\lambda)$ is predicted to vanish when $\lambda = -0.0842$. Once a laminar boundary layer separates, it usually reattaches in the form of a turbulent boundary layer. This is known as the phenomenon of laminar separation bubbles. The code simulates this to a degree in that if the critical λ is reached

then it skips over the transition criteria and immediately starts to calculate the boundary layer according to Head's method. The algorithm may be seen in Figure 4.3.

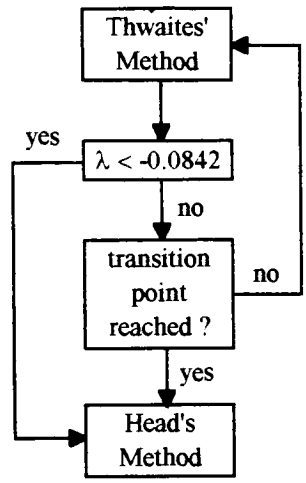


Figure 4.3:
Laminar Separation Algorithm

The treatment of turbulent separation, on the other hand, has no basis. In general, turbulent separation is predicted when H = 2.4 (Reference 12). If this criteria is met, then the conditions from the last, non-separated panel are imposed on the remainder and skin friction calculations are stopped for those separated panels. This, of course, is a very crude method for handling separation. However, while executing the program it was very rarely found that turbulent separation was achieved.

4.2 Code Verification

Regardless of the sophistication, or lack thereof, of any numerical model, there must be some way to test the output before any confidence in the method's validity is warranted. In fact, it would be ideal if there were multiple approaches to verify results. For this reason, some considerable time was spent refining the code.

At one point during the programming, it was thought that the code was at a sufficient level of accuracy because the aerodynamic coefficients were matching fairly closely with published data. However, after close scrutiny, it was found that some bogus assumptions were made with respect to flow conditions. One indicator that the solution was incorrect was the fact that the calculated ideal (inviscid) and real (viscous) lift coefficients were extremely close in magnitude. Contrary to what might be expected, as illustrated in Reference (12), the ideal and real lift coefficients should be substantially different. This is because the viscous update to the inviscid solution has a tendency to corrupt the lift calculations. This is a inherent disadvantage to the method being used. However, the ideal lift, if the method is being executed correctly, should give a very good approximation to the real lift. As it turned out, the drag coefficients were fairly close the published data because the skin friction drag dominated over the form drag since the cases being studied were of fairly low thickness. This example just proves that it was necessary to meticulously examine initial results before accepting the model as valid.

After resolving the mentioned problems and after refining other aspects of the code, the program was executed for three different NACA series airfoils: (1) the NACA 0012 symmetric airfoil, (2) the NACA 4412 cambered airfoil, and (3) the NACA 23021

cambered airfoil. Both the 0012 and the 4412 were analyzed because they were used as start points in the optimization analysis and the 23021 was chosen arbitrarily. Aerodynamic coefficients were obtained for all the test cases for angles of a attack between -10 and 10 degrees, where the analyses were executed in 2 degree increments. Also the Reynolds number for all the runs was six million. The results of the analyses are shown on coefficient plots in Figures 4.4 through 4.6 for the respective airfoils (data plots from Reference 13). Ideal lift coefficients were used for reasons described above, and the corresponding total real drag coefficients were also plotted. Moment coefficients are not shown since they are calculated about the leading edge, not the aerodynamic center.

For the most part, the 0012 data looks decent. The lift curve matches very nicely with the published data. A trend for low drag values in the negative α region is exhibited on the drag coefficient curve. This was not a big concern since the present work only investigates an angle of attack of four degrees. However, this is a definite area that could use improvement. Besides the negative side of the curve, the data is in very close proximity to the published values.

The NACA 4412 analysis was not as successful as the 0012 but the data follows the general trend of the curves. Also, unlike the 0012 analysis, the program seemed to handle the negative side of the curves better than the positive for the 4412. Again decent results were obtained in the vicinity of $\alpha = 4°$.

The last analysis that was performed showed more tendencies toward poor results at the extremes of the α's investigated. Despite this, the results are believed to show an adequate level of accuracy for the problem at hand. The poor results at the high and low

angle of attacks could be due to the lack of sophistication of the separation model, thus, not capturing the decrease in lift and increase in drag expected from poor pressure recovery. Nevertheless, It is believed that the model has a sufficient level of sophistication for the project scope. Improvement of the separation model could be the focus of later studies. It should also be added that, By virtue of the solution method, the results shouldn't be expected to have a high level of accuracy. The focus of the project was to create an efficient solution method for the optimization process and this was satisfied since the calculation of panel methods with an integral boundary layer model is very cheap, computationally speaking. If the coefficients are being improved during the optimization process while satisfying the constraints, then the optimization may be thought to be successful. Indeed, the key consideration is that the analysis model has to be valid but not necessarily precise because other more accurate modeling techniques may be used afterward to better determine flow characteristics. This leads into the next chapter which discusses the CFD modeling of the optimized solutions.

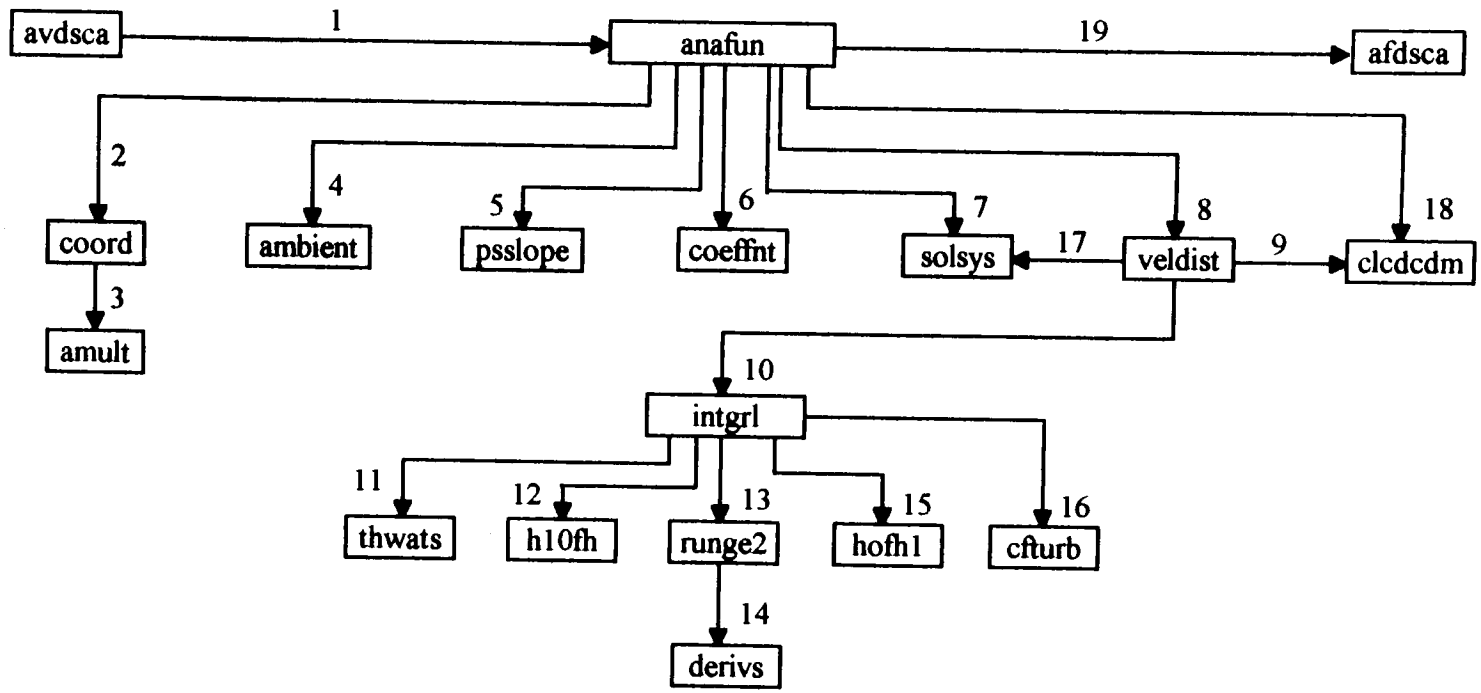
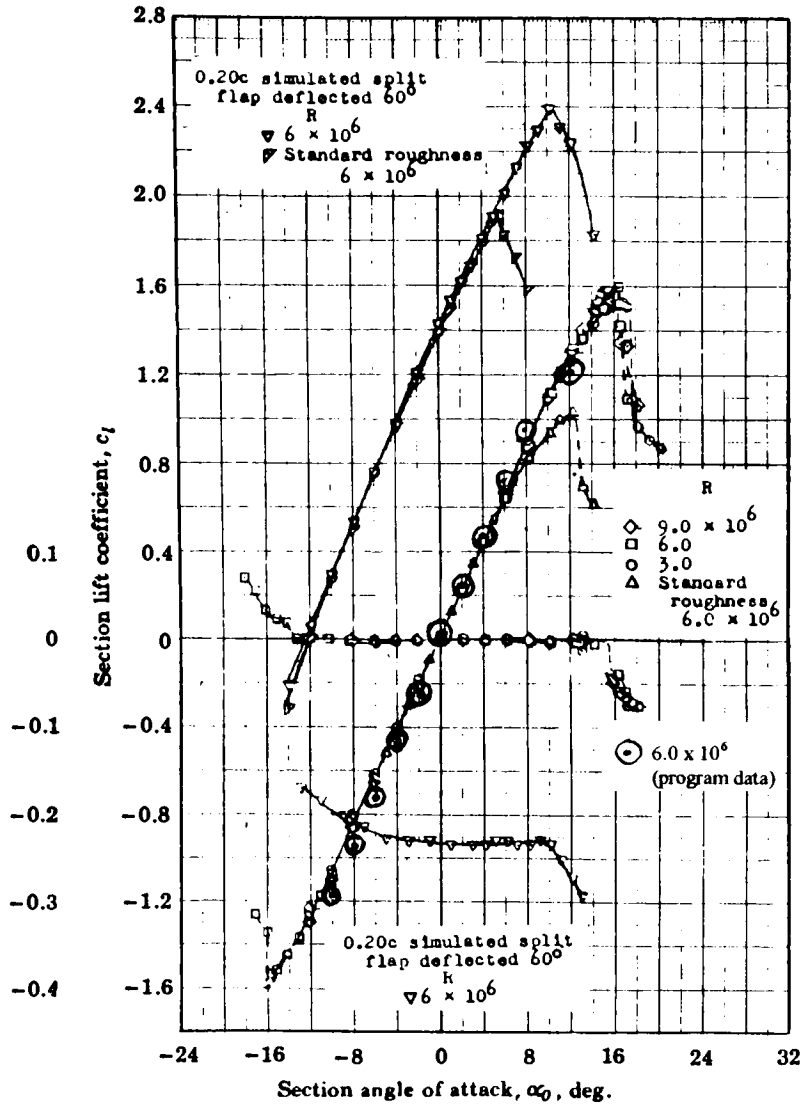


Figure 4.1:
Subroutine Flow Diagram



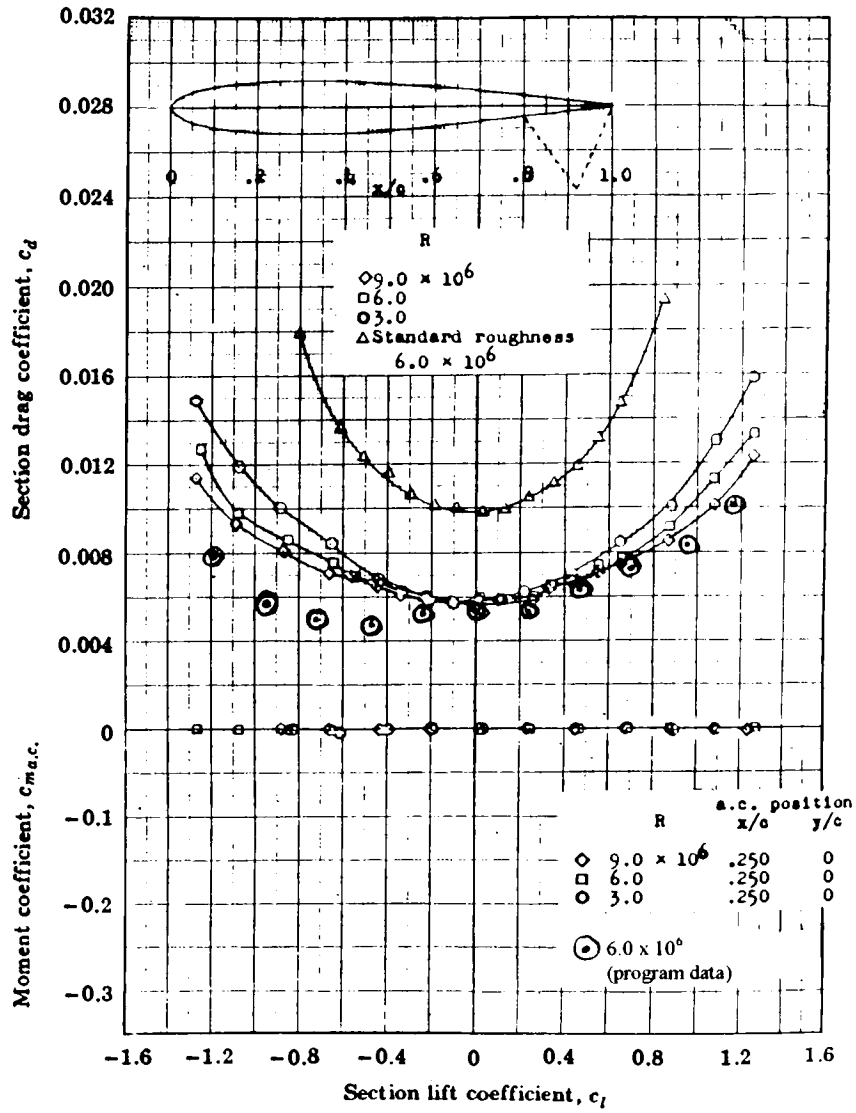
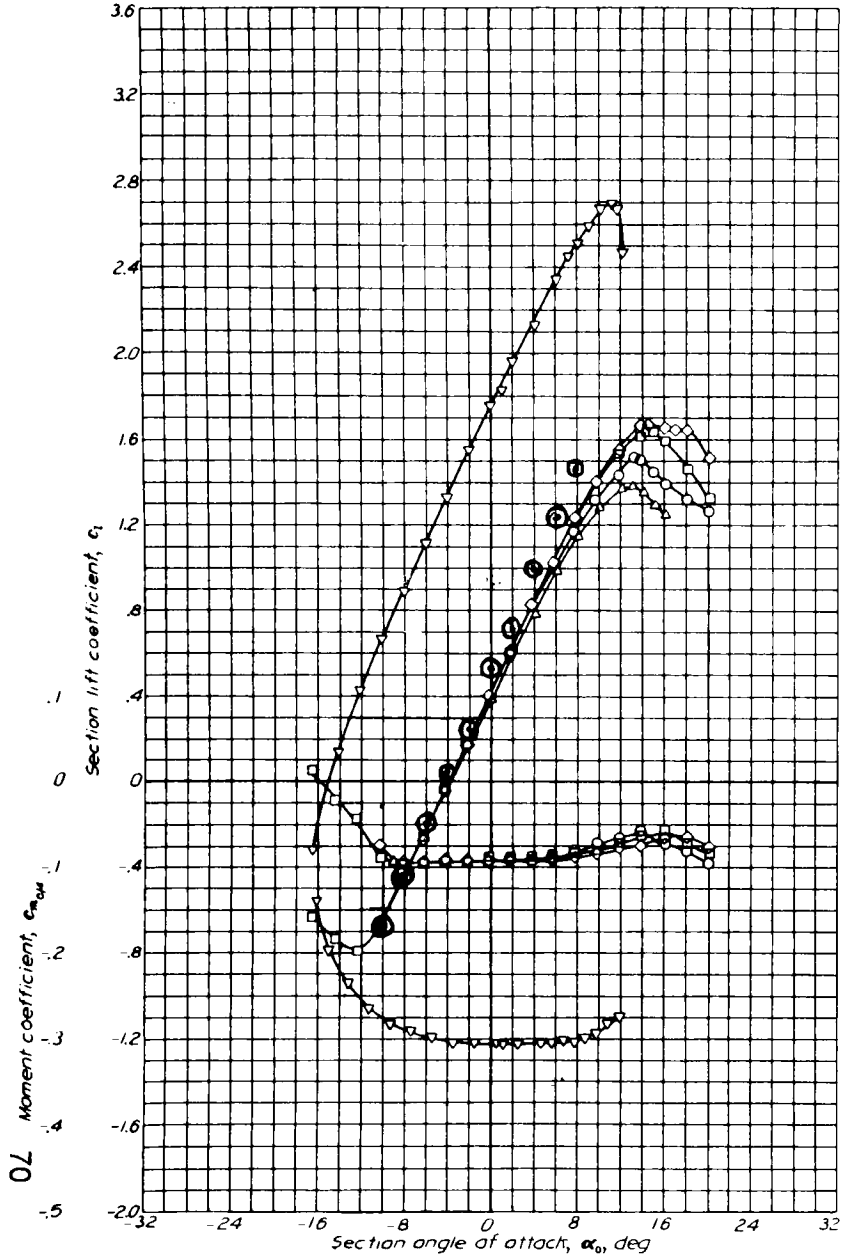


Figure 4.4:
Coefficient Comparison Plots for NACA 0012



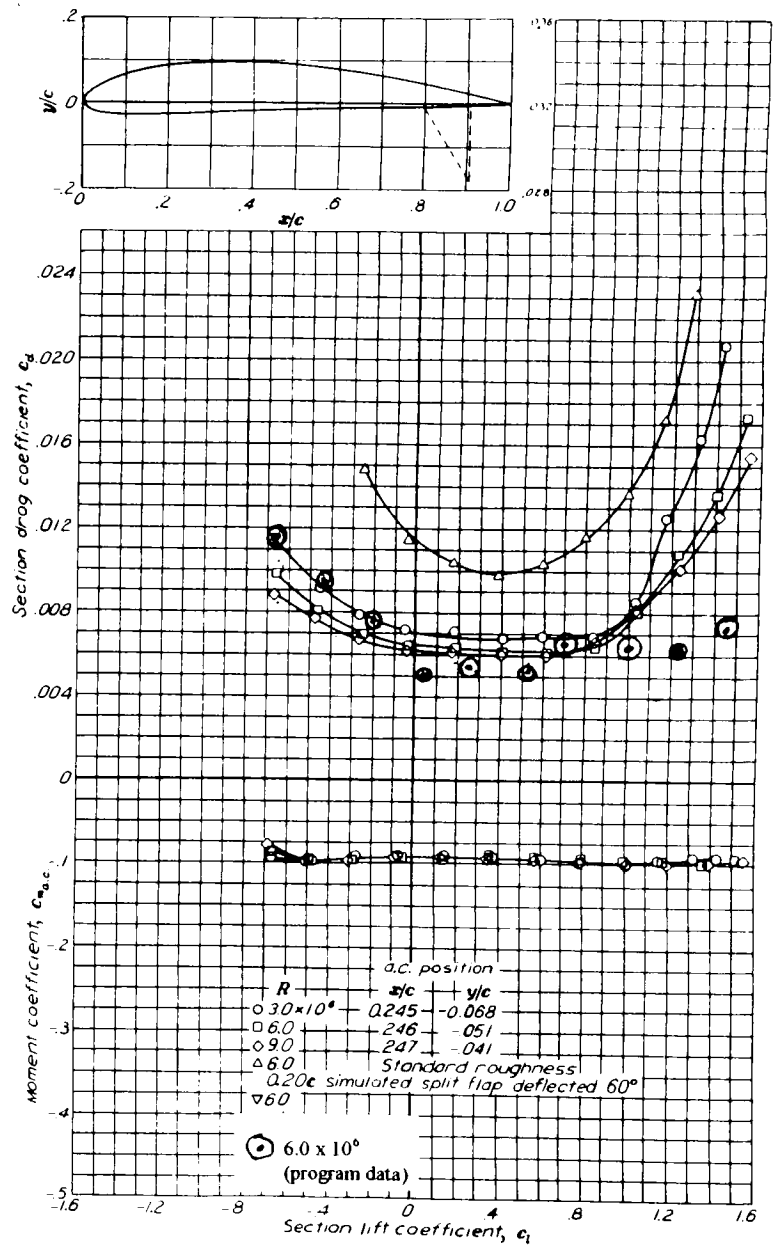


Figure 4.5:
Coefficient Comparison Plots for NACA 4412

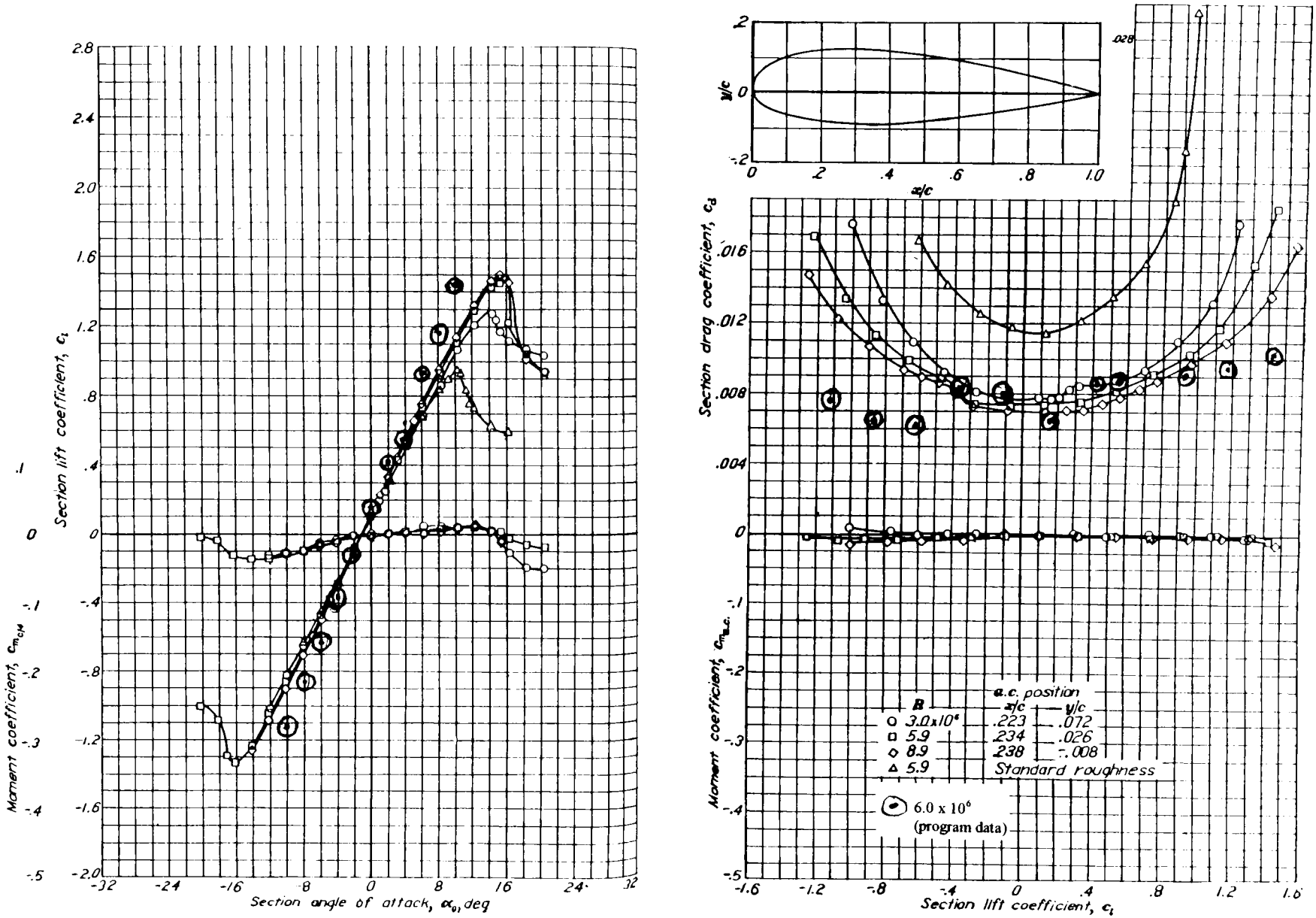


Figure 4.6:
Coefficient Comparison Plots for NACA 23021

5. CFD MODELING OF OPTIMIZED SOLUTIONS

It was addressed last chapter that a more sophisticated solver could be used to further verify the optimization results and to better determine the aerodynamic coefficients for the airfoils. The purpose of this chapter is to describe the process involved in modeling an airfoil using Computational Fluid Dynamics (CFD) for a specific software package - Fluent. The results of the actual runs are presented with the optimization results in Chapter 6.

The software used to model the airfoils was Fluent version 4.23 and was run off of a server with an Ultrix version 4.3 operating system.

Modeling an airfoil in Fluent was not a trivial task. In fact, it took numerous modeling iterations to determine the appropriate conditions for the problem. Most of the modeling time was spent modifying the geometry and grid to acquire reasonable results. Unfortunately, CFD is not yet at a level of sophistication where a user can simply enter a geometry, set the fluid conditions, and run the model. Instead, careful consideration must be given to the each specific step of the problem.

Similar to finite elements, the main steps in CFD modeling are geometry generation, grid generation, and boundary condition setting. Once these steps are performed, a model may be analyzed by the CFD solver. The process by which these steps were executed is presented in the following sections. Also, a NACA 0012 was analyzed to determine the validity of the modeling assumptions. The results of this analysis are also included.

5.1 Geometry and Grid Generation

Before beginning the modeling process, a geometry and a corresponding grid need to be created. This is by far the most difficult part of CFD modeling and should be given careful consideration before starting.

Fluent has a preprocessor called PreBFC where both the geometry and grid are created. It is a CAD-like program that allows a geometry to be created through the use of points, lines, curves, and so on. The grid generator has the option of creating a Cartesian, an axisymmetric, or a body fitted grid. Being most appropriate, a body fitted grid was used for the airfoil problem.

It is a good idea to first sketch out a geometry before it is actually generated. Careful consideration should be given to how the grid will be mapped to this geometry. The strategy determined for the airfoil problem may be seen in Fig. 5.1. The drawing shows an airfoil centered in a circle (the circle is not to scale) with a cyclic boundary connecting the two entities. Actually, there are two cyclic boundaries, CYC and CYCP, overlapping each other. The geometry was created as shown to accommodate an O-type grid which can be thought of as a rectangular grid that is wrapped around the airfoil such that the two opposing sides become overlapped. These opposing sides of the grid are the cyclic boundaries which impose the flow conditions from one face onto the other. That is, flow property continuity is imposed across the faces. The geometry's corresponding rectangular computational grid mapping sketch may be seen in Fig 5.2. It is important to make these two sketches at the same time because the creation of one depends on the

other. In both figures, the dashed lines indicate that the corresponding endponits are to be mapped at the same I^{th} grid index.

With that being said, the geometry may be created as shown in Fig. 5.1. The airfoil surface was created by connecting data points generated from the optimization program with a spline (actually, a cubic Bezier spline similar to those used in the original geometry definition). After creation, the surface was broken into eight subsections with each subsection being assigned a different zone number. Zone numbers are used by Fluent to keep track of data for the respective sections of interest. As will be shown later, the segments were given different zone numbers so that wall force data could be obtained for each subsection. The wall force data can then be used to calculate the aerodynamic coefficients. Just one zone number could have been used for the airfoil surface but this would have resulted in wall forces being calculated for the whole skin at once. This is undesirable because the accuracy of the wall force calculation would suffer; that is, the wall forces would be more accurate if many smaller sections were used. After varying the number of subsections, it was determined that having eight segments yielded good results. Also, the outer bound was broken into subsections that correspond to the individual airfoil segments. These boundary segments, in turn, served to define the flow inlets and outlets. Since only an angle of attack of four degrees was investigated, the inlets may be defined as subcurves L3 through L7, and the outlets defined as L8, L1, and L2.

One aspect of the geometry that affected the accuracy of the solution was the outer boundary radius. If the boundary radius is too small, then it could have an adverse affect on the flow conditions in the vicinity of the airfoil, hence, rendering a solution to be

invalid. One indicator that a solution had been influenced by the boundary was that, when looking at pressure contours for the solved model, many erratic isobars could be seen emanating from the airfoil to the boundary. To prevent this from happening, it was necessary to make many initial runs, increasing the boundary radius until it was found that the pressure contours looked normal. It was decided that a radius of fifteen times the chord length (which was equal to one) was sufficient to prevent the boundary from affecting the solution.

The final geometry for a sample airfoil (NACA 0012) may be seen in Figs. 5.3a and 5.3b where Fig. 5.3a shows the whole geometry and Fig. 5.3b shows a close-up view of the airfoil

The next step after the geometry had been created was to map the grid. A 220 x 70 cell grid was used for the airfoil problem. These dimensions were arrived at by, again, making multiple iterations to test the affect of different grid sizes on the solution. The mapping was performed according to the drawing shown in Fig 5.2.

Since the flow field is so large compared to the size of the airfoil, it is desirable that the grid be weighted so that there would be a higher grid density towards the airfoil surface. Otherwise, for the grid size chosen, there would not be enough resolution in the vicinity of the airfoil surface to provide any degree of accuracy. Therefore, grid weighting for this problem serves two purposes: (1) it provides grid resolution near the airfoil surface, and (2) it lessens the grid size requirements. The grid weighting parameters were adjusted through multiple trials and it was found that specifying a weighting factor of 70 on the endpoints P1 and P1P of curves CYC and CYCP, respectfully, yielded the best

results. This weighting factor is higher than what would typically be used but it seemed reasonable in light of the large boundary size. The weighted grid may be seen in Figs. 5.4a and 5.4b

Weighting was also performed for curves B1 and T1 at the endpoint P5. This turned out to be very important for the solution of the problem. If the weighting was not performed then the grid aspect ratios were too high near the stagnation point. Aspect ratios are recommended to be as close to one as possible in regions of high gradients and should not exceed five in any case. As is the case for the present model, high aspect ratios can not always be avoided but they should be minimized. So, since the gradients in the vicinity of the leading edge can be very high, the grid needed to be weighted to minimize the aspect ratios of the surrounding cells. A close-up view of the grid around the leading edge may be seen in Fig. 5.5. Cell aspect ratios were definitely lessened to a large degree as a result of the weighting procedure. However, it would have been better if the aspect ratio could have been reduced for all the cells around the airfoil. This could have potentially been done by increasing the grid density but the results obtained from this setup were fairly accurate.

One other problem that needed to be resolved was that for cambered airfoils with downward sloping trailing edges, there were problems with the grid validity. The problem was that the grid would map incorrectly at the intersection of the trailing edge and the cyclic boundaries. To correct the problem, the cyclic boundary needed to be angled such that it extended from the trailing edge at a slope that bisected the angle defined by the top and bottom surfaces near the trailing edge. This is illustrated by Figs. 5.6a and 5.6b.

Subsequently, the grid mapping coordinates needed to be modified and are shown in Fig. 5.7. Incidentally, having the cyclic boundary as shown helps the solver satisfy the Kutta condition at the trailing edge since it has difficulties with grids that are oriented at awkward angles with respect to the flow streamlines.

5.2 The Fluent Solver

After the grid has been completed it can be brought into the Fluent solver where boundary conditions are specified. Flow parameters such as density and viscosity are specified as well. Also, if special models need to be utilized, they are turned on in the solver module of Fluent. Since the analysis model for the optimization used dimensionless variables, it became necessary to quantify a suitable flow speed. This can be done easily since the Reynolds number was held constant at six million throughout the optimization runs. Reynolds number is defined as

$$R_e = \frac{\rho V_\infty c}{\mu} \tag{5.1}$$

Choosing ambient conditions for density (ρ = 1.225 kg/m^3) and viscosity (μ= 1.79 x 10^{-5} Ns/m^2) , and choosing a unit chord (c = 1 m) yields

$$V_\infty = \frac{\mu R_e}{\rho c} = \frac{(1.79\text{x}10^{-5} Ns/m^2)(6\text{x}10^6)}{(1.225 kg/m^3)(1m)} = 87.7 m/s \tag{5.2}$$

A renormalized group theory (RNG) turbulence model was used in all the problem solutions. It was recommended in the Fluent manual (Reference 15) that the RNG turbulence model be used for airfoil problems where separation may be encountered. Although the RNG model was used for all models, a trial run was made with the default k-ε turbulence model and there was negligible difference. This was probably due to the fact that the problem being considered did not have any flow separation.

Default underrelaxation parameters were used after trying to solve the same problem from many different combinations. Either the solutions tended toward divergence for increased values or iterations progressed too slowly. Therefore it was decided to use

the default values. Typically, the underrelaxation parameters can be increased to expedite the convergence of a solution. For the solution of the first airfoil problem, a run time of about 12 hours was required to reach convergence. However, for subsequent models, solution from previous models could be used as analysis starting points. This drastically reduced the run time to about 1.5 hours.

A validation case was run using the described modeling and is presented in the following section.

5.3 Fluent Model Validation With the NACA 0012 Airfoil

As was the case with the analysis model, the CFD model was run for a well known airfoil for which there is published data. This should be done for all types of CFD models since the solutions can be thrown off dramatically for small changes in modeling setup. So, the NACA 0012 was chosen as a test case and was modeled as described above. After solution, wall force data was acquired through postprocessing and is as follows:

```
WALL FORCES BY ZONE :
---------------------
UNITS = NEWTONS
```

WALL ZONE	NORMAL FORCES X-DIR.	NORMAL FORCES Y-DIR.	SHEAR FORCES X-DIR.	SHEAR FORCES Y-DIR.
W1	0.000E+00	0.000E+00	0.000E+00	0.000E+00
W3	2.526E+01	1.879E+02	1.308E+00	-1.625E-01
W4	4.193E+01	4.985E+02	2.316E+00	-1.882E-01
W5	1.704E+01	8.506E+02	3.321E+00	-6.906E-02
W6	-3.848E+02	1.145E+03	7.093E+00	1.549E+00
W7	1.109E+02	1.259E+02	4.306E+00	-4.093E-01
W8	5.093E+00	-2.227E+02	4.589E+00	1.025E-01
W9	1.753E+01	-2.335E+02	3.919E+00	3.225E-01
WA	1.625E+01	-1.668E+02	3.219E+00	4.061E-01

Table 5.1
Wall Force Data

The total forces in the respective directions may be added and then the sums may be resolved onto the L and D axis defined by the four degree angle of attack. After finding the total drag and lift forces, their respective coefficients can be calculated (see Appendix II for sample calculations of c_D and c_L). The following table compares the Fluent results with published data for the NACA 0012 (Reference (13)):

	Fluent	Published	% Difference
c_L	0.457	0.44	3.9
c_D	0.00669	0.0067	0.14

Table 5.2
Comparison of Validation Results

Both values of the aerodynamic coefficients are in good agreement to the published data. It can, therefore, be concluded that the model developed is valid for calculating the flow about airfoils.

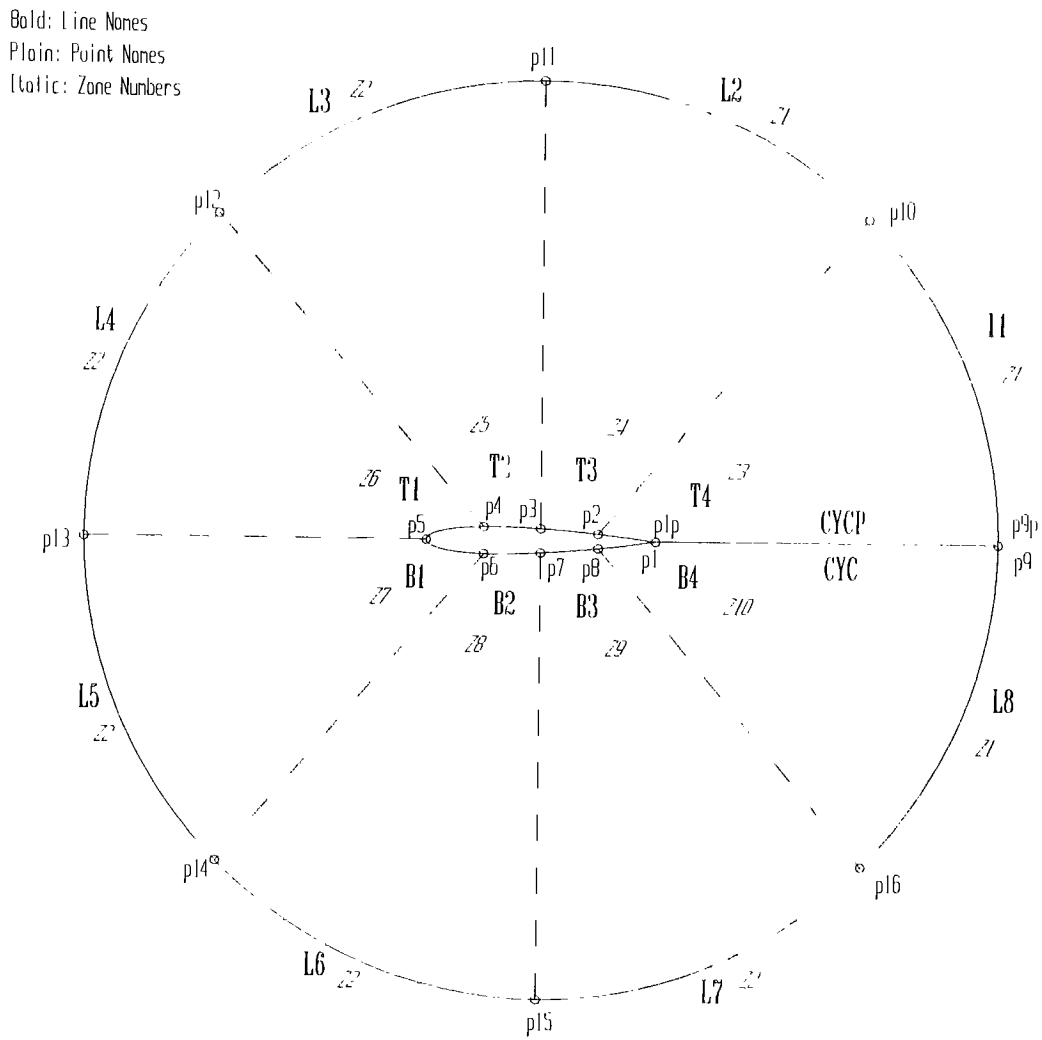


Figure 5.1:
Geometry Strategy

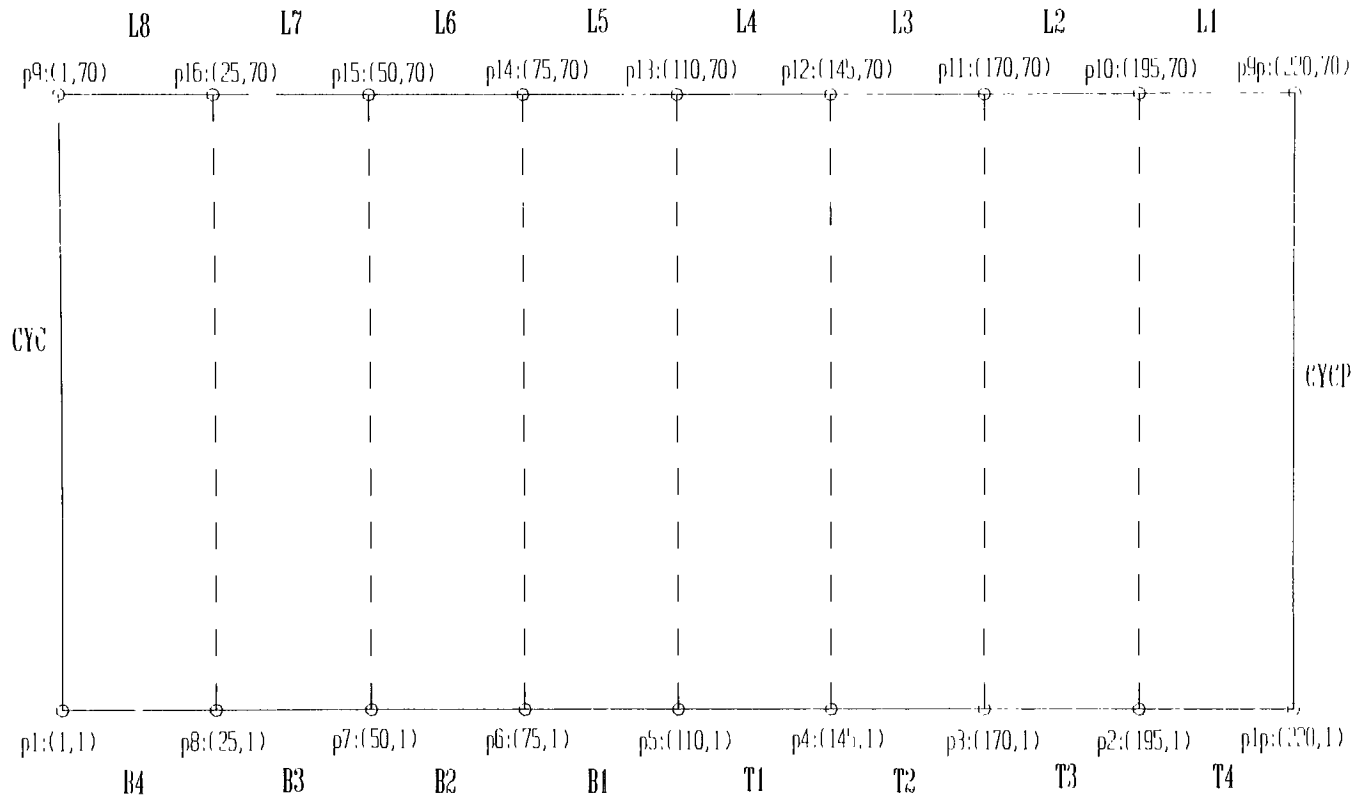


Figure 5.2:
Grid Mapping Strategy

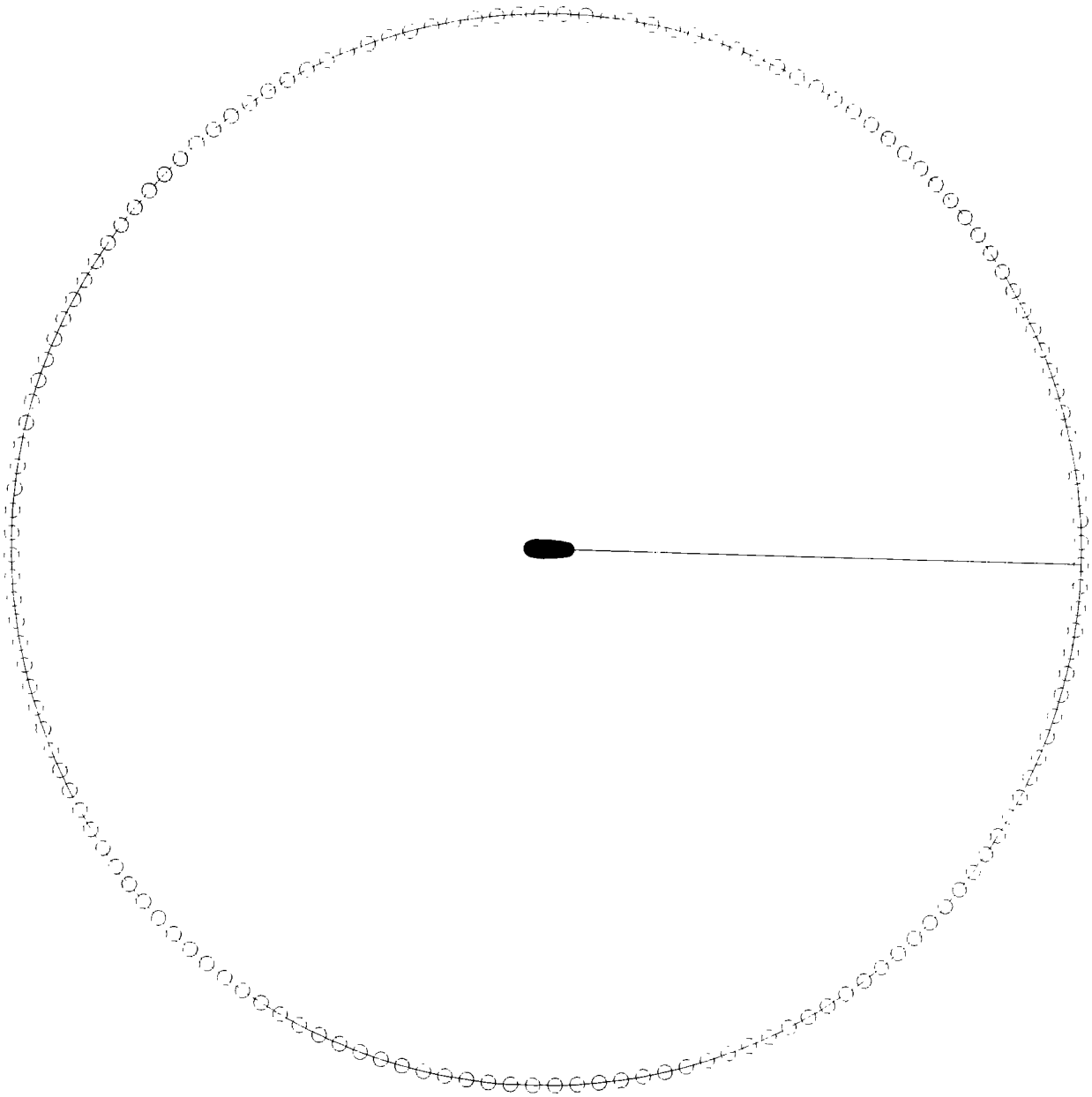


Figure 5.3a:
Sample Geometry

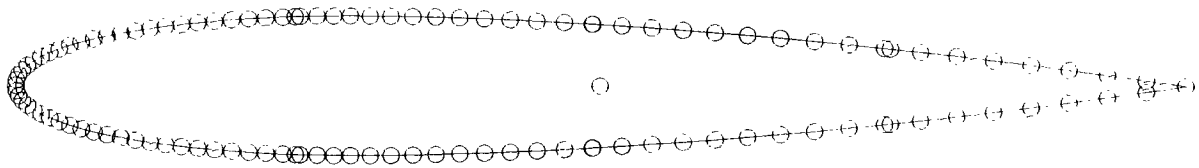


Figure 5.3b:
Close-up of Sample Geometry

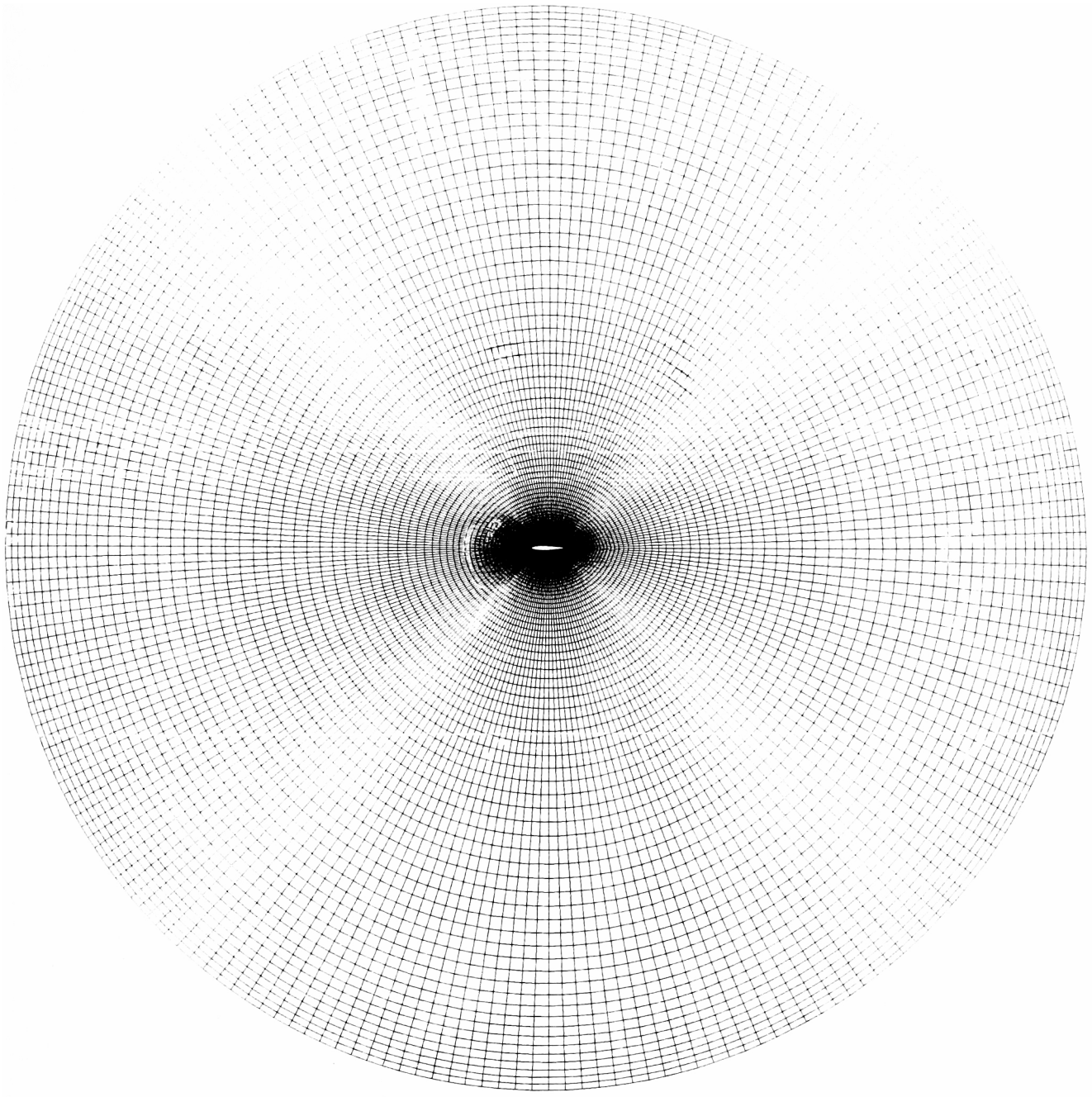


Figure 5.4a:
Weighted Grid for Sample Geometry

Figure 5.4b:
Close-up of Weighted Grid for Sample Geometry

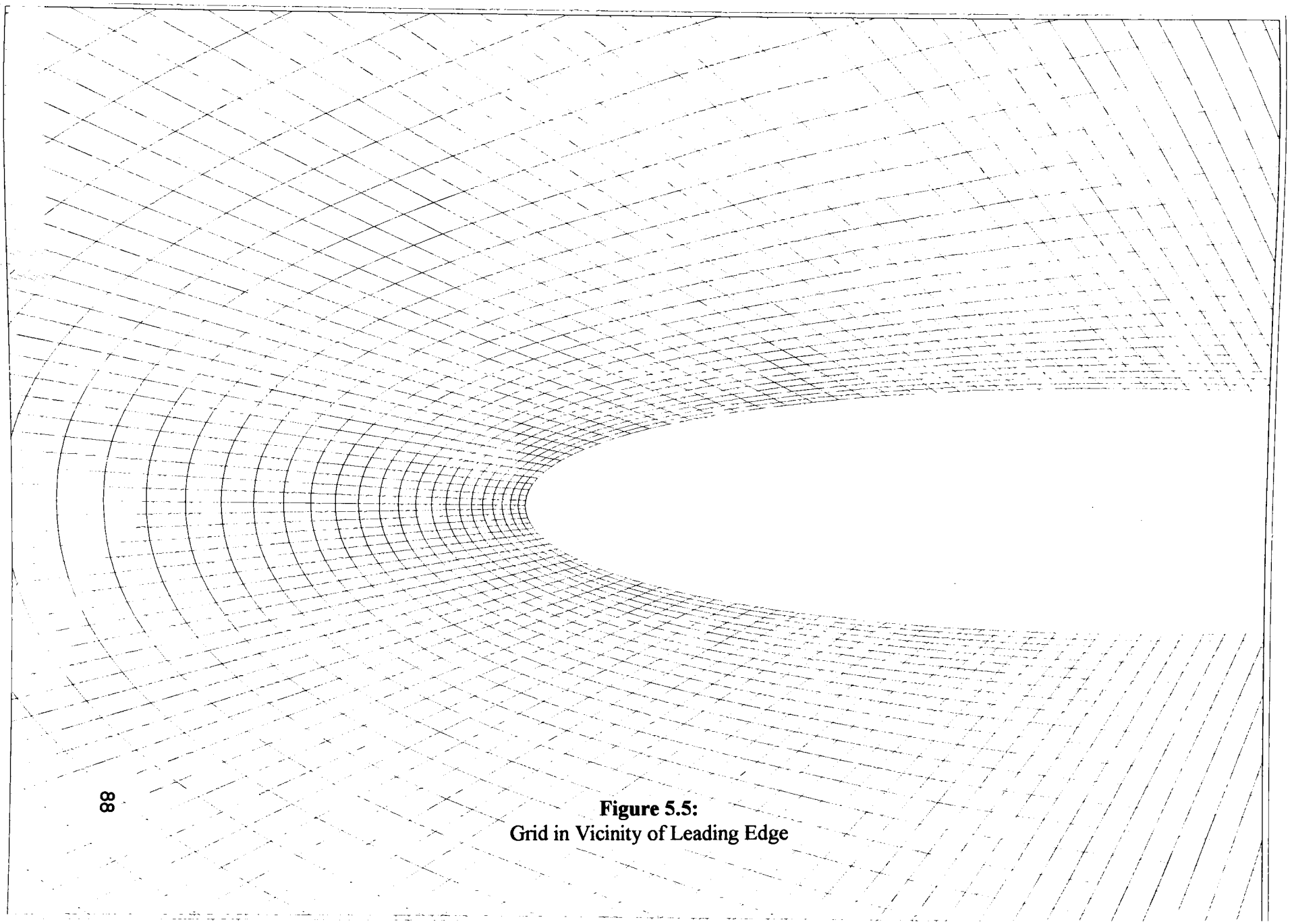


Figure 5.5:
Grid in Vicinity of Leading Edge

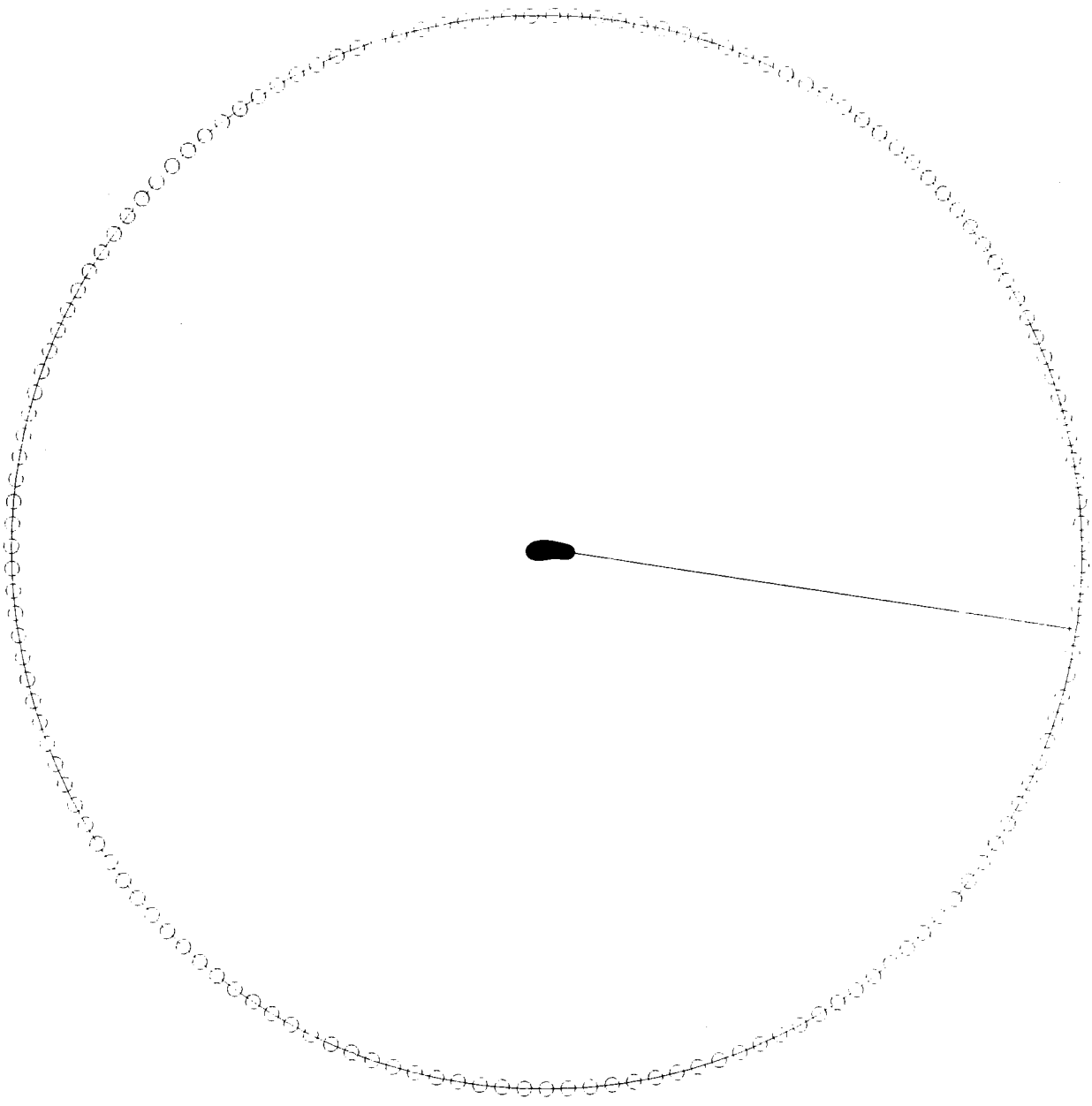


Figure 5.6a:
Geometry for Cambered Airfoil

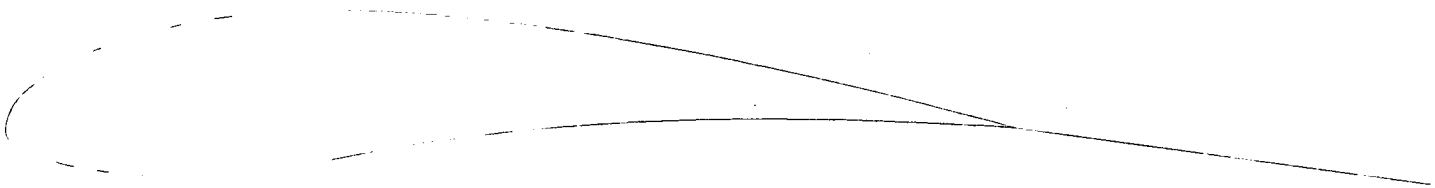


Figure 5.6b:
Close-up of Geometry for Cambered Airfoil

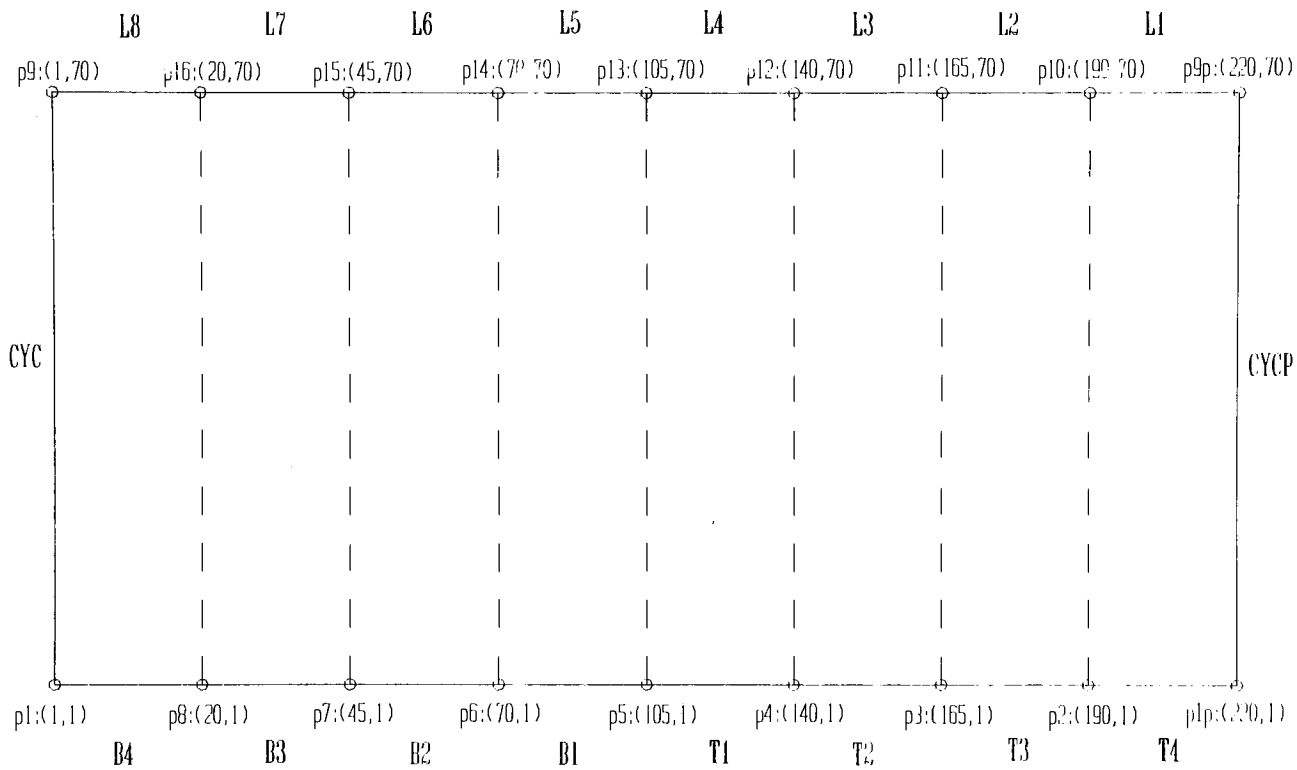


Figure 5.7:
Grid Mapping Coordinates for Cambered Airfoil

6. OPTIMIZATION RESULTS

As previously mentioned, OptdesX optimization software was used to run the optimization problems. The software was run from a DEC server loaded with an OSF/1 version 3.2 operating system.

Each problem case was analyzed by stopping after every iteration to save results. Consequently, actual run times are not available but every iteration was on the order of 3-4 minutes with almost all of the time spent in the analysis model. Approximately twenty-five analysis calls and 1 gradient call were made per iteration. Also, on average, optimization runs took about five or six iterations to converge.

In the following sections, the problem cases that were investigated are defined and the corresponding results are presented. Also, Fluent comparisons to selected optimization solutions are presented.

6.1 Problems Considered

As in most airfoil optimization methods, the aerodynamic coefficients are of special interest. For this reason, they were incorporated into the analysis model and through using OptdesX, their defining functions can be specified as either an objective (function to be optimized) or a constraint (function that limits feasible design space). The problems to be considered are defined as follows:

Problem A: Max c_L, unconstrained c_D and c_M

Problem B: Max c_L, $c_D \leq 0.007$, unconstrained c_M

Problem C: Max c_L, $c_D \leq 0.007$, $c_M \leq -0.15$

Problem D: Min c_D, unconstrained c_L and c_M

Problem E: Min c_D, $c_L \geq 0.7$, unconstrained c_M

Problem F: Min c_D, $c_L \geq 0.7$, $c_M \leq -0.15$

where all coefficients are the viscous, or real, versions. These were optimized and constrained instead of the ideal coefficients because the real coefficients capture the influence of the boundary layer, whereas the ideal coefficients do not.

Also, it has been presented that two distinct airfoil definition schemes were to be investigated. The first, scheme 1, requires the three Bezier vertices B_8, B_9, and B_{10} to define a horizontal line. The second definition approach, scheme 2, relaxes the requirement on the same three points in that they have to lie on a line without any constraint imposed on the slope of the line.

The last point of investigation are the different starting points to be considered. It has been mentioned that the NACA 0012 and the NACA 4412 airfoils would be used as different starting points to the optimization problems.

To be consistent in solution procedure, the side constraints on all design variables and the limits of all constraints were held fixed throughout the optimization trials. These limits for both definition schemes and starting points may be seen in Appendix III. Also, the same hybrid GRG-SQP method was used to optimize all the airfoil shapes. Furthermore, the same gradient perturbation of 0.001 was also held constant. This allowed for a basis of comparison between the solutions.

This investigation outline totals to twenty-four separate problems for consideration. The following section presents the results for each case. Also, if warranted, comparisons between cases are made.

6.2 Problem Results

The results from the various optimization solutions are presented here. They are addressed in the order defined in the last section. Plots of the progression of the airfoil shapes at each iteration are presented. Some iterations may be omitted for clarity. Tabular data of the aerodynamic coefficients for each trial is presented along with the plots. Also, Fluent modeling results from selected problems are presented and compared at the end of the section.

Problem A: Max c_L, unconstrained c_D and c_M

NACA 0012 Start Point:

Scheme 1:

From Fig. 6.1 the potential of the method is apparent. The solution airfoil looks nothing like the original. The flat portion on the bottom surface is characteristic of the horizontal line definition scheme. Also, many characteristics of a high lift airfoil can be seen. For example, the surface distance on the top surface increased greatly while the opposite happened for the bottom surface. This would result in a greater pressure differential between the two surfaces, hence, causing greater lift. Another high lift characteristic that the airfoil possesses is camber. As seen in Table 6.1 the optimization increased the viscous lift coefficient by 260 %. But, as would be expected for an unconstrained c_D problem, the drag coefficient increased by 101 %.

Scheme 2:

Again, similar characteristics were exhibited by the problem solution shown in Fig. 6.2. Although, the bottom front surface is nicely rounded opposed to the flat surface seen in Fig 6.1. Again a very high increase in lift may be seen in Table 6.2 (306 %). A comparison of the two solutions from Figs. 6.1 and 6.2 may be seen in Fig. 6.3. Although, the two airfoils were started from the same point they turned out fairly different, but some similar tendencies may also be seen .

NACA 4412 Start Point:

Scheme 1:

As may be seen in Fig. 6.4, some problems were encountered in that the solution had a kink on the lower surface. This is an unacceptable airfoil because slope continuity is usually required in an airfoil. The kink occurred because the x-coordinates of B_8, B_9, and B_{10} were all very close together. The only way to fix this was to modify the limits of the constraints x9-x10, and x8-x9. They were set to be greater than 0.05. The modified shape may be seen in Fig. 6.5. It tended towards the same shape as the previous but the kink disappeared. For this airfoil solution, the lift was increased by 75% (see Table 6.3).

Scheme 2:

This solution, seen in Fig. 6.6, attained the highest lift out of all the maximum c_L problems. One can notice from Table 6.4 that the lift

increased by 84 %. So, from an aerodynamic standpoint, this was the best section achieved, but from an optimization standpoint, the NACA 0012 scheme 2 solution was the best with a 306 % increase in the objective. Also, A comparison plot between this and the previous solution may be seen in Fig. 6.7. The two solutions were almost identical on the top surface but very different on the bottom, most likely because of the different definition schemes. Another observation that may be made is that the solutions from the different start points were very dissimilar.

Problem B: Max c_L, $c_D \leq 0.007$, unconstrained c_M

NACA 0012 Start Point:

Scheme 1:

Figure 6.8 shows a solution with similar characteristics to the first problem considered in that it has a flat bottom-front surface. But notice that the point of maximum thickness on the top surface has decreased. This is due to the active c_D constraint. Table 6.5 shows that the lift was increased by 168 % which is considerably less than the first problem but is still quite good.

Scheme 2:

Again, a decrease in thickness is shown when comparing this solution (Fig. 6.9) to its unconstrained counterpart. The c_D constraint was again active and, as shown in Table 6.6, the lift was increased by 219 %. This is still a very good solution considering its start point. The two solutions from Figs.

6.8 and 6.9 are compared in Fig. 6.10. They are very close in shape except for the bottom-front. Regardless of this difference, the aerodynamic coefficients were almost identical. Both of these solutions were run through Fluent and were verified to be close in aerodynamic characteristics (see the end of the section for Fluent comparison).

NACA 4412 Start Point:

Scheme 1:

Similar to what was seen before, there is a kink in the solution of Fig. 6.11. Again, the limits were adjusted to be greater than 0.05 yielding the shape in Fig 6.12. For this section, the constraint was slightly inactive and the resulting lift was only increased by 5.1 % (see Table 6.7). Still, the lift was comparable to the previous two solutions.

Scheme 2:

Again, this combination produced the highest lift even though the increase was low (20.8 %) according to Table 6.8. The solution may be seen in Fig. 6.13 and the comparison between this and the last solution may be seen in Fig. 6.14.

Problem C: Max c_L, $c_D \leq 0.007$, $c_M \leq -0.15$

NACA 0012 Start Point:

Scheme 1:

Unfortunately, for all the constrained c_M problems, the moment coefficient had little to no affect on the solution since the constraints were only active,

if at all, for the first one or two iterations. For this reason the figures for these airfoils will be presented but numerical results will not be discussed. See Figure 6.15 for the results of this airfoil. There actually were some significant differences between this airfoil and its corresponding constrained c_D problem.

Scheme 2:

The solution to this problem, seen in Fig. 6.16, was very similar to the its constrained c_D counterpart.

NACA 4412 Start Point:

Scheme 1:

As would be expected, the kink was again present in the solution shown in Fig. 6.17. However, no attempt to modify the limits was made because it would almost certainly be the same as the constrained c_D solution.

Scheme 2:

Another identical solution is shown in Fig. 6.18. Both solutions to the NACA 4412 start point problem never had the c_M constraint active, but the NACA 0012 trials had c_M active for the first iteration, hence, sending the solutions in slightly different directions.

Problem D: Min c_D, unconstrained c_L and c_M

NACA 0012 Start Point

Scheme 1:

The solution shown in Fig. 6.19 is quite different from those seen in the Max c_L problem solutions. The NACA 0012 start point airfoils for this set of Min c_D problems had difficulties initially moving from the 0012 airfoil. In fact, the solutions often needed to be physically perturbed to get them to move. This was done by moving one vertex at a time until the solution moved. It usually only took a single perturbation to move the solution. As shown in Table 6.9, the drag was reduced by 33.1 % which is very good.

Scheme 2:

A similar shape to Fig. 6.19 is shown in Fig. 6.20. The drag for this shape was reduced by 28.5 % (see Table 6.10). It is worth noting that this and most of the other shapes for the Min c_D problems have very low form drag. This can be seen by noticing that the total drag to the skin friction drag are almost equal. A comparison of Figs. 6.19 and 6.20 is shown in Fig. 6.21 where it can be seen that the two shapes are very similar.

NACA 4412 Start Point:

Scheme 1:

It can be seen from the solution in Fig. 6.22 that there were still tendencies for kinks on the bottom surface but the problem resolved itself by the time a solution was found. The final shape is thinner than the original which could be expected from a minimum drag problem. The drag for this section was reduced by an impressive 45.7 % (see Table 6.11). Not only did this

airfoil have the largest reduction in drag but it also had the lowest value for c_D.

Scheme 2:

The shape determined for this solution was similar to the previous and is shown in Fig. 6.23. A comparison of the two is shown in Fig. 6.24. Also, as indicated in Table 6.12, the drag for this airfoil was reduced by 27.4 %.

Problem E: Min c_D, $c_L \geq 0.7$, unconstrained c_M

NACA 0012 Start Point:

Scheme 1:

With the addition of the c_L constraint, the solution changed quite a bit. The solution may be seen in Figure 6.25. For this shape, the drag was reduced by 19.6 % (see Table 6.13).

Scheme 2:

The shape changed significantly for this solution as well. Refer to Fig. 6.26 for a plot of the results. As indicated in Table 6.14, the drag was reduced by 13.2 %. A comparison plot for this and the last solution is shown in Fig. 6.27 Again, the shapes were fairly close.

NACA 4412 Start Point:

Scheme 1:

Once again, the solution for this combination tended to kink as shown in Fig. 6.28. The overall drag reduction was 26.3 % as indicated in Table 6.15. This solution had the lowest drag as it did in the previous problem.

Scheme 2:

The solution for this case was fairly similar to the corresponding unconstrained c_D problem and may be seen in Figure 6.29. The drag was reduced by 23.9 % (see Table 6.16). a comparison of this and the last solution is shown in Fig. 6.30.

Problem F: Min c_D, $c_L \geq 0.7$, $c_M \leq -0.15$

NACA 0012 Start Point;

Scheme 1:

Like the previous constrained c_M problems, this and the other forthcoming solutions did not deviate much, if at all, from the constrained c_D problems. Again, only the plots are presented. The present solution may be seen in Fig. 6.31.

Scheme 2;

The solution for this airfoil is presented in Fig. 6.32.

NACA 4412 Start Point:

Scheme 1;

Figure 6.33 shows the solution for this airfoil.

Scheme 2:

Refer to Fig. 6.34 for the solution to this airfoil

Now that the results have been presented, some general observations may be made. As was the case, it could be expected that scheme 2 would produce the highest lift

airfoils since it allows for higher camber. Furthermore, it seems logical that the 4412 start point would produce the highest lift since it starts with more lift than the 0012. A similar case could be made for the minimum drag airfoils. The horizontal definition scheme lends itself better to low drag shapes. Since the airfoils started at approximately the same c_D, it makes sense that either start point could yield the lowest drag.

It was hoped that the present method would allow the airfoils to converge upon some global optimum, this, however, was not the case. The nonlinearity of the analysis model was too great which made the solutions settle into local optima. Many pairs of the cases that were started from the same point converged upon very similar solutions which indicates that the optima are a strong function of the starting airfoil shape.

As previously mentioned, Fluent runs were made to verify the results of selected optimized airfoils. The results are as follows (% difference calculated based on Fluent results):

1) Unconstrained Max c_L, 4412 Start Point, Scheme 2:

	Fluent	Analysis Model	% Difference
c_D	0.0117	0.0105	10.25
c_L	1.71	1.822	6.1

2) Max c_L with $c_D \leq 0.007$, 0012 Start Point, Scheme 2:

	Fluent	Analysis Model	% Difference
c_D	0.00773	0.00700	9.5
c_L	1.027	1.101	7.2

3) Max c_L with $c_D \leq 0.007$, 0012 Start Point, Scheme 1:

	Fluent	Analysis Model	% Difference
c_D	0.00764	0.00700	8.4
c_L	1.037	1.101	6.1

These two (2 and 3) were both run to see if they were actually as close in aerodynamic coefficients as predicted. As can be seen, they were fairly close which is somewhat odd because the shapes are dramatically different in the vicinity of the bottom-front segment.

4) Max c_L with $c_D \leq 0.007$ and $c_M \leq -0.15$, 0012 Start Point, Scheme 2:

	Fluent	Analysis Model	% Difference
c_D	0.00889	0.00700	21.3
c_L	1.052	1.095	4.1

5) Unconstrained Min c_D, 0012 Start Point, Scheme 2:

	Fluent	Analysis Model	% Difference
c_D	0.00421	0.00402	4.4
c_L	0.742	0.670	9.7

6) Min c_D with $c_L \geq 0.7$, 4412 Start Point, Scheme 1:

	Fluent	Analysis Model	% Difference
c_D	0.00430	0.00397	7.7
c_L	0.865	0.793	8.3

7) Min c_D with $c_L \geq 0.7$ and $c_M \leq -0.15$, 0012 Start Point, Scheme 2:

	Fluent	Analysis Model	% Difference
c_D	0.00570	0.00520	8.7
c_L	0.925	0.861	6.9

Except for the c_D value in number (4), the optimization results were within a reasonable amount of error. One must keep in mind that the focus of the program was

more to have a valid mathematical model that would be efficient for the optimization process. Because of the lack of sophistication of the analysis program, the data is actually better than expected.

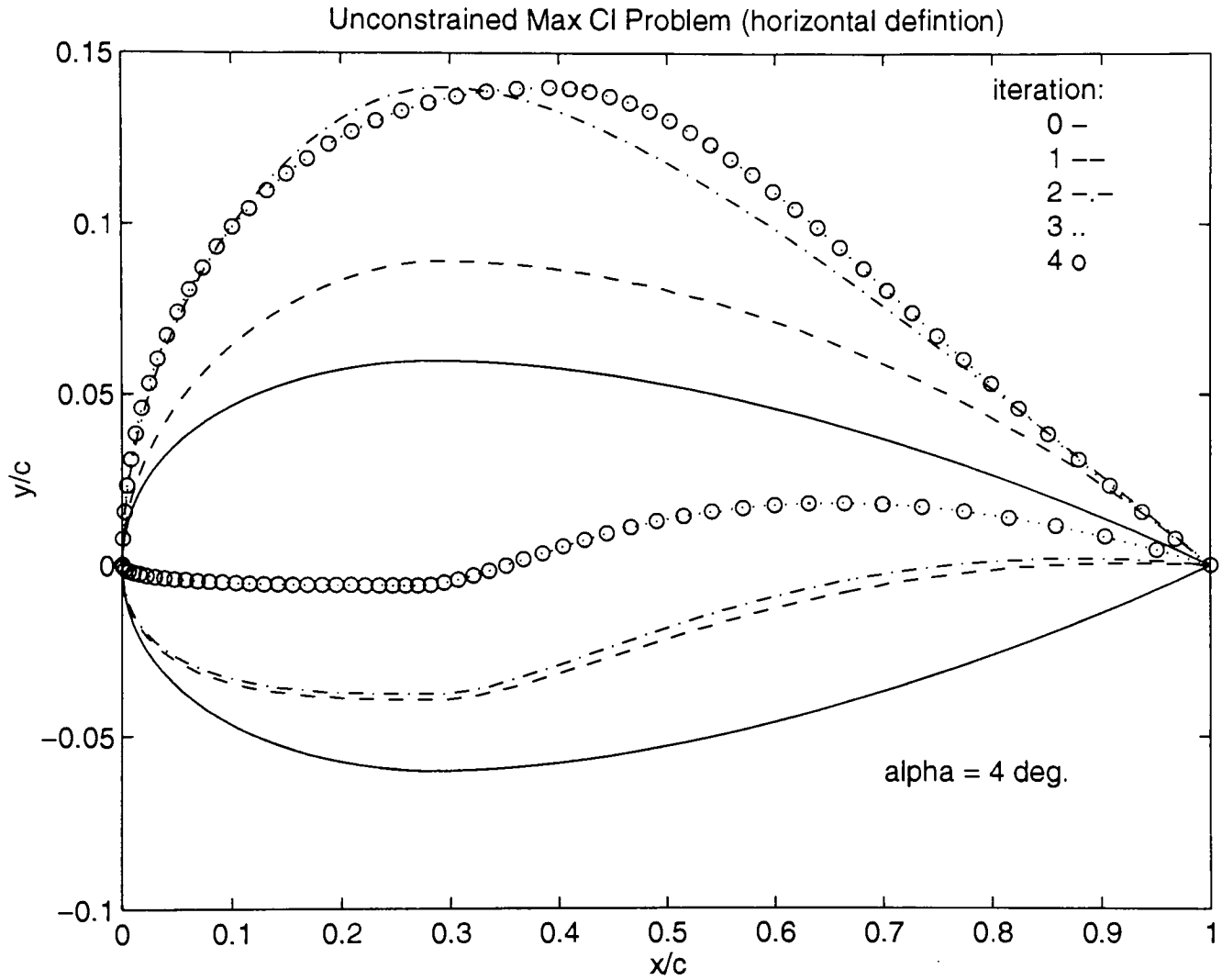


Figure 6.1:
Problem A: NACA 0012 Start Point, Scheme 1

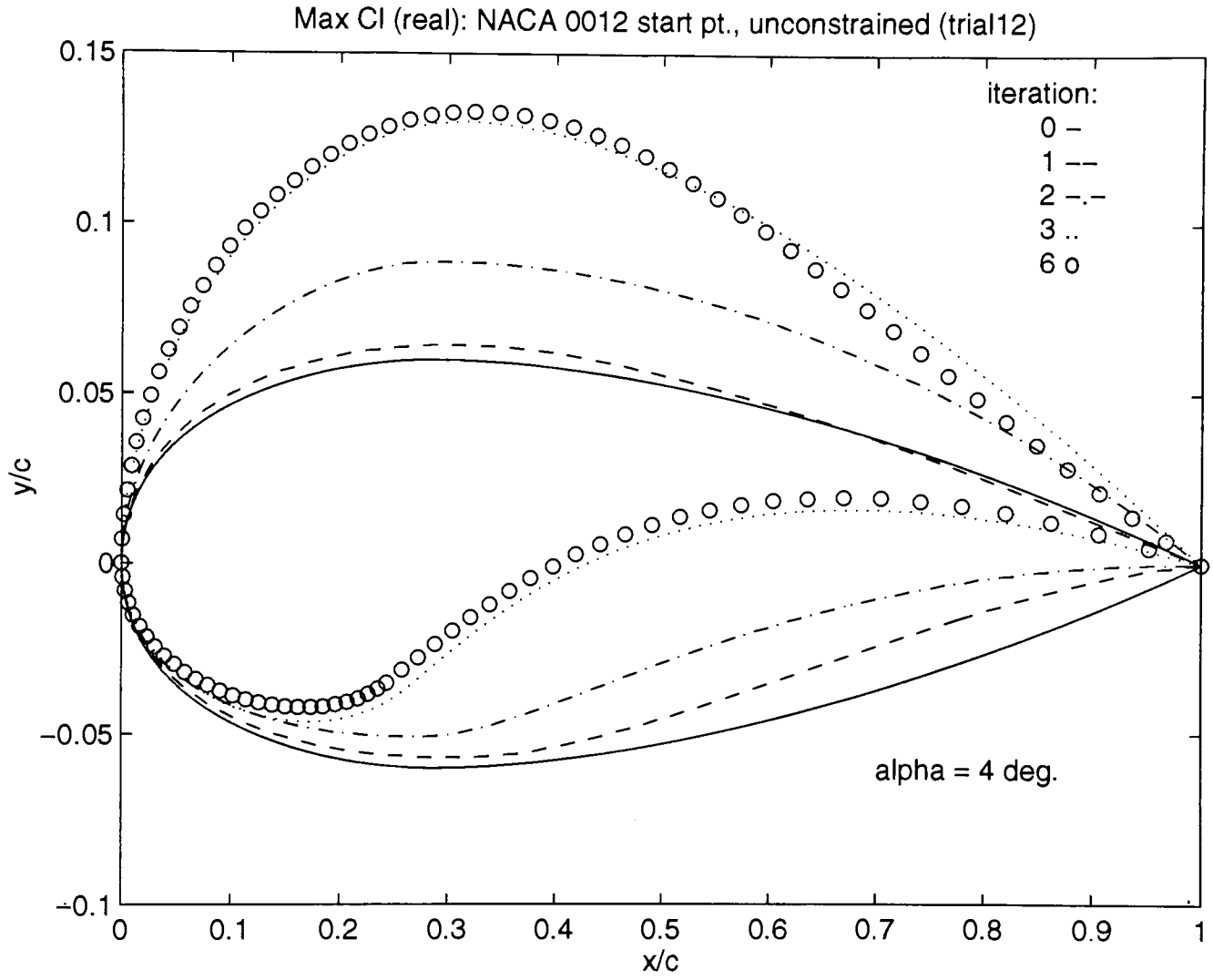


Figure 6.2:
Pornlem A: NACA 0012 Start Point, Scheme 2

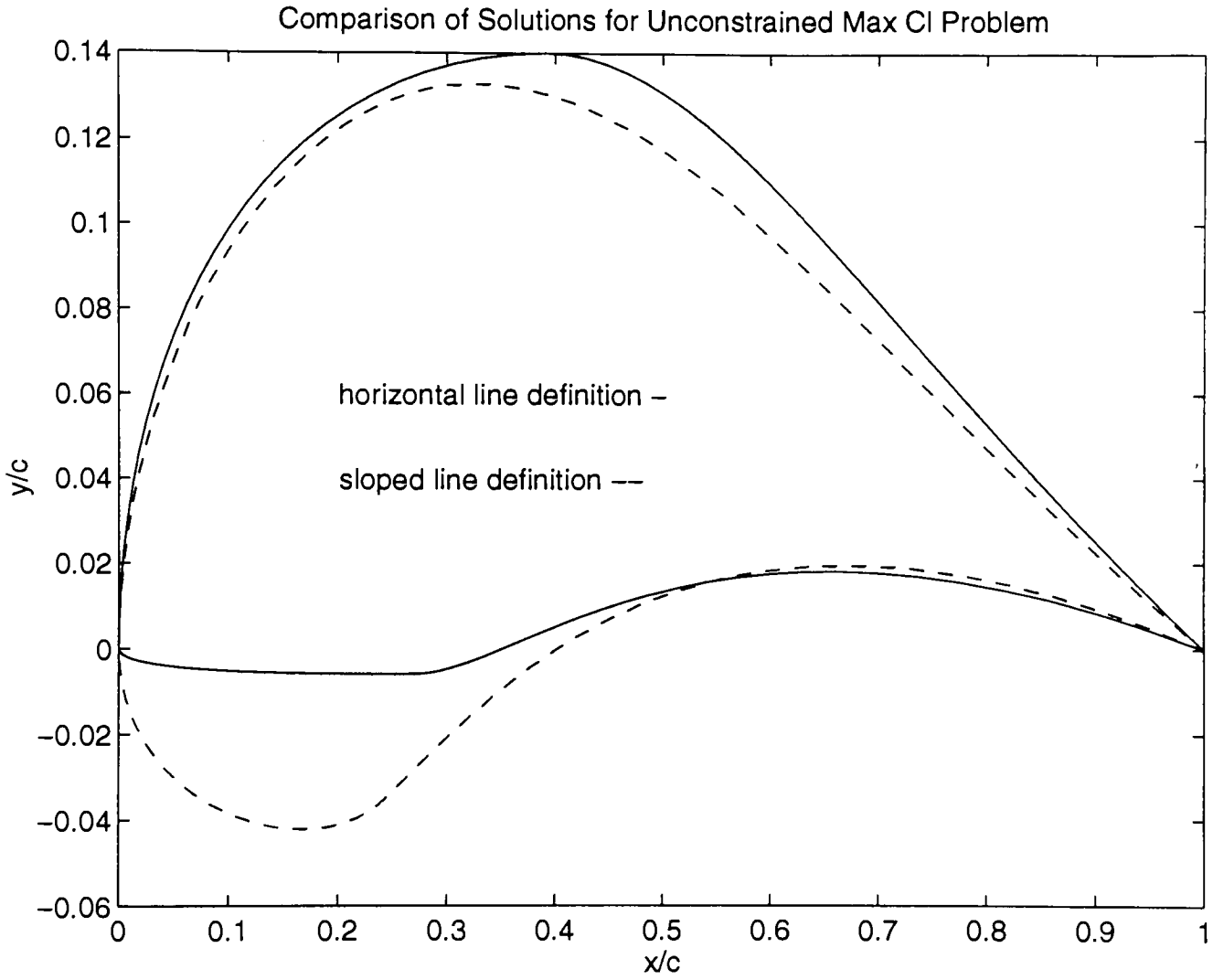


Figure 6.3:
Problem A: NACA 0012 Start Point Comparison

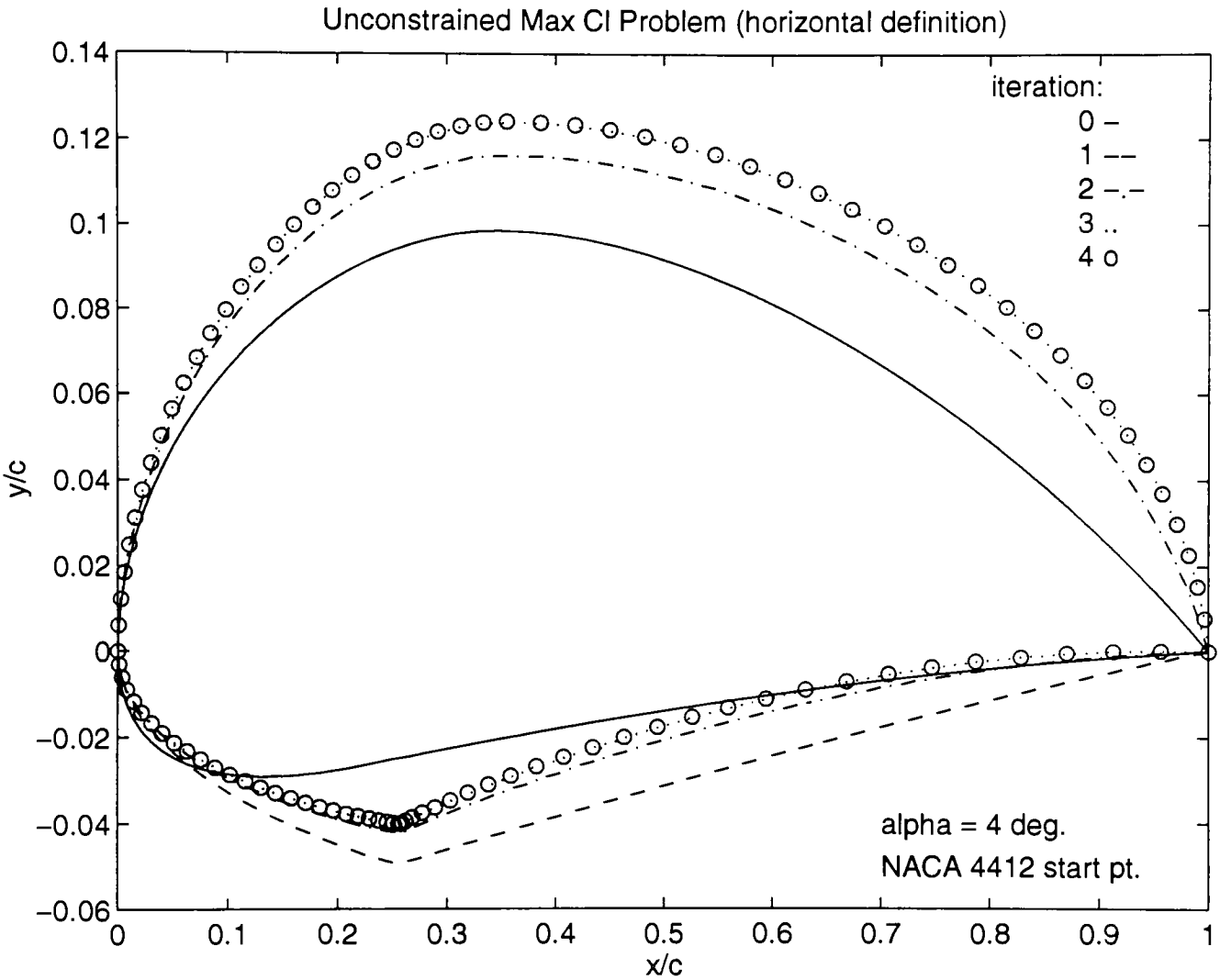


Figure 6.4:
Problem A: NACA 4412 Start Point, Scheme 1

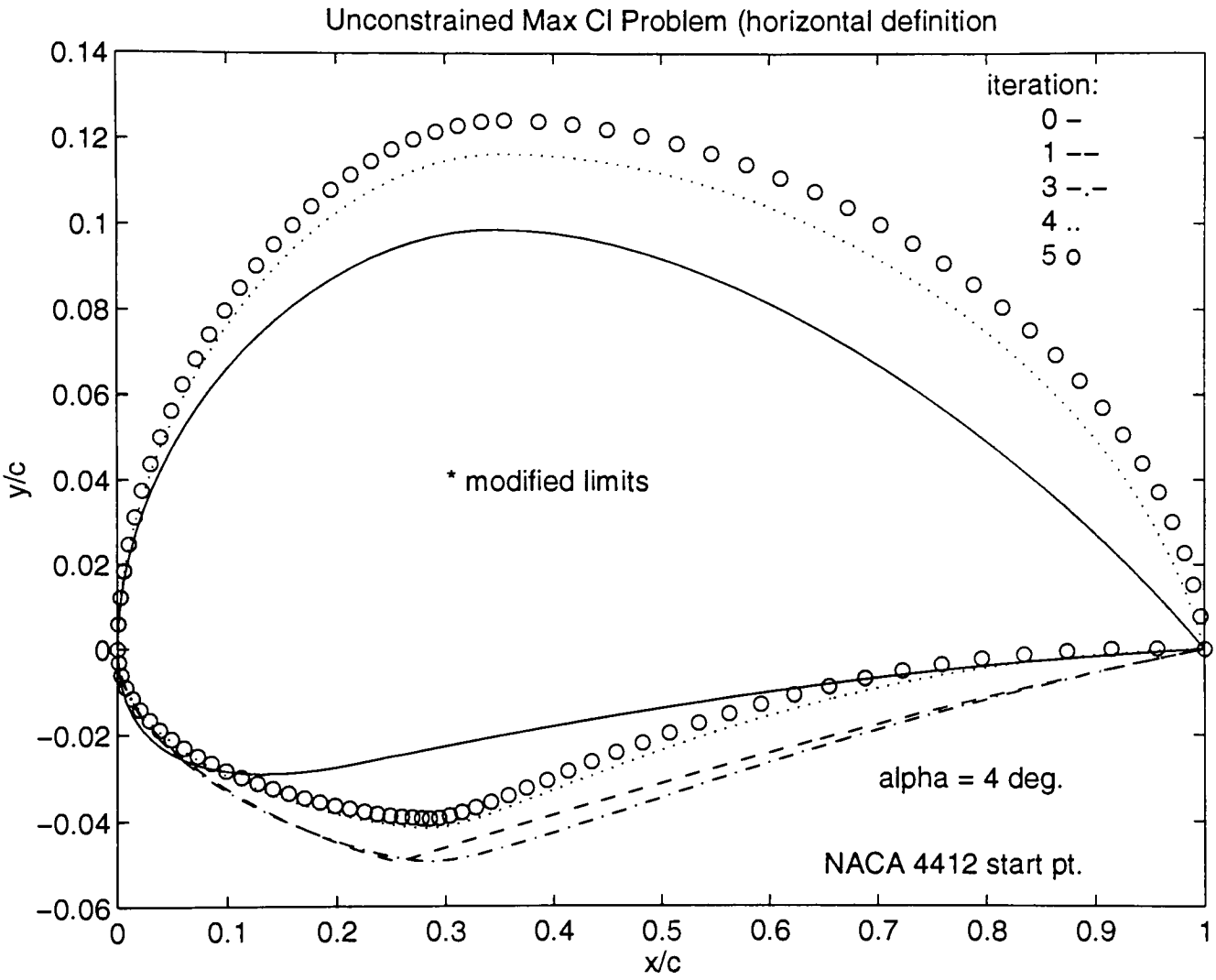


Figure 6.5:
Problem A: NACA 4412 Start Point, Scheme 1 (modified limits)

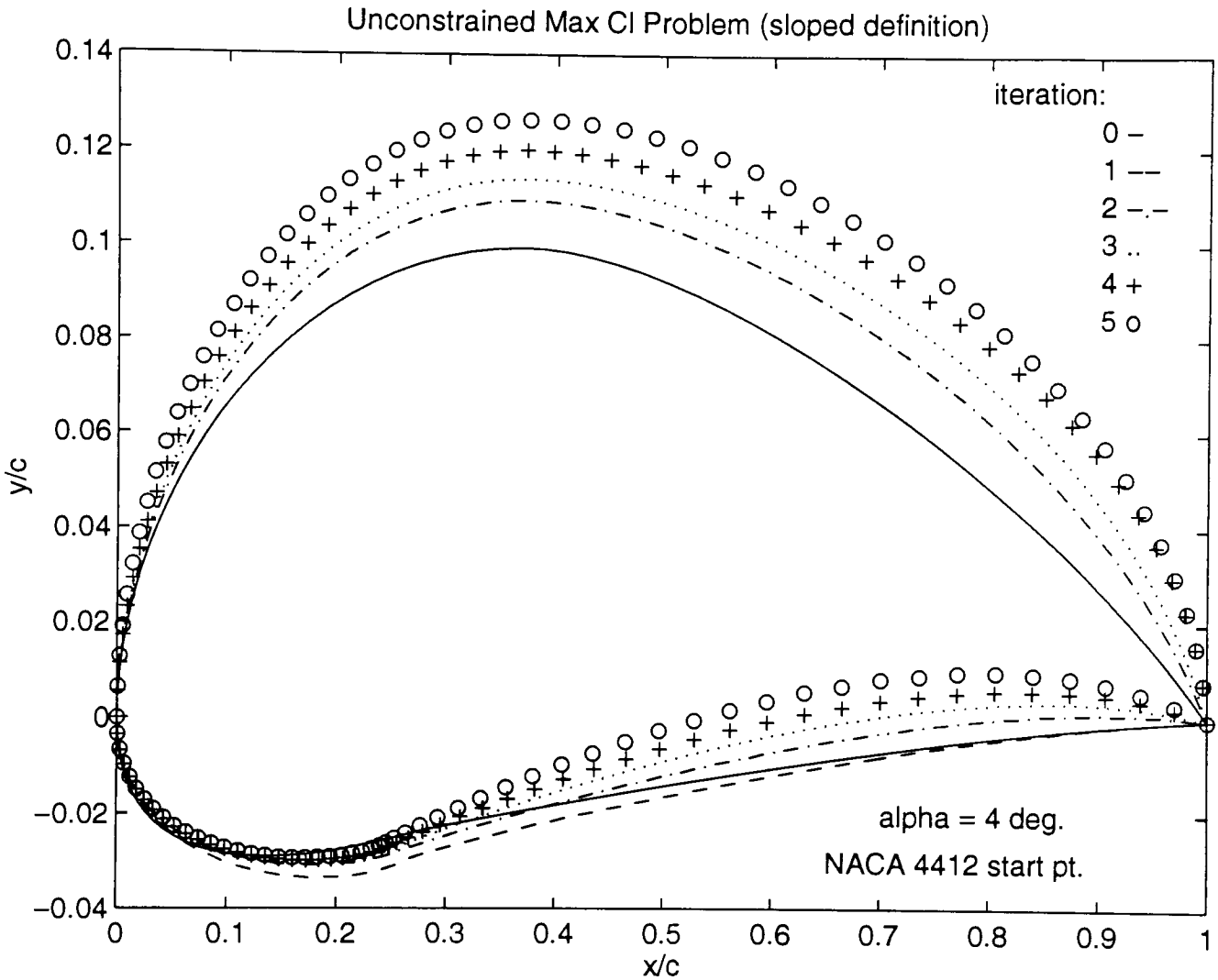


Figure 6.6:
Problem A: NACA 4412 Start Point, Scheme 2

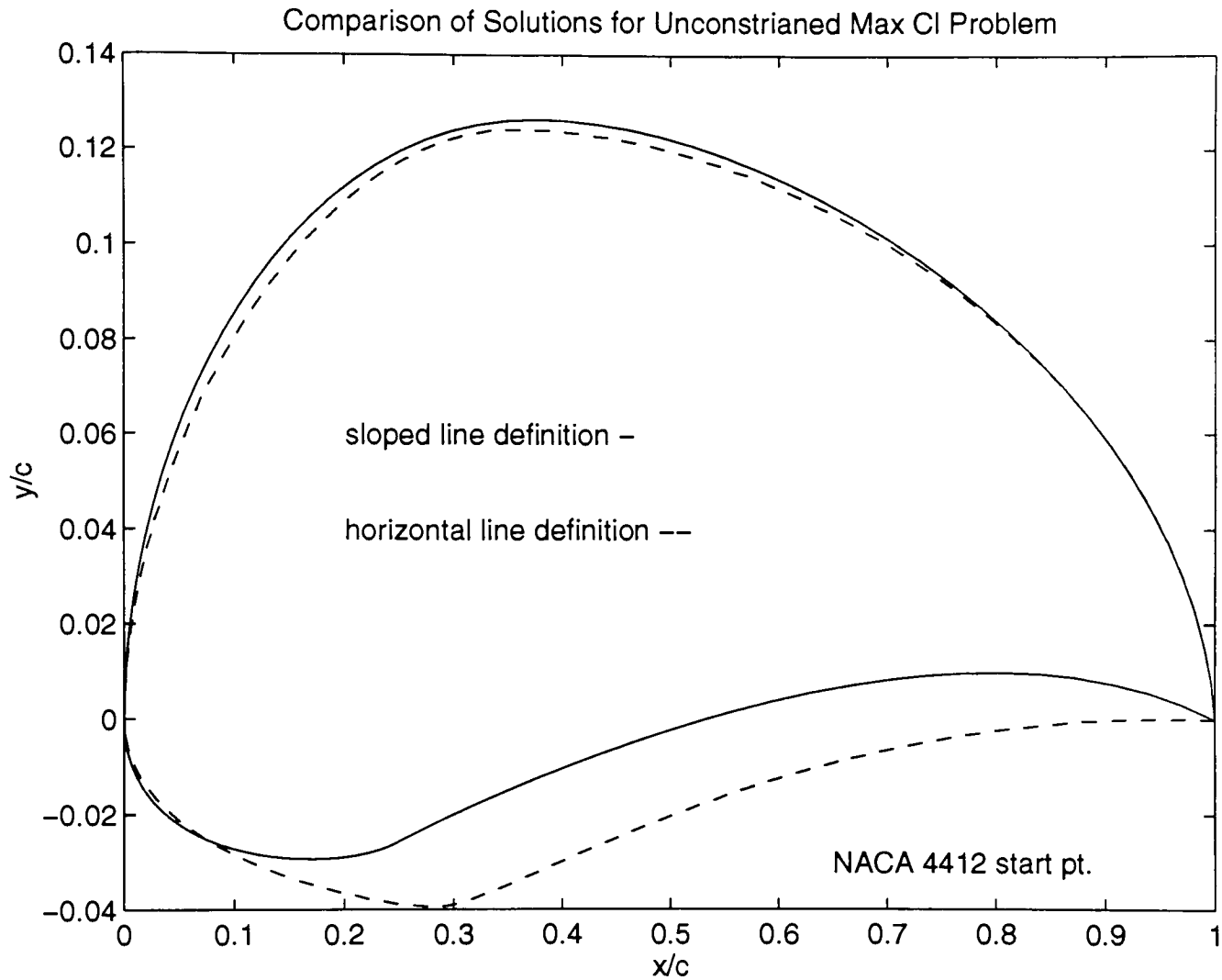


Figure 6.7:
Problem A: NACA 4412 Start Point Comparison

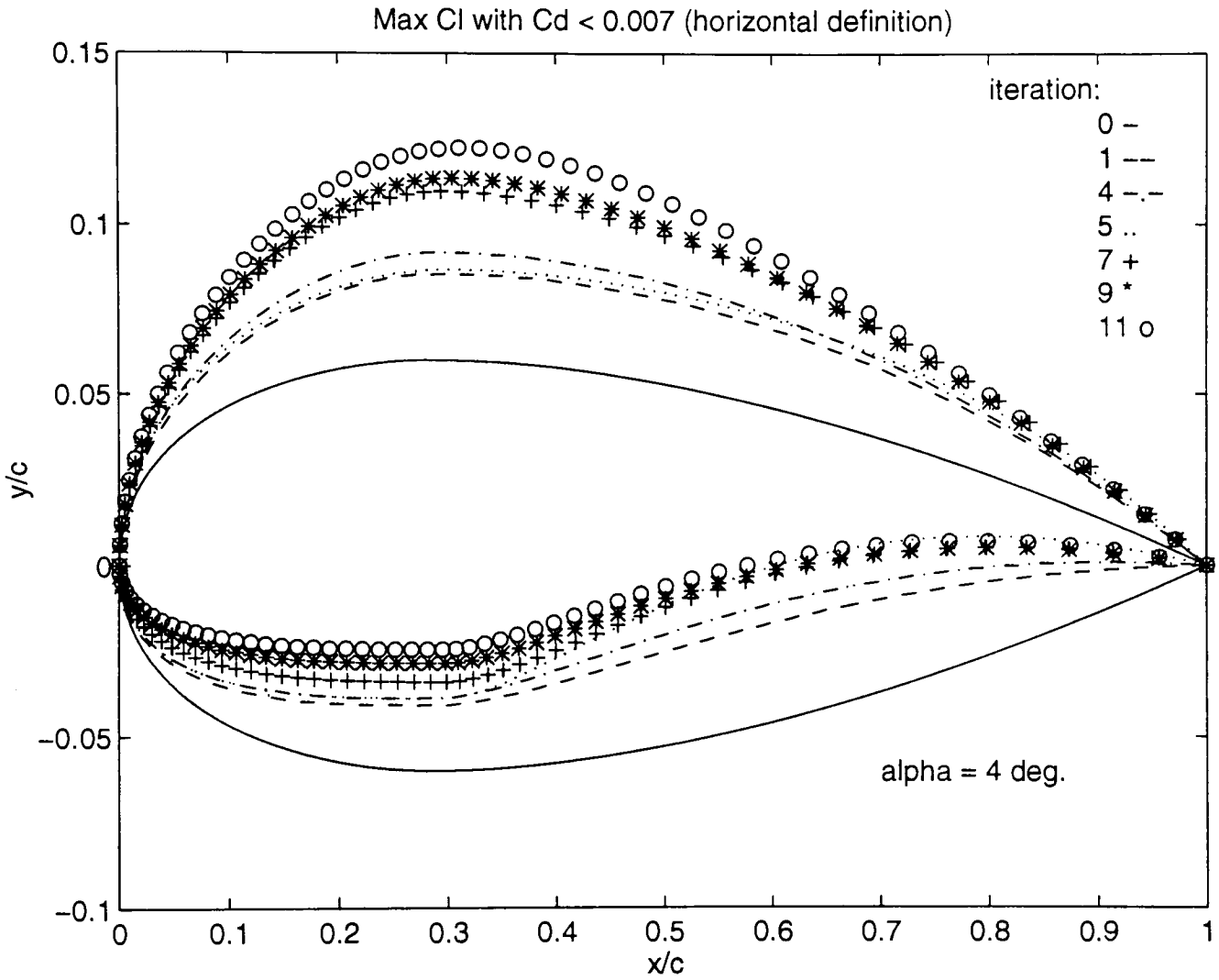


Figure 6.8:
Problem B: NACA 0012 Start Point, Scheme 1

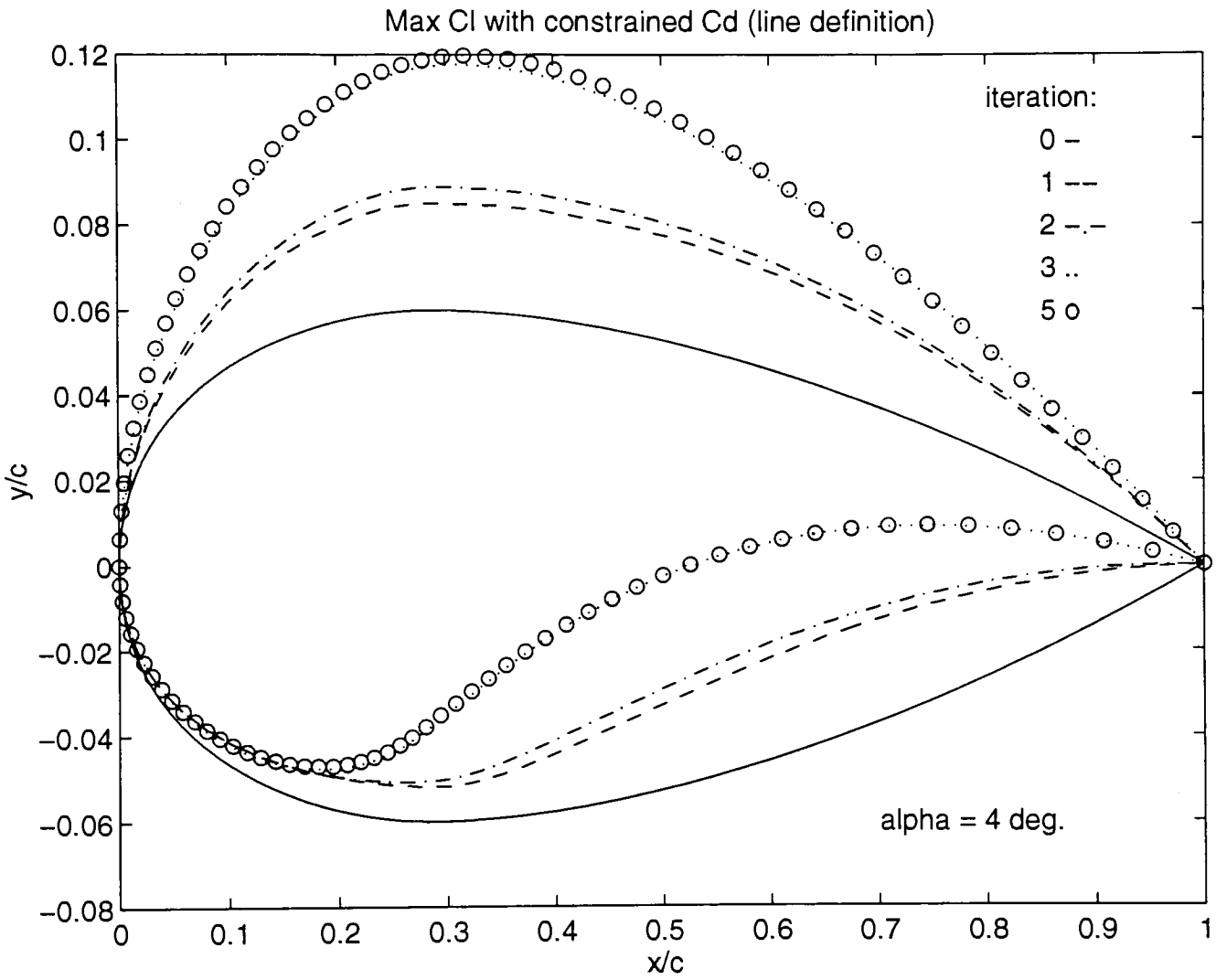


Figure 6.9:
Problem B: NACA 0012 Start Point, Scheme 2

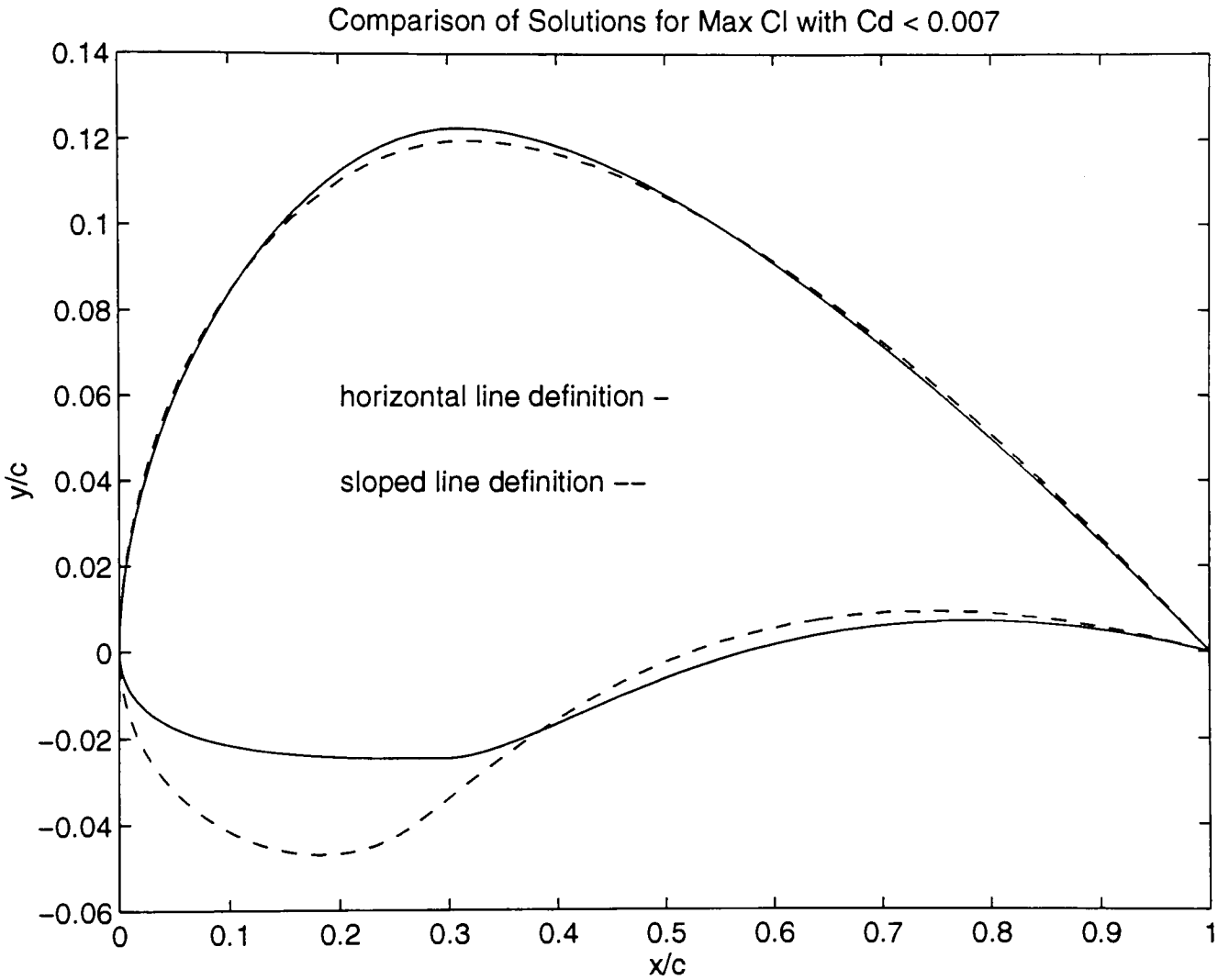


Figure 6.10:
Problem B: NACA 0012 Strart Point Comparison

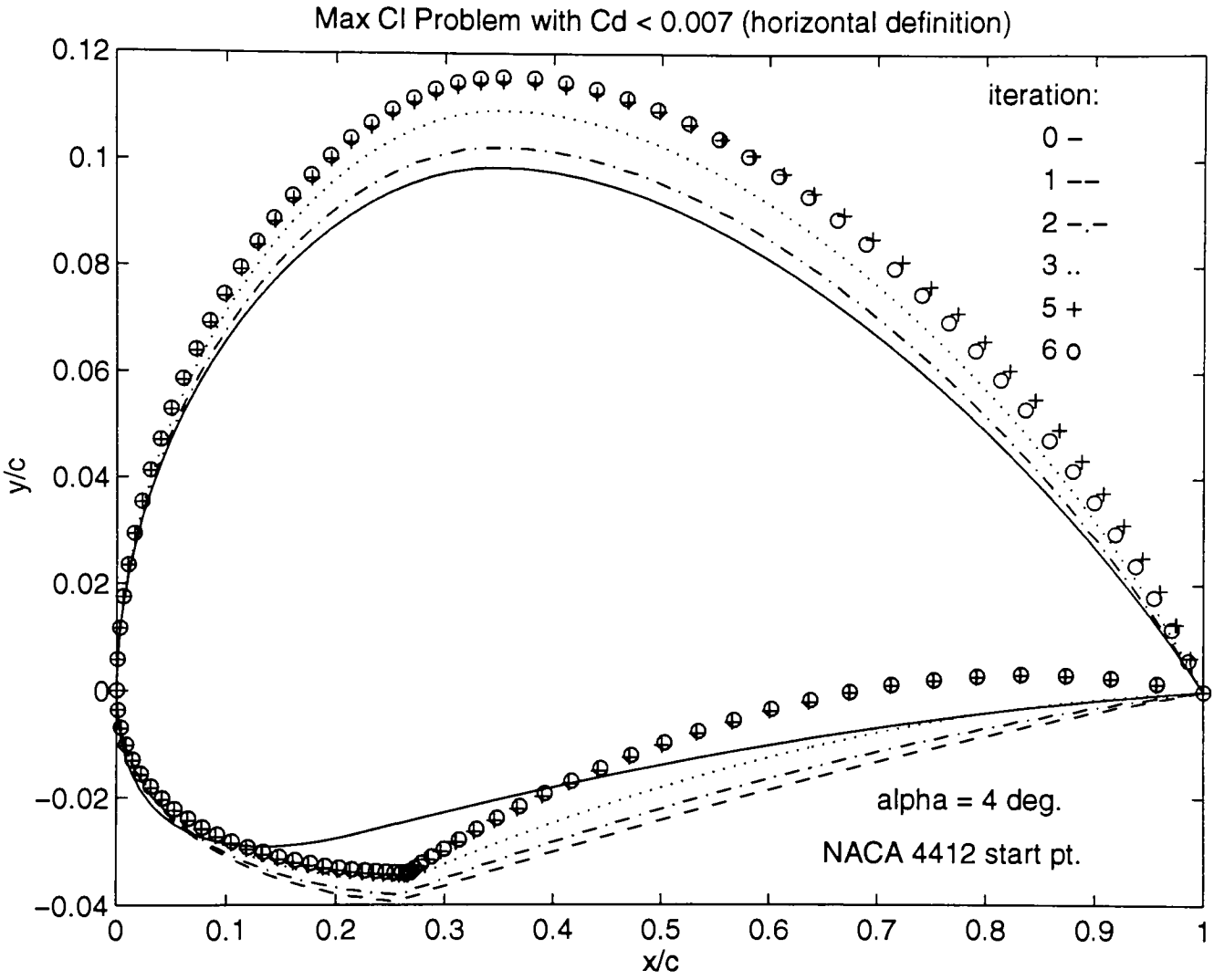


Figure 6.11:
Problem B: NACA 4412 Start Point, Scheme 1

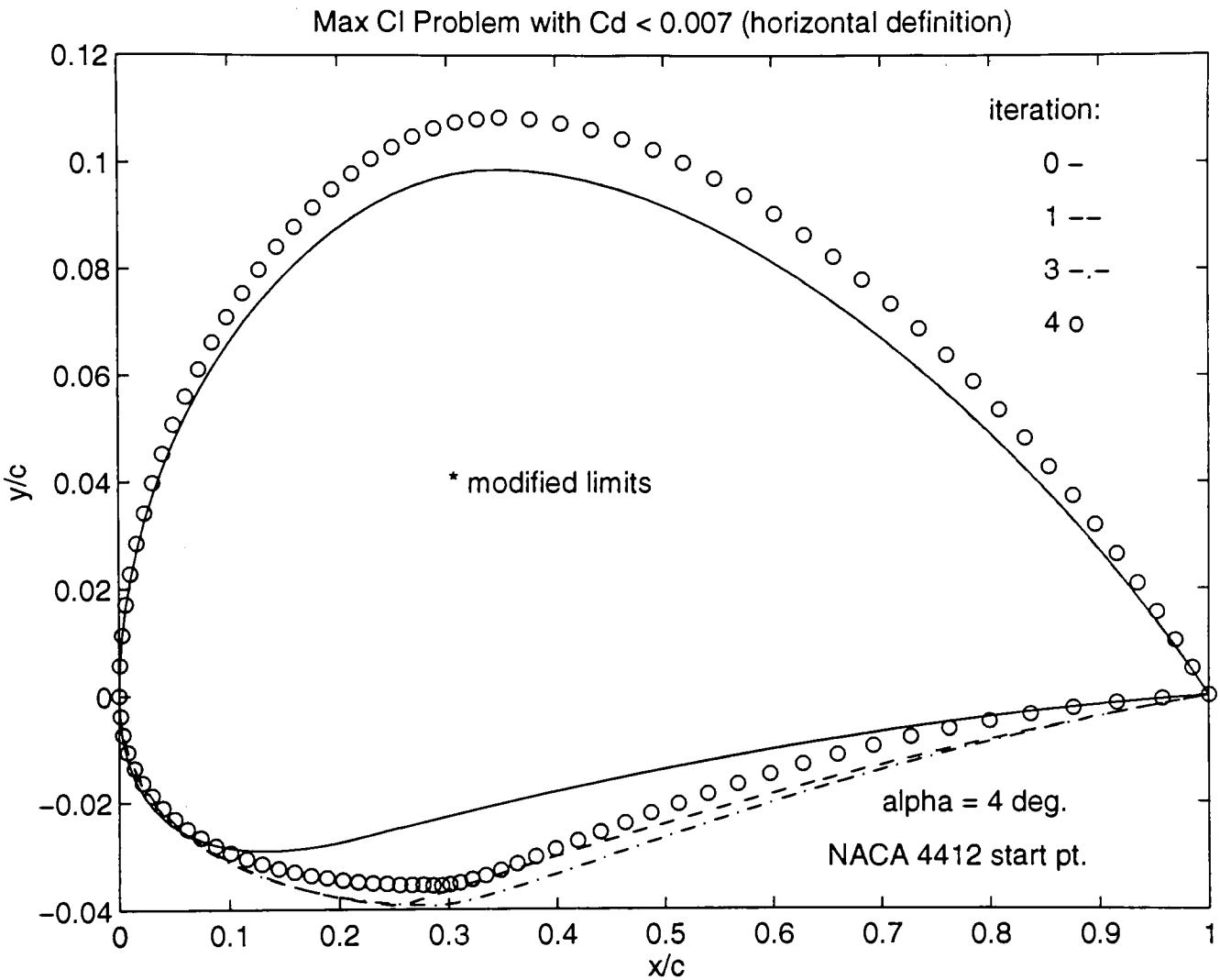


Figure 6.12:
Problem B: NACA 4412 Start Point, Scheme 1 (modified limits)

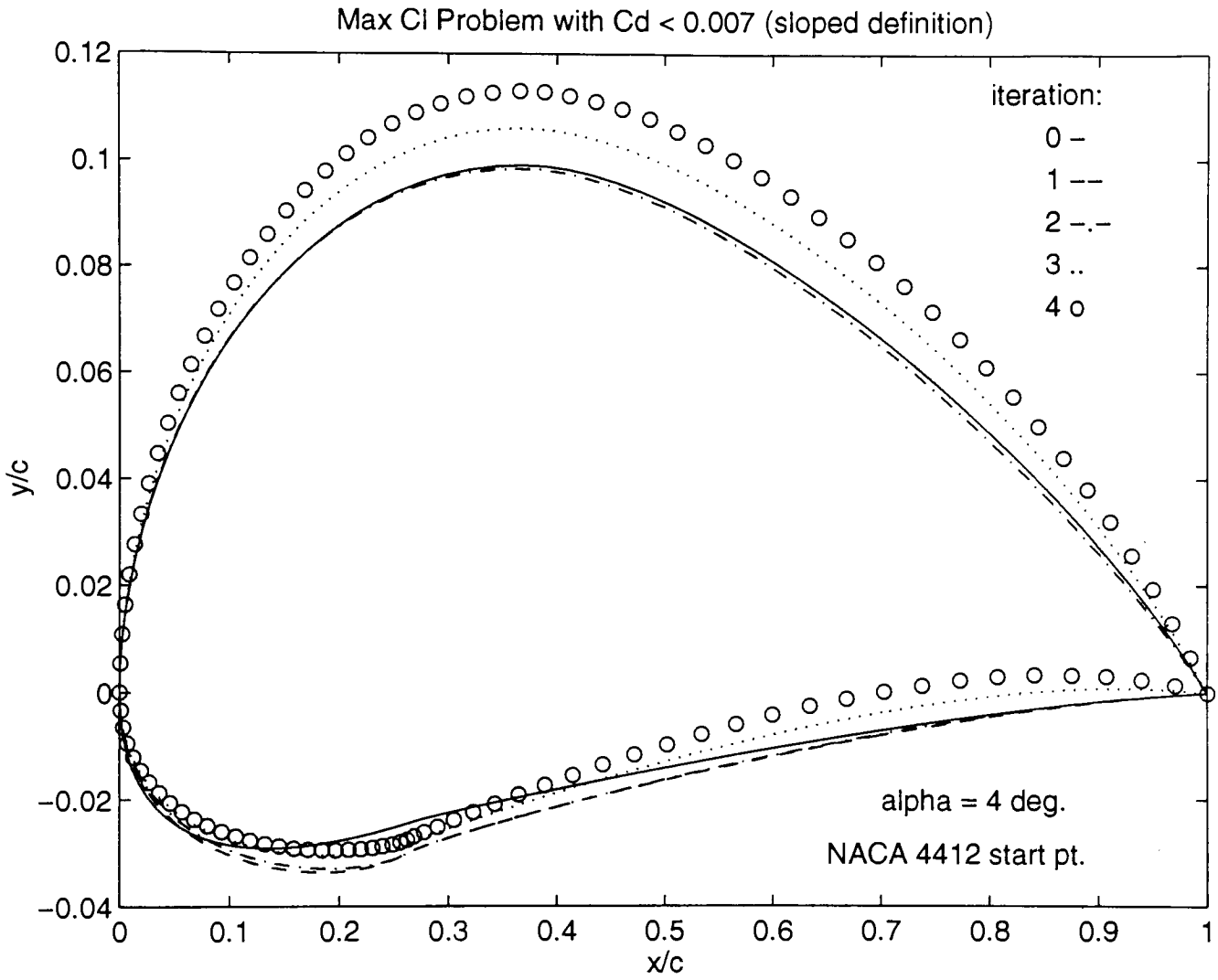


Figure 6.13:
Problem B: NACA 4412 Start Point, Scheme 2

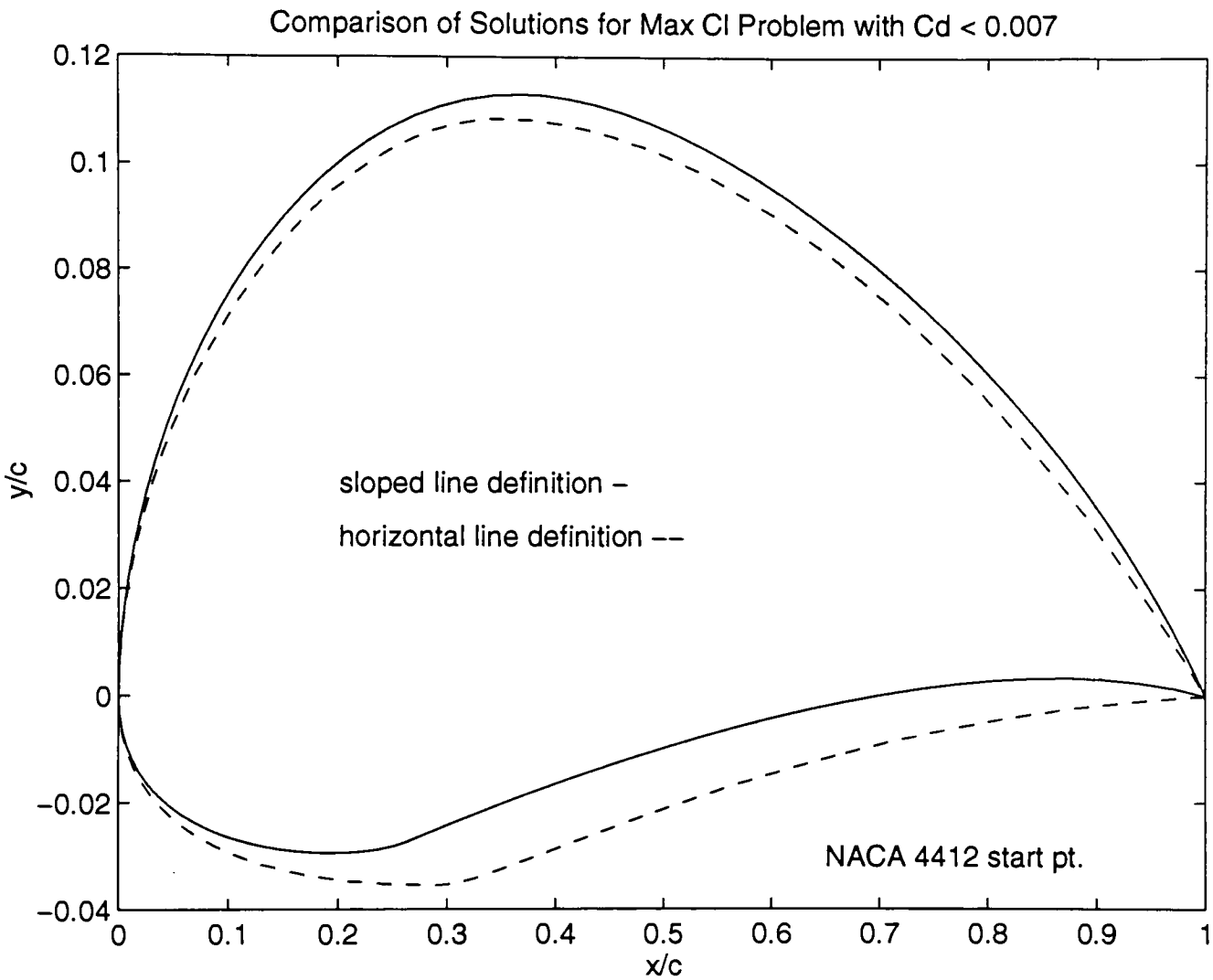


Figure 6.14:
Problem B: NACA 4412 Start Point Comparison

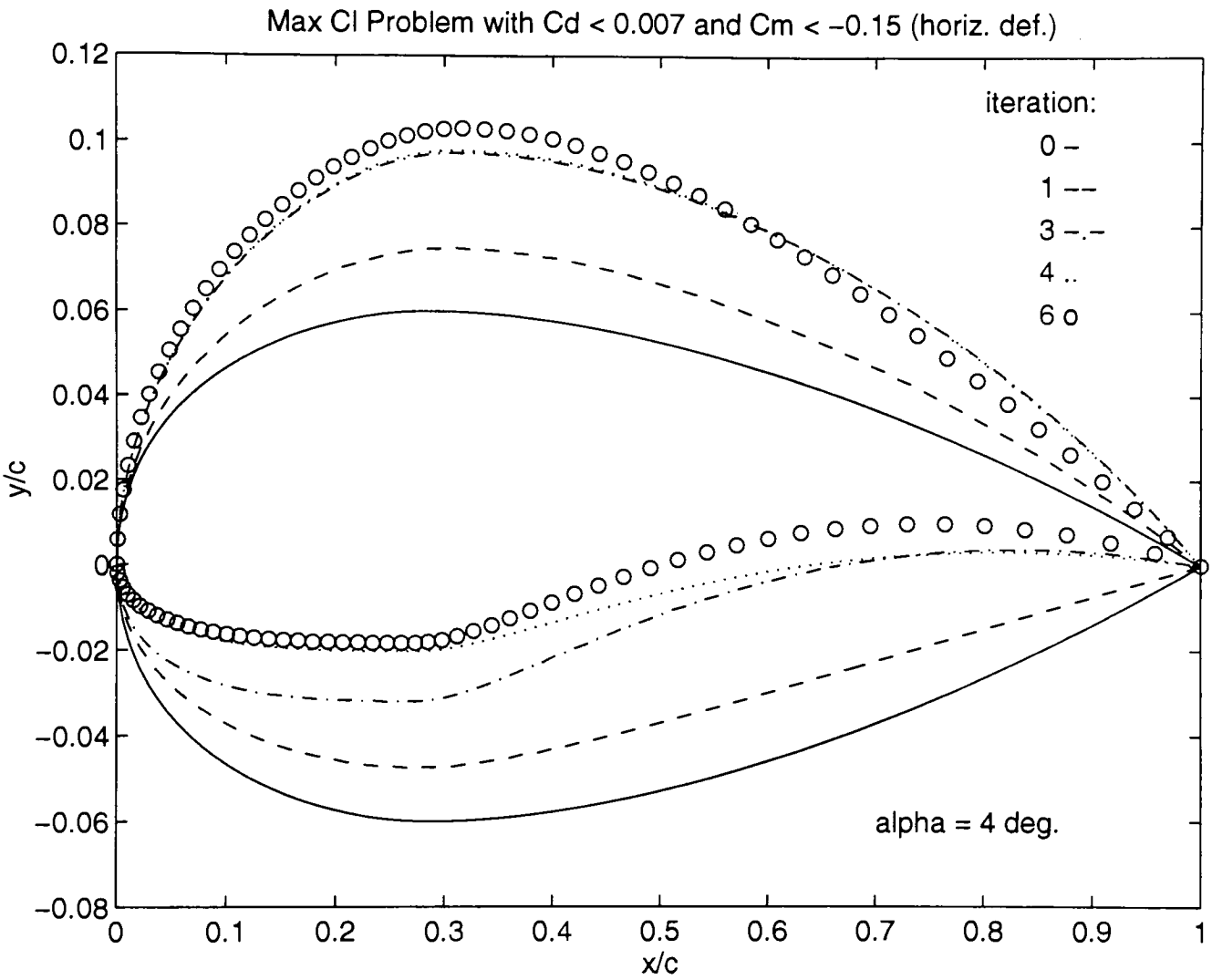


Figure 6.15:
Problem C: NACA 0012 Start Point, Scheme 1

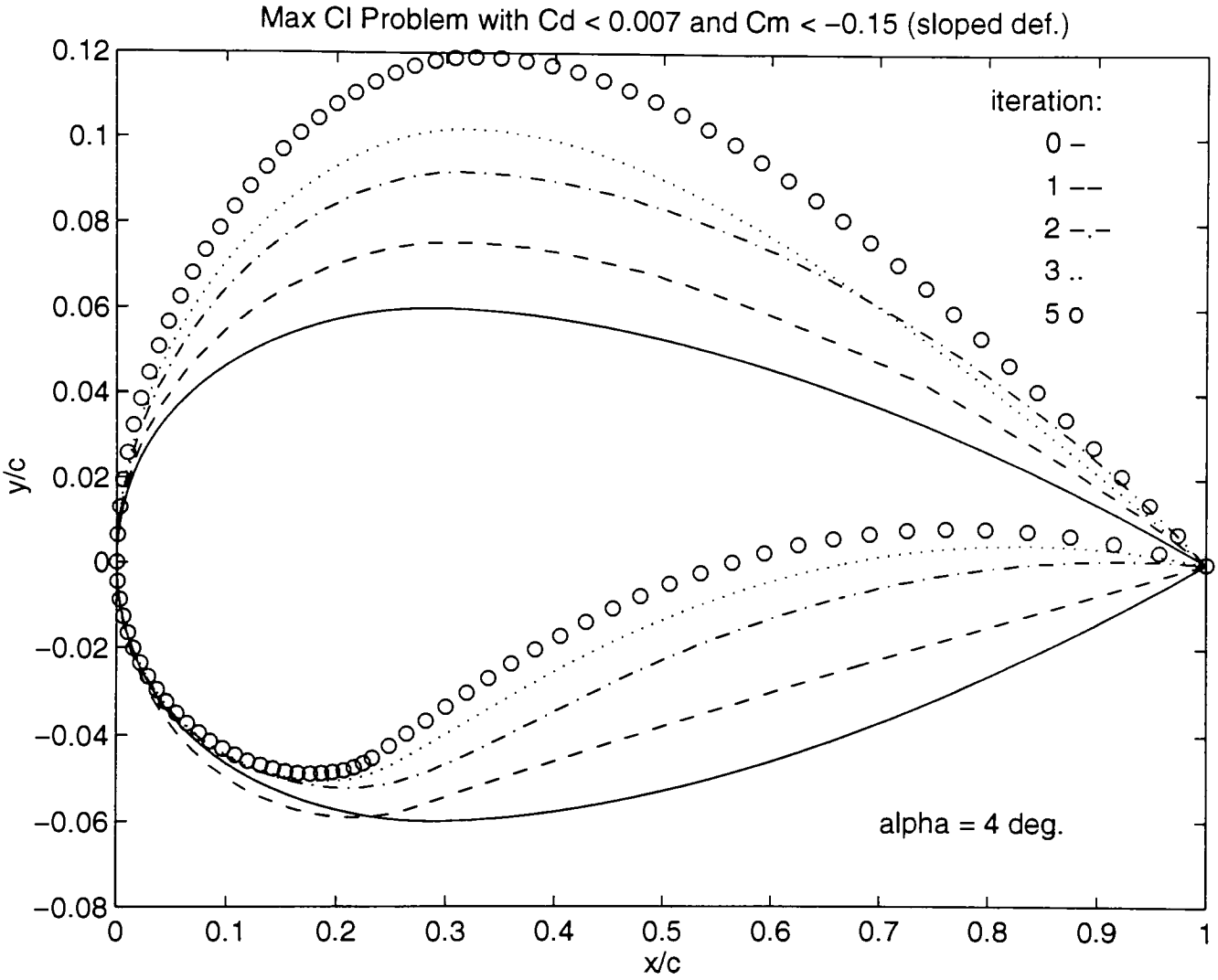


Figure 6.16:
Problem C: NACA 0012 Start Point, Scheme 2

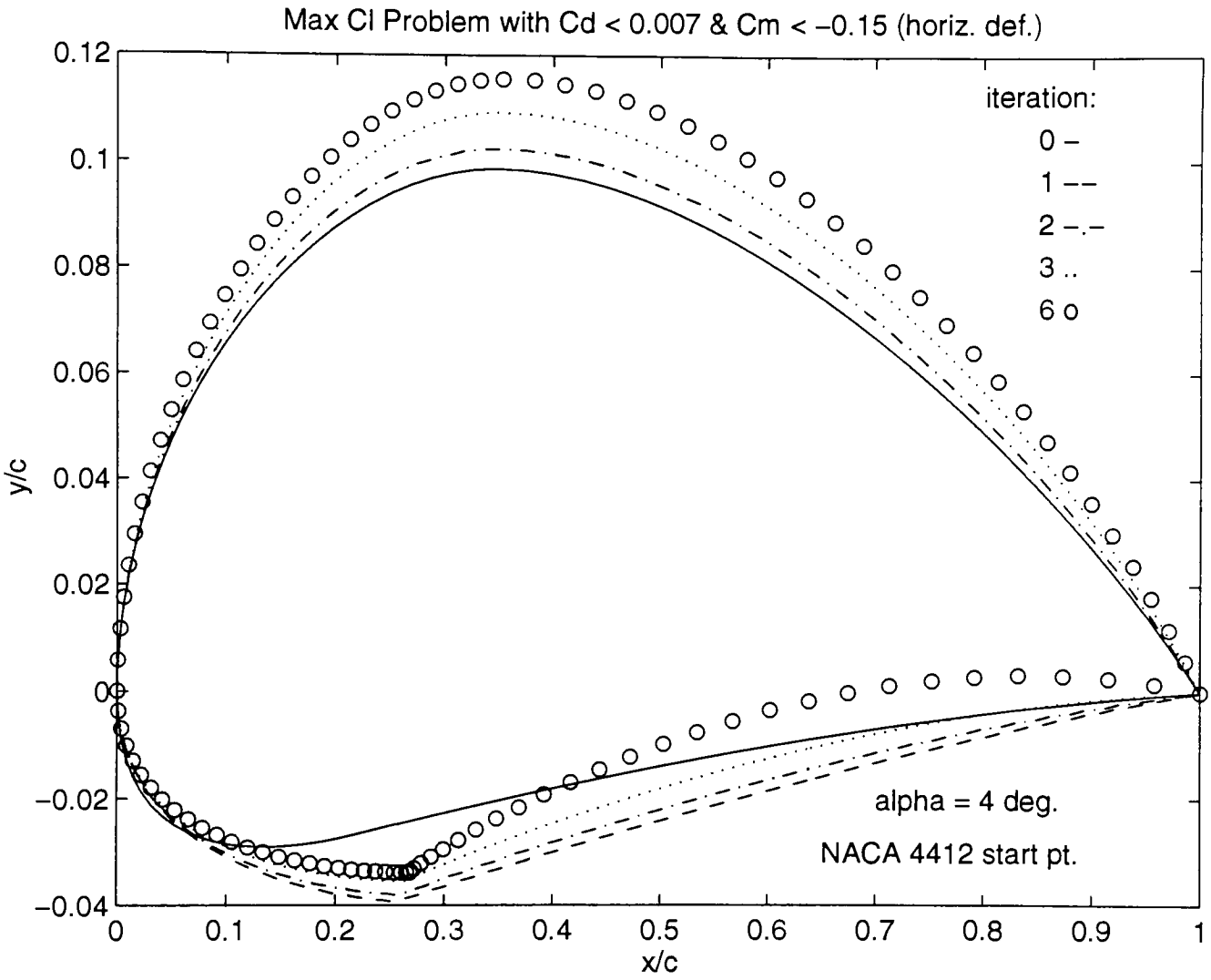


Figure 6.17:
Problem C: NACA 4412 Start Point, Scheme 1

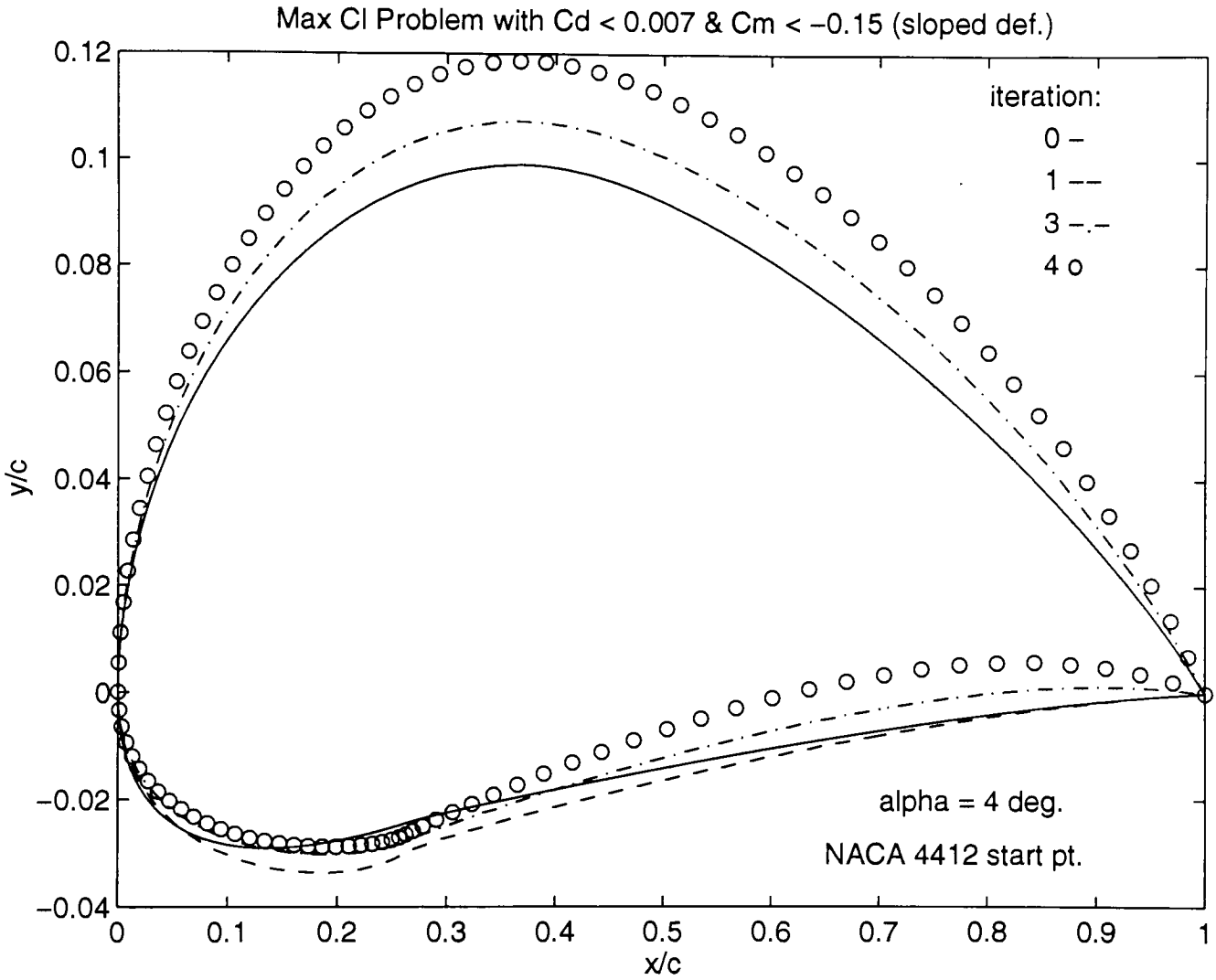


Figure 6.18:
Problem C: NACA 4412 Start Point, Scheme 2

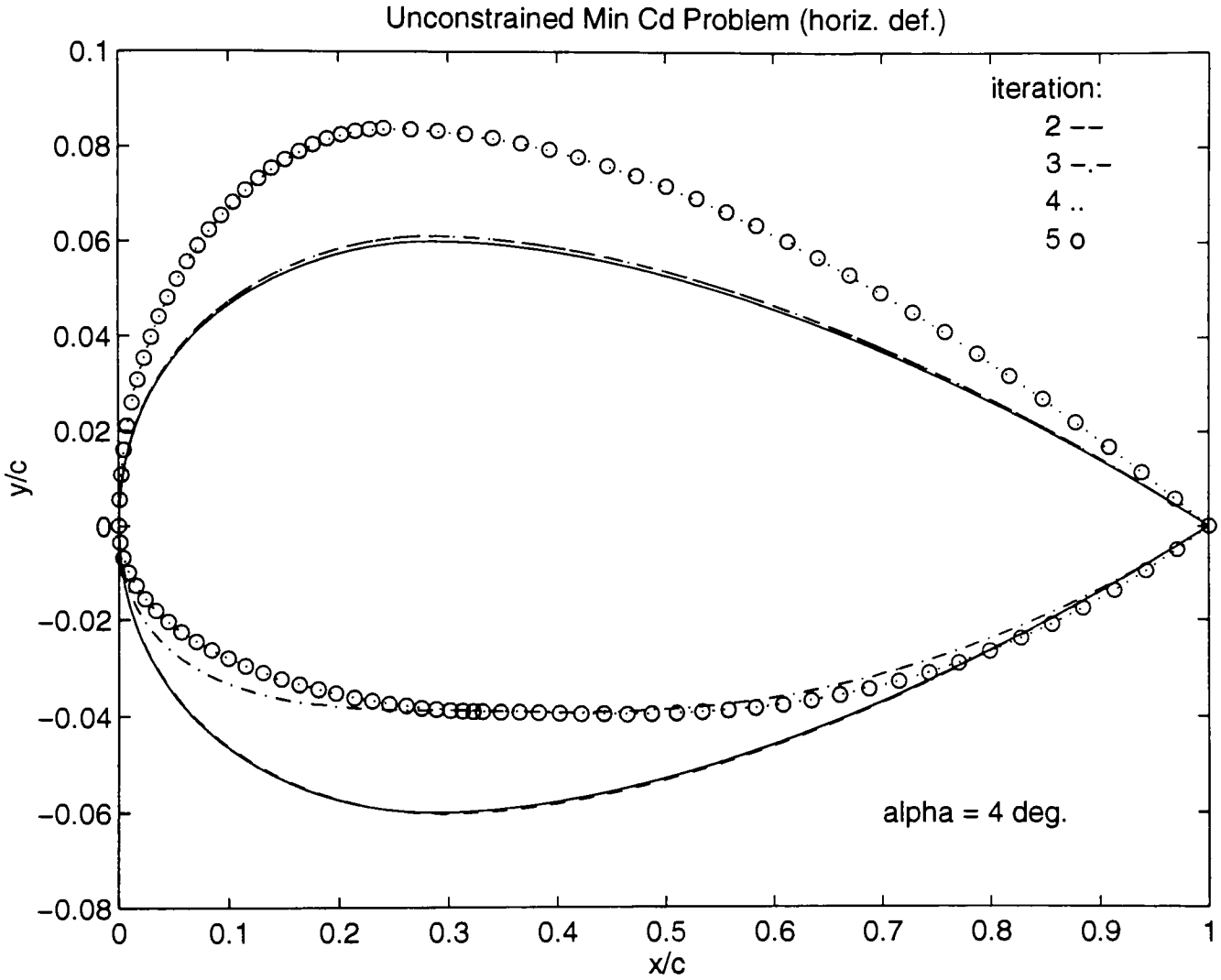


Figure 6.19:
Problem D: NACA 0012 Start Point, Scheme 1

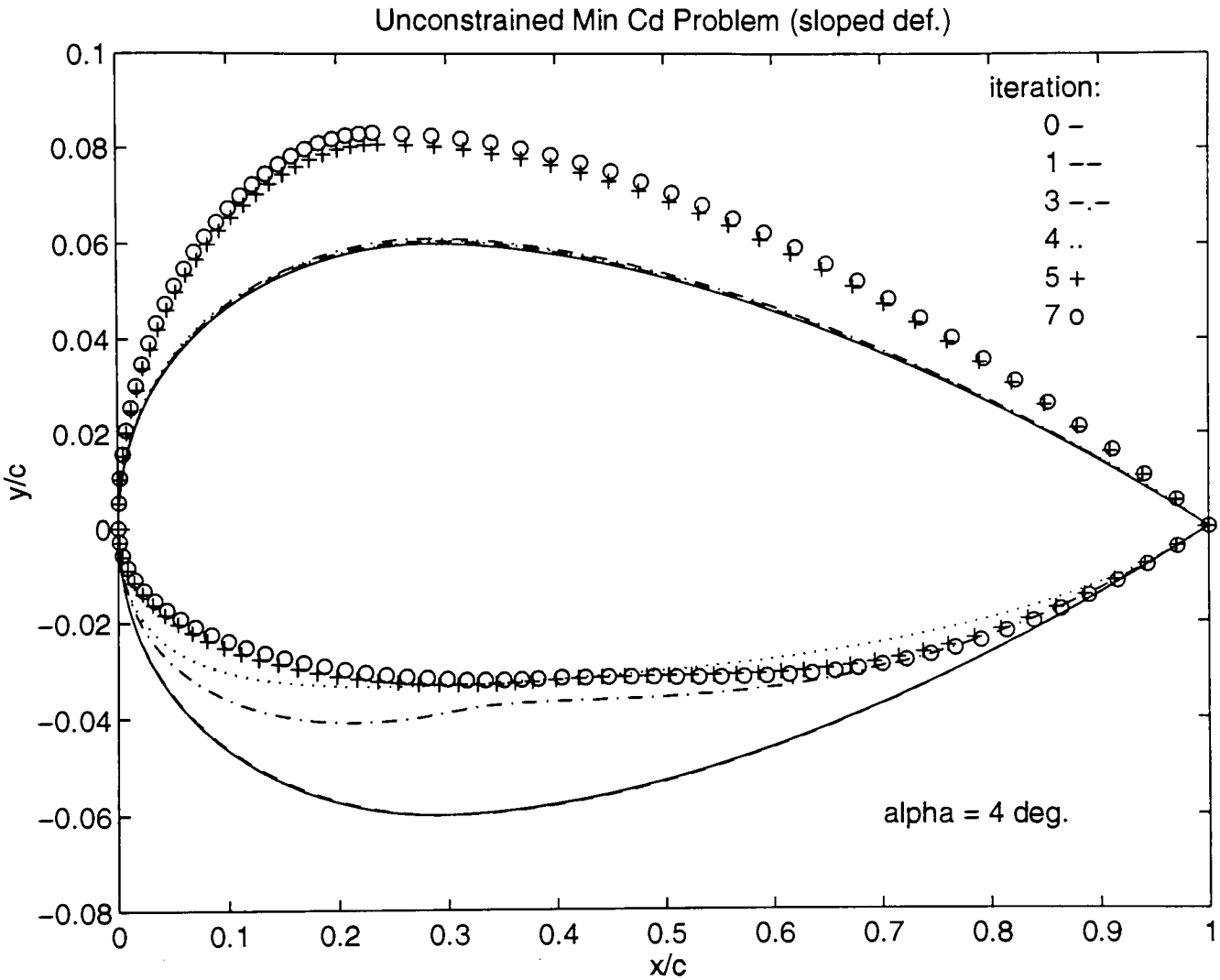


Figure 6.20:
Problem D: NACA 0012 Start Point, Scheme 2

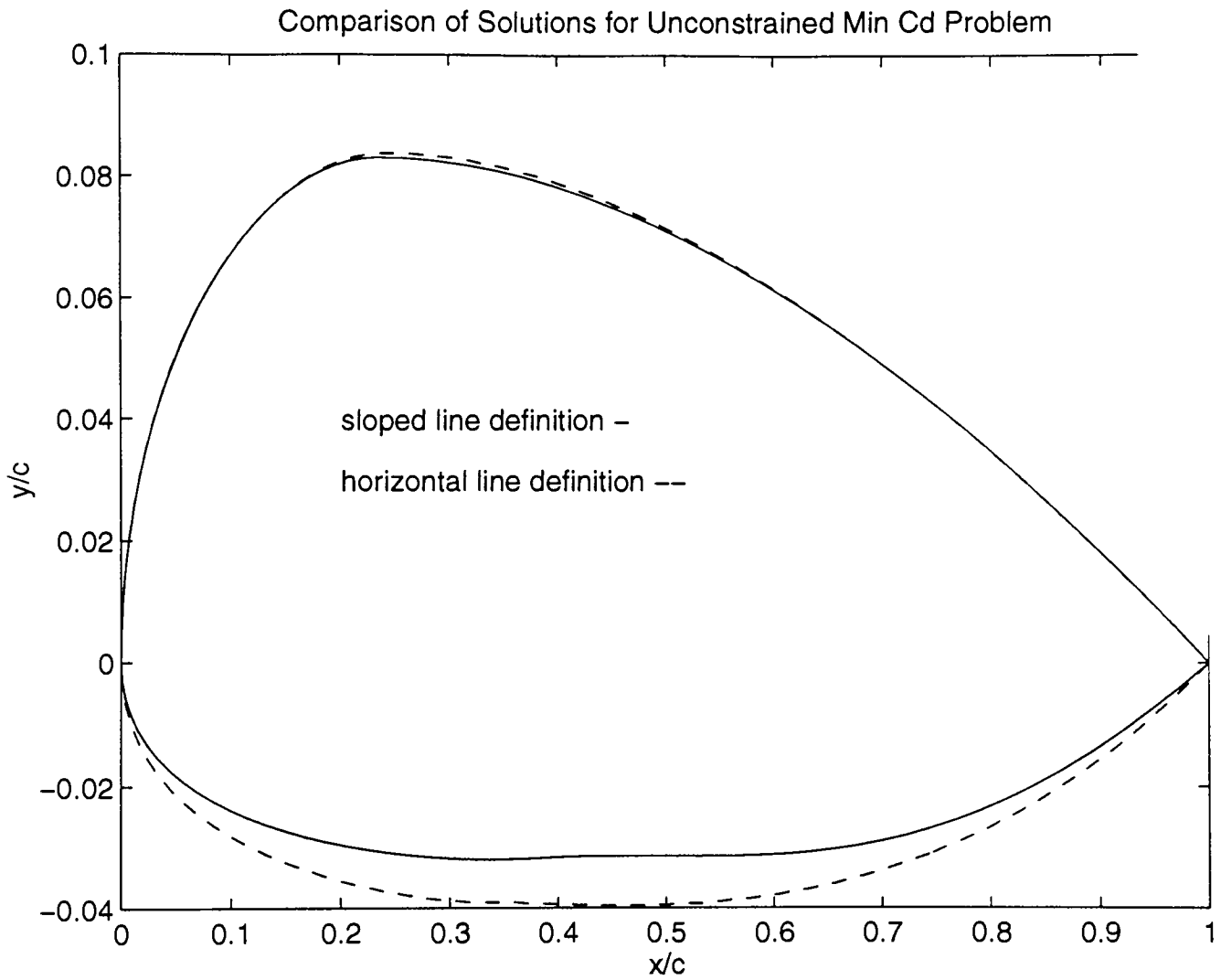


Figure 6.21:
Problem D: NACA 0012 Start Point Comparison

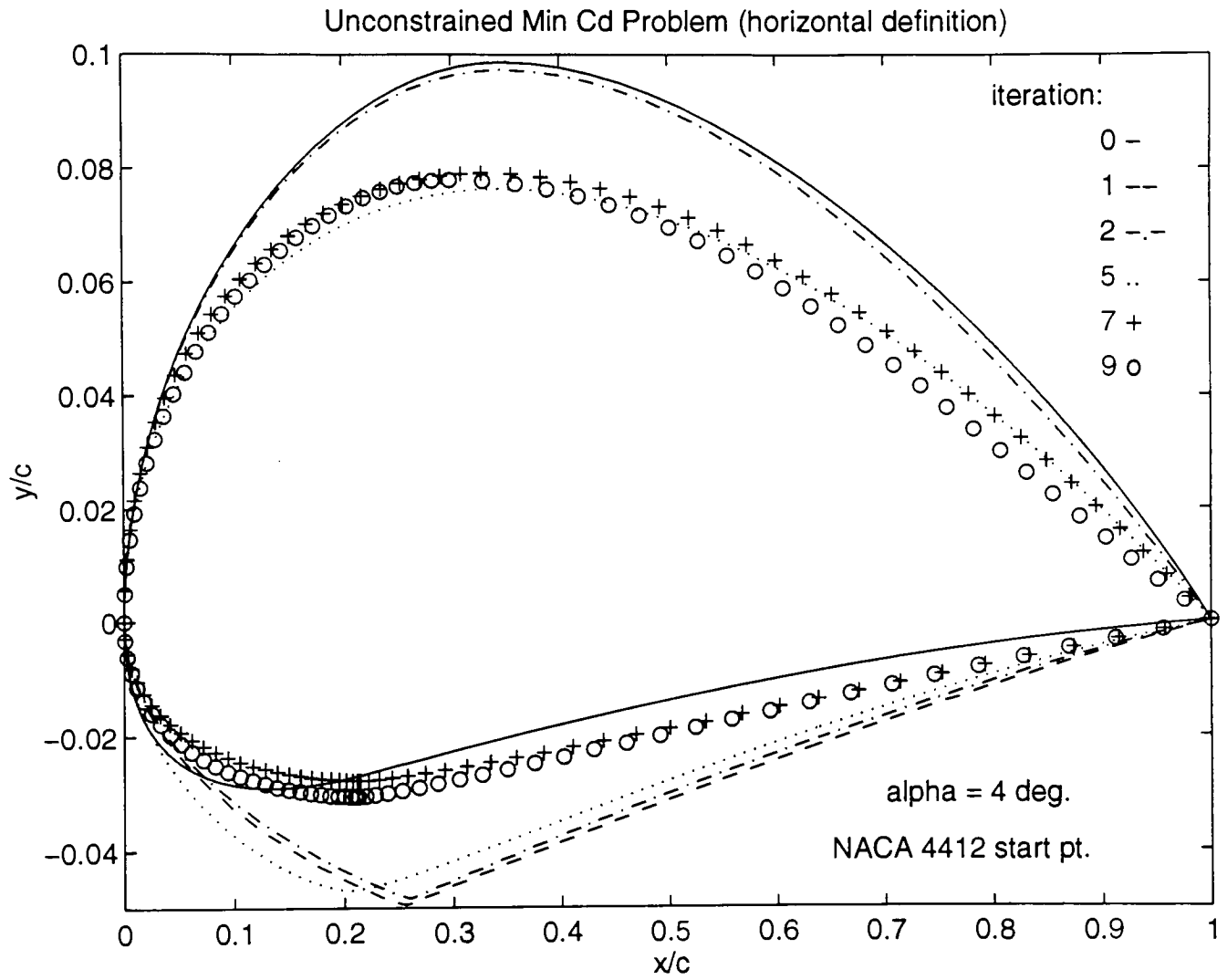


Figure 6.22:
Problem D: NACA 4412 Start Point, Scheme 1

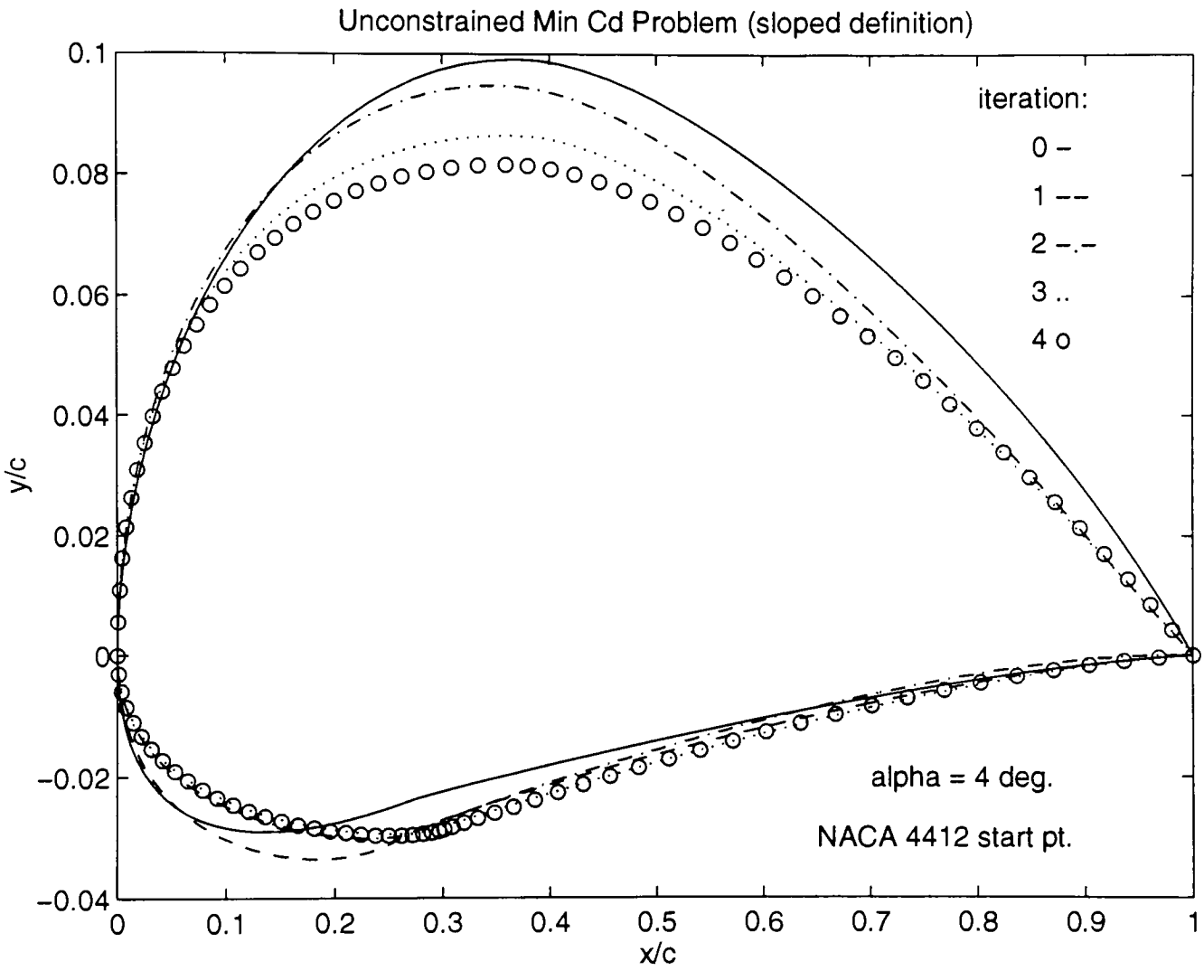


Figure 6.23:
Problem D: NACA 4412 Start Point, Scheme 2

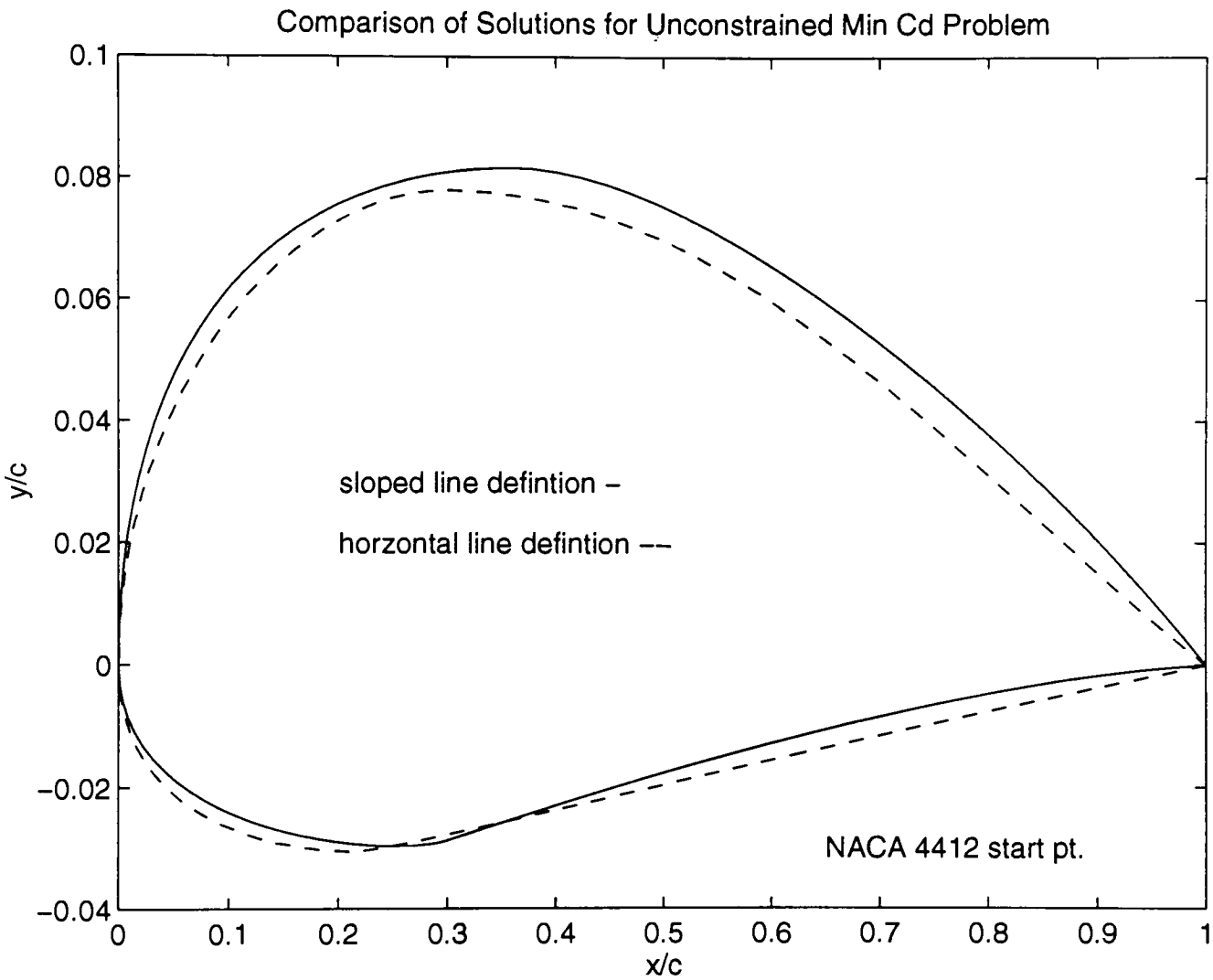


Figure 6.24:
Problem D: NACA 4412 Start Point Comparison

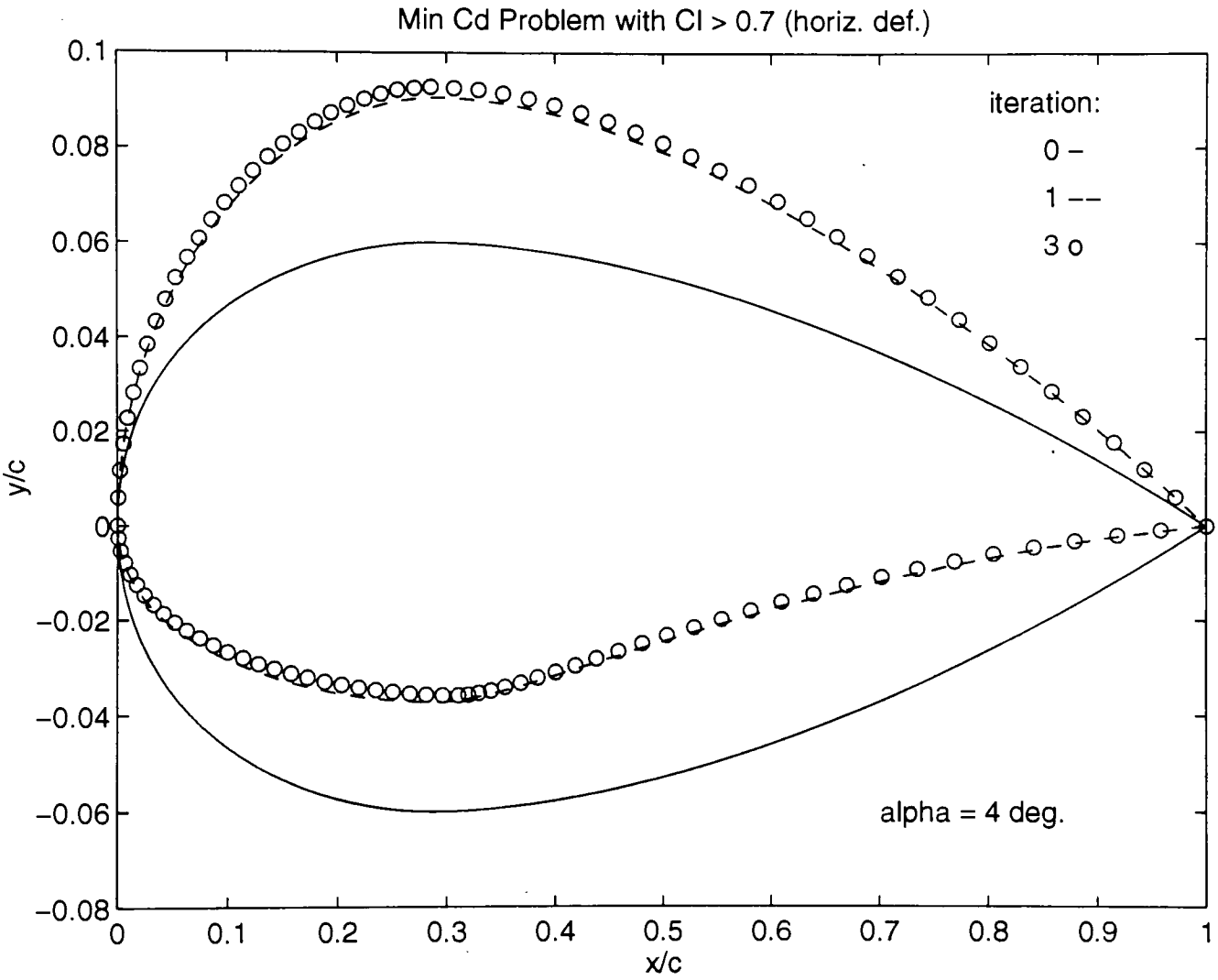


Figure 6.25:
Problem E: NACA 0012 Start Point, Scheme 1

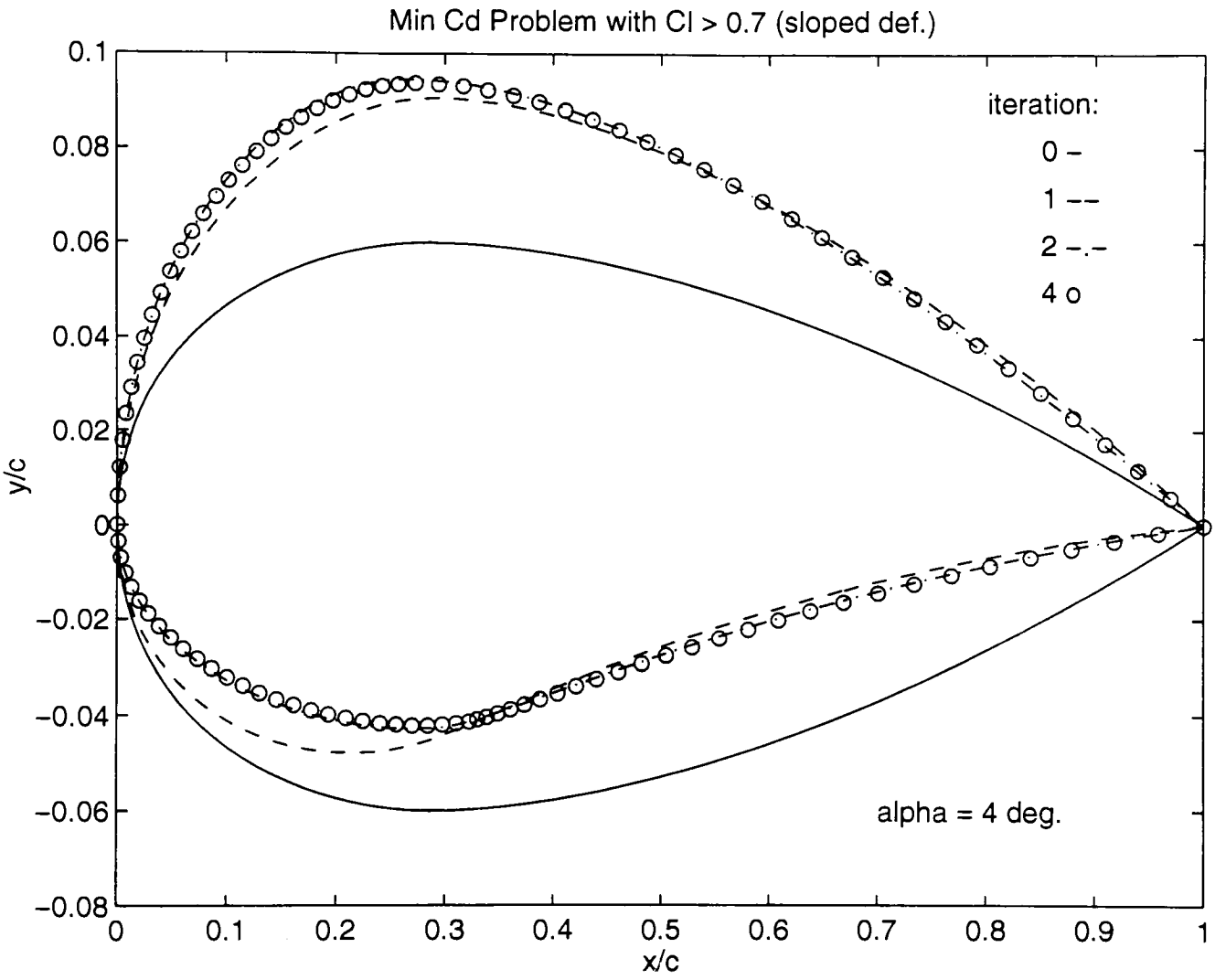


Figure 6.26:
Problem E: NACA 0012 Start Point, Scheme 2

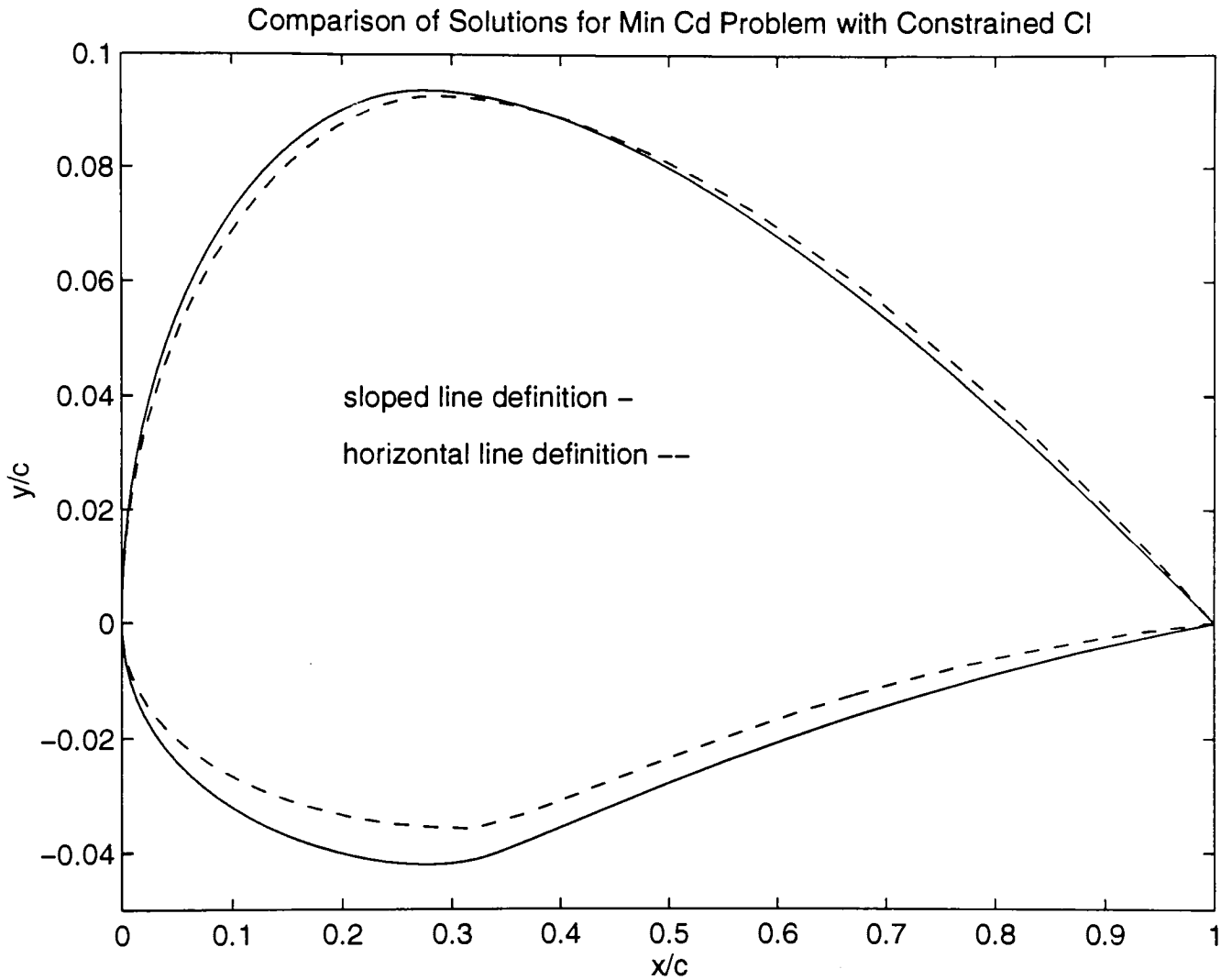


Figure 6.27:
Problem E: NACA 0012 Start Point Comparison

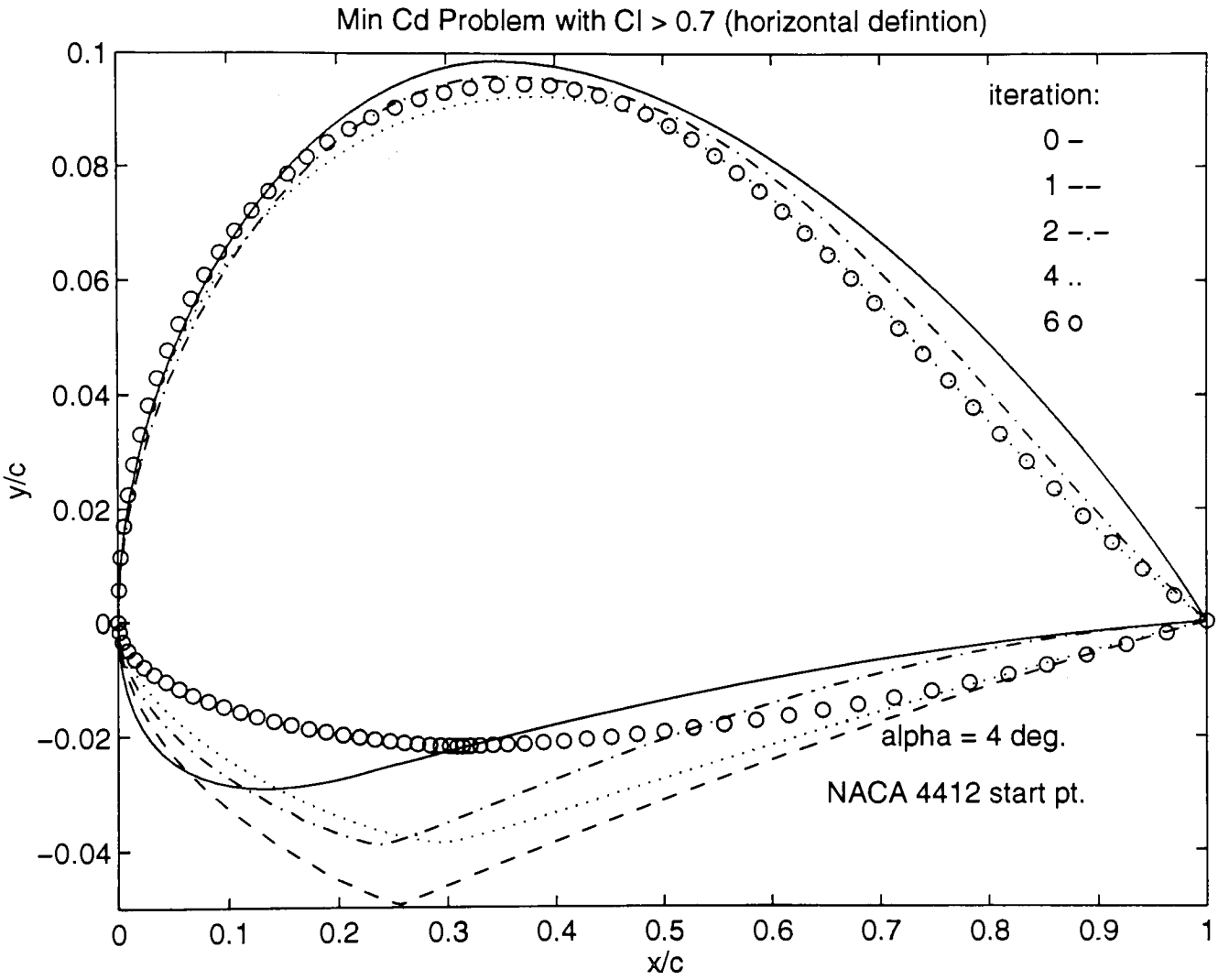


Figure 6.28:
Problem E: NACA 4412 Start Point, Scheme 1

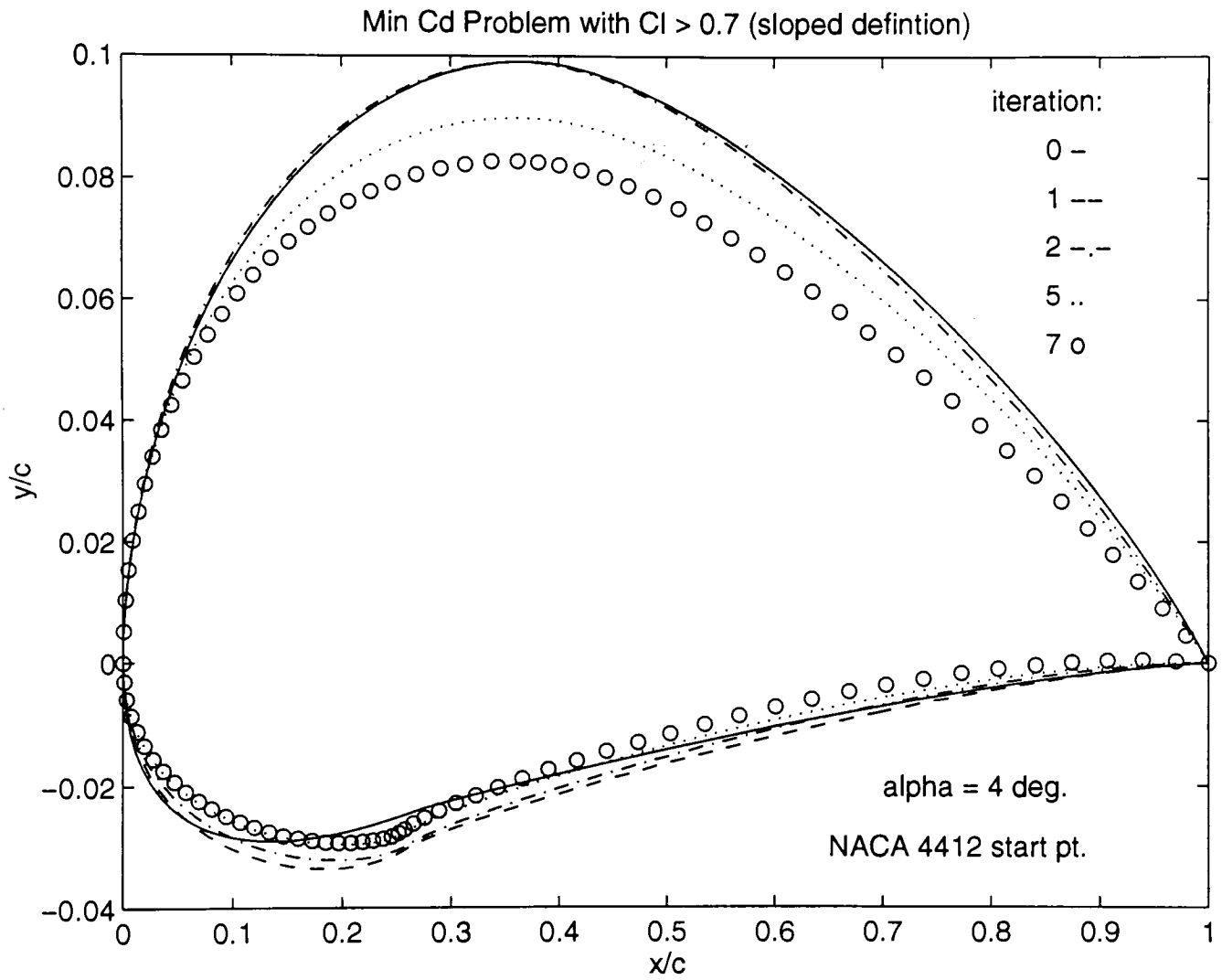


Figure 6.29:
Problem E: NACA 4412 Start Point, Scheme 2

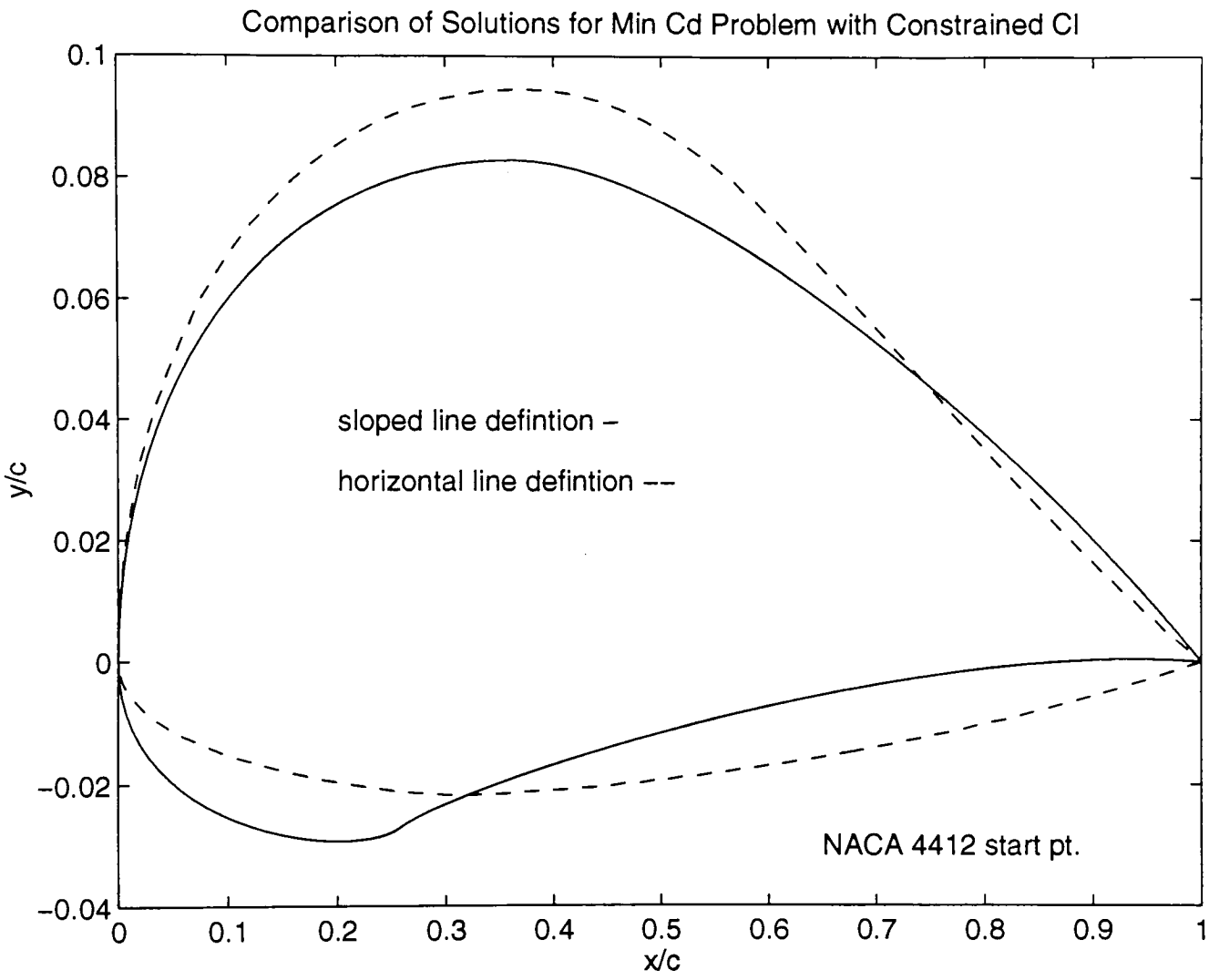


Figure 6.30:
Problem E: NACA 4412 Start Point Comparison

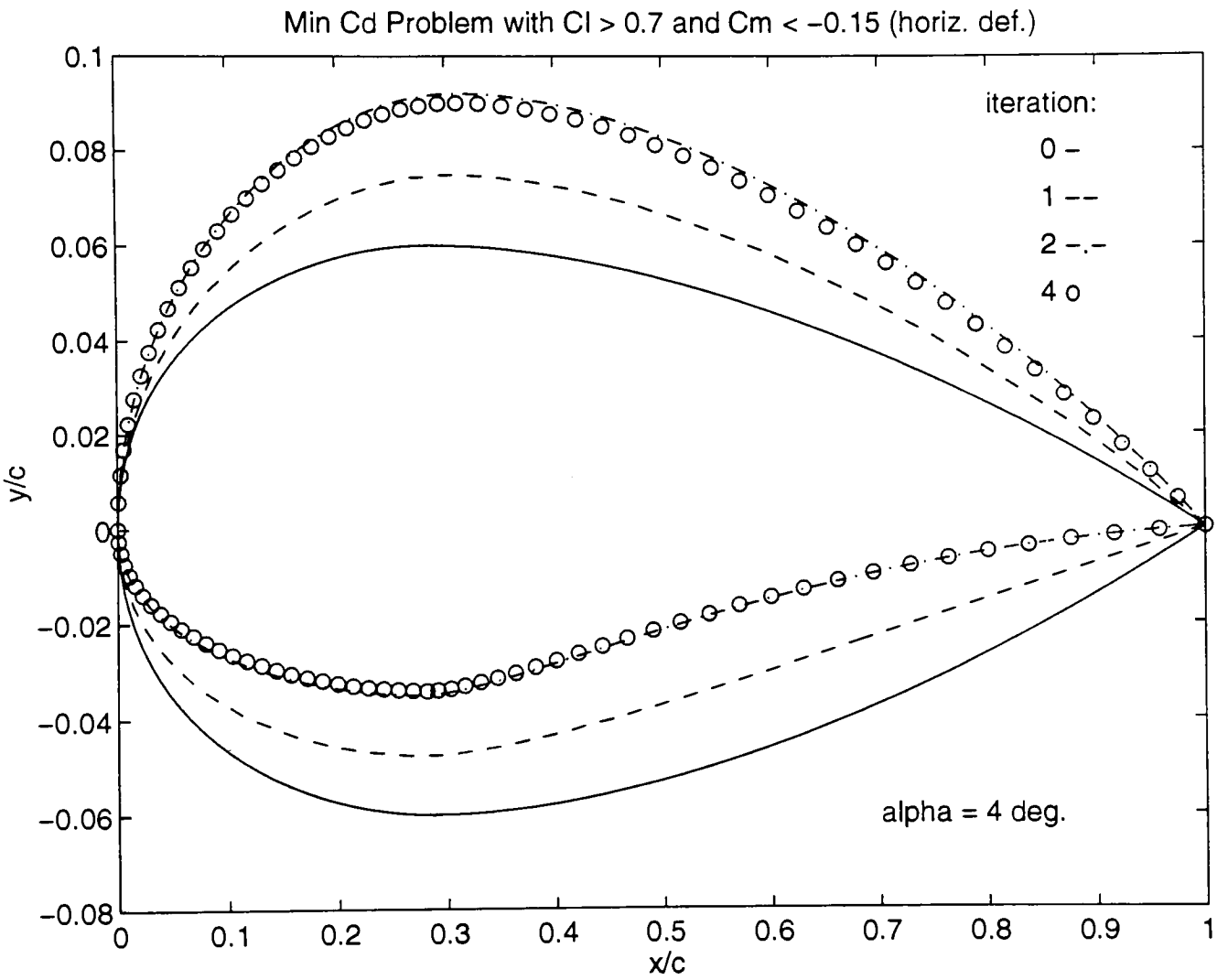


Figure 6.31:
Problem F: NACA 0012 Start Point, Scheme 1

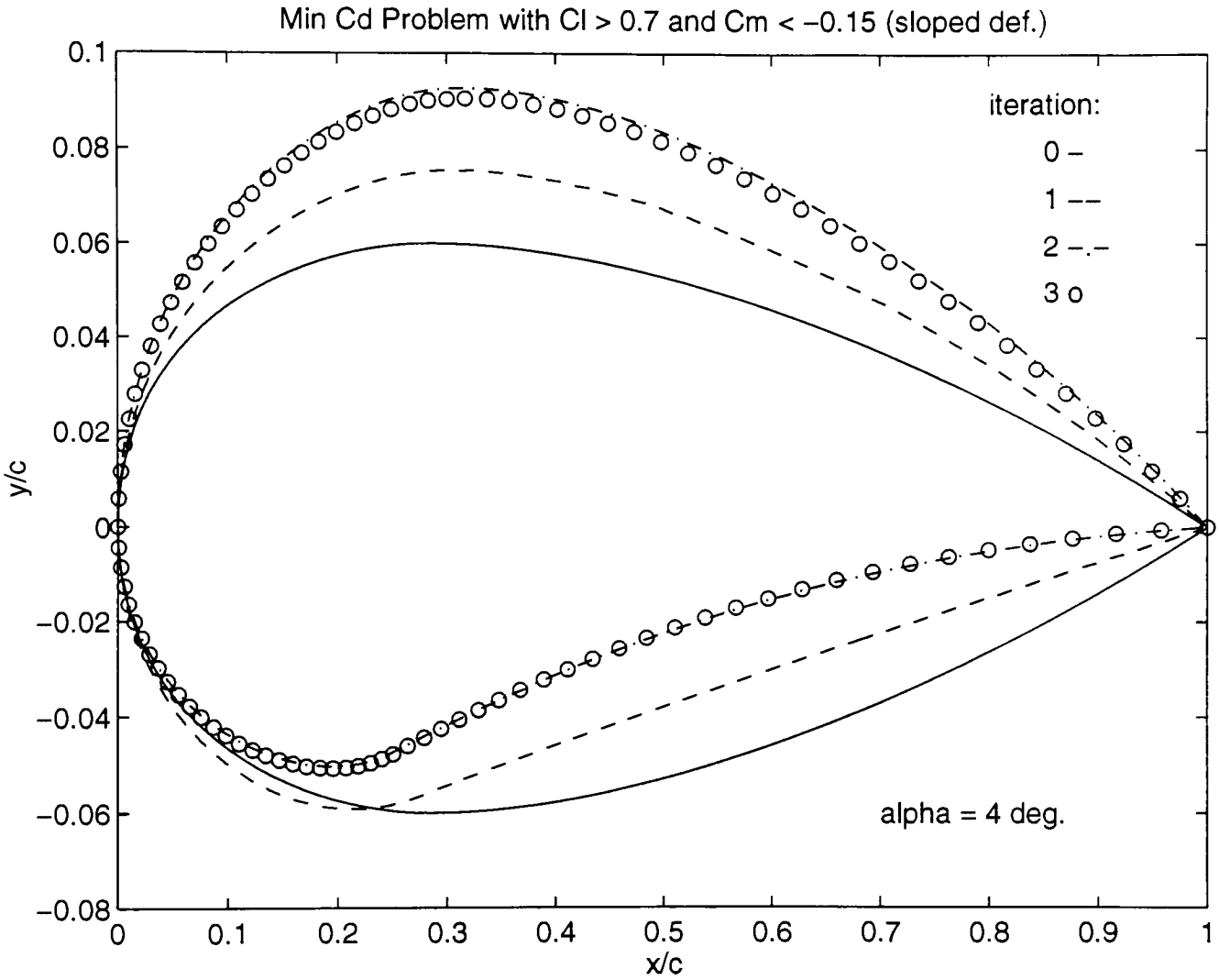


Figure 6.32:
Problem F: NACA 0012 Start Point, Scheme 2

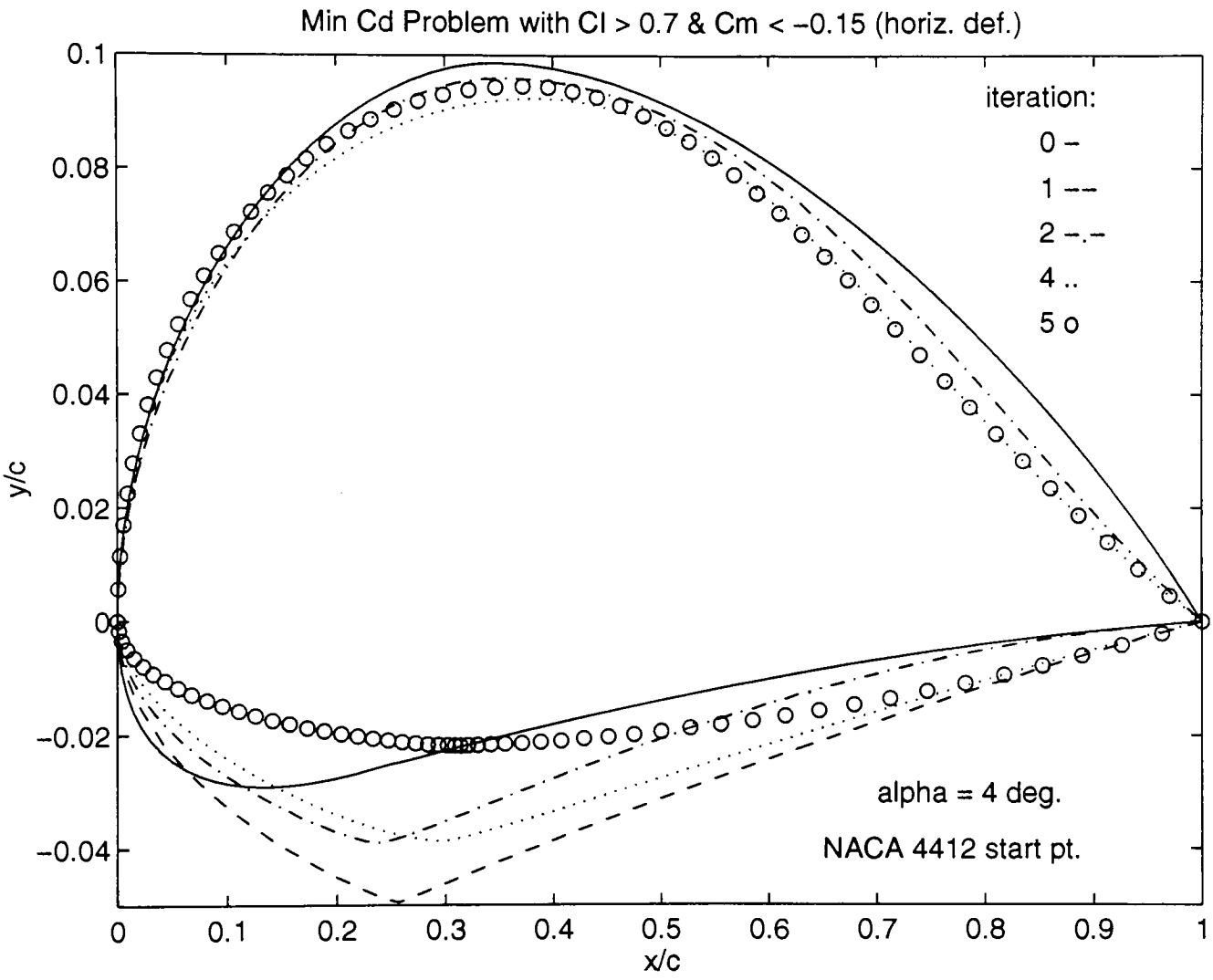


Figure 6.33:
Problem F: NACA 4412 Start Point, Scheme 1

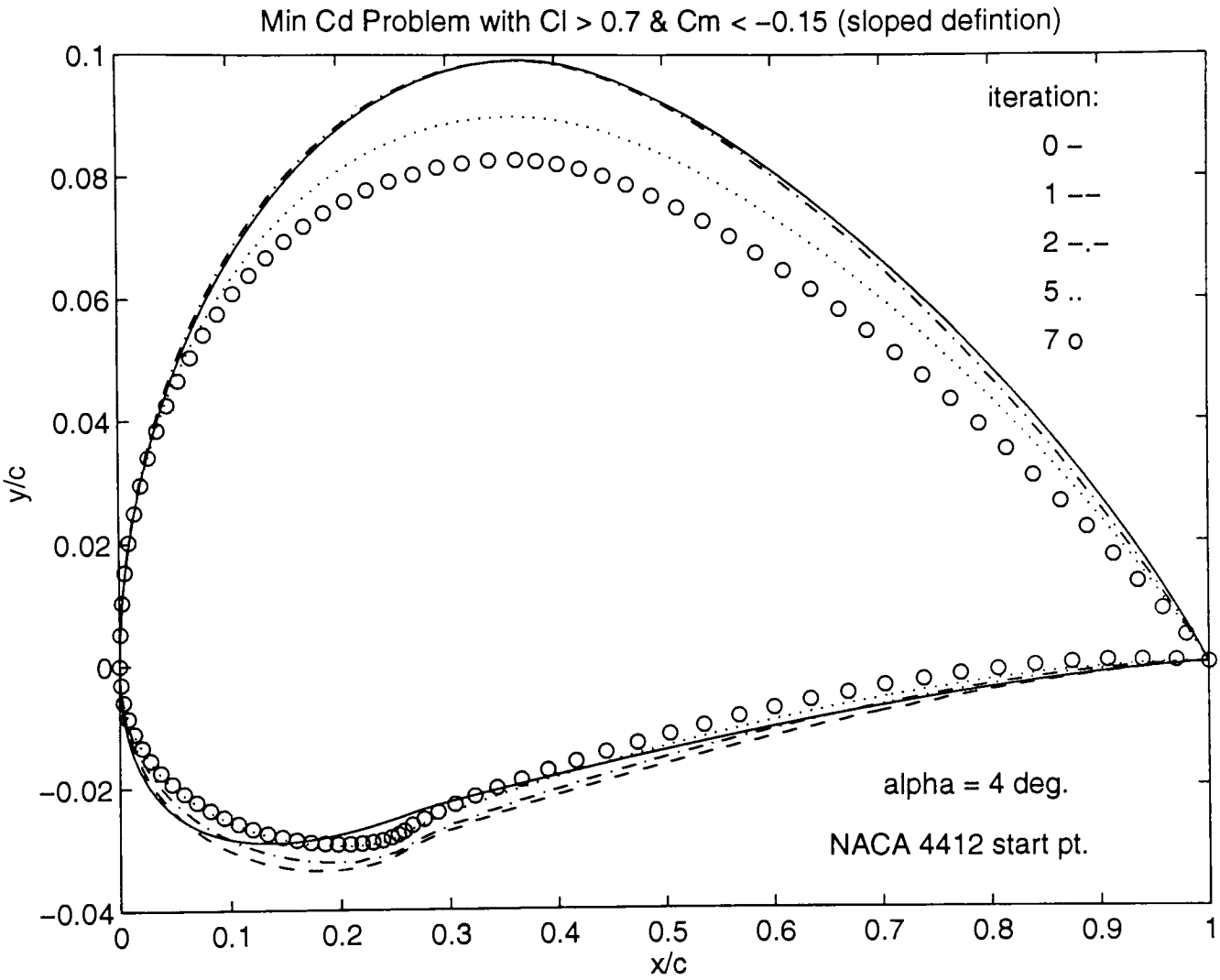


Figure 6.34:
Problem F: NACA 4412 Start Point, Scheme 2

Table 6.1: Problem A: NACA 0012 Start Point, Scheme 1

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00312	0.00362	0.858	0.764	-0.299	-0.260
2	0.00310	0.00386	0.901	0.819	-0.318	-0.284
3	0.00452	0.00638	1.101	1.013	-0.383	-0.348
4	0.00480	0.0123	1.233	1.422	-0.431	-0.519

Table 6.2: Problem A: NACA 0012 Start Point, Scheme 2

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00483	0.00516	0.830	0.731	-0.290	-0.248
2	0.00490	0.00534	0.866	0.770	-0.306	-0.266
3	0.00485	0.00815	1.103	1.081	-0.450	-0.400
4	0.00491	0.0150	1.124	1.276	-0.393	-0.462
5	0.00492	0.0178	1.324	1.324	-0.391	-0.480
6	0.00492	0.0178	1.326	1.326	-0.391	-0.481

Table 6.3: Poblem A: NACA 4412 Start Point, Scheme 1

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00514	0.00715	0.922	0.853	-0.331	-0.303
2	0.00524	0.00718	0.918	0.849	-0.329	-0.301
3	0.00518	0.00718	0.915	0.846	-0.327	-0.300
4	0.00517	0.00874	1.396	1.342	-0.575	-0.522
5	0.00522	0.0105	1.712	1.643	-0.737	-0.707

Table 6.4: Problem A: NACA 4412 Start Point, Scheme 2

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00552	0.00723	0.992	0.951	-0.363	-0.349
2	0.00522	0.00837	1.235	1.157	-0.488	-0.454
3	0.00533	0.00835	1.398	1.358	-0.573	-0.577
4	0.00543	0.0103	1.708	1.663	-0.733	-0.714
5	0.00543	0.0105	1.822	1.770	-0.787	-0.765

Table 6.5: Problem B: NACA 0012 Start Point, Scheme 1

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00312	0.00362	0.858	0.763	-0.299	-0.260
2	0.00310	0.00386	0.901	0.819	-0.318	-0.284
3	0.00307	0.00387	0.904	0.825	-0.319	-0.286
4	0.00304	0.00397	0.918	0.843	-0.325	-0.294
5	0.00504	0.00608	1.004	0.900	-0.372	-0.329
6	0.00502	0.00582	0.993	0.869	-0.365	-0.312
7	0.00515	0.00678	1.061	0.964	-0.386	-0.346
8	0.00306	0.00470	1.049	0.939	-0.378	-0.331
9	0.00473	0.00651	1.052	0.996	-0.374	-0.353
10	0.00461	0.00700	1.101	1.060	-0.391	-0.377
11	0.00461	0.00700	1.101	1.060	-0.391	-0.377

Table 6.6: Problem B: NACA 0012 Start Point, Scheme 2

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00483	0.00516	0.830	0.731	-0.290	-0.248
2	0.00489	0.00534	0.866	0.770	-0.306	-0.266
3	0.00490	0.00694	1.102	1.016	-0.407	-0.372
4	0.00489	0.00700	1.098	1.032	-0.404	-0.378
5	0.00487	0.00700	1.101	1.042	-0.404	-0.381

Table 6.7: Problem B: NACA 4412 Start Point, Scheme 1

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00578	0.00700	0.953	0.888	-0.345	-0.319
2	0.00521	0.00700	0.949	0.885	-0.342	-0.317
3	0.00511	0.00693	0.946	0.882	-0.341	-0.316
4	0.00501	0.00669	1.061	0.987	-0.396	-0.365

Table 6.8: Problem B: NACA 4412 Start Point, Scheme 2

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00552	0.00723	0.992	0.951	-0.363	-0.349
2	0.00531	0.00671	0.972	0.936	-0.351	-0.339
3	0.00538	0.00700	1.097	1.055	-0.413	-0.398
4	0.00543	0.00700	1.204	1.158	-0.465	-0.448

Table 6.9: Problem D: NACA 0012 Start Point, Scheme 1

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00519	0.00612	0.481	0.397	-0.124	-0.090
2	0.00516	0.00605	0.484	0.400	-0.125	-0.091
3	0.00437	0.00541	0.534	0.460	-0.142	-0.111
4	0.00414	0.00416	0.628	0.522	-0.170	-0.125
5	0.00412	0.00412	0.629	0.525	-0.170	-0.126

Table 6.10: Problem D: NACA 0012 Start Point, Scheme 2

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00519	0.00612	0.481	0.397	-0.124	-0.090
2	0.00513	0.00601	0.484	0.400	-0.125	-0.091
3	0.00543	0.00652	0.542	0.462	-0.147	-0.114
4	0.00411	0.00528	0.560	0.489	-0.152	-0.122
5	0.00422	0.00438	0.659	0.565	-0.186	-0.147
6	0.00401	0.00406	0.674	0.572	-0.191	-0.147
7	0.00402	0.00402	0.670	0.568	-0.187	-0.145

Table 6.11: Problem D: NACA 4412 Start Point, Scheme 1

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00519	0.00715	0.922	0.853	-0.331	-0.303
2	0.00510	0.00661	0.887	0.832	-0.311	-0.291
3	0.00509	0.00650	0.880	0.827	-0.308	-0.288
4	0.00509	0.00650	0.880	0.827	-0.308	-0.288
5	0.00540	0.00540	0.763	0.645	-0.257	-0.207
6	0.00409	0.00453	0.750	0.673	-0.240	-0.208
7	0.00405	0.00405	0.803	0.703	-0.265	-0.223
8	0.00377	0.00378	0.733	0.688	-0.230	-0.213
9	0.00377	0.00377	0.733	0.688	-0.230	-0.213

Table 6.12: Problem D: NACA 4412 Start Point, Scheme 2

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00552	0.00723	0.992	0.951	-0.363	-0.349
2	0.00506	0.00555	0.883	0.813	-0.298	-0.270
3	0.00511	0.00511	0.847	0.760	-0.283	-0.247
4	0.00514	0.00514	0.848	0.743	-0.286	-0.242

Table 6.13: Problem E: NACA 0012 Start Point, Scheme 1

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00506	0.00521	0.834	0.756	-0.276	-0.245
2	0.00493	0.00498	0.851	0.778	-0.282	-0.253
3	0.00493	0.00495	0.850	0.777	-0.281	-0.253

Table 6.14: Problem E: NACA 0012 Start Point, Scheme 2

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00591	0.00602	0.820	0.742	-0.276	-0.245
2	0.00487	0.00504	0.803	0.740	-0.256	-0.231
3	0.00488	0.00489	0.805	0.734	-0.258	-0.230
4	0.00487	0.00488	0.805	0.735	-0.258	-0.230

Table 6.15: Problem E: NACA 4412 Start Point, Scheme 1

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00519	0.00715	0.922	0.853	-0.331	-0.303
2	0.00510	0.00571	0.850	0.796	-0.288	-0.268
3	0.00500	0.00501	0.764	0.700	-0.241	-0.217
4	0.00500	0.00500	0.765	0.701	-0.241	-0.217
5	0.00453	0.00473	0.771	0.713	-0.244	-0.219
6	0.00397	0.00397	0.793	0.728	-0.246	-0.220

Table 6.16: Problem E: NACA 4412 Start Point, Scheme 2

Iteration	Skin Friction	Real Drag	Ideal Lift	Real Lift	Ideal Moment	Real Moment
1	0.00552	0.00723	0.992	0.951	-0.363	-0.349
2	0.00524	0.00637	0.972	0.939	-0.350	-0.339
3	0.00522	0.00620	0.968	0.936	-0.347	-0.337
4	0.00520	0.00606	0.965	0.934	-0.346	-0.336
5	0.00505	0.00579	0.963	0.932	-0.344	-0.334
6	0.00532	0.00537	0.870	0.796	-0.299	-0.269
7	0.00528	0.00532	0.865	0.795	-0.297	-0.268

7. CONCLUSIONS AND RECOMMENDATIONS

The objectives of the current work were fulfilled in that (1) a mathematical model was developed to analyze the flow about an airfoil that incorporated boundary layer effects, (2) the code was incorporated into an optimizer and optimal airfoils were found for all the described cases, and (3) a CFD model was generated to verify the results of the optimal solutions. For the most part, the work was successful. But, as can be expected with any large project, some problems were encountered. Also, it was determined that much work could still be done but would be beyond the scope of the present thesis.

Probably the most deficient part of the project was the analysis model. Many problems were experienced while running optimization models. For instance, the boundary layer calculator would sometimes have difficulties calculating the conditions for the infeasible designs proposed during search direction determination and the code would subsequently crash. This is of some concern because the hybrid GRG-SQP method used, usually takes small steps in the infeasible direction and the returns to an active constraint boundary. This did not occur often enough that it hindered the progress of the work but it is an area that should be corrected. It would be desirable to incorporate some type of mechanism that, before the code was allowed to crash, informed OptdesX of the ill-conditioned problem and made to either change search directions or reduce step sizes.

Since the moment coefficients turned out not to affect the solution, some time could be spent trying to determine a more suitable way of controlling pitching moment. Even when the constraint was active it quickly became inactive with increasing lift. One

possible way to approach this is to set upper and lower limits on the moment coefficient, in affect, making the pitching moment a side constraint.

Another thing that could be done is to develop a postprocessor routine that would save all the required information at every iteration without having to physically stop the optimization. Currently, optimization runs are made one iteration at a time so that results can be saved. This not only slows the optimization process because calculations are stopped but also because information from previous iterations is lost, hence, OptdesX is not allowed to take full advantage of the variable metric update used in search direction determination.

Other work that could be done includes investigating different angles of attack. The current work only looked at a four degree angle of attack. Also, different Reynolds numbers could be investigated.

Despite the mentioned deficiencies, some valuable information was acquired from the project. For instance, it was determined that the optimal solutions were highly dependent on the staring airfoil shape. As illustrated in the result plots, the optimal sections were not limited to airfoils close in shape to the starting point. This is a notable improvement over many existing methods. Also, it was found that different geometry definition methods could be used for different types of problems. For example, the sloped line definition method could be used if a high lift airfoil is desired and the horizontal line definition method could be used if low drag airfoils are required.

The optimization results were very successful from an optimization standpoint as well as a design standpoint. For the unconstrained maximum lift problem, the objective

was improved upon by as much as 306 percent. Likewise drag was improved upon by as much as 45.7 percent. Also the results were proven to be accurate within a reasonable degree through the use of CFD. Overall, the method was successful but left room for improvement.

LIST OF REFERENCES

1. Eppler, R., and Somers, D. M., *A Computer Program for the Design and Analysis of Low Speed Airfoils*, NASA TM 80210, 1980.
2. Vanderplaats, G. N., Hicks, R. N., and Murman, E. M., *Applications of Numerical Optimization Techniques to Airfoil Design*, NASA Ames Research Center, NASA SP-347, Part II, pp. 749-768, March, 1975.
3. Vanderplaats, G. N., and Hicks, R. N., *Numerical Optimization Using a Reduced Number of Design Variables*, NASA Ames Research Center, NASA TM X-73, 151, July 1976.
4. Liebeck, R. H., and Ormsbee, A. J., *Optimization of Airfoils for Maximum Lift*, Journal of Aircraft, Vol. 7, No. 5, 1969.
5. Coiro, D. P., and Nicolosi, F., *Design and Optimization of Glider Components*, Technical Soaring, Vol. XIX, No. 2, 1995.
6. Chang, I. C., et. al., *Optimization of the Wing-Body Configuration by the Euler Equations*, AIAA 94-1899, 12th AIAA Applied Aerodynamics Conference, 1994.
7. Venkataraman, P., *A New Procedure for Airfoil Definition*, Paper AIAA 95-1875, 13th AIAA Applied Aerodynamics Conference, San Diego, CA., June 1995.
8. Parkinson, A., et. al., *OptdesX - A Software System for Optimal Engineering Design*, Design Synthesis Inc., Provo, Utah, 1992.
9. Vanderplaats, G. N., *Numerical Optimization Techniques for Engineering Design, with Applications*, McGraw-Hill Series in Mechanical Engineering, 1984.
10. Parkinson, A., and Wilson, M., *Development of a Hybrid GRG-SQP Algorithm for Constrained Nonlinear Programming*, J. of Mech. Trans. and Automation in Design, Trans. ASME, Vol. 110, pp. 308, Sep. 1988.
11. Rogers, D. F., and Adams, J. A., *Mathematical Elements for Computer Graphics*, McGraw-Hill, 1990.

12. Moran, J., *An Introduction to Theoretical and Computational Aerodynamics*, John Wiley and Sons, 1984.

13. Abbot, I. H., and Von Doenhoff, A. E., *Theory of Wing Sections*, Dover Publications, 1959.

14. Venkataraman, P., *A Design Optimization Technique for Airfoil Shapes*, Paper AIAA 94-1813, 12th AIAA Applied Aerodynamics Conference, Colorado Springs, CO., June 1994.

15. *Fluent User's Guide v4.2*, Fluent Inc., Lebanon, NH, 1993.

Appendix I: FORTRAN Program of Analysis Model

```
c=====7==================================================================2===
c...subroutine anapre
c      Preprocessing Routine
c-------------------------------------------------------------------------
      subroutine anapre(modelN)
      character*17 modelN
c...set model name (16 chars max)

      modelN='Optimum Airfoil'
      return
      end

c=====7==================================================================2===
c...subruotine anafun
c      Analysis Routine
c-------------------------------------------------------------------------
      subroutine anafun
c....
c       Airfoil is described using Bezier Polygons.
c       Hess-Smith-Douglas Method is used for initial source panels
c       and a single vortex distribution (MORAN)
cc
cc      Boundary layer transition, seperation are predicted using
cc      Twhaites Method, Michael's method, and Head's Method (MORAN)
c
c       There are fourteen design variables
c
c       There are 22 constraints - all linear
cc      enforcing geometrical compatibility in the definition of
c       airfoil geometry
cc
cc      by Venkat- September 30, 1993
c
c       parameter ntotal=200, pi=3.1415926565
        implicit real*8(a-h,o-z)

c       dimension xa(210), ya(210), xinit(210), yinit(210)
        dimension xa(210), ya(210)
c       dimension dx(210), dy(210), sinthe(210), costhe(210)
        dimension sinthe(210), costhe(210)
c       dimension dist(210), xmid(210), ymid(210), a(210,210)
        dimension dist(210), xmid(210), ymid(210)
c       dimension cp(210), vtan(210), vgrad(210), theta(210)
        dimension cp(210), vtan(210)
c       dimension c (210)
c
c       common/design/ xt0,yt0,xt1,yt1,xt2,yt2,xt3,yt3,
c    *                 xt4,yt4,xt5,yt5,xt6,yt6,
c    *                 xb7,yb7,xb8,yb8,xb9,yb9,
c    *                 xb10,yb10,xb11,yb11

        common/envdata/ alpha, vfree, anufree, rhofree, chord,
     *                 pref, calf, salf

        common/param/ntotal, nseg, nact, nupper, nlower, npanels,
     *                pi, pinv, nact1
```

```
      common/ac/a(210,210),c(210)
      common/rey/ re
      common/cd/ cdf
      common/hfact/ h
      common/dxdy/ dx(210),dy(210)
      common/vc/ vcl,vcd,vcm
      common/xc/ xcl,xcd,xcm

      ntotal = 210
      pi = 3.1415926585
      pinv = 0.5/pi
      ifirst = 1
cc    define the Bezier Vertices as design variables

c...get AV values from OptdesX (Variable name 16 chars max)
cc      call avdsca(?var,'?VariableName')

        call avdsca(xt0,'x0-front')
        print *,'xt0 = ',xt0
        call avdsca(yt0,'y0-front')
        call avdsca(xt1,'x1-top left')
        call avdsca(yt1,'y1-top left')
        call avdsca(xt2,'x2-top left')
        call avdsca(yt2,'y2-top left')
        call avdsca(xt3,'x3-top center')
        call avdsca(yt3,'y3-top center')
        call avdsca(xt4,'x4-top right')
        call avdsca(yt4,'y4-top right')
        call avdsca(xt5,'x5-top right')
        call avdsca(yt5,'y5-top right')
        call avdsca(xt6,'x6-back')
        call avdsca(yt6,'y6-back')
        call avdsca(xb7,'x7-bottom right')
        call avdsca(yb7,'y7-bottom right')
        call avdsca(xb8,'x8-bottom right')
        call avdsca(yb8,'y8-bottom right')
        call avdsca(xb9,'x9-bottom center')
        call avdsca(yb9,'y9-bottom center')
        call avdsca(xb10,'x10-bottom left')
        call avdsca(yb10,'y10-bottom left')
        call avdsca(xb11,'x11-bottom left')
        call avdsca(yb11,'y11-bottom left')
c
        write(6,*) 'xt0:',xt0
        write(6,*) 'yt0:',yt0
        write(6,*) 'xt1:',xt1
        write(6,*) 'yt1:',yt1
        write(6,*) 'xt2:',xt2
        write(6,*) 'yt2:',yt2
        write(6,*) 'xt3:',xt3
        write(6,*) 'yt3:',yt3
        write(6,*) 'xt4:',xt4
        write(6,*) 'yt4:',yt4
        write(6,*) 'xt5:',xt5
        write(6,*) 'yt5:',yt5
        write(6,*) 'xt6:',xt6
        write(6,*) 'yt6:',yt6
        write(6,*) 'xb7:',xb7
        write(6,*) 'yb7:',yb7
        write(6,*) 'xb8:',xb8
        write(6,*) 'yb8:',yb8
        write(6,*) 'xb9:',xb9
        write(6,*) 'yb9:',yb9
        write(6,*) 'xb10:',xb10
```

```
          write(6,*) 'yb10:',yb10
          write(6,*) 'xb11:',xb11
          write(6,*) 'yb11:',yb11

c...new variable calls
c         call avdsca(npoints,'npoints')
          call avdsca(alpha,'alpha (deg.)')
          call avdsca(vfree,'vfreen (m/s)')
          call avdsca(anufree,'anufree')
          call avdsca(rhofree,'rhofree (kg/m^3)')
          call avdsca(chord,'chord (m)')
          call avdsca(pref,'pref (Pa)')
          call avdsca(re,'reynolds #')
          call avdsca(h,'shape factor, h')

c...send functions to OptdesX (Function names 16 chars max)
cc        call afdsca(?fun,'?Function Name')

c     the functional constraints are geometric to provide an acceptable
c     shape definition. Constraints are set up so that they are positive.
c
c     the ordinates yt3,yt4,yb9,yb10 are not design variables. This
c     allows the location of the maximum points on the top and bottom
c     surface.

      yt3 = yt2
      yt4 = yt2
c     yb9 = yb8
c     yb10 = yb8

c       constrain (xb10,yb10) such that it lies on the line defined
c       by (xb9,yb9) and (xb8,yb8)

      slope = (yb8-yb9)/(xb8-xb9)
      yb10 = slope*(xb10-xb8)+yb8

c     constraints on the absisca's

      xt12 = xt2-xt1
      xt32 = xt3-xt2
      xt43 = xt4-xt3
      xt54 = xt5-xt4
      xt65 = xt6-xt5

      xb67 = xt6-xb7
      xb78 = xb7-xb8
      xb89 = xb8-xb9
      xb910 = xb9-xb10
      xb1011 = xb10-xb11

c     constraints on the ordinates

      yt10 = yt1-yt0
      yt21 = yt2-yt1
      yt45 = yt4-yt5
      yt56 = yt5-yt6

      yb011 = yt0-yb11
      yb1110 = yb11-yb10
      yb78 = yb7-yb8
      yb67 = yt6-yb7

      yb89 = yb8-yb9
      yb910 = yb9-yb10
```

```
      call afdsca(yt3,'yt3=yt2')
      call afdsca(yt4,'yt4=yt2')
c     call afdsca(yb9,'yb9=yb8')
c     call afdsca(yb10,'yb10=yb8')
      call afdsca(yb10,'yb10 on line')

      call afdsca(xt12,'abscissca 1-2')
      call afdsca(xt32,'abscissca 3-2')
      call afdsca(xt43,'abscissca 4-3')
      call afdsca(xt54,'abscissca 5-4')

      print *,'four functions called...'

      call afdsca(xt65,'abscissca 6-5')
      call afdsca(xb67,'abscissca 6-7')
      call afdsca(xb78,'abscissca 7-8')
      call afdsca(xb89,'abscissca 8-9')

      print *,'another four functions called...'

      call afdsca(xb910,'abscissca 9-10')
      call afdsca(xb1011,'abscissca 10-11')
      call afdsca(yt10,'ordinate 0-1')
      call afdsca(yt21,'ordinate 2-1')

      print *,'another four functions called...'

      call afdsca(yt45,'ordinate 4-5')
      call afdsca(yt56,'ordinate 5-6')
      call afdsca(yb011,'ordinate 0-11')
      call afdsca(yb1110,'ordinate 10-11')

      print *,'another four functions called...'

      call afdsca(yb78,'ordinate 7-8')
      call afdsca(yb67,'ordinate 6-7')

      call afdsca(yb89,'ordinate 8-9')
      call afdsca(yb910,'ordinate 9-10')

      print *,'another two functions called...'

c     call afdsca(yt23,'equality y 2-3')
c     call afdsca(yt34,'equality y 3-4')
c     print *,'another two functions called'
c     call afdsca(yb89,'equality y 8-9')
c     call afdsca(yb910,equality y 9-10')
c     print *,'yb910 = ',yb910


cc
cc    obtain airfoil coordinates from subroutine coord(xa,ya)
cc
cc    call coord(xa,ya)
cc
c     call coord(xa,ya,npoints)
c     call coord(xa,ya)

      nwhich = 0
      if(nwhich.eq.0)then
       call coord(xa,ya,xt0,yt0,xt1,yt1,xt2,yt2,xt3,yt3,
     *                  xt4,yt4,xt5,yt5,xt6,yt6,
     *                    xb7,yb7,xb8,yb8,xb9,yb9,
     *                   xb10,yb10,xb11,yb11)
```

```
      write(6,*) 'called and returned from coord(xa,ya)'
     else
      open(unit=14,file='input.dat',status='old')
 20   read(14,*,end=30) i,xa(i),ya(i)
       nact = nact + 1
       if(xa(i).eq.0.0) nlower = i-1
       go to 20
 30   continue
      npanels = nact-1
      nupper = npanels-nlower
      write(6,*) 'nact,npanels',nact,npanels
      write(6,*) 'nupper,nlower',nupper,nlower
      close(unit=14)
     endif

     write(6,*) 'about to write the airfoil coordinates'
     open(unit=12,file="coord.dat",status="unknown")
     do 100 i = 1, nact
      write(12,*) i,xa(i),ya(i)
c     write(6,*) i,xa(i),ya(i)
 100 continue
     close(unit=12)
     print *,'wrote airfoil coordinates'

cc
cc

c    if (ifirst.eq.1) then
c        open (unit=15,file="initial.dat",status='unknown')
c        else
c        open (unit=11,file="inbet.dat",status = 'unknown')
c    endif

c    write(6,*) 'about to write the airfoil coordinates'

c    do 10 i=1,nact
c     xinit(i) = xa(i)
c     yinit(i) = ya(i)
c     write(6,*) xa(i),ya(i)
c     write(15,*) xa(i),ya(i)
c     if (ifirst.eq.1) write(10,*) xa(i),ya(i)
c     if (ifirst.ne.1) write(11,*) xa(i),ya(i)
c10  continue
c    if (ifirst.eq.1) close(unit = 10,keep)
c    if (ifirst.ne.1) write(11,15)
c15  format(//////)

c    print *,'wrote airfoil coordinates'

c    nact = nact+1
c    ifirst = ifirst+1

cc   calculate objective function -- max lift coefficient
cc                                -- min drag coefficient
cc
cc   the aerodynamics starts here
cc   subroutines used:
cc       ambient
cc       pslope(xa,ya,dx,dy,dist,sinthe,costhe)
cc       coeffnt(xa,ya,sinthe,costhe,a,c,nact1,xmid,ymid)
cc       solsys(a,c,nact,ntotal,ind)
cc       veldist(xa,ya,xmid,ymid,dx,dy,costhe,sinthe,a,nact1,cp,vtan)
cc       clcdcm(dx,dy,cp,cl,cd,cm)
cc       intgrl(xa,ya,vtan,vgrad,theta) - to be done
```

```
cc
cc    these subroutines are found in filename ='aerodyn.f'
cc
cc    get ambient conditions
cc
      call ambient
      write(6,*) 'returned from subroutine ambient'

cc
cc    set up panel length and orientation
cc
c     call psslope(xa,ya,dx,dy,dist,sinthe,costhe)
      call psslope(xa,ya,dist,sinthe,costhe)
      write(6,*) 'returned form psslope'

cc
cc    set up the influence coefficients
cc
c     call coeffnt(xa,ya,sinthe,costhe,a,c,xmid,ymid)
      call coeffnt(xa,ya,sinthe,costhe,xmid,ymid)
      write(6,*) 'returned form coeffnt'


cc
cc    solve the linear system using gauss elimination
cc
c     call solsys(a,c,nact,ntotal,ind)
      call solsys(nact,ntotal,ind)
      write(6,*) 'returned from so1sys'

cc
      if (ind.eq.1) write(6,201)
 201  format(//5x,'The system is singular')
cc
cc
cc    obtain velocity distribution and pressure coefficients
cc
c     call veldist(xa,ya,xmid,ymid,costhe,sinthe,a,c,cp,vtan)
c     call veldist(xa,ya,xmid,ymid,costhe,sinthe,cp,vtan)
      call veldist(xa,ya,xmid,ymid,costhe,sinthe,loop,cp,vtan)
      write(6,*) 'returned from veldist'

cc
cc    calculate aerodynamic coefficients
cc
c     call clcdcm(dx,dy,xmid,ymid,cp,cl,cd,cm)
      call clcdcm(xmid,ymid,cp,loop)
      write(6,*) 'returned from clcdcm'
cc
cc    others to follow from here
cc

      write(6,*) 'xcl,xcd,xcm:'
      write(6,*) xcl,xcd,xcm

      call afdsca(cdf,'skin friction')
      call afdsca(xcd,'ideal drag coeff')
      call afdsca(vcd,'real drag coeff')
      call afdsca(xcl,'ideal lift coeff')
      call afdsca(vcl,'real lift coeff')
      call afdsca(xcm,'ideal mom. coeff')
      call afdsca(vcm,'real mom. coeff')

      print *,'returned from calculating cd,cl,cm'
```

```
      return
      end

c=====================================================================
c...subroutine anapos
c      Postprocessing Routine
c---------------------------------------------------------------------
-----
      subroutine anapos
      return
      end
```

```
c=======================================================================
c      subroutine coord(x,y,npoints)
c      subroutine coord(x,y)
       subroutine coord(x,y,xt0,yt0,xt1,yt1,xt2,yt2,xt3,yt3,
     *                  xt4,yt4,xt5,yt5,xt6,yt6,
     *                    xb7,yb7,xb8,yb8,xb9,yb9,
     *                   xb10,yb10,xb11,yb11)
cc
cc     subroutine to obtain airfoil coordinates from Bezier
cc     geometry description. to be accessed by OptdesX for
cc     airfoil optimization.
cc
cc     number of Bezier segments = nseg = 4
cc     max number of ordinates = ncord = 200
cc     number of points in each segment = npoints = 25
cc     number of actual coordinates = nact
cc     to conform to panel definition later, the airfoil
cc     coordinates are generated clockwise starting from the
cc     bottom left segment
cc
cc     programmed by venkat, 10/29/93
cc     curves are generated using
cc
cc         [v**3 v**2 v 1 ][ -1  3 -3  1 ][ b11  b12]
cc                         [  3 -6  3  0 ][ b21  b22]
cc                         [ -3  3  0  0 ][ b31  b32]
cc                         [  1  0  0  0 ][ b41  b42]
cc
cc     v ;parameter ; (b?1,b?2) ; vertices of the polygon
cc
cc


       implicit real*8(a-h,o-z)
       parameter npoints = 28
       dimension x(1), y(1), b(4,4,2)
       dimension coeff(4,4),param(1,4),vert(4,2),xy(1,2),bet(4,2)
cc
c      common/design/ xt0,yt0,xt1,yt1,xt2,yt2,xt3,yt3,
c     *                xt4,yt4,xt5,yt5,xt6,yt6,
c     *                xb7,yb7,xb8,yb8,xb9,yb9,
c     *                xb10,yb10,xb11,yb11
c
       common/param/ntotal,nseg,nact,nupper,nlower,npanels,
     *                     pi,pinv,nact1
c
c      npoints=28
c
c          write(6,*) 'xt0:',xt0
c          write(6,*) 'yt0:',yt0
c          write(6,*) 'xt1:',xt1
c          write(6,*) 'yt1:',yt1
c          write(6,*) 'xt2:',xt2
c          write(6,*) 'yt2:',yt2
c          write(6,*) 'xt3:',xt3
c          write(6,*) 'yt3:',yt3
c          write(6,*) 'xt4:',xt4
c          write(6,*) 'yt4:',yt4
c          write(6,*) 'xt5:',xt5
```

```
c          write(6,*) 'yt5:',yt5
c          write(6,*) 'xt6:',xt6
c          write(6,*) 'yt6:',yt6
c          write(6,*) 'xb7:',xb7
c          write(6,*) 'yb7:',yb7
c          write(6,*) 'xb8:',xb8
c          write(6,*) 'yb8:',yb8
c          write(6,*) 'xb9:',xb9
c          write(6,*) 'yb9:',yb9
c          write(6,*) 'xb10:',xb10
c          write(6,*) 'yb10:',yb10
c          write(6,*) 'xb11:',xb11
c          write(6,*) 'yb11:',yb11
c
      write(6,*) 'in coord'
      write(6,*) 'npoints:',npoints
cc
      nseg = 4
cc
cc    read the coefficient matrix
cc
      do 10 i=1,4
       do 10 j=1,4
        coeff(i,j) = 0.0
 10   continue
      coeff(1,1) = -1.0
      coeff(1,2) = 3.0
      coeff(1,3) = -3.0
      coeff(1,4) = 1.0
      coeff(2,1) = 3.0
      coeff(2,2) = -6.0
      coeff(2,3) = 3.0
      coeff(3,1) = -3.0
      coeff(3,2) = 3.0
      coeff(4,1) = 1.0
cc
cc    start airfoil definition
cc    from bottom to right

      print *,'the coefficient is stored'

cc
cc    define bottom right polygn vertices
cc
      b(1,1,1) = xt6
      b(1,1,2) = yt6
      b(1,2,1) = xb7
      b(1,2,2) = yb7
      b(1,3,1) = xb8
      b(1,3,2) = yb8
      b(1,4,1) = xb9
      b(1,4,2) = yb9
cc
cc    define bottom left polygon vertices
cc
      b(2,1,1) = xb9
      b(2,1,2) = yb9
      b(2,2,1) = xb10
      b(2,2,2) = yb10
      b(2,3,1) = xb11
      b(2,3,2) = yb11
      b(2,4,1) = xt0
      b(2,4,2) = yt0
cc
```

```
cc    define top left polygon vertices
cc
      b(3,1,1) = xt0
      b(3,1,2) = yt0
      b(3,2,1) = xt1
      b(3,2,2) = yt1
      b(3,3,1) = xt2
      b(3,3,2) = yt2
      b(3,4,1) = xt3
      b(3,4,2) = yt3
cc
cc    define top right polygon vertices
cc
      b(4,1,1) = xt3
      b(4,1,2) = yt3
      b(4,2,1) = xt4
      b(4,2,2) = yt4
      b(4,3,1) = xt5
      b(4,3,2) = yt5
      b(4,4,1) = xt6
      b(4,4,2) = yt6
cc
cc    start computing upper and lower surfaces
cc

      print *,'stored the polygon vertices'

      ij = 1
      do 20 jseg = 1, nseg


cc    read the values of the vertices into vert(i,j)

      do 25 i = 1,4
      do 25 j = 1,2
       vert(i,j) = b(jseg,i,j)
       write(6,*)vert(i,j)
 25   continue

cc    call amult to multiply coeff(i,j) * vert(i,j)

      call amult(4,4,4,2,coeff,vert,bet)

      print *,'multiplied',jseg
      do 26 i = 1,4
       write(6,*)vert(i,1),vert(i,2)
 26   continue


cc
cc    construct the parameter matrix param(1,4) and calculate x,y
cc
      param(1,4) = 1.0
      if(jseg.eq.1) ii=1
      if(jseg.gt.1) ii=2


      do 30 i=ii,npoints
       vv = (float(i)-1.0)/(float(npoints)-1)
cc     write(6,*)vv
       print *,'vv =',vv
       param(1,1) = vv**3
       param(1,2) = vv**2
       param(1,3) = vv
```

```
cc
cc      call amult to calculate x and y
cc
        call amult(1,4,4,2,param,bet,xy)

        print *,'multiplication of parameter'

cc
        print *,'xy =',xy(1,1),xy(1,2)

        if(xy(1,1).eq.0.0) nlower = ij
        x(ij) = xy(1,1)
        y(ij) = xy(1,2)
        print *,'ij,x,y',ij,x(ij),y(ij)

        ij = ij +1

        print *,'ij,x,y',ij,x(ij-1),y(ij-1)

cc      write(6,*)x(in),y(in)
 100    format(1x,2f10.0)
cc
 30    continue
 20    continue
       nact = ij-1
       npanels = nact - 1
       write(6,*) 'nact',nact
cc     do 35 i = 1, nact
cc     if(x(i).eq.0.0) i = nlower
 35    continue
c      nupper = nact - nlower
       nupper = npanels - nlower
       print *, 'nupper,nlower,npanels',nupper,nlower,npanels
       print *, 'nact', nact
       return
       end

c-------------------------------------------------------------------------
       subroutine amult(i1,j1,i2,j2,a,b,c)
cc
cc     matrix multiplication
cc
       implicit real*8(a-h,o-z)
       dimension a(i1,j1),b(i2,j2),c(i1,j2)

cc

       do 20 i = 1,i1
       do 20 j = 1,j2
       c(i,j) = 0.0
       do 20 k = 1,j1
       c(i,j) = c(i,j) + a(i,k)*b(k,j)
 20    continue
       return
       end
```

```
c=======================================================================
c
c     This file contains subroutines for ideal.f
c
c=======================================================================
cc
      subroutine ambient
cc
cc    get ambient and free stream data
cc
      implicit real*8(a-h,o-z)
cc
      common/envdata/alpha,vfree,anufree,rhofree,chord,pref,
     *               calf, salf

      write(6,*) 'in ambient'
cc
      alpha = alpha*3.141592658/180.
c
      salf = dsin(alpha)
      calf = dcos(alpha)
c
      write(6,*) 'salf,calf',salf,calf
c
c     vfree = 150
c     anufree = 1.5723e-04
c     rhofree = 2.3769e-03
c     chord = 5.0
c     pref = 2116.2
cc
      print *,'alpha,..etc',alpha,pref
cc
      return
      end

c=======================================================================
c     subroutine psslope(x,y,dx,dy,dist,sinthe,costhe)
      subroutine psslope(x,y,dist,sinthe,costhe)
cc
cc    sets the slope of the panels and calculate the length
cc
      implicit real*8(a-h,o-z)
cc
c     dimension x(1),y(1),dx(1),dy(1),dist(1)
      dimension x(1),y(1),dist(1)
      dimension sinthe(1),costhe(1)
cc
      common/param/ntotal,nseg,nact,nupper,nlower,npanels,
     *             pi,pinv,nact1
      common/dxdy/ dx(210),dy(210)
cc
      do 10 i = 1,npanels
       dx(i) = x(i+1)-x(i)
       dy(i) = y(i+1)-y(i)
       dist(i) = dsqrt(dx(i)*dx(i) + dy(i)*dy(i))
       sinthe(i) = dy(i)/dist(i)
       costhe(i) = dx(i)/dist(i)
 10   continue
```

```
      print *,'dx(1),dy(1)..',dx(1),dy(1),dist(1),sinthe(1)
      print *,'dx(npanels)..',dx(npanels),dist(npanels)
      return
      end

c=======================================================================
c     subroutine coeffnt(x,y,sinthe,costhe,a,c,xmid,ymid)
      subroutine coeffnt(x,y,sinthe,costhe,xmid,ymid)
cc
cc    setting up the coefficoent matrix for the source panels
cc
      implicit real*8(a-h,o-z)
cc
      dimension x(1),y(1),sinthe(1),costhe(1)
c     dimension a(ntotal,ntotal),c(1),xmid(1),ymid(1)
      dimension xmid(1),ymid(1)
cc
      common/param/ntotal,nseg,nact,nupper,nlower,npanels,pi,
     *                pinv,nact1
      common/envdata/alpha,vfree,anufree,rhofree,chord,pref,
     *               calf, salf
      common/rhs/ tempo(210),a1(210,210)
      common/ac/ a(210,210),c(210)
cc
cc
      write(6,*) 'in coeffnt'
      write(6,*) 'nact:',nact
      write(6,*) 'salf,calf',salf,calf
c
      open(unit=11,file="theta1.dat",status='unknown')
      do 160 i = 1, nact
       write(11,*) i,sinthe(i),costhe(i)
c      write(6,*) 'i,sinthe(i),costhe(i):'
c      write(6,*) i,sinthe(i),costhe(i)
 160  continue
      close(unit=11)
cc
cc    initialize coefficients
cc
      do 90 j = 1,nact
       a(nact,j) = 0.0
 90   continue
cc
cc    initialize the velocity at the midpoints of i panel to zero
cc
      open(unit=12,file='mid2.dat',status='unknown')
      do 120 i = 1,npanels
       xmid(i) = 0.5*(x(i) + x(i+1))
       ymid(i) = 0.5*(y(i) + y(i+1))
       write(12,*) xmid(i),ymid(i)
       a(i,nact) = 0.0
cc
cc
cc    contribution of the j source panel at the i panel
cc
       do 110 j = 1,npanels
        flog = 0.0
        ftan = pi
cc
c       print *,'flog,ftan',flog,ftan
        if(j.eq.i) go to 100
cc
        dxj = xmid(i)-x(j)
        dxjp = xmid(i)-x(j+1)
```

```
        dyj = ymid(i)-y(j)
        dyjp = ymid(i)-y(j+1)
        flog = 0.5*dlog((dxjp*dxjp+dyjp*dyjp)/(dxj*dxj+dyj*dyj))
        ftan = datan2(dyjp*dxj-dxjp*dyj,dxjp*dxj+dyjp*dyj)
cc
c       print *,'flog,ftan',flog,ftan
c       print *,'pi,pinv',pi,pinv
cc
 100    continue
        ctimtj = costhe(i)*costhe(j) + sinthe(i)*sinthe(j)
        stimtj = sinthe(i)*costhe(j) - costhe(i)*sinthe(j)
cc
        a(i,j) = pinv*(ftan*ctimtj + flog*stimtj)
cc
c       print *,'ctimtj,stimtj',ctimtj,stimtj
        b = pinv*(flog*ctimtj - ftan*stimtj)
cc
        a(i,nact) = a(i,nact) + b
cc
c       print *,'a(i,nact)',a(i,nact)
        if((i.gt.1).and.(i.lt.npanels)) go to 110
cc
cc      if i panel touches the trailing edge, add
cc      contributions due to the Kutta Condition
cc
        a(nact,j) = a(nact,j) - b
        a(nact,nact) = a(nact,nact) + a(i,j)
cc
 110    continue
cc
cc      evaluation of the right hand side
cc
        c(i) = sinthe(i)*calf - costhe(i)*salf
 120   continue
       close(unit=12)
cc
       print *,'c(1)',c(1)
       c(nact) = -(costhe(1) + costhe(npanels))*calf
      *          -(sinthe(1) + sinthe(npanels))*salf
cc
c
c         STORE VALUES OF RHS VECTOR BEFORE SENDING [A] TO GAUSS
c
       open(unit=20,file='a.dat',status='unknown')
       open(unit=21,file='c.dat',status='unknown')
c      open(unit=22,file='aa1.dat',status='unknown')
c      open(unit=23,file='tempo.dat',status='unknown')
       do 130 i=1,nact
        write(21,*) i,c(i)
c       write(6,*) 'i,c(i):',i,c(i)
        tempo(i) = c(i)
c       write(23,*) i,tempo(i)
        do 140 j = 1,nact
         write(20,*) i,j,a(i,j)
         a1(i,j) = a(i,j)
c        write(22,*) i,j,a1(i,j)
 140    continue
 130   continue
       close(unit=20)
       close(unit=21)
c      close(unit=22)
c      close(unit=23)

c      print *,'a(nact,nact),c(nact)',a(nact,nact),c(nact)
```

```
cc
      return
      end

c=======================================================================
c     subroutine veldist(x,y,xmid,ymid,costhe,sinthe,a,c,cp,vtan)
c     subroutine veldist(x,y,xmid,ymid,costhe,sinthe,cp,vtan)
      subroutine veldist(x,y,xmid,ymid,costhe,sinthe,loop,cp,vtan)
cc
cc    calculation of pressure distribution
cc
      implicit real*8(a-h,o-z)
cc
      dimension x(1),y(1),xmid(1),ymid(1),costhe(1),sinthe(1)
c     dimension a(1,ntotal),c(1),vtan(1),cp(1)
      dimension vtan(1),cp(1)
c
c     new dimension statements
c
      dimension xt(210),yt(210),xb(210),yb(210)
      dimension prod(210),prodn(210),dstar(210),dstarn(210),
     *          xdstar(210),ydstar(210)
      dimension s(210)
      dimension pgrad(210)
c     dimension ve(210)
c     dimension vtan1(210)
cc
      common/param/ntotal,nseg,nact,nupper,nlower,npanels,
     *                    pi,pinv,nact1
      common/envdata/alpha,vfree,anufree,rhofree,chord,pref,
     *               calf, salf
c
c      new common statements
c
      common/rhs/ tempo(210),a1(210,210)
c     common/skal/ nzero,ymult
      common/cd/ cdf
      common/vel/ ve(210)
      common/ac/ a(210,210),c(210)
      common/trans/ xtrans

      write(6,*) 'in veldis'
c
      loop = 0
c
      ymult = 0.0
c     open(unit=20,file='a1.dat',status='unknown')
      open(unit=21,file='rhs1.dat',status='unknown')
      do 290 i=1,nact
       write(21,*) i,c(i),tempo(i)
       do 280 j = 1,nact
c       write(20,*) i,j,a(i,j)
 280   continue
 290  continue
c     close(unit=20)
      close(unit=21)
 10   continue
cc
cc    vorticity is found in c(nact)
cc
      gamma = c(nact)
cc
      do 130 i = 1,npanels
       vtan(i) = calf*costhe(i) + salf*sinthe(i)
```

```
cc
c
c     add contributions of jth panel
c
       do 120 j = 1,npanels
        flog = 0.0
        ftan = pi
        if(j.eq.i) go to 100
cc
        dxj = xmid(i)-x(j)
        dxjp = xmid(i)-x(j+1)
        dyj = ymid(i)-y(j)
        dyjp = ymid(i)-y(j+1)
        flog = 0.5*dlog((dxjp*dxjp+dyjp*dyjp)/(dxj*dxj+dyj*dyj))
        ftan = datan2(dyjp*dxj-dxjp*dyj,dxjp*dxj+dyjp*dyj)
 100    continue
        ctimtj = costhe(i)*costhe(j) + sinthe(i)*sinthe(j)
        stimtj = sinthe(i)*costhe(j) - sinthe(j)*costhe(i)
cc
        aa = pinv*(ftan*ctimtj + flog*stimtj)
cc
        b = pinv*(flog*ctimtj - ftan*stimtj)
cc
        vtan(i) = vtan(i) - b*c(j) + gamma*aa
cc
 120    continue
        cp(i) = 1.0 - vtan(i)*vtan(i)
 130   continue
cc
cc
c
c        CALL CLCDCM FOR IDEAL SOLUTION
c
      if (loop.eq.0) then
       call clcdcm(xmid,ymid,cp,loop)
      endif
c
c        (invert viscous solution)
c
c     if (loop.eq.1) then
c      do 310 i = 1,nact
c       vtan1(i) = -vtan(npanels+1-i)
c       cp(i) = 1.0 - vtan1(i)**2
c310   continue
c      do 320 i = 1,nact
c       vtan(i) = vtan1(i)
c320   continue
c     endif
c
c        FIND STAGNATION POINT
c
       stag = vtan(1)
       do 150 i = 2,npanels
        if (vtan(i).lt.0) then
         go to 150
        else
         nstagp = i
         nbottom = i
         go to 160
        endif
 150    continue
c
c160    ntop = (nact+1)-nbottom
 160    ntop = npanels-(nbottom-2)
```

```
c       do 170 i = npanels,nstagp,-1
c        vtan(i+1)=vtan(i)
c170    continue
c        vtan(nstagp) = 0.0
        print *, 'nstagp location:',nstagp
c
c       WRITE UPDATED Cp TO DATA FILE AND RETURN TO MAIN
c
       if (loop.eq.1) then
        open(unit=16,file='update.dat',status='unknown')
        open(unit=32,file='rhs3.dat',status='unknown')
        open(unit=33,file='mid.dat',status='unknown')
        do 260 i = 1,npanels
         write(16,*) i,vtan(i),cp(i)
         write(32,*) i,c(i)
         write(33,*) xmid(i),ymid(i)
 260    continue
        close(unit=16)
        close(unit=32)
        close(unit=33)
        go to 20
       endif
c
       open(unit=15,file='velocity.dat',status='unknown')
        do 140 i = 1,npanels
         write(15,*) i,vtan(i),cp(i)
 140    continue
        close(unit=15)
c
c        FIND TRANSITION POINT FOR LOWER SURFACE
c
       do 370 i=1,nbottom
        if(cp(i+1).gt.cp(i))then
         ntrans = i
         xtrans = x(i)
         go to 380
        endif
 370   continue
 380   continue
       if(xtrans.lt.0.1) xtrans = 0.1
       write(6,*) 'xtrans:',xtrans
       write(6,*) 'ntrans:',ntrans
c
       write(6,*) 'setting up ve,x,y for intgrl'
c
c       SET UP Ve, X, AND Y FOR CALLING INTEGRAL
c
c      open(unit=16,file='xbyb.dat',status='unknown')
       sign = -1.0
        do 180 nsurf = 1,2
         if(sign.eq.-1.)then
          do 190 i = 1,nbottom
           ve(i) = sign*vtan(nstagp+1-i)
           xb(i) = xmid(nstagp+1-i)
           yb(i) = ymid(nstagp+1-i)
c          write(6,*) i,xb(i),yb(i),ve(i)
 190      continue
          mode=0
          write(6,*) 'calling intgrl: lower'
c         call intgrl(nbottom,xb,yb,ve,skind,sigdx)
          call intgrl(nbottom,xb,yb,skind,sigdx,prod,dstar,mode)
          write(6,*) 'back from intgrl: lower'
          skindb = skind
          sigdxb = sigdx
```

```
          print *, 'bottom: skindb,sigdxb',skindb,sigdxb
c
c           REARRANGE BOTTOM PRODUCTS AND THICKNESS
c
          do 210 i = 2,nbottom
           prodn(i-1)=prod(nbottom+1-(i-1))
           dstarn(i-1) = dstar(nbottom+1-(i-1))
 210      continue
c
         else
          do 200 i = 1,ntop
           ve(i) = vtan(nstagp-1+(i-1))
           xt(i) = xmid(nstagp-1+(i-1))
           yt(i) = ymid(nstagp-1+(i-1))
 200      continue
          mode=1
          write(6,*) 'calling intgrl: upper'
c         call intgrl(ntop,xt,yt,ve,skind,sigdx)
          call intgrl(ntop,xt,yt,skind,sigdx,prod,dstar,mode)
          write(6,*) 'back from intgrl: upper'
          skindt = skind
          sigdxt = sigdx
          print *, 'top: skindt,sigdxt',skindt,sigdxt
c
c           REARRANGE TOP PRODUCTS AND THICKNESS
c
          do 220 i = 2,ntop
           prodn(nbottom+i-2)=prod(i)
           dstarn(nbottom+i-2)=dstar(i)
 220      continue
         endif
c
         sign = 1.0
 180    continue

        cdf = abs(skindt/sigdxt)+abs(skindb/sigdxb)
c       cdf = (skindt+skindb)/(sigdxt+sigdxb)
        print *, 'sigdxb,sigdxt',sigdxb,sigdxt
        print *, 'cdf,skindb,skindt:',cdf,skindb,skindt
c
c           CALCULATE SURFACE DISTANCE STARTING AT T.E.
c
        s(1) = 0.0
        do 230 i = 2,npanels
         dxmid = xmid(i)-xmid(i-1)
         dymid = ymid(i)-ymid(i-1)
         s(i) = s(i-1)+sqrt(dxmid*dxmid+dymid*dymid)
 230    continue
c
c           CALCULATE PRODUCT GRADIENTS FOR NORMAL VELOCITY (BASED
c           ON 3 POINT QUADRATIC APPROXIMATOR)
c
        open(unit=31,file='pgrad.dat',status='unknown')
        prod1 = prodn(3)
        s1 = s(3)
        prod2 = prodn(1)
        s2 = s(1)
        s(npanels+1) = s(npanels-2)
        do 240 i = 1,npanels
         prod3 = prod1
         s3 = s1
         prod1 = prod2
         s1 = s2
         prod2 = prodn(npanels-2)
```

```
        if (i.lt.npanels)  prod2 = prodn(i+1)
        s2 = s(i+1)
        fact = (s3-s1)/(s2-s1)
        pgrad(i) = ((prod2-prod1)*fact-(prod3-prod1)/fact)/
     *                (s3-s2)
        write(31,*) i,pgrad(i)
 240   continue
       close(unit=31)
c
c         ADD ON NORMAL VELOCITY CORRECTION TO RIGHT HAND SIDE
c         OF COEFFICIENT MATRIX
c
       pgrad(nact) = 0.0
       do 250 i = 1,nact
        c(i) = tempo(i)+pgrad(i)
c       c(i) = tempo(i)
        do 330 j = 1,nact
         a(i,j) = a1(i,j)
 330    continue
 250   continue
c
c     open(unit=22,file='aa2.dat',status='unknown')
      open(unit=23,file='rhs2.dat',status='unknown')
c     open(unit=24,file='a11.dat',status='unknown')
      do 340 i=1,nact
       write(23,*) i,c(i),tempo(i)
       do 350 j = 1,nact
c       write(22,*) i,j,a(i,j)
c       write(24,*) i,j,a1(i,j)
 350   continue
 340  continue
c
c     open(unit=23,file='rhs2.dat',status='unknown')
c     do 360 i=1,nact
c      write(23,*) i,c(i),tempo(i)
c360  continue
c     close(unit=22)
      close(unit=23)
c     close(unit=24)
c
c         CALL GAUSS FOR REEVALUATION
c
c      call solsys(a,c,nact,ntotal,ind)
       call solsys(nact,ntotal,ind)
c
       if (ind.eq.1) write(6,201)
 201   format(//5x,'The system is singular')
c
c         CALCULATE B.L. COORDINATES
c
       open(unit=17,file='blcoord.dat',status='unknown')
       do 270 i = 1,npanels
        xdstar(i) = xmid(i)-dstarn(i)*sinthe(i)
        ydstar(i) = ymid(i)+dstarn(i)*costhe(i)
        write(17,*) i,xdstar(i),1.1*ydstar(i)
 270   continue
       close(unit=17)
c
       loop = 1
       go to 10
c
c      write(6,*) 'nstagp =',nstagp
c      write(6,*) 'nbottom,ntop',nbottom,ntop
cc
```

```
c     print *,'gamma,cp,...',gamma,cp(1),cp(npanels)
cc
 20   return
      end

c=======================================================================
c     subroutine clcdcm(dx,dy,xmid,ymid,cp,cl,cd,cm)
      subroutine clcdcm(xmid,ymid,cp,loop)
cc
cc    calcualtion of aerodynamic coefficients
cc
cc
      implicit real*8(a-h,o-z)
cc
c     dimension dx(1),dy(1),xmid(1),ymid(1),cp(1)
      dimension xmid(1),ymid(1),cp(1)
cc
      common/param/ntotal,nseg,nact,nupper,nlower,npanels,pi,
     *                        pinv,nact1
      common/envdata/alpha,vfree,anufree,rhofree,chord,pref,
     *               calf, salf
      common/cd/ cdf
      common/vc/ vcl,vcd,vcm
      common/xc/ xcl,xcd,xcm
      common/dxdy/ dx(210),dy(210)
c
      write(6,*) 'in clcdcm'
      write(6,*) 'loop:', loop
cc
cc     CLEAR COEFFICIENTS
cc
      cfx = 0.0
      cfy = 0.0
      cl = 0.0
      cd = 0.0
      cm = 0.0
cc
      do 100 i = 1,npanels
       cfx = cfx + cp(i)*dy(i)
       cfy = cfy - cp(i)*dx(i)
       cm = cm + cp(i)*(dx(i)*xmid(i) + dy(i)*ymid(i))
cc
 100  continue
      cd = cfx*calf + cfy*salf
      cl = cfy*calf - cfx*salf
c
c      SAVE IDEAL COEFFICIENTS
c
      if(loop.eq.0)then
       xcd = cd
       xcl = cl
       xcm = cm
       write(6,*) 'ideal coefficients:'
      endif
c
c      ADD SKIN FRICTION DRAG TO FORM DRAG
c
      if(loop.eq.1)then
       write(6,*) 'real coefficients:'
       write(6,*) 'cd,cdf:',cd,cdf
       cd = abs(cd) + cdf
       vcd = cd
       vcl = cl
       vcm = cm
```

```
      endif
cc
      write(6,*) 'alpha (deg):', asin(salf)*180/3.1415926585
      write(6,*) 'cl, cd, cm:', cl,cd,cm
cc
cc    print *,'alpha,cl,cd..',alpha,cl,cd,cm
      return
      end

c=====================================================================
c
c      BOUNDARY LAYER STUFF
c
c=====================================================================
c
c     subroutine intgrl(nx,x,y,ve,skind,sigdx)
      subroutine intgrl(nx,x,y,skind,sigdx,prod,dstar,mode)

c      INTEGRAL METHOD FOR CALCULATION OF BOUNDARY-LAYER
c      GROWTH ON AN AIRFOIL, STARTING AT A STAGNATION POINT
c
c      THWAITES' METHOD USED FOR LAMINAR-REGION
c      MICHEL'S METHOD USED TO FIX TRANSITION
c      HEAD'S METHOD USED FOR TURBULENT-FLOW REGION
c
      implicit real*8(a-h,o-z)
c
      dimension x(210),y(210)
      dimension yy(50)
      dimension dstar(210),prod(210)
c     dimension ve(210)
      common/grad/   xx(200),vgrad(200),theta(200)
      common/rey/ re
      common/par/ naca
      common/par1/ tau,epsmax,ptmax
      common/vel/ ve(210)
      common/hfact/ h
      common/trans/ xtrans

      real lambda,l

      write(6,*) 'in intgrl'

      pi = 3.1415926535
      loop = 0

c     do 500 i = 1,nx
c      write(6,*) i,x(i),y(i),ve(i)
c500  continue
c
c      FIND DISTANCES BETWEEN NODES ALONG SURFACE
c
      xx(1) = 0.0
      do 100 i = 2,nx
      dx = x(i) - x(i-1)
      dy = y(i) - y(i-1)
 100  xx(i) = xx(i-1) + sqrt(dx*dx + dy*dy)

      print *, 'distances found'
c
c      FIND VELOCITY GRADIENTS AT NODES
c
      v1 = ve(3)
      x1 = xx(3)
```

```
      v2 = ve(1)
      x2 = xx(1)
      xx(nx+1) = xx(nx-2)
      do 110  i = 1,nx
       v3 = v1
       x3 = x1
       v1 = v2
       x1 = x2
       v2 = ve(nx-2)
       if (i.lt.nx) v2 = ve(i+1)
       x2 = xx(i+1)
       fact = (x3-x1)/(x2-x1)
       vgrad(i) = ((v2-v1)*fact-(v3-v1)/fact)/(x3-x2)
c      if (i.eq.1) then
c       write(6,*) 'v3',v3
c       write(6,*) 'x3',x3
c       write(6,*) 'v1',v1
c       write(6,*) 'x1',x1
c       write(6,*) 'v2',v2
c       write(6,*) 'x2',x2
c       write(6,*) 'fact',fact
c       write(6,*) 'vgrad(1)',vgrad(1)
c      endif
 110  continue
        write(6,*) 'vgrad(1)',vgrad(1)
c
      print *, 'gradients found'
c     do 530 i = 1,nx
c      write(6,*) vgrad(i)
c530  continue
c
c      SET INTERACTION MODE
c
c     write(6,*) 'INTERACTIVE->1, NON->0'
c     read(5,*) mode
      if (mode.eq.0) then
       ians = 1
c      xtrans = .3
c      re = 6e6
       go to 510
      else
       ians = 2
       continue
      endif
c
c      SELECT TRANSITION CRITERIA
c
c     write(6,*) 'Do you want to fix transition location? (1)'
c     write(6,*) 'Or use Michels Criterion (2)'

c     read(5,*) ians

c     if (ians.eq.2) go to 120
c     write(6,*) 'Input transition location'
c     read(5,*) xtrans
c
      print *, 'criteria selected'
c
c      INPUT REYNOLDS NUMBER BASED ON REFERENCE V AND L
c
c120  write(6,*) 'input Re'
c     read(5,*) re
c
 510  print 1000
```

```
c
      skind = 0.0
      sigdx = 0.0
c
      print *, 'onto laminar calculations'
      open(unit=11,file='prod.dat',status='unknown')
c
c      LAMINAR FLOW REGION
c
      print *, 'vgrad(1)', vgrad(1)
      if (vgrad(1).lt.0.0) vgrad(1) = abs(vgrad(1))
c     if (vgrad(1).lt.0.0) vgrad(1) = vgrad(2)/2.
      theta(1) = sqrt(0.75/re/vgrad(1))
      i = 1
 200  lambda = theta(i)**2*vgrad(i)*re
c     write(6,*) 'lambda:',lambda
      if (lambda.lt.-.0842) go to 400
      call thwats(lambda,h,1)
      if(i.eq.1) h1 = h
      if(i.eq.2.and.h.gt.h1) h = h1
c     write(6,*) 'l,h(2):',l,h
c
c        displacement thickness, 'dstar', and counter, ns
c        (ns to be compared with nx)
c
      dstar(i) = h*theta(i)
      ns = i
      prod(i)=ve(i)*dstar(i)
      write(11,*) i,y(i),prod(i)
c
      cf = 2.*l/re/theta(i)
      if (i.gt.1) cf = cf/ve(i)
      print 1010, x(i),y(i),ve(i),vgrad(i),theta(i),h,cf
      if (i.gt.1)then
        dx = xx(i)-xx(i-1)
      else
        dx = xx(i)
      endif
c
c        skin friction calculator
c
      if(cf.ge.0.0)then
       skind = skind+cf*dx
       sigdx = sigdx+dx
      endif
c      sigdx = sigdx+dx
c
      i = i+1
      if (i.gt.nx) return
      dth2ve6 = .225*(ve(i)**5+ve(i-1)**5)*(xx(i)-xx(i-1))/re
      theta(i) = sqrt(((theta(i-1)**2)*(ve(i-1)**6)+dth2ve6)
     *            /(ve(i)**6))
c     write(6,*) 'dth2ve6,theta:',dth2ve6,theta(i)
      if (i.eq.2) theta(2) = theta(1)
c
c     print *, 'testing for transition...'
c
c       TEST FOR TRANSITION
c
c     if (ians.eq.'m') go to 210
      if (ians.eq.2) go to 210
      if (x(i).lt.xtrans) go to 200
      go to 300
 210  rex = re*xx(i)*ve(i)
```

```
      ret = re*theta(i)*ve(i)
      retmax = 1.174*(1.+22400./rex)*rex**.46
      if (ret.lt.retmax) go to 200
c
c       TURBULENT FLOW REGION
c
 300  itrans = i
      write(6,*) 'itrans:',itrans
      write(6,*) 'x(itrans):',x(itrans)
c310  if (mode.eq.0) then
 310   h = 1.3
c     else
c      write(6,*) 'INPUT H AT TRANSITION (~1.3-1.4)'
c      read(5,*) h
c     endif
c
      if (h.lt.1.0) return
      i = itrans
      yy(2) = h10fh(h)
      yy(1) = theta(i-1)
 320  dx = xx(i)-xx(i-1)
      call runge2(i-1,i,dx,yy,2)
      theta(i) = yy(1)
      h = hofh1(yy(2))
      rtheta = re*ve(i)*theta(i)
      cf = cfturb(rtheta,h)
      print 1020, x(i),y(i),ve(i),vgrad(i),theta(i),h,cf
c
c        skin friction stuff
c
      skind = skind+cf*dx
      sigdx = sigdx+dx
c
c        more displacement thickness and counter stuff
c
      dstar(i) = h*theta(i)
      ns = i
      prod(i)=ve(i)*dstar(i)
      write(11,*) i,y(i),prod(i)
c
      if (h.gt.2.4) go to 410
      i = i+1
      if (i.le.nx) go to 320
      write(6,*) 'no separation, returning'
      close(unit=11)
      return
 400  write(6,*) 'LAMINAR SEPARATION'
      write(6,*) 'REATTACHING A TURBULENT BOUNDARY LAYER'
      go to 300
c     return
 410  write(6,*) 'TURBULENT SEPARATION'
c     if (loop.eq.1) then
       do 520 k = i,nx
        theta(k) = theta(i-1)
        dstar(k) = dstar(i-1)
        prod(k) = prod(i-1)
c       sigdx = sigdx+(xx(i)-xx(i-1))
 520   continue
c     write(6,*) 'STILL TURBULENT SEPARATION... RETURNING'
      close(unit=11)
      return
c     endif
c     loop = 1
c     write(6,*) 'TRYING A DIFFERENT VALUE FOR H'
```

```
c     h=1.2
c     go to 310
 1000 format(///,9x,'x',8x,'y',7x,'ve',6x,'vdot',5x,'theta',8x,'h',
     *           8x,'cf',/)
 1010 format(' 1',f10.5,f9.5,2f9.4,f11.7,f9.4,f10.6)
 1020 format(' t',f10.5,f9.5,2f9.4,f11.7,f9.4,f10.6)
 1030 format(a1)
c
      end
c=======================================================================

      subroutine thwats(lambda,h,l)
c
      implicit real*8(a-h,o-z)
c
c      THWAITE'S CORRELATION FORMULAS
c
      real   l, lambda
      if (lambda.lt.0.0) go to 100
      l = .22+lambda*(1.57-1.8*lambda)
      h = 2.61-lambda*(3.75-5.24*lambda)
c     write(6,*) 'l,h:',l,h
      go to 200
 100  l = .22 + 1.402*lambda + .018*lambda/(.107+lambda)
      h = 2.088 + .0731/(.14+lambda)
c     write(6,*) 'l,h:',l,h
 200  return
      end
c=======================================================================

      function h10fh(h)
c
      implicit real*8(a-h,o-z)
c
c       HEAD'S CORRELATION FORMULA FOR H1(H)
c
      if (h.gt.1.6) go to 100
      h10fh = 3.3 + .8234*(h-1.1)**(-1.287)
      return
 100  h10fh = 3.3+1.5501*(h-.6778)**(-3.064)
      return
      end
c=======================================================================

      function hofh1(h1)
c
      implicit real*8(a-h,o-z)
c
c        INVERSE OF H1(H)
c
c     print *, 'h1:',h1
      if (h1 .lt. 3.3) go to 110
      if (h1 .lt. 5.3) go to 100
      hofh1 = 1.1+.86*(h1-3.3)**(-.777)
c      print *, 'returning from hofh1, 1'
      return
 100  hofh1 = .6778 + 1.1536*(h1-3.3)**(-.326)
c      print *, 'returning from hofh1, 2'
      return
 110  hofh1 = 3.0
c      print *, 'returning from hofh1, 3'
      return
      end
c=======================================================================
```

```
      function cfturb(rtheta,h)
c
      implicit real*8(a-h,o-z)
c
c       LUDWEIG-TILLMAN SKIN FRICTION FORMULA
c
c      print *, 'in fct cfturb'
      cfturb = .246*(10.**(-.678*h))*(rtheta)**(-.268)
c      print *, 'cfturb=',cfturb
      return
      end
c=======================================================================

      subroutine derivs(i)
c
      implicit real*8(a-h,o-z)
c
c      SET DERIVATIVES DY VECTOR Y
c
      common/rnk/ yt(50),yp(50)
      common/grad/ xx(200),vgrad(200),theta(200)
      common/rey/ re
      common/vel/ ve(210)

c     print *, 'in derivs'
c
      h1 = yt(2)
c      print *, '1: h1=',h1
      if (h1 .le. 3.) return
c      print *, '2'
      h = hofh1(h1)
c      print *, '3: h=',h
      rtheta = re*ve(i)*yt(1)
c      print *, '4: i=',i
c      print *, '4: ve(i)=',ve(i)
c      print *, '4: yt(1)=',yt(1)
c      print *, '4: rtheta=',rtheta
      yp(1) = -(h+2.)*yt(1)*vgrad(i)/ve(i)
     *        + .5*cfturb(rtheta,h)
c      print *, '5: yp(1)=',yp(1)
      yp(2) = -h1*(vgrad(i)/ve(i)+yp(1)/yt(1))
     *        +.0306*(h1-3.)**(-.6169)/yt(1)
c      print *, '6: yp(2)=',yp(2)
c
c     print *, 'returning from derivs'

      return
      end
c=======================================================================

      subroutine runge2(io,il,dx,yy,n)
c
      implicit real*8(a-h,o-z)
c
c          2ND-ORDER RUNGE-KUTTA METHOD FOR SYSTEM
c          OF N FIRST-ORDER EQUATIONS
c
      dimension ys(50),yy(50)
      common/rnk/ yt(50),yp(50)
c
c     print *, 'in runge2'
c
      intvls = il-io
      if (intvls.lt.1) go to 200
```

```
      do 130 i = 1,intvls
       do 100 j = 1,n
100    yt(j) = yy(j)
       call derivs(io+i-1)
       do 110 j = 1,n
       yt(j) = yy(j) + dx*yp(j)
110    ys(j) = yy(j) + .5*dx*yp(j)
       call derivs(io+1)
       do 120 j = 1,n
120    yy(j) = ys(j) + .5*dx*yp(j)
130   continue
200   return
      end
```

```
c=====================================================================
c      subroutine solsys(a,c,nn,m,ind)
       subroutine solsys(nn,m,ind)
c
c         Purpose: solution of the linear system of equations
c
c                        AX=C . . . . . (1)
c
c         Technique: gaussian elimination with partial pivoting
c
c         Arguments:
c
c                   a    is an nn x nn matrix
c                   c    is an nn vector
c                   nn   is the dimension of system (1); 1<=nn<=m
c                   m    is the maximum value of nn; 1<=m<=20
c                  ind   is the error indicator
c                        ind=0 . . . no error
c                        ind=1 . . . system (1) is singular
c
c         Remark: The solution of sytem (1) is given back in vector c.
c
c---------------------------------------------------------------------
c
       implicit real*8(a-h,o-z)
c
c
c
c
c      dimension a(m,1),c(1),b(210),iv(210)
       dimension b(210),iv(210)
       common/ac/ a(210,210),c(210)
c
       write(6,*) 'in solsys'
c
c      initialization of variables
c
       ind=0
       if(nn.eq.1) go to 10
       n=nn-1
       do 1 i=1,nn
        iv(i)=i
  1    b(i)=c(i)
c
c      LU decomposition of matrix a
c
       do 6 k=1,n
        kk=k+1
c
c      searching for the pivot in column k
c
        rmax=abs(a(iv(k),k))
        irho=k
        do 2 i=kk,nn
         d=abs(a(iv(i),k))
         if (rmax.ge.d) go to 2
         rmax=d
```

```
          irho=i
 2       continue
c         write(6,*) 'first test' ,k
          if (rmax.eq.0.0d0) go to 9
c         write(6,*) 'got past first test for singularity'
          if (irho.eq.k) go to 3
          imax=iv(k)
          iv(k)=iv(irho)
          iv(irho)=imax
 3        ik=iv(k)
c
c         one step of LU decomposition
c
          do 4 i=kk,nn
          ii=iv(i)
          a(ii,k)=a(ii,k)/a(ik,k)
 4        b(ii)=b(ii)-a(ii,k)*b(ik)
          do 5 j=kk,nn
          do 5 i=kk,nn
          ii=iv(i)
 5        a(ii,j)=a(ii,j)-a(ii,k)*a(ik,j)
 6       continue
         in=iv(nn)
c          write(6,*) 'second test'
         if (a(in,nn).eq.0.0d0) go to 9
c          write(6,*) 'got past second test for singularity'
c
c        Back Substitution
c
         c(nn)=b(in)/a(in,nn)
         do 8 i=1,n
         j=nn-i
         ij=iv(j)
         sum=0.0d0
         jj=j+1
         do 7 k=jj,nn
 7       sum=sum+a(ij,k)*c(k)
 8       c(j)=(b(ij)-sum)/a(ij,j)
         return
 10      continue
c          write(6,*) 'third test'
         if (a(1,1).eq.0.0d0) go to 9
c          write(6,*) 'got past third test for singularity'
         c(1)=c(1)/a(1,1)
         return
 9       ind=1
         return
         end
```

Appendix II: Sample Calculations of c_D and c_L From Fluent Wall Force Data

The wall force data from Fluent is given in the following form:

```
WALL FORCES BY ZONE :
---------------------
UNITS = NEWTONS

WALL       NORMAL FORCES             SHEAR FORCES
ZONE    X-DIR.       Y-DIR.       X-DIR.       Y-DIR.
---- ----------   ----------   ----------   ----------
 W1   0.000E+00    0.000E+00    0.000E+00    0.000E+00
 W3   2.526E+01    1.879E+02    1.308E+00   -1.625E-01
 W4   4.193E+01    4.985E+02    2.316E+00   -1.882E-01
 W5   1.704E+01    8.506E+02    3.321E+00   -6.906E-02
 W6  -3.848E+02    1.145E+03    7.093E+00    1.549E+00
 W7   1.109E+02    1.259E+02    4.306E+00   -4.093E-01
 W8   5.093E+00   -2.227E+02    4.589E+00    1.025E-01
 W9   1.753E+01   -2.335E+02    3.919E+00    3.225E-01
 WA   1.625E+01   -1.668E+02    3.219E+00    4.061E-01
```

Each column can be added to yield:

Normal Forces: $F_{x,n} = -150.797$ N

$F_{y,n} = 2180.45$ N

Shear Forces: $F_{x,s} = 30.071$ N

$F_{y,s} = 1.5295$ N

The respective x and y-component forces may be added together to get the total forces F_x and F_y:

$$F_x = -150.797 \text{ N} + 30.071 \text{ N} = -120.726 \text{ N}$$

$$F_y = 2180.45 \text{ N} + 1.5295 \text{ N} = 2181.9795 \text{ N}$$

Next, these forces can be resolved onto the L and D axes defined by the four degree angle of attack to get F_L and F_D:

$$F_L = F_x \sin 4° + F_y \cos 4°$$

$$= (-120.726\ N)\sin 4° + (2181\ N)\cos 4°$$

$$= 2168.24\ N$$

$$F_D = F_x \cos 4° + F_y \sin 4°$$

$$= (-120.726\ N)\cos 4° + (2181\ N)\sin 4°$$

$$= 31.78\ N$$

Now that the lift and drag forces are known, and knowing $\rho = 1.225\ kg/m^3$, $V_\infty = 87.7$ m/s, and $A = cS = (1m)(1m) = 1\ m^2$ (assuming a unit span), c_L and c_D can be calculated:

$$c_L = \frac{F_L}{\frac{1}{2}\rho V_\infty^2 A}$$

$$= \frac{2168.24N}{\frac{1}{2}(1.225kg/m^2)(87.79m/s)(1m^2)}$$

$$= 0.457$$

$$c_D = \frac{F_D}{\frac{1}{2}\rho V_\infty^2 A}$$

$$= \frac{31.78N}{\frac{1}{2}(1.225kg/m^2)(87.79m/s)(1m^2)}$$

$$= 0.00669$$

Appendix III: Start Point and Variable Limit Data

NACA 0012 Start Point, Scheme 1

```
------ OptdesX v. 2.0  ------ Model: Optimum Airfoil ------

Date: May 18 13:59

============================ Variables ============================
```

Name	Value	T	Map	Min	Max
y1-top 1eft	0.04209039	C	1	0.000000	0.1500000
x2-top 1eft	0.1582809	C	1	0.05000000	0.5000000
y2-top 1eft	0.06001964	C	1	0.000000	0.1500000
x3-top center	0.2855014	C	1	0.1000000	0.9000000
x4-top right	0.4351009	C	1	0.2000000	0.9000000
x5-top right	0.6940482	C	1	0.5000000	0.9800000
y5-top right	0.04554018	C	1	0.000000	0.1500000
x7-bottom right	0.6940482	C	1	0.5000000	0.9500000
y7-bottom right	-0.04554018	C	1	-0.1000000	0.02000000
x8-bottom right	0.4351009	C	1	0.2000000	0.9000000
y8-bottom right	-0.06001964	C	1	-0.1000000	0.000000
x9-bottom cente	0.2855014	C	1	0.1000000	0.9000000
x10-bottom 1eft	0.1582809	C	1	0.05000000	0.5000000
y11-bottom 1eft	-0.04209039	C	1	-0.1000000	0.000000

Name	Value	Name	Row	File
x0-front	0.000000			
y0-front	0.000000			
x1-top 1eft	0.000000			
y3-top center	0.06001964			
y4-top right	0.06001964			
x6-back	1.000000			
y6-back	0.000000			
y9-bottom center	-0.06001964			
y10-bottom 1eft	-0.06001964			
x11-bottom 1eft	0.000000			
alpha (deg.)	4.000000			
vfreen (m/s)	88.00000			
anufree	1.790000e-05			
rhofree (kg/m^3)	1.225000			
chord (m)	1.000000			
pref (Pa)	101325.0			
reynolds #	6000000.			
shape factor, h	1.300000			

```
============================ Functions ============================
```

Name	Value	O	T	Map	Allowable	Indifference
skin friction	0.005202947			1	0.006000000	0.004000000
ideal drag coef	0.0004913578			1	0.001000000	0.000000
real drag coeff	0.006164937	v		1	0.004000000	0.003000000
ideal lift coef	0.4786968			1	2.000000	5.000000
real lift coeff	0.3961095		>	1	0.5000000	0.6000000
ideal mom. coef	-0.1234409			1	-0.1080000	0.000000
real mom. coeff	-0.08983321			1	-0.07150000	0.000000
abscissca 1-2	0.1582809		<	1	0.5000000	0.2000000
abscissca 3-2	0.1272205		<	1	0.5000000	0.2000000
abscissca 4-3	0.1495995		<	1	0.5000000	0.2000000
abscissca 5-4	0.2589473		<	1	0.5000000	0.2000000

abscissca 6-5	0.3059518	<	1	0.5000000	0.2000000
abscissca 6-7	0.3059518	<	1	0.5000000	0.2000000
abscissca 7-8	0.2589473	<	1	0.5000000	0.2000000
abscissca 8-9	0.1495995	<	1	0.5000000	0.2000000
abscissca 9-10	0.1272205	<	1	0.5000000	0.2000000
abscissca 10-11	0.1582809	<	1	0.5000000	0.2000000
ordinate 0-1	0.04209039	<	1	0.07000000	0.02000000
ordinate 2-1	0.01792925	<	1	0.07000000	0.02000000
ordinate 4-5	0.01447946	<	1	0.07000000	0.02000000
ordinate 5-6	0.04554018	<	1	0.07000000	0.02000000
ordinate 0-11	0.04209039	<	1	0.05000000	0.02000000
ordinate 10-11	0.01792925	<	1	0.05000000	0.02000000
ordinate 7-8	0.01447946	<	1	0.05000000	0.02000000
ordinate 6-7	0.04554018	<	1	0.05000000	0.02000000

Name	Value	Name	Weight
yt3=yt2	0.06001964		
yt4=yt2	0.06001964		
yb9=yb8	-0.06001964		
yb10=yb8	-0.06001964		

NACA 0012 Start Point, Scheme 2

------ OptdesX v. 2.0 ------ Model: Optimum Airfoil ------

Date: Jun 9 10:46

========== Variables ==========

Name	Value	T	Map	Min	Max
y1-top left	0.03039909	C	1	0.000000	0.1500000
x2-top left	0.09117680	C	1	0.05000000	0.5000000
y2-top left	0.06011704	C	1	0.000000	0.1500000
x3-top center	0.3052042	C	1	0.1000000	0.9000000
x4-top right	0.5092053	C	1	0.2000000	0.9000000
x5-top right	0.8601437	C	1	0.5000000	0.9800000
y5-top right	0.02366456	C	1	0.000000	0.1500000
x7-bottom right	0.8428542	C	1	0.5000000	0.9500000
y7-bottom right	-0.02635697	C	1	-0.1000000	0.05000000
x8-bottom right	0.5150580	C	1	0.2000000	0.9000000
y8-bottom right	-0.05737329	C	1	-0.1000000	0.02000000
x9-bottom cente	0.3384985	C	1	0.1000000	0.9000000
y9-bottom cente	-0.05990206	C	1	-0.1000000	0.000000
x10-bottom left	0.1038991	C	1	0.05000000	0.5000000
y11-bottom left	-0.03255699	C	1	-0.1000000	0.000000

Name	Value
x0-front	0.000000
y0-front	0.000000
x1-top left	0.000000
y3-top center	0.06011704
y4-top right	0.06011704
x6-back	1.000000
y6-back	0.000000
y10-bottom left	-0.06326210
x11-bottom left	0.000000
alpha (deg.)	4.000000
vfreen (m/s)	88.00000
anufree	1.790000e-05
rhofree (kg/m^3)	1.225000
chord (m)	1.000000
pref (Pa)	101325.0
reynolds #	6000000.
shape factor, h	1.300000

Name	Row	File

========== Functions ==========

Name	Value	O	T	Map	Allowable	Indifference
skin friction	0.005200088			1	0.006000000	0.004000000
ideal drag coef	0.0003167802			1	0.001000000	0.000000
real drag coeff	0.005621189			1	0.004000000	0.003000000
ideal lift coef	0.4843278			1	2.000000	5.000000
real lift coeff	0.3272960	^		1	2.000000	5.000000
ideal mom. coef	-0.1266045			1	-0.1080000	0.000000
real mom. coeff	-0.05720270			1	-0.08000000	-0.1000000
abscissca 1-2	0.09117680		<	1	0.5000000	0.2000000
abscissca 3-2	0.2140274		<	1	0.5000000	0.2000000
abscissca 4-3	0.2040011		<	1	0.5000000	0.2000000

abscissca 5-4	0.3509384	<	1	0.5000000	0.2000000
abscissca 6-5	0.1398563	<	1	0.5000000	0.2000000
abscissca 6-7	0.1571458	<	1	0.5000000	0.2000000
abscissca 7-8	0.3277962	<	1	0.5000000	0.2000000
abscissca 8-9	0.1765595	<	1	0.5000000	0.2000000
abscissca 9-10	0.2345994	<	1	0.5000000	0.2000000
abscissca 10-11	0.1038991	<	1	0.5000000	0.2000000
ordinate 0-1	0.03039909	<	1	0.07000000	0.02000000
ordinate 2-1	0.02971795	<	1	0.07000000	0.02000000
ordinate 4-5	0.03645248	<	1	0.07000000	0.02000000
ordinate 5-6	0.02366456	<	1	0.07000000	0.02000000
ordinate 0-11	0.03255699	<	1	0.05000000	0.02000000
ordinate 10-11	0.03070512	>	1	0.000000	0.02000000
ordinate 7-8	0.03101632	<	1	0.05000000	0.02000000
ordinate 6-7	0.02635697	<	1	0.05000000	0.02000000
ordinate 8-9	0.002528770	<	1	0.05000000	0.02000000
ordinate 9-10	0.003360045	<	1	0.05000000	0.02000000

Name	Value	Name	Weight
yt3=yt2	0.06011704		
yt4=yt2	0.06011704		
yb10 on line	-0.06326211		

NACA 4412 Start Point, Scheme 1

------ OptdesX v. 2.0 ------ Model: Optimum Airfoil ------

Date: Jun 13 11:30

========== Variables ==========

Name	Value	T	Map	Min	Max
y1-top 1eft	0.04987590	C	1	0.000000	0.1500000
x2-top 1eft	0.1687511	C	1	0.05000000	0.5000000
y2-top 1eft	0.09863604	C	1	0.000000	0.1500000
x3-top center	0.3459586	C	1	0.1000000	0.9000000
x4-top right	0.5898137	C	1	0.2000000	0.9000000
x5-top right	0.8737301	C	1	0.5000000	0.9800000
y5-top right	0.03972514	C	1	0.000000	0.1500000
x7-bottom right	0.6171246	C	1	0.5000000	0.9500000
y7-bottom right	-0.005359016	C	1	-0.1000000	0.05000000
x8-bottom right	0.2607737	C	1	0.2000000	0.9000000
y8-bottom right	-0.02477605	C	1	-0.1000000	0.02000000
x9-bottom cente	0.2567351	C	1	0.1000000	0.9000000
x10-bottom left	0.2390857	C	1	0.05000000	0.5000000
y11-bottom left	-0.04435326	C	1	-0.1000000	0.000000

Name	Value
x0-front	0.000000
y0-front	0.000000
x1-top 1eft	0.000000
y3-top center	0.09863604
y4-top right	0.09863604
x6-back	1.000000
y6-back	0.000000
y9-bottom center	-0.02477605
y10-bottom 1eft	-0.02477605
x11-bottom left	0.000000
alpha (deg.)	4.000000
vfreen (m/s)	88.00000
anufree	1.790000e-05
rhofree (kg/m^3)	1.225000
chord (m)	1.000000
pref (Pa)	101325.0
reynolds #	6000000.
shape factor, h	1.300000

Name	Row	File

========== Functions ==========

Name	Value	O	T	Map	Allowable	Indifference
skin friction	0.003916747			1	0.006000000	0.004000000
ideal drag coef	-0.0009351347			1	0.001000000	0.000000
real drag coeff	0.005394147			1	0.003000000	0.002000000
ideal lift coef	0.9967275			1	2.000000	5.000000
real lift coeff	0.9395008	^		1	2.000000	5.000000
ideal mom. coef	-0.3639720			1	-0.1080000	0.000000
real mom. coeff	-0.3411043			1	-0.1500000	-0.1600000
abscissca 1-2	0.1687511		<	1	0.5000000	0.2000000
abscissca 3-2	0.1772075		<	1	0.5000000	0.2000000
abscissca 4-3	0.2438551		<	1	0.5000000	0.2000000

abscissca 5-4	0.2839164	<	1	0.5000000	0.2000000
abscissca 6-5	0.1262699	<	1	0.5000000	0.2000000
abscissca 6-7	0.3828754	<	1	0.5000000	0.2000000
abscissca 7-8	0.3563509	<	1	0.5000000	0.2000000
abscissca 8-9	0.004038600	<	1	0.5000000	0.2000000
abscissca 9-10	0.01764940	<	1	0.5000000	0.2000000
abscissca 10-11	0.2390857	<	1	0.5000000	0.2000000
ordinate 0-1	0.04987590	<	1	0.07000000	0.02000000
ordinate 2-1	0.04876014	<	1	0.07000000	0.02000000
ordinate 4-5	0.05891090	<	1	0.07000000	0.02000000
ordinate 5-6	0.03972514	<	1	0.07000000	0.02000000
ordinate 0-11	0.04435326	<	1	0.05000000	0.02000000
ordinate 10-11	-0.01957721	>	1	0.000000	0.02000000
ordinate 7-8	0.01941703	<	1	0.05000000	0.02000000
ordinate 6-7	0.005359016	<	1	0.05000000	0.02000000

Name	Value		Name	Weight
yt3=yt2	0.09863604			
yt4=yt2	0.09863604			
yb9=yb8	-0.02477605			
yb10=yb8	-0.02477605			

NACA 4412 Start Point, Scheme 2

------ OptdesX v. 2.0 ------ Model: Optimum Airfoil ------

Date: Jun 13 13:16

========== Variables ==========

Name	Value	T	Map	Min	Max
y1-top left	0.04458123	C	1	0.000000	0.1500000
x2-top left	0.1436718	C	1	0.05000000	0.5000000
y2-top left	0.09913947	C	1	0.000000	0.1500000
x3-top center	0.3678981	C	1	0.1000000	0.9000000
x4-top right	0.5460722	C	1	0.2000000	0.9000000
x5-top right	0.8651925	C	1	0.5000000	0.9800000
y5-top right	0.04501393	C	1	0.000000	0.1500000
x7-bottom right	0.7342057	C	1	0.5000000	0.9500000
y7-bottom right	-0.002717187	C	1	-0.1000000	0.05000000
x8-bottom right	0.2959381	C	1	0.2000000	0.9000000
y8-bottom right	-0.02227282	C	1	-0.1000000	0.02000000
x9-bottom cente	0.2712898	C	1	0.1000000	0.9000000
y9-bottom cente	-0.02384712	C	1	-0.1000000	0.000000
x10-bottom left	0.1923185	C	1	0.05000000	0.5000000
y11-bottom left	-0.03956147	C	1	-0.1000000	0.000000

Name	Value
x0-front	0.000000
y0-front	0.000000
x1-top left	0.000000
y3-top center	0.09913947
y4-top right	0.09913947
x6-back	1.000000
y6-back	0.000000
y10-bottom left	-0.02889106
x11-bottom left	0.000000
alpha (deg.)	4.000000
vfreen (m/s)	88.00000
anufree	1.790000e-05
rhofree (kg/m^3)	1.225000
chord (m)	1.000000
pref (Pa)	101325.0
reynolds #	6000000.
shape factor, h	1.300000

Name	Row	File

========== Functions ==========

Name	Value	O	T	Map	Allowable	Indifference
skin friction	0.005340677			1	0.006000000	0.004000000
ideal drag coef	-0.0008125669			1	0.001000000	0.000000
real drag coeff	0.007057980	v		1	0.003000000	0.002000000
ideal lift coef	0.9999108			1	2.000000	5.000000
real lift coeff	0.9600351	>		1	0.7000000	0.8000000
ideal mom. coef	-0.3658146			1	-0.1080000	0.000000
real mom. coeff	-0.3515703	<		1	-0.1500000	-0.1600000
abscissca 1-2	0.1436718	<		1	0.5000000	0.2000000
abscissca 3-2	0.2242263	<		1	0.5000000	0.2000000
abscissca 4-3	0.1781741	<		1	0.5000000	0.2000000

abscissca 5-4	0.3191203	<	1	0.5000000	0.2000000
abscissca 6-5	0.1348075	<	1	0.5000000	0.2000000
abscissca 6-7	0.2657943	<	1	0.5000000	0.2000000
abscissca 7-8	0.4382676	<	1	0.5000000	0.2000000
abscissca 8-9	0.02464830	<	1	0.5000000	0.2000000
abscissca 9-10	0.07897130	<	1	0.5000000	0.2000000
abscissca 10-11	0.1923185	<	1	0.5000000	0.2000000
ordinate 0-1	0.04458123	<	1	0.07000000	0.02000000
ordinate 2-1	0.05455824	<	1	0.07000000	0.02000000
ordinate 4-5	0.05412554	<	1	0.07000000	0.02000000
ordinate 5-6	0.04501393	<	1	0.07000000	0.02000000
ordinate 0-11	0.03956147	<	1	0.05000000	0.02000000
ordinate 10-11	-0.01067041	>	1	0.000000	0.02000000
ordinate 7-8	0.01955563	<	1	0.05000000	0.02000000
ordinate 6-7	0.002717187	<	1	0.05000000	0.02000000
ordinate 8-9	0.001574300	<	1	0.05000000	0.02000000
ordinate 9-10	0.005043939	<	1	0.05000000	0.02000000

Name	Value		Name	Weight
yt3=yt2	0.09913947			
yt4=yt2	0.09913947			
yb10 on line	-0.02889106			