FLUID MECHANICS TUTORIALS IN GKS SUPPORTED FORTRAN

by

Jeffrey W. Santavicca

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTERS OF SCIENCE

in

Mechanical Engineering

APPROVED:

F. J. Pierce

C. L. Dancey

D. J. Nelson

July 1992

Blacksburg, VA

FLUID MECHANICS TUTORIALS IN GKS SUPPORTED FORTRAN

by

Jeffrey W. Santavicca

Committee Chairman: Dr. Felix J. Pierce
Mechanical Engineering

(ABSTRACT)

The purpose of this thesis was to develop a collection of fluid mechanics tutorials written in FORTRAN for the IBM PC. These programs are an improvement on existing software, covering more subjects that are typically covered in introductory and intermediate courses, and including improved graphics capabilities through the implementation of the Graphical Kernel System, or GKS.

Fluid mechanics topics and GKS are brought together to form a package of 30 interactive programs in fluid mechanics. The thesis includes two chapters devoted to the Graphical Kernel System, and a sample program designed to serve as a tutorial to introduce the user to GKS.

Each fluid mechanics program covers a specific topic and is given its own documentation in the thesis. Background information as well as program operations are discussed.

"I Shall Return."

- General Douglas MacArthur
when asked of his desire
to pursue a Ph.D. at the
University of the
Philippines

ACKNOWLEDGEMENTS

I would like to express my thanks to the following people for their assistance, inspiration, and tolerance during the past year.

Dr. F. J. Pierce, who made an eight o' clock undergraduate class interesting enough for me to attend graduate school working with him. Most people choose major professors whose research projects are appealing and funded. Dr. Pierce is the type of man that makes such things irrelevant. Dr. C. Dancey, and Dr. D. Nelson for serving on my advisory and examining committees.

Mr. Kenneth Zachmann for making a technical subject fun and interesting. More instructors should teach subjects like Hotel Atomica. He now has at least one Engineer who owes him his inspiration.

Terry and Patricia Santavicca (aka. Mom and Dad) who quickly assessed the U.S. economy and graciously sent me back to school. They raised a well adjusted young man who has hopefully made them proud. Amy Santavicca, who while living in England was just as much a part of my life as if she were still here at school with me.

Tracy Philcox whose support over the past two years has been invaluable. I never could have completed this thesis if it had not been for her encouragement and understanding. I thank you the most.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Traditionally, fluid mechanics has been taught with textbooks, blackboards, pencil and paper. Now, however, computers have made interactive learning both simple and relatively inexpensive. Thus, it was the objective of this project to construct a package of FORTRAN programs dealing with classic fluid mechanics subjects for interactive use both in the classroom and in private study.

This package of programs was to be an improvement on existing software by fulfilling two objectives. First, a larger library of programs was to be included whose subjects are covered by introductory and intermediate lessons, and second, output displays were to be enhanced with improved computer graphics capabilities.

Inspiration for these programs was obtained from Olfe's Fluid Mechanics Programs For The IBM PC [1], while the subjects explored are contained in textbooks by authors such as Anderson [2], Shames [3], and White [4]. Since these programs are intended to be curriculum oriented, professors may employ them for in-class demonstrations supporting classroom instruction, while students could be expected to augment their understanding and perform homework exercises with the various programs. Perhaps the main scholastic advantage of these programs is that the user is able to quickly determine the effects of changing the parameters in a specific fluid mechanics situation.

Specific areas of fluid mechanics are explored by these programs, including the following:

Hydrostatics
Flow Visualization, Kinematics, Fluxes and Forces
Potential Flow
Viscous Flow
Compressible Flow

These categories are similar to those used by Olfe [1]. In addition, a sample heat transfer conduction and convection program is included, along with a color contour plotter for graphical display of results.

An alternative breakdown of the programs could place them in introductory, intermediate, and advanced (graduate) subject areas. Introductory concepts would certainly include Hydrostatics, and some programs dealing with Flow Visualization, Kinematics, Fluxes and Forces. Other programs could be used as demonstrations to whet a novice student's appetite for continuing study in fluid mechanics. Flow Visualization, Kinematics, Fluxes and Forces along with Potential Flow, Viscous Flow, and Compressible Flow are aimed at intermediate and graduate applications. The single heat transfer program is an introductory sample for an introductory Heat Transfer class.

The programs are written in the WATFOR-77 version of ANSI standard FORTRAN 77. WATFOR-77 and its compliment WATFOR-87 support the international two-dimensional standard for graphics, Graphical Kernel System, or GKS. GKS contains enhanced graphics capabilities that do not exist in such languages as BASIC. An overview of GKS, including its advantages, will be provided later. FORTRAN was chosen (as opposed to PASCAL, for example, which may also implement GKS) due to its prevalence in engineering fields. Its numerical capabilities are still unsurpassed.

CHAPTER 2: THE PROGRAMS

Following is a list of the programs falling under the categories discussed in the Introduction. Here, each program is given a brief description. In the following chapters, each of the programs is discussed in detail.

HYDOSTATICS

ATMOS - Graphical and numerical synopsis of the standard atmosphere

SUBPLATE - Hydrostatic consideration of a submerged flat plate

SUBCURVE - Hydrostatic consideration of a submerged curved surface

FLOW VISUALIZATION, KINEMATICS, FLUXES AND FORCES

LINES - Pathlines and streamlines for unsteady flow past a circular cylinder

TIMELINE - Timelines created by flow in a two-dimensional channel, and a circular pipe

DEFORM - Deformation and rotation of a fluid element

WATERJET - Force caused by deflecting a water jet

POTENTIAL FLOW

DOUBLET - Creation of a doublet by combining a source and a sink

SHEET - Uniform flow produced by an infinite row of vortices

CORNER - Flow into or around a corner

HALFBODY - Uniform flow past a source generating a Rankine half-body

RANKINE - Uniform flow past a source-sink pair generating a Rankine oval

KELVIN - Uniform flow past a pair of vortices generating a Kelvin oval

CYLINDER - Uniform flow past a doublet generating a circular cylinder, including effects of circulation

THINFOIL - Flow past thin airfoils

KTFOIL - Flow past a Karman-Trefftz airfoil with the Joukowski transformation as a subcase

VISCOUS FLOW

BOUNDARY - Laminar and turbulent solutions for boundary layer, momentum, and displacement thicknesses for a flat plate

BLASIUS - Blasius solution to the boundary layer equations for a flat plate

WEDGE - Boundary layer solution for flow past a wedge, Falkner-Skan solution

THWAITES - Boundary layer separation consideration, Thwaites method

ANNULUS - Velocity profiles for axial flow between concentric cylinders

MOODY - Frictional considerations for pipe flow, includes the Moody diagram

COMPRESSIBLE FLOW

COMPRESS - (1) Flow of a one-dimensional calorically perfect, ideal gas that is reversible, adiabatic, with variable area (Isentropic) (2) Flow of a one-dimensional calorically perfect, ideal gas that is adiabatic with constant area and friction (Fanno Flow)

(3) Flow of a one-dimensional calorically perfect, ideal gas that is reversible with constant area, and heat addition or removal (Rayleigh flow)
(4) Plane normal shock for a one-dimensional calorically perfect, ideal gas.

OBLIQUE - Flow across an oblique shock

PRANDTLM - Prandtl-Meyer flow past an expansion corner

NOZZLE - Flow through a converging-diverging (Laval) nozzle

FREEJET - Expansion of supersonic flow exiting an underexpanded nozzle

HEAT TRANSFER

HEAT - Sample conduction and convection finite difference model

CONTOUR - Color contour plotting program for a matrix of output data, for example, from HEAT

As noted earlier, the programs may be considered for use in introductory, intermediate, and advanced academic settings. On the next page, Table 2.1 gives a suggested breakdown.

TABLE 2.1: Recommended Academic Breakdown of Programs

	Introductory	Intermediate	Advanced
HYDROSTATICS			
ATMOS	X		
SUBPLATE	X		
SUBCURVE	X		
FLOW VISUALIZATION, ETC.			
LINES	X	X	
TIMELINE		X	
DEFORM			X
WATERJET	X		
POTENTIAL FLOW			
DOUBLET		X	
SHEET		X	
CORNER	X	X	
HALFBODY		X	X
RANKINE		X	X
KELVIN		X	
CYLINDER		X	X
THINFOIL		X	
KTFOIL		X	X
VISCOUS FLOW			
BOUNDARY	X	X	
BLASIUS		X	X
WEDGE		X	X
THWAITES		X	X
ANNULUS			X
MOODY	X		
COMPRESSIBLE FLOW			
COMPRESS			X
OBLIQUE			X
PRANDTLM			X
NOZZLE	X	X	X
FREEJET			X
HEAT TRANSFER			
HEAT	X		
CONTOUR	X		

2.1 VIRGINIA TECH CURRICULUM

Since these programs were written with the curriculum of Virginia Tech in mind, it is appropriate to include a separate section to discuss how the package specifically relates to VPI.

Fluid mechanics falls under the broader discipline of Mechanical Engineering. Thus, the categories of introductory, intermediate, and advanced correlate to **Fluid Mechanics** (ME 3404), **Fluid Mechanics II** (ME 4404), and **Internal Flow** and **Advanced Thermodynamics Of Fluid Flow** (ME 5404-5414) respectively, while the heat transfer program is an exercise taken directly from **Heat And Mass Transfer** (ME 3304).

WATFOR has been the version of FORTRAN required by the undergraduate Engineering departments since the 1988-89 school year, although instruction in GKS is not currently offered in the undergraduate curriculum. However, **Introduction To Computer-Aided Design And Manufacturing** (ME 4634), does include programming with CADCD, which is similar in many aspects to GKS.

CHAPTER 3: GRAPHICAL KERNEL SYSTEM

Graphical Kernel System, or GKS, is the international two-dimensional standard for graphics. It was accepted in October 1982 by ANSI as the American graphics standard, and in 1983 by ISO as the international standard. The fundamental purpose of GKS is to provide a graphics package which is device independent.

While GKS includes functions for generative computer graphics, and display analysis and programming, only the former was included in these programs. In other words, only the passive creation of images was employed, as opposed to the interactive possibilities of GKS. Interactive uses are reserved for more complicated applications, such as the development of a CAD system. Although GKS can support numerous types of workstations, only a PC with a Rastor display was considered while creating the programs.

GKS constructs images with **primitives**. Primitives are fundamental graphic elements such as points, lines, circles, and text. These elements are qualified by the application of **attributes**. Attributes include color, line type, character height, etc. By combining these primitives and specifying their attributes, complex displays may be created.

FORTRAN supports GKS by including it in a set of external subroutines. Each subroutine is on a one-to-one correspondence with a specific function. Therefore, the key word which implements the GKS utilities is "CALL". A GKS subroutine may be identified by its beginning with the letter "G". The rest of the subroutine name is an abbreviation of the GKS function. For example, the command to define text color is CALL

GSTXCI, which stands for "G" Set Text Color Index. The subroutines specific to the programs are listed below:

GACWK - Activate Workstation
GCLKS - Close Kernel System
GCLWK - Close Workstation
GDAWK - Deactivate Workstation
GFA - Fill Area
GGDP - Generalized Drawing Primitive
GIVGA19 - Install Video Graphics Array Mode
GOPKS - Open Kernel System
GOPWK - Open Workstation
GPL - Polyline
GRINIT - Graphics Initialization
GSCHH - Set Character Height
GSCHUP - Set Character Up Vector
GSCHXP - Set Character Expansion Factor
GSCR - Set Color Resolution
GSELNT - Select Normalization Transformation
GSFACI - Set Fill Area Color Index
GSFAIS - Set Fill Area Index Style
GSLN - Set Line Type
GSPLCI - Set Polyline Color Index
GSTXCI - Set Text Color Index
GSVP - Set Viewport
GSWN - Set Window
GTX - Text

Initialization of WATFOR's graphic capabilities must be performed in DOS before compilation and execution of any program. In addition, one must link with the library containing the GKS functions. To do this, one must enter the following DOS commands:

```
WGKS

SET LIBRARY=WGKS;WATFOR
```

It is simple and convenient to put these commands in a batch file which will also access the WATFOR editor. Experience with the WATFOR editor is assumed when dealing with these programs.

In the next section, the aforementioned subroutines are described in greater detail.

3.1 GRAPHICAL KERNEL SYSTEM - FORTRAN SUBROUTINE COMMANDS

The following synopsis includes only those GKS subroutines employed in the programs. The WATFOR-77 GKS reference manual is recommended for further instruction.

SCREEN INITIALIZATION

The following succession of subroutine calls is necessary to completely initialize GKS for graphics creation.

GOPKS (error file identifier)

Purpose: To open the Graphical Kernel System

error file identifier - INTEGER - determines where errors will be sent by the program. This is always set to 0, which is the default value. In this case, error messages will be written on the terminal

GIVGA19 (workstation type)

Purpose: To prepare GKS to accept 256 colors in medium resolution

workstation type - INTEGER - identifies the type of workstation to be used by the programmer. Its value is always 19 which signifies 256 color, medium resolution graphics.

GOPWK (workstation id, connection identifier, workstation type)

Purpose - To open a workstation so that graphics may be generated

workstation id - INTEGER - always given the value, 1, which corresponds to the programmer's PC.

connection identifier - INTEGER - always given the value, 0, which stores graphics in a file that may be read by any GKS system

workstation type - INTEGER - always given the value, 19, which corresponds to 256 color, medium resolution graphics

GACWK (workstation id)

Purpose: To activate the workstation in which graphics are to be generated

workstation id - INTEGER - always given the value, 1, which corresponds to the programmer's PC.

The four preceding commands are used in succession to activate a graphics resolution which is capable of supporting 256 colors. This is a lower resolution which is only used with VGA monitors when 16 color, high resolution graphics will not support the necessary number of colors. The string of command to activate GKS in this manner would be:

```
CALL GOPKS ( 0 )
CALL GIVGA19 ( 19 )
CALL GOPWK ( 1 , 0 , 19 )
CALL GACWK ( 1 )
```

GDAWK (workstation id)

Purpose: To deactivate a workstation so that no more graphics may be generated there

workstation id - INTEGER - always given the value, 1, which corresponds to the programmer's PC.

GCLWK (workstation id)

Purpose: To close a workstation

workstation id - INTEGER - always given the value, 1, which corresponds to the programmer's PC.

GCLKS

Purpose: To close GKS so that no more graphics may be created

The three preceding commands are used in succession to clear the workstation and prepare the screen for a new graphics resolution. In order to do this, the commands would be:

```
CALL GDAWK ( 1 )
CALL GCLWK ( 1 )
CALL GCLKS
```

These commands, along with the initializing subroutines, may be used in conjunction with the following subroutine, GRINIT.

GRINIT (resolution)

Purpose: To set screen resolution

resolution - INTEGER - corresponds to the degree to which detailed graphics can be constructed

The resolution values used in these programs are 0 and 2.

0 - graphics resolution, used for color graphics displays

2 - text resolution, used for tables and to display program explanations

The subroutine, GRINIT, is a short hand method of initializing graphics. For a MCGA monitor, GRINIT performs the same functions as the formal activation of GKS previously presented. In this case, 256 color, medium resolution will result when using a GRINIT resolution of 0. If a VGA monitor is being used, the GRINIT subroutine will initialize graphics in 16 color, high resolution mode. If 256 colors are desired, the long hand method of initializing graphics must be employed for a VGA monitor.

When using GRINIT, it is not necessary to deactivate and close the workstation. GRINIT automatically performs this function before initiating graphics.

GSVP (transformation number, left value, right value, bottom value, top value)

Purpose: To define a graphics viewport

transformation number - INTEGER - an identification number that is assigned to a particular viewport

left value, right value, bottom value, top value - REAL -values that indicate which portion of the screen is to be identified by the given transformation number

The screen is initially defined with the bottom left and upper right corners having coordinates (0.0,0.0) and (1.0,1.0) respectively. Thus if one desired to identify the bottom right quadrant of the screen as viewport number 2, the command would be:

CALL GSVP (2, 0.5, 1.0., 0.0, 0.5)

When drawing in the active viewport, all graphics extending past the viewport's boundaries will be clipped, although they may still lie within the

bounds of the screen as a whole. However, an error will not result as in BASIC.

GSWN (transformation number, scaled left, scaled right, scaled bottom, scaled top)

Purpose: To scale a viewport

transformation number - INTEGER - identification number assigned to a particular viewport

scaled left, scaled right, scaled bottom, scaled top - REAL values used to specify the dimensions of the viewport

Once a viewport has been defined, it may be scaled to meet the programmers needs. This avoids the necessity of computed scale factors which are usually necessary to plot on a prescaled screen as in BASIC. All of the scaled values may be positive or negative. In addition, the right scaled value and the top scaled value may be respectively less than the left and bottom scaled values. The following example shows how a viewport, 2, may be scaled to match the predefined BASIC screen:

```
CALL GSWN (2, 0.0, 319.0, 199.0, 0.0)
```

GSELNT (transformation number)

Purpose: To identify the active viewport

transformation number - INTEGER - identification number assigned to a particular viewport

Frequently, there are numerous viewports employed in one program. The program is able to construct graphics in only one viewport at a time. Fortunately, once graphics have been drawn, changing active viewports will not erase them. The following example shows how to activate viewport 2.

```
CALL GSELNT ( 2 )
```

PRIMITIVES

Most primitives require two arrays with a specific number of subscripts. The number of subscripts and the significance of these arrays will be addressed for each GKS subroutine individually. These arrays will be referred to as X(n) and Y(n) for convenience.

GPL (number of points, X, Y)

Purpose: To draw a line

number of points - INTEGER - denotes the number of points used to construct a line: Points will be drawn from the first point to the second, from the second to the third, etc.

X - REAL - the X coordinates of the points: X(1) is the X coordinate of the first point, X(2) is the X coordinate of the second point, etc.

Y - REAL - the Y coordinates of the points

For these programs, lines are always drawn with two points. Thus, the endpoints of the lines are always (X(1),Y(1)) and (X(2),Y(2)). It is important **not** to subscript the arrays when calling the GPL subroutine even though it is necessary to do so elsewhere in the program. Please note that it

is not necessary to use X and Y for the arrays. Any array with at least two subscripts may be used.

GGDP (number of points, X, Y, primitive denotation, length of data variable named LDAT, character*80 vector named DAT)

Purpose: To draw circles, arcs, and ellipses

number of points - INTEGER - denotes the number of points used to construct the primitive

X - REAL - the X coordinates of the points: X(1) is the X coordinate of the first point, X(2) is the X coordinate of the second point, etc.

Y - REAL - the Y coordinates of the points

primitive denotation - INTEGER - signifies which primitive is to be drawn:

-1 - circle:
number of points = 2
X(1),Y(1) - coordinates of center
X(2),Y(2) - coordinates of another point on the circle

-2 - arc:
number of points = 3
X(1),Y(1) - coordinates of the center of a circle from which the arc is taken
X(2),Y(2) - coordinates of first point on the arc
X(3),Y(3) - coordinates of last point on the arc

Note: The arc will be drawn counterclockwise from the first point to the last

-3 - ellipse:
number of points = 3
X(1),Y(1) - coordinates of center
X(2),Y(2) - coordinates of a point on the ellipse
X(3),Y(3) - coordinates of a point on the ellipse

Note: The second and third points construct a line which defines one axis of the ellipse

length of data variable named LDAT - INTEGER - dummy variable which should be a declared INTEGER named LDAT set equal to one

character*80 named DAT - INTEGER - dummy variable which should be a declared CHARACTER*80 but does not have to be assigned a string

GFA (number of points, X, Y)

Purpose: To construct a filled area

number of points - INTEGER - denotes the number of points used to construct the filled area

X - REAL - the X coordinates of the points which define the area boundaries

Y - REAL - the Y coordinates of the points which define the area boundaries

The GFA subroutine simplifies the process of changing the color of any area of the screen. The area under consideration may be any polygon, the vertices of which are contained in the coordinate arrays X and Y. These arrays must be dimensioned with at least the number of points which define the polygon. For example, if a square area is to be filled, there are four vertices. Therefore, the coordinate arrays must be dimensioned as X(4) and Y(4). The command to construct the filled square area would be:

CALL GFA (4, X, Y)

It is important to note that the area boundary is defined by a line from the first point contained in the coordinate arrays to the second, from the second to the third, etc., until the last point is connected to the first. The area may be filled in one of four styles specified by the GSFAIS subroutine.

GTX (x start, y start, 'character string')

Purpose: To put text on screen

x start - REAL - X coordinate location from which the character string will be written

y start - REAL - Y coordinate location from which the character string will be written

'character string' - CHARACTER - text to be written to the screen

Text written with the GTX subroutine will be written left to right with the bottom left corner of the text placed on the x start, y start coordinates. Either variables or discreet values may be used for x start and y start, and 'character string' may be a CHARACTER variable, or a string may be placed in the subroutine call explicitly. However, 'character string' may **NOT** be a REAL or INTEGER variable. Using an INTEGER or REAL variable will cause the program to freeze. Finally, text written with this command does not "wrap". Once the text has left the screen, it will not return on the other side. The following example writes the phrase "Gee, writing a thesis is fun!" at the location X=1.5, Y=-0.5:

```
CALL GTX (1.5,-0.5,'Gee, writing a thesis is fun!')
```

ATTRIBUTES

COLORS

GKS has the capabilities for 256 different colors in medium resolution, or 16 colors in high resolution when using a PC workstation and

the graphics initialization subroutines previously discussed. The colors are identified by INTEGER values from 0 to 255, of which, sixteen are predefined. The definition of these colors depend on how graphics were initialized. The preassigned colors are listed below.

GRINIT (0)	GIVGA19 (19)
0 - black	0 - black
1 - red	1 - blue
2 - green	2 - green
3 - brown	3 - cyan
4 - blue	4 - red
5 - purple	5 - purple
6 - cyan	6 - brown
7 - white	7 - white
8 - grey	8 - grey
9 - orange	9 - bright blue
10 - bright green	10 - bright green
11 - bright yellow	11 - bright cyan
12 - bright blue	12 - orange
13 - bright purple	13 - bright purple
14 - bright cyan	14 - bright yellow
15 - bright white	15 - bright white

There are two important aspects of the colors in GKS. First, putting text to the screen with either the WRITE or PRINT statement is always done in color 7. Second, if a primitive is written in one color which is later redefined, the primitive will change color.

GSCR (workstation id, color index, red intensity, green intensity, blue intensity)

Purpose: To define new, and redefine old colors

workstation id - INTEGER - always 1, referring to the programmer's PC

color index - INTEGER - color identifier which may be a value from 0 to 255

red intensity, green intensity, blue intensity - REAL -indicates the "amount" of red, green, and blue to be "mixed" to form a color: The value for each intensity may range from 0.0 to 1.0 with 0.0 corresponding to none of the color and 1.0 corresponding to full intensity.
Examples of red, green, and blue intensities for pure colors are as follows:

1.0, 1.0, 1.0 - white
0.0, 0.0, 0.0 - black
0.0, 1,0, 1.0 - brown
1.0, 0.0, 0.0 - red
1.0, 1.0, 0.0 - yellow
0.0, 0.0, 1.0 - blue
0.0, 1.0, 0.0 - green
1.0, 0.0, 1.0 - purple
1.0, 0.5, 0.0 - orange

The GSCR subroutine creates colors like a projection television by mixing red, green, and blue intensities. This command allows the programmer to redefine colors, create colors, and change the colors of existing primitives. GSCR is especially effective in the designing of color contour programs. The following example changes the color of text displayed using the WRITE statement (color 7) to blue-green:

CALL GSCR (1, 7, 0.0, 0.8, 0.3)

GSPLCI (color index)

Purpose: To define the current drawing color

color index - INTEGER - corresponds to the identifying value of the desired color

Primitives constructed with GPL and GGDP are drawn in colors specified by a preceding GSPLCI subroutine. Once these primitives have been drawn, they may be made to change colors by redefining their original color index using the GSCR subroutine. The color of text written with the GTX subroutine and filled areas are not specified using this subroutine.

GSFACI (color index)

Purpose: To define the current fill area color

color index - INTEGER - corresponds to the identifying value of the desired color

This subroutine performs the same function as GSPLCI but only for the purpose of filling areas. The color of primitives created with the GTX, GGDP, and GPL subroutines are not specified by this subroutine. Again, the color of the area may be changed using the GSCR command.

GSTXCI (color index)

Purpose: To define the current text color

color index - INTEGER - corresponds to the identifying value of the desired color

Only the color of text written with the GTX subroutine may be specified with GSTXCI. The color of text displayed with the WRITE or PRINT statement, and primitives created with GPL, GGDP, and GFA, are not specified with this subroutine.

TEXT

The following subroutines control the attributes of text written with the GTX subroutine. For information on the text writing subroutine GTX, see the discussion of PRIMITIVES in this chapter. Text colors are discussed under the COLORS heading.

GSCHH (character height)

Purpose: To define the height of the characters written with the GTX subroutine

character height - REAL - intended height of the characters expressed in vertical screen units.

Usually, the programmer has redefined the screen scaling. Therefore it is necessary to set the character height to an appropriate value. It should be noted that there is no default value for the character height.

GSCHXP (expansion factor)

Purpose: To expand or contract the text in the written direction

expansion factor - REAL - value aimed at producing properly formed letters

Sometimes when the screen is scaled, one direction may be significantly larger than the other. In this case, the characters of the string produced by GTX may either be spread out, or squashed. This command is used to overcome overexpansion or underexpansion by specifying an expansion factor less than one or greater than one respectively. Negative

values are not allowed. Programming usually involves a trial and error procedure to use this command to its full effectiveness.

GSCHUP (x vector, y vector)

Purpose: To change the text writing direction

x vector, y vector - REAL - direction of a vector which defines the written direction of the text.

One may use this subroutine to reorient text written with the GTX subroutine, but not the WRITE or PRINT statement. The vector may be defined with the knowledge that the positive X direction is considered 1,0 and the positive Y direction is 0,-1. For example, the following statement will create a line of text which would be properly oriented in the landscape format:

CALL GSCHUP (-1.0, 0.0)

MISCELLANEOUS

GSLN (line type)

Purpose: To specify the type of line to be drawn with the GPL subroutine

line type - INTEGER - denotes the type of line to be constructed

1 - solid (default)
2 - dashed
3 - dotted
4 - dash-dotted

Only the four specifications listed above are permitted in the GSLN subroutine. Once the line type has been changed from the default (solid), all primitives constructed with the GPL or GGDP commands will be displayed using the new line type unless the GSLN subroutine is employed again. Redefining the line type will not affect primitives which have already been constructed.

GSFAIS (fill area style)

Purpose: To define the style with which the area is to be filled

fill area style - INTEGER - denotes the style with which the defined area is to be filled

0 - hollow (default)
1 - solid
2 - pattern
3 - hatch

Only the four specifications listed above are permitted in the GSFAIS subroutine. Once the fill area style has been changed from the default (hollow), all areas will be filled using the new style unless the GSFAIS subroutine is employed again. Redefining the fill area style will not affect previously filled areas. A hollow style has the affect of creating a border around the defined area, while the other styles create a "filled" area in a specified color.

On the following pages is a listing of a sample program which demonstrates the use of these GKS subroutines. The program should better

acquaint the reader with the GKS commands employed in these programs. This should facilitate understanding of the codes.

```
*$INCLUDE GRAPH
*$NOCHECK
*$NOEXTENSIONS

      PROGRAM GKSSAMPL

* This program is intended as an overview of the GKS sub-
* routines used in the programs. A more complete compila-
* tion of the commands may be found in the WATFOR-77
* Language Reference Manual by G. Coschi and J. B.
* Schueler.
* In order to explain important concepts, it is often
* necessary in this sample program to use subroutines
* which have yet to be discussed. However, all the GKS
* subroutines used in this sample as well as those
* employed in the programs are demonstrated here.

      IMPLICIT NONE
      REAL X1(2),Y1(2),X2(2),Y2(2),X3(2),Y3(2),X4(2),Y4(2)
      REAL XC,X(3),Y(3),PI,XA(5),YA(5)
      INTEGER I,COL(4),LDAT
      CHARACTER*80 DAT
      CHARACTER*10 STRING

      DATA COL/12,1,11,15/
      PI=3.14159273

************************************************************
*       GRAPHICAL KERNEL SYSTEM EXPLANATION                *
************************************************************

* The programs are written in the GKS supported WATFOR ver-
* sion of FORTRAN. This sample program explores the GKS
* subroutines employed in the programs. The following GKS
* commands are reviewed:

*      1. GOPKS      7. GCLKS     13. GSLN      19. GSPLCI
*      2. GIVGA19    8. GRINIT    14. GGDP      20. GSFAIC
*      3. GOPWK      9. GSVP      15. GFA       18. GSTXCI
*      4. GACWK     19. GSENLT    16. GSFAIS    19. GSCHH
*      5. GDAWK     11. GSWN      17. GTX       20. GSCHXP
*      6. GCLWK     12. GPL       18. GSCR      24. GSCHUP

************************************************************
*                  SCREEN CONTROL SUBROUTINES              *
************************************************************
```

```
*****
* 1 * CALL GOPKS (error file identifier)
*****

* error file identifier - INTEGER - identifies where errors
*         will be sent by the program

* GOPKS opens GKS so that graphics may be generated.  A
* value of 0 is always used in the programs for the error
* file identifier since this will cause the program to
* write error messages to the terminal.

*-------------------------------------------------------*

*****
* 2 * CALL GIVGA19 (workstation type)
*****

* workstation type - INTEGER - identifies the type of
*         workstation used by the programmer.  In the programs,
*         the value is 19 which corresponds to a 256 color,
*         medium resolution graphics monitor.


*-------------------------------------------------------*

*****
* 3 * CALL GOPWK (workstation id, connection identifier,
*****                   workstation type)

* workstation id - INTEGER - always given the value, 1,
*         which corresponds to the programmers PC
* connection identifier - INTEGER - always given the value,
*         0, which causes GKS to store graphics in a file which
*         may be read by any GKS system
* workstation type - INTEGER - this must be the same value
*         as the workstation type used in GIVGA19.

* GOPWK opens a workstation where graphics may be
* generated.  With the workstation type set to 19, the
* generated graphics will be in medium resolution with 256
* color capability.

*-------------------------------------------------------*

*****
* 4 * CALL GACWK (workstation id)
*****
```

```
*  workstation id - INTEGER - always given the value, 1,
*        which corresponds to the programmers PC

*  GACWK activates an open workstation so that graphics
*        creation may begin.

*-------------------------------------------------------*

*  The four preceding subroutines are used in succession to
*  activate a graphics resolution which is capable of
*  supporting 256 colors in medium resolution on a VGA or an
*  MCGA monitor. Later, the GRINIT subroutine call will be
*  discussed. GRINIT has the function of automatically per-
*  forming the aforementioned succession of commands. For
*  an MCGA monitor GRINIT will automatically initialize GKS
*  for 256 colors in medium resolution, but for a VGA
*  monitor, GRINIT will initialize GKS for 16 colors in high
*  resolution. Therefore, the long hand method of using
*  GOPKS, GIVGA19, GOPWK, and GDAWK is necessary for a VGA
*  monitor when more than 16 colors are required.

*        The following commands prepare GKS for 256 color,
*        medium resolution graphics. A blue triangle will
*        then be drawn.

         CALL GOPKS(0)
         CALL GIVGA19(19)
         CALL GOPWK(1,0,19)
         CALL GACWK(1)

*        Set the points corresponding to the triangle vertices

         CALL GSVP(1,0.,1.,0.,1.)
         X(1)=X(2)=0.
         X(3)=1.
         Y(1)=1.
         Y(2)=Y(3)=0.

*        Set the color to blue and draw the triangle

         CALL GSFACI(9)
         CALL GFA(3,X,Y)
         PAUSE

*-------------------------------------------------------*

*****
*  5 *  CALL GDAWK (workstation id)
*****
```

```
* workstation id - INTEGER - always given the value, 1,
*         which corresponds to the programmers PC

* When finished with creating graphics, the GDAWK command
* deactivates the workstation so that no graphics may be
* generated there.

*-------------------------------------------------------*

*****
* 6 * CALL GCLWK (workstation id)
*****

* workstation id - INTEGER - always given the value, 1,
*         which corresponds to the programmers PC

* Once the workstation has been deactivated, the GCLWK
* subroutine will close the workstation to all further
* graphics creation.

*-------------------------------------------------------*

*****
* 7 * CALL GCLKS
*****

* GCLKS closes GKS.  No graphics may be created on any
* workstation unless GKS is reopened with the CALL GOPKS
* command.

*-------------------------------------------------------*

* The three preceding commands are necessary to clear the
* workstation (programmer's PC monitor) and prepare for a
* new resolution.  Oftentimes, programs will switch between
* text and graphics resolutions (explained under subroutine
* GRINIT).  Before a new resolution may be defined, the
* workstation must be deactivated and closed, after which
* GKS must be closed.

*         The following commands will clear the workstation and
*         prepare the program for a new resolution.

          CALL GDAWK(1)
          CALL GCLWK(1)
          CALL GCLKS

*-------------------------------------------------------*
```

```
*****
* 8 * CALL GRINIT (resolution)
*****

* resolution - INTEGER:  0 - graphics resolution used for
*                            graphical display
*                        2 - text resolution used for
*                            documentation

* GRINIT is a short hand method used to initiate GKS
* graphics.  It is equivalent to the SCREEN command in
* BASIC.  Text screens including program explanations,
* user input, and tables are done with the resolution set
* to 2, while graphics are displayed under a resolution of
* 0.  When using a VGA monitor, GRINIT will initialize for
* 16 color, high resolution graphics.  When using an MCGA
* monitor, GRINIT will initialize for 256 color, medium
* resolution graphics.

* When changing resolutions, it is not necessary to close
* GKS before using GRINIT.  GRINIT automatically performs
* this function if it is necessary.

*       Initialize the screen for text.

        CALL GRINIT(2)
        WRITE(*,*)'This is text resolution.'
        WRITE(*,*)'These words will appear the same'
        WRITE(*,*)'on a VGA or an MCGA monitor.'
        PAUSE

*       Initialize the screen for graphics.

        CALL GRINIT(0)
        WRITE(*,*)'This is graphics resolution'
        WRITE(*,*)
        WRITE(*,*)'On a VGA monitor, these words will'
        WRITE(*,*)'be printed in small letters like a'
        WRITE(*,*)'word processor.'
        WRITE(*,*)
        WRITE(*,*)'On an MCGA monitor, these words will'
        WRITE(*,*)'be printed in large block letters.'

*       Draw a blue line from the lower left corner to the
*       upper right.

        X(1)=Y(1)=0.
        X(2)=Y(2)=1.
        CALL GSPLCI(12)
```

```
        CALL GPL(2,X,Y)
        PAUSE

* The subroutine GRINIT will be used in this sample
* program for graphics initialization.

*-------------------------------------------------------*

*****
* 9 * CALL GSVP (transformation number, left value, right
*****                value, bottom value, top value)


* transformation number - INTEGER - identifies the viewport
* values - REAL - indicate the portion of the screen to be
*          identified by the transformation number

* GSVP defines a viewport in which graphics may be drawn in
* any resolution. The screen is considered to be a box of
* length and height one, with the origin at the bottom left
* corner. The programmer may choose any portion of the
* screen in which to draw graphics. For instance, if the
* top left quadrant of the screen is to be defined as view-
* port 3, the command would be:

*        CALL GSVP (3, 0.0, 0.5, 0.5, 1.0)

* where the INTEGER 3 identifies the viewport. The left
* value is the left most position that is the boundary of
* the viewport, and likewise for the right, bottom, and top
* values. It is important to note that graphics will be
* clipped if they cross the boundaries of the currently
* active viewport, even if they would have appeared within
* the total screen area.

* Below, four viewports are defined:
* viewport # 1 - upper right quadrant
*            2 - entire screen
*            3 - box in center
*            4 - random box

        CALL GRINIT(0)
        CALL GSVP(1,0.02,0.5,0.5,0.98)
        CALL GSVP(2,0.0,1.0,0.0,1.0)
        CALL GSVP(3,0.25,0.75,0.25,0.75)
        CALL GSVP(4,0.1,0.3,0.45,0.8)

*-------------------------------------------------------*
```

```
******
* 10 * GSELNT (transformation number)
******

* transformation number - INTEGER - identifies the viewport

* GSELNT specifies the active viewport. Whichever viewport
* is active will be the only one in which graphics may be
* drawn.

* The next lines of code specify the previously defined
* viewports which are then outlined in the following colors

* viewport # 1 - blue
*            2 - red
*            3 - yellow
*            4 - white

*       Initialize points to draw the lines which outline the
*       viewports.

        X1(1)=X1(2)=Y1(1)=X3(1)=Y3(1)=Y3(2)=0.
        Y1(2)=X2(1)=X2(2)=X3(2)=Y4(1)=Y4(2)=1.

*       The loop increments through the viewports which are
*       specified by the GSELNT subroutine, a color is then
*       specified, and the four lines creating the outline
*       are drawn.

        DO I=1,4
         CALL GSELNT(I)
         CALL GSPLCI(COL(I))
         CALL GPL(2,X1,Y1)
         CALL GPL(2,X2,Y1)
         CALL GPL(2,X3,Y3)
         CALL GPL(2,X3,Y4)
        ENDDO

        PAUSE

*-------------------------------------------------------*

******
* 11 * GSWN (transformation number, scaled left, scaled
******       right, scaled bottom, scaled top)


* transformation number - INTEGER - identifies the viewport
* scaled values - REAL - used to dimension the viewport
```

```
* GSWN defines a window in the specified viewport. The
* programmer is able to avoid the use of scale factors by
* dimensioning the screen to his or her liking. The de-
* fault scaling is 0 bottom, 1 top, 0 left, 1 right.

* The following example dimensions the screen from a
* bottom value of -2.0 to a top value of 2.0, from a left
* value of 0.0 to a right value of 2*Pi, and then draws
* the curve Y = 2 SIN (X).

*       Initialize the screen for graphics, define the view-
*       port, scale the viewport, specify the viewport, and
*       define drawing color as purple.

        CALL GRINIT(0)
        CALL GSVP(1,0.0,1.0,0.0,1.0)
        CALL GSWN(1,0.0,2.0*PI,-2.0,2.0)
        CALL GSELNT(1)
        CALL GSPLCI(5)

*       Write the label to the screen.

        WRITE(*,*)'                                    Y = 2 SIN (X)'

*       Creating the axes for the sine curve.

        X1(1)=X2(1)=X2(2)=Y1(1)=Y1(2)=0.
        X1(2)=2.*PI
        Y2(1)=-2.
        Y2(2)=2.
        CALL GPL(2,X1,Y1)
        CALL GPL(2,X2,Y2)

*       Drawing the sine curve in yellow.

        CALL GSPLCI(11)
        X3(1)=Y3(1)=0.
        DO XC=0.,2.*PI,.03
          X3(2)=XC
          Y3(2)=2.*SIN(XC)
          CALL GPL(2,X3,Y3)
          X3(1)=X3(2)
          Y3(1)=Y3(2)
        ENDDO
        PAUSE

*************************************************************
*                          PRIMITIVES                       *
*************************************************************
```

```
******
* 12 * GPL (2, X, Y)
******

* 2 - INTEGER - indicates the number of points used to
*       draw the line
* X,Y - REAL ARRAY DIM(2) - contains coordinates of the
*       points between which the line will be drawn

* GPL is the subroutine call to drawn a line. X(1),Y(1)
* are the coordinates of one endpoint of the line while
* X(2),Y(2) are the coordinates of the other. When the
* GPL subroutine is used, the arrays are not to be sub-
* scripted just as with any other subroutine call.

* Lines between the following points will be drawn:

*       0,0 - 0,3 - blue
*       -5,-3.75 - 5, 3.75 - red
*       -1,3 - 1,3 - yellow

*       Initialize the screen for graphics, define the
*       viewport, dimension the viewport, and specify the
*       viewport.

        CALL GRINIT(0)
        CALL GSVP(1,0.0,1.0,0.0,1.0)
        CALL GSWN(1,-5.0,5.0,-3.75,3.75)
        CALL GSELNT(1)

*       Define the color as blue, set the endpoints of the
*       line, and plot the line.

        CALL GSPLCI(12)
        X(1)=0.0
        Y(1)=0.0
        X(2)=0.0
        Y(2)=3.0
        CALL GPL(2,X,Y)

*       Define the color as red, set the endpoints of the
*       line, and plot the line.

        CALL GSPLCI(1)
        X(1)=-5.0
        Y(1)=-3.75
        X(2)=-X(1)
        Y(2)=-Y(1)
        CALL GPL(2,X,Y)
```

```
*       Define the color as yellow, set the endpoints of the
*       line, and plot the line.

        CALL GSPLCI(11)
        X(1)=-1.0
        Y(1)=3.0
        X(2)=1.0
        Y(2)=3.0
        CALL GPL(2,X,Y)

        PAUSE

*-------------------------------------------------------*

******
* 13 * GSLN (line-type)
******

* line-type - INTEGER - specifies the type of line to be
*         drawn

*   1 - solid (default)
*   2 - dashed
*   3 - dotted
*   4 - dash-dotted

* GSLN allows the programmer to specify the type of line
* to be drawn. The default is a solid line. Dashed,
* dotted, and dash-dotted lines may also be created. Once
* the line type is specified, all primitives created using
* the GPL or GGDP subroutines will be drawn in that mode
* until the line type is redefined.

* The following example constructs four lines using each
* of the line types described above.

*       Initialize the screen for graphics, define a viewport,
*       and specify the viewport.

        CALL GRINIT(0)
        CALL GSVP(1,0.,1.,0.,1.)
        CALL GSELNT(1)

*       Define the endpoints of the four lines.

        X1(1)=X2(1)=X3(1)=X4(1)=0.
        X1(2)=X2(2)=X3(2)=X4(2)=1.
        Y1(1)=Y1(2)=.8
        Y2(1)=Y2(2)=.6
```

```
      Y3(1)=Y3(2)=.4
      Y4(1)=Y4(2)=.2

*     Draw a blue dash-dotted line.

      CALL GSLN(4)
      CALL GSPLCI(12)
      CALL GPL(2,X1,Y1)

*     Draw a red dotted line.

      CALL GSLN(3)
      CALL GSPLCI(1)
      CALL GPL(2,X2,Y2)

*     Draw a yellow dashed line.

      CALL GSLN(2)
      CALL GSPLCI(11)
      CALL GPL(2,X3,Y3)

*     Draw a green solid line.

      CALL GSLN(1)
      CALL GSPLCI(10)
      CALL GPL(2,X4,Y4)

      PAUSE

*-------------------------------------------------------*

******
* 14 * GGDP (number of points, X, Y, primitive denotation,
******       length of data variable named LDAT, char-
*            acter*80 vector named DAT)

* number of points - INTEGER - 2 for a circle, 3 for an arc
* X,Y - REAL ARRAY DIM(2) - coordinates of points used for
*       drawing the primitive:

*           circle:  X(1),Y(1) - coordinates of center
*                    X(2),Y(2) - coordinates of another point
*                                on circle

*           arc:     X(1),Y(1) - coordinates of center of
*                                circle from which the arc is
*                                taken
*                    X(2),Y(2) - point on arc
*                    X(3),Y(3) - point on arc
```

```
*                    NOTE:  arcs are drawn counterclockwise
*                           from X(2),Y(2) to X(3),Y(3)

* primitive denotation - INTEGER - -1 for a circle, -2
*       for an arc
* LDAT - INTEGER - dummy variable set equal to 1
* DAT - CHARACTER*80 - dummy variable - no need for a pre-
*       set value

* GGDP allows the construction of circles, arcs, and
* ellipses. Since only circles and arcs are employed in
* these programs, only they have been discussed. The
* following examples produced are a series of concentric
* circles with two arcs surrounding them.

*       Initialize the screen for graphics, define a viewport,
*       scale the viewport, specify the viewport, and specify
*       the drawing color as green.

        CALL GRINIT(0)
        CALL GSVP(1,0.0,1.0,0.0,1.0)
        CALL GSWN(1,-5.0,5.5,-3.75,3.75)
        CALL GSELNT(1)
        CALL GSPLCI(10)

*       Specify the circle center.

        X(1)=0.0
        Y(1)=0.0

*       Specify other points on the circles by keeping the Y
*       coordinate fixed and incrementing the X coordinate.

        Y(2)=0.0
        LDAT=1
        DO X C=0.1,1.5,0.1
          X(2)=XC
          CALL GGDP(2,X,Y,-1,LDAT,DAT)
        ENDDO

*       Change the drawing color to cyan.

        CALL GSPLCI(14)

*       Specify the arc center.

        X(1)=0.0
        Y(1)=0.0
```

```
*       Specify the starting point of the arc.

        X(2)=2.0
        Y(2)=2.0

*       Specify the ending point of the arc.

        X(3)=-2.0
        Y(3)=2.0

*       Draw the first arc.

        CALL GGDP(3,X,Y,-2,LDAT,DAT)

*       Specify new starting and ending points for the second
*       arc.

        X(2)=-X(2)
        Y(2)=-2.0
        X(3)=-X(3)
        Y(3)=-2.0

*       Draw the second arc.

        CALL GGDP(3,X,Y,-2,LDAT,DAT)

        PAUSE

*-------------------------------------------------------*

******
*  15 *  GFA (number of points, X, Y)
******

*  number of points - INTEGER - the number of points used to
*        specify the area to be filled
*  X,Y - REAL ARRAY DIM(number of points) - coordinates of
*        the points used to specify the area to be filled

*  GFA is used to define and fill an area of the screen.
*  The area may be any polygon.  Any number of points may be
*  used to define the area with the border being constructed
*  with lines connecting the points defined with the X
*  and Y arrays.  The border begins with the first X,Y
*  coordinates and progresses numerically until the last
*  point is connected to the first.

*  Within the area, the programmer may specify the fill
*  style using the GSFAIS subroutine.
```

```
*-----------------------------------------------------*

******
* 16 * GSFAIS (area fill style)
******

* area fill style - INTEGER - specifies the style with
*         which an area is to be filled

*                                   0 - hollow (default)
*                                   1 - solid
*                                   2 - pattern
*                                   3 - hatch

* The subroutine GSFAIS is used in conjunction with GFA to
* specify the fill style for an area.  The following ex-
* ample creates a hollow yellow triangle, a solid red rec-
* tangle, a blue pattern filled right triangle, and green
* hatched random polygon.

*       Initialize the screen for graphics, define a viewport,
*       and specify the viewport.

        CALL GRINIT(0)
        CALL GSVP(1,0.,1.,0.,1.)
        CALL GSELNT(1)

*       Define the vertices of the triangle.

        YA(1)=YA(2)=.8
        YA(3)=1.
        XA(1)=.1
        XA(2)=.4
        XA(3)=.25

*       Specify the area color as yellow and the fill style as
*       hollow.  Construct the area.

        CALL GSFACI(11)
        CALL GSFAIS(0)
        CALL GFA(3,XA,YA)

*       Define the vertices of the rectangle.

        XA(1)=XA(2)=YA(1)=YA(4)=.6
        XA(3)=XA(4)=.9
        YA(2)=YA(3)=.8

*       Specify the area color as red and the fill style as
```

```
*       solid.  Construct the area.

        CALL GSFACI(1)
        CALL GSFAIS(1)
        CALL GFA(4,XA,YA)

*       Define the vertices of the right triangle.

        XA(1)=XA(2)=.1
        XA(3)=.4
        YA(1)=YA(3)=0.
        YA(2)=.5

*       Specify the area color as blue and the fill style as
*       pattern.  Construct the area.

        CALL GSFACI(12)
        CALL GSFAIS(2)
        CALL GFA(3,XA,YA)

*       Define the vertices of the polygon.

        XA(1)=XA(2)=.6
        YA(1)=0.
        YA(2)=.5
        XA(3)=.7
        YA(3)=.3
        XA(4)=1.
        YA(4)=.4
        XA(5)=.9
        YA(5)=.1

*       Specify the area color as green and the fill style as
*       hatched.  Construct the area.

        CALL GSFACI(10)
        CALL GSFAIS(3)
        CALL GFA(5,XA,YA)

        PAUSE

*-------------------------------------------------------*

******
* 17 * GTX (x start, y start, 'character string')
******

* x start, y start - REAL - coordinates of the beginning of
*        the text string
```

```
* 'character string' - CHARACTER - line of text to be put
*         to the screen

* GTX allows the user to put text primitives to the screen
* in different colors. The character string to be put to
* the screen may be either a character variable, or the
* string may be placed in the subroutine call itself. A
* real or integer variable may not be used in the GKS com-
* mand. This will "hang" the program necessitating a
* reboot.

* The following example puts the character variable STRING
* to the screen as well as the line 'I hope you can learn
* from this'. The screen is left in its default scaling of
* 0 on the left, 1 on the right, 0 at the bottom, and 1 at
* the top.

*       Initialize the screen for graphics, define a viewport,
*       and specify the viewport.

        CALL GRINIT(0)
        CALL GSVP(1,0.0,1.0,0.0,1.0)
        CALL GSELNT(1)

*       Define the text color as blue.

        CALL GSTXCI(12)

*       Write text to the screen using a predefined character
*       variable.

        STRING='0123456789'
        CALL GTX(0.0,0.5,STRING)

*       Write text to the screen using a character string
*       within the subroutine call.

        CALL GTX(0.1;0.7,'I hope you can learn from this.')

        PAUSE

************************************************************
*                          COLORS                          *
************************************************************

* GKS has the capabilities for 256 different colors in
* medium resolution, and 16 colors in high resolution.
* Each color is identified by an INTEGER from 0 to 255.
* Text written with the WRITE or PRINT is done in color 7.
```

* If a primitive is drawn in one color which is then
* redefined, the primitive will change color. The first
* sixteen colors are predefined as follows: Note that the
* predefined values are different depending of whether
* GRINIT or the combination of GOPKS, GIVGA19, GOPWK, and
* GACWK is used to initialize graphics.

* GRINIT	GOPKS,GIVGA19,GOPWK,GACWK
* 0 - black	0 - black
* 1 - red	1 - blue
* 2 - green	2 - green
* 3 - brown	3 - cyan
* 4 - blue	4 - red
* 5 - purple	5 - purple
* 6 - cyan	6 - brown
* 7 - white	7 - white
* 8 - grey	8 - grey
* 9 - orange	9 - bright blue
* 10 - bright green	10 - bright green
* 11 - bright yellow	11 - bright cyan
* 12 - bright blue	12 - orange
* 13 - bright purple	13 - bright purple
* 14 - bright cyan	14 - bright yellow
* 15 - bright white	15 - bright white

******
* 18 * GSCR (workstation id, color index, red intensity,
****** green intensity, blue intensity)

* workstation id - INTEGER - always set equal to 1 which
* refers to the programmer's PC
* color index - INTEGER - a number 0 to 255 which identi-
* fies the color
* red, green, blue intensity - REAL - value from 0.0 to
* 1.0 which refers to the amount of red, green, or
* blue to be mixed with the others to form a color

* GSCR creates colors like a projection television by mix-
* ing degrees of red, green, and blue. An intensity of 0.0
* is considered null, while 1.0 is considered full. Some
* intensity combinations for "pure" colors are presented
* here:

* 1.0, 1.0, 1.0 - white
* 0.0, 0.0, 0.0 - black
* 1.0, 0.0, 0.0 - red

```
*                    1.0, 1.0, 0.0 - yellow
*                    0.0, 0.0, 1.0 - blue
*                    0.0, 1.0, 0.0 - green
*                    1.0, 0.0, 1.0 - purple
*                    1.0, 0.5, 0.0 - orange

* The following example shows how a line may have its color
* changed through the above spectrum.  The color is number
* 15.

*       Initialize the screen for graphics, define a viewport
*       and specify the viewport.

        CALL GRINIT(0)
        CALL GSVP(1,0.0,1.0,0.0,1.0)
        CALL GSELNT(1)

*       Define the endpoints of the line

        X(1)=Y(1)=0.  X(2)=Y(2)=1.

*       Define color number 15 as white, define the line color
*       as number 15, and draw the line.

        CALL GSCR(1,15,1.0,1.0,1.0)
        CALL GSPLCI(15)
        CALL GPL(2,X,Y)

        PAUSE

*       Redefine color 15 as black.

        CALL GSCR(1,15,0.0,0.0,0.0)
        PAUSE

*       Redefine color 15 as red.

        CALL GSCR(1,15,1.0,0.0,0.0)
        PAUSE

*       Redefine color 15 as yellow.

        CALL GSCR(1,15,1.0,1.0,0.0)
        PAUSE

*       Redefine color 15 as blue.

        CALL GSCR(1,15,0.0,0.0,1.0)
```

```
      PAUSE

*     Redefine color 15 as green.

      CALL GSCR(1,15,0.0,1.0,0.0)
      PAUSE

*     Redefine color 15 as purple.

      CALL GSCR(1,15,1.0,0.0,1.0)
      PAUSE

*     Redefine color 15 as orange.

      CALL GSCR(1,15,1.0,0.5,0.0)
      PAUSE

*     Redefine color 15 as white.

      CALL GSCR(1,15,1.0,1.0,1.0)

*-------------------------------------------------------*

******
* 19 * GSPLCI (color index)
******

* color index - INTEGER - used to specify the drawing color
*       of primitives constructed with GPL and GGDP but NOT
*       GTX or GFA

* The possible color indices are those mentioned at the
* beginning of the COLOR division of this sample program.
* The following example draws fifteen lines in colors 1
* through 15.

*     Initialize the screen for graphics, define a viewport,
*     dimension the viewport, and specify the viewport.

      CALL GRINIT(0)
      CALL GSVP(1,0.0,1.0,0.0,1.0)
      CALL GSWN(1,0.0,16.0,0.0,1.0)
      CALL GSELNT(1)

*     Specify the Y coordinates of all fifteen lines.

      Y(1)=0.
      Y(2)=1.
```

```
*       Specify the X coordinates of each of the 15 lines in a
*       DO loop.

        DO I=1,15

*         Change the color for each of the lines

          CALL GSPLCI(I)
          X(1)=X(2)=REAL(I)
          CALL GPL(2,X,Y)
        ENDDO

        PAUSE

*-------------------------------------------------------*

******
* 20 * GSFACI (color index)
******

* color index - INTEGER - used to specify the color used to
*         fill an area

* GSFACI allows the programmer to decide which color will
* be used to perform the fill area function. The following
* example constructs three triangles: a yellow right tri-
* angle, a blue isosceles triangle, and a yellow scalene
* triangle.

*       Initialize the screen for graphics, define a viewport
*       and specify the viewport.

        CALL GRINIT(0)
        CALL GSVP(1,0.,1.,0.,1.)
        CALL GSELNT(1)

*       Set the fill area style to solid.

        CALL GSFAIS(1)

*       Define the vertices of the right triangle.

        XA(1)=XA(2)=0.
        XA(3)=.3
        YA(1)=YA(3)=.1
        YA(2)=.7

*       Specify yellow as the fill area color and construct
*       the area.
```

```
      CALL GSFACI(11)
      CALL GFA(3,XA,YA)

*     Define the vertices of the isosceles triangle.

      XA(1)=.4
      XA(2)=.8
      XA(3)=.6
      YA(1)=YA(2)=.5
      YA(3)=1.

*     Specify blue as the fill area color and construct the
*     area.

      CALL GSFACI(12)
      CALL GFA(3,XA,YA)

*     Define the vertices of the scalene triangle.

      YA(1)=YA(2)=.1
      YA(3)=.4
      XA(2)=1.

*     Specify red as the fill area color and construct the
*     area.

      CALL GSFACI(1)
      CALL GFA(3,XA,YA)

      PAUSE

*-------------------------------------------------------*

******
* 21 * GSTXCI (color index)
******

* color index - INTEGER - used to specify the drawing color
*       of text primitives

* Below, the code produces 15 lines of text.

*     Initialize the screen for graphics, define a viewport,
*     and specify the viewport.

      CALL GRINIT(0)
      CALL GSVP(1,0.0,1.0,0.0,1.0)
      CALL GSELNT(1)
```

```
*       Set the X position for text location.

        XC=0.

*       Set the Y position for each of the 15 lines of text
*       using a DO loop.

        DO I=1,15
         XC=XC+1.0/20.0

*        Define the text color and print the word "Hello".

         CALL GSTXCI(I)
         CALL GTX(0.,XC,'Hello')

        ENDDO

        PAUSE

*************************************************************
*                        TEXT ATTRIBUTES                    *
*************************************************************

******
* 22 * GSCHH (character height)
******

* character height - REAL - height of text expressed in
*        scaled vertical screen units

* GSCHH allows the programmer to heighten or shrink the
* characters in a text string which is to be put to the
* screen using the GTX subroutine.  The following example
* demonstrates this.

*       Initialize the screen for graphics, define a viewport,
*       scale the viewport, and specify the viewport.

        CALL GRINIT(0)
        CALL GSVP(1,0.0,1.0,0.0,1.0)
        CALL GSWN(1,0.0,20.0,0.0,30.0)
        CALL GSELNT(1)

*       Define the text color as blue.

        CALL GSTXCI(12)

*       Define the text height as 10 vertical screen units and
*       write the word "BIG".
```

```
      CALL GSCHH(10.0)
      CALL GTX(0.0,10.0,'BIG')

*     Define the text color as red.

      CALL GSTXCI(1)

*     Define the text height as 5 vertical screen units and
*     write the word "MEDIUM".

      CALL GSCHH(5.0)
      CALL GTX(0.0,5.0,'MEDIUM')

*     Define the text color as yellow.

      CALL GSTXCI(11)

*     Define the text height as 1 vertical screen unit and
*     write the word "SMALL".

      CALL GSCHH(1.0)
      CALL GTX(0.0,3.0,'SMALL')

      PAUSE

*-------------------------------------------------------*

******
* 23 * GSCHXP (expansion factor)
******

* expansion factor - REAL - value aimed at producing
*       properly formed characters put to the screen with the
*       GTX subroutine

* Oftentimes, the screen is dimensioned such that text
* written with GTX appears either overexpanded or under-
* expanded. Expansion factors greater than one expand text
* while factors less than one contract text. With GSCHXP
* the programmer may expand or contract the text until it
* is properly formed. This is usually a trial and error
* procedure.

*     Initialize the screen for graphics, define a viewport,
*     dimension the viewport, and specify the viewport.

      CALL GRINIT(0)
      CALL GSVP(1,0.0,1.0,0.0,1.0)
      CALL GSWN(1,0.0,30.0,-10.0,10.0)
```

```
      CALL GSELNT(1)

*     Define the text color as purple and the text height as
*     1.1 screen units.

      CALL GSTXCI(13) CALL GSCHH(1.1)

*     Set the text expansion factor as 1.0 (no expansion).

      CALL GSCHXP(1.0)
      CALL GTX(0.0,0.0,'Factor = 1.O')

*     Set the text expansion factor as 1.2.

      CALL GSCHXP(1.2)
      CALL GTX(0.0,2.0,'Factor = 1.2')

*     Set the text expansion factor as 1.5.

      CALL GSCHXP(1.5)
      CALL GTX(0.0,4.0,'Factor = 1.5')

*     Set the text expansion factor as 2.0.

      CALL GSCHXP(2.0)
      CALL GTX(0.0,6.0,'Factor = 2.O')

*     Set the text expansion factor as 0.8.

      CALL GSCHXP(0.8)
      CALL GTX(0.0,-2.0,'Factor = O.8')

*     Set the text expansion factor as 0.5.

      CALL GSCHXP(0.5)
      CALL GTX(0.0,-4.0,'Factor = O.5')

      PAUSE

*-------------------------------------------------------*

******
* 24 * GSCHUP (x vector, y vector)
******

* x vector, y vector - REAL - these parameters describe a
*       vector which specifies the direction that the text
*       will be written.  The positive X direction is con-
```

```
*       sidered 0.0,1.0 while the positive Y direction is
*       0.0,-1.0.

* GSCHUP allows the programmer to reorient text written
* with the GTX subroutine by specifying a vector described
* above.

*       Initialize the screen for graphics, define a viewport,
*       specify the viewport, and define the text color as
*       cyan.

        CALL GRINIT(0)
        CALL GSVP(1,0.0,1.0,0.0,1.0)
        CALL GSELNT(1)
        CALL GSTXCI(14)

*       Write text upside down and backwards to the right.

        CALL GSCHUP(0.0,-1.0)
        CALL GTX(0.5,0.5,' VECTOR = 0.0,-1.0')

*       Write text in the normal orientation.

        CALL GSCHUP(0.0,1.0)
        CALL GTX(0.5,0.5,' VECTOR = 0.0,1.0')

*       Write text downward.

        CALL GSCHUP(1.0,0.0)
        CALL GTX(0.5,0.5,' VECTOR = 1.0,0.0')

*       Write text upward.

        CALL GSCHUP(-1.0,0.0)
        CALL GTX(0.5,0.5,' VECTOR = -1.0,0.0')

        PAUSE

        END
```

CHAPTER 4: HYDROSTATICS

Traditionally, hydrostatics has been an introductory subject due to its usually being a relatively simple kind of fluid mechanics problem. Following is the equation describing equilibrium of a fluid element

$$\nabla p = \rho(\vec{g} - \vec{a}) + \mu \nabla \vec{V} \tag{4.1}$$

where $\vec{g}$ is the acceleration due to gravity which acts in the negative Z direction, $\vec{a}$ is the particle acceleration vector, and $\vec{V}$ is the velocity vector. For the hydrostatic case, the viscous and acceleration terms vanish. With only the Z direction (directed upward) considered, Equation (4.1) reduces to

$$\frac{dp}{dz} = -\rho g \tag{4.2}$$

This is the hydrostatic equation.

For liquids, a free surface is usually considered at which the pressure is atmospheric. In addition, the acceleration due to gravity is taken to be a constant. If the origin of the Z axis is considered zero at the free surface and the positive Z direction directed upward from the free surface, the integrated form of Equation (4.2) is

$$P = P_{atm} - \rho \, g \, z \tag{4.3}$$

For most problems involving liquids, the density may be considered constant. However, this is not typically true for gases. An example is the pressure distribution of the atmosphere in which the density varies with

altitude above the earth. In this case, the density is pressure and temperature dependant.

The three programs in this chapter solve Equation (4.2) for three specific cases. ATMOS concerns the temperature and pressure distributions for the United States Standard Atmosphere. SUBPLATE deals with pressure forces on one side of a flat plate submerged in water, and SUBCURVE considers pressure forces on one side of a curved surface submerged in water.

4.1 ATMOS

Program ATMOS concerns the pressure and temperature distributions of the United States Standard Atmosphere up to an altitude of 79 km. The temperature and pressure profiles are approximated by the following models.

A model comprised of seven linear profiles is employed for the temperature distributions shown in Figure 4.1. Temperature values and altitudes at the endpoints of each linear region are specified in the program in arrays T and Z respectively. Lapse rates for each layer are computed according to the following equation.

$$B = -\left(\frac{T_{i+1} - T_i}{z_{i+1} - z_i}\right) \tag{4.4}$$

and are stored in the array B. At a given altitude, the temperature is found by determining the layer in which the altitude lies, and employing the relationship

$$T = T_i - B_i \ (z - z_i) \tag{4.5}$$

which is Equation (4.4) rearranged for the temperature.

The atmospheric pressure profile is comprised of three regions. The first extends from sea level to the maximum altitude of the first temperature layer. For this region the lapse rate equation of the first temperature layer, together with an ideal gas model, are combined with the hydrostatic relation, which is then integrated, producing the following result

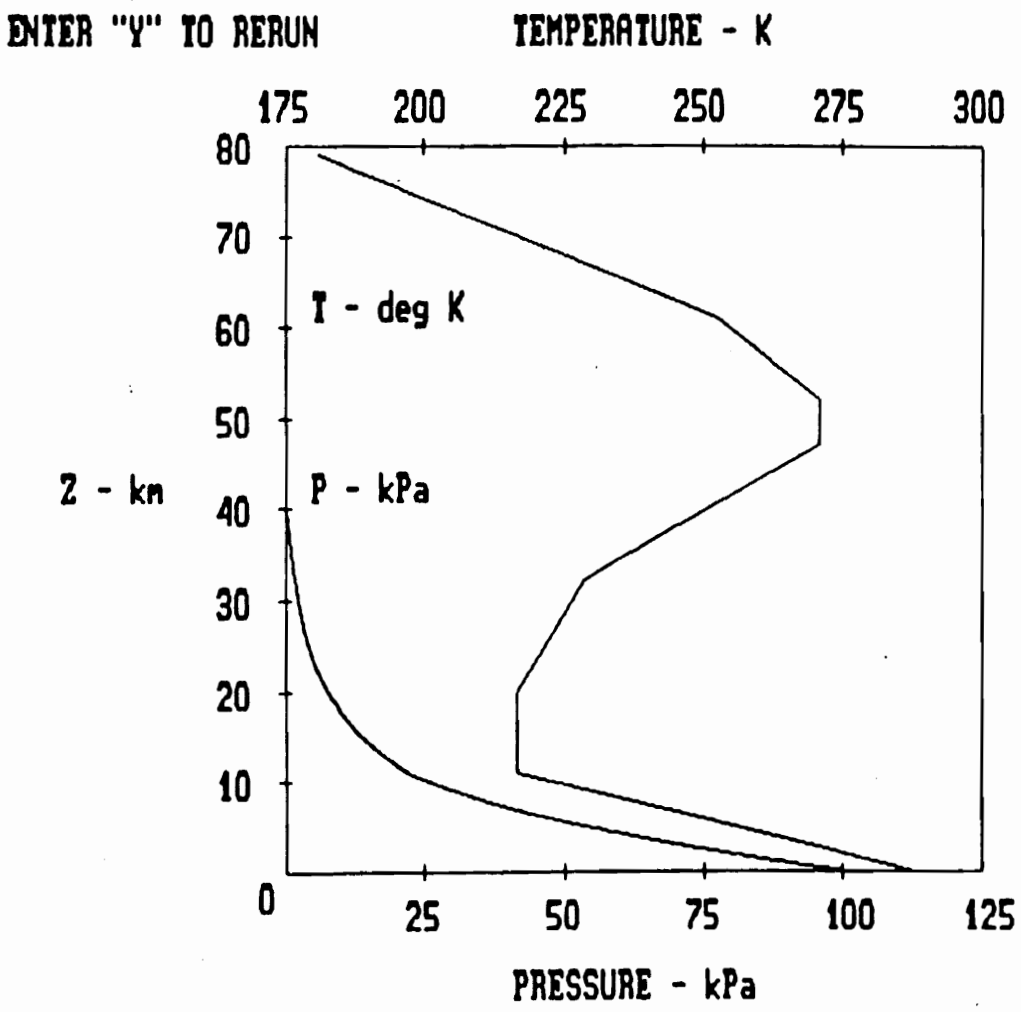


FIGURE 4.1: Temperature and Pressure Profiles For The Standard Atmosphere

$$P = P_o \left(1 - \frac{Bz}{T_o}\right)^{g/RB} \tag{4.6}$$

T_o and P_o are the sea level conditions with values 101.33 kPa and 288.15 K, while the exponent equals 5.26 for air. The second pressure region corresponds to the second temperature layer. Here the temperature profile is vertical (lapse rate equal to zero), therefore, a new approximation must be used. Density in the hydrostatic equation is replaced according to the ideal gas equation and integrated.

$$P = P_i \exp\left(\frac{-g(z - z_i)}{RT_o}\right) \tag{4.7}$$

Equation (4.7) is valid from 11 km to 20 km. Above 20 km the pressure profile is assumed to be linear with the pressure being effectively zero at the end of the last temperature region at an altitude of 79 km. This model is for the calculation of specific values. When plotting the temperature and pressure profiles as a function of altitude, the pressure is considered zero at 40 km. Above this altitude, the pressure is not plotted, although the temperature continues to be modeled as greater than zero.

The user may choose to have both the temperature and pressure profiles plotted, as a function of altitude, from sea level to the mesopause (79 km), or to calculate specific values. If a plot is desired, there is no user interaction as the program creates the display shown in Figure 4.1.

If specific values are to be calculated, the user first chooses either English or SI units. Then the altitude must be supplied. The program will

then determine the temperature and pressure region corresponding to the entered altitude. The appropriate equations are then employed to calculate the pressure and temperature. Program output also includes ratios of the temperature and pressure at the specified altitude to the respective values at sea level. The density is determined using the ideal gas equation. All of the calculations are performed in SI units and are converted to English units with the use of a conversion factor called CFZ in the program.

A more complete analysis of the standard atmosphere model may be found in almost any undergraduate textbook, for example, White [4].

4.2 SUBPLATE

SUBPLATE calculates the hydrostatic force and the center of pressure on an inclined flat plane surface. The user may choose among three area shapes, a rectangle, circle, or triangle. Figure 4.2 depicts some hydrostatic considerations of a submerged surface analysis. The resultant force acts normal to the surface and is located at the center of pressure. Locations of the center of pressure and center of gravity of the area are measured relative to the coordinate systems shown in Figure 4.2.

Table 4.1 lists the formulas used to calculate the important parameters of the problem. The equations are grouped by surface shape, i.e. rectangle, circle, triangle.

The first choice proposed to the user is whether English or SI units are to be used by the program. Subsequently, there are several numerical inputs which are independent of the surface shape. These include the inclination angle, which must be a value from 0 to 90 degrees, and the depth of the surface measured from the free surface to the most elevated tip of the flat plane surface. The user then selects either a rectangular, circular, or triangular flat plane surface.

Inputs for a rectangle include the length and breadth of the surface. The breadth is the dimension parallel to the free surface. Only the radius is necessary for a circular surface. When considering a triangle, the user must specify values for the base, height, and vertex displacement. The base is the dimension parallel to the free surface and is always considered to be the deepest line of the triangle. The vertex displacement, s, is defined as the

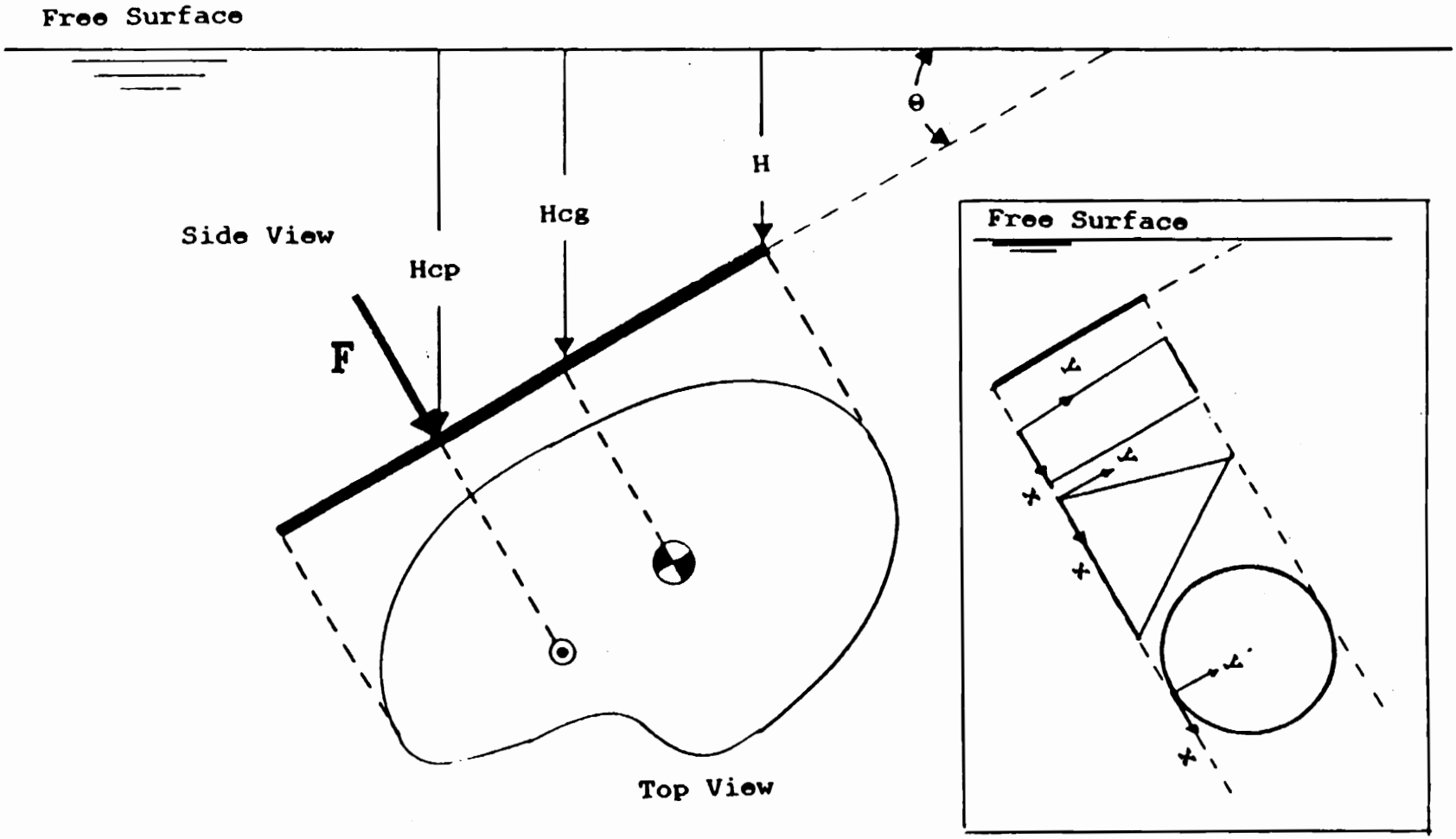


FIGURE 4.2: Hydrostatic Consideration of a Submerged Flat Plane Surface

TABLE 4.1: Calculation of Parameters for a Submerged Flat Plane Surface

D - Distance from free surface to top edge of plane surface
θ - Inclination angle
Xcg - X coordinate of center of gravity
Ycg - Y coordinate of center of gravity
Hcg - Distance from free surface to center of gravity
Xcp - X coordinate of center of pressure = $-\rho$ g Ixy SIN θ /F
Ycp - Y coordinate of center of pressure = $-\rho$ g Ixx SIN θ /F
Hcp - Distance from free surface to center of pressure

Rectangle:

B - Breadth
L - Length

$Ixx = L B^3 / 12$
$Ixy = 0$

$F = \rho$ g Hcg B L

$Xcg = B/2$

$Xcp = 0$

$Ycg = L/2$

$$Ycp = \frac{-L^2 \sin\theta}{12 Hcg}$$

$Hcg = D + L \text{ SIN } \theta / 2$
$Hcp = Hcg - Ycp \text{ SIN } \theta$

Circle:

R - Radius

$Ixx = R / 4$
$Ixy = 0$

$F = \rho$ g Hcg πR^2

$Xcg = 0$

$$Xcp = \frac{-R^2 \sin\theta}{4 Hcg}$$

$Ycg = R$
$Hcg = D + R \text{ SIN } \theta$
$Hcp = Hcg - Ycp \text{ SIN } \theta$

$Ycp = 0$

Triangle:

b - base
h - Height
s - Vertex Displacement

$Ixx = b h^3 / 36$
$Ixy = \frac{b (b-2s) h}{72}$

$F = \frac{\rho g \, Hcg \, b \, h}{2}$

$Xcg = (2b - s) / 3$
$Ycg = h / 3$

$Xcp = \frac{-(b-2s) h \text{ SIN } \theta}{36 Hcg}$

$Hcg = D + 2h \text{ SIN } \theta / 3$

$$Ycp = \frac{-h^2 \sin\theta}{18 Hcg}$$

$Hcp = Hcg - Ycp \text{ SIN } \theta$

distance, measured parallel to the triangle base, between the triangle vertex and a line normal to and bisecting the base.

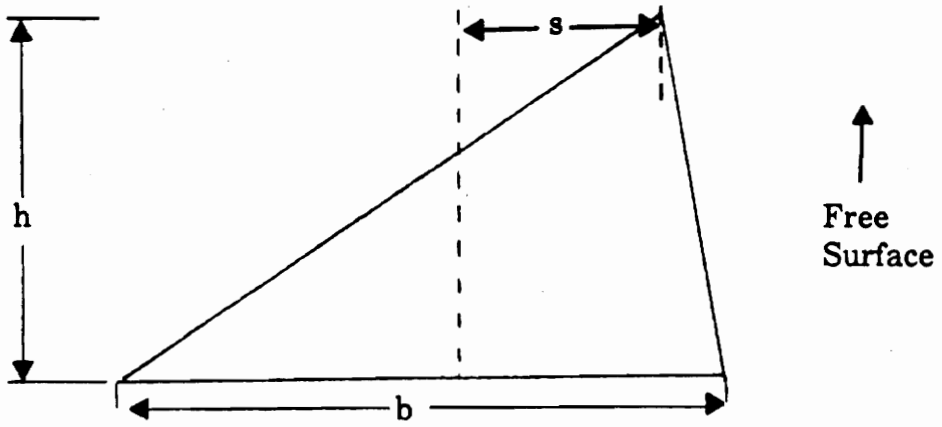


Figure 4.3: Dimensions of a Triangle

The program output is a table of the user inputs and the parameters listed in Table 4.1. Units are chosen by the user. The program will convert forces to kN or kips as necessary.

4.3 SUBCURVE

The hydrostatic force acting on a submerged curved surface may be determined by analyzing the horizontal and vertical force components. For a fluid with a free surface, the vertical force is simply the weight of the column of water above the curved surface, while the horizontal force is the hydrostatic force on a projected area perpendicular to the free surface. Since the curved surface is taken to be of uniform breadth (direction parallel to the free surface), the projected area is treated as a rectangle at an inclination angle of 90 degrees.

The equation describing the curved surface is

$$Y = Y_o \left(\frac{X}{X_o} \right)^n \tag{4.8}$$

where the user selects the horizontal and vertical lengths, X_o and Y_o respectively, and the value of the exponent, n, all of which define the shape of the curved surface. Other user inputs include the breadth of the curved surface and the height of the free surface above the uppermost edge of the curved surface.

The program calculates the horizontal and vertical components of the resultant force. Either English or SI units may be chosen by the user. The equations for the two force components become,

$$Fh = \rho \, g \, (Y_o/2 + H) \, Y_o \, B \tag{4.9}$$

$$Fv = \rho \, g \left(H + Y_o - \frac{Y_o}{n+1} \right) X_o B \tag{4.10}$$

where H is the height of the free surface above the curved surface, and B is the breadth. The resultant force does not act through the centroid of the surface. Rather, it acts through the center of pressure, CP, which is determined by assuming that no unbalanced moments will act on the plate. Also calculated are the force moment arms in the X and Y directions, which are measured relative to the lowest point of the curved surface. The equations for calculating the moment arms can be found within the program itself. The values of X_{ma} and Y_{ma} are found by first determining the coordinates of the center of pressure and then calculating the distance between CP and the lowest edge of the curved surface.

Output includes a scaled drawing of the curved surface, the free surface, and the vectors representing the horizontal and vertical components of the resultant force. Values for the force components are listed along with those for the moment arms. Figure 4.4 shows a sample program output including the calculated quantities.

Horizontal Force: 294.87 kN
Vertical Force: 490.11 kN
X Moment Arm: 1.35 m
Y Moment Arm: 0.89 m

Enter "Y" to Rerun

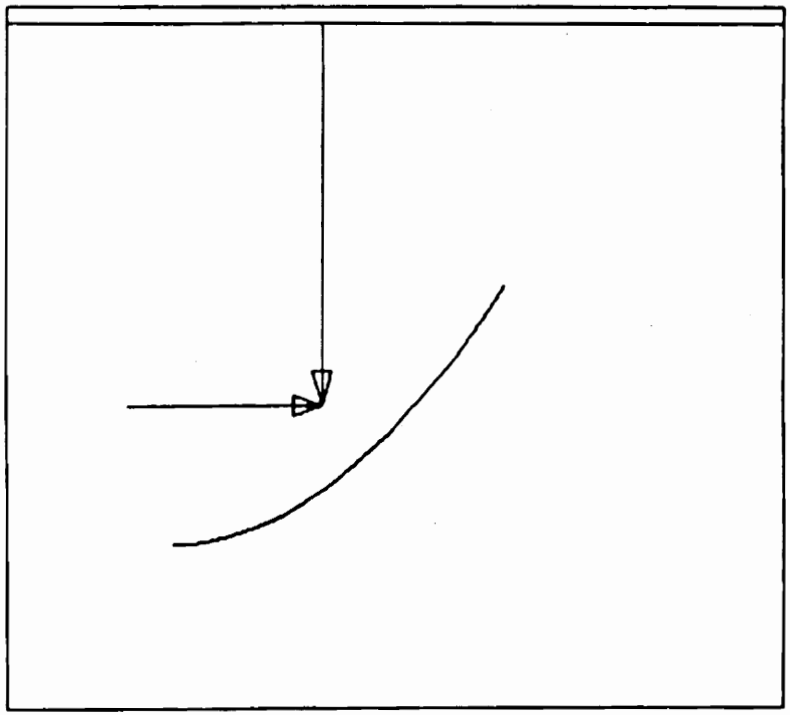


FIGURE 4.4: Hydrostatic Consideration of a Submerged Curved Surface (Xo = 3m, Yo = 2m, n = 2, Depth = 2m, Breadth = 5m)

CHAPTER 5: FLOW VISUALIZATION, KINEMATICS, FLUXES AND FORCES

This chapter deals with a wide range of fluids fundamentals. These programs are intended to support topics and concepts introduced in a first year fluids mechanics course.

Basic to the understanding of fluid mechanics, and especially useful in potential flow, are the concepts of streamlines, pathlines, and timelines. Since most introductory lessons concern steady flows, students often miss the distinction between pathlines and streamlines. The program LINES involves a cylinder moving from right to left across the screen. Because the program considers an unsteady flow, the difference between pathlines and streamlines is illustrated. Timelines are shown being advected downstream in a channel or pipe in the program TIMELINE.

The deformation of a square fluid particle is visualized in program DEFORM. The user inputs values for the rotation, shear strain, and extensional strain rates, and the program then animates the deformation of the fluid particle.

WATERJET calculates the force produced by a fluid stream impinging on a deflection plate.

5.1 LINES

A streamline is a line everywhere tangent to the velocity vectors at a given instant, and a pathline is the actual path traversed by a given fluid particle. For steady flows, both lines are the same and are collectively referred to as streamlines. Since most introductory fluid mechanics courses focus on steady flows, students often loose the distinction between the two types of lines.

Pathlines may be physically constructed by highlighting a particular fluid particle (hydrogen bubble or dye injection) and following that particle's motion, for example, with time exposure photography. Streamlines may be found in a similar manner by photographing highlighted particles with a series of relatively short time exposure photographs which provide approximate instantaneous velocity vectors.

Since pathlines and streamlines are usually different for unsteady flows, the case considered here is that of flow past a moving cylinder. To create an unsteady flow, the fluid is taken to be at rest while the cylinder moves from right to left across the screen. A potential flow solution is assumed. (See program CYLINDER.)

Two options are initially proposed to the user; streamlines may be plotted, or a single pathline can be constructed. Streamlines are drawn according to the following nondimensional stream function,

$$S = \frac{Y}{(X+T)^2+Y^2} \tag{5.1}$$

where $S = -\psi/Ua$, $X = x/a$, $Y = y/a$, and $T = Ut/a$. U is the cylinder speed and **a** is the cylinder radius. T, the nondimensional time, may take values from -7 to +7. At T = -2, the cylinder is at its right most position on the screen, T = 0 corresponds to the cylinder at the center of the screen, and T = +2 corresponds to the cylinder at the left most screen position. Using the definition of the stream function to obtain the velocities in the X and Y directions, the following results are obtained.

$$u = \frac{dX}{dT} = \frac{Y^2 - (X+T)^2}{Y^2 + (X+T)^2}$$

$$v = \frac{dY}{dT} = \frac{-2(X+T)Y}{Y^2 + (X+T)^2} \tag{5.2}$$

If these equations are integrated, the result provides a pathline. A numerical integration of the equations is performed in the program.

If the user elects to plot streamlines, a value for the nondimensional time, T, must be entered. In this unsteady case, the observer is fixed while the cylinder moves across the reference frame. Therefore, the streamlines are plotted relative to the observer (in an inertial reference frame) for the specified instant in time. Because the flow field is symmetric about the X axis, only the upper half of the cylinder and the streamlines in the positive Y domain are drawn.

In a subsequent program, CYLINDER, a steady flow past a circular cylinder is considered, where streamlines are plotted relative to the

cylinder, which is fixed relative to the reference frame.

If pathlines are to be constructed, the user must select the initial vertical displacement of the fluid particle from the X axis. The cylinder is then caused to move across the screen. The first time step considers the jump from T = - ∞ to T = -7, then a smaller, uniform step size is employed between T = -7 and T = +7. Finally one more integration is carried out from T = +7 to T = ∞. A predictor-corrector scheme is employed for these integrations. Appendix A, Numerical Methods, contains further details. Figures 5.1 and 5.2 show examples of plotted streamlines and pathlines.

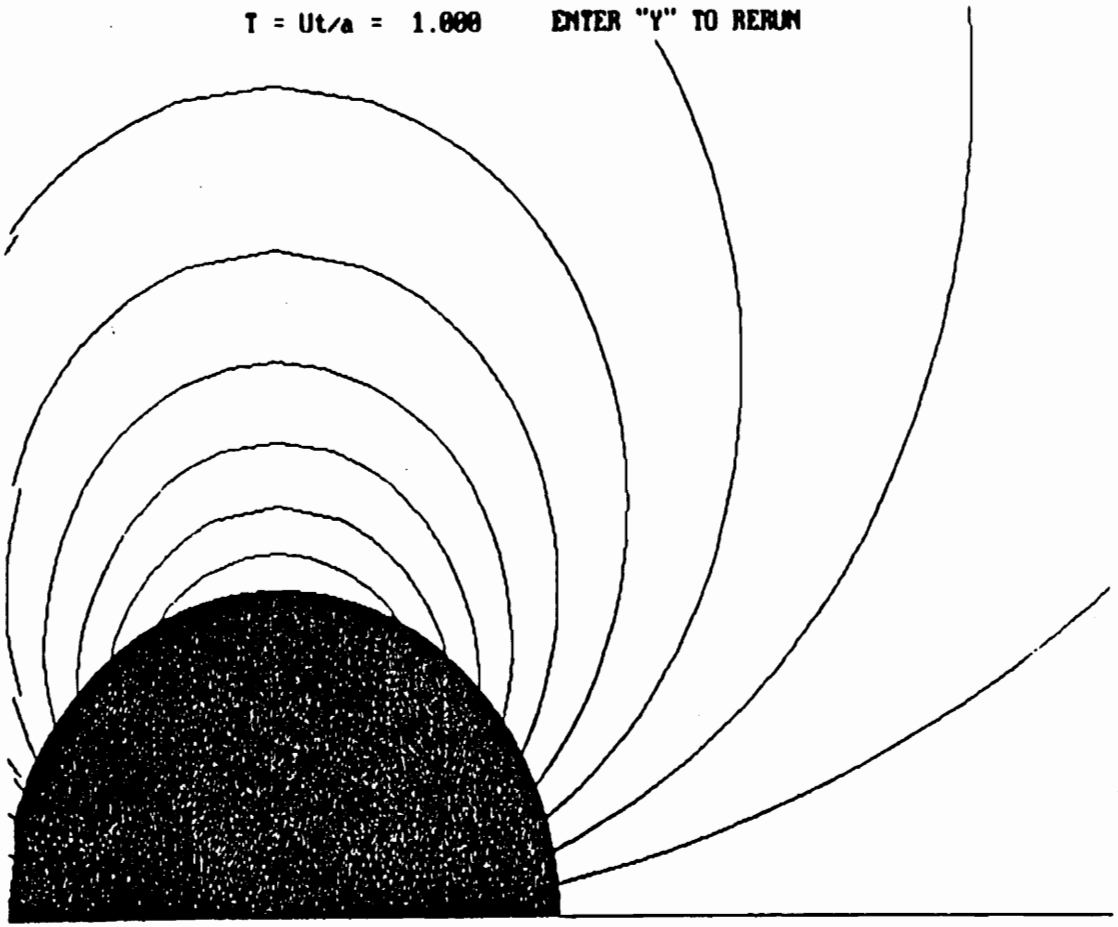


FIGURE 5.1: Streamlines Generated by Program LINES

CYLINDER IS APPROACHING FROM THE RIGHT

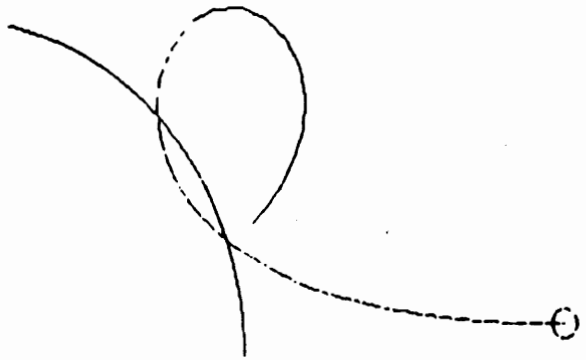


FIGURE 5.2: Pathline Generated by Program LINES
(Initial Y Position = 0.1)

5.2 TIMELINE

A timeline is defined in White [4] as a set of fluid particles that form a line at a given instant in time. For instance, a line of hydrogen bubbles perpendicular to the flow direction may be produced by electrolysis across the entrance to a channel. The bubbles will be carried downstream by the flow with the displacement of individual bubbles being proportional to the velocity of fluid particles surrounding each bubble. At any instant, the bubbles form a timeline. Effectively, this is what is being considered in this program. The program is capable of considering flow in both a two-dimensional channel or three-dimensional circular pipe.

A power law fluid is considered where the shear stress, τ , is proportional to the strain rate, ε , to some power, **n**, as with

$$\tau = K\varepsilon^n \tag{5.3}$$

A Newtonian fluid is defined as a fluid where the shear stress is directly proportional to the strain rate (**n** = 1). If **n** is greater than one, the local shear stress is greater than that of a Newtonian fluid, thus the situation is termed "shear thickening". The opposite case, "shear thinning", occurs when **n** is less than one. As a timeline is convected downstream in a developing flow, the region closest to a pipe or channel centerline is not initially affected by the no-slip condition at the wall. This results in a "flat" region of the timeline in the neighborhood of the pipe or channel center. Shear thinning fluids tend to maintain this flat region further downstream.

It is to be assumed that the flow in the channel or pipe is steady and fully developed. In this case, a force balance shows that the shear stress is a linear function of the distance from the center of the pipe or channel. Patel [5] integrates Equation 5.3 for the two components of the shear stress giving the following results:

$$\frac{V}{V_{max}} = 1 - Y^m \qquad \text{where} \qquad m = \frac{n+1}{n} \tag{5.4}$$

where V is the fluid velocity in the flow direction, V_{max} is the maximum fluid velocity (located at the centerline), Y is the nondimensional distance from the pipe (Y = r/R) or channel (Y = 2y/h) centerline, and **m** is defined as the velocity profile exponent. Values of **m** greater that 2 correspond to shear thinning fluids, while values less than 2 correspond to shear thickening fluids. If m equals 2, the fluid is Newtonian.

The ratios of the average fluid velocity to the maximum fluid velocity for both the two-dimensional channel and the three-dimensional pipe are calculated by the program. Also determined is the momentum flux correction factor, ß. Expressions for these values are listed below.

2D: $$\frac{V_{avg}}{V_{max}} = \frac{m}{m+1} \qquad \beta = \frac{2(m+1)}{(2m+1)} \tag{5.5}$$

3D: $$\frac{V_{avg}}{V_{max}} = \frac{m}{m+2} \qquad \beta = \frac{m+2}{m+1} \tag{5.6}$$

On execution of the program, the user is prompted to enter a value of **n** for the fluid power law relation. Timelines are then plotted downstream at various time intervals. The user may have the previous timelines erased, thus displaying only the latest one. Also, the program may be made to pause between the plotting of the timelines.

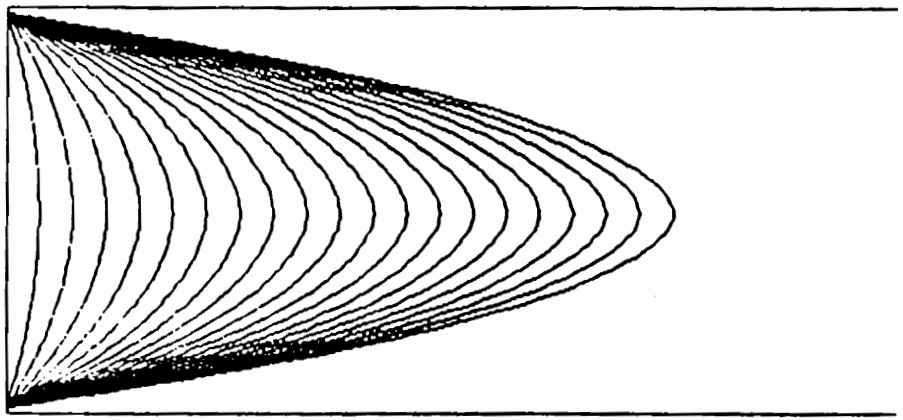
The fluid power law exponent, **n**, and the velocity profile exponent, **m**, are written to the screen. Additionally, values for V_{avg}/V_{max} and ß are displayed for both channel and pipe flow. If the previous timelines have been erased, the location of V_{avg} is shown for both two and three-dimensional flows. Figure 5.3 shows the timelines advected downstream for a Newtonian fluid.

n= 1.00

Velocity profile exponent = 2.0000

2D: Vavg/Vmax = 0.667 ; Beta = 1.200

3D: Vavg/Vmax = 0.500 ; Beta = 1.333



ENTER "Y" TO RERUN

FIGURE 5.3: Timelines Advected Downstream By A Newtonian Fluid

5.3 DEFORM

The analysis of the deformation of a fluid particle is usually performed on a square fluid element as in White [6]. In this program a two-dimensional square element of an incompressible fluid is considered. Four types of motion and the resultant deformation of a fluid element are possible: (1) translation, (2) rotation, (3) extensional strain or dilation, and (4) shear strain. Since translational motion is trivial, only the last three kinds of deformation are demonstrated in this program.

The following are the expressions for the rotation rate, ω_z, shear strain rate, ε_{xy}, and the extensional rates, ε_{xx} and ε_{yy}, in the X and Y directions respectively.

$$\omega_z = \frac{1}{2}\left[\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right]$$

$$\varepsilon_{xy} = \frac{1}{2}\left[\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right]$$

$$\varepsilon_{xx} = \frac{\partial u}{\partial x} \qquad \varepsilon_{yy} = \frac{\partial v}{\partial y} \tag{5.7}$$

For the case of an incompressible fluid, the continuity equation requires that $\varepsilon_{xx} = -\varepsilon_{yy}$. Therefore, the user needs to enter only the three quantities: ω_z, ε_{xy}, and ε_{xx}. The solutions employed in the program for the velocities in the X and Y directions, u and v respectively, are

$$u = \frac{\partial x}{\partial t} = Cx + By \qquad v = \frac{\partial y}{\partial t} = Ax + Cy \tag{5.8}$$

where

$$A = \varepsilon_{xy} + \omega_z \qquad B = \varepsilon_{xy} - \omega_z \qquad C = \varepsilon_{xx} = -\varepsilon_{yy} \tag{5.9}$$

Since a nontranslating, incompressible fluid is being considered, it is necessary only to solve for the corner positions of the fluid particle. These are then plotted relative to an origin fixed at the element's center. Lines are then drawn connecting the corner points.

There are three types of solutions to the deformations depending on the sign of $S = AB + C^2 = \varepsilon_{xy}^2 + \varepsilon_{xx}^2 - \omega_z^2$. If $S < 0$, then rotation dominates and a periodic solution is found where the fluid element repeatedly elongates and returns to its original shape. The case where $S > 0$ causes the fluid element to rotate, while flattening exponentially. Finally, if $S = 0$, no rotation occurs as the corner points progress linearly with time causing the element to flatten in one direction.

Once the user has entered values for $\omega_z, \varepsilon_{xy}$ and ε_{xx}, an initially square particle in displayed on the screen. Every time the user presses the RETURN key, one time step passes, and a new fluid element shape is calculated and displayed, while the old element is erased. If the RETURN key is held continuously, the program animates the deformation. If the fluid element grows too large to fit within the screen boundaries, the program will reduce the size of the particle by half. Figure 5.4 is a sample program output, where the earlier fluid elements have not been erased to give the reader an appreciation of the progressive deformation.

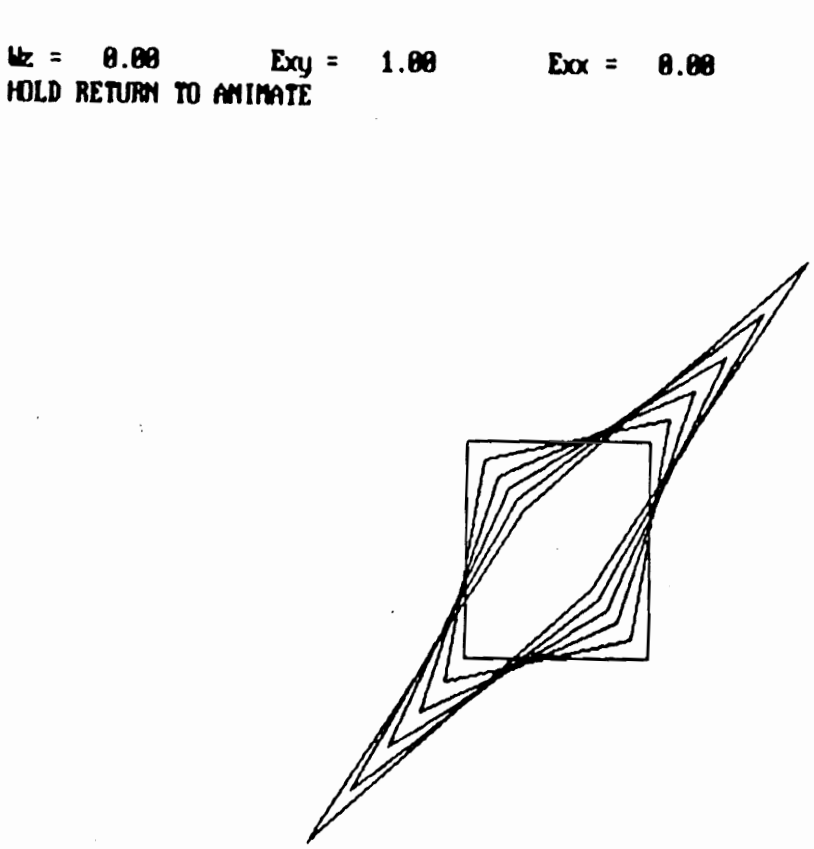


FIGURE 5.4: Progressive Deformation of an Initially Square Fluid Element

5.4 WATERJET

A common example demonstrating an application of the conservation of linear momentum for a control volume is a waterjet impinging upon a deflecting surface. This example demonstrates that a force is exerted on the surface as a result of the change in direction of the momentum of the fluid. Figure 5.5 shows a waterjet impinging a surface. The waterjet and the surface have constant velocities V_j and V_s respectively. The deflection occurs within the control volume shown, and the deflection angle is θ. Normal vectors are drawn at the fluid entry and exit points into and out of the control volume. The outward pointing normals are essential in determining the sign of the momentum vector used in the momentum equation.

When running the program, the user must first choose which system of units is to be used for input values and calculations. The velocities of the jet and surface must be supplied next, with the condition that the jet velocity is greater than that of the surface. The diameter of the jet, D_j, and the deflection angle, θ, are also user specified quantities. The fluid considered is water.

The program calculates a relative velocity $V_R = V_j - V_s$, and the flow rate, Q, where Q is the rate at which fluid enters and leaves the control volume which surrounds the deflection surface. The force on the plate is found using the conservation of linear momentum principle, with the following scalar result.

$$F = \rho Q V_R \sin\theta \tag{5.10}$$

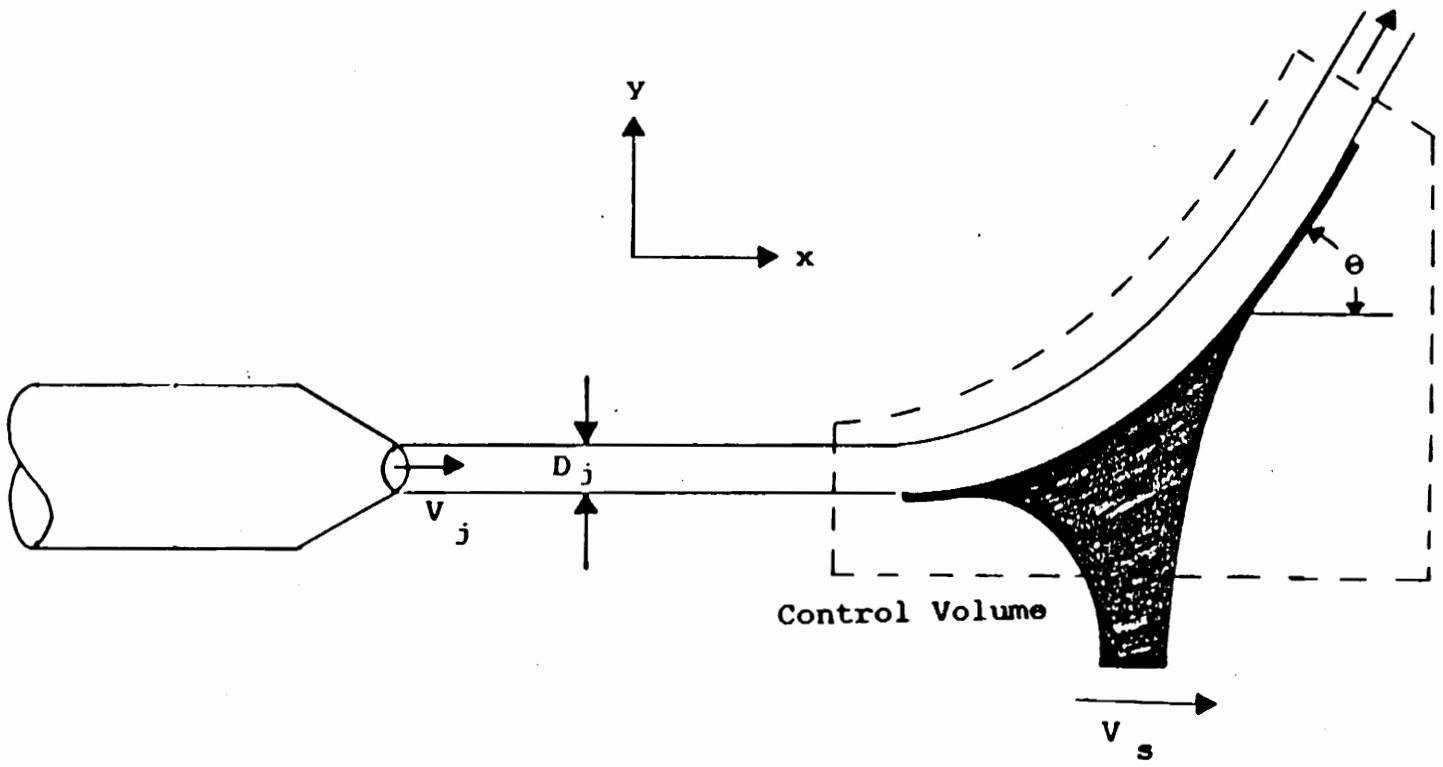


FIGURE 5.5: Water Jet Impinging on a Deflection Surface

Components of the force are

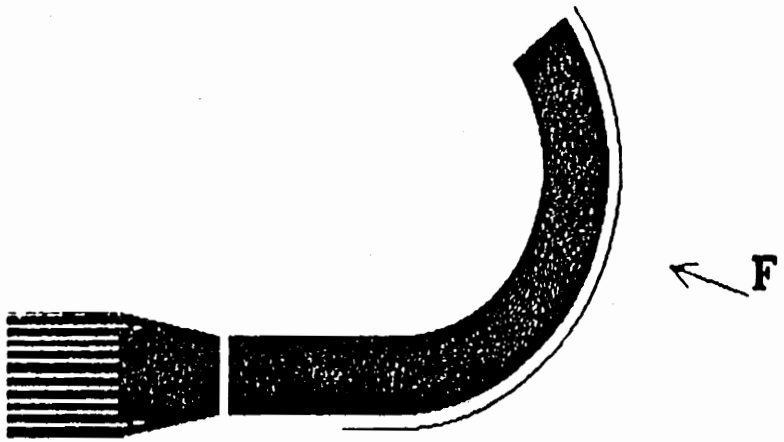
$$F_x = F\cos\phi \qquad F_y = F\sin\phi \qquad \phi = \theta + \pi/2 \tag{5.11}$$

where θ is the deflection angle. The vector representation of the force and its components are shown in Figure 5.5.

Output from the program includes the quantities described above, and a schematic of the jet and surface. Forces will be converted to kN or kips, as necessary. The water is shown deflecting through the angle, θ, and the force is placed along a line of action corresponding to its components. Figure 5.6 is output for the case of θ = 135 degrees, V_j = 8 m/s, V_s = 3 m/s, and D_j = 0.25 m.

```
      WATER JET
-----------------------

Angle:     135.0   deg
Vrel:      5.000   m/s
D:         0.250     m
-----------------------
Q:         0.393 m^3/s
F:         2.263    KN
FX:        2.091    KN
FY:        0.866    KN
```



ENTER "Y" TO RERUN

FIGURE 5.6: Sample Output for Program WATERJET

CHAPTER 6: POTENTIAL FLOW

Potential flow deals with an idealized model of fluid flow that is steady, two-dimensional, incompressible, inviscid, and irrotational. Many types of flows may be approximated with potential theory. In high Reynolds number flows about streamlined bodies, viscous effects are usually confined to thin regions called boundary layers adjacent to body surfaces. Actual flow solutions may combine potential and boundary layer flow theory to obtain better approximations to the real flow. Potential theory is likely to fail in regions very close to physical bodies where boundary layers occur, and downstream, directly behind bluff bodies where wake and separation regions are likely to occur. In this chapter several two-dimensional steady flows will be examined. In all of these flows, the density is taken to be constant, viscosity effects are neglected, and no body forces are considered.

Potential theory is based on the existence of a velocity potential and a stream function which satisfy Laplace's equation. The stream function is the parameter of choice for these programs since it better conveys the concept of fluid motion to the beginner. Derivatives of the stream function, ψ, are related to the velocity components with

$$u = \partial\psi / \partial y \qquad v = -\partial\psi / \partial x \tag{6.1}$$

where u and v are the rectilinear flow velocities in the X and Y directions respectively. In cylindrical coordinates, the velocity components are given by

$$v_r = \frac{1}{r}\frac{\partial \psi}{\partial \theta} \qquad v_\theta = -\frac{\partial \psi}{\partial r} \tag{6.2}$$

In the following table, the stream function is defined for several elementary flows. To produce more interesting flows it is possible to superpose any number of these stream functions. The total stream function is then defined as the sum of the individual stream functions.

TABLE 6.1: Stream Function Definitions for Elementary Plane Flows

Flow Condition	Stream function, ψ	
	Rectilinear	Polar
Uniform Stream	U_∞ y	U_∞ r SIN θ
U_∞ is a constant equal to the uniform flow velocity.		
Source / Sink	m ARCTAN(y/x)	m θ
m is the source or sink strength. It is positive for a source and negative for a sink.		
Vortex	$-K \ln\left(\frac{x}{\cos\theta}\right)$	- K ln r
K is the circulation Γ, divided by 2π. The vortex is counterclockwise for a positive K value.		

Programs DOUBLET, SHEET, and CORNER are elementary plane flow solutions which do not create closed bodies. HALFBODY, RANKINE, KELVIN, and CYLINDER concern closed bodies created through superposition. THINFOIL and KTFOIL are intended as an introduction to airfoil theory.

A more detailed discussion of potential theory may be found in White [4], Shames [3], or Milne-Thompson [7].

6.1 DOUBLET

A source sink pair is formed by placing a source of strength **m** at a distance **-a** along the X axis while a sink of strength **-m** is placed at X = **a**. Superposition results in the following stream function equation,

$$\psi = \frac{-2mya}{x^2 + y^2 - a^2} \tag{6.3}$$

the derivation of which may be found in White [4].

If the distance **a** is allowed to approach zero as the source/sink strength becomes infinite, the resulting flow is termed a doublet. The product **2 a m** is given a new symbol, μ, the doublet strength, and the resulting stream function is

$$\psi = \frac{-\mu y}{x^2 + y^2} \tag{6.4}$$

The program simulates the creation of a doublet by bringing a source and a sink together at the origin. The doublet streamlines are a set of circles tangent to the X axis. Figure 6.1 shows the doublet created by the program.

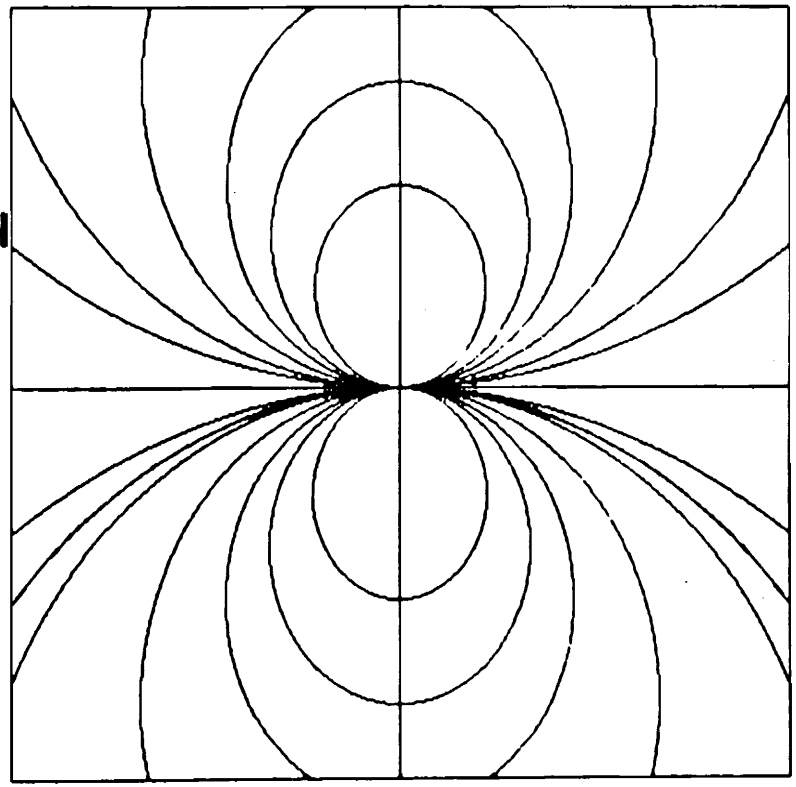
No user input is required for the program, which simply animates the creation of a doublet. This is done by drawing six flow fields with the distance, **a**, being 5, 4, 3, 2, 1, and 0 for the separate flow fields. Each field is plotted in black so that it does not appear on the screen. When the user

COMPUTING

READY

PRESS RETURN
FOR DISPLAY

A = 5
A = 4
A = 3
A = 2
A = 1
DOUBLET



ENTER "Y" TO RERUN

FIGURE 6.1: Creation of a Doublet

presses the RETURN key, the streamlines corresponding to a particular **a** value are changed to blue. In this way the user may pause between flow fields or pass through them in rapid succession, thus animating the creation of the doublet.

Each of the source sink pairs is plotted using a nondimensionalized stream function, S = $\frac{\psi}{m}$. The stream function equation is rearranged for x.

$$x = \left[\frac{2ay}{\tan(-S)} - y^2 + a^2 \right]^{\frac{1}{2}} \tag{6.5}$$

For a given stream function value, the Y coordinate is incremented from the origin to the top of the screen while the corresponding X coordinates are calculated. The points are then plotted in rectilinear coordinates. A series of circles represents the doublet flow (**a** = 0).

6.2 SHEET

If an infinite series of identical counterclockwise vortices are placed along the X axis, spaced a distance, **a**, apart, a vortex sheet is formed. When the vortex sheet is viewed from afar, the flow appears to be uniform in the negative X direction above the X axis, and uniform in the positive X direction below the X axis. Figures 6.2 and 6.3 illustrate the distributed vortices and the flow seen from a distance.

At a given point in the flow field, the stream function influence of an individual vortex is

$$\psi_i = K \ln r_i \tag{6.6}$$

where K is the vortex strength and r is the radial distance from the ith vortex to the point. The stream function sum of all the vortices is then

$$\psi = K \sum_{i=1}^{\infty} \ln r_i = -\frac{1}{2} K \ln\left[\frac{1}{2}\left(\cosh\frac{2y}{a} - \cos\frac{2x}{a}\right)\right] \tag{6.7}$$

where the stream function is expressed in rectilinear coordinates.

The spacing, **a**, is chosen by the user. Larger values of **a** are equivalent to a "close up" view of the sheet. As **a** is decreased, the sheet begins to resemble the bilateral uniform flow.

Given the stream function in Equation 6.7, streamlines are plotted in the following manner. A nondimensional stream function is defined as $S = -2\frac{\psi}{K}$. Values of S are found by substituting points along the Y axis into

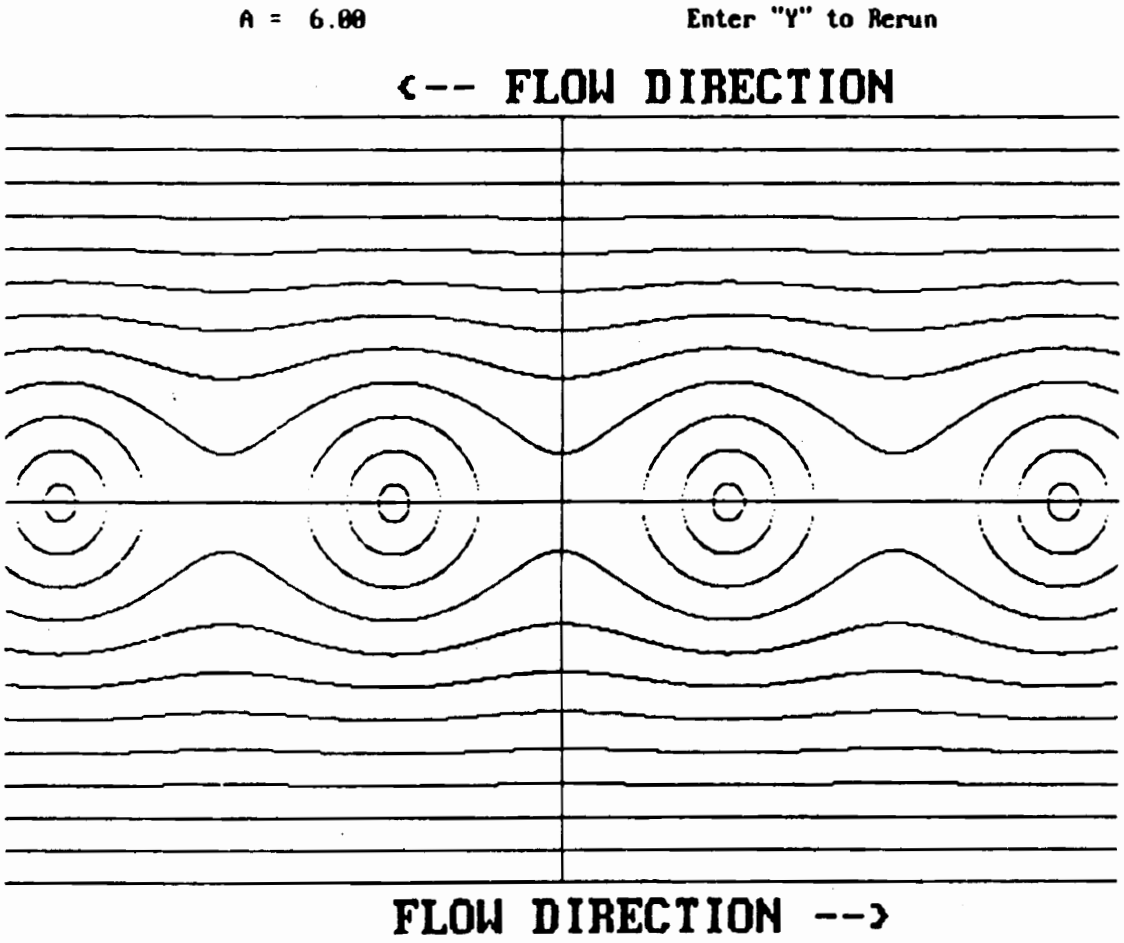


FIGURE 6.2: Distributed Vortices Generating a Vortex Sheet

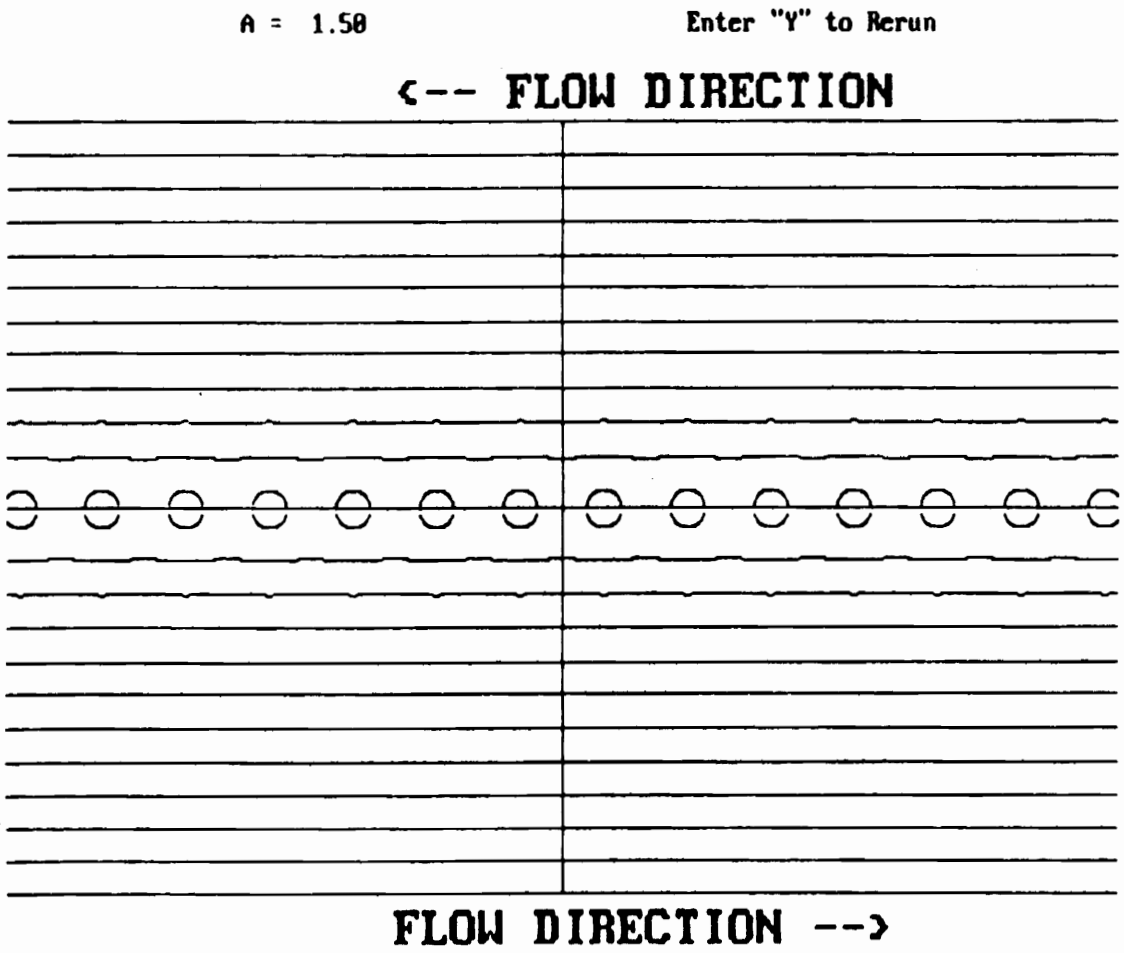


FIGURE 6.3: A Vortex Sheet Viewed from Afar

the stream function equation. The stream function is rearranged such that the Y coordinate is the independent variable.

$$y = \frac{a}{2\pi}\ln\left(b + \sqrt{(b^2 - 1)}\right) \tag{6.8}$$

$$where \quad b = 2\exp\left(\frac{2S}{K}\right) + \cos\frac{2\pi x}{a}$$

Given a value of S, the X coordinate is incremented while values of the Y coordinate are calculated. Points are then plotted in rectilinear coordinates.

6.3 CORNER

Corner flow cannot be easily represented as a combination of sources, sinks, and vortices. However, corner flow may be represented by transforming a uniform flow from the ζ plane into the z plane with $z = \zeta^n$. The resulting stream function has a simple representation,

$$\psi = Ar^n \sin\theta \tag{6.9}$$

where both A and n are constants. The flow turns through an angle, β, which is equal to $\frac{\pi}{n}$. Since the largest possible value of Beta is 360 degrees, corner flow does not exist for n < 1/2. Realistically, flows with β greater than 180 degrees (1/2 < n < 1) are likely to separate. This is true except for flow wih extremely low Reynolds numbers (creeping flows).

Flow around a corner may be viewed as half of a stagnated flow directed towards a corner, where one of the corner walls is the stagnation streamline. For stagnated flow, the flow encounters a corner of angle $2\beta = \frac{2\pi}{n}$. Except for the inviscid assumption, the potential theory treatment of flow directed towards a corner is very realistic. Examples of both flow around a corner, and stagnated flow are shown in Figure 6.4. If the inviscid assumption is dropped and the corner angle, β, is greater than 90 degrees, flow past a wedge results. The viscous boundary layer flow over a wedge is treated in program WEDGE, yet to be discussed (Viscous Flow, Chapter 7).

CORNER ANGLE = 80.0

ENTER "Y" TO RERUN

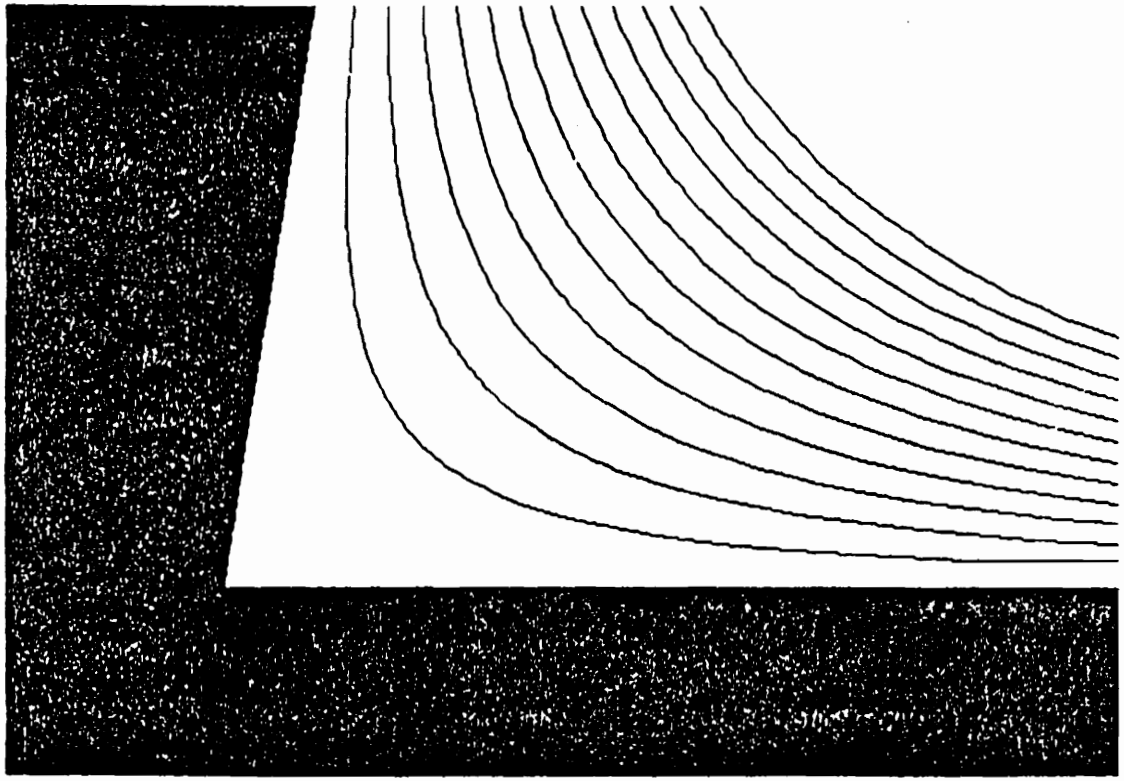


FIGURE 6.4a: Potential Flow Around a Corner

CORNER ANGLE = 240.0

ENTER "Y" TO RERUN

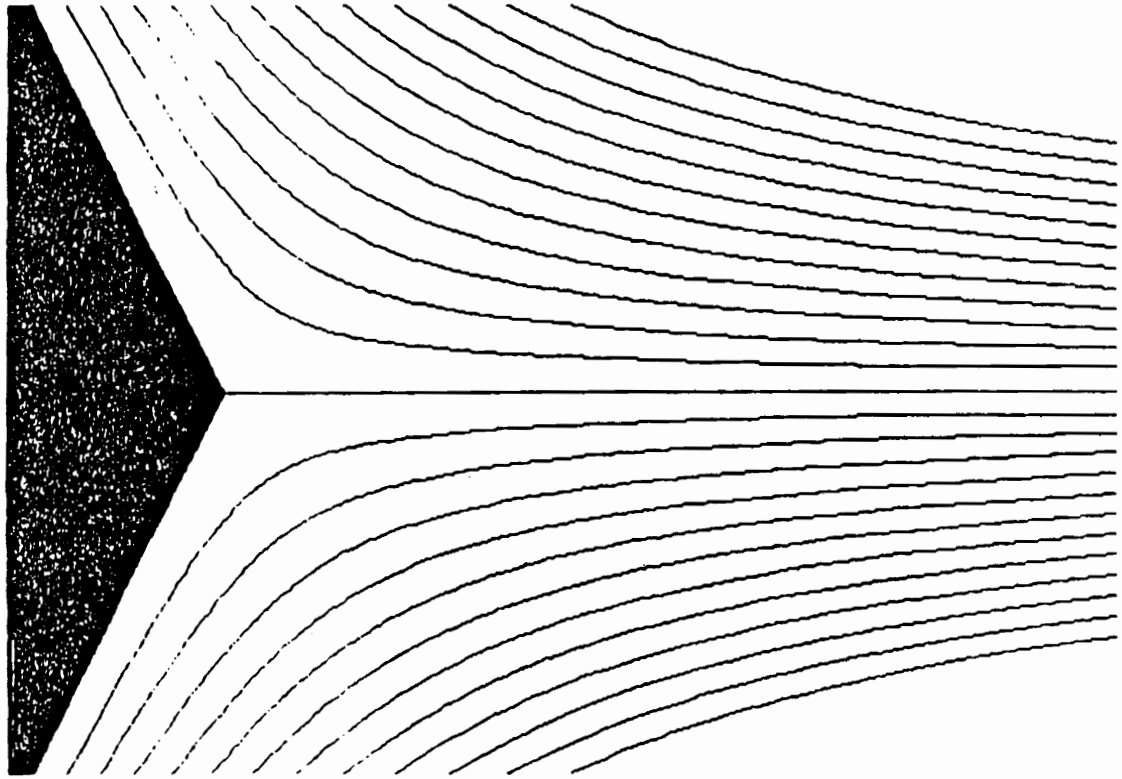


FIGURE 6.4b: Potential Flow Normal to a Corner

The user chooses either flow around or directed towards a corner, and a value of β is entered. A nondimensionalized stream function, $S = \psi/A$, is used to plot the streamlines. The stream function is solved for R. θ is then incremented from zero to β and the corresponding R values are found, given a calculated value of S.

$$R = \left(\frac{S}{\sin \theta} \right)^{1/n} \tag{6.10}$$

If the streamlines for flow directed towards a corner are to be plotted, the stagnation streamline is drawn first. Then the streamlines and their mirror images are plotted together. In addition to the stagnation streamline, thirteen streamlines, including their mirror images, as necessary, are plotted.

6.4 HALFBODY

A Rankine half-body is formed by placing a source in a uniform flow. It is convenient to locate the source at the origin of the coordinate axes, with the uniform flow directed in the positive X direction. In this case the combined stream function is

$$\psi = U_\infty r \sin\theta + m\theta \tag{6.11}$$

where U_∞ is the free stream velocity and m is the source strength. The half-body itself is formed by the stagnation streamline. Since no flow may cross a streamline, the stagnation streamline represents the half-body placed in a uniform flow with slip boundaries. This means that viscous effects do not occur at its surface.

One stagnation point occurs for this flow. At a distance, **a**, in the negative X direction, the flow velocity is zero. This distance, **a**, is termed the "stagnation radius". It is easily shown, using the definition of the stream function, that $a = -m/U_\infty$. The streamline passing through this point is called the stagnation streamline, and for the Rankine half-body, $\psi_s = \pm\pi m$. It is noted that the boundaries of the half-body do not continue to grow in the Y direction, but approach asymptotes of $y = \pm\pi a$. A diagram of a Rankine half-body including the stagnation radius and asymptotes is shown in Figure 6.5. A complete treatment of the half-body may be found in White [4].

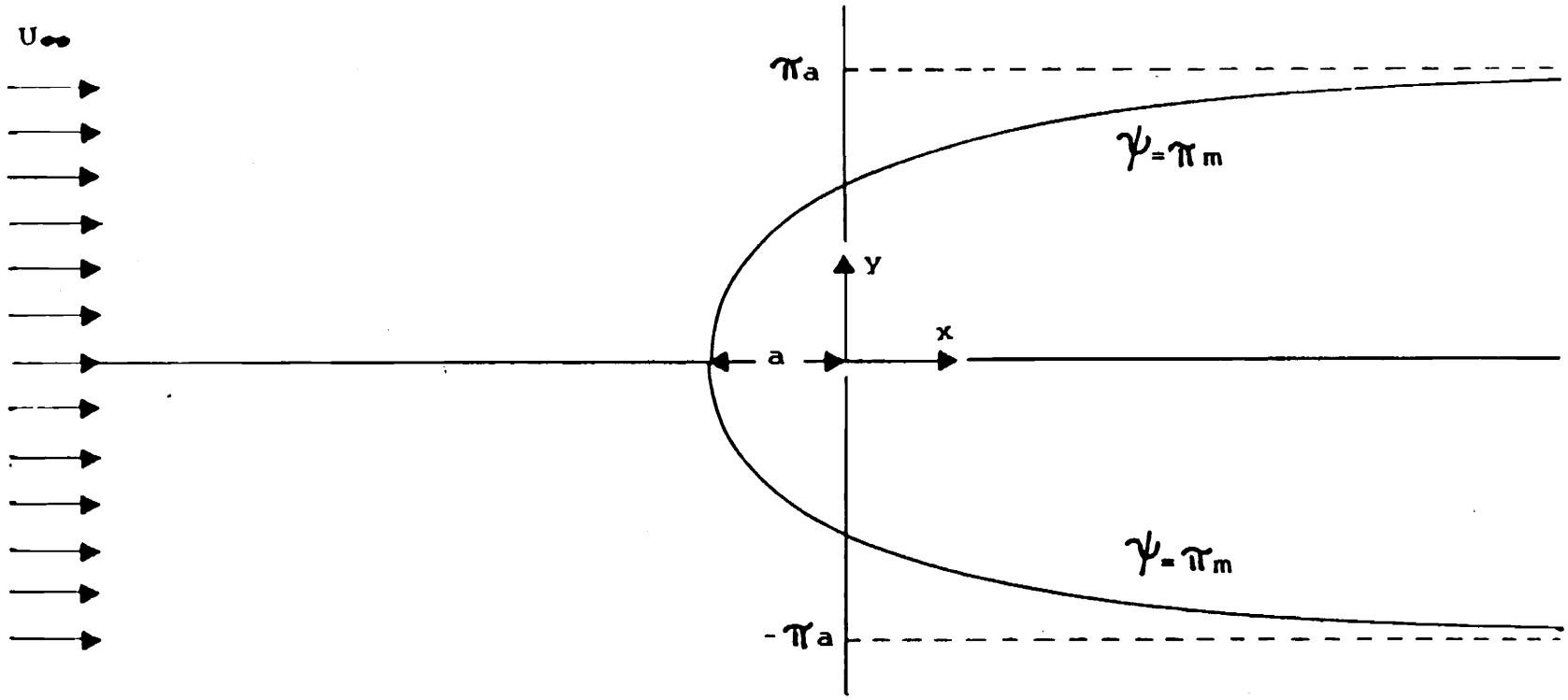


FIGURE 6.5: The Rankine Half Body

A nondimensionalized form of the stream function is employed in the program.

$$S = UR\sin\theta + \theta \tag{6.12}$$

The nondimensional variables are defined as follows: S = ψ/m, $U = U_\infty a/m$, and R = r / **a**. In this case the stagnation radius is equal to 1/U and, the body width asymptotes are at π/U. The nondimensionalized stagnation stream function value is π.

The user inputs a value for the nondimensional free stream velocity, U. Values for **a** and the asymptotes are listed. An axis is then plotted from x = -10.0 to +10.0 and y = -7.5 to +7.5. U values less than 0.1 are not permitted since this would result in the stagnation point being off the screen. Larger values of U tend to make the body more slender while smaller values of U result in the body becoming fuller. Interior streamlines are plotted in white, the body streamline in purple, and the exterior streamlines in green. The axes are in blue, and the asymptotes are dashed orange lines. Figure 6.6 shows a half-body constructed by the program for U = 1/2.

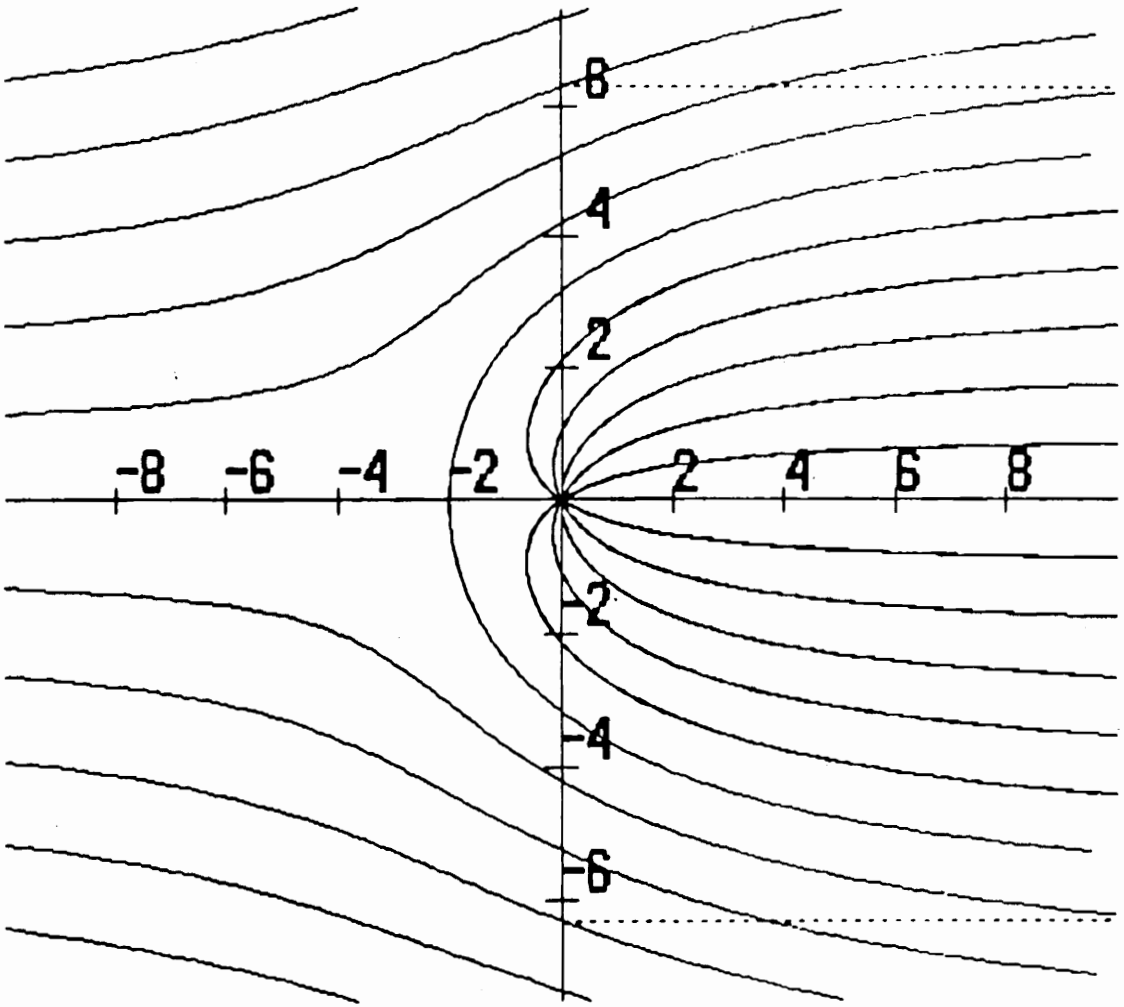


FIGURE 6.6: Sample Output for Program HALFBODY
(U = 0.5)

6.5 RANKINE

If a source and a sink are respectively placed at locations x = -a and x= +a in a positively directed uniform flow, the resulting flow field forms a Rankine Oval. It is easy to see that the Oval is an extension of the half-body except that in this case, **a** is not the stagnation radius. Rather, it describes the spacing of the source-sink pair. For the Rankine Oval, expressed in both rectilinear and cylindrical coordinates, the total stream function is

$$\psi = U_\infty r \sin\theta + m(\theta_1 - \theta_2) = U_\infty y - m \tan\left[\frac{2ay}{x^2 + y^2 - a^2}\right] \qquad (6.13)$$

Figure 6.7 demonstrates the coordinate systems used to construct the body. Note that the stagnation streamline has the value $\psi = 0$.

Unlike the half-body, the Oval possesses two stagnation points, located at

$$X_s = \pm a \left(1 + \frac{2}{Ua}\right)^{1/2} \qquad (6.14)$$

Since the stagnation streamline is closed, it may be thought of as the boundary of a solid body. This is another example of an inviscid approximation to flow around a body.

The accuracy of the inviscid approximation for flows past solid bodies is limited. Success is improved at high Reynolds numbers where viscous effects are concentrated in thin boundary layers. Unfortunately, at high

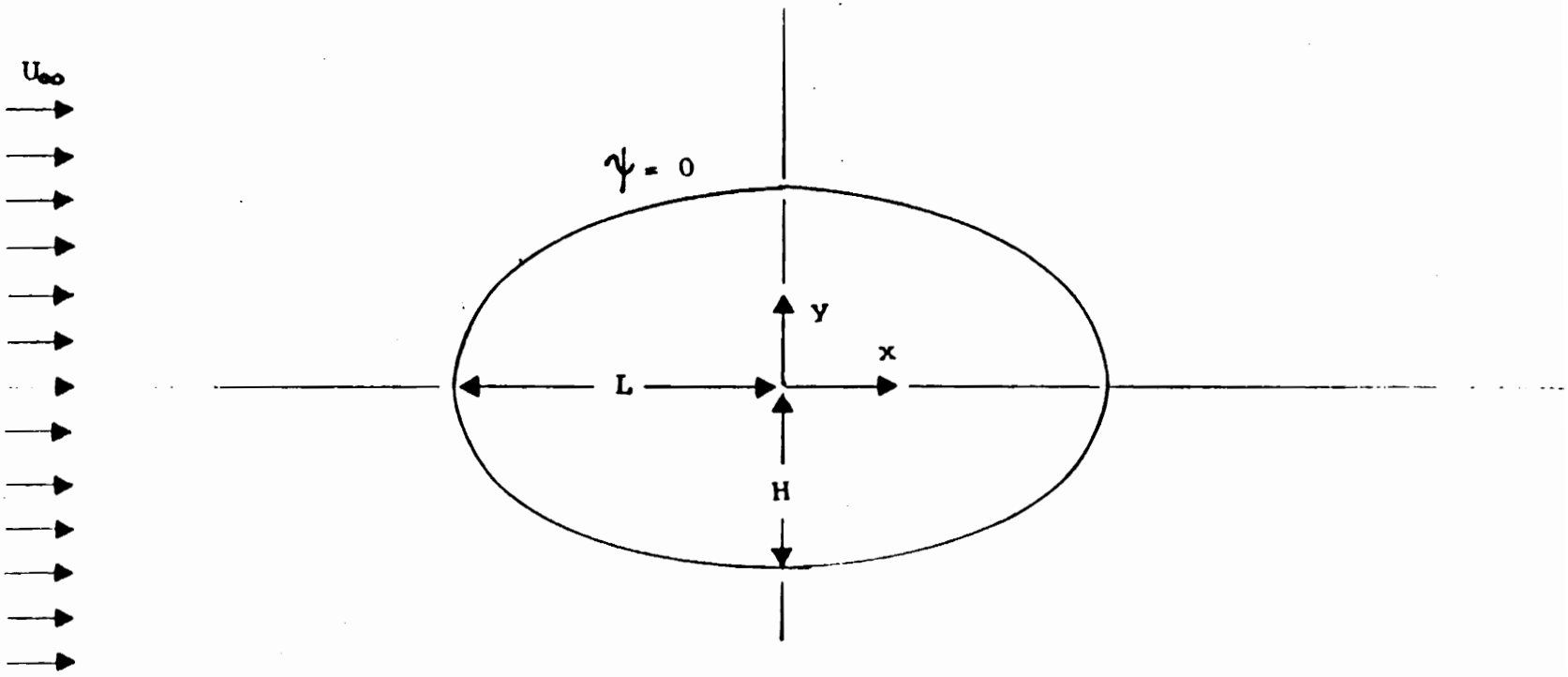


FIGURE 6.7: The Rankine Oval

Reynolds numbers, separation is likely to occur downstream of thicker bodies. If low Reynolds numbers are considered, the likelihood of separation decreases, but viscous effects extend further into the flow.

A nondimensionalized version of the stream function is used in this program

$$S = U\,Y - \arctan\left[\frac{Y}{X^2 + Y^2 - 1}\right] \qquad (6.15)$$

where $S = \psi / m$, $U = U_\infty\, a/m$, $X = x/a$, and $Y = y/a$. The distance between the source and sink is set at a constant value of two ($a = 1$).

The user inputs a value of the nondimensional free stream velocity, U. As U approaches zero, the body becomes circular, while large values tend to make the Oval more slender. For values of U less than 0.35 only the interior streamlines are plotted (red). Both interior (red) and exterior (green) streamlines are plotted if the value of U is greater than or equal to 0.35 and less than or equal to 2.5. If the value of U is greater than 2.5 then only the exterior streamlines are plotted (green).

S values are found by substituting coordinate values along the Y axis into the stream function equation. Coordinates are found as fractions of the shoulder height, which is defined as the maximum extent of the body in the Y direction. Its mathematical definition in both ordinary and non-dimensional form is

$$\frac{h}{a} = \cot\frac{h/a}{2m/U_\infty a} \qquad H = \cot\frac{H}{2/U} \qquad (6.16)$$

A Newton iteration is used to determine the value of H. Streamlines are plotted in the first quadrant by incrementing Y and solving for X. Mirror images are constructed in the other four quadrants. Figure 6.8 shows a Rankine Oval created by the program for U = 2.

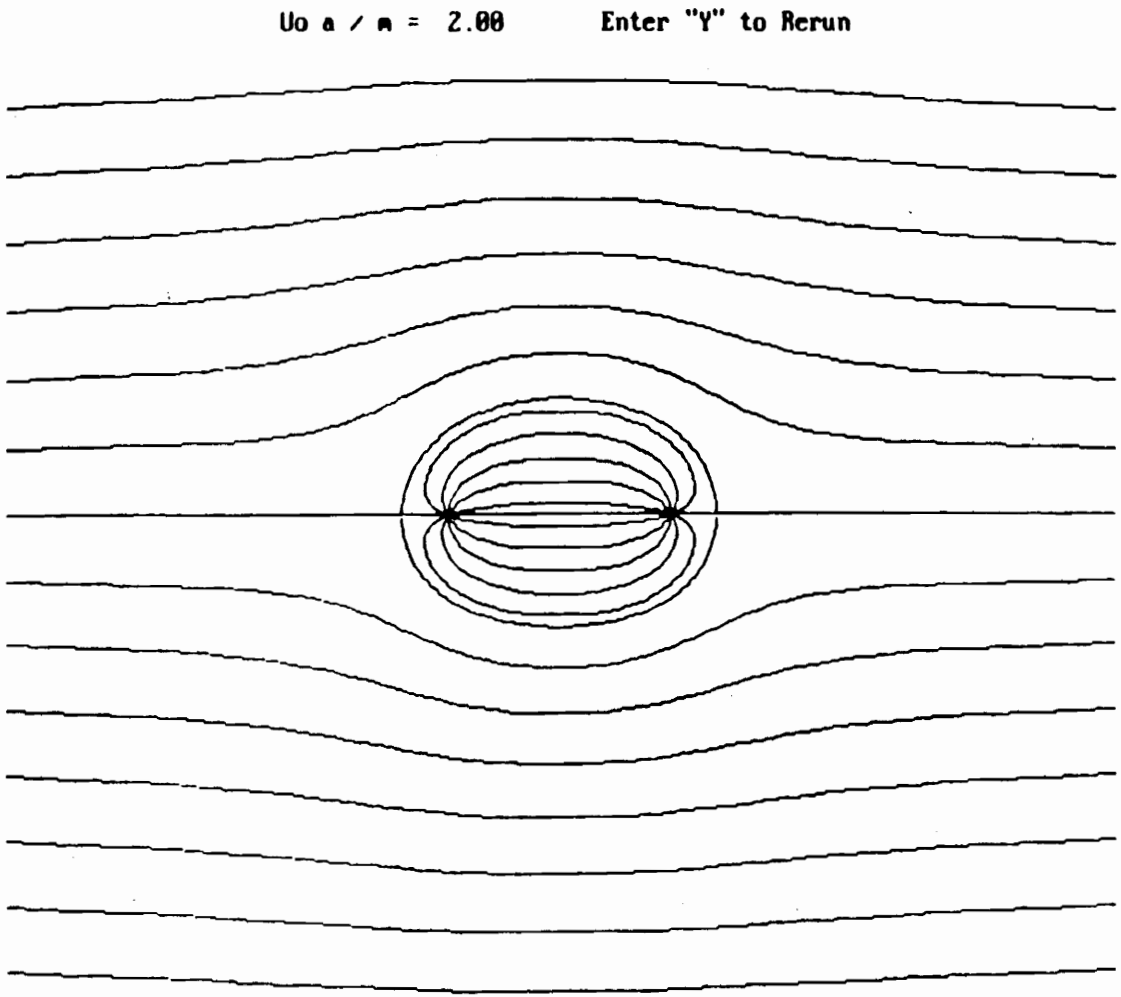


FIGURE 6.8: Sample Output for Program RANKINE

6.6 KELVIN

A Rankine Oval represents the flow past a body which has its long axis in the same direction as the flow. On the other hand, a Kelvin Oval represents flow past a body which has its long axis oriented perpendicular to the free stream. It is produced by combining a free stream flowing in the positive X direction, a clockwise vortex at y = **+a** and, a counterclockwise vortex of equal strength at y = **-a**. The stream function for the oval is then

$$\psi = U_\infty y - \frac{K}{2}\ln\left[\frac{x^2+(y-a)^2}{x^2+(y+a)^2}\right] \tag{6.17}$$

where U is the free stream velocity and $K = \Gamma/2\pi$ is the vortex strength.

As with the Rankine Oval, potential theory fails to predict separation downstream of the Kelvin Oval. In fact, separation is more likely to occur downstream of the Kelvin Oval due to its long axis being perpendicular to the flow.

Again, a nondimensional stream function is employed in the program

$$S = UY + \ln\left[\frac{X^2+(Y-1)^2}{X^2+(Y+1)^2}\right] \tag{6.18}$$

where $S = \psi/m$, $U = 2U_\infty a/m$, X = x/a, and Y = y/a. The distance **2a** is held constant at one third the screen height.

The user inputs a value for the nondimensional free stream velocity between 0.4621 and 4.0. Values of U less than 1/3ln(4) = 0.4621 are not

permitted since the body streamline would exceed the screen boundaries. For values of U less than 0.65 only the interior streamlines will be plotted (purple). If the value of U is greater than 0.65, both interior (purple) and exterior (blue) streamlines are plotted. Smaller values of U correspond to ovals which are fuller and more circular. Larger values of U make the body more slender. Eventually, when U = 4, the Kelvin Oval will squeeze together at its center and become two separate regions of flow both above and below the X axis, resembling a figure eight. Since only oval shapes are considered, the program will not accept values of U larger than this limit.

Negative values of S correspond to streamlines inside the body (purple), while positive S streamlines exist outside the body (blue). The stagnation streamline (S = 0) is plotted in yellow.

Figure 6.9 shows a Kelvin Oval generated by the program for U = 1. Both interior and exterior streamlines are plotted.

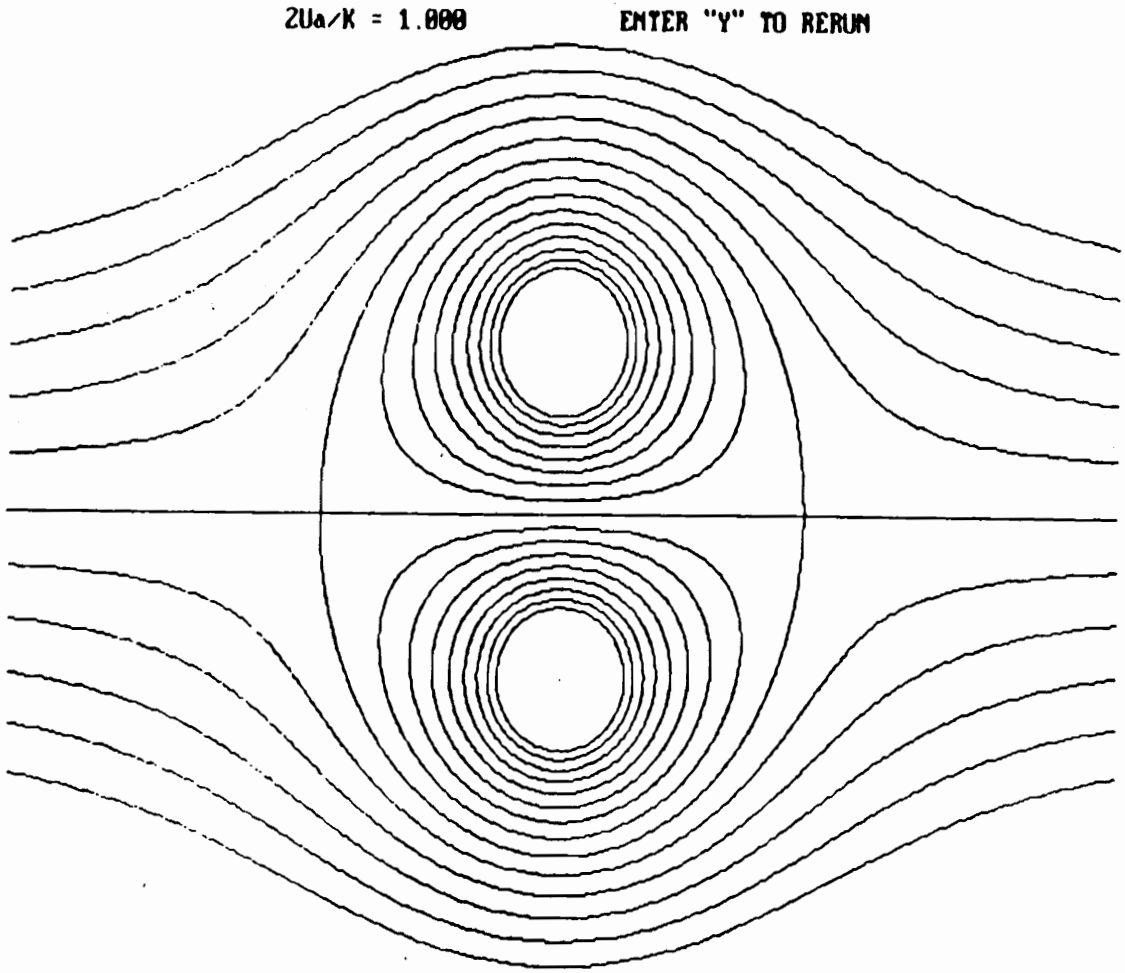


FIGURE 6.9: Sample Output for Program KELVIN

6.7 CYLINDER

Perhaps the potential flow solution most thoroughly studied in undergraduate fluid mechanics is the flow past a circular cylinder with circulation (White [4]). Such a body is created by combining a uniform flow directed in the positive X direction with a doublet and a vortex both placed at the origin. In order to produce lift, the vortex circulation must be nonzero. No lift is predicted by potential theory if $\Gamma = 0$. Since viscous effects are ignored in potential theory, no viscous drag is predicted. The stream function for flow past a cylinder is

$$\psi = U_\infty \sin\theta\left(r - \frac{a^2}{r}\right) - \frac{\Gamma}{2\pi}\ln\frac{r}{a} \tag{6.19}$$

where **a** is the cylinder radius and Γ is the vortex circulation. When the circulation is zero, the flow field is symmetric about both axes, while if it is nonzero, the pattern is symmetric about the Y axis only.

As with the Rankine and Kelvin ovals, potential theory fails to predict separation downstream of the cylinder. Therefore, the inviscid treatment of the flow past a cylinder with circulation is more valid in modeling the flow at higher Reynolds numbers with thin viscous layers for the forward portion of the body.

For the case of zero circulation, stagnation points occur at x = +a and x = -a. For the case of nonzero circulation when $|K| = \Gamma/2\pi < 2$ the stagnation points are located (in cylindrical coordinates) on the body surface at the two solutions to the following equation

$$\sin\theta_s = \frac{K}{2U_\infty a} \tag{6.20}$$

where the stagnation angles are measured counterclockwise from the positive X axis. If |K| > 2, then only a single stagnation point exists, but not on the cylinder body. Its location is along the Y axis at the value

$$y_s = \frac{a}{2}\left[\beta + \left(\beta^2 - 4\right)^{\frac{1}{2}}\right] \quad where \quad \beta = \frac{K}{U\,a} \tag{6.21}$$

However, this case does not represent flow about a circular cylinder and is of no practical interest.

The nondimensional form of the stream function used in the program is given by

$$S = \left(R - \frac{1}{R}\right)\sin\theta - K\ln R \tag{6.22}$$

where S = ψ/ma, R = r/a, and $K = \Gamma/2\pi U_\infty a$. The stagnation streamline has a value S = 0 and forms a circle of radius R = 1, and the stagnation point(s) are located as stated earlier. Since what occurs within the cylinder boundary is unimportant, streamlines within the cylinder are not drawn. Thus, the body is drawn as a solid yellow circle. Streamlines are blue,

except for the stagnation streamline which is purple. For a fixed value of S, a single streamline is constructed by incrementing R and solving for θ. K is the parameter specified by the user. Negative values of K result in higher velocity flow above the cylinder with the lift force tending to force the cylinder in the positive Y direction. The reverse is true for positive K inputs. Since the flow field is symmetric about the Y axis, streamlines are plotted in the first and fourth quadrant and mirrored into the second and third. The stagnation points are found by the method discussed earlier. Figure 6.10 shows flow fields generated by the program for four different values of K.

Once streamlines have been drawn, the pressure coefficient is plotted in rectilinear coordinates. Cp is the nondimensional pressure coefficient on the surface of the cylinder, and is given by

$$Cp = \frac{P - P_{atm}}{1/2 \rho U^2} = 1 - 4\sin^2\theta + 4K\sin\theta - K^2 \qquad (6.23)$$

Distributions for both the pressure (low speed) region, and suction (high speed) sides of the cylinder are plotted on the same axes as shown in Figure 6.11.

A subsequent program involves the mathematical transformation of a cylinder with circulation to more realistic airfoil shapes. The Karman-Treffetz transformation maps the cylinder into new coordinates resulting in a Karman-Treffetz airfoil. A subcase of the Karman-Treffetz transformation is the Joukowski transformation, which also generates more realistic airfoil shapes. Both of these transformation are considered in program KTFOIL.

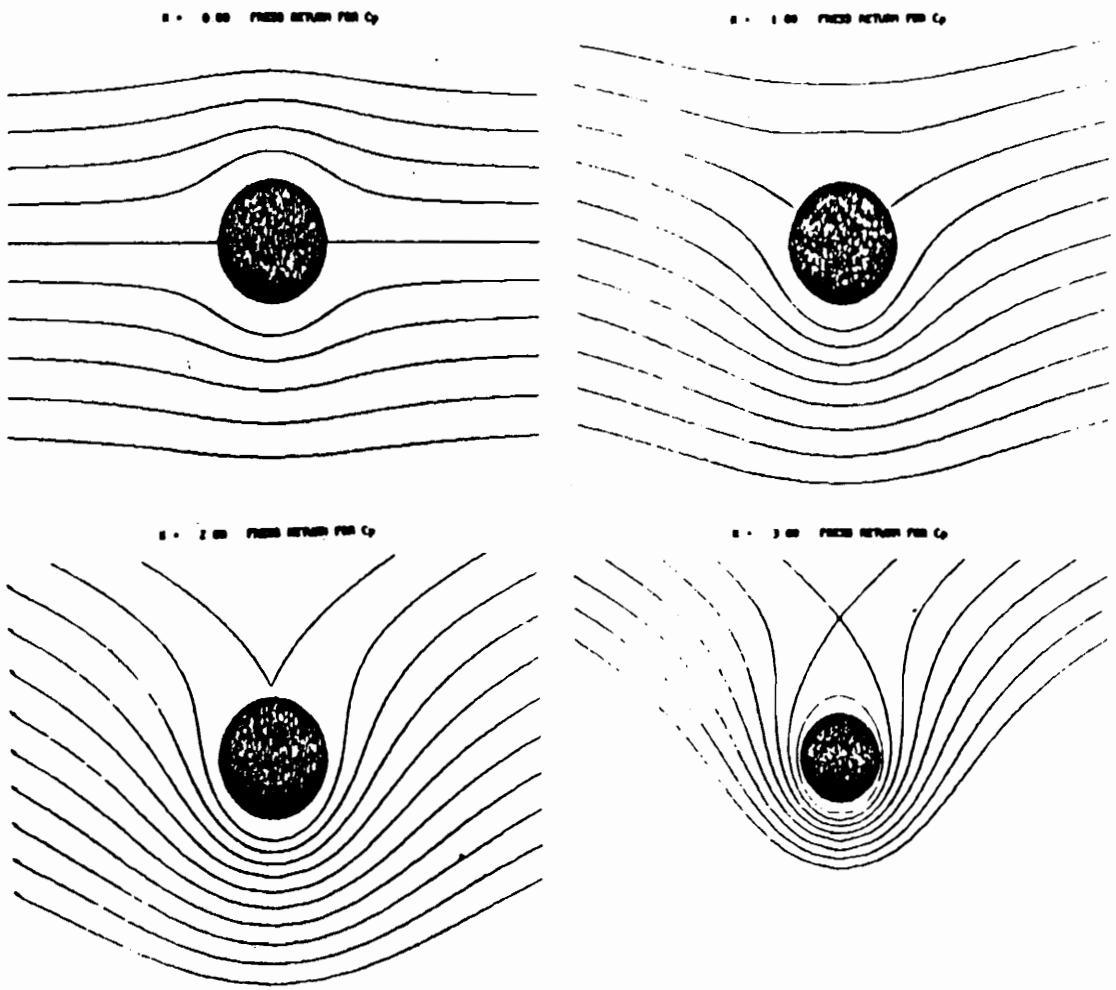


FIGURE 6.10: Sample Output for Program CYLINDER

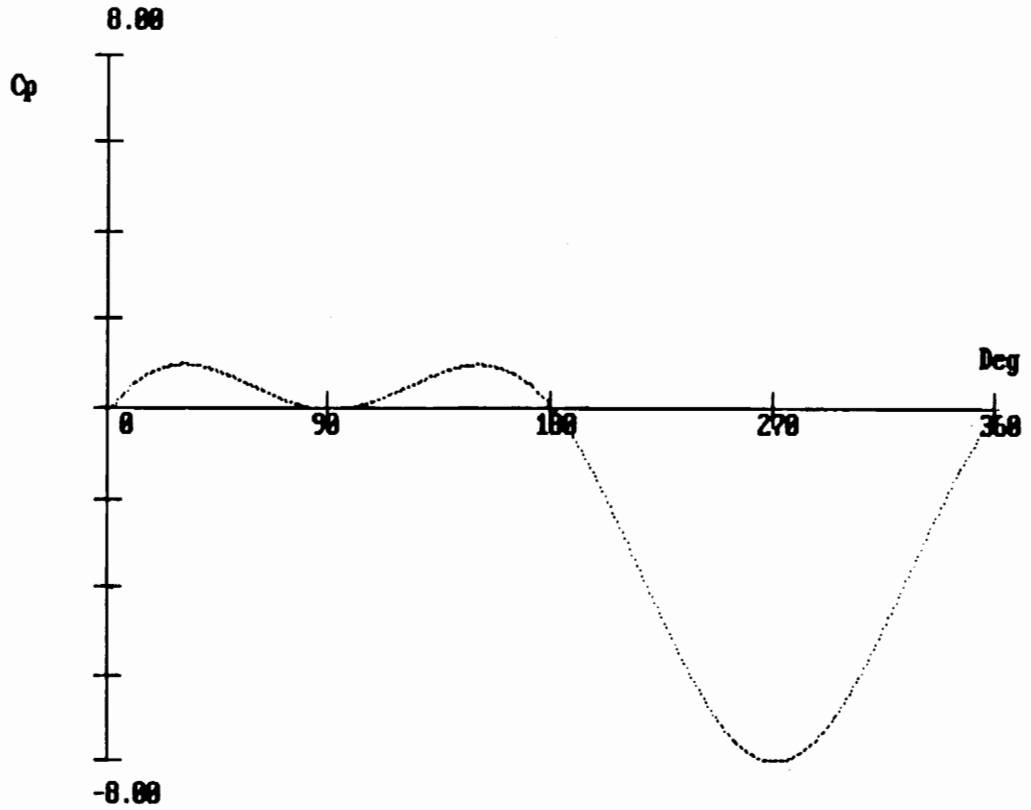


FIGURE 6.11: Pressure Coefficient Plot Generated by Program CYLINDER (K = 1.0)

6.8 THINFOIL

Thin airfoil theory is usually the first category of airfoil theory approached in fluid mechanics. This theory considers an airfoil of essentially zero thickness. An airfoil is best described by its mean camber line which lies midway between the upper and lower surfaces of the airfoil profile. In the case of a thin airfoil, the mean camber line is the airfoil. A streamline following the desired thin airfoil shape is constructed with a vortex sheet along the mean camber line. The strengths of the individual vortices must be chosen so that no flow crosses the mean camber line.

The simplest thin airfoil shape is a flat plate. This configuration is considered in White [4]. This program considers a mean camber line that follows the cubic equation

$$z = 4\,h\,[\,x - (k+1)\,x^2 + k\,x^3\,] \qquad (6.24)$$

where the vertical height, z, and the horizontal distance, x, have both been nondimensionalized by the chord length, c. Therefore, x = 0 at the airfoil leading edge and x = 1 at the trailing edge. At both the leading and trailing edges, z = 0. Values for both h and k are specified by the user. The magnitude of the airfoil camber is described by h. It is the maximum camber height divided by the chord length. Change in curvature of the airfoil shape is governed by the parameter k. It determines where the maximum height, h, is located. The case of k = 0 describes the case of constant curvature, i.e., a circular arc. Positive k values shift the maximum

height nearer the trailing edge, while negative values of k shift the maximum height nearer the leading edge.

When defining the airfoil shape, the user may elect to include a trailing edge flap. In this case, a portion of the trailing edge is replaced with a straight line at a angle measured relative to the absolute X, Z coordinate system. Negative angle values are measured counterclockwise.

Once these values have been specified by the user, the program calculates the following parameters:

$$C_L = \frac{F\ L}{1/2\,\rho U^2} = 2\pi\alpha + (4-3k)\pi h \tag{6.25}$$

$$C_{mac} = \frac{M}{1/2\,\rho U^2 L^2} = \left(\frac{7}{8}k - 1\right)\pi h \tag{6.26}$$

$$\alpha_{LO} = -(2-3/2\,k)h \tag{6.27}$$

$$X_{cp} = \frac{1}{4} - \frac{C_{mac}}{L} \tag{6.28}$$

C_L is the lift coefficient, which depends on the airfoil shape and the angle of attack, α. C_{mac} is the moment coefficient about the aerodynamic center (quarter chord position). At some angle of attack, the lift force on the airfoil would be zero. This is the zero lift angle, α_{LO}. The lift force and moment may be considered to act at a single point defined as the center of pressure, X_{cp}, which is measured from the leading edge. These parameters are

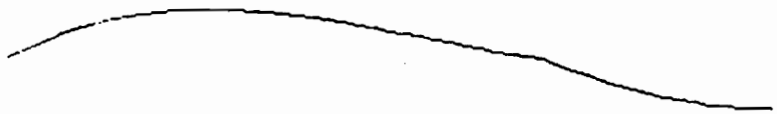
displayed on the output screen along with the plotted airfoil shape and the mean camber line equation, as shown in Figure 6.12

In Figure 6.12, the lift coefficient is left in terms of the angle of attack, α, written as A in the program. The program then waits for the user to press the RETURN key before proceeding with the plot of the pressure coefficient. After the RETURN key has been pressed, the user may enter a value for the angle of attack, which is the last input before the pressure coefficient diagram is constructed. The pressure difference between the pressure (lower) and suction (upper) sides of the airfoil is plotted as a function of x. Typical results are shown in Figure 6.13, which includes the consideration of a trailing edge flap. Note that the curve in Figure 6.13 diverges toward the leading edge. A singularity in the mathematical calculation of the pressure coefficient exists there. For real flows past a thin airfoil, the value of Cp returns sharply to zero. Cp is found by integrating the lift coefficient along the X direction. In addition to the plot, the angle of attack, lift coefficient, and center of pressure values are written to the screen.

* All lengths are nondimensionalized
  by the airfoil chord length.

Z = 0.40 (X - 2.50 X^2 1.50 X^3)

Flap length = .30 Angle = 10.0 deg



CL = 6.283A + 0.568

Zero-Lift Angle = -5.18 deg

Cmac = -0.01

PRESS RETURN FOR Cp PLOT.

FIGURE 6.12: Sample Thin Airfoil Shape Generated by Program THINFOIL (k = 1.5, h = 0.1, Trailing Edge Angle = 10 degrees)

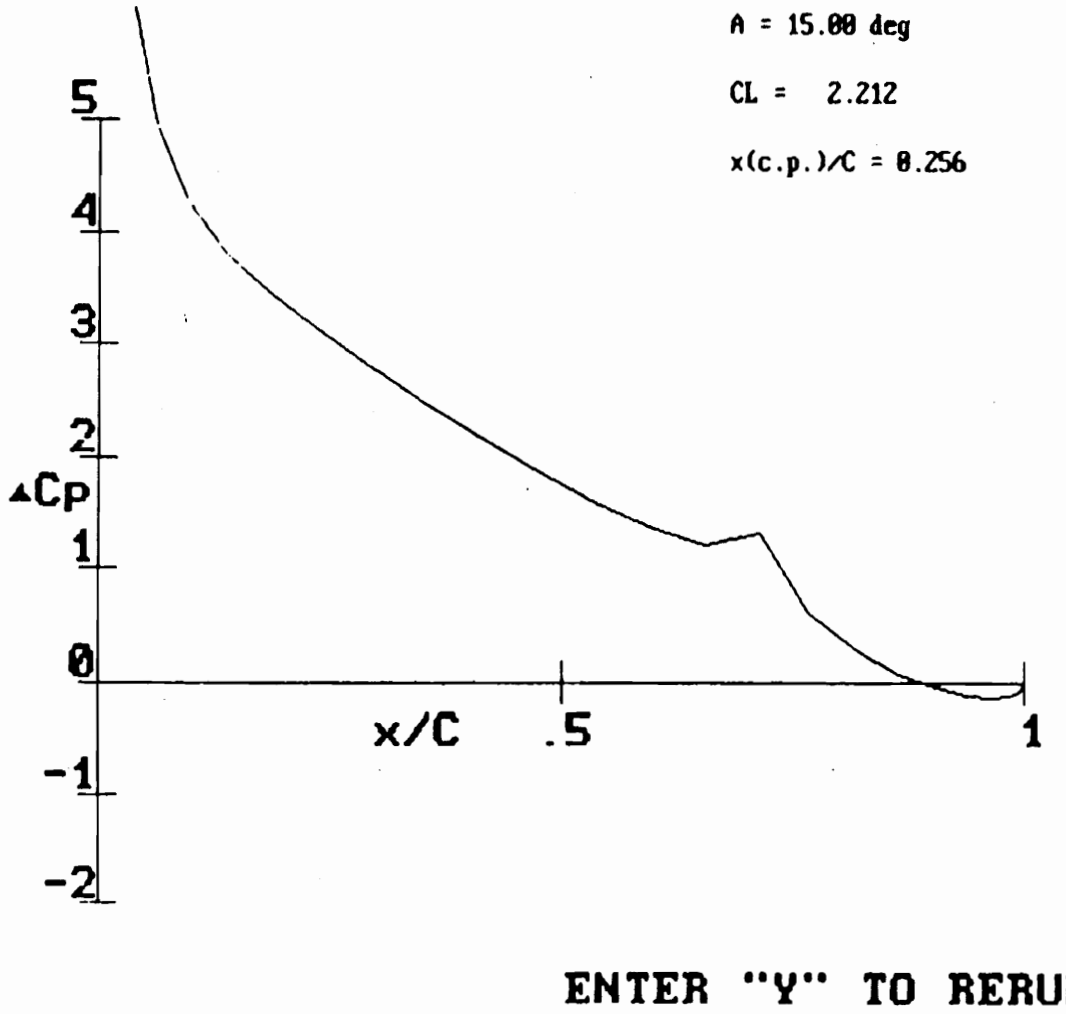


FIGURE 6.13: Pressure Coefficient Plot Generated by Program THINFOIL (k = 1.5, h = 0.1, Trailing Edge Angle = 10 degrees Angle of Attack = 15 degrees)

6.9 KTFOIL

In the program THINFOIL, an airfoil of negligible thickness was considered. In this program, airfoils of substantial thickness are created using a conformal transformation. This transformation is used to map the flow past a cylinder into the flow past an airfoil shape. Using complex variables, the circular cylinder is placed in the $\zeta = \xi + i\eta$ plane, and the airfoil in the z = x + iy plane. von Karman and Treffetz generalized the work of Joukowski in 1918 with the following transformation equation.

$$\frac{z - nc}{z + nc} = \left(\frac{\zeta - c}{\zeta + c} \right)^n \tag{6.29}$$

The center of the cylinder to be transformed lies in the second quadrant of the $\zeta = \xi + i\eta$ plane at the coordinates (ξ_0, η_0), written in the code as ($-X_0$, Y_0) in the program, where X_0 and Y_0 have been nondimensionalized by the length, c. c is the ξ coordinate value where the circle crosses the positive ξ axis. This transforms to the point x = nc on the X axis, and corresponds to the airfoil trailing edge. The transformation Equation (6.29) is discussed in Prandtl and Tietjens [8] and in Milne-Thompson [7].

If the power, n, is equal to two, the Karman-Treffetz transformation becomes the Joukowski transform. Because the trailing edge angle is defined as (2 - n) π , the trailing edge angle is zero for the Joukowski transform.

When running the program, the user supplies the value of the trailing edge angle. For the Joukowski transformation, the user should input zero

for this parameter. With the trailing edge angle, the program calculates the value of the power, n. The user then inputs the coordinates of the transformation circle center, (X_0, Y_0). Three options are then proposed to the user. The program may:

(1) - generate a plot of the airfoil shape similar to Figure 6.14, and provide a formula for computing the lift coefficients in terms of the angle of attack

(2) - generate a plot of the airfoil and the flow field for an inputted angle of attack, similar to Figure 6.15

(3) - generate a plot of the pressure coefficient along the pressure and suction sides of the airfoil, similar to Figure 6.16.

Construction of the airfoil shape in options (1) and (2) is done by selecting points on the circle in the ζ plane at increments of the angle called B in the program. Equation (6.29) is used to solve for the airfoil coordinates. In option (1) the isolated airfoil is drawn at a zero angle of attack. Option (2) presents the airfoil in reduced size at the user supplied angle of attack. Streamlines for the flow field are constructed by mapping the streamlines around a circular cylinder from the ζ plane to the Z plane. A Newton iteration is used to determine the minimum distance between the airfoil and a streamline.

The lift force is given by the Kutta-Joukowski theorem, $F = -\rho U \Gamma$, where ρ is the density, U is the free stream velocity, and Γ is the circulation. The value of Γ is governed by the requirement that a stagnation point occur at the airfoil trailing edge for a given angle of attack. The lift coefficient is then

CL = 7.183 x SIN(A) + 0.513 x COS(A)
A = Angle of attack

Enter "Y" to Rerun.

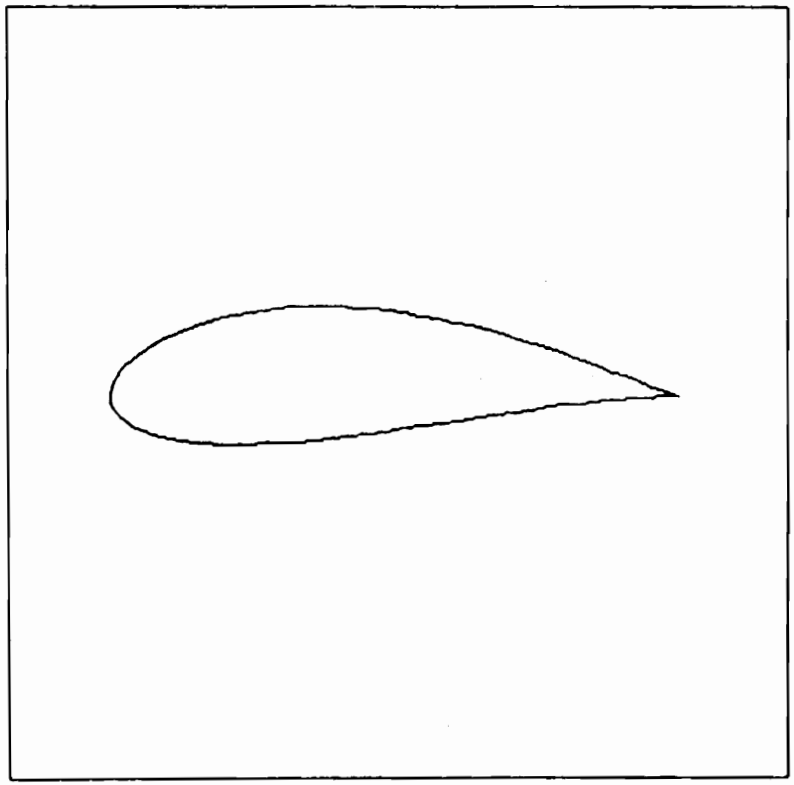


FIGURE 6.14: Airfoil Shape Generated By Program KTFOIL (Xo = 0.12, Yo = 0.08)

Angle of attack = 10.00
Lift Coefficient = 1.75

Enter "Y" to Rerun

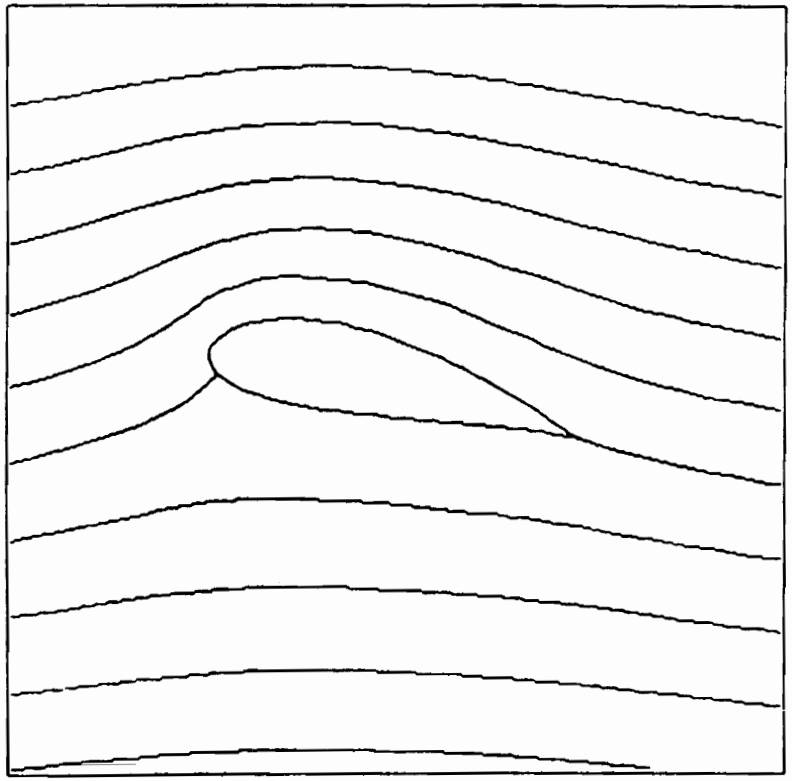


FIGURE 6.15: Flow Field Around an Airfoil Generated by Program KTFOIL (Xo = 0.12, Yo = 0.08)

Trailing edge angle = 12.00

Angle of attack = 0.17

Lift coefficient = 1.753

XO = 0.120
YO = 0.000

Enter "Y" to Rerun.

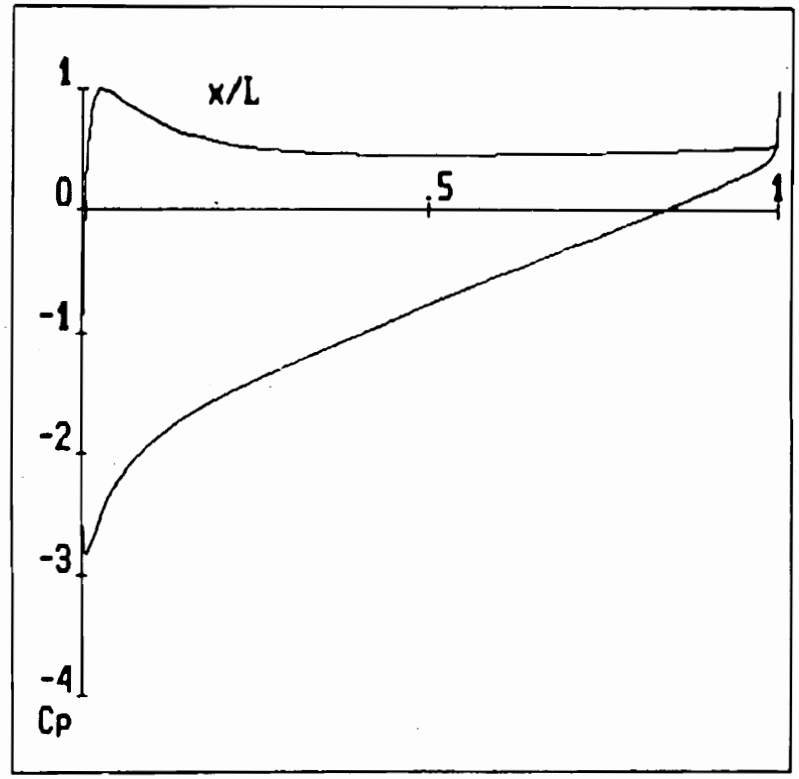


FIGURE 6.16: Pressure Coefficient Plot Generated by Program KTFOIL

$$C_L = C_1 \sin A + C_2 \sin A$$

where $C_1 = 8\pi\ (1+X_0)/L$

and $C_2 = 8\pi\ Y_0/L$ (6.30)

where L is the chord length nondimensionalized by the length, c. In option (1), the formula for the lift coefficient is presented, while options (2) and (3) display a calculated value for C_L.

CHAPTER 7: VISCOUS FLOW

In Chapter 5, potential flow dealt with two dimensional, steady, inviscid flows. Most real fluids cannot be modeled as inviscid. Therefore, some viscous effects are considered in this chapter. Physically, this means that the viscous terms cannot be omitted from the momentum equations, resulting in more complex mathematical forms. The momentum equations expressed in each of the three Cartesian directions are then referred to as the Navier-Stokes equations. For incompressible flow with no body forces and constant viscosity, the equations are as follows.

$$\frac{Du}{Dt} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + 2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

$$\frac{Dv}{Dt} = -\frac{1}{\rho}\frac{\partial P}{\partial y} + 2\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}\right)$$

$$\frac{Dw}{Dt} = -\frac{1}{\rho}\frac{\partial P}{\partial z} + 2\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}\right) \qquad (7.1)$$

Viscous flow is most often associated with boundary layer theory. A boundary layer is a thin region next to the surface of a physical boundary where a fluid's viscous effects are concentrated. Additionally, the possible separation of the boundary layer must be considered.

Flow past a flat plate is the most elementary boundary layer problem proposed to fluid mechanics students. The Program BOUNDARY examines the definitions of the different thicknesses associated with a boundary layer.

Within the boundary layer, the velocity profile is of key interest. Programs BLASIUS and WEDGE numerically solve the boundary layer equations and obtain the velocity profiles for flow parallel to a flat plate for constant free stream pressure and variable free stream pressure respectively.

THWAITES employs an integral method to predict boundary layer growth along a flat plate with an arbitrary variation of free stream pressure up to the point of separation. Figure 7.1 shows the coordinate systems used in these four programs.

Program ANNULUS considers the pressure driven viscous flow between concentric cylinders, with the option of including axial motion of the outer cylinder. The program generates velocity profiles for these flows.

All the programs in this chapter concern laminar flow except program MOODY. Fully developed laminar and turbulent flows inside a circular pipe are usually treated in introductory fluid mechanics. Program MOODY will construct the Moody diagram, or give specific values of Reynolds number or the friction factor.

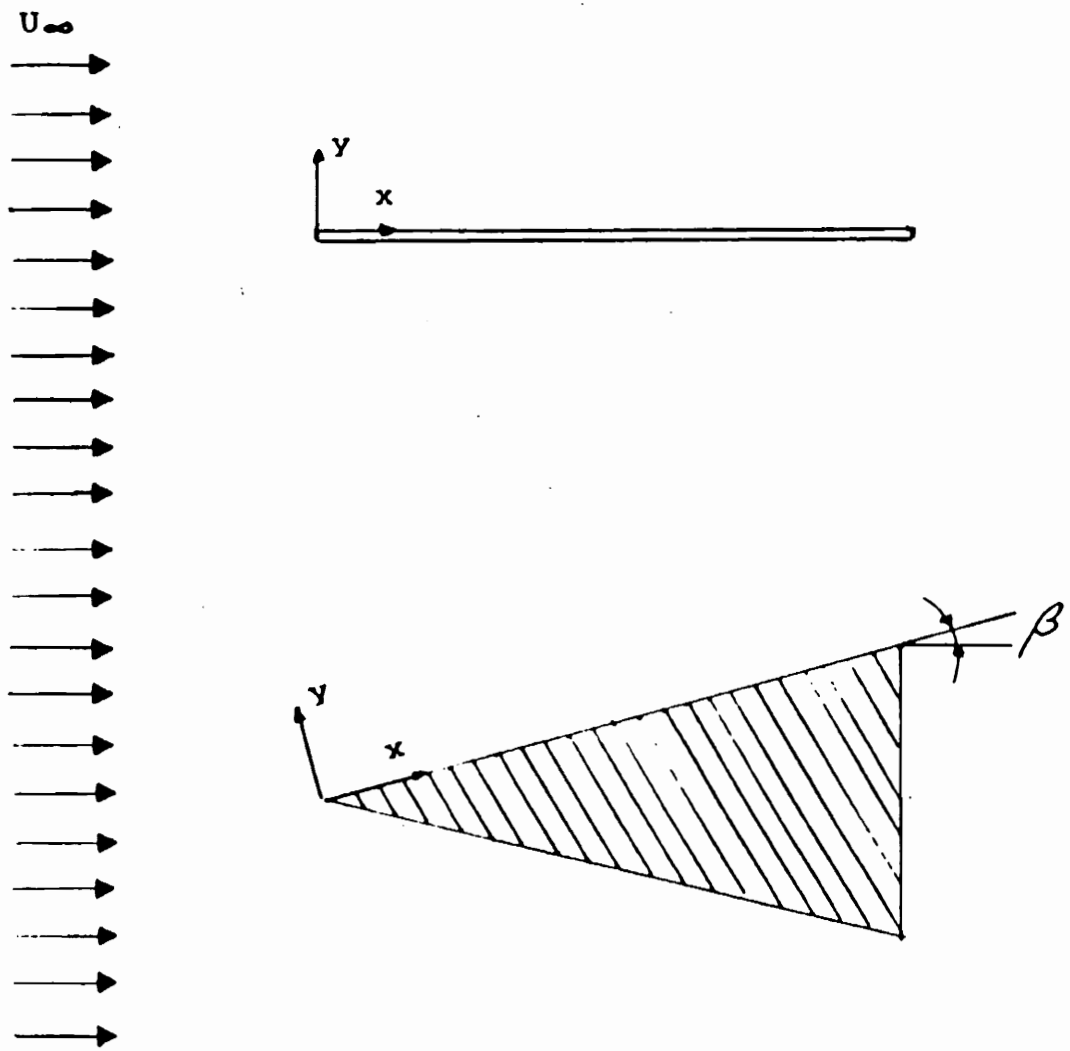


FIGURE 7.1: Coordinate Systems Used for Programs BOUNDARY, BLASIUS, WEDGE, and THWAITES

7.1 BOUNDARY

Boundary layer theory was first proposed by Prandtl in 1904. His physical concept was one of a layer near the boundary of a solid body where, for higher Reynolds numbers, the viscous effects of a fluid were dominant, while outside this layer the flow could be approximated as inviscid. Program BOUNDARY explores the viscous case of parallel laminar flow along a flat plate of negligible thickness.

This program considers a uniform flow approaching a flat plate with a constant velocity U_∞. Traversing the flow normal to the plate, fluid particles near the plate are more affected by viscosity and are subsequently slowed. The velocity of those particles closer to the plate is decreased more than those at a greater distance, and at the plate surface, the flow velocity is zero. At some distance, δ, away from the plate surface, the fluid velocity is assumed to be about 99% of the free stream velocity U_∞. This distance, δ, is referred to as the boundary layer thickness, and the region between the plate and δ is called the boundary layer. Within this layer the viscous effects are dominant. The boundary layer thickness is not constant along the plate length. In the usual case, it grows as the viscous effects propagate from the plate surface into the boundary layer. Thus, δ is a function of the distance, X, along the plate length.

Within the boundary layer, the velocity is not uniform, but takes on what is often modeled to be a parabolic curve. The details of the profile are a function of both directions, X and Y, as seen in Figure 7.2. However, the profiles are assumed to possess the same shape regardless of their X

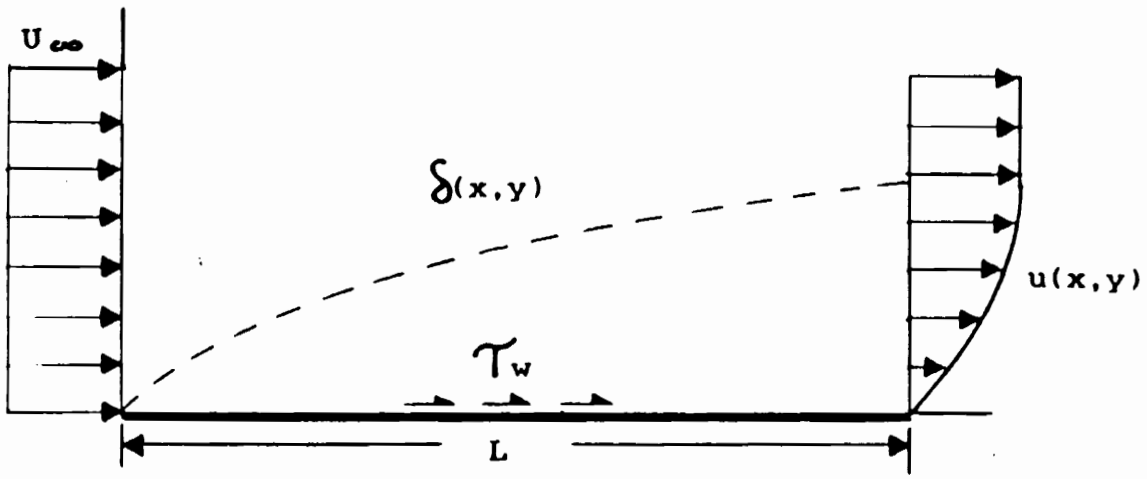


FIGURE 7.2: Boundary Layer Thickness and Velocity Profile in the Viscous Boundary Layer

location. Thus, an equation modeling the velocity profiles may be nondimensionalized in the Y direction by the distance, δ. One possible model is a quadratic expression

$$\frac{u}{U_\infty} = a + b\left(\frac{y}{\delta}\right) + c\left(\frac{y}{\delta}\right)^2 \tag{7.2}$$

where the following boundary conditions apply:

1. The velocity at the wall is zero (no slip): u(0) = 0

2. The velocity at δ is the free stream velocity U_∞: u(δ) = U

3. The profile is symmetric about a horizontal line passing through the point where y = δ. A smooth transition occurs between the boundary layer velocity profile and the free stream:

$$\left.\frac{du}{dy}\right)_{y=\delta} = 0$$

Using these conditions the following velocity profile is obtained.

$$u(x,y) = U_\infty\left(\frac{2y}{\delta} - \frac{y^2}{\delta^2}\right) \tag{7.3}$$

The boundary layer thickness has already been defined. Two other important quantities are the displacement thickness, δ^*, and the momentum thickness, θ. Streamlines outside the boundary layer are deflected by an amount, δ^*, such that the continuity equation is satisfied.

Mathematical definitions of the displacement and momentum thicknesses follow.

$$\theta = \int_{o}^{\delta} \frac{u}{U_\infty}\left(1 - \frac{u}{U_\infty}\right)dy \qquad \delta^* = \int_{0}^{\delta}\left(1 - \frac{u}{U_\infty}\right)dy \tag{7.4}$$

For the assumed laminar flow velocity profile model, the boundary layer thickness can be expressed as a function of X and Reynolds number based on X, while δ^* and θ may be expressed as fractions of δ.

$$\delta = x\ 5.5\ Re_x^{-\frac{1}{2}} \qquad \delta^* = \frac{\delta}{3} \qquad \theta = \frac{2\delta}{15} \tag{7.5}$$

For a flat plate in a constant free stream velocity flow, at a Reynolds number of approximately $5x10^5$, the flow undergoes transition to turbulence and the velocity profiles change significantly (White [4]). For this turbulent boundary layer case, White [4] provides the approximate formulas

$$\delta = x\ 0.16\ Re_x^{-\frac{1}{7}} \qquad \delta^* = \frac{\delta}{8} \qquad \theta = \frac{7\delta}{72} \tag{7.6}$$

For program BOUNDARY, the user must input both the constant free stream velocity and the length of the plate. Variable free stream velocities and pressure gradients are considered in later programs. Default values for the transition Reynolds number (500,000) and kinematic viscosity (water 25 C) are provided by the program. The user may accept either of these values or enter new ones. The program calculates the Reynolds number, boundary

layer, displacement, and momentum thicknesses along the plate in the X direction. If the Reynolds number reaches the turbulent transition value, Equations (7.6) are used instead of Equations (7.5). The thickness of the boundary layer at any location after the transition point is calculated by subtracting the turbulent boundary layer thickness from the laminar boundary layer thickness at the transition point and adding the result to the turbulent boundary layer thickness at the location x.

The program will produce a plot of the thicknesses along the plate length with the turbulent transition point being noted (if it exists) as shown in Figure 7.3. Although the boundary layer, displacement, and momentum thicknesses are in scale relative to one another, they have been magnified vertically, relative to the plate length, for clarity. A table of the thicknesses for different locations along the plate is provided after the plot.

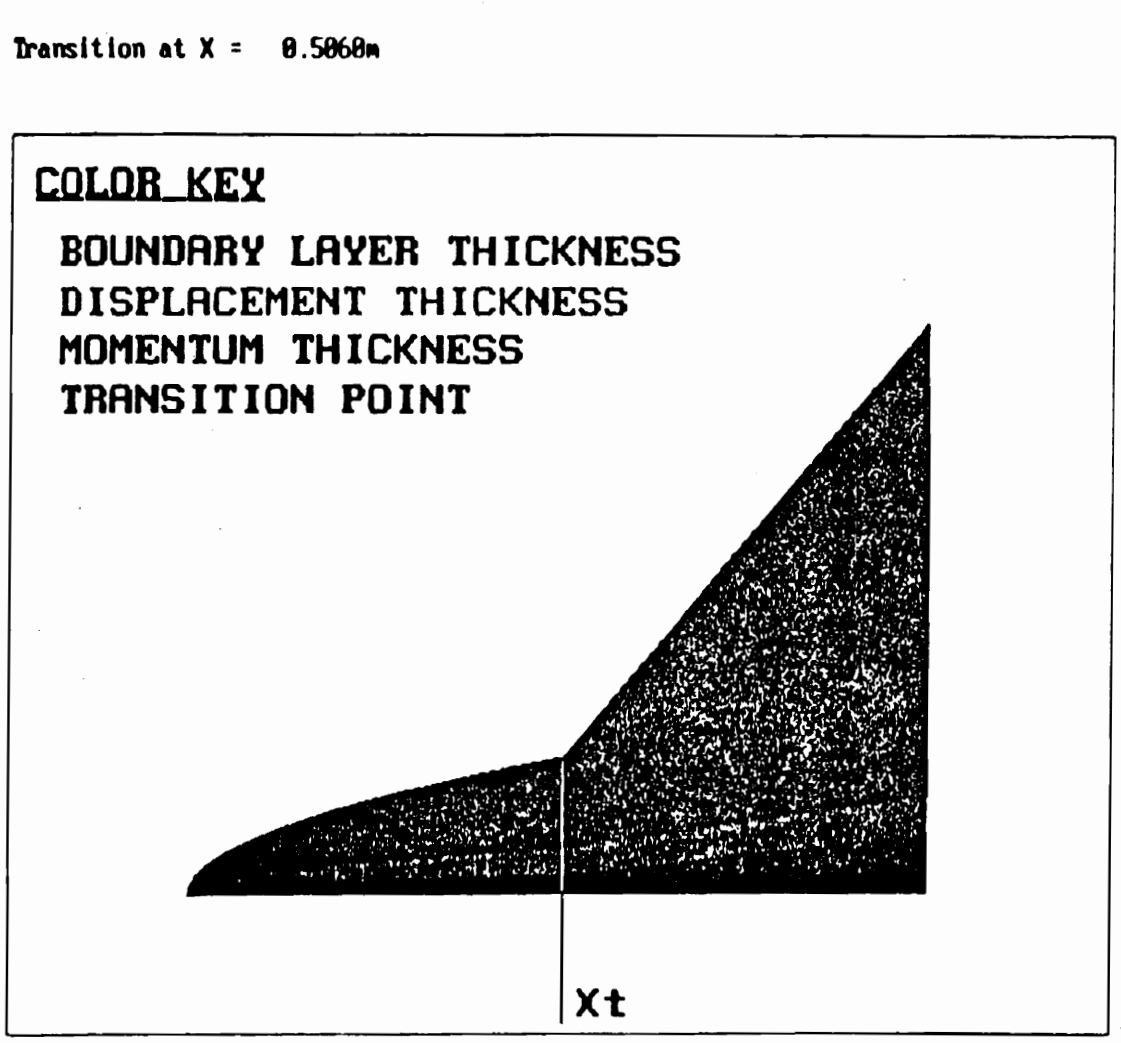


FIGURE 7.3: Sample Output for Program Boundary
(U = 1m/s, L = 1m, Default Transition Reynolds Number and Kinematic Viscosity)

7.2 BLASIUS

A quadratic equation was used as an approximate model for the velocity profile in a laminar flow in the program BOUNDARY. An exact solution to the Navier-Stokes equations for uniform laminar flow past a flat plate, the classic Blasius flow, is generated in the program BLASIUS. Blasius postulated that although the boundary layer grew along the plate length, the velocity profiles looked "similar". He considered a nondimensional form of the velocity profile that was the same at any X position. Blasius asumed a stream function

$$\psi = \sqrt{U_\infty \upsilon x}\, F \tag{7.7}$$

where U_∞ is the free stream velocity, υ is the kinematic viscosity, and F is a function only of the similarity variable, η. With the similarity variable

$$\eta = y\left(\frac{U_\infty}{\upsilon x}\right)^{\frac{1}{2}} \tag{7.8}$$

the dimensionless velocity profile, u/U_∞, becomes a function of only the dimensionless variable η.

$$\frac{u}{U_\infty} = \frac{dF}{d\eta} \tag{7.9}$$

If Equation (7.9) and its derivatives are substituted into the boundary layer equation, a single third-order, nonlinear ordinary differential equation for F results, with

$$F''' + 1/2\ F\ F'' = 0 \tag{7.10}$$

where the following boundary conditions then apply.

$$F(\eta=0) = F'(\eta=0) = 0 \qquad F'(\eta \to \infty) = 1 \tag{7.11}$$

In order to obtain a solution to the above equation, the single third-order equation for F is replaced by three simultaneous first-order differential equations, with

$$F' = G \qquad G' = H \qquad H' = 1/2\, F\, H \tag{7.12}$$

where $G = F' = u/U_\infty$ is the nondimensional velocity, and $H = F''$ may be considered a nondimensional shear stress. A fourth-order Runge-Kutta integration scheme is used to solve the three simultaneous first order differential equations. The three boundary conditions in Equations (7.11) give $G(0) = F'(0) = 0$ and $F(0) = 0$ at the surface and $G(\eta \to \infty) = F'(\eta \to \infty) = 1$.

Because arrays in FORTRAN must begin with a subscript of 1, the above quantities are written as G(1) and F(1) in the program. A shooting method is employed to solve the equations iteratively, hence, a value H(1) must be assumed initially. The boundary layer is divided into 100 segments. With the assumed value of H(1), the program solves for G(101). A Newton iteration procedure is then employed to find successive H(1) values. The iteration terminates when G(101) = 1.0, within a tolerance of 5×10^{-5}.

The graphical output includes curves representing both the nondimensional velocity, $F' = u/U_\infty$, and the nondimensional shear stress, F". The X axis varies from nondimensional values of $u/U_\infty = 0.0$ to 1.6, while the Y axis varies from 0.0 to 6.0 and is labeled, Y ($Y = \eta$). The boundary layer thickness y/δ, at $u/U_\infty = 0.99$, is approximately equal to five.

After the initial value of H is entered, the user may elect to have the iterative solutions plotted. In this case, the failed solutions from the Runge-Kutta integration are displayed until the program reaches the final iteration, which is within the arbitrary tolerance for G(101) = 1.0.

Figure 7.4 displays an iterative solution for 0.5 as the initial value of H. This display prompts the user for a table of F, G, and H values versus the nondimensional distance η.

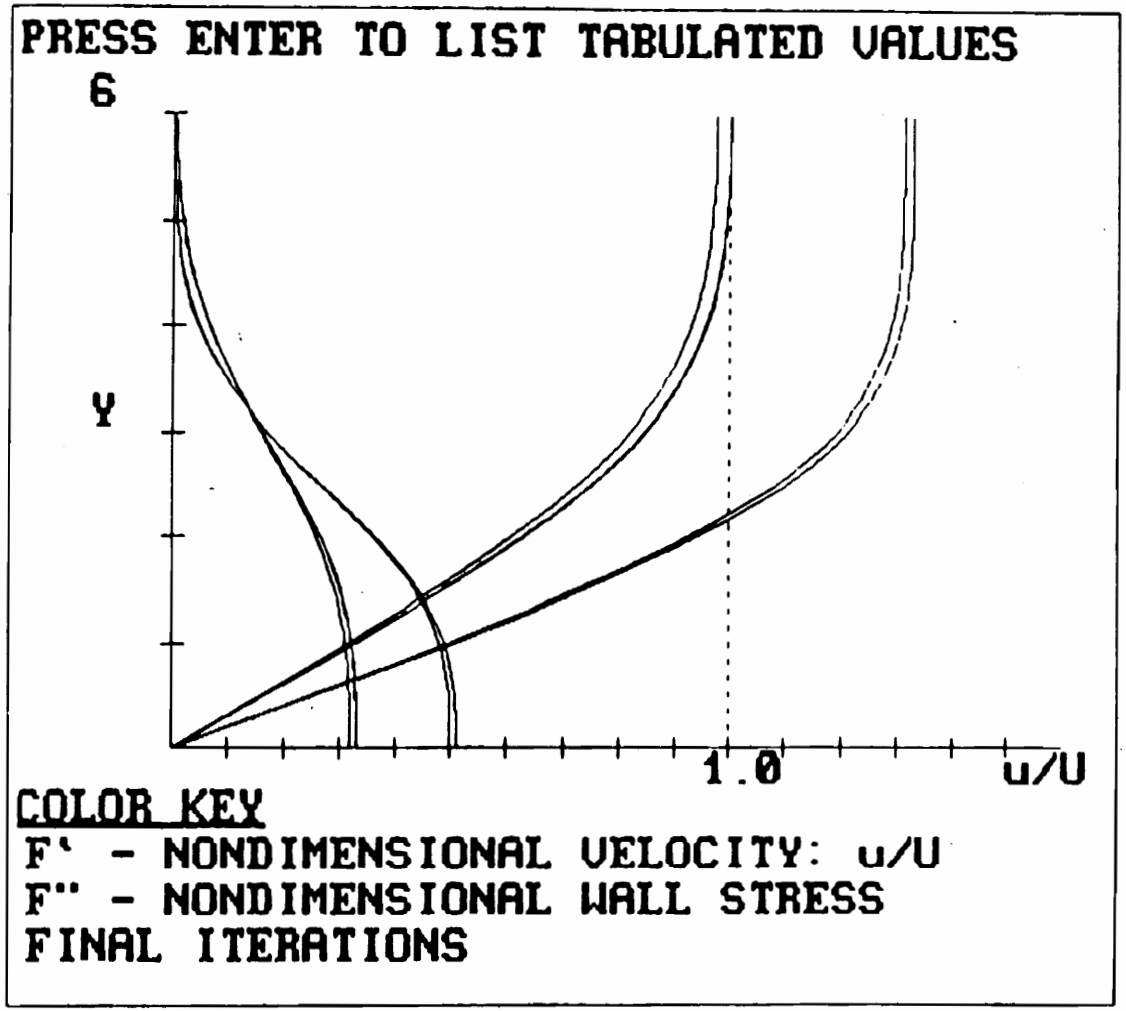


FIGURE 7.4: Iterative Blasius Solution for Parallel Flow Past a Flat Plate
(H(1) = 0.5)

7.3 WEDGE

The two previous programs considered a constant free stream flow (no pressure gradient) past a flat plate. WEDGE considers a laminar, variable free stream velocity (nonzero pressure gradient) of the form $U(x) = K\, x^m$. This is the classic Falkner-Skan flow. Positive values of the constant, m, correspond to flow past a wedge while negative values represent flow past an expansion corner. The wedge half angle is referred to as π ß/2, and the exponent, m = ß/(2- ß) (see White [6] and Figure 7.5). The case of ß = 0 is the flat plate.

Again, as in the Blasius solution for a flat plate, a similarity analysis is used. Here the similarity variable

$$\eta = \frac{y}{\delta} = y\left(\frac{m+1}{2}\frac{u(x)}{\upsilon x}\right)^{\frac{1}{2}} \tag{7.13}$$

and an assumed unknown stream function result in the following third order differential equation.

$$F''' + F\,F'' + ß\,[\,1 - (F')^2\,] = 0 \tag{7.14}$$

This is the common form of the Falkner-Skan equation. As in the program BLASIUS, the third-order differential equation is replaced by three, simultaneous, first-order differential equations.

$$F' = G \qquad G' = H \qquad H' = -F\,H - ß\,(1 - G^2) \tag{7.15}$$

The same boundary conditions still apply: F(0)=0, G(0)=0, and

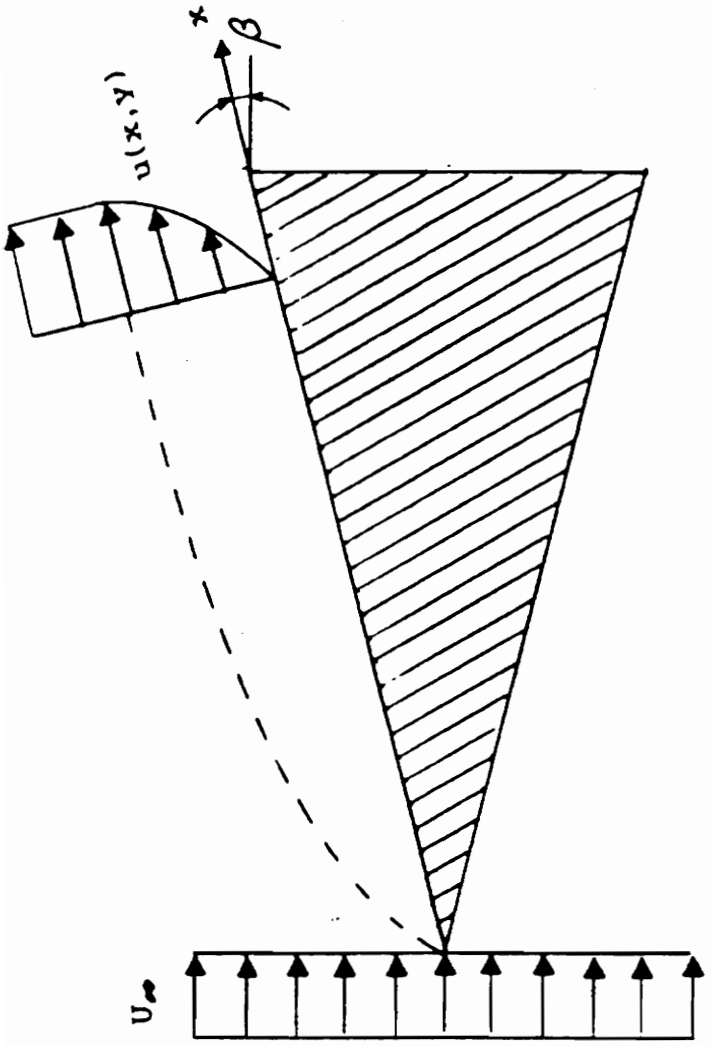


FIGURE 7.5: Flow Past a Wedge With Half Angle, β

G($\eta \rightarrow \infty$)=1. These first two boundary conditions are written as F(1) and G(1) in the program since FORTRAN does not allow a zero subscript for arrays. As in BLASIUS, the boundary layer is divided into 100 segments. Thus, the third boundary condition is G(101) = 1.

A fourth-order Runge-Kutta integration routine is again employed to solve the three, simultaneous differential equations. An initial value for H(1) is calculated by the program in terms of a user specified value for ß. A Newton iteration then determines subsequent values of H(1) for each subsequent solution until G(101)=1 within an arbitrary tolerance.

When running the program, the user first decides if the separate iterative solutions are to be plotted. If not, the user may supply up to ten different values for ß, which must be within the range of -.198838 to 1. A ß value less than the lower limit will result in the nondimensional stress being less than zero, indicating a separated flow. A value of 1 corresponds to a wedge half angle of ninety degrees (flow normal to a plane surface). Again, plots of F' and F", representing u/U(x) and the nondimensional stress respectively, are drawn on coordinate axes of X = u/U(x) = 0.0 to 1.6 and Y = η = 0.0 to 6.0. It turns out, unlike the Blasius solution, that a nondimensional distance of Y = η = 3.5 is effectively the edge of the viscous region, or boundary layer, where the value of F' is essentially one.

If only one ß value is plotted, the user may also display tabulated values of F, G, and H as a function of the nondimensional distance from the wedge, Y = η.

Figure 7.6 shows the velocity profiles generated by the program for five different ß values.

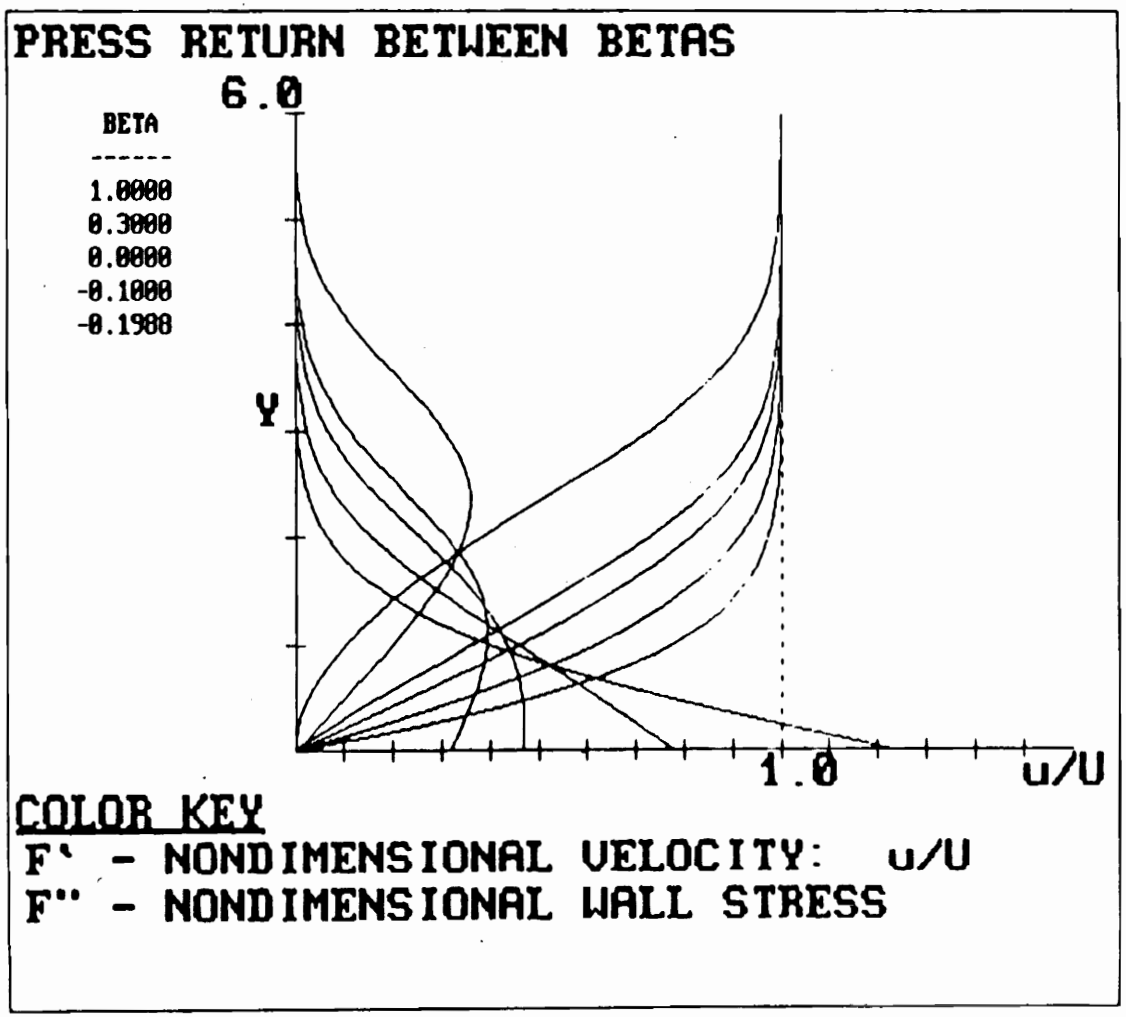


FIGURE 7.6: Falkner-Skan Solution for Different Values of the Half Angle, β

7.4 THWAITES

In programs BOUNDARY and BLASIUS, a uniform and constant velocity flow past a flat plate was examined. WEDGE explored the effect of varying the free stream velocity with $U(x)=K\,x^m$. THWAITES treats the case of an arbitrarily varying free stream velocity, U(x). In order to generalize the results, a nondimensional form of the free stream velocity is used. An expression for U(x)/Uo = f (x/L) is constructed where Uo is a reference velocity and L is a reference length, typically the flat plate length. A polynomial expression is used where the user determines the order and coefficients.

For the case of variable free stream velocity, the momentum integral equation is of the form

$$\frac{\tau_\omega}{\rho U^2} = \frac{d\theta}{dx} + (2+H)\frac{\theta}{U}\frac{dU}{dx} \tag{7.16}$$

where θ is the momentum thickness, H is the shape factor, and τ_ω is the shear stress at the plate surface. Thwaites (1949) found it convenient to express this relation in terms of a nondimensional parameter, λ, defined as

$$\lambda = \frac{\theta^2}{\upsilon}\frac{dU}{dx} \tag{7.17}$$

which correlated a large body of variable free stream velocity, laminar boundary layer flows. Using this new parameter with correlated data in the momentum integral equation, the momentum integral equation becomes

$$\frac{d}{dx}\left[\frac{\theta^2}{\upsilon}\right]=\frac{d}{dx}\left[\frac{\lambda}{U}\right]=\frac{F(\lambda)}{U} \tag{7.18}$$

The function $F(\lambda)$ is well modeled as linear, with $F = a - b\lambda$, hence

$$\frac{d}{dx}\left[\frac{\lambda}{U}\right]=a-b\lambda \tag{7.19}$$

This equation may be integrated, and with a = 0.45 and b = 6.0, the result is

$$\lambda=\frac{0.45}{U^6}\frac{dU}{dx}\int_o^x U^5dx \tag{7.20}$$

In the program, U is replaced by U/Uo, and x with x/L.

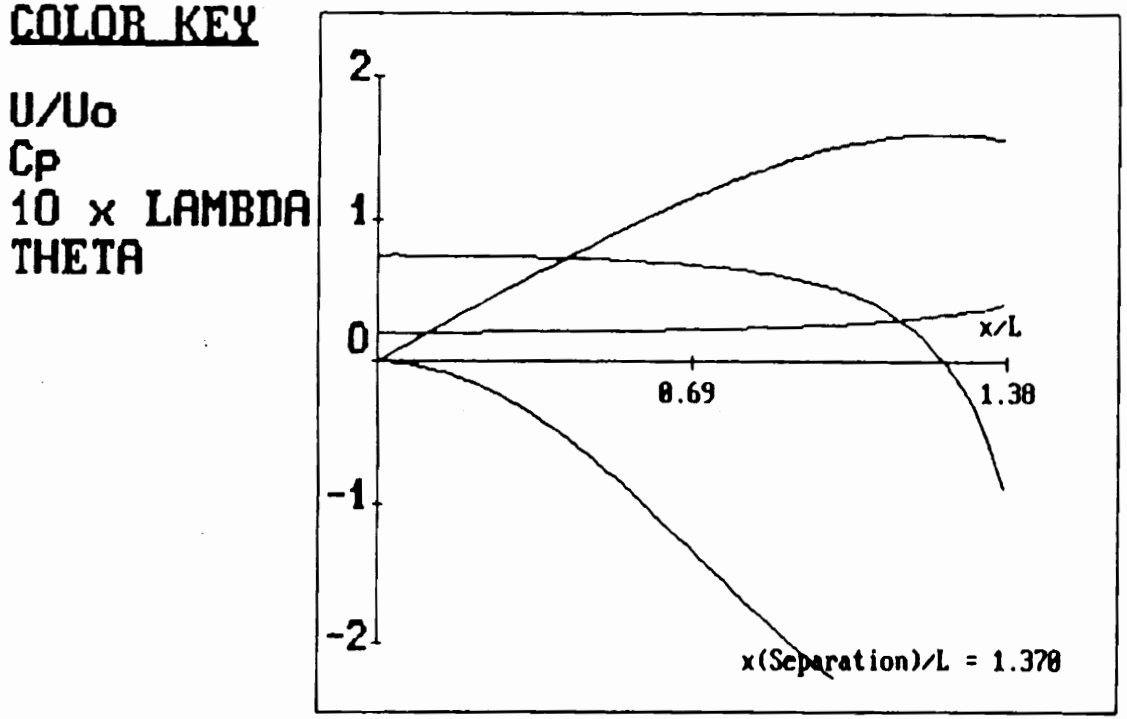
The program integrates Equation (7.20) using Simpson's 1/3 rule. (See appendix A.) A range of x/L = 0 to 2 is used with an integration step of d(x/L) = 0.01. For $\lambda > -0.09$, the wall shear stress is greater than zero. For λ below this value, separation of the boundary layer occurs. While integrating this expression, the program checks to see if separation occurs. If the value of λ is found to be less than -0.09, the x/L location of the separation point is found by interpolating between the last two integration steps.

Referring to Figure 7.7, the program calculates values for 10* λ, U/Uo, Cp and, THETA. λ is the Thwaites parameter previously defined, U/Uo is the user defined free stream velocity, THETA is defined as $\theta\sqrt{U_0/\upsilon L}$, and Cp is the pressure coefficient, defined as $(P-Po)/(1/2\,\rho\,Uo^2) = -(U/Uo)^2$. Tabulated values are also available for inspection after the plot has been constructed.

Figure 7.7 is the case of a cylinder placed normal to the flow where

$$U/Uo = 1.814\,(x/L) - 0.271\,(x/L)^3 - 0.0471\,(x/L)^5 \qquad (7.21)$$

This polynomial was proposed by Hiemenz (White [6]) to describe the experimentally measured free stream velocity for the laminar boundary layer flow past a circular cylinder. Note that separation occurs at x/L = 1.370, and with L taken as the cylinder radius, this is a radian measure. This method predicts separation at an angle of 78.5 degrees, which is close to the experimentally observed value of 80.5 reported in White [6].



Enter "Y" to Rerun, "T" for Table

FIGURE 7.7: Thwaites Prediction of Separation for Flow Past a Circular Cylinder

7.5 ANNULUS

Thus far, only viscous flows past a single surface have been examined. In program ANNULUS, the laminar flow between concentric cylinders, as shown in Figure 7.8, is considered. Two effects are used to drive the flow. These include a pressure gradient and axial motion of the outer cylinder.

When deriving the expression for the velocity distribution, the momentum equation reduces to

$$\frac{dP}{dx} = \frac{1}{r}\frac{d}{dr}\left(r\mu\frac{du}{dr}\right) \tag{7.22}$$

this is integrated twice to give

$$u = -\frac{1}{4}\frac{dP}{dx}\left[r_o^2 - r^2 - \frac{r^2{}_o - r_i^2}{\ln\frac{r_o}{r_i}}\ln\frac{r}{r_o}\right] + u_o\left[1 - \frac{\ln\frac{r}{r_o}}{\ln\frac{r_i}{r_o}}\right] \tag{7.23}$$

where u is the velocity in the X direction, u_0 is the axial velocity of the outer cylinder, r is the distance from the cylinder centers, μ is the coefficient of viscosity, r_o is the outer cylinder radius, and r_i is the inner cylinder radius. Axial motion of the outer cylinder displaces the velocity profiles in the flow direction.

Upon program execution, the user is first asked if axial motion of the outer cylinder is to be included.

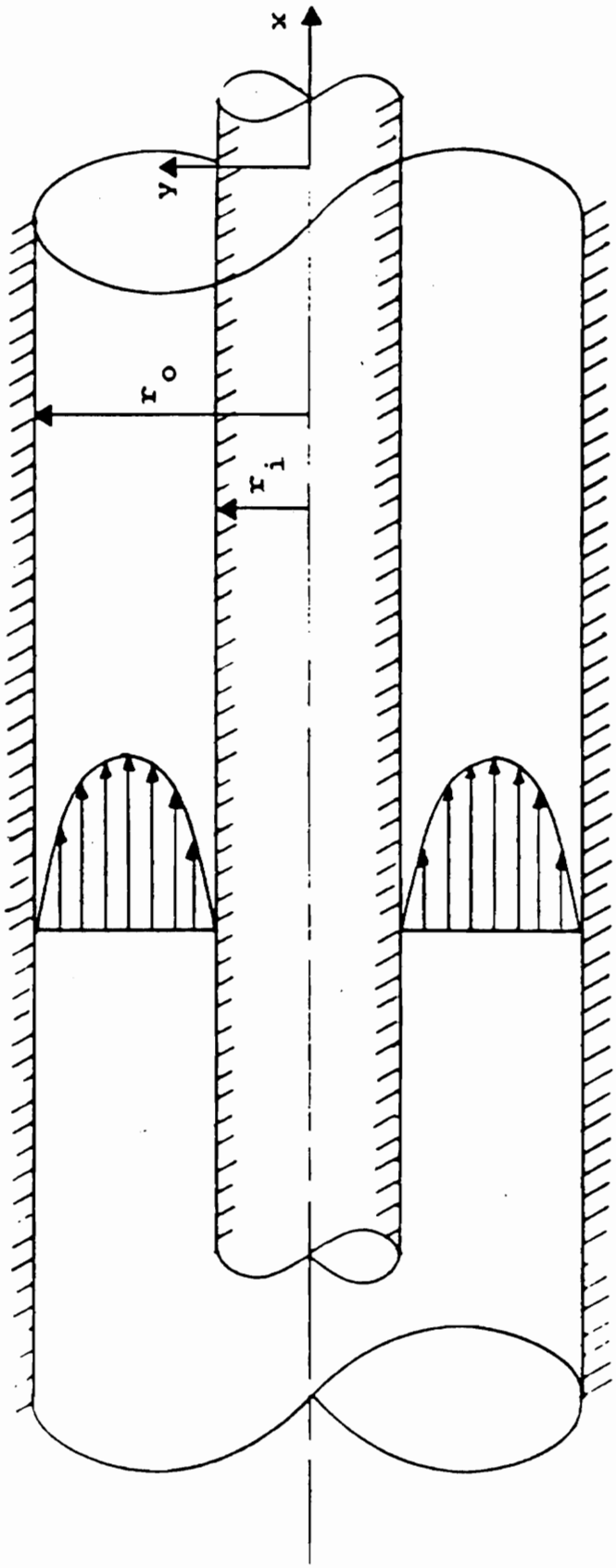


FIGURE 7.8: Annulus Schematic

If axial motion is to be considered, the user may choose to plot velocity profiles for various diameter ratios at a fixed pressure gradient, or for various pressure gradients at a fixed diameter ratio. The user specifies the fixed quantity as well as up to ten values of the varied parameter. In this case, Equation (7.23) is nondimensionalized by dividing by u_o.

If the outer cylinder is to remained fixed, the pressure gradient is held constant while different velocity profiles for various diameter ratios are plotted. The fixed pressure ratio is assigned a nondimensional value of 1, while the user may specify up to ten values of the diameter ratio. In this case, Equation (7.23) is nondimensionalized by dividing by the maximum velocity. u_{max} may be found by differentiating Equation (7.23) with respect to r and setting the result equal to zero, with

$$u_{max} = -(r_o - r_1)\frac{dP}{dx}8\mu \tag{7.24}$$

When more than one diameter ratio is considered, with no axial motion of the outer cylinder, the maximum velocity is computed for the minimum diameter ratio.

If only one diameter is entered for the case of no axial motion, then the effective and hydraulic diameters, defined below, are computed. Values for the effective diameter and the friction factor times the Reynolds number based on the hydraulic diameter are displayed. The hydraulic diameter is defined as

$$D_h = \frac{4x\text{Area}}{\text{Wetted Perimeter}} = \frac{4\pi\left(r_o^2 - r_i^2\right)}{2\pi\left(r_o + r_i\right)} = 2\left(r_o - r_i\right) \tag{7.25}$$

and a parameter ζ, which may be considered a correction factor for the hydraulic diameter, is computed as

$$\zeta = \frac{\left(r_o - r_i\right)^2\left(r_o^2 - r_i^2\right)}{r_o^4 - r_i^4 - \left(r_o^2 - r_i^2\right) / \ln\left(r_o / r_i\right)} \tag{7.26}$$

and f Re_{Dh} is calculated as 64 ζ. The effective diameter is found by dividing the hydraulic diameter by ζ.

Figure 7.9 shows screen output for the case of a single pressure gradient, with five diameter ratios, and axial motion of the outer cylinder.

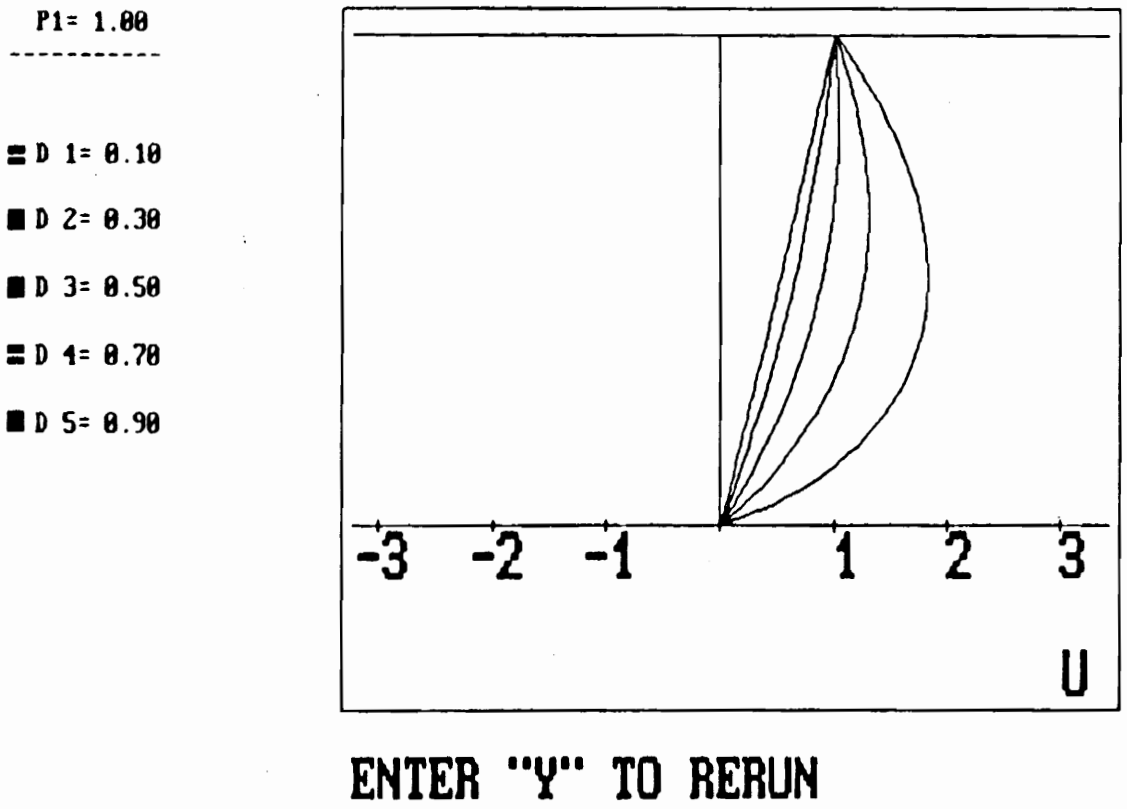


FIGURE 7.9: Sample Output for Program ANNULUS

7.6 MOODY

A relatively simple extension of the Bernoulli equation is to consider head loss due to friction along pipe walls in fully developed flows. For a circular pipe at a constant elevation with no minor losses (i.e. valves, nonuniform cross section, etc.) affecting the flow, the head loss is defined as

$$h = \frac{fL}{D}\frac{V^2}{2g} \tag{7.27}$$

where f is the friction factor, which is a nondimensional form of the wall shear stress, $f = 8\tau_\omega / (\rho\ V^2)$, L is the length of the pipe in the flow direction, D is the pipe diameter, and V is the average flow velocity.

For laminar flow, the relationship between the friction factor and the Reynolds number is given by f = 64/Re. For turbulent flow, it can be determined by dimensional analysis that f is a function of both Reynolds number based on pipe diameter, and the pipe surface roughness. Using extensive experimental data for fully developed turbulent flows in both smooth and rough pipes, Colebrook [9] developed the approximate implicit formula

$$f^{-\frac{1}{2}} = -2.0\log\left(\frac{\varepsilon / D}{3.7} + \frac{2.5}{\mathrm{Re}_D f^{\frac{1}{2}}}\right) \tag{7.28}$$

where D is the pipe diameter and ε is a measure of the surface roughness. This relation is accurate to 15% for design calculations and is the accepted formula for turbulent flow.

In 1944 Moody [10] presented a graphical form of both the laminar friction factor relation as well as Equation (7.28) in what is now called the Moody diagram. This plot may be reproduced by the program, as seen in Figure 7.10.

The program offers the option of computing the Reynolds number, or the friction factor, given the appropriate inputs. If the friction factor is to be determined, then a Reynolds number must be supplied. For Reynolds number values greater than 2300, a roughness ratio, ε/D, must be entered, and a turbulent flow solution uses an iterative solver. A direct calculation is performed for the laminar case.

If the Reynolds number is to be determined, then the friction factor must be supplied. The user is asked whether a laminar or turbulent solution is to be assumed. For a turbulent solution, the user must provide a roughness ratio, and an iterative procedure is performed. Again, a direct calculation is employed for the laminar case.

The Newton Raphson iterative method used in the program is reviewed in Appendix A.

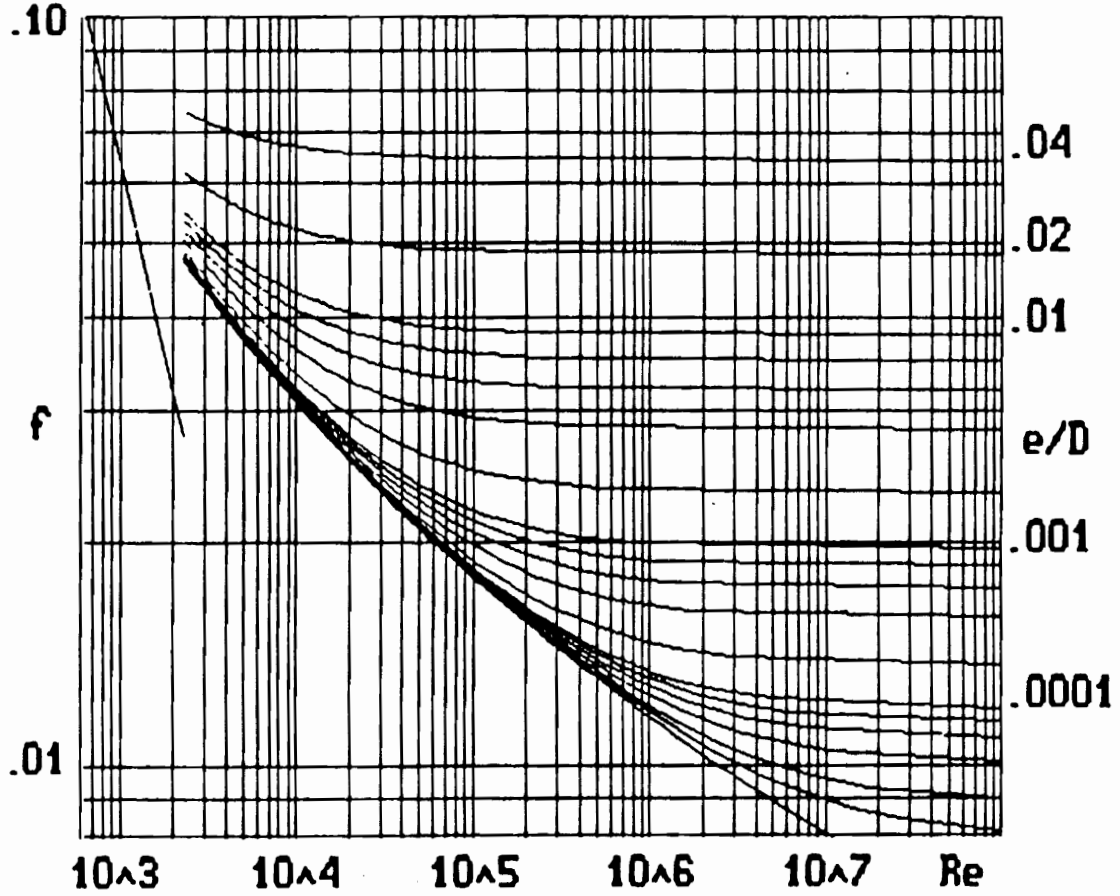


FIGURE 7.10: Moody Diagram Produced by Program MOODY

CHAPTER 8: COMPRESSIBLE FLOW

Until this chapter only incompressible flows have been considered. At lower speeds, gases are usually modeled as incompressible with density variations neglected. Typically in airflows when the Mach number is less than about 0.3, density variations may be neglected. For flows of gases at higher speeds, density changes usually may **not** be neglected, and such flows are termed compressible flows.

The parameter most often associated with compressible flows is the Mach number, M. It is defined as the local fluid speed, V, divided by the local sound speed, a, with M=V/a. A Mach number of one corresponds to a flow moving at the local speed of sound, with greater values describing **supersonic** flows. Classifications based on Mach number are made in White [4].

$M < 0.3$: **Incompressible Flow**: density effects are negligible

$0.3 < M < 0.8$: **Subsonic Flow**: density effects become important, but shock waves do not appear

$0.8 < M < 1.2$: **Transonic Flow**: shock waves begin to appear, dividing regions of supersonic and subsonic flow

$1.2 < M < 3.0$: **Supersonic Flow**: subsonic regions disappear and shock waves are present

$3.0 < M$: **Hypersonic Flow**: shock waves and other flow variations are extremely strong

It is also convenient to define stagnation (total) parameters when dealing with compressible flow. Total temperature, pressure, and density,

and sound speed (T_0, P_0, ρ_0, a_0) are the usual variables defined in this manner. Stagnation temperature, T_0, is defined as the temperature of a fluid if it were brought to rest adiabatically with no work done. Stagnation pressure, P_0, is the pressure that would follow from an isentropic, no work stagnation process. The stagnation density, ρ_0, corresponds to the state with stagnation temperature, T_0, and pressure, P_0. Stagnation sound speed, a_0, is the local speed of sound of the stagnated fluid. The nought subscript denotes the stagnation condition. These parameters are important reference properties.

The sonic state in a flow is the state where the local Mach number is one, M=1. Properties (i.e. temperature, sound speed, pressure, and density) at the sonic state are called critical properties, and are denoted by a star superscript. Like the stagnation properties, the values of T^*, P^*, ρ^*, and a^* are useful reference properties. In variable area flows, the area of the location of the sonic point is designated by A^*.

The largest of the programs included in this chapter, COMPRESS, examines four fundamental one dimensional compressible flows of a calorically perfect, ideal gas for (1) isentropic, variable area flow, (2) duct flow with friction, (3) reversible duct flow with heat transfer, and (4) flow across a normal shock. Although these categories are different in model assumptions, they are typically taught as a group, and hence are included together in this one program. The program is set up to provide numerical values for particular flow variables. The flows may also be described through the use of plots and tables.

Program OBLIQUE considers flow passing through a shock at an angle, and PRANDTLM deals with Prandtl-Meyer waves created by a supersonic flow passing an expansion corner. FREEJET employs the method of characteristics to plot expansion and compression waves created by flow exiting an underexpanded, supersonic nozzle. NOZZLE considers compressible flow through a converging-diverging nozzle, including cases when a shock waves exists.

8.1 COMPRESS

In the program COMPRESS, compressible one-dimensional flow of an ideal, calorically perfect gas is considered. Properties (T, P, ρ, V, h, a, etc.) of the fluid may be altered in any of the following manners. A reversible and adiabatic flow may occur in a variable area passage. Alternatively, heat may be reversibly added or removed from a constant area flow. Friction (i.e. from duct walls) may change the properties of an adiabatic, constant area flow. Finally, a constant area, adiabatic flow may pass from supersonic to subsonic states through a discontinuity which strongly alters the flow parameters within a near-zero distance. Such a discontinuity is called a shock. In this program, only shocks normal to the flow direction are considered.

The four categories of flow listed in the preceding paragraph are sub programs of COMPRESS. The user may choose from one of the four sub programs listed below:

1. **Isentropic Flow**: one-dimensional, variable area, reversible and adiabatic flow of a calorically perfect, ideal gas.

2. **Duct flow With Heat Transfer** - Rayleigh Flow: one-dimensional, constant area flow of a calorically perfect, ideal gas with reversible heat addition or removal

3. **Duct Flow With Friction** - Fanno Flow: one-dimensional, constant area and adiabatic flow of a calorically perfect, ideal gas with friction.

4. **Flow Across A Normal Shock**: one-dimensional, constant area and adiabatic flow of a calorically perfect, ideal gas through a discontinuity, with the flow passing from a supersonic to a subsonic state.

Next, a value for the specific heat ratio must be entered. The user then chooses among creating a plot, listing a table of values, or calculating specific values.

8.1.1 Isentropic Flow

For the case of a one-dimensional, variable area, reversible and adiabatic flow of a calorically perfect, ideal gas, the continuity, momentum, and energy equations may be expressed as:

$$\rho_1 V_1 A_1 = \rho_2 V_2 A_2$$

$$P_1 A_1 + \rho_1 V_1^2 A_1 = P_2 A_2 + \rho_2 V_2^2 A_2$$

$$h_1 + \frac{V_1^2}{2} = h_2 + \frac{V_2^2}{2} \tag{8.1}$$

Because the flow is reversible and adiabatic, only area variations cause differences in the fluid parameters between states one and two. However, instead of developing relationships for the parameters between states one and two, ratios of the static properties to their stagnation counterparts are typically considered. The ratio of each property to its stagnation value can be expressed as a function of only Mach number and specific heat ratio. These expressions are found using various combinations of the energy equation, the ideal gas equation of state, the definition of sound speed, the definition of the Mach number, and the isentropic process equation.

$$\frac{T}{T_o} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{-1}$$
$$\frac{P}{P_0} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{-\gamma/(\gamma-1)} \qquad (8.2)$$
$$\frac{P}{P_0} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{-1/(\gamma-1)}$$

If, at any point in any flow, the flow is imagined as brought to rest with no work and no heat addition or removal, the static to total temperature ratio in Equations (8.2) is valid. If at the same point, the flow is imagined as stopped reversibly with no work and no heat addition or removal, the static to total pressure ratio in Equations (8.2) is valid. Thus, Equations (8.2) are valid for any **point** in any flow (Duct flow with heat transfer, duct flow with friction, flow with a shock, flow in a nozzle, etc.). The flow process itself does not need to be reversible and adiabatic since only a point is considered.

In certain flow situations, either the total temperature or total pressure may be considered constant. For flow through a nozzle, flow across a shock, and duct flow with friction, the flow process itself involves no work or heat transfer. In such a case, the stagnation temperature is constant throughout the flow. Flow in a nozzle does not involve work or heat transfer, and the flow is reversible. In such a case, both the stagnation temperature and stagnation pressure are constant. Because Rayleigh flow (duct flow with heat transfer) does involve heat transfer, neither the total

temperature nor the total pressure may be considered constant throughout the flow.

Two other ratios are typically considered when dealing with isentropic variable area flow. These relate the area and the flow velocity to their counterparts at the sonic state.

$$\frac{A}{A^*} = \frac{1}{M}\left[\frac{2}{\gamma+1}\left(1+\frac{\gamma-1}{2}M^2\right)\right]^{(1/2)(\gamma+1)(\gamma-1)}$$
$$\frac{V}{V^*} = M\left[\frac{2}{\gamma+1}\left(1+\frac{\gamma-1}{2}M^2\right)\right]^{(1/2)} \qquad (8.3)$$

The area, A, should change gradually for consistency with the one-dimensional assumption. In a variable area flow, the value of A* is the area where the sonic state occurs. The minimum area in a flow passage is called the throat area, A_t. If a sonic state is to occur in a flow passage, it must occur at the throat. However, a given flow passage need not contain a sonic state. Although a sonic state need not physically exist, it represents a useful reference point. In a given variable area, isentropic flow, the values of A^* and V^* are constant.

When executing this code, the user has the option of producing a plot, listing a table of values, or calculating specific values. A plot resembling Figure 8.1 may be produced, where the static to total ratios given in Equations (8.2), and the ratios relative to the sonic state given in Equations (8.3), are plotted as a function of Mach number.

The user may also elect to produce a table. For given values of the Mach number, the table lists values of the ratios of the static to total

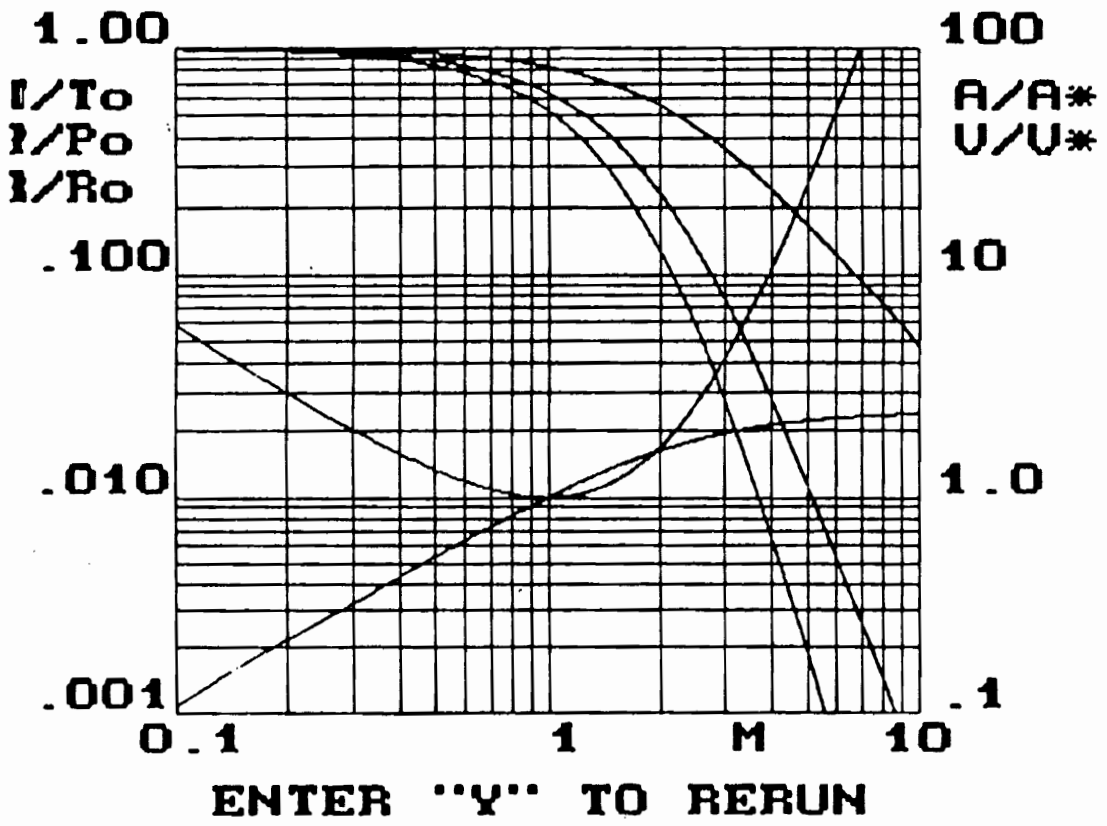


FIGURE 8.1: Plot of Isentropic Ratios Versus Mach Number

temperature, T/T_0, static to total pressure, P/P_0, and static to total density, ρ/ρ_0 as well as values for the area ratio, A/A^*, and the velocity ratio, V/V^*. The table produced is similar to Table A.1 of Anderson [2] and Table B.1 of White [4].

If the user opts to calculate specific values, one parameter value must be entered while the program solves for the rest. The user may choose any of the ratios, or the Mach number, as the input parameter. If the Mach number is entered, the equations may be solved directly for T/T_0, P/P_0, ρ/ρ_0, A/A^*, and V/V^*. Otherwise, the appropriate equation is used to determine the Mach number. This may be done explicitly for all of the relations expect the area ratio. For the area ratio, a Newton iteration must be used to numerically determine the Mach number. Since the function representing the area ratio is double valued, if it is supplied by the user, the user must specify whether a supersonic or a subsonic solution is to be determined. An initial guess is then calculated for the Newton iteration. Once the Mach number is determined, the other ratios may be calculated.

8.1.2 Flow With Heat Transfer Rayleigh Flow

For the case of a one-dimensional, constant area, and reversible flow of a calorically perfect, ideal gas with heat addition or removal, the continuity, momentum, and energy equations may be expressed as:

$$\rho_1 V_1 = \rho_2 V_2$$
$$P_1 + \rho_1 V = P_2 + \rho_2 V_2 \tag{8.4}$$
$$h_1 + \frac{V_1^2}{2} + q = h_2 + \frac{V_2^2}{2}$$

where q is the heat transferred to the fluid per unit mass. These equations, along with the ideal gas equation of state, the definition of sound speed, and the definition of Mach number, may be employed to produce the following Mach number relations:

$$\frac{T}{T^*} = \frac{(\gamma+1)^2 M^2}{\left(1+\gamma M^2\right)^2}$$

$$\frac{P}{P^*} = \frac{(\gamma+1)}{1+\gamma M}$$

$$\frac{T_o}{T_o^*} = \frac{(\gamma+1)M^2\left[2+(\gamma-1)M^2\right]}{\left(1+\gamma M^2\right)^2} \tag{8.5}$$

$$\frac{P_o}{P_o^*} = \frac{\gamma+1}{1+\gamma M^2}\left[\frac{2+(\gamma-1)M^2}{\gamma+1}\right]^{\gamma/(\gamma-1)}$$

$$\frac{V}{V^*} = \frac{\rho^*}{\rho} = \frac{(\gamma+1)M^2}{1+\gamma M^2}$$

Here the fluid properties are related to their values at the sonic condition.

From the energy equation, the heat transfer is proportional to the change in stagnation temperature; e.g. $q = Cp\,(T_{o2} - T_{o1})$. If T_{o2} is greater than T_{o1}, then heat has been added to the flow.

As with isentropic flow, the user may choose to generate a plot, a table, or to calculate specific values. The plot will resemble Figure 8.2, which shows that if a sufficient amount of heat is added, the flow will become choked (M = 1). This is true regardless of whether the flow is initially supersonic or subsonic.

The program will also create a table which lists, for given values of Mach number, values of the ratios T/T^*, V/V^*, ρ/ρ^*, P/P^*, T_o/T_o^*, and P_o/P_o^*, given in Equations (8.4). This table is similar to Table A.3 in Anderson [2] and Table B.3 in White [4].

As with the isentropic option, the user may choose to have specific values calculated for each of the ratio parameters. If the Mach number is not the entered quantity, the user must specify if a supersonic or a subsonic solution is desired. Then the appropriate equation must be solved for the Mach number. This may done explicitly for all of the ratios except the total pressure ratio, where a Newton iteration is used. Once the Mach number has been calculated, the values of the ratios T/T^*, V/V^*, ρ/ρ^*, P/P^*, T_o/T_o^*, and P_o/P_o^* are found and displayed.

8.1.3 Constant Area Duct Flow With Friction

For the case of a one-dimensional, constant area, adiabatic flow of a calorically perfect, ideal gas with friction, the continuity, momentum, and energy equations may be expressed as:

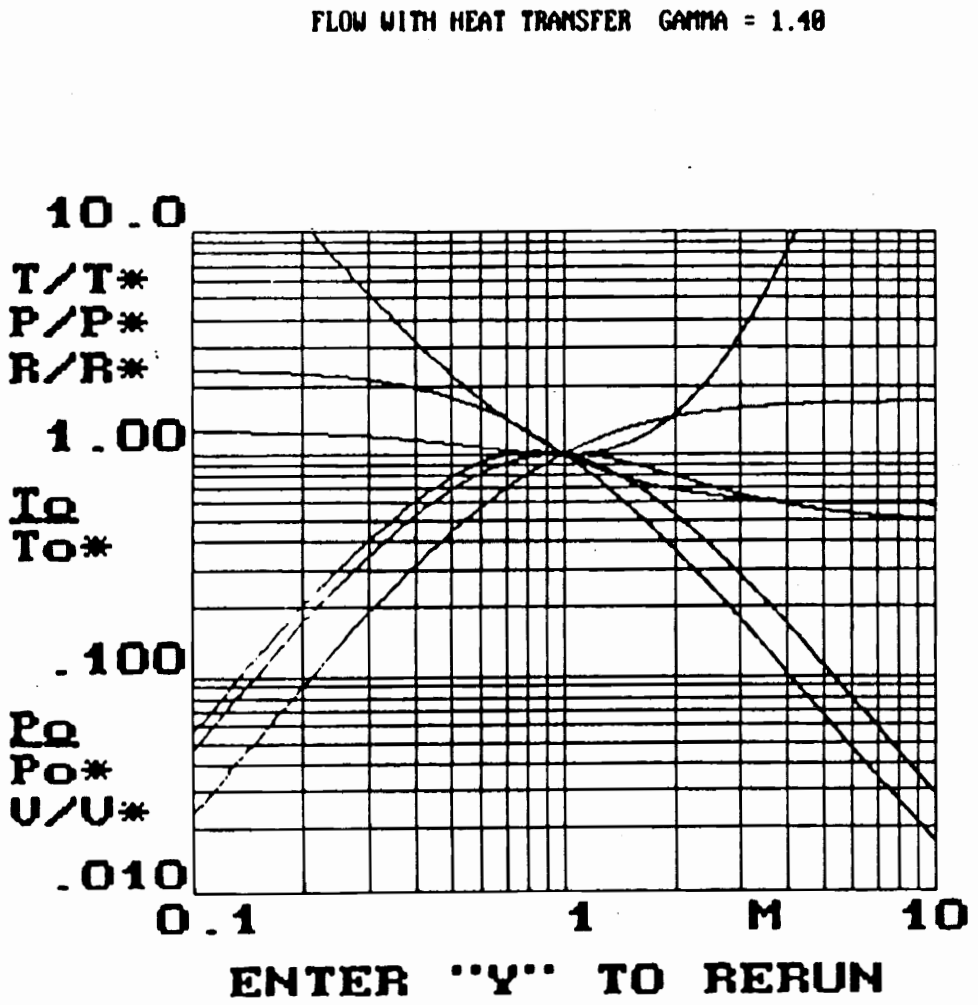


FIGURE 8.2: Plot of Rayleigh Flow Ratios Versus Mach Number

$$\rho_1 V_1 = \rho_2 V_2$$

$$P_1 + \rho_1 \frac{V_1^2}{2} = P_2 + \frac{V_2^2}{2} + \frac{1}{2}\rho u \frac{4fL}{D}$$

$$h_1 + \frac{V_1^2}{2} = h_2 + \frac{V_2^2}{2} \tag{8.6}$$

The variable, D, is the duct diameter, f is the average friction factor between states one and two, and L is the pipe length between states one and two. The friction factor in the momentum equation above is the same as that considered in the Moody diagram. (See program MOODY.)

An alternate form of the friction factor term is expressed in terms of a quantity, L^*, in which case f is the average friction factor between a state at x=0, with Mach number, M, and the sonic state where $x=L^*$. L^* is the pipe distance required to choke (M=1) a flow that has a Mach number M at a state where x=0. Thus the friction factor term in the momentum equation above may be expressed as

$$\frac{4fL}{D} = \left.\frac{4fL^*}{D}\right)_1 - \left.\frac{4fL^*}{D}\right)_2 \tag{8.7}$$

where the subscripts 1 and 2 denote the same two states as in Equations (8.6). Choking can occur only at the end or exit of such a duct.

Equations (8.6) may be manipulated along with the ideal gas equation of state, the definition of sound speed, and the definition of the Mach number to obtain the following functions of the Mach number and the specific heat ratio.

$$\frac{T}{T^*} = \frac{\gamma+1}{2+(\gamma-1)M^2}$$

$$\frac{P}{P^*} = \frac{1}{M}\left[\frac{\gamma+1}{2+(\gamma-1)M^2}\right]^{\frac{1}{2}}$$

$$\frac{V}{V^*} = \frac{\rho^*}{\rho} = M\left[\frac{\gamma+1}{2+(\gamma-1)M^2}\right]^{\frac{1}{2}} \tag{8.8}$$

$$\frac{Po}{Po} = \frac{1}{M}\left[\frac{2+(\gamma-1)M^2}{\gamma+1}\right]^{(\frac{1}{2})(\gamma+1)/(\gamma-1)}$$

$$\frac{fL^*}{D} = \frac{1-M^2}{\gamma M^2} + \frac{\gamma+1}{2\gamma}\ln\left[\frac{(\gamma+1)M^2}{2+(-1)M}\right]$$

The user may choose to generate a plot, a table, or to calculate specific values. For a user specified value of the specific heat ratio, the values of the ratios given in Equations (8.8) will be plotted versus Mach number. The plot will be similar to Figure 8.3.

If the user elects to produce a table, values of the ratios given in Equations (8.8): T/T^*, V/V^*, ρ/ρ^*, P/P^*, $P_o/P_o{}^*$, and fL^*/D will be listed for different values of the Mach number. The table is similar to Table A.4 in Anderson [2] and Table B.4 in White [4].

If specific values are to be calculated, the user must specify which of the ratios, the friction factor term, or the Mach number is to be supplied. If

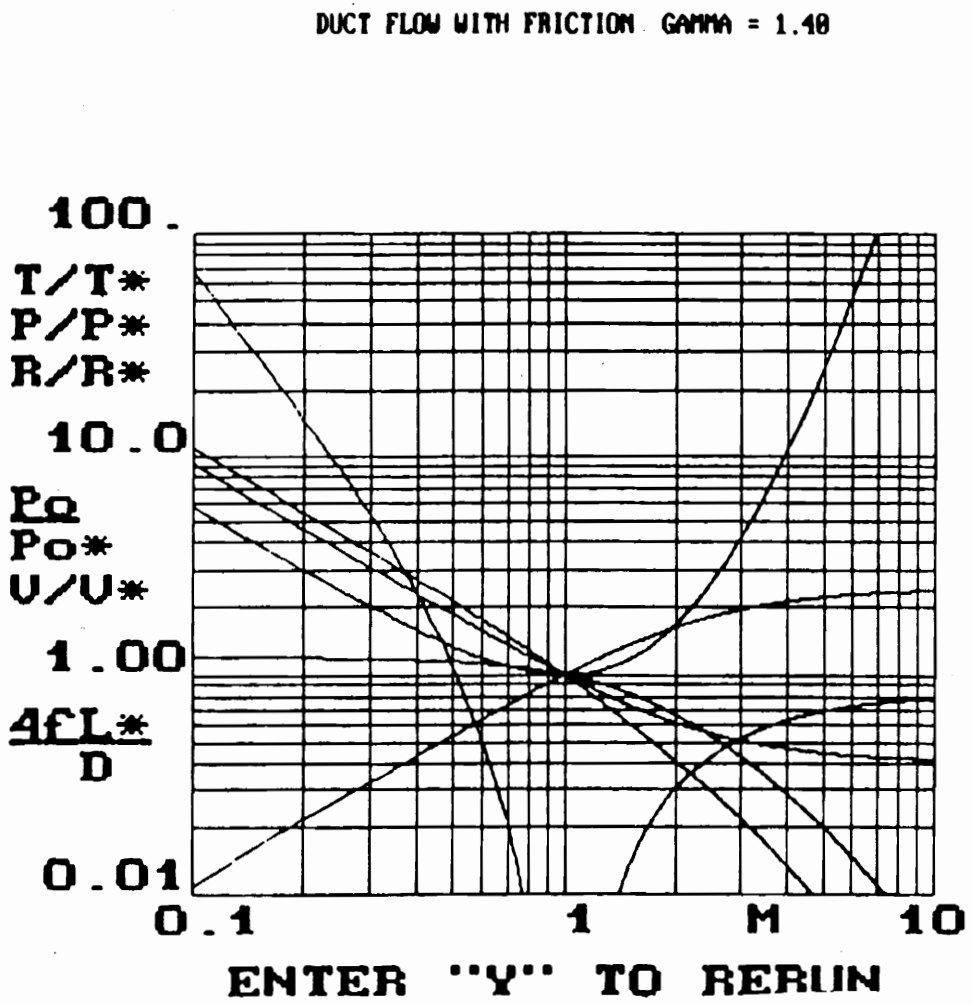


FIGURE 8.3: Plot of Fanno Flow Ratios Versus Mach Number

the Mach number is not specified, it is found first using the appropriate equation. All of the equations may be solved explicitly for the Mach number, except for the friction factor and total pressure equations. Since the equations for both the friction factor and the total pressure ratio are double valued, the user must specify whether a subsonic or supersonic solution is to be found. Also, the equations cannot be solved directly for the Mach number, hence, an initial value is assumed depending on which solution (supersonic or subsonic) is to be calculated, and a Newton iteration is used to obtain Mach number values. Once the Mach number has been calculated, all of the remaining parameters are evaluated.

8.1.4 Flow Across A Normal Shock

Flow Across A Normal Shock is a one-dimensional, constant area, and adiabatic flow process of a calorically perfect, ideal gas through a discontinuity aligned perpendicular to the flow direction, with the flow passing from a supersonic to a subsonic state.

Such a discontinuity can be formed as modeled in Figure 8.4, where an initially weak compression wave propagates through a medium ultimately developing a region where the fluid properties are strongly altered across a discontinuity. In Figure 8.4, the region of the pressure wave above the horizontal axis has a greater pressure and temperature, and a greater disturbance speed. The region below the horizontal axis has lower pressure and temperature, and a lower disturbance speed. The flow region above the axis tends to overtake the lower region. This process tends to

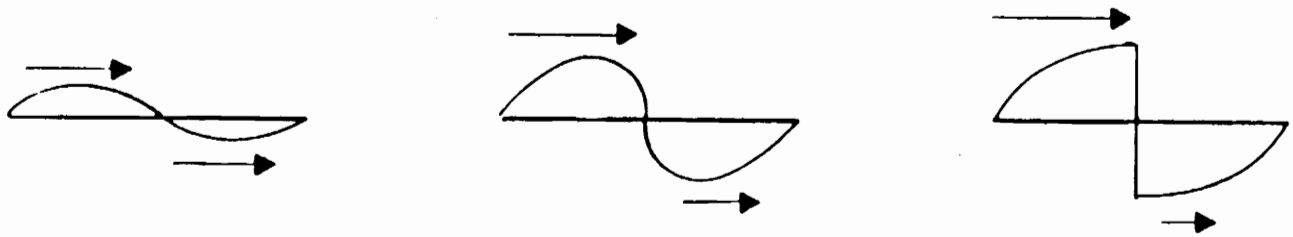


FIGURE 8.4: Formation of a Normal Shock By a Strong Disturbance

steepen the compression wave. When the pressure wave becomes vertical, a discontinuity exists and this is called a shock wave.

In the three previous sections (isentropic flow, constant area flow with heat transfer, and constant area duct flow with friction), changes in flow properties were imagined to be continuous or "smooth". This is not the case when shock waves are present in a flow. Large variations in the flow properties may occur across essentially zero distances. In this program, only shocks normal to the flow direction are considered. For normal shocks, the Mach number upstream of the shock is always supersonic, and the Mach number downstream of the shock is always subsonic.

The following are the equations for fluid motion across a normal shock

$$\rho_1 V_1 = \rho_2 \ V_2$$

$$P_1 + \rho_1 V_1^2 = P_2 + \rho_2 V_2^2$$

$$h_1 + \frac{V_1^2}{2} = h_2 + \frac{V_2^2}{2} \tag{8.9}$$

where the subscript, 1, denotes the supersonic flow state immediately upstream of the shock, while the subscript, 2, denotes the subsonic state immediately downstream of the shock. There is no area consideration since the distance between states 1 and 2 is the same as the shock thickness, which, for this analysis, is modeled as zero.

The normal shock equations, based on the upstream Mach number, relating the properties of the downstream state to the upstream state are derived in Anderson [2] and are as follows:

$$\frac{T_2}{T_1} = \frac{\left[2+(\gamma-1)M_1^2\right]\left[2\gamma M_1^2-(\gamma-1)\right]}{(\gamma+1)^2 M_1^2}$$

$$\frac{P_2}{P_1} = \frac{1}{\gamma+1}\left[2\gamma M_1^2-(\gamma-1)\right]$$

$$\frac{V_2}{V_1} = - = \frac{(\gamma+1)^2 M_1^2}{2+(\gamma-1)M_1} \qquad (8.10)$$

$$\frac{Po_2}{Po_1} = \frac{A_1^*}{A_2^*} = \left[\frac{(\gamma+1)M_1^2}{2+(\gamma-1)M_1^2}\right]^{\gamma/(\gamma-1)}\left[\frac{\gamma+1}{2\gamma M_1^2-(\gamma-1)}\right]^{1/(\gamma-1)}$$

$$M_2^2 = \frac{2+(\gamma-1)M_1^2}{2\gamma M_1^2-(\gamma-1)}$$

Again the subscript, 1, denotes the upstream state, and the subscript, 2, identifies the downstream state. The area ratio relates the area of the sonic states upstream and downstream of the shock, and is useful when combining a nozzle and a shock analysis.

As in the three previous sections, the user may choose to generate a plot, a table, or to calculate specific values. Figure 8.5 shows an example of the plot generated by the program where values of the parameters given in Equations (8.10) are plotted as a function of the upstream Mach number, M_1.

If the user elects to construct a table, values of T_2/T_1, P_2/P_1, V_2/V_1, ρ_2/ρ_1, Po_2/Po_1, A_2^*/A_1^*, and the downstream Mach number, M_2, are listed for different values of the upstream Mach number M_1. The table produced by the program is similar to Table A.2 in Anderson [2] and Table B.2 in White [4].

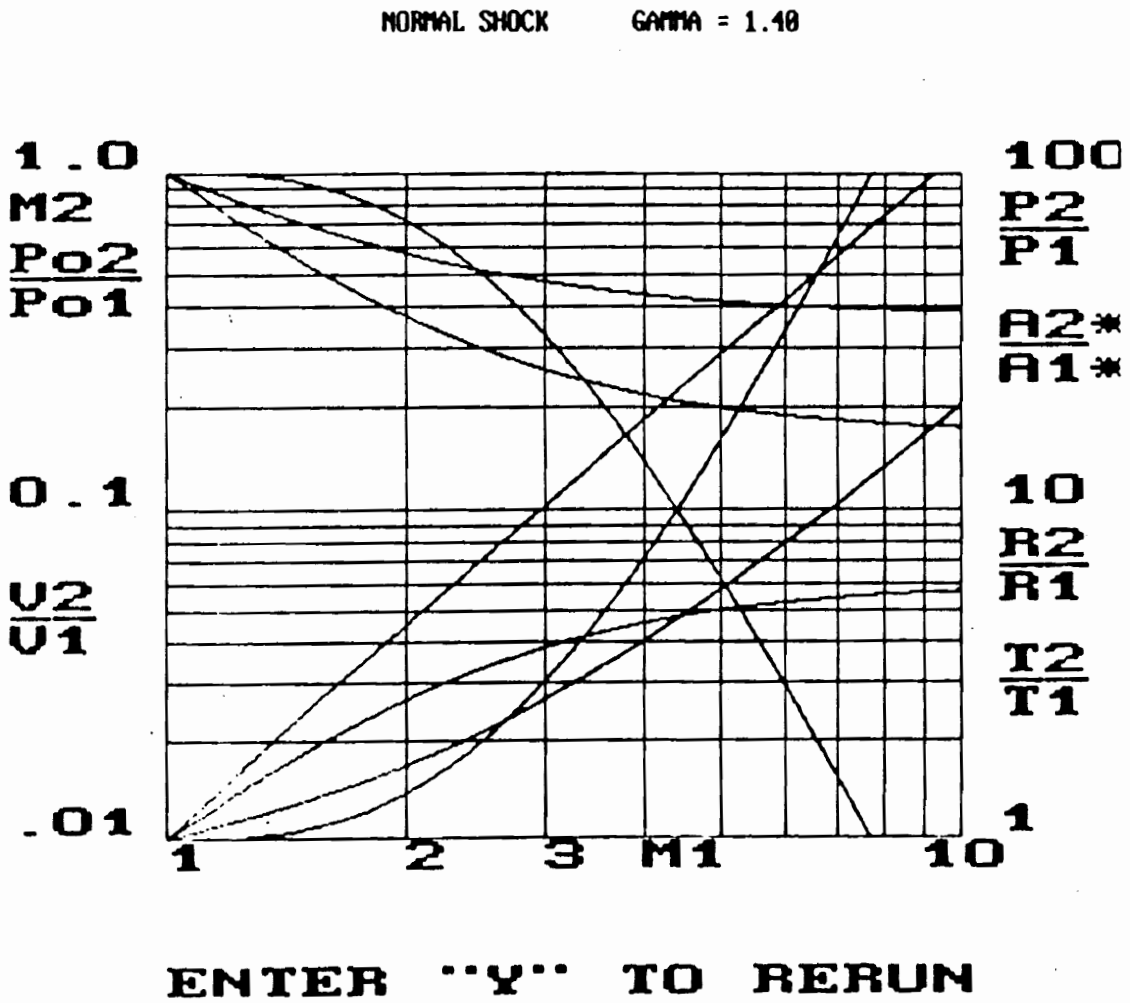


FIGURE 8.5: Plot of Normal Shock Relations Versus Upstream Mach Number

When calculating specific values, the user may input any of the ratios, the upstream Mach number, or the downstream Mach number. If the upstream Mach number, M_1, is not entered, it is determined first using the appropriate relation. The other parameters given in Equations (8.10) are then calculated. All of the parameters given in Equations (8.10) may be solved explicitly for the upstream Mach number except the total pressure ratio. In this case, a Newton iteration is employed to determine the upstream Mach number, M_1.

8.2 OBLIQUE

The normal shock considered in program COMPRESS can be viewed as a special case of a more general family of oblique shocks waves. An oblique shock wave occurs when a supersonic flow approaches a discontinuity (as described in program COMPRESS - Normal Shock) at some angle, ß, called the wave angle, as shown in Figure 8.6. The flow itself is then deflected by an angle, θ, appropriately named the deflection angle. Oblique shocks are common in supersonic flows over wedge and airfoil shapes.

The following relationship among θ, ß, and Mach number is derived in Anderson [2] and becomes the basis of the oblique shock analysis.

$$\tan\theta = 2\cot ß\left[\frac{M_1^2\sin^2\beta - 1}{M_1^2(\gamma + \cos 2\beta) + 2}\right] \tag{8.11}$$

A plot of the equation for various upstream Mach numbers, M_1, is presented in Figure 8.7. Note that for a given deflection angle, θ, there are two possible values of the wave angle, ß. The larger of the two values corresponds to the "strong" shock solution, while the lesser corresponds to a "weak" shock. Usually, under real flow conditions, the "weak" shock solution occurs. Also, notice that for a given specific heat ratio, there exists a maximum deflection angle for a particular wave angle. In a physical flow, if θ is greater than θ_{max}, then the shock will become detached and curved. This case is not considered in the program.

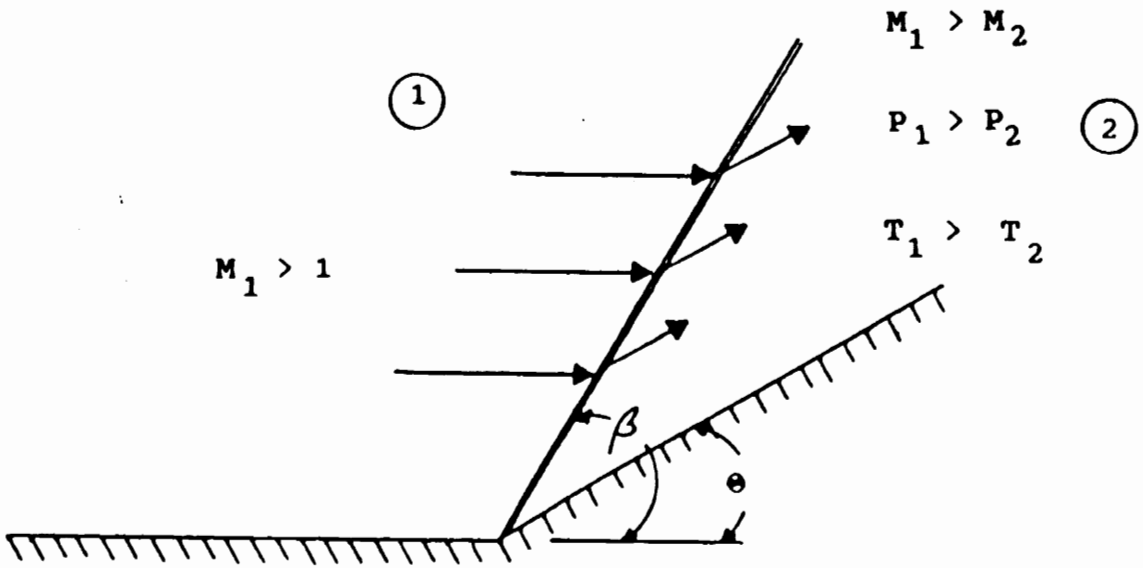


FIGURE 8.6: Supersonic Flow Past A Convex Corner Producing an Oblique Shock

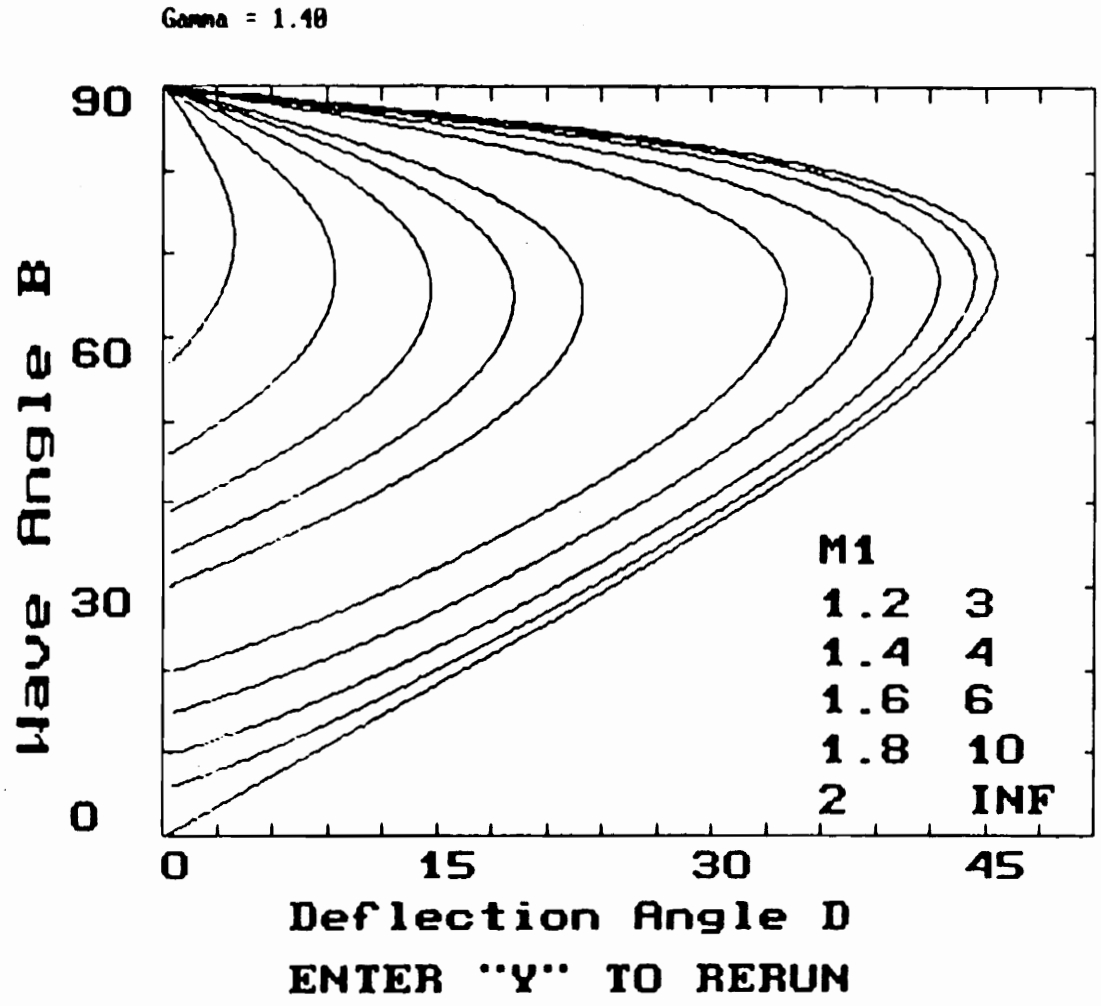


FIGURE 8.7: θ - β - M Plot for Oblique Shocks

When executing the program for a user specified value of the specific heat ratio, the user may choose between creating the θ-ß -M plot or, calculating specific values. These specific values are the parameters given in Equations (8.10) of Section 8.1.4: T_2/T_1, P_2/P_1, V_2/V_1, ρ_2/ρ_1, Po_2/Po_1, A_2^*/A_1^*, and the downstream Mach number, M_2.

When calculating specific values, the user must first enter the upstream Mach number, M_1, and either the deflection or wave angle. If θ is the input parameter, a Newton iteration is used to determine ß. The normal shock relations are then employed for calculating the values. This is done by determining the Mach number normal to the shock $Mn_1 = M_1 \sin \theta$. The normal upstream Mach number is then used to calculate the downstream normal Mach number Mn_2. The absolute Mach number M_2, which is the characteristic Mach number in the downstream flow direction, is found by $M_2 = Mn_2/\sin(ß - \theta)$. Calculations of the various parameters are performed in the same manner as in program COMPRESS.

8.3 PRANDTLM

Shock waves generally occur when a supersonic flow approaches some obstruction (which may be thought of as decreasing the effective flow area). A converse of sorts to this occurrence is a supersonic flow which passes an expansion corner. A model for this case is a series of expansion waves emanating from the corner as shown in Figure 8.8. Unlike a shock wave, these expansion, or Prandtl-Meyer waves (also called Mach lines), are modeled as an isentropic change in the flow variables occurring across the wave. Note in the figure that the expansion waves occur within a region bounded by the forward and rearward Mach lines. The expansion is, in fact, continuous within this region, although it is typically analyzed by assuming a finite number of discrete changes. Three angles are labeled in the diagram. θ is a measure of the expansion corner angle. μ_1 is the angle of the forward expansion wave (Mach line), while μ_2 is the angle of the trailing wave (Mach line). It can be shown that the angle, μ, is related to the Mach number with

$$\mu = \arcsin(1/M) \tag{8.12}$$

Properties of the expansion waves are calculated using the equation

$$\omega(M) = \left(\frac{\gamma+1}{\gamma-1}\right)^{\frac{1}{2}} \tan^{-1}\left[\frac{\gamma-1}{\gamma+1}\left(M^2-1\right)\right]^{\frac{1}{2}} - \tan^{-1}\left(M^2-1\right)^{\frac{1}{2}} \tag{8.13}$$

Because both Prandtl and Meyer, in 1907 and 1908 respectively, developed this wave theory independently, the function, ω, is called the Prandtl-Meyer

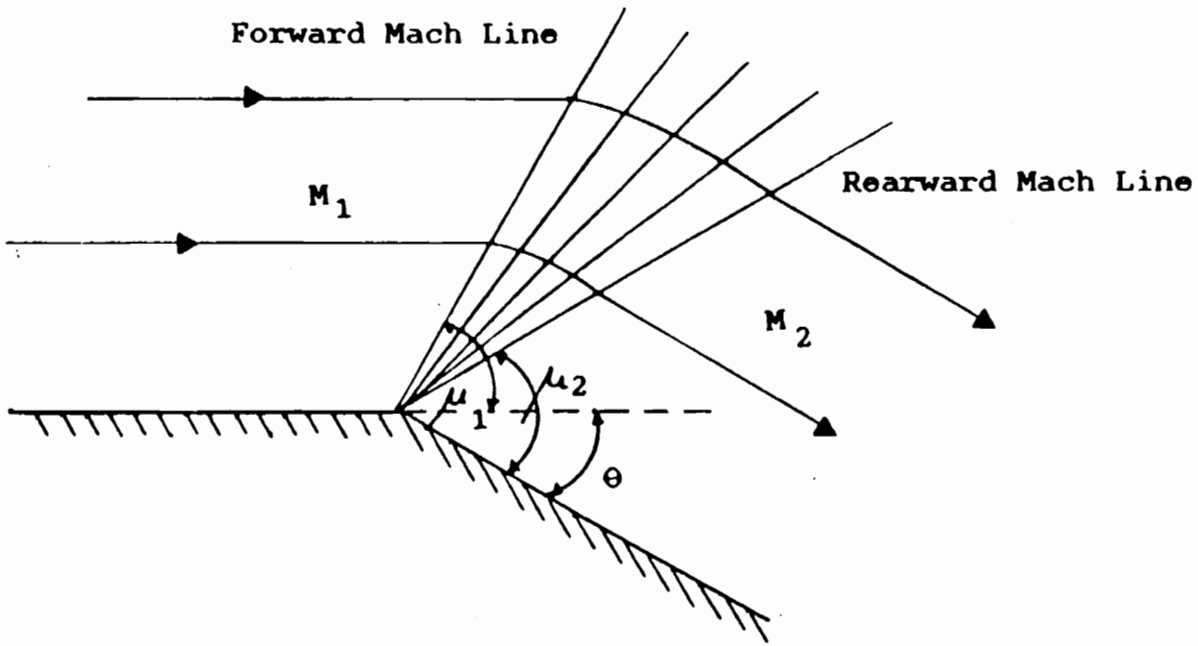


FIGURE 8.8: Supersonic Flow Past an Expansion Corner

function. The relation was developed for calorically perfect, ideal gases and has the property $\Delta\theta = \Delta\omega(M)$. For typical expansion problems, one generally knows the upstream Mach number and the expansion angle θ. With that information it is possible to solve for the downstream Mach number, M_2, using the Prandtl-Meyer function. In practice, the expansion regime is modeled as occurring across an arbitrary, but discreet, number of expansion waves (Mach lines).

The user may choose to tabulate values of the Prandtl-Meyer function, ω, and μ for different values of the Mach number. This table is similar to Table A.5 in Anderson [2] and Table B.5 in White [4].

Specific values may also be calculated. In this case the isentropic relations listed in program COMPRESS are used. The user is free to enter any of the static to total ratios, the Mach number, or Prandtl-Meyer function. If the Mach number is not entered, the program will first determine its value, then solve for the other parameters. If the Prandtl-Meyer function is selected as the input variable, a Newton iteration is performed in order to determine the Mach number. Otherwise, the Mach number may be calculated explicitly.

8.4 FREEJET

This program considers the free expansion of a supersonic flow exiting a nozzle where the exit pressure is greater than the ambient pressure. This situation is called an underexpanded nozzle flow. In such a case, the flow is modeled as a Prandtl-Meyer process until the pressure of the fluid equals that of the surroundings.

As the flow expands upon exiting the nozzle, Prandtl-Meyer expansion waves of a single family emanate from the nozzle corners. These waves of the same family are directed toward the centerline of the flow, where they reflect and become waves of the second family. This forms a cross-hatched area where waves of the two families intersect. This area is called the nonsimple region. In the program the expansion is modeled by defining N waves emanating from one exit corner of the nozzle. The finite number of waves are of the same family (same general flow direction) and are straight until they reach the nonsimple region. Within the nonsimple region, the segments between intersection points are straight. The flow solution is performed by numbering the wave-wave and wave-boundary intersection points beginning with the point where the first Mach wave intersects the jet centerline. Employing the method of characteristics, the flow direction, θ, the Mach number, the ratio of static to total temperature, and the ratio of static to total pressure at each point are determined. A Newton iteration is performed to determine the Mach number at each intersection point. The isentropic relations based on Mach number and

specific heat ratio are used to calculate the static to total ratios. This procedure is called the lattice point method.

The expansion waves reflect from the jet boundaries (including the centerline) as compression waves. This is in order to maintain the constant pressure boundary condition. When the compression waves reach the jet boundaries they reflect again, but as expansion waves. Here another non-simple region occurs. This cycle would be repeated indefinitely if viscous effects and mixing were neglected. A complete cycle is characterized by the jet boundary expanding and contracting again. Such a form is necessary in order to equalize pressures.

When executing the program, the user inputs a value for the ratio of the nozzle exit pressure to the ambient pressure (Pe/Pa), and the nozzle exit Mach number. Low values of the pressure ratio create larger expansions and more complicated wave patterns. For scaling purposes, the nozzle exit area is proportioned so as to allow the jet boundaries to remain within the confines of the screen.

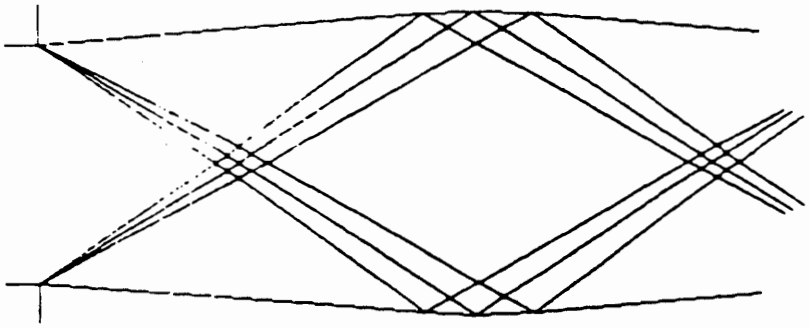
Only the wave pattern above the X axis is calculated since the flow is symmetric about the horizontal centerline. Instead of thinking of the waves as reflecting off the centerline, one may consider two sets of expansion waves emanating from both of the nozzle exit corners. In this case, the waves do not reflect from the centerline, but intersect with the same family of Mach lines on the other side of the centerline. When the program generates the wave pattern, the pattern is mirrored below the centerline.

The nozzle and jet boundaries are displayed in yellow, expansion waves in blue, and compression waves are shown in red. After the jet has

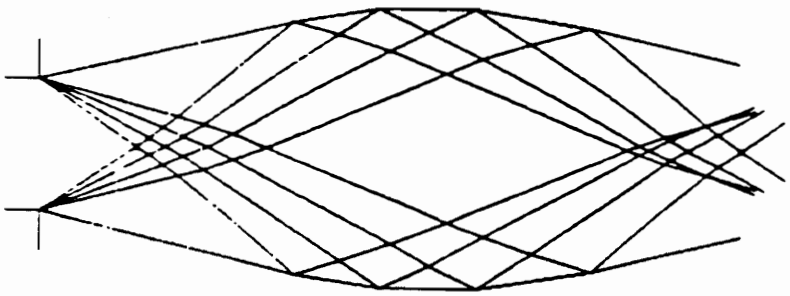
been constructed, the user may elect to display a table. The table lists the values of the flow direction, θ, Mach number, M, as well as the static to total temperature and pressure ratios for intersection points.

Figure 8.9 shows examples of different jet shapes for decreasing exit to ambient pressure ratios at a given exit Mach number of 2.

Me = 2.00 Pa/Pe = 0.000 Gamma = 1.40
Press "T" for table, "R" to rerun.



Me = 2.00 Pa/Pe = 0.500 Gamma = 1.40
Press "T" for table, "R" to rerun.



Me = 2.00 Pa/Pe = 0.200 Gamma = 1.40
Press "T" for table, "R" to rerun.

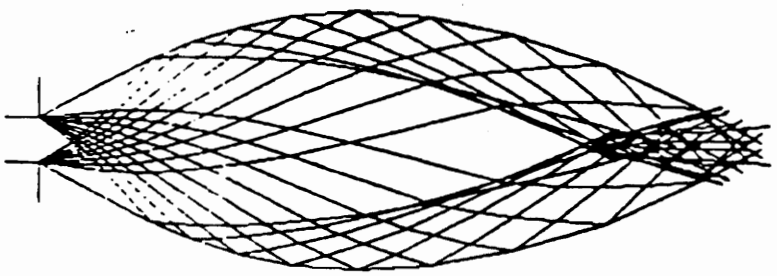


FIGURE 8.9: Sample Output For Program FREEJET

8.5 NOZZLE

In this program, the compressible flow of a calorically perfect, ideal gas through a converging-diverging (Laval) nozzle is considered. It is assumed that there is no heat transfer to the fluid and no friction acts along the nozzle surface. Flow in the converging portion is always subsonic. However, the flow in the diverging portion may be supersonic, subsonic, or both supersonic and subsonic. In the case where both supersonic and subsonic flow exists in the diverging portion, a normal shock separates the supersonic flow from the subsonic.

The flow in the diverging portion of the nozzle is governed by the ratio of the nozzle exit area to the nozzle throat area, and the back pressure. The back pressure is the pressure which exists some distance from the nozzle exit, and is not necessarily equal to the nozzle exit plane pressure.

When analyzing a nozzle, two isentropic, choked solutions are typically considered. The subsonic solution considers the flow in the diverging portion of the nozzle as entirely subsonic going through an isentropic expansion from the choked throat state. The supersonic solution is the nozzle design point solution, and assumes the flow in the diverging portion to be entirely supersonic with an isentropic expansion. For both solutions the exit pressure is equal to the back pressure.

It is usually convenient when analyzing a converging-diverging nozzle to nondimensionalize pressures by dividing them by the upstream stagnation pressure (reservoir pressure). In this case, the ratio of the back pressure to reservoir pressure is of key interest. A useful approach to nozzle

analysis is to plot this nondimensional pressure as a function of the distance along the nozzle centerline. In this case, the subsonic and supersonic isentropic, choked solutions result in exit pressure ratios, designated Pa and Pc respectively. Figure 8.10 shows the pressure distributions for the subsonic and supersonic solutions along with the exit pressure ratios Pa and Pc.

In Figure 8.10, Pb corresponds to a supersonic solution, with a normal shock in the exit plane.

For a given value of the back pressure ratio, seven possible nozzle solutions exist. The seven solutions correspond to the following points or regions.

(1) Pback > Pa

(2) Pback = Pa

(3) Pa > Pback > Pb

(4) Pback = Pb

(5) Pb > Pback > Pc

(6) Pback = Pc

(7) Pc > Pback

Each situation is discussed separately.

(1) Pback > Pa. In this case the flow throughout both portions of the nozzle is subsonic, with the Mach number at the throat being less than one. Thus, the value of A^* is not the throat area. The solution is found, for the

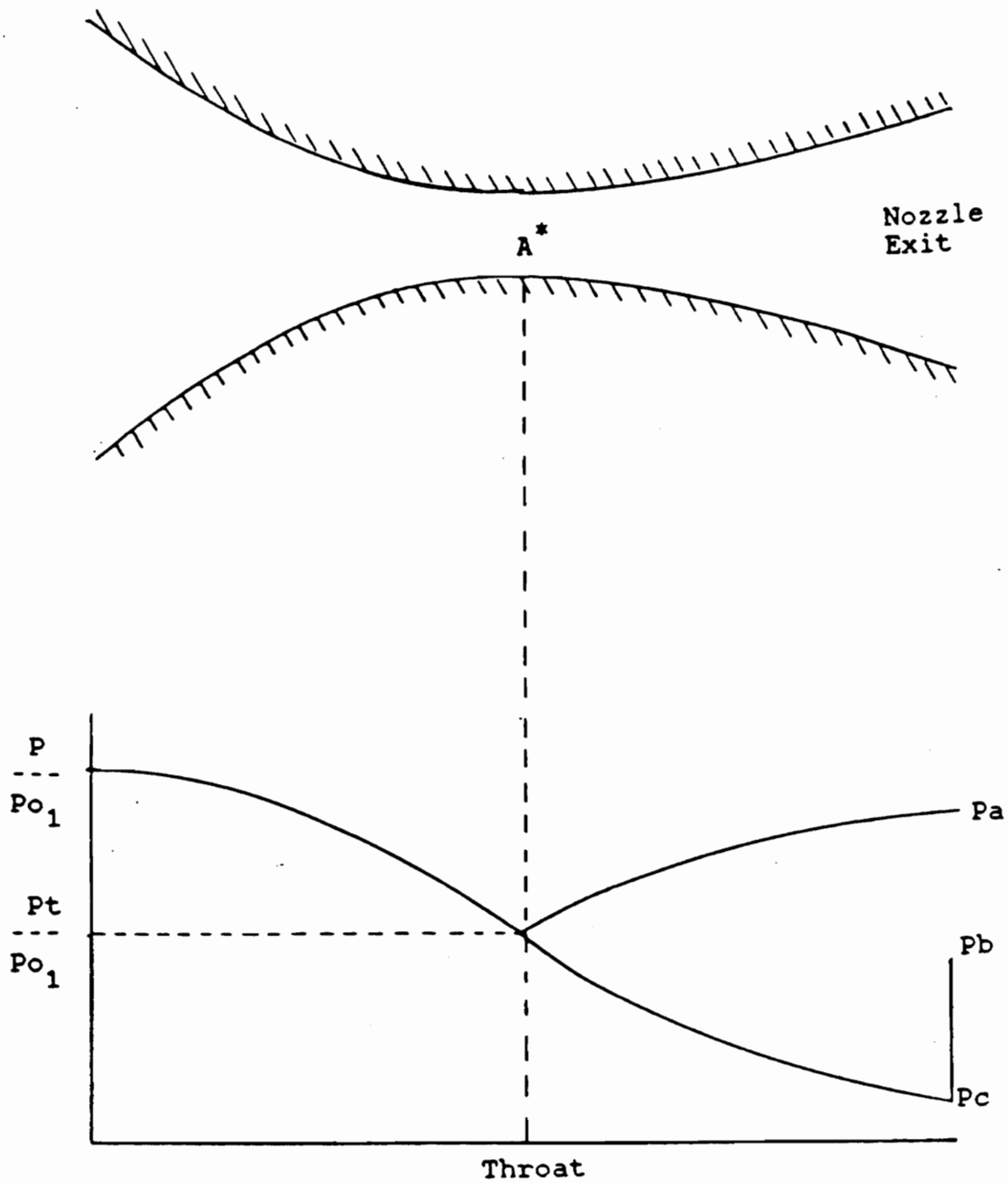


FIGURE 8.10: Supersonic and Subsonic Isentropic, Choked Solutions for Flow Through a Laval Nozzle

given back pressure ratio, by determining the exit Mach number for isentropic flow. With this exit Mach number, the value of A^* may be found. It is assumed that the area of the nozzle, as a function of the distance along its centerline, is known. Then for any point along the centerline, the ratio of the local nozzle area to A^* is known. This A/A^* value provides the Mach number at the point, using a Newton iteration. Because the area ratio is a double valued function, the initial value of the Mach number used in the Newton iteration is chosen so as to obtain a subsonic solution. Once the local Mach number is determined, the static to total pressure ratio may be calculated.

(2) Pback = Pa. In this case, the solution is the choked subsonic isentropic solution. The pressure distribution (as shown in Figure 8.10) may be found as when Pback > Pa, except that the value of A^* need not be calculated since it is equal to the physical throat area.

(3) Pa > Pback > Pb. This is the most complicated situation for which solutions must be found. The flow in the diverging portion of the nozzle is initially supersonic, and the solution follows the supersonic design point solution. At some point along the nozzle, a plane normal shock occurs, followed by a subsonic, isentropic diffusion process in the diverging portion of the nozzle. The location of the normal shock is fixed by the back pressure. Given a nozzle back pressure, the determination of the shock location requires an iterative process.

Assuming a location of the shock, the following procedure is used. The supersonic design point solution, which is found by the process described earlier for the choked isentropic subsonic solution, is assumed up

to the shock location. However, for the supersonic solution, an initial Mach number greater than one is used in the area ratio with the Newton iteration. At the normal shock location, the upstream Mach number, M_1, is determined along with the area ratio A_1/A_1^*, and the pressure ratio P_1/Po_1. Across the shock, values for A_2/A_1^*, the static pressure ratio, P_2/P_1, and the total pressure ratio, Po_2/Po_1, are calculated. An isentropic subsonic solution is then calculated downstream of the shock to the nozzle exit plane. With the area of the nozzle as a function of distance known, the Mach number may be determined using the A/A^* function. The loss of stagnation pressure across the shock changes the value of the reference A^*, with $A_2^* > A_1^*$. Once the Mach number is determined using the Newton iteration procedure, it may be used to determine the static to total pressure ratio downstream of the shock. In order to plot the pressure ratio as shown in Figure 8.10, the effects of the change in the total pressure across the shock must be considered.

The location of the shock was correctly assumed if the ratio of the exit pressure to the reservoir pressure is equal to the back pressure ratio. Usually, a trial and error procedure must be employed to determine the shock location.

(4) Pback = Pb. In this case, the supersonic design point solution is assumed until the nozzle exit plane. At the exit plane, a normal shock occurs with the exit plane pressure equal to the back pressure.

(5) Pb > Pback > Pc. When this occurs, the supersonic design point solution is assumed up to the nozzle exit plane. In this interval, the exit pressure ratio is less than the back pressure ratio, and the nozzle is said to

be overexpanded. Recompression of the flow occurs outside of the nozzle, until the exit pressure ratio equals the back pressure ratio.

(6) Pback = Pc. In this case, the nozzle solution is the supersonic design point solution with the exit pressure ratio equal to the back pressure ratio.

(7) Pc > Pback. Again, the supersonic design point solution is assumed up to the nozzle exit plane. In this case, however, the exit pressure is greater than the back pressure. Thus, the nozzle is said to be underexpanded, and a free expansion occurs outside of the nozzle until the pressure of the fluid equals the back pressure. This phenomenon is illustrated in program FREEJET.

On execution of the program, the user inputs values for the specific heat ratio and the ratio of the nozzle exit area to the throat area. A plot of the isentropic, choked subsonic and supersonic (design point) solutions are plotted. The values of the static to reservoir pressure ratios Pa, Pb, and Pc are calculated and displayed, and their locations are marked on the pressure plot. The user may then enter a value for the ratio of the back pressure to the reservoir pressure.

The program considers a two-dimensional nozzle with the known area being a sine function of the length along its centerline. Once the back pressure ratio is specified, a nozzle is drawn according to the user inputted area ratio. The program then determines which of the seven solutions is appropriate for the back pressure ratio.

Once the solution is determined, the program color codes the nozzle. This is done by specifying a color spectrum, with white corresponding to a pressure ratio of 1 and blue corresponding to a ratio of 0. The color spectrum is provided next to the pressure curve. At each location along the nozzle centerline, the pressure ratio is calculated and assigned one of 100 predefined colors. The area of the nozzle at the location is then drawn in the determined color. At the same time, the program highlights the pressure curve solution including the normal shock if appropriate. Figure 8.11 shows program output for $\gamma = 1.4$, an area ratio of 2.0, and a back pressure ratio between Pb and Pa.

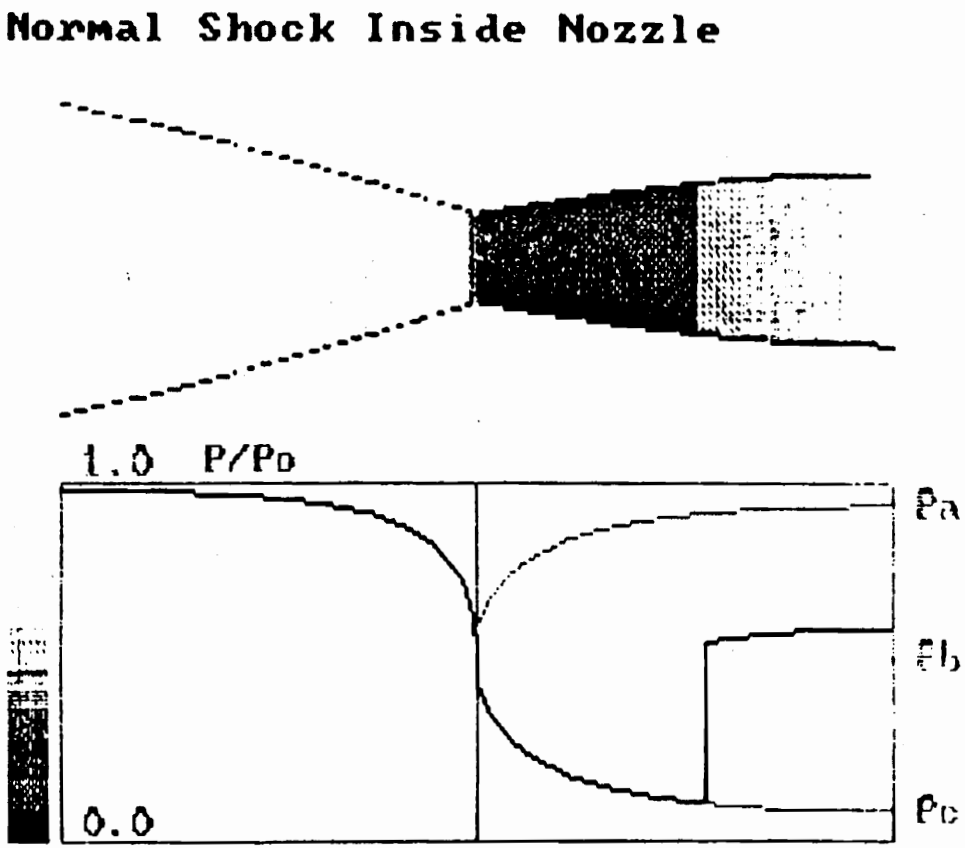


FIGURE 8.11: Sample Output for Program NOZZLE
(γ = 1.4, Ae/At = 2.0, Pb < Pback < Pa)

CHAPTER 9: HEAT TRANSFER

The purpose of the program HEAT is to demonstrate the use of program CONTOUR. CONTOUR is a color contour plotting program that can color code any rectangular matrix of data. The complete workings of this program will be discussed below.

9.1 HEAT

HEAT considers a substrate like the one shown in Figure 9.1, which comprises a section of ceramic plate. Due to symmetry, it is necessary to consider only this portion. Nodes have been placed within a grid ($\Delta X = \Delta Y = 2$mm) on the substrate. At one of the nodes within the substrate, a heating element is to be activated. The upper nodes are exposed to air at 30 C. At time zero the heating element is activated and generates 50 W/m. A transient finite difference solution is used to find the steady state temperatures at the nodes in the substrate. The following properties apply:

Air Temperature, T_a = 30 C
Air Heat Transfer Coefficient, h = 100 W/m-K
Substrate Initial Temperature, T_s = 30 C
Substrate Thermal Conductivity, K_s = 2.0 W/m-K
Generated Heat Flux, q' = 50 W/m
Delta X = Delta Y = 2.0 mm
Length of Plate, XL = 12.0 mm
Width of Plate, YL = 6.0 mm

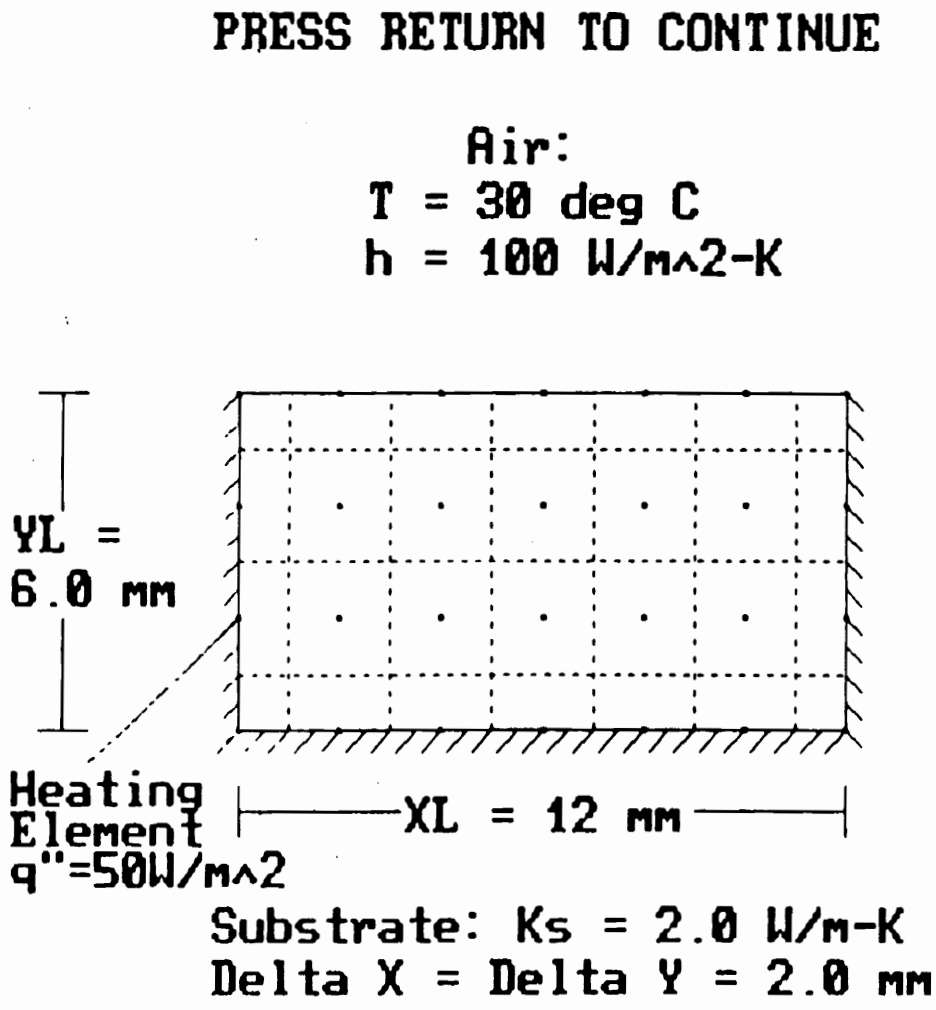


FIGURE 9.1: Substrate for Program HEAT

The governing differential equation for this type of heat transfer problem is expressed as

$$\rho c \frac{\partial T}{\partial t} = \frac{\partial}{\partial x} K \frac{\partial T}{\partial x} + \frac{\partial}{\partial y} K \frac{\partial T}{\partial y} \tag{9.1}$$

For simplification, finite difference equations are determined in terms of the aforementioned parameters, the Biot number, and the Fourier number. Equations for each of the nodes can be found in Appendix B. An appropriate time step is selected (delta t = 0.5 seconds). It is assumed that the substrate has been exposed to the ambient air for sufficient time such that its temperature is uniformly equal to that of the air. At time zero, the heating element is activated.

No user inputs are required for the program. First, the schematic shown in Figure 9.1 is displayed. The program then determines the temperature at each node for each time step. When the temperature changes between successive time increments become arbitrarily small, the solution has reached a steady state. The steady state temperatures at each node are written to the screen and a file which is to be interpreted by program CONTOUR.

9.2 CONTOUR

This program is able to color code the results of any data set in the form of a rectangular matrix. The first line of the data file must contain values representing the number of columns followed by the number of rows

in the matrix. The program then reads the data (one value for each line) starting as row one and incrementing through the columns. Row two is then considered, and so on.

While the program is reading the values, it determines the maximum and minimum of the data set. The minimum value is assigned the color blue, while the maximum is assigned the color white. Colors for all the other values are found by interpolating between these values. Values between the minimum and 33 percent of the maximum are assigned shades between blue and red, those between 33 and 66 percent become a color between red and yellow, and the values between 66 percent and the maximum are assigned a shade between yellow and white. In program CONTOUR, each data value is assigned one of a hundred predefined colors.

As the data is assigned its color, a square of that color is drawn within the display region. The right two thirds of the screen is dimensioned using the data file line stating the number of columns and rows. Here the matrix is redisplayed in color code. A key showing the values corresponding to blue, red, yellow, and white is constructed to the left. The program also asks the user what the matrix data represents, (i.e. temperature, stress, velocity, etc.), and the units (i.e. degrees C, MPa, m/s, etc.). This information is displayed along with the color coded matrix and the color key. It plays no important role in the program, but is displayed for aesthetic reasons only. Figure 9.2 displays the color coded data from program HEAT at the steady state.

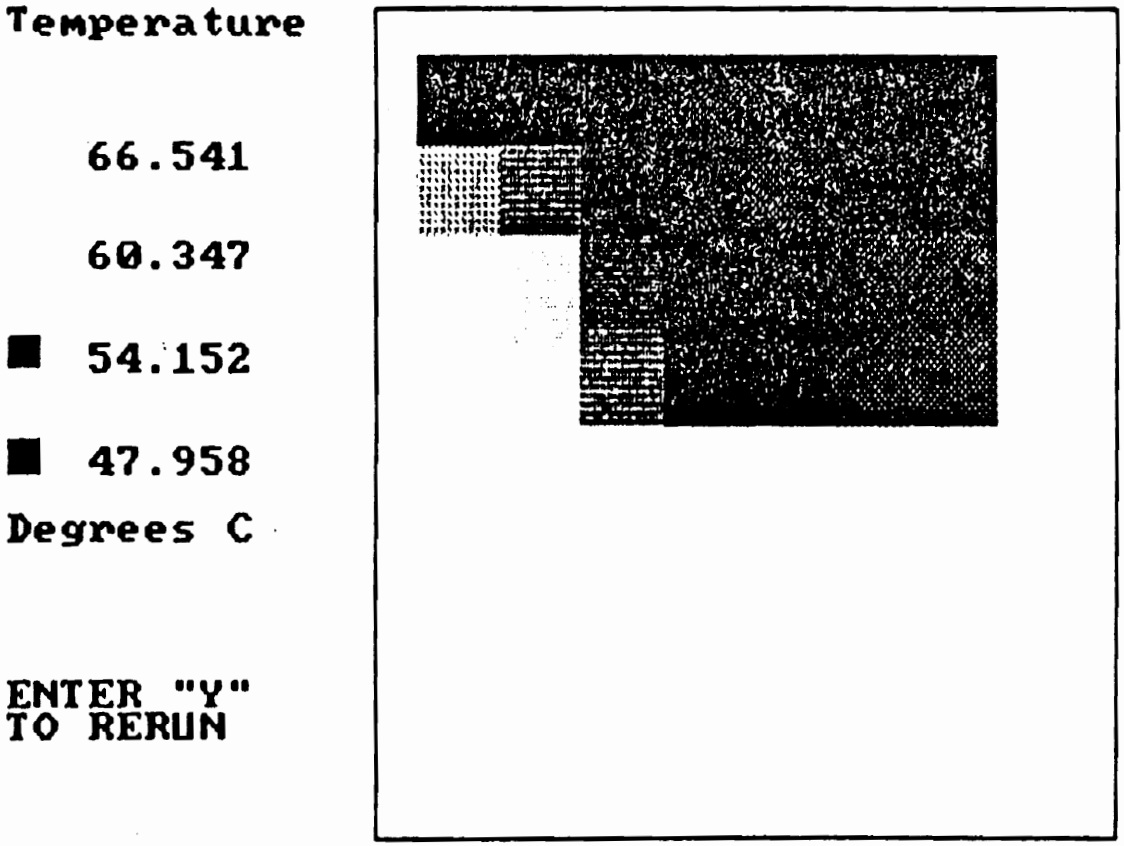


FIGURE 9.2: Color Coded Matrix of Data from Program HEAT

APPENDIX A: NUMERICAL METHODS

In this appendix, several numerical procedures used in the programs are reviewed. These brief explanations are taken from Applied Numerical Methods by Carnahan, Luther, and Wilkes [11]. A more detailed examination of these topics may be found in any advanced calculus text.

A.1 NEWTON ITERATION

This is an iterative procedure used to determine the root of an algebraic equation. Programs RANKINE, KELVIN, CYLINDER, KTFOIL, BLASIUS, WEDGE, MOODY, COMPRESS, OBLIQUE, PRANDTLM, and FREEJET all use this technique.

The function, f(x), describes some curve in the X, Y plane. It is assumed that at some point, R, the curve f(x) crosses the X axis. The root, R, may be determined by the following procedure. An initial guess is proposed as the root; call it x{i}. At this value, the function has a value f(x{i}) and a derivative f'(x{i}). If a line is drawn tangent to the curve at the point (x{i}, f(x{i})) it will intersect the X axis at some point x{i+1}. It is assumed that this intersection is closer to the root, R, than the initial guess, x{i}. Hence it becomes the new guess. See Figure A.1. The slope of the tangent is

$$f'(x\{i\}) = f(x\{i\})/[\, x\{i\} - x\{i+1\} \,] \qquad (A.1)$$

If this equation is rearranged, it becomes the definitive form of the Newton iterative procedure:

$$x\{i+1\} = x\{i\} - f(x\{i\})/f'(x\{i\}) \qquad (A.2)$$

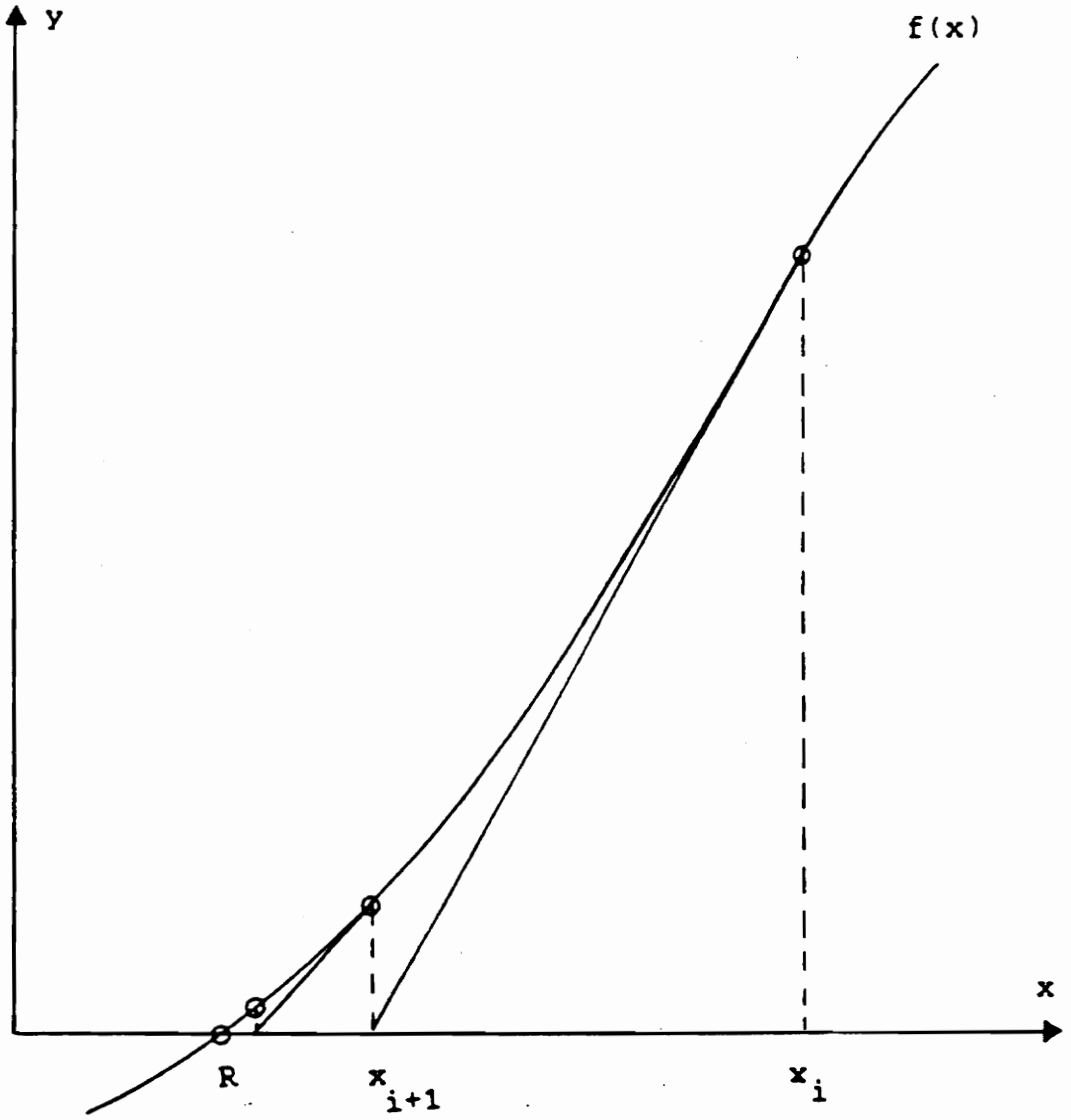


FIGURE A.1: Newton Iteration Procedure

Successive guesses are determined until the difference between x{i} and x{i+1} is less than some tolerance (0.000001 in the programs).

To employ this technique for a given algebraic relation, all terms must be taken to one side of the equation such that zero appears on the opposite side. Also, the derivative of that relation must be found with respect to the parameter whose root is desired. Then an iteration loop must be constructed within the program to determine successive guesses at the root. Finally a logical statement must be contained within the loop to provide an escape once the difference between successive values of x is below some arbitrarily prescribed tolerance.

A.2 SIMPSON'S 1/3 RULE

Simpson's 1/3 rule is used in program THWAITES to numerically integrate the von Karman integral expression. For a given function, f(x), which is to be integrated along some range of x, the area under the curve is divided into intervals spaced evenly at some distance, h. If one were to employ either the rectangle or trapezoid rule for integration, the local truncation error would be of the order h^3. However, using the Simpson's 1/3 rule reduces the truncation error to an order of h^5.

For a given span of 2h, three x coordinate values must be examined. These x values have three corresponding values of f(x). Simpson's rule curve fits a second order polynomial through these three points along the curve f(x). See Figure A.2. Integration of the polynomial results in the following relation

$$\int_{x_{i-2}}^{x_i} f(x) = \frac{h}{3}[\, f(x\{i\text{-}2\}) + 4\, f(x\{i\text{-}1\}) + f(x\{i\})\,] \qquad (A.3)$$

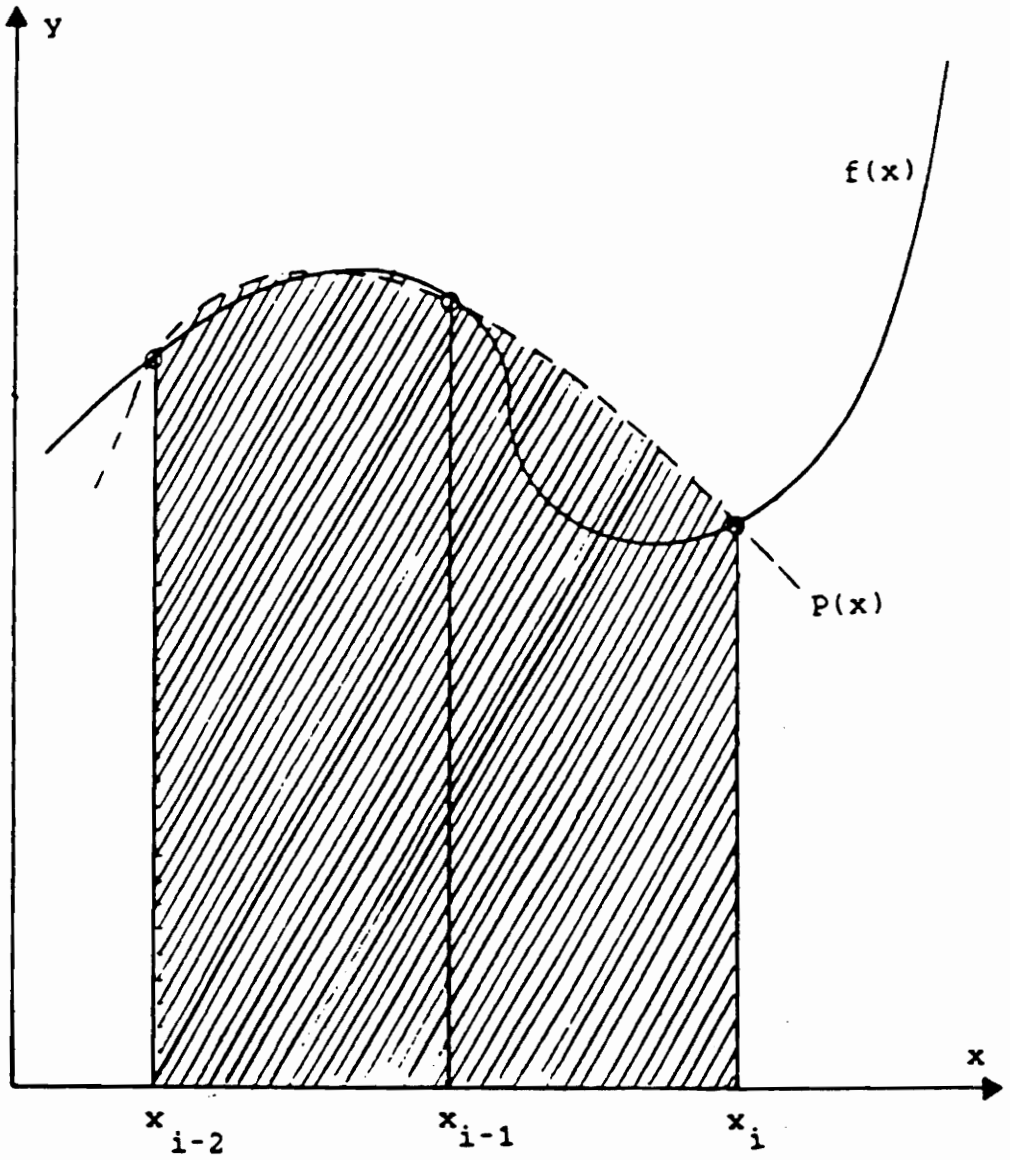


FIGURE A.2: Simpson's 1/3 Rule

Individual integrals over 2h spans may be added together in order to determine the complete integral over the entire range of x.

A.3 PREDICTOR-CORRECTOR METHOD

This method is used in program LINES. The predictor-corrector method is a simple technique used to solve the differential equation

$$dy/dx = f(x,y) \tag{A.4}$$

The procedure integrates the relation a distance, h, from x{i} to x{i+1}. The first step is called the predictor step used to estimate a value of y{i+1}. The Euler method predicts

$$y\{i+1\} = y\{i\} + f(x\{i\},y\{i\})\, h \tag{A.5}$$

This value of y{i+1} can be used to find an intermediate value of f(x,y) at h/2. This corrector step then leads to a better estimate of the value of y{i+1}. See Figure A.3.

$$y\{i+1\} = y\{i\} + \frac{h}{2} [\, f(x\{i\},y\{i\}) + f(x\{i+1\},y\{i+1\}) \,] \tag{A.6}$$

The truncation error of the procedure is of the order h^3.

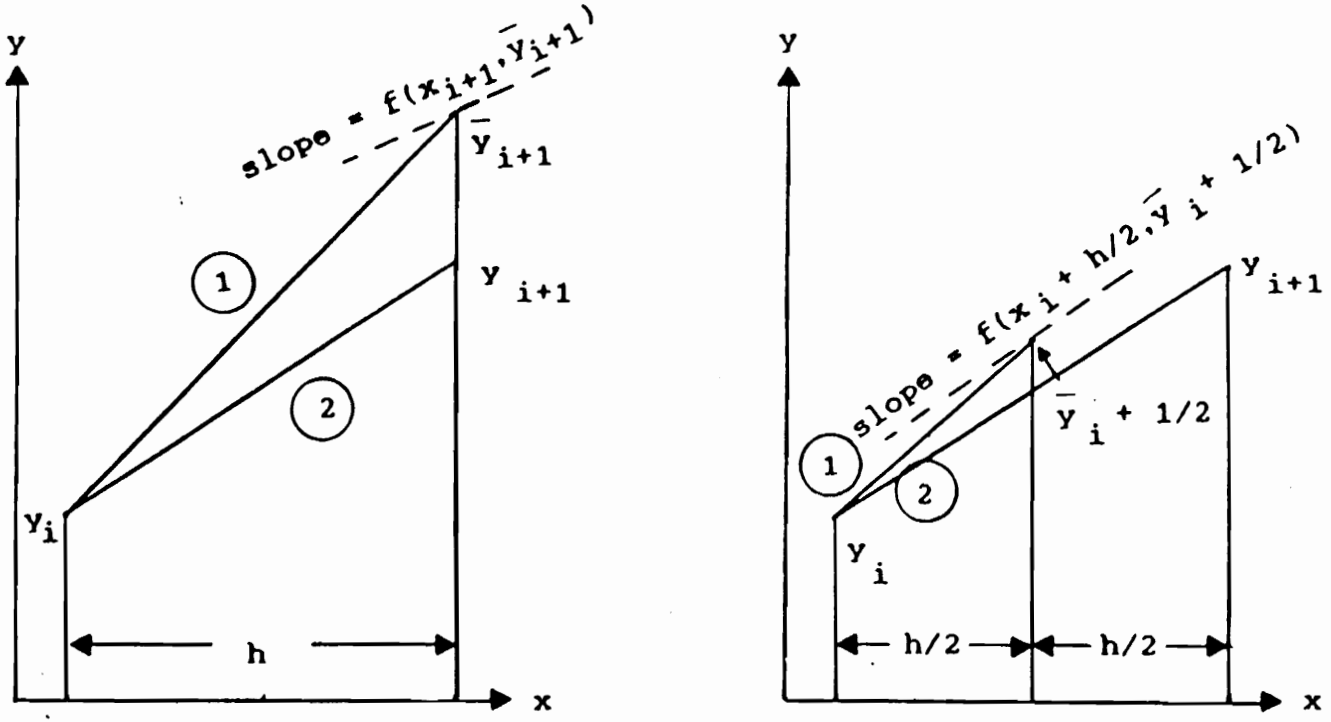


FIGURE A.3: Predictor-Corrector Method

A.4 FOURTH ORDER RUNGE-KUTTA

Programs BLASIUS and WEDGE use this technique of solving the differential equation dy/dx = f(x,y) which reduces the truncation error to the order of h^5. A Taylor expansion series of order h^4 is used to approximate the function, y(x), along the interval, h. This produces the following expression

$$y_{i+1} = y_i + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \tag{A.7}$$

where the K values are as follows

$$K_1 = f(x_i, y_i)$$

$$K_2 = f(x_i + 1/2\,h, y_i + 1/2\,h\,K_1)$$

$$K_3 = f(x_i + 1/2\,h, y_i + 1/2\,h\,K_2)$$

$$K_4 = f(x_i + h, y_i + h\,K_3) \tag{A.8}$$

It is interesting to note that if f(x,y) is only a function of x (f(x,y) = f(x)), then the fourth order Runge-Kutta technique reduces to Simpson's 1/3 rule.

Appendix B: Finite Difference Equations for Program HEAT

B.1 List of Finite Difference Equations for Program HEAT

Here, the finite difference equations used in program HEAT are presented in the order in which they occur in the program.

T' - Temperature at current time step

T - Temperature at previous time step

Ta - Air temperature

(n) - Node subscript

Fo - Fourier number = $\alpha \Delta t / \Delta x^2$

Bi - Biot number = $h \Delta x / K$

K - Substrate thermal conductivity

q - Generated heat

Node 1: Top Left

1: T'(1) = Fo[2BiTa+2T(2)+2T(8)]+[1-4FoBi]T(1)

Nodes 2-6: Top Interior

2: T'(2) = Fo[T(1)+T(3)+2T(9)+2BiTa)]+[1-4Fo-2BiFo] T(2)

3: T'(3) = Fo[T(2)+T(4)+2T(10)+2BiTa)]+[1-4Fo-2BiFo] T(3)

4: T'(4) = Fo[T(3)+T(5)+2T(11)+2BiTa)]+[1-4Fo-2BiFo] T(4)

5: T'(5) = Fo[T(4)+T(6)+2T(12)+2BiTa)]+[1-4Fo-2BiFo] T(5)

6: T'(6) = Fo[T(5)+T(7)+2T(13)+2BiTa)]+[1-4Fo-2BiFo] T(6)

Node 7: Top Right

7: T'(7) = Fo[2BITa+2T(6)+2T(14)]+[1-4Fo-2BiFo] T(7)

Node 8: Left Side

8: T'(8) = Fo*[2T(9)+T(1)+T(15)]+[1-4*Fo] T(8)

Nodes 9-13: Interior

9: T'(9) = Fo[T(8)+T(10)+T(16)+T(2)]+[1-4Fo] T(9)

10: T'(10) = Fo[T(9)+T(11)+T(17)+T(3)]+[1-4Fo] T(10)

11: T'(11) = Fo[T(10)+T(12)+T(18)+T(4)]+[1-4Fo] T(11)

12: T'(12) = Fo[T(11)+T(13)+T(19)+T(5)]+[1-4Fo] T(12)

13: T'(13) = Fo[T(12)+T(14)+T(20)+T(6)]+[1-4Fo] T(13)

Node 14: Right Side

14: T'(14) = Fo[2T(13)+T(7)+T(21)]+[1-4Fo] T(14)

Node 15: Left Side Heat Generating

15: T'(15) = Fo[T(16)+T(8)+T(22)]+[1-4Fo] T(15) +qFo/K

Nodes 16-20: Interior

16: $T'(16) = Fo[T(15)+T(17)+T(23)+T(9)]+[1-4Fo]\ T(16)$

17: $T'(17) = Fo[T(16)+T(18)+T(24)+T(10)]+[1-4Fo]\ T(17)$

18: $T'(18) = Fo[T(17)+T(19)+T(25)+T(11)]+[1-4Fo]\ T(18)$

19: $T'(19) = Fo[T(18)+T(20)+T(26)+T(12)]+[1-4Fo]\ T(19)$

20: $T'(20) = Fo[T(19)+T(21)+T(27)+T(13)]+[1-4Fo]\ T(20)$

Node 21: Right Side

21: $T'(21) = Fo[2T(20)+T(14)+T(28)]+[1-4Fo]\ T(21)$

Node 22: Left Corner

22: $T'(22) = Fo[T(23)+T(15)]+[1-4Fo]\ T(28)$

Nodes 23-27: Bottom Interior

23: $T'(23) = Fo[T(22)+T(24)+2T(16)]+[1-4Fo]\ T(23)$

24: $T'(24) = Fo[T(23)+T(25)+2T(17)]+[1-4Fo]\ T(24)$

25: $T'(25) = Fo[T(24)+T(26)+2T(18)]+[1-4Fo]\ T(25)$

26: $T'(26) = Fo[T(25)+T(27)+2T(19)]+[1-4Fo]\ T(26)$

27: $T'(27) = Fo[T(26)+T(28)+2T(20)]+[1-4Fo]\ T(27)$

Node 28: Right Corner

28: $T'(28) = Fo[T(27)+T(21)]-[1-4Fo]\ T(28)$

B.2 Sample Derivation of a Finite Difference Equation

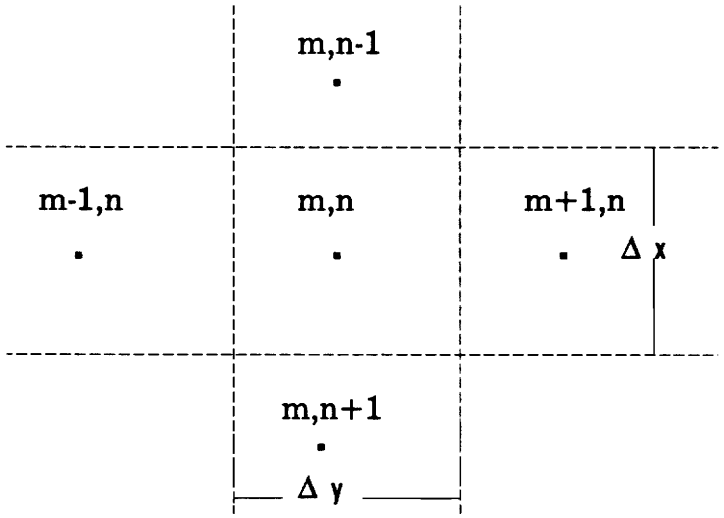


Figure B.1: Nodal Element for which a Finite Difference Equation is Derived

1. Add the temperature differences multiplied by the area separating the elements and the substrate thermal conductivity, and divided by the distance between the nodes. Set this result equal to the node temperature difference between time steps multiplied by the length and width of the element as well as the density and specific heat, and divided by the time step.

$$\Delta x K[T(m,n+1)-T(m,n)]/\Delta y + x K[T(m,n-1)-T(m,n)]/\Delta y +$$

$$\Delta x K[T(m+1,n)-T(m,n)]/\Delta y + xK[T(m-1,n)-T(m,n)]/\Delta y = c\Delta x\Delta y[T'(m,n)-T(m,n)]/\Delta t \quad \text{(B.1)}$$

2. Rearrange Equation (B.1). Note that $\Delta x = \Delta y$.

$$-4KT(m,n)+K[T(m,n+1)+T(m,n-1)+T(m+1,n)+T(m-1,n)] = \rho c\Delta x^2[T'(m,n)-T(m,n)]/\Delta t \quad \text{(B.2)}$$

3. Rearrange Equation (B.2) for T'(m,n).

$$T'(m,n) = [1-4k\Delta t/c\Delta x^2]T(m,n) + k\Delta t/c\Delta x^2\,[T(m,n+1)+T(m,n-1)+T(m+1,n)+T(m-1,n)] \quad \text{(B.3)}$$

4. Substitute in the definition of the Fourier number.

$$T'(m,n) = Fo[T(m,n+1)+T(m,n-1)+T(m+1,n)+T(m-1,n)+ [1-4Fo]T(m,n) \quad \text{(B.4)}$$

REFERENCES

1. Daniel B. Olfe, Fluid Mechanics Programs for the IBM PC, McGraw-Hill, New York, 1987.

2. John D. Anderson, Modern Compressible Flow with Historical Perspective, 2nd ed., New York, 1990.

3. Irving H. Shames, Mechanics of Fluids, 2nd ed., McGraw-Hill, New York, 1982.

4. Frank M. White, Fluid Mechanics, 2nd ed., McGraw-Hill, New York, 1986.

5. R. D. Patel, "Non-Newtonian Flows," in Handbook of Fluids in Motion, N. P. Cheremisinoff and R. Gupta (eds), Ann Arbor Science Publishers, Ann Arbor, Chap. 6., 1983.

6. Frank M. White, Viscous Fluid Flow, 2nd ed., McGraw-Hill, New York, 1991.

7. L. M. Milne-Thompson, Theoretical Aerodynamics, 5th ed., Macmillan, London, 1968.

8. L. Prandtl, and O. G. Tietjens, Applied Hydro- and Aerodynamics, McGraw-Hill, New York, 1934.

9. C. F. Colebrook, "Turbulent Flow in Pipes, with Particular Reference to the Transition between the Smooth and Rough Pipe Laws", **J. Inst. Civ. Eng. Lond.**, vol. 11, pp. 133-156, 1938-1939.

10. L. F. Moody, "Friction Factors for Pipe Flows", **ASME Trans.**, vol. 66, pp.671-684, 1944.

11. Brice Carnahan, H. A. Luther, and James O. Wilkes, Applied Numerical Methods, New York, 1969.

VITA

Jeffrey W. Santavicca was born on February 27, 1969 in Washington DC. He spent the first eighteen years of his life in Laurel, MD. At the age of 13, he taught himself BASIC on a 4K Texas Instruments PC. While attending Hammond High School in Columbia, MD, Jeff immersed himself in a technical curriculum. Here, he learned PASCAL and FORTRAN, and, after suffering a concussion, decided to become a Chemical Engineer.

Jeff elected to pursue his Bachelor's of Science at Virginia Tech due to its being, "just far enough away from home." Jeff quickly learned that Chemical Engineering was just not the cool thing to do, and enrolled in Mechanical Engineering. Still unable to change the oil in his car, Jeff avoided a curriculum involving machine design. He found his calling in Fluid Mechanics and Thermodynamics.

After obtaining his B.S. in August of 1991, Jeff took a two week sabbatical to Europe. Upon returning with his new found sophistication, he began working towards his M.S. under the instruction of Dr. F. J. Pierce. Remaining at Virginia Tech, because beer is cheap in Blacksburg, Jeff redeveloped his interest in computers after enrolling in a CAD course and while working on his thesis.

Upon completion of his Master's degree, Jeff plans to return to Maryland and repay his parents for their support and money by performing endless hours of yard work. After completing his serfdom, he hopes to find employment at an, as yet, undetermined location. He has been known to say, "Guam sounds nice."