Supersonic Civil Airplane Study and Design: Performance and Sonic Boom

Samson Cheung

January 1995
NCC2-617
Supersonic Civil Airplane Study and Design: Performance and Sonic Boom

Samson Cheung

This final report summarizes the work performed from July 1989 to Jan. 1995. The work is supported by NASA Co-operative Agreement NCC2-617. This report consists of four parts. The first part is the introduction of the research effort. The second part describes the work and results from July 1989 to June 1993. The third part describes the work and results from July 1993 to January 1995. A summary is given at the end of this report.

1 INTRODUCTION

The present supersonic civil airplane, the Concorde, is a technological break-through in aviation history. However, it is an economical disaster for two main reasons. The first is her low aerodynamic performance, that allows only 100 passengers to be carried for a short-range flight with expansive airfare. Another reason is that the shock waves, generated at supersonic cruise, coalesce and form a classical N-wave on the ground, forming a double bang noise termed sonic boom, which is environmentally unacceptable. To enhance the U.S. market share in supersonic civil transport, an airframer’s market risk for a low-boom airplane has to be reduced.

Since aircraft configuration plays an important role on aerodynamic performance and sonic boom shape, the configuration of the next generation supersonic civil transport has to be tailored to meet high aerodynamic performance and low sonic boom requirements. Computational fluid dynamics (CFD) can used to design airplanes to meet these dual objectives. The work and results in this report are used to support NASA’s High Speed Research Program (HSRP).

In this five years of study and research, CFD tools and techniques have been developed for general usages of sonic boom propagation study and aerodynamic design. In the beginning of the 90’s, sonic boom extrapolation technique was still relied on the linear theory developed in the 60’s for the nonlinear techniques were computationally expensive. A fast and accurate sonic boom extrapolation methodology (Section 3.2), solving the Euler equations for axisymmetric flow, has brought the sonic boom extrapolation technique up to the 90’s standard.

Parallel to the research effort on sonic boom extrapolation, CFD flow solvers have been coupled with a numeric optimization tool to form a design package for aircraft configura-
tion. This CFD optimization package has been applied to configuration design on a low-boom concept (Section 2.3) and an Oblique All-Wing concept (Section 2.4) prior to the wind-tunnel models are built and tested at Ames. The tunnel test results have validated the CFD technique and design tools.

Moving to the world of parallel computing, the aerospace industry needs a numeric optimization tool suitable for parallel computers. A nonlinear unconstrained optimizer for Parallel Virtual Machine has been developed for aerodynamic design and study. Study in Section 3.3 demonstrates the capability of this optimizer on aerodynamic design.

2 PREVIOUS WORK/RESULTS

The work and results described in this section was begun in July 1989. The first project was to use CFD tools and existing linear theory to predict waveform signatures at some distances from flight vehicles. The aim of this study was to demonstrate and develop the technique of sonic boom prediction by CFD. The next step was to apply this developed technique to low-boom configurations.

The second project, which was the continuation of the first one, was to develop a CFD optimization package for design process on meeting the dual objectives of high aerodynamic performance and low sonic boom loudness. This optimization package was applied to three different High Speed Civil Transport (HSCT) baseline configurations and a generic body of revolution.

A wind-tunnel model (Ames Model 3) was built based on one of the modified HSCT baseline configuration. This model was tested in June 1993. The test results were used to validate the design method. Publication of the result was limited due to the sensitive nature of the project.

A counterpart of the conventional HSCT concept was the Oblique All-Wing (OAW) concept. CFD computational supports, as well as optimization calculations, were provided to the OAW design team consisting personnels from NASA Ames Research Center, industry, and university. The aim of the project was to design a realistic configuration for wind-tunnel test. The model was built and tested at Ames in June 1994.

2.1 Sonic Boom Prediction Technique

In the early stage of sonic boom prediction activity, two major things were involved. The very first thing was to identity the capability of CFD in sonic boom prediction. The second thing was to apply these CFD tools to predict sonic boom signals of varies configurations after necessary code modification, grid refinement study, and comparison with supersonic linear theory.
2.1.1 Method Validation

A three-dimensional parabolized Navier-Stokes code, UPS3D, developed at Ames was used as the flow-solver. It is a space-marching code with finite-volume approach. The near field solution of a simple wing/body configuration was calculated by UPS3D, and the overpressure signal at some desired distances were obtained either by the axisymmetric option of UPS3D or a quasi-linear extrapolation code, based on Whitham's F-function theory. Later I realized that using Lighthill integral to calculate the F-function for non-axisymmetric aircraft was more accurate, I wrote a Fortran code, LHF, for sonic boom prediction based on Lighthill integral. This code is available from Ames Software Library. A copy of LHF is attached in Appendix A. The figure below is a brief summary of the sonic boom extrapolation process.

A series of studies on grid refinement, including solution adaptive grid, and on sensitivity of initial distance of extrapolation were conducted. It was found that viscous calculation was unnecessary for sonic boom prediction. However, the grid must be sufficiently fine in the regions of shock and expansion waves. In order to capture all the nonlinear effects in a three-dimensional flow, the near-field overpressure should be captured at about one span length below the flight track before extrapolating to the far field. The detail results were published in AIAA Journal of Aircraft and NASA Technical note.

In summary, the tools for sonic boom prediction had been identify and validated in the above study. The combination of CFD and Whitham’s method gave a relative efficient tools for sonic boom prediction. Nevertheless, the CFD codes was still computationally expensive for design optimization runs.
2.1.2 Boom Prediction for Low-Boom Configurations

With the experience on grid refinement study and the extrapolation procedure, the prediction tools were being used to predict the sonic boom of two low-boom configurations designed by Boeing aircraft company and Langley research center.

Each of the two configurations consisted of two separated parts, namely, the wing and the fuselage. The wing was defined by data in spanwise cuts, whereas the fuselage was defined by data in streamwise cuts. In order to create a single wing-fuselage surface grid for UPS3D code, a grid generator (SAMGRID) was written to defined the wing in streamwise cuts and aggregated the wing to the fuselage. Computation results of the two configurations are shown below.

The sonic boom signals calculated from the CFD prediction tools were compared to the wind-tunnel data of the Langley's configuration. The computational results of the Boeing's configuration was used to validate the linear design method used by Boeing.

2.2 Supersonic Airplane Design

The need for simultaneous sonic boom and aerodynamic optimization was highlighted when it became clear that designed to a strict sonic boom constraint suffered an unacceptable performance penalty. Therefore, low-boom design studies must carefully balance the trade-off between sonic boom loudness and aerodynamic performance. A CFD optimization package was developed to demonstrate the methodology for the optimization of supersonic airplane designs to meet the dual objectives of low sonic boom and high aerodynamic performance.

In this project, an optimizer with linear and nonlinear constraints was first identified, and then an efficient CFD flow solver was chosen. This CFD code had to be sufficiently fast because more than 90% of the computational time were used in CFD calculations. Before this optimization was used to design low-boom wind-tunnel model (Section 2.3), it was tested and exercised by improving aerodynamic performance of a low-boom wing/body configuration and a body of revolution.
2.2.1 CFD Optimization Package

Several computational tools interconnect in the optimization procedure are listed below:

- UPS3D: 3-D parabolized Navier-Stokes code; inviscid calculation only (Ref. 1)
- NPSOL: numerical optimization code; a sequential quadratic programming algorithm in which the search direction is the solution of a quadratic programming subproblem
- HYDPGEN: hyperbolic grid generator; a sufficiently fast and robust to operate within an automated optimization environment.
- LHF: sonic boom extrapolation code (Appendix A); a routine based on Whitham's F-function and the equal-area rule
- SAMGRID: wing/body surface grid generator (Appendix B); a sufficiently fast and robust to operate within an automated optimization environment
- DB: sonic boom loudness calculation; a code gives perceived loudness (PLdB) of the sonic boom can be determined by Stevens' Mark VII method which involves Fast Fourier Transform on the energy spectrum of the sonic boom

This CFD optimization package is robust and efficient on Cray-YMP. The application of this package will be described in the following sections.

2.2.2 Aerodynamic and Sonic Boom Optimization

The optimization design package was exercised using a recently-developed low-boom wing-body configuration, Boeing 1080-991 (also called Haglund model), designed by George Haglund. This optimization technique was applied separately to the two objectives of high aerodynamic performance and low sonic-boom loudness.

For aerodynamic enhancement, control points are set on the cambers of the wing, with the thickness kept fixed. The left figure below shows the differences on a inboard airfoil section of the original and the modified. The polar plot shows the improvement of L/D of the modified configuration over the original by 3.8%. The right figure below shows that the modified wing had less wave drag than the original one at the leading edge. This means...
that the leading thrust is improved by the optimization process. The whole process takes about 4 CPU hours on Cray-YMP.

For sonic boom improvement, F-function was employed as an entity to define the equivalent area distribution and sonic boom shape. The original Haglund model was supposed to give a flat-top pressure waveform at the ground. However, calculations showed that the waveform had an intermediate shock followed right after the bow shock; whereas the flat-top waveform would have no intermediate shock. The design code redistributed the equivalent area of the fuselage (without changing the wings), and re-captured the flat-top characteristic of the pressure waveform. The figure below compares the sonic boom signatures among the original, optimized, and target flat-top. Due to the sensitive nature of the configuration, the change of the configuration will not be shown here. The details of this optimization methodology and results were considered as sensitive materials and were presented in the 2nd Annual Sonic Boom Workshop.10

2.2.3 Drag Minimization on Haack-Adams Body

The purpose of this study was threefold:
• to search for a design method to minimize the drag of a supersonic projectile
• to demonstrate the capability of the CFD optimization package described above
• to search for computational grid density effect on optimization performance

The baseline configuration chosen for this study was called Haack-Adams body11, a body of revolution with a pointed nose and a base of finite area. This body was thought to be the minimum-drag body under the slender body theory. Wind-tunnel data were available for CFD validation. The method of optimization made use of the Fourier Sine expansion, which had three main advantages over the traditional techniques based on shape functions and control points:

• The volume of the body was fixed without putting external constraints. External constraints cost more computational time. For some cases, fixed volume is not feasible.
• Global minimum was search.
• Number of design variables was substantially reduced.
The figure below summarizes the result of this study. The nose of the body was trimmed to reduce the wave drag. Since the total volume was constrained, volume was added near the end of body. Total wave drag reduction was by 6%. The results were presented in a AIAA meeting\textsuperscript{12} and published in Journal of Aircraft Vol. 32, No. 1, Jan/Feb. 1995.

\subsection*{2.3 Low-Boom Wind-Tunnel Configuration (Ames Model 3)}

Efforts were made to design a new wing/body/nacelle configuration, which had a lower sonic boom relative to the baseline, 1080-911 from Boeing Company, of low boom HSCT concept. The CFD optimization package described in Section 2.2.1 were employed to modify this baseline configuration. The result of the optimization was used to build a wind-tunnel model, Ames Model 3, tested at Ames 9'x7' wind tunnel in June 1993. Due to the sensitive nature of the configuration, no planform shapes will be shown here. However, the left and right figures below show the computational grid and the optimization result, respectively. The plot at the lower right-hand corner of the right figure shows the sonic booms of the baseline and Model 3 respectively. The baseline configuration has a loudness level about 100 PLdB; whereas Model 3 has about 92 PLdB. The results of this research were presented in the 3rd Annual Sonic Boom Workshop.\textsuperscript{13}
2.4 Oblique-All Wing (OAW) Computation and Design

Oblique flying-wing\textsuperscript{14} is an alternative supersonic aircraft concept. Ames, Boeing, Douglas, and Stanford University joined and formed a design team in 1992 to investigate the feasibility of OAW for commercial use. The study included aerodynamic performance, stability, structure, landing gear, airplane exits, and airport regulations. The design team decided to build a wind-tunnel model for wind-tunnel testing in June 1994. My job was to provide Navier-Stokes CFD supports and, if possible, optimization results. The figure below shows some of the wings that were analyzed since the beginning of this study.

The flow solver being used was Overflow code, a 3-D Navier-Stokes code using the diagonal with ARC3D algorithm\textsuperscript{15}. One of the most challenging works of this project was to reduce the separation on the left wing (trailing wing). The separation on the upper surface of the wing and the corresponding vortices are shown in the left side of the figure below. It was found that bending of the wing could abate the separation, as well as improve the lift-to-drag ratio. The right side of the figure shows a weaker separation pattern on the ended
wing. Due to the sensitive nature of this study, the results can only be presented in the weekly group meetings at Ames and a controlled distributed NASA Contractor Report.

3 CURRENT WORK/RESULTS

Currently, research effort was concentrated on one theme that is sharpening the tools for HSCT design. Three research topics are focused: near-field CFD calculation and sonic boom softening of Boeing Reference-H, improvement of sonic boom extrapolation, and aerodynamic design on parallel computer.

In order to study and design a real complex aircraft, a relatively fast CFD technique has to be developed for optimization environment. Coupling a fast space-marching code and a time iterative code with overset grid concept can take the advantage of marching code at the fuselage/wing region and solve the complex flow field near the wing/nacelle region at the same time.

A very efficient wave propagation code for mid-field sonic boom prediction has been developed based on the method of characteristics. This code solves the Euler equations for 1.2 minutes on Cray-YMP; whereas, the axisymmetric CFD method described in Section 2.1.1 takes 40 minutes on the same computer.

Number crunching problems, like CFD calculations, on parallel machines can be efficiently done in today’s computing environment. This may lead to the future of aerodynamic research and design. In order to exercise HSCT design on parallel computers, a nonlinear optimization routine has been developed for a network based parallel computer system in which a cluster of engineering workstations serves as a virtual parallel machine.

3.1 Sonic Boom and Performance Study of Reference-H

Research effort on low-boom configuration concept has been invested for the past four years. A new proposed route structure for HSCT’s incorporating supersonic corridors over land and water has relaxed the sonic boom constraint somewhat. The objective of this study is twofold. First is to exercise the methodology of combining two different CFD codes to solve the near-field solution of a realistic HSCT configuration in an efficient and accurate manner. Second is to reduce the sonic boom loudness of a performance configuration concept, Reference-H, without jeopardizing the aerodynamic performance. The basic components of Reference-H are a fuselage, a pair of swept wings, and four nacelles.

3.1.1 Reference-H Near-Field Study

The CFD codes used in this study are the UPS3D code and the OVERFLOW code. Both CFD codes has been described in Section 2.1.1 and 2.4, respectively. The former is an efficient space-marching code. However, it fails in the region where subsonic pocket exists; especially in the region of the wing/nacelle integration. The latter is a time-iterative code with Chimera overset grid concept, which makes the code more viable in solving the
region of wing/nacelle integration. In this study, only inviscid flow is considered. Figure below summarizes the result of the CFD calculations.

The near-field solution is studied for the case of Mach number 2.4 and angle of attack 4.5 degrees. Wind-tunnel data of the Reference-H validate the CFD method. Study shows that flow particles turn significantly over the outer nacelle compared with the inner nacelle. It indicates that the effect of the nacelle orientation might improve the aerodynamic performance.

3.1.2 Sonic Boom Softening

The sonic boom of the Reference-H configuration is also obtained. The calculation shows that the boom is an N-wave of 104 PLdB with 2.5 psf. bow shock on the ground. Details of the sonic boom prediction technique can be found in Ref. 10. Boom modification for performance aircraft is very much different from the low-boom aircraft for cruise Mach number and lift are higher. Therefore, the technique developed previously can not be strictly applied to Reference-H. However, changing the equivalent area can be helpful. The result of this study was presented in the 4th Sonic Boom Workshop. Another approach to reduce the boom is by experimenting the sweep angle. The figure above show one of the exercises done on the Ref-H. This exercise successfully shows Boeing how much boom

CURRENT WORK/RESULTS February 14, 1995
reduction can be achieved by redistributing the lift. An closer on-going technology communication with airframe industry is needed in order to achieve the goal of sonic boom softening on performance aircraft. A team consisting myself and other personnells from Boeing and NASA Langley has been formed to achieve the goal.

3.2 Sonic Boom Mid-Field Extrapolation (WPSYM)

In the beginning of 90’s, sonic boom extrapolation technique was still relied on the linear theory developed in the 60’s for the nonlinear techniques were computationally expensive. Today, a fast and accurate sonic boom extrapolation methodology is needed to bring the sonic boom extrapolation technique up to the 90’s standard for HSCT design. The objective of this study is to develop an efficient and accurate higher-order computational method, solving the Euler equations, for supersonic aero-acoustic wave propagation.

An axisymmetric wave propagation code (WPSYM) has been developed for mid-field sonic boom extrapolation. This propagation code has been demonstrated as an efficient and accurate tool over the previous CFD method, described in Section 2.1.1 and Ref. 4, on a generic wing-body configuration. The figure below shows that a 3-D near-field solution is obtained from UPS3D code; the result is then interfaced to two axisymmetric sonic boom extrapolation codes, namely, the axisymmetric version of UPS3D and the recent wave propagation code (WPSYM). The former takes 40 minutes on Cray-YMP, and the latter takes 1.2 minutes on the same machine. The x-y plot in the figure compares the numerical extrapolation results to wind-tunnel data. The result has been shown in NASA Technical Highlight and the methodology has been presented in the 4th Annual Sonic Boom Workshop at NASA Langley in June 1994.16
3.3 Optimizer on PVM (IOWA)

Moving to the world of parallel computing, the aerospace industry needs a numeric optimization tool in the parallel environment. One of the promising parallel computing concepts is the network-based distributed computing. The Parallel Virtual Machine (PVM) is a software package that allows a heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource. PVM allows users to link up engineering workstations to work as a single distributed-memory (parallel) machine. Merritt Smith and I wrote a manual on PVM for beginning users. A copy of the manual is attached in Appendix C.

A parallel optimizer based on nonlinear Quasi-Newton method has been developed and coupled with an efficient CFD code for basic aerodynamic design and study. This optimizer is called IOWA (parallel Optimizer With Aerodynamics). The figure below is a demonstration of IOWA. A Boeing arrow wing/body configuration is chosen in this study. The fuselage radius is changed so that the wave drag is minimized. The parallel CFD optimization process takes 24 wall-clock hours on 4 SGI workstations to reduce the wave drag by 6.5%. The optimized result is a "coke bottle" shape fuselage, as expected by supersonic area rule. The convergence history of the optimization process is also shown in the figure. The optimizer is also coupled with a parallel CFD code, MEDUSA, to perform viscous 2-D multizone airfoil optimization supported by overset grid concept. The results will be presented at NASA CAS conference in March 1995.

3.4 Oblique All-Wing (OAW): CFD support

The OAW design team has asked for CFD support on the latest configuration OAW-3 from which a wind-tunnel model has been built and tested at Ames in June 1994. The figure below shows the chimera grid topology on the OAW-3 with fin. The design team want to compare the CFD result with the result from pressure sensitive paint (PSP). Therefore,
CFD calculations have to be done prior to the wind-tunnel test because color map from CFD result is need for PSP calibration.

4 SUMMARY

The computational tools for sonic boom prediction, aerodynamic calculation, and configuration design of the current HSCT concept have been validated and applied to build wind-tunnel model for further testing and validation. The techniques developed in this five-year research and their applications, such as sonic boom prediction technique (Section 2.1), design of Ames Model 3 (Section 2.3) by CFD optimization (Section 2.2), and sonic boom softening for performance configuration (Section 3.1), have clearly shown support to the HSRP as it moved to its phase two period.

An accurate sonic boom extrapolation tool has always been an issue. It is because the flow phenomena in the atmosphere are nonlinear, but the common technique for extrapolation is linear acoustic theory developed in the 60’s. On the other hand, CFD technique is too computationally expensive. Recently, a fast and accurate sonic boom extrapolation methodology (Section 3.2), solving the Euler equations for axisymmetric flow, has brought the sonic boom extrapolation technique up to the 90’s standard.

Parallel computing is a fast growing subject in the field of computer science because of the promising speed in number crunching computations. A new optimizer (Section 3.3) for parallel computing concept has been developed and tested for aerodynamic drag minimization. This optimimizer is also coupled with a parallel CFD code so the whole optimization process is parallel. This is a promising method for CFD optimization making use of the computational resources of workstations, which unlike supercomputers spend most of their time idle.

Finally, the OAW concept is so attractive because of its overall performance in theory. In order to fully understand the concept, a wind-tunnel model is built. CFD Navier-Stokes calculations helps to identify the problem of the flow separation (Section 2.4), and also help to design the wing deflection for roll trim and alleviating the flow separation.
5 References


Appendix A

LHF (Fortran Listing)
PROGRAM LHF

This program calculates
1) the Light\-bull F-Function on body surface.
- \( a \) Input data
- \( b \) Define parameters

2) the overpressue signature at given distance \( R_i \).
3) the loadless level of the sonic boom at \( R_i \).

INPUT
- \( \text{LHF.in}(3) \): Input parameter
- \( \text{areas.in}(3) \): Equivalent area distribution, \( (\text{INAREA} = 0) \)
- \( \text{R}(\text{area}) \): F-Function distribution, \( (\text{INAREA} = 2) \)
- \( \text{coef.dat}(11) \): Coefficients due to \( \text{LHF.in}(3) \).
- \( \text{sio.dat}(3) \): Surface grid \( - \text{GRID} \) planer format, \( (\text{INAREA} = 1) \)

Default case \( \text{Wing-body (INAREA=1)} \)

OUTPUT
- \( \text{areas.out}(3) \): Equivalent area distribution and its derivative.
- \( \text{out}(11) \): F-Functions on the body surface and at distance \( R_i \).
- \( \text{signature.out}(13) \): Pressure signature at distance \( R_i \).
- \( \text{curve.out}(14) \): Integral curve of the shifted F-function.

By: Samson Cheung

NAC Institute
NASA Ames Research Center
M/S 258-D1
 Moffett Field, CA, 94035
Date: 1992/3/4
Version 2.1

PARAMETER (INMAX-220, IMAX-1, JMAX-351, HMAX-303)

DIN:
- \( E \) (IMAX, IMAX, JMAX, KMAX, LMAX, JMAX, JMAX, KMAX)
- \( \text{REAL}(\text{INMAX}) \), \( \text{S} \)(IMAX), \text{SP}{(\text{INMAX})}, \text{TAV}{(\text{INMAX})}, \text{PTAV}{(\text{INMAX})}

LOGICAL: \( \text{MACH} = \text{FALSE} \)

OPEN(UNIT=1), File='LHF.in', Status='OLD'

Read the input parameters
- \( \text{PANELIST} = \{ \text{PARA, \text{PRAC, PFAC, RO}, R1, \text{INAREA, LIFT, BODY} \}

READ(1, PARA)

WRITE(1, PARA)

Input free-stream Mach number = \( \text{PRAC} \)
If \( \text{PRAC} < 0 \), sonic boom versus time, else versus distance

Is the surface grid contains the whole configuration, or only half-plane or only quarter-plane?
- \( \text{PFAC} = 1 \) Whole-configuration
- \( \text{PFAC} = 3 \) Half-plane
- \( \text{PFAC} = 0 \) Quarter-plane

\( R_i \) will be the distance where the signature is captured.

If \( \text{read in area distribution} \), \( \text{INAREA} = 0 \)

Read the grid
- \( \text{INAREA} = 1 \)

The wing-body case
- \( \text{INAREA} = 2 \)

Read the F-function
- \( \text{INAREA} = 3 \)

Read a signature at \( R_i \)
- \( \text{RO} \) > 0

\( \text{PI} = 4\times\text{ATAN}(1) \)

\( \text{JDEL} = \text{JMAX} \)

\( \text{JWIN} = \text{JMAX} \)

If \( \text{RO} = 0 \), we read the pressure signature at \( R_i \) and extrapolate

If \( \text{RO} < 0 \), GOTO 790

Find the area distribution of body configuration (sample case).
- \( \text{IF} \text{INAREA} \text{LT} 0 \text{ THEN} \)
- \( \text{IF} \text{INAREA} \text{EQ} 0 \text{ THEN} \)

\( \text{CLOSE}(2) \)

READ(2, , END=75) \( \text{TAV}(2), S(2) \)

\( S(2) = \text{S} \times \text{PFAC} \)

CONTINUE

75 \text{CONTINUE}

CLOSE(2)

GOTO 790

\text{ENDIF}

Read in the given area distribution
- \( \text{IF} \text{INAREA} \text{GT} 0 \text{ THEN} \)

OPEN(UNIT=2, File='area.in')

DO \( 50 J = 1, \text{JMAX} \)

READ(2, , END=75) \( \text{TAV}(2), S(2) \)

CONTINUE

50 \text{CONTINUE}

CLOSE(2)

GOTO 790

\text{ENDIF}

Read in the F-function or define a F-function by calling \text{FDFAC}

\text{IF} \text{INAREA} \text{EQ} 2 \text{ THEN} \)

\text{CALL} \text{FUNC} \{ \text{TAV}, \text{PTAV}, \text{JDEL} \}

\text{OPEN(UNIT=4, File='f.dat')}

\text{DO} \( 100 J = 1, \text{JMAX} \)

\text{READ}(4, , END=100) \( \text{TAV}(2), \text{S}(2) \)

100 \text{CONTINUE}

\text{ENDIF}

Read in the F-function or define a F-function by calling \text{FDFAC}

\text{IF} \text{INAREA} \text{EQ} 2 \text{ THEN} \)

\text{CALL} \text{FUNC} \{ \text{TAV}, \text{PTAV}, \text{JDEL} \}

\text{OPEN(UNIT=4, File='f.dat')}

\text{DO} \( 100 J = 1, \text{JMAX} \)

\text{READ}(4, , END=100) \( \text{TAV}(2), \text{S}(2) \)

100 \text{CONTINUE}

\text{ENDIF}
CALL DIFARC(TAU,FTAU,J-1,TAU,FTAU,JDIM,10.0)
CALL DAREA(S,FTAU,TAU,JDIM)
GOTO 270
ENDIF
READ(11) EDIM, JDIM
DO 100 J-1,JDIM
READ(11) (X(X,L,J), Y(X,L,J), Z(X,L,J), I-1,EDIM), (L,EDIM),
4 (I,EDIM), (L,EDIM)
4 100 CONTINUE
CLOSE(11)
CALL SAREA(EDIM, JDIM, JDIM, X, Y, Z, XMAX, YMAX, ZMAX)
DO 220 J-1, JDIM
S(J) = FFAC'S(J)
T(A-1) = X(1,1,J)
220 CONTINUE
CALL DIFARC(TAU,FTAU,J-1,TAU,FTAU,JDIM,10.0)
DO 300 J-1, JDIM
A(J) = TAU(J)*S(J)
300 CONTINUE
OPEN(UNIT=12, FILE='area.out')
WRITE(12,400) IF(TAUSER IFFORMED A AREA DISTRIBUTION)
DO 450 J-1, JDIM
WRITE(12,580) TAU(J), S(J)
450 CONTINUE
WRITE(12,550) TAU(J), S(J)
WRITE(12,550) TAU(J), S(J)
CLOSE(12)
CALL FLN(TAU, FMACH, JDIM, RO, TOLX)
CLOSE(3)
STOP
END
SUBROUTINE LIGHTI(TAU,R,SP,N,FTAU)
DIMENSION R(N),SP(N),TAU(N),FTAU(N)

C
P= 4.*ATAN(1.)
BETA=SQRT(PNACH**2.-1.0)
TAU(1)=0.
FTAU(1)=0.
C
DO 95 N=1,N
FTAU(N)=0.
C
DO 100 J=1,N
DO 102 I=2,N
IF(ABS(R(I))LE.1.E-10) THEN
Z1 = 1.E-10
F4 = 0.
GOTO 99
ENDIF
AB=2.0./BETA*R(I))
ABI=ABS(AB)
F1=SQRT(ABI)
F2=SP(I)-SP(I-1)
F3=F1+F2
F4=F3/(2.0+f1)
Z1=TAU(J)-TAU(I))/(BETA*R(I))
XLO=1.0
IF (Z1.LT.XLO) GO TO 96
IF (Z1.LT.XLO) GO TO 97
IF (Z1.LT.XLO) GO TO 98
96 RZI=0.
FTAU(J)=FTAU(J)+RZI+F4
GO TO 99
97 RZI=.23937-.11*11.-.2175+.7531
FTAU(J)=FTAU(J)+RZI+F4
GO TO 99
98 BB=2.0/(2.0+f1)
BBI=ABS(BB)
BZI=SQRT(BB)
FTAU(J)=FTAU(J)+BZI+F4
C
99 CONTINUE
100 CONTINUE
RETURN
END
SUBROUTINE EQUAPEA(EDIM, LDIM, JDIM, X, Y, Z, EMX, LMAX, JMAX, E)

This subroutine finds the cross-section area of a surface grid
which has symmetry plane at Y-axis. For each marching station
(for each x) the area is found. The area is approximated by
triangular rule.

DIMENSION X(EMX, LMAX, JMAX), Y(EMX, LMAX, JMAX), Z(EMX, LMAX, JMAX)

REAL S(LDIM)

C DO 10 J = 1, JDIM
   C DAREA = 0.
   C  DO 5 KL = 2, LDIM +1
   C   K = Y(KL, JDIM, J) - Y(KL-1, LDIM, J)
   C   ADD = Z(KL, LDIM, J) * S(KL+1, LDIM, J)
   C   DAREA = DAREA + 0.5*KL*ADD
   C CONTINUE
   C
   C The unwated base area:
   C
   ER = Y(1, LDIM, J) - Y(EDIM, LDIM, J)
   C ADD = Z(1, LDIM, J) * S(EDIM, LDIM, J)
   C BASE = ABS(0.5*ADD*ER)
   C
   C The area surrounded by half the plane
   C
   S(J) = ABS(DAREA) - BASE
   C CONTINUE
   C RETURN
   C END
SUBROUTINE FTR(F, P, MAC, NP, RO, R1, TORX)

This program uses F-function theory to predict the pressure
signature at far field when an initial pressure signature is given

PARAMETER (INAX=1400)
DIMENSION F(NP), X(NMAX), P(INAX)
DIMENSION YSTP(NMAX)
DIMENSION DR(3)

OPEN(UNIT=1, FILE=’p.out’)

Input of initial parameters and pressure signal

MAC = Free-stream Mach number
RO = Initial distance from the body (altitude)
NF = Final distance from the body (altitude)
NP = Number of data points (NP < NMAX)
NMAX = This should be large enough to resolve the signature
TORX > 0, sonic boom versus time, also versus distance

NAMELIST /EPSCALE, ESSCALE, SCALE, PE, PG, PS, IBASE, ALT

Read the input parameters
READ(1, EPSCALE)
WRITE(1, EPSCALE)

Define the parameters used in the F-function theory

GAMMA = 1.4
B = SQRT(MAC**2 - 1)
CAP = (GAMMA-1)/(GAMMA+1)*B/(1-B+B**2)
SRR1 = SQRT(R1)

IF RO > 0, extrapolate from RO to R1. First calculate the F-fs.

IF (RO < 0 .AND. THEN
OPEN(UNIT=1, FILE=’p.out’)
DO 15 I=1,NMAX
HEAD(RO, TORX)(IT, P(I))
15 CONTINUE
DO 30 I=1, NP
F(I) = SQRT(2.*B*RO)*P(I)/(GAMMA+CAP+SRR1)*P(I)
X(I) = X(I) - B*RO + CAP*SRR1*P(I)
30 CONTINUE
ENDIF

Y is transposed coordinate

WRITE(1, 48)
DO 50 I=1, NP
Y(I) = X(I) - CAP*SRR1*P(I)
50 CONTINUE
WRITE(1, 48), X(I), P(I)

Find the largest and smallest values of Y

YMAX = MAX(Y(I))
YMIN = MIN(Y(I))
DO 55 I=1, NP
IF (Y(I) .GT. YMAX) THEN
Y(I) = YMAX
55 CONTINUE

Print out the integral curve of the shifted F-function

CALL INTY(NP, T, Y)

Need to march in Y-direction, define the step

YSTP(1) = YTMIN
YSTP(N) = YTMAX
DO 70 J=1, NMAX
YSTP(J) = YSTP(J-1) + YS
70 CONTINUE

March through the shifted F-function, check area balance and
place the shock

CALL MARCH(NMAX, NP, Y, YSTP, T)

Obtain the solution

NOTE: IF TORX > 0, the sonic boom is in the form of F-Plot vs time
as in Fig. As in the form of F-Plot vs distance.

DO 150 I=1, NP
P(I) = GAMMA+CAP+SRR1*P(I)/(3**0.5)*CAP
X(I) = X(I) + P(I)
150 CONTINUE

Make the data points evenly distributed manner and

Scale the sonic if desired

DO 200 J=1, NP
X(I) = X(I)*YSCALE
P(I) = P(I)*YSCALE
200 CONTINUE

Atmospheric aspect

ALT = Altitude
AE = speed of sound at ground, in ft/sec
PG = reference pressure lb/ft^2 = SQRT(PAE*PG)
P0 = pressure at the ground
PG = pressure at flight altitude
V0 = MAC*AE
T0 = X(1)/V0
IF (TORX < 0 .AND. THEN
DO 300 J=1, NP
X(I) = X(I)/X0 - T0
300 CONTINUE

The signal (DP vs Time) is calculated, use a empirical program to
calculate the rise time, and embed the rise time into the signature.
C Note: Unit used is still the stupid English unit!
C CALL MISSILE(FNACE,P,X,NP,ALT,TRISE)
C Obtain the noise level
C CALL NOISE(DBVL1,X,P,NP)
C Write the db(PL) value out
WRITE(14,500)DBVL1,DPVL2,DPVL3
421 FORMAT(6X,'Noise level ',F10.4,'PL ',X,F10.4,'dB(A),',X,F10.4,'dB(C)')
ENDIF
C Write the sonic boom
WRITE(14,555)R1
C Format to display pressure signal at Ri= .F10.4
DO 670 I=1,NP
WRITE(14,700)X(I),P(I)
CONTINUE
670 CONTINUE
CLOSE(1)
C RETURN
END
SUBROUTINE INTF(NP,Y,F)

C This program prints out the integral curve of the shifted F-function

DIMENSION F(NP),Y(NP)

OPEN(UNIT=34,FILE='icurve.f',FORM='FORMATTED')

C

SUMF = 0
DO 100 J=1,NP
   DY = Y(J) - Y(J-1)
   SUMF = SUMF + 0.5*DY*(F(J)+F(J-1))
100 WRITE(34,130)Y(J),SUMF
CONTINUE

120 FORMAT(42R# Integral curve of the shifted F-function)

CLOSE(34)
RETURN
END
SUBROUTINE SHEET(NMFP,Y,YSTP,F,INDEX,FS,YS,STP,FST)

DIMENSION Y(NMFP),F(NMFP)

COMMON/SHOCK/ INDEX

YEND = Y(NMFP)
FIRST = 1.
DO 500 J=2,NMAX
  YS = YSTP(J)

CALL POINT(NF,Y,INDEX,FS,YS,STP)

CALL AREA(NP,Y,F,FS,FS,INDEX)

IF(INDEX.GT.2) THEN
  IF(ILEQ.3) INDEX = 3
  IE=INDEX(INDEX)
  CALL AREA(NP,Y,F,FS,FS,IE,IFLAT2)
ELSE
  The tail shock is already formed, leave program.
  IF(INDEX.LE.1 .AND. YS.GT.YEND-1.5) RETURN
  FIRST=1
  GOTO 500
ENDIF

IF(FIRST.GT.0) IFLAT1 = IFLAT2
IF(IFLAT2.EQ.0) RETURN
FIRST = -1.
ELSE
  IF IFLAT = 0, IS is the point that have area balanced.
  IF IFLAT2 and IFLAT are in different signs, i.e.,
  the computed point should be between I and I-1.
  Use bisection method to find the correct point(IINSTN)
  IF(IFLAT2.IFAT2 .LT. 0.) THEN
    Y1 = YSTP(J-1)
    Y2 = YSTP(J)
    NC = 500
    DO 200 IC=1,NC
      YS = 0.5*(Y1+Y2)
      CALL POINT(NP,Y,F,INDEX,FS,YS,STP)
      IF(IFLAT.EQ.3) INDEX = 3
      IE=INDEX(INDEX)
      CALL AREA(NP,Y,F,FS,FS,IE,IFLATO)
      IF(IFLAT.GT.0) RETURN
      IF(IFLAT.EQ.0)  RETURN
      IF(IFLAT.LT.0) THEN
        Y2 = YS
        IFLAT2 = IFLAT1
      ELSE
        Y1 = YS
        IFLAT1 = IFLAT2
      ENDIF
      WRITE(*,*) 'After ',NC,' steps of bisection'
      RETURN
  ELSE
    IFLAT1 = IFLAT2
  GOTO 500
ENDIF

FIRST = 1.
500 CONTINUE
RETURN
END
SUBROUTINE POINT(NP, Y, P, INDEX, PS, YS, INSCT)

DIMENSION Y(NP), P(NP)

DIMENSION INDEX(40), FS(40)

C Find the points FS on the F-function when YS is given

INDEX = the index runs from 1 to NP
INSC = f of points being intersect, at least point to do integration

IF(YS AT Y(I)) THEN
INSC = +1
PS(INSC) = 0.
INDEX(INSC) = 1
ENDIF

DO 100 I = 2, NP

FAC = YS - Y(I)
IF(FAC = YS - Y(I)) THEN
IF(ABS(FAC) LE 1.E-14) THEN
WRITE(*, *) 'ZEROOOOOOO', YS, Y(I), I
INSC = 0
RETURN
ENDIF
INSC = INSC + 1
SI = (F(I) - F(I-1)) / (Y(I) - Y(I-1))
FS(INSC) = F(I-1) + SI * (Y(I) - Y(I-1))
INDEX(INSC) = I
ENDIF
100 CONTINUE

IF(YS GT Y(NP)) THEN
INSC = +1
FS(INSC) = Y(NP)
INDEX(INSC) = NP
ENDIF

RETURN
END
SUBROUTINE AREA(NP,F,Y1,F1,Y1,IFLAT)

DIMENSION Y(NP),F(NP),F1(NP)

COMMON/SHOCK/ INSCT

C Fine the integral of F by trapezoidal rule
C Integrating from Y(i) to Y(i+1)
C and Y(i+1) to Y(i+2)
C and Y(i+2) to Y(i+1)
C Thus E2 should be
C subtracted out
C and E1 should be added in

EI = 0.5*(Y(1S)-Y1)*(F(1S)+F1(1))
IF(F1(1S)-EQ.0.) EI = 0.
E1 = 0.5*(Y(1S)-Y1)*(F(1)+F1(1))

AREAL = E1

DO 10 I = 1,IE

SLAP = 0.5*(Y(I+1)-Y(I))*(F(I+1)+F(I))

AREAL = AREAL + SLAP

10 CONTINUE

A = AREAL - E1

IF(A.GT.0.) IFLAT=1
IF(A.LT.0.) IFLAT=-1
IF(N2S(A).LT.1.E-7) IFLAT=0

RETURN

END
SUBROUTINE IQUDBODY(JDI, S, TAU)
C  This subroutine finds the area distribution of
C  the wing-body configuration.
DIMENSION S(JDIM), TAU(JDIM)
P1 = 4.*ATAN(1.)
ANG = 21.*PI/180.
ANG1 = 80.*PI/180.
DIM = 25.52*FLOAT(JDIM-1)
S(1) = 0.
DO 2 J = 2, JDIH
  TAU(J) = TAU(J-1)+DX
  TTT = TAU(J)-.701
  IF(TTT < 0.) TTT = 0.
  RR = 0.54-0.011*TTT**2
  S(J) = PI*RR*RR
  IF(TAU(J).GT.12.25 .AND. TAU(J).LT.15.77688849) THEN
  AA = 4.*0.5*0.05*TAN(ANG)* ((TAU(J)-8.21)**2-1.)*RR
  S(J) = S(J) + AA
  ENDIF
  IF(TAU(J).GT.15.77688849 .AND. TAU(J).LT.16.2) THEN
  AA = 4.*(0.5*AA+0.5*(S(J-1)+S(J))*RR)
  S(J) = S(J) + AA
  ENDIF
  IF(TAU(J).GT.16.2) THEN
  SLOPE = (0.15-0.54)/(17.83-17.52)
  BB = 0.54+SLOPE*(TAU(J)-17.52)
  S(J) = PI*RR**2
  ENDIF
  CONTINUE
  RETURN
END
SUBROUTINE CONE(JDIM, S, TAU)
C This subroutine finds the area distribution of the cone-cylinder.
C with half-angle 3.24 degree and 8.6 units of length.
       DIMENSION S(JDIM), TAU(JDIM)
PI = 4. * ATAN(1.)
ANG = 3.24 * PI / 180.
DX = 16. / FLOOR(JDIM - 1)
S(1) = 0.
TAU(1) = 0.
DO 2 J = 2, JDIM
   TAU(J) = TAU(J - 1) + DX
   IF (TANG(J) .LE. 8.6) THEN
      R = TANG(J) * TAN(ANG)
   ELSE
      R = 8.6 * TAN(ANG)
   ENDIF
   S(J) = PI * R * R
2 CONTINUE
RETURN
END
SUBROUTINE SEAMS(JDIM, S,FAM)
   C This subroutine finds the area distribution of the Sears-Adams body
   C with filmness ratio 23.5
   C DIMENSION S(JDIM), TAU(JDIM)
   C # of point on body + 1 # of point on sting = JDIM
   C JDIM = JDIM + 2, 1
   C BLD = JDIM + JBDY
   C PI = 4.*ATAN(1.)
   C BL = 1.
   C TOTL = 1.9* BL
   C F = 23.5
   C DTHETA = PI/FLOAT(JBDY-1)
   C DX = BL/FLOAT(JBDY-1)
   C RMAX = BL/(2.*F)
   C RI = 0.
   C TAU(1) = 0.
   C Consists of Sears-Adams body
   C write('','')'Input Abase/Amax'
   C read('','') AB
   C write('','')'Input Rmax'
   C read('','') AA
   C RMAX = AA*BL
   C CONST = AA/PE
   C CI = 1./2.*(2.*RMAX/BL - 1.)
   DO 2 J=2,JBDY
   C THETA = PI-DTHETA*FLOAT(J-1)
   C TAU(J) = (1. + COS(THETA))**BL/2.
   C TAU(J) = FLOA(T(J-1))*DI
   C THETA = ACOS(2.*TAU(J)/BL - 1.)
   C Sears-Adams body
   C POS = (SIN(THETA)**3)**1
   C Sears-Adams body
   ADD a sting
   C DE = (TOTL-Bl)/FLOAT(JBDY)
   DO 5 J=2,JBDY+1,JDIM
   C TAU(J) = TAU(J-1) + DE
   C HI(J) = S(J-1)
   5 CONTINUE
   RETURN
   END
SUBROUTINE BULLET(JDIM, S, TAU)

C This subroutine find the area distribution of a bullet with a form

C R = AT GAMS

C DIMENSION S(JDIM), TAU(JDIM)

PI = 4.*ATAN(1.)
BL = 4.
GAMA = 0.65
RBASE = 0.25
A = RBASE/(BL**GAMA)
TOTLEN = BL + 2.*BL
DX = TOTLEN/FLOAT(JDIM-1)
S(1) = 0.
TAU(1) = 0.
DO J=2, JDIM
    TAU(J) = TAU(J-1) + DX
    IF (TAU(J) .GE. BL) THEN
        R = RBASE
    ELSE
        R = A*TAU(J)**GAMA
    END IF
    S(J) = PI*R*R
    CONTINUE
    RETURN
END
SUBROUTINE WING(JDIN, S, TAU)

C This subroutine find the area distribution of the low-aspect-ratio wing

DIMENSION S(JDIM), TAU(JDIM)

PI=4.*ATAN(1.)
STING = 0.
DX = 1./FLOAT(JDIM-1)
S(1) = 0.
TAU(1) = 0.
DO J=2,JDIM

TAU(J)=TAU(J-1)+DX

IF(TAU(J).LE.2.)THEN
Z = (PI/12.5)*((TAU(J)-0.5*TAU(J)*TAU(J))
S(J) = Z
ENDIF
IF(TAU(J).GT.1.70007) THEN
STING=PI*0.0625*0.0625
S(J) = Z + STING + Z*0.125
ELSE
STING=PI*0.0625*0.0625
S(J) = STING
ENDIF
2 CONTINUE
RETURN
END
SUBROUTINE BRFUNC(JDIM,S,TAU)

This subroutine obtains the B-function from fort.10 and add it into the equalisation.

PARAMETER (NMAX=800)

DIMENSION S(JDIM),TAD(JDIM),B(NMAX),Z(NMAX)

COMMON/PAR/ PARAC,PPAC

OPEN(UNIT=33,FILE='coeff.dat',FORM='FORMATED')

READ(33,12)

READ(33,12)

READ(33,12)

READ(33,12)

READ(33,12)

DO 19 J=1,NMAX

READ(33,15,END=17) B(J),CL,CD,SLG,B(J),CH

B(J) = B(J)*PPAC

10 CONTINUE

11 CONTINUE

12 CONTINUE

13 FORMAT(6,13.5)

14 CONTINUE

CLOSE(33)

NPOINT = J-1

OPEN(UNIT=33,FILE='bfs.dat')

DO 20 J=1,NPOINT

WRITE(33,'X') X(J),B(J)

20 CONTINUE

C

ISTART = 1

DO 50 J=1,JDIM

DO 30 I=ISTART,NPOINT

IF (ABS(X(I)-TAD(J)).LE.1.E-10) THEN

S(J) = S(J)+S(J)

ISTART = I

GOTO 40

ENDIF

IF (X(I).GT.TAD(J)) THEN

IF(I.EQ.1) THEN

BP = 0.

IP = 0.

ELSE

BP = B(I-1)

IF (X(I-1).LT.X(I)) THEN

SLOPE = (B(I)-BP)/(X(I)-X(I-1))

BT = B(I) + SLOPE*(TAD(J)-X(I))

S(J) = S(J) + BT

ISTART = I

ENDIF

ELSE

IF (X(I).LE.NPOINT) GOTO 30

S(J) = S(J) + B(NPOINT)

ISTART = I

GOTO 40

ENDIF

ENDIF

30 CONTINUE

40 CONTINUE

50 CONTINUE

RETURN

END
SUBROUTINE WB(JDIM, SP, TAU)
C       This subroutine obtains the wing-body interference correction
C       and add it into the derivative of equivalent area
C       This is a test for wing-body case
DIMENSION SP(JDIM), TAU(JDIM)
COMMON/DR20/ DR120, PFAC
C
DO 10 J=1,JDIM
   IF(TAU(J).GE.8.2 .AND. TAU(J).LE.12.25)
      SP(J) = SP(J) + 4.* TAU(J) + 0.54
   IF(TAU(J).GT.12.25 .AND. TAU(J).LE.16.29)
      SP(J) = SP(J) + 4.* TAU(J) + 0.54
   CONTINUE
10 RETURN
END
SUBROUTINE FUNC(TAU,FTAU,JDIM)

DIMENSION TAU(JDIM),FTAU(JDIM)

NAMELIST /FFUNC/ YF,ELAM,C,B,B,E,ELAM,BL,TR,DEL

READ(1,FFUNC)
WRITE(6,FFUNC)

TAU(1)=0.
FTAU(1)=0.

DO 10 J=2,JDIM

TAU(J)=TAU(J-1)+DY

IF(TAU(J).EQ.0.) GOTO 6

IF(TAU(J).GE.YF/2. AND. TAU(J).LE.YF)

6 CONTINUE

WRITE(6,FFUNC)

DO 20 J=1,JDIM

WR_TE(13,80) TAU(J), FTAU(J)

20 CONTINUE

75 FORMAT(2X, F8.4, IX)

RETURN

END
SUBROUTINE AREA(S,PTAU,TAU,JDIM)
DIMENSION S(JDIM),TAU(JDIM),PTAU(JDIM)
DIMENSION F(500)

C Obtain the equivalent area from F-function via Abel Transform

A(A) = \int_{0}^{B} \left( \frac{f(t)/\sqrt{t}}{y-t} \right) dt 

C

I = 0
TAU(I) = 0.
DO 10 J=1,JDIM
SS = 0.
DO 7 I=1,J-1

F(500) = 0.
DO 5 I=I+1,10
25 = TAU(K-1) - TAU(I)

SS = SS + 2.*F(500)*25

CONTINUE

7 CONTINUE

DO 20 J=1,JDIM

TAU(J) = TAU(J-1)

WRITE(12,15)
FORMAT(2X,6F12.8)

20 CONTINUE

RETURN
END
SUBROUTINE ELTIME(PINCH,P,T,HP,ALT,IRISE)

An empirical method to calculate the rise time of a sonic boom

RISE time derived from regression analysis of Air Force sonic boom

flight test data. Good for H-wave type of signal, may be somewhat

conservative (shorter rise time).

All unit used are English units !!!

PINCH = Free-stream Mach number

P = Sonic boom

PSB = Shock strength

ALT = Altitude (ft)

PO = Free-stream pressure (lb/ft^2)

PH = Rise time (sec)

TEMP = Temperature = \text{(15.489.87} /5.5)K

DIMENSION PH,HP

PO = 2114.2

TEMP = 518.69

IOUTH = 0

CONTINUE

IOUTH = IOUTH + 1

CONTINUE

DO 1 I=1,HP

IF(PH.EQ.0.0) GOTO 10

PSB = 0.

PH = 0.

DO 30 I=1,HP

IF(PH.EQ.0.0) GOTO 10

PSB = ABS(P(PH,1)-P(IH))

ELSE

GOTO 30

ENDIF

CONTINUE

IOUTH = 0

CONTINUE

IF(PH.EQ.0.0) RETURN

PH = 1.

PH = 0.

PH = 0.

DO 300 I=ISH0,ISHI-1

T(I) = T(I) + DRT

CONTINUE

DO 300 I=ISHI,ISHI+NP

T(I) = T(I) + RT

CONTINUE

IF(IOUTH.LT.10) GOTO 12

RETURN

END

SUBROUTINE ELTIME(PINCH,P,T,HP,ALT,IRISE)

An empirical method to calculate the rise time of a sonic boom

RISE time derived from regression analysis of Air Force sonic boom

flight test data. Good for H-wave type of signal, may be somewhat

conservative (shorter rise time).

All unit used are English units !!!

PINCH = Free-stream Mach number

P = Sonic boom

PSB = Shock strength

ALT = Altitude (ft)

PO = Free-stream pressure (lb/ft^2)

PH = Rise time (sec)

TEMP = Temperature = \text{(15.489.87} /5.5)K

DIMENSION PH,HP

PO = 2114.2

TEMP = 518.69

IOUTH = 0

CONTINUE

IOUTH = IOUTH + 1

CONTINUE

DO 1 I=1,HP

IF(PH.EQ.0.0) GOTO 10

PSB = 0.

PH = 0.

DO 30 I=1,HP

IF(PH.EQ.0.0) GOTO 10

PSB = ABS(P(PH,1)-P(IH))

ELSE

GOTO 30

ENDIF

CONTINUE

IOUTH = 0

CONTINUE

IF(PH.EQ.0.0) RETURN

PH = 1.

PH = 0.

PH = 0.

DO 300 I=ISH0,ISHI-1

T(I) = T(I) + DRT

CONTINUE

DO 300 I=ISHI,ISHI+NP

T(I) = T(I) + RT

CONTINUE

IF(IOUTH.LT.10) GOTO 12

RETURN

END
SUBROUTINE DISTARC2(X,Y,Z,NEWW,NEW,NNEW,MAX,PAC,IFLAT)

DIMENSION X(N),Y(N),Z(NEW,NNEW),NEW(NNEW)

This program redistributes the points (X,Y) by subroutine DISTRI
based on the arc length. XAC is the first grid spacing. Note that
if (MAX, IEEE) .GT. MAX, a warning message is written.

IF (IFLAT .NE. 0) THEN
  WRITE (*,*) 'SUB DISTARC : MAX is less than N or NNEW'
  STOP
END IF

DIMENSION S(MAX),TOTARC(MAX),Z(NEW,NMAX),TH(NEW)

PARAMETER (MAX=2000)

MAXIMUM number of points allowed is MAX
IF (MAX,IEEE) .GT. MAX, WRITE message is written.

IF (IFLAT .NE. 0) THEN
  WRITE (*,*) 'SUB DISTARC : MAX is less than N or NNEW'
  STOP
ENDIF

Look for total arc length

TOTARC(1) = 0.
DO 10 I=2,N
  ARC = SQRT((X(I)-X(I-1))**2 + (Y(I)-Y(I-1))**2)
  TOTARC(I) = TOTARC(I-1) + ARC
10 CONTINUE

Apply subroutine DISTRI to obtain the stretching function S

IF (IFLAT .NE. 1) THEN
  DELT = PAC + TOTARC(1)/FLOAT(NNEW-1)
  CALL DISTRI(DELT,NNEW,TH,IFLAT)
ELSE
  S(I) = 0.
  DO 20 K=2,NNEW
    S(I) = S(I-1) + 1./FLOAT(NNEW-1)
 20 CONTINUE
ENDIF

Redistribution, put new array in a temporary arrays Xm and Ym

X(1) = X(1)
Y(1) = Y(1)
X(MNEW) = X(N)
Y(MNEW) = Y(N)

DO 40 J = 2, NNEW
  ARCHM = S(J)**2/TOTARC(M)
  DO 30 K = 2,M
    IF (TOTARC(K) .GE. ARCHM) THEN
      XM(2) = X(K)
      YM(2) = Y(K)
      GOTO 40
    ELSE
      IF (TOTARC(K) .GE. ARCHM) THEN
        I1 = X(I-1)
        I2 = X(I)
        T1 = Y(I-1)
        T2 = Y(I)
        XM(2) = I1 + (X(I)-X(K-1))*
               (ARCHM-TOTARC(K-1))/(TOTARC(K)-TOTARC(K-1))
        YM(2) = T1 + (Y(I)-Y(K-1))*
               (ARCHM-TOTARC(K-1))/(TOTARC(K)-TOTARC(K-1))
      END IF
      CALL LIMIT(E1,E2,E5,E6,E7,E8)
      XM(2) = E1
      YM(2) = E7
      GOTO 60
    END IF
  30 CONTINUE
40 CONTINUE
**SOURCE PROGRAM**

**LHF.f**

---

**LINE #**

1077 **-----------------------------------------------**

1078 SUBROUTINE DISTRI(FANG,ETCS,5,IFINC)

1079   PARAMETER (MAX=500)

1080   DIMENSION S(MAX),DUM(MAX)

1081   C Calculating the stretching function S when given

1082   C the first spacing, FANG, and the number of points ETCS

1083   C If IFINC=1, distribution is clustering at outer grid

1084   C

1085   IF(MAX.LE.ETCS) THEN

1086       WRITE(*,*)'SUB DISTRI : MAX is less than ETCS'

1087       STOP

1088       ENDIF

1089   IF(ETCS.EQ.1) THEN

1090       S(I) = 0.

1091       GOTO 40

1092       ENDIF

1093   C

1094   C

1095   C

1096   DI = FANG

1097   EPM = ETCS-1

1098   DIETA = 1./FLOAT(EPM)

1099   EDBETA = 1.5

1100   CALL GEBY(DI,EPM,C.0001,100,EDBETA)

1101   CALL FII(ETCS,EDBETA,DIETA,5)

1102   C

1103   C

1104   C

1105   DO 37 I=1,ETCS

1106       S(I) = MAX/(ETCS-1) - S(I)

1107       CONTINUE

1108       DO 38 I=1,ETCS

1109       S(I) = 1./MAX

1110       CONTINUE

1111       ENDIF

1112       CONTINUE

1113       RETURN

1114       END
SUBROUTINE F2(L1, THETA, DET, Z)
C COMPUTES NORMALIZED NORMAL DISTANCE, Z(L)
DIMENSION Z(25)
IF(TBETA.EQ.1.) THEN
   DO 10 L=1,11
      Z(L)=0.
   CONTINUE
ELSE
   DO 20 L=1,11
      ETA=(L-1)*DET
      RR=(THETA-1.)/(TBETA-1.)
      EEE=1.-ETA
      RTheta=RR*EEE
      Z(L)=(TBETA-1.)*(RR-RTheta)/(RTheta+1.)
   CONTINUE
END IF
RETURN
END
SUBROUTINE GRDET(DFM, NPT, IFCC, ICC, BETA)

DIMENSION T(150)

ICL=ICC
IPCC=FPCC*DFM
BETA=BETA
DFM=DFM
DET=1./DFM
BR=1
BP=1
ICLCICC/10
DO 10 J=1,ICLC
BP=BETA
BP0.5*(BETA=1.)
CALL FIZ(2, B, NP, DET, E)
FP*2.25-1
IF(FP.GT.0.)GO TO 15
BETA=2*BETA-1.
10 CONTINUE
15 CONTINUE
DO 5 BP=1,1ICLC
CALL FIZ(2, B, NP, DET, E)
P=5.25-1
IF(FP.GT.0.)THEN
FP=F
BP=BETA
ELSE
FP=F
BP=BETA
END IF
5 CONTINUE
IF(MCP.FLT.FPCC)GO TO 4
WRITE(6,100) BETA, F
100 FORMAT(180,168 EXCEEDED MAX. NO. OF ITS... BETA, F, 3G13.6)
4 CONTINUE
CALL FIZ(2, B, NP, DET, E)
END
This subroutine linearly interpolate YLOCAL when given (X1,Y1) & (X2,Y2)

```
SUBROUTINE LININT(X1,X2,Y1,Y2,YLOCAL)

CIF(X1.EQ.X2) THEN
YLOCAL=(Y2-Y1)/2.
ENDIF
SLOPE=(Y2-Y1)/(X2-X1)
YLOCAL=SLOPE*(XLOCAL-X2)+Y2
CONTINUE
RETURN
END
```

...
SUBROUTINE MARCH(NMAX, Y, YSTP, F)
DIMENSION F(NP), Y(NP)
DIMENSION YSTP(NMAX)
DIMENSION INDEX(40), FS(40)

This subroutine marches the y direction and checks if the areas are balanced and then places the shock.

KOUNT = 0
YEND = Y(NP)

CONTINUE
DO IND=1,40
INDEX(IND)=0
INSCT = 0
ENDDO
CALL SBRPT(NMNX, NP, Y, YSTP, F, INDEX, FS, YS, 0)

300 CONTINUE

310 CONTINUE

320 CONTINUE

Form the shock
IF(INST.EQ.1 .AND. YS.GE.YEND) RETURN
FDIS = FS(INST)-FS(INST-1)
IF(FLOAT(INST)-INDEX(1)).EQ.0.0) THEN
WRITE(15,*) 'SDT: ZERO DIVISION ABOUT TO HAPPEN IN MARCH'
df = 1.0E3
ELSE
DF = FDIS/FLOAT(INDEX(INST)-INDEX(1))
ENDIF

IF(INDEX(INST).EQ.INSCT) THEN
Y(INSCT) = YS
ELSE
Y(INST) = YS
ENDIF
GO TO 400
ENDIF
GO TO 100
ENDIF
END
SUBROUTINE SIMPSON(X,F,N,X0,X1,SUM)
DIMENSION X(N),F(N)
C
K = odd
K = K/2
SUM = 0.
DO 10 J=1,N
  ODD = ODD + 2.*F(2*J-1)
  EVEN = EVEN + 4.*F(2*J)
10 CONTINUE
C
SUM = F(1) + ODD + EVEN + F(N)
SUM = SUM*(X1-X0)/(6.*FLOAT(N))
RETURN
END
Appendix B

SAMGRID (Fortran Listing)
PROGRAM SAMGRID
include 'agrid.com'

Dr. Samson Cheung
Date: Dec., 1993
This subroutine reads a surface grid in airfoil sections and formats it to produce a surface grid of axisymmetric cross-sections.
Date: Dec., 1993 Version 3.0
read input geometry

OPEN(UNIT=12, FILE='AGRID.IN', STATUS='OLD', FORM='FORMATTED')
OPEN(UNIT=10, FILE='AGRID.IN', STATUS='OLD', FORM='FORMATTED')
OPEN(UNIT=4, FILE='AGRID.IN', STATUS='OLD', FORM='FORMATTED')

NSEC = # of sections (streamwise stations) of the new grid
NPTS = # of pts in the circumferential direction (MUST be odd)
NMAX = # of streamwise stations
FAC = the first grid spacing in DIST
EX = X leading edge
ARKING = 0, arrow wing
ETIP = number of points in the wing-around direction on one surface
NOUT = # of output points in the upper part of the wing
NIN = # of output points in the lower part of the wing

Read surfgrid dimensions (NSEC x NPTS x 1)
RANFIL /WING, NSEC, NPTS, FAC
READ(40, WING)
WRITE(*, WING)

CALL WINGIN

Setup distribution of cross-sections to be obtained (solid)

IF(NSEC.EQ.1) THEN
  EX = 1 / NSEC
  DIST(1) = (NSEC-1) / NSEC * (XMAX-XMIN)
ELSE
  EX = NMAX / NSEC
  DIST(1) = (NSEC-1) / NSEC * (XMAX-XMIN)
END IF

DO 100 I = 1, NSEC
  DO 100 J = 1, NPTS
  EDIST(J) = (J-1) / NSEC * (XMAX-XMIN)
  IE = IE + 1
100  CONTINUE

ETIP = NPTS / 2

DO 1000 I = 1, NSEC
  DO 1000 J = 1, NPTS
    EDIST(J) = (J-1) / NSEC * (XMAX-XMIN)
7000 CONTINUE

The nose of the wing
DO 187 I = 1, NPTS
  YOUT(I, 1) = YMIN(1)
  ZOUT(I, 1) = ZDIST(NPTS-1)
187  CONTINUE

Begin main loop for each x-station
DO 9990 I = 1, NSEC
  DO 9990 J = 1, NPTS
    Call REDIST(LOCAL, IE, IFAC, 1)
    DO 9990 K = 1, NSEC
      DO 9990 L = 1, NSEC
        LOCAL = LOCAL + 1
     79990 CONTINUE

Do the lower surface
CALL REDIST(LOCAL, IE, IFAC, 1)
DO 9990 K = 1, ETIP
  YOUT(I, K) = LOCAL
  ZOUT(I, K) = YMIN(K)
79990 CONTINUE

Do the upper surface
CALL REDIST(LOCAL, IE, IFAC, 1)
DO 9990 K = ETIP+1, NSEC
  YOUT(I, K) = LOCAL
  ZOUT(I, K) = ZDIST(NPTS-1)
79990 CONTINUE

For the computational grid of SURF grid code
the wake has to have two different pts in same
physical location, such that (Y1, Z1)-(Y2, Z2). Here
the calculation divided into upper and lower parts.
If needed, set E1 = E2
IF(SAMGRID.EQ.1) THEN
  IF(E1.EQ.E2) THEN
    IF(E1.EQ.0) THEN
      E1 = E1 + 1
      E2 = E2 + 1
    ELSE
      IF(E1.EQ.1) THEN
        E2 = E2 + 1
      ELSE
        E1 = E1 + 1
      END IF
    END IF
  END IF
ENDIF

500 CONTINUE
9990 CONTINUE
C Proceed to next x coordinate
9990 CONTINUE
C Write out new surfgrid in point format
EN=1
WRITE(50)NP, KS, NSEC
DO 1234 I=1, NSEC
   WRITE(50)(OUT(I,L), L=1, NP)
   (YOUT(I,L), L=1, NP)
1234 CONTINUE

Write out original database in plotid format
NL=NUMNL
WRITE(11) (XB(I,1), I=1, NUM), (XB(I,2), I=NL, 1, -1),
           (YB(I,1), I=1, NUM), (YB(I,2), I=NL, 1, -1),
           (ZB(I,1), I=1, NUM), (ZB(I,2), I=NL, 1, -1),
           (WB(I,1), I=1, NUM), (WB(I,2), I=NL, 1, -1)
CONTINUE

Read the fuselage grid and combine the fuselage with the
wing grid to form a whole configuration.
CALL RGRID

Read the noseles grid and combine the noseles with the
wing-body grid.
CALL PROGRID

CLOSE(10)
CLOSE(30)
STOP
END
SUBROUTINE ADDGRID(NP1, NP2, I, J, K, MSEC, EDIM)

This subroutine allows us to add a grid line between streamwise section

NPI1 and NPI2, and the new dimension is MSEC again

PARAMETER (MAX=400)
DIMENSION VTMP(MAX), XTEM(MAX), YTEM(MAX), ZTEM(MAX)
DIMENSION X(NP1), Y(NP1), Z(NP1)

IF(MAX.LE.NPI1) THEN
  WRITE('** Sub Addgrid: Max is less than NPI1 **
  STOP
ENDIF

IF(NP1.GE.NPI2) THEN
  WRITE('** No plane is added in the streamwise direction **
  STOP
ENDIF

C Interpolating the new grid, and put it in a temporary array

II = X(NPI1)
II = X(NPI2)
XX = 0.5*(X(NPI1)+X(NPI2))
DO 10 K = 1, EDIM
  VI = Y(NPI1,K)
  VJ = Y(NPI2,K)
  ZI = Z(NPI1,K)
  ZJ = Z(NPI2,K)
  CALL LINTER(XI, XI, X2, Y1, Y2, Z1, Z2)
  YTEM(K) = VI
  ZTEM(K) = ZI
10 CONTINUE

C Re-number the last stations

MSEC = MSEC+1
DO 10 L = MSEC, MSEC+1-1
  XI(L) = X(L-1)
10 DO 30 I = 1, EDIM
  YI(L) = Y(L-1)
  ZI(L) = Z(L-1)
30 CONTINUE

C Put the temporary array in the grid

DO 50 K = 1, EDIM
  X(NPI1,K) = XX
  Y(NPI1,K) = YTEM(K)
  Z(NPI1,K) = ZTEM(K)
50 CONTINUE

RETURN
END
SUBROUTINE CIRCLE (XS, YS, EMAX, X, YFIL, ANCOR)

DIMENSION X(MAX), Y(MAX), X(MAX), YFIL, ANCOR

Given a set of points (Y(I),X(I)) I=1,...,MAX and radius of filled RPFI.
this subroutine replaces the points (Y(I),X(I)) I=1,...,EE by the filled
points on filled circle.

Look for the center of the filled circle (YC, XC)

TA=XS
TB=YS
BB=(TA*TA+TB*TB+2)*A
R = BB/5.

A = 1-R**4
B = 2-R**2 - 2-R**4
C = 2-2B

DET=B*B-A*C

IF(DET.LE.0.) THEN
  WRITE(5,*)'Determinant is less than 0., DET
  GO TO 200
ENDIF

YCI = (-D+SQRT(D)) / (2.-A)
ZCI = (BB-YCI*TBY+TA) / (2.*A)
YC3 = (BB-ZCI*TBY+TA) / (2.*A)

IF(YC3.GE.ZC3) THEN
  ZC=ZC1
  YC=YC3
  ENDIF

TC=TC1
ELSE
  ZC=ZC2
  YC=TC2
  ENDIF

Find the total arc length given

TYPARC=0.
DO 50 EE=1,EE-1
  CONTINUE
  ARC = TYPARC*PI/EE-1
  ARC = ARC+ANCOR
  CONTINUE

Find the coordinates for each point

DO 100 EE=1,EE-1
  YCI=XS
  YC3=YS
  YC=TA
  EB=ZC

  SY=TA-YB
  SS=ZB-EB
  BB=RFIL+2-ARC**2
  R = SS/5.

  A = 1-R**4
  B = 2-R**2 - 2-R**4
  C = 2-2B

  DET=B*B-A*C

  IF(DET.LE.0.) THEN
    WRITE(5,*)'Determinant is less than 0., DET
    GO TO 200
  ENDIF

  YCI = (-D+SQRT(D)) / (2.-A)
  ZCI = (BB-YCI*TBY+TA) / (2.*A)
  YC3 = (BB-ZCI*TBY+TA) / (2.*A)

  IF(YC3.GE.ZC3) THEN
    ZC=ZC1
    YC=YC3
  ENDIF

  TC=TC1
  CONTINUE

  GO TO 100

100 CONTINUE
100 RETURN
END
C**************************************************************************
C SUBROUTINE CEPLINE(I,J,X,Y,N,NEX,NHEN,NHEN)
C**************************************************************************

C THIS SUBROUTINE PRODUCES A MONOTONE CUBIC SPLINE INTERPOLANT
C TO THE DATA (X(I),Y(I)) I=1,...,N AND COMPUTES VALUES AT
C THE NEW POINTS XNEW(1),...,XNEW. THESE ARE RETURNED IN
C YNEW. THE ALGORITHM USED IS THAT OUTLINED BY FISCHER AND
C WRITTEN BY JEFF CORDOVA 10/31/86.
C**************************************************************************
C REAL D(NMAX), DEL(NMAX), R(NMAX)
C**************************************************************************
C SPLINE COEFFICIENT CALCUATIONS
C**************************************************************************

C DO 100 I=1,N-1
C X1(I) = X(I+1) - X(I)
C 100 CONTINUE
C DO 200 I=1,N-1
C DEL(I) = (Y(I+1) - Y(I)) / X1(I)
C 200 CONTINUE
C
C SPLINE COEFFICIENTS
C**************************************************************************

C *** LINEAR INTERPOLATION FOR N=2 CASE ***
C IF (N EQ. 2) THEN
C D(1) = DEL(1)
C D(2) = DEL(1)
C GO TO 399
C ENDIF
C
C *** MONOTONE SPLINE COEFFICIENTS FOR N > 3 CASE ***
C
C FIRST BOUNDARY POINT (USE THREE POINT FORMULA ALIGNED TO BE
C SHAPE PRESERVING)
C
C EXSM = X(I-2) + X(I-1)
C W1 = (X(I) - EXSM) / EXSM
C W2 = (X(I) - EXSM) / EXSM
C D(1) = W1*DEL(I) + W2*DEL(I-1)
C IF (PCREST(D(1),DEL(I)).LE.0.) THEN
C D(1) = 0
C ELSEIF (PCREST(DEL(I),DEL(I)).LT.0.) THEN
C DMAX = 3.*DEL(I)
C IF (ABS(D(1)) .GT. ABS(DMAX)) D(1) = DMAX
C ENDIF

C INTERIOR POINTS (NO BITLESS MODIFICATION OF BUTLAND FORMULA)
C
C CONST = 1. / 3
C DO 300 I=3,N-1
C TOP = DEL(I-1) * DEL(I)
C TOP = TOP * .5 * (1. + SIGN(1.,TOP))
C ALPHA = CONST * (X(I-1) - X(I)) / (X(I) - X(I-2))
C BETA = DEL(I) + ALPHA
C D(1) = TOP / BOT
C 300 CONTINUE

C LAST BOUNDARY POINT (USE THREE POINT FORMULA ALIGNED TO BE
C SHAPE PRESERVING)
C
C EXSM = X(I-2) + X(I-1)
C W1 = (X(I) - EXSM) / EXSM
C W2 = (X(I) - EXSM) / EXSM
C D(N) = W1*DEL(N-2) + W2*DEL(N-1)
C IF (PCREST(D(N),DEL(N-1)).LE.0.) THEN
C D(N) = 0
C ELSEIF (PCREST(DEL(N-2),DEL(N-1)).LT.0.) THEN
C DMAX = 3.*DEL(N-1)
C IF (ABS(D(N)) .GT. ABS(DMAX)) D(N) = DMAX
C ENDIF

C Spline Evaluation
C**************************************************************************

C EXHEN(I).LE.X(1)
C
C IFEND = 1
C CYTWO = D(1) + D(1+1) - 2.*DEL(I+1) / (X(I)+X(I+1))
C CTWO = (3.*DEL(I) - 2.*D(2) - D(I+1)) / X1(I)
C CRAY = XNEW(I) - X(I)
C IBEG = IEONE
C IBEG = IBEG + 1
C IF (CRAY .GT. 0) THEN
C I = XNEW(I) - X(I)
C DO 400 J=IBEG,IBEND-1
C T = XNEW(I) - X(J)
C 400 CONTINUE
C
C EXHEN(I).GT.X(N)
C
C IFEND = 1
C T = XNEW(I) - X(N)
C THEN(I) = T*D(N-1) + T*(CTWO + T*CTREI)
C 500 CONTINUE

C RETURN
C END
FUNCTION PCBST(ARG1, ARG2)

PCBST = SIGN(1., ARG1) * SIGN(1., ARG2)

C IF ((ARG1.EQ.0.) .OR. (ARG2.EQ.0.)) PCBST = 0.

C RETURN
END
FUNCTION ISRCHGE(N,X,INCX,FTARGET)  
DIMENSION I(*)
IF(N.LE.0) THEN  
ISRCHGE = 0  
RETURN  
ELSE  
IT = 1 + (N-1) * INCX  
ISRCHGE = 1  
DO 10 I=1,IT,INCX  
IF(X(I).GE.FTARGET) GOTO 11  
ISRCHGE = ISRCHGE + 1  
10 CONTINUE  
11 CONTINUE  
ENDIF  
RETURN  
END
SUBROUTINE DISTARC(X,Y,N,NNEW,YNEW,NNEW,PS,TFLAT)
C      DIMENSION X(N),Y(N),XNEW(NNEW),YNEW(NNEW)
C      This program redistribute the points (X,Y) by subroutine DISTRI
C based on the arc length. FCL is the first grid spacing. Note that
C the end points of the two sets are the same.
C      IFFLAT=0, grid points will cluster near the first point, -1 near the end.
C      Input array is (X(1),Y(1)), (1),...,N,
C      Output array is (XNEW(1),YNEW(1)), (1),...,NNEW
C
C      PARAMETER (MAX=400)
C      DIMENSION S(MAX),TOTARC(MAX),EN(MAX),TH(MAX)
C      Maximum number of points allowed is MAX
C      IF(MAX LE N .OR. MAX LE NNEW) THEN
C      WRITE('**','**') 'SUB DISTARC : MAX is less than N or NNEW'
C      STOP
C      ENDIF
C
C      Look for total arc length
C      TOTARC(1) = 0
C      DO 10 I=2,N
C      ARC = SQRT( (X(I-1)-X(I-2))**2 + (Y(I-1)-Y(I-2))**2 )
C      TOTARC(I) = TOTARC(I-1) + ARC
C 10     CONTINUE
C
C      Apply subroutine DISTRI to obtain the stretching factor S
C      For FLAT, equal spacing is used
C      IF(FCL(FLAT = 0) THEN
C      DELT = PS/TOTARC(N)
C      ELSE
C      S(1) = 0.
C      DO 25 I=2,NNEW
C      S(I) = S(I-1) + 1./FLOAT(NNEW-1)
C 25    CONTINUE
C      ENDIF
C
C      Redistribution, put new array in a temporary arrays XN and YN
C      XN(1) = X(1)
C      YN(1) = Y(1)
C      X(NNEW) = X(N)
C      Y(NNEW) = Y(N)
C
C      DO 30 J = 2,NNEW
C      ARNEW = S(J)/TOTARC(N)
C      DO 35 K = 2,N
C      IFABS(TOTARC(K)-ARNEW) .LE. 1.E-7) THEN
C      XN(2) = X(K)
C      YN(2) = Y(K)
C      GOTO 40
C      ENDIF
C 30     CONTINUE
C      IF(TOTARC(K) .GT. ARNEW) THEN
C      XI = X(K-1)
C      X2 = X(K)
C      Y1 = Y(K-1)
C      Y2 = Y(K)
C      XX = XI + (X2-X1)**3 / (TOTARC(K-1) - TOTARC(K-1))
C 35     CALL LINTRT(X1,X2,Y1,Y2,XX,Y)
C      ELSE
C      IFABS(X2-X1) .LE. 1.E-7) THEN
C      TN(2) = Y2 + (ARNEW-TOTARC(K-1))
C      ELSE
C      TN(2) = Y2
C      ENDIF
C 40    CONTINUE
C 35    CONTINUE
C 30  CONTINUE
C 40  CONTINUE
C 35  CONTINUE
C 30  CONTINUE
C
C      Write the temporary arrays into the output XNEW, YNEW
C      DO 50 J = 1,NNEW
C      XNEW(J) = XN(J)
C      YNEW(J) = YN(J)
C 50    CONTINUE
C
C      RETURN
C      END
**SOURCE TEXT**

```
569 SUBROUTINE DISTRI(FANG, KFC8, S, IFINE)
570 PARAMETER(MAX=400)
571 DIMENSION S(MAX), DUM(MAX)
572 C........Calculating the stretching function & when given
573 C the first spacing, FANG, and the number of points KFC8
574 C........If IFINE=1, distribution is equalizing at outer grid
575 IF(MAX.LE.KFC8) THEN
576 WRITE('*,*) 'SUB-DISTRI : MAX is less than KFC8'
577 STOP
578 ENDIF
579 IF(KFC8.EQ.1) THEN
580 S(1) = 0.
581 GOTO 40
582 ENDIF
583 C
584 DI1 = FANG
585 EF = KFC8-1
586 DIETA = 1./FLOAT(EF)
587 RDBETA = 1./DIETA
588 CALL GRBET(D1, EF, 0.0001, 100, RDBETA)
589 CALL PSI1(KFC8, RDBETA, DIETA, S)
590 C
591 IF (IFINE.EQ.1) THEN
592 DO 37 E=1, KFC8
593 DUM(KFC8-E+1) = S(E)
594 37 CONTINUE
595 DO 38 E=1, KFC8
596 S(E) = 1. - DUM(E)
597 38 CONTINUE
598 ENDIF
599 END
```
SUBROUTINE ED_E(NC,NU,NL,XL,XBK,IBASE,YBASE,ZBASE,LS)
DIMENSION ZBASE(NPK,2,LS),XBASE(NP_,2,LS),YBASE(NPK,2,LS)
ZLE = ZBASE(I,I,I)
DO 200 K = 1,NC
IF(ZBASE(I,I,K).GT.XBK) THEN
  XI = ZBASE(I,I,K-1)
  X2 = ZBASE(I,I,K)
  Z1 = ZBASE(I,I,K-1)
  Z2 = ZBASE(I,I,K)
  CALL LININT(X1,X2,Z1,Z2,XBK,ZBK)
  GOTO 210
ENDIF
200 CONTINUE
210 CONTINUE
IF(_BASE(1,1,E).GT.XB_)
GOTO 700
CALL LININT(Z_E,XBK,XL,:XB_,E_E(1,1,_),XL_)
XL_OLD = X_SE(I,1,E)
XLEL - XBASE(NL,I,K)
DO 280 I = 1,NL
F = XBASE(I,1,K)-XLEOLD
E = XTL-XBASE(I,1,K)
XBASE(I,1,K) = (F*XTL + E*XLE)/(F+E)
280 CONTINUE
XTL = XBASE(NL,1,K)
DO 300 E = 1,NE
F = ZBASE(1,2,E)-XLEOLD
E = XTL-XBASE(1,2,E)
ZBASE(1,2,E) = (F*XTL + E*XLE)/(F+E)
300 CONTINUE
500 CONTINUE
700 CONTINUE
RETURN
END
**SUBROUTINE EQSPACE**

```fortran
include "sgrid.com"
L2 = L-I
DO 130 L=1,L2
   XIN(LL) = X(LL)
   DO 120 K=1,EDIM
      ZIN(LL,K) = Z(LL,K)
      YIN(LL,K) = Y(LL,K)
   120 CONTINUE
130 CONTINUE

XTOT = X(L2)-X(1)
DX = XTOT/FLOAT(L2-1)
DO 160 JL=2,L2
   X(JL) = X(JL-1)+DX
   DO 150 KL=1,L2
      IF(ABS(XIN(KL)-X(JL)) .LE. 1.E-7) THEN
         DO 140 _I,EDIM
            Z(KL,I) = ZIN(JL,I)
            Y(KL,I) = YIN(JL,I)
         140 CONTINUE
         GOTO 1G0
      END IF
      IF(XIN(KL) .GT._x(3L)) THEN
         DO 145 _I,EDIM
            X1 = XIN(KL-1)
            X2 = XIN(KL)
            Y1 = YIN(KL-1,I)
            Y2 = YIN(KL,I)
            Z1 = ZIN(KL-1,I)
            Z2 = ZIN(KL,I)
            XX = X(JL)
            CALL LININT(X1,X2,Y1,Y2,XX,YY)
            CALL LININT(X1,X2,Y1,Y2,XX,ZZ)
            Y(JL,I) = YY
            Z(JL,I) = ZZ
         145 CONTINUE
         GOTO 160
      END IF
      CONTINUE
150 CONTINUE
160 CONTINUE
RETURN
END
```
SUBROUTINE FILET(Y,1,EDIM,MB1,MB2,MT1,MT2,RF1L)
PARAMETER (MB3=400)
DIMENSION Z(EDIM),Y(EDIM)
DIMENSION DI(MB1),D2(MB2)
COMMON /ARC/ ZROOT,ATIP,ANCORE

C This subroutine takes a wing-fuselage section, (Y1(x),Z1(x)) k=1,EDIM,
C and finds the two (top and bottom) intersections of the wing and the
C fuselage.
C
C And then, for example, at the bottom intersection (Y(x),Z(x)), it
C extends to a segment of points, (Y(x),Z(x)) k=EF1 to EF2, where
C
C EF1-X-MB1, EF2-X-MB2.
C
C Similar procedure for the top part.
C
C And then, call subroutine CIRCLE to replace the segment by a segment
C of a circle with radius RF1L.

IF(RF1L.EQ.0.) GOTO 735
IF(MAX.LE.EDIM) THEN
WRITE(6,'(A)') 'MAX is less than EDIM'
STOP
END

C Bottom part of the aircraft
IF(MB1.EQ.0. AND. MB2.EQ.0.) GOTO 135
DO 130 K=1,EDIM
IF (X.EQ.EF1 .AND. Z.EQ.ZROOT) THEN
EF1=EF1-1
EF2=EF2+1
END IF
CALL CIRCLE(EF1,EF2,EDIM,Y,Z,RF1L,ANCORE)
130 CONTINUE

C We have >EF2-EF1=1 pts in filet area, employ two more points
C from the original grid and redistribute them, the grid spacing looks
C smoother.
EF1=EF1+1
EF2=EF2+1
DO 220 ED=EF1,EF2
D1(ED-EF1+1) = Y(ED)
D2(ED-EF1+1) = Z(ED)
220 CONTINUE

C We have >EF2-EF1=1 pts in filet area, employ two more points
C from the original grid and redistribute them, the grid spacing looks
C smoother.
EF1=EF1-1
EF2=EF2-1
DO 220 ED=EF1,EF2
D1(ED-EF1+1) = Y(ED)
D2(ED-EF1+1) = Z(ED)
220 CONTINUE

C Top part of the aircraft
IF(MB1.EQ.0. AND. MB2.EQ.0.) GOTO 735
DO 700 K=MT1,EDIM
IF(K.EQ.EF1 .AND. Z.EQ.ZROOT) THEN
EF1=EF1+1
EF2=EF2+1
END IF
CALL CIRCLE(EF1,EF2,EDIM,Y,Z,RF1L,ANCORE)
700 CONTINUE

C We have >EF2-EF1=1 pts in filet area, employ two more points
C from the original grid and redistribute them, the grid spacing looks
C smoother.
EF1=EF1+1
EF2=EF2+1
DO 420 ED=EF1,EF2
D1(ED-EF1+1) = Y(ED)
D2(ED-EF1+1) = Z(ED)
420 CONTINUE

C TOP(MTIIE0.0 .AND. MB2.EQ.0) GOTO 735
DO 700 K=MT1,EDIM
IF(K.GT.KTIP .AND. Z(K).LE.ZROOT) THEN
KFI-K-MT1
KF2=K+MT2
KFI=KFI+I
KF2=KF2+I
DO 420 KD=KFI,KF2
DI(KD-KFI+I) = Y(KD)
D2(KD-KFI+I) = Z(KD)
420 CONTINUE
N=KF2-KFI+I
CALL DISTARC(D1,D2,N,D1,D2,N,-10.0,0)
DO 530 KD=KFI,KF2
Y(KD)-DI(KD-KFI+I)
Z(KD)-D2(KD-KFI+I)
530 CONTINUE
GOTO 735
ENDIF
700 CONTINUE
735 CONTINUE
RETURN
END
SUBROUTINE PZI(LI,TBETA,DET,Z)
C
COMPUTES NORMALIZED NORMAL DISTANCE, Z(L)
C
PARAMETER (MAX=400)
DIMENSION Z(MAX)

IF(MAX.LE.LI) THEN
  WRITE(*,*)'SUB P2I : MAX is less than LI'
  STOP
ENDIF

IF(TBETA.EQ.1.) THEN
  DO 10 L=1,LI
    Z(L)=0.
  CONTINUE
ELSE
  ETA=(L-1.)*DET
  RR=-(TBETA+1.)/(TBETA-1.)
  EEE=1.-ETA
  BBETA=RR*EEE
  Z(L)=(TBETA-1.)*(RR-BBETA)/(BBETA-1.)
  CONTINUE
ENDIF
RETURN
END
**SUBROUTINE GEBET (DFM, NPT, E\_CC, IC\_C, BETA)**

**SOURCE PROGRAM DATE**

**7/07/94**

**PAGES**

**NOTE**

**LINE #**

**SOURCE TEXT**

759 SUBROUTINE GEBET (DFM, NPT, E\_CC, IC\_C, BETA)

801 C BISECTION METHOD USED TO DETERMINE STRETCHING PARAMETER, BETA.

803 C WHICH GIVES DESIRED GY AT THE WALL

804 C

805 DIMENSION I( MAX)

806 IF (MAX .LE. NPT) THEN

807 WRITE (*, *) 'SUB GEBET : MAX is less than NPT.'

808 STOP

809 ENDIF

810 C

811 IC\_C = IC\_C

812 FPCC = FPCC * DFM

813 BETA = BETA

815 ZI = DFM

816 DET = NPT

817 BETA = 1

818 ITCC = ITCC * 10

819 DO 10 I = 1, ITCC

820 BET\_A = 0.5 * (BET\_A + 1)

821 CALL FZI (2, BET\_A, DET, Z)

822 FP = FP + Z

823 IF (FP .GE. 2) GO TO 15

824 CONTINUE

825 BETA = 2 * BETA - 1

826 CONTINUE

827 DO 15 CONTINUE

828 DO 5 I = 1, IC\_C

829 CALL FZI (2, BETA, DET, Z)

830 FP = FP + Z

831 IF (FP .GT. 0) THEN

832 FP = FP

833 ELSE

834 END IF

835 IF (ABS (FP) .LT. E\_CC) GO TO 4

836 CONTINUE

837 IF (ABS (FP) .LT. F\_CC) GO TO 4

838 CONTINUE

839 IF (ABS (FP) .LT. E\_CC) GO TO 4

840 CONTINUE

841 WRITE (4, 100) BETA, FP

842 100 FORMAT (169, 35B16, E16.6)

843 4 CONTINUE

844 C

845 C

846 C

847 C

848 C

849 C

850 END
SUBROUTINE LIINT(X1,X2,Y1,Y2,LOCAL,YLOCAL)

C This subroutine linearly interpolate YLOCAL when given (X1,Y1) & (X2,Y2)

IF (ABS(X1-X2) LE .1.E-7) THEN

YLOCAL = (Y2+Y1)/2.

GO TO 100

ENDIF

SLOPE = (Y2-Y1)/(X2-X1)

YLOCAL = SLOPE*(XLOCAL-X2) + Y2

100 CONTINUE

RETURN

END
SUBROUTINE MOUNT(JN,INUM,IPRM)

INCLUDE "NEP10.COM"
DIMENSION NPAI(2,NPI), NWK(4)
DIMENSION XENG(NPI), YENG(NPI), ZENG(NPI)

COMMON XENG, YENG, ZENG

MC = N of pts added in the grid (=) pts at nacelle)
IPRM = 0 no writing out
N = 40
NPS = NPTS + MC
IF(INUM(1).EQ.LNUM(3)) AND. LNUM(2).EQ.LNUM(4)) THEN
NPTS = NPTS + MC
NPS = NPTS + MC
ELSE
NPTS = NPTS + MC
ENDIF
NPS = (NPTS+1)/2
NPSM = (NPSM+1)/2
NAC = NWK(JN)
NACPP = NAC(JN)
I IF(IPRINT .NE. 0) WRITE(*,*) LNUM(3), LNUM(2)-LNUM(1)+1
HOW THE BIG JOB!
DO 100 JN=LNUM(1),LNUM(2)
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
CONTINUE
DO 200 K = 1, KE
YNG(K) = YOUT(L, K)
ZNG(K) = ZOUT(L, K)
ENDO

IF(INTRAIL.EQ.2) THEN
DO 200 K = 1, KE
YNG(K) = YOUT(L, K)
ZNG(K) = ZOUT(L, K)
ENDO
ELSE
DO 200 K = 1, NPAIR(1, KE)
YINT(K-I) = YOUT(L, K)
ZINT(K-I) = ZOUT(L, K)
ENDO
ENDIF

IF(YOUT(L, NPAIR(1, KE)) .LT. YNAC(KE-1)) THEN
YNAC(KE-1) = YOUT(L, K)
ZAC(KE-1) = ZINT(K-I)
ENDIF

IF(YOUT(L, NPAIR(1, KE)) .GT. ZNAC(KE)) THEN
ZNAC(KE) = ZOUT(L, K)
YINT(K-I) = YOUT(L, K)
ZINT(K-I) = ZOUT(L, K)
ENDIF

CALL LININT(YNAC(NE-I), YNAC(NE), ZNAC(NE-I), ZNAC(NE), YINT(K), ZINT(K-I))

DO = NPAIR(1, KE) - 1
DO K = 1, NPAIR(1, KE)
YINT(K-I) = YOUT(L, K)
ZINT(K-I) = ZOUT(L, K)
ENDDO

CALL LININT(YNAC(NE-I), YNAC(NE), ZNAC(NE-I), ZNAC(NE), YINT(K), ZINT(K-I))

DO = NPAIR(1, KE) - 1
DO K = 1, NPAIR(1, KE)
YINT(K-I) = YOUT(L, K)
ZINT(K-I) = ZOUT(L, K)
ENDDO

ENDC

CONTINUE
DO = NPAIR(1, KE) - 1
DO K = 1, NPAIR(1, KE)
YINT(K-I) = YOUT(L, K)
ZINT(K-I) = ZOUT(L, K)
ENDDO

ENDC

CONTINUE
DO = NPAIR(1, KE) - 1
DO K = 1, NPAIR(1, KE)
YINT(K-I) = YOUT(L, K)
ZINT(K-I) = ZOUT(L, K)
ENDDO

ENDC

CONTINUE
DO = NPAIR(1, KE) - 1
DO K = 1, NPAIR(1, KE)
YINT(K-I) = YOUT(L, K)
ZINT(K-I) = ZOUT(L, K)
ENDDO

ENDC

CONTINUE
DO = NPAIR(1, KE) - 1
DO K = 1, NPAIR(1, KE)
YINT(K-I) = YOUT(L, K)
ZINT(K-I) = ZOUT(L, K)
ENDDO

ENDC
DO 600 I=NP1, NPTS
   IF (ZOUT(L,E) .LE. ZNAC(NFRIS)) THEN
     NZ2 = L
     CALL LININT(ZOUT(L,NZ2),ZOUT(L,NZ2-1),YOUT(L,NZ2),
       YOUT(L,NZ2-1),ZNAC(NFRIS),Y2)
   620 CONTINUE
   GOTO 605
END IF
600 CONTINUE

C Store the wingtip grid
DO 620 I=NP1, NNI
   YNG(NPWN+I) = YOUT(L,I)
   ZNG(NPWN+I) = ZOUT(L,I)
620 CONTINUE

C Redistribute the points above the trailing edge
NJ = NNI-NN1+1
TW(1) = Y1
SWE(1) = Z1
TW(NJ) = YT
SWE(NJ) = ZT
DO E=1, NJ
   YOUT(E) = Y2(E)
   ZOUT(E) = Z2(E)
ENDDO

C Store the wingroot grid
DO E=NP2, NPTS
   YNG(E) = YOUT(E)
   ZNG(E) = ZOUT(E)
ENDDO

700 CONTINUE

IF(IPRINT .NE. 0)THEN
   WRITE(IPRINT,E=1,NPWN),
     YNG(E),ZNG(E),E=1,NPWN
   ENDIF
750 CONTINUE

800 CONTINUE
IF(LNUM(1),EQ.LNUM(3),.AND. LNUM(2),EQ.LNUM(4)) THEN
   NPTS = NPTS - MC
ENDIF
RETURN
END
SUBROUTINE NACGRID
include "samgrid.com"
C This subroutine read the nacelle grid and combine it with the
C wing grid.
C
COMMON /ENG/ XENG(2,NP1),YENG(2,NP1),ZENG(1,NP1,NP1)
& XENC(2,NP1),YENC(2,NP1),ZENC(1,NP1,NP1)
COMMON /NACX/ XNAC(NP),YNAC(NP),ZNAC(NP,NP)
COMMON /NACF/ XNACP(2),YNACP(2),ZNACP(2)
COMMON /NACFL/ XNACFL,NACP,LNUM(4)
C (XENG,YENG,ZENG) Coordinates of engine
C NNAC(*) Number of stations in nacelle
C NNAC(+) Number of points in each station
C (1,+,*)) inner nacelle, (2,+,*)) outer nacelle
C
C Read the nacelles geometry:
C
C Inner nacelle geometry
CALL NACIN
XIN = NNAC(1)
XIN + XNAC(1) + XNAC
DO 100 L=1,NMAC
XENG(1,L) = XNAC(L)
DO 10 L=1,NMAC
XENG(1,L) = XNAC(L)
100 CONTINUE
C
C Outer nacelle geometry
CALL NACIN
XOUT1 = XNAC(1)
XOUT = XNAC(NMAC)
NNAC(2) = NNAC
NNAC(2) = NNACP(2)
DO 190 L=1,NMAC
XENG(3,L) = XNAC(L)
DO 190 L=1,NMAC
XENG(3,L) = XNAC(L)
190 CONTINUE
C
C In a case of 1 nacelles, three zone will be made.
C
C NETING Station(s) will be added to the wing at inlet and outlet
C of the nacelle
C
C MOUNT Mount the nacelle under the wing and/or wake line
C
JN = 1 Inner nacelle
JN = 2 Outer nacelle
C
C The first zone, only one nacelle appears
WRITE(*,*) 'zone 1'
JN = 1
CALL NETING(LNUM,XIN,XOUT1)
CALL MOUNT(JN,LNUM,21)
LNUM(3) = LNUM(1)
LNUM(4) = LNUM(2)
C
C The second zone consists two nacelles appear
WRITE(*,*) 'zone 2'
JN = 1
CALL NETING(LNUM,XIN1,XOUT1)
CALL MOUNT(JN,LNUM,0)
LNUM(3) = LNUM(1)
LNUM(4) = LNUM(2)
C
C The third zone, only one nacelle appears
WRITE(*,*) 'zone 3'
JN = 1
CALL NETING(LNUM,XIN2,XOUT2)
CALL MOUNT(JN,LNUM,22)
LNUM(3) = LNUM(1)
LNUM(4) = LNUM(2)
C
C The fourth zone, only one nacelle appears
WRITE(*,*) 'zone 4'
JN = 1
CALL NETING(LNUM,XIN2,XOUT2)
CALL MOUNT(JN,LNUM,23)
C
RETURN
END
SUBROUTINE MWING(LNUM,XX,XX)
  include "agrld.com"
  DIMENSION XWNG(NPI,NPI),YWNG(NPI,NPI),ZWNG(NPI,NPI)
  DIMENSION LNUM(4)
  
  DO 10 L=1,NSEC
    DO 10 K=1,NPTS
      XWNG(L,K) = XOUT(L,K)
      YWNG(L,K) = YOUT(L,K)
      ZWNG(L,K) = ZOUT(L,K)
  10 CONTINUE
  XWNG(1,NSEC) = XX
  YWNG(1,NSEC) = YY
  ZWNG(1,NSEC) = ZZ
  DO 20 L=1,NSEC
    IF(NN.EQ.1) XX = XNW(L)
    IF(NN.EQ.2) XX = XNE(L)
    
  20 CONTINUE
  CALL MWNT(XX,XWNG,LNUM,YY,YWNG,LNUM,ZZ,ZWNG,LNUM)
  
END

C -Rewrite the coordinate of the wing
DO 10 L=1,NSEC
  DO 10 K=1,NPTS
    XWNG(L,K) = XOUT(L,K)
    YWNG(L,K) = YOUT(L,K)
    ZWNG(L,K) = ZOUT(L,K)
  10 CONTINUE
  XWNG(1,NSEC) = XX
  YWNG(1,NSEC) = YY
  ZWNG(1,NSEC) = ZZ
  
C -Find out where the x-location of start and end of nsecle
DO 55 NN=1,l
  IF(MN .EQ. 1) XN=INSTRT
  IF(MN .EQ. 2) XN=INEND
  IF(ABS(XWNG(LN,1)-XX).LT.1.E-7) THEN
    IF(LN.EQ.1) NN = NN+1
    ENDDO
    LNUM(MN) = LN
    GO TO 55
  ENDIF
C -Create an extra station in the wing
IF(XWNG(LW,1).LT.XX) THEN
  XNW = XX
  LNW = LN
  DO 35 K=1,NPTS
    XWNG(K) = XX
    YWNG(K) = YOUT(K)
    ZWNG(K) = ZOUT(K)
  35 CONTINUE
  
C -If we are in wake, make sure top pts intersect the bottom pts
DO 24 L=1-LW,NSEC
  DO 24 K=1,NPTS
    IF(ABS(Y(LW,K)-YOUT(K)).LT.1.E-6 .AND.
     OR ABS(Z(LW,K)-ZOUT(K)).LT.1.E-6) THEN
      Y(LW,K) = YOUT(K)
      Z(LW,K) = ZOUT(K)
  24 CONTINUE
  
C -Put the rest of the station into (x,y,z)
DO 30 L=1-LW,NSEC
  DO 30 K=1-NPTS
    X(K) = XWNG(L,K)
    Y(K) = YOUT(L,K)
    Z(K) = ZOUT(L,K)
  30 CONTINUE
  
C -Rewrite the coordinate of the wing
DO 55 NW=1-l
  IF(MN .EQ. 1) XN=NSECRT
  IF(MN .EQ. 2) XN=NSEND
  IF(ABS(XWNG(LW,1)-XX).LT.1.E-7) THEN
    IF(NW.EQ.1) NW = NW+1
    ENDDO
    LNUM(MN) = LN
    GO TO 55
  ENDIF
  
C -Add two stations in the wing, these two stations lie exactly on
C INSTRT and INEND.
DO 55 NN=1-l
  IF(NN .EQ. 1) XX=INSTRT
  IF(NN .EQ. 2) XX=INEND
  IF(ABS(XWNG(LN,1)-XX).LT.1.E-7) THEN
    IF(NN.EQ.1) NN = NN+1
    ENDDO
    LNUM(MN) = LN
    GO TO 55
  ENDIF
C -The wing section is very close to nsecle's station (INSTRT or INEND)
DO 910 L=1,NSEC
  DO 910 K=1,NPTS
    XWNG(L,K) = XOUT(L,K)
    YWNG(L,K) = YOUT(L,K)
    ZWNG(L,K) = ZOUT(L,K)
  910 CONTINUE
  
RETURN
**UNIX FORTRAN Program**

**SOURCE PROGRAM**

```
#include "sgrid.com"

DIMENSION D1(NP), D2(NP), S(NP)

KTIP = (KDIM+1)/2

C From the nose to the leading edge
C .. from station 1 to M1
C
C First point of the nose
DO 12 K=1,KDIM
X1(K) = XIN(K)
Y1(K) = YIN(K,1)
Z1(K) = ZIN(K,1)
12 CONTINUE
C
C Loop for all stations, from station 1 to station M1
DO 50 I=M1,2

L = 1
XILOCAL = XIN(N)
Store the input to dummy array
DO 31 I=1,NFP
YINC(I) = YIN(M,I)
ZINC(I) = ZIN(M,I)
31 CONTINUE
C
C TREF is value of the first point of the wing
C TREF is the corresponding index of each station.
TREF = TOUT(1,NP1)
C
C OPTIONS -
C Boeing Baseline Configuration
DO 43 I=1,NFP
IF(TINC(I).GE.TREF) THEN
IFREF = I
GOTO 44
ENDIF
43 CONTINUE
C
C Langley's Low-Boom Configuration
C KREF = 31
C
C Lower part of the nose (-Y to TREF)
KE=1
KE=KREF
KE=KE+1
DO 74 I=1,KE
EK = KE+1
DI(EK) = TINC(EK)
D2(EK) = ZINC(EK)
74 CONTINUE
CALL DISTARC(D1,D2,1,YNEW, ZDIST, KTIP, PNBOT, 1)
DO 20 I=1,KDIM
Y(L,I) = YNEW(I)
Z(L,I) = ZDIST(I)
X(L) = XLOCAL
20 CONTINUE
50 CONTINUE
RETURN
END
```
This is the main subroutine to redistribute the points from streamwise cut to streamwise cut. When there is "streamwise-SIMP", it means the output is for SIMP code.

PARAMETER(INT=600)
DIMENSION S(INT)
NUM = number of point in the surface;
NL = number of point in the lower surface;
NC = number of spanwise sections.
X = X of pt in planes, y = Y of pt in planes.

IF(INTEQ) THEN
WHITE(*+) SUB RESID = INT is less than ETIP.
STOP
ENDIF

IF((ETIP EQ 2)) THEN
NUL = NUL
ELSE
ENDIF

The streamwise distance passes the leading edge.

IF(LONAL.GT.EL(1)) THEN
3-station is at the wing tip.
IF(LONAL.LE.(EL(1)+CHORD(1))) THEN
Should call WINGNWK if wake is contained in this station.
CALL WINGNWK(LONAL,EL(1)+CHORD(1));
GOTO 200
ENDIF
ENDIF

Streamwise distance is between the leading edge and the trailing edge.

DO 60 M=1,NC
DO 60 N=2,NL
IF(XBASE(I,IFLAT,M-1)+EL.1.E-7)
ABS(XBASE(I,IFLAT,M-1)+EL.I.E-7)
X1 = XBASE(I,IFLAT,M-1)
X2 = XBASE(I,IFLAT,M-1)
Y1 = YBASE(I,IFLAT,M-1)
Y2 = YBASE(I,IFLAT,M-1)
Z1 = ZBASE(I,IFLAT,M-1)
Z2 = ZBASE(I,IFLAT,M-1)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
YINT(M) = YY
ZINT(M) = ZZ
GOTO 200
ENDIF
ENDIF

ELSE

The X-station passes the wing tip, should have wake.
CALL WINGNWK(LONAL,EL(1),IFLAT);
GOTO 200
ENDIF
ENDIF

IF(LONAL.LE.EL(1)) THEN
Should call WINGNWK if wake is contained in this station.
IF(LONAL.GT.EL(1)+CHORD(1)) THEN
CALL WINGNWK(LONAL,EL(1)+CHORD(1));
GOTO 200
ENDIF
ENDIF

X = 0
DO 160 I = 1,NUL
ET = ET + 1

Create a point at the root.

IF(ABS(XBASE(I,IFLAT,1)+EL.LONAL)) THEN
X1 = XBASE(I,IFLAT,1)
X2 = XBASE(I,IFLAT,1)
Y1 = YBASE(I,IFLAT,1)
Y2 = YBASE(I,IFLAT,1)
Z1 = ZBASE(I,IFLAT,1)
Z2 = ZBASE(I,IFLAT,1)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
YINT(ET) = YY
ZINT(ET) = ZZ
GOTO 200
ENDIF
ENDIF

IF(ABS(XL)+2,NC)
XL = LONAL-XBASE(I,IFLAT,M-1)
X = LONAL-XBASE(I,IFLAT,M-1)
IF((IXL.LT.LT.0)) THEN
X1 = XBASE(I,IFLAT,M-1)
X2 = XBASE(I,IFLAT,M-1)
Y1 = YBASE(I,IFLAT,M-1)
Y2 = YBASE(I,IFLAT,M-1)
Z1 = ZBASE(I,IFLAT,M-1)
Z2 = ZBASE(I,IFLAT,M-1)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
YINT(ET) = YY
ZINT(ET) = ZZ
GOTO 200
ELSEIF(ABS(XLE + X) | LT. 1.E-7) THEN
IF(ABS(XL) | LT. 1.E-7) THEN

1663  YINT(KT) = YBASE(1,IFLAT,M-1)
1664  ZINT(KT) = ZBASE(1,IFLAT,M-1)
1665  ELSE
1666  YINT(KT) = YBASE(1,IFLAT,M)
1667  ZINT(KT) = ZBASE(1,IFLAT,M)
1668  ENDIF
1669  GOTO 160
1670  ENDIF
1671  CONTINUE
1672  CONTINUE
1673  CONTINUE
1674  C
1675  CONTINUE
1676  C
1677  C Right now YINT and ZINT is from wing tip to the root, in order to
1678  C the cubic spline program, need from root to tip.
1679  C
1680  DO 220 KT = 1,ET
1681  S(KT) = YINT(KT)
1682  220 CONTINUE
1683  DO 230 KT = 1,ET
1684  TINT(KT) = S(ET-KT+1)
1685  230 CONTINUE
1686  DO 240 KT = 1,ET
1687  S(KT) = ZINT(KT)
1688  240 CONTINUE
1689  DO 250 ET = 1,ET
1690  ZINT(KT) = S(KT-ET+1)
1691  250 CONTINUE
1692  C
1693  C OPTIONS :
1694  C Distribute & Coordinates (pipe around direction) from root to
1695  C leading edge and back (xdist).
1696  C If X = 1, grid points will cluster near the wing tip, =0 near the root.
1697  C For NSEW, X = 0 / For Bowing, X = 1
1698  C
1699  IF(L_N) THEN  CALL DSPINT(YINT,INT,KT,XX,XX2,ZINT,KT)
1700  IF(L_N) THEN  CALL CSPLINE(YINT,INT,KT,XX,XX2,ZINT,KT)
1701  C
1702  RETURN
1703  END
SUBROUTINE SUBTRACGRID(NPL1, X, Y, Z, NPI, NSEC, KDIM)

DIMENSION X(NPI), Y(NPI), Z(NPI)

This subroutine allows us to subtract a grid line NPL1 in streamwise section, and the new dimension is NSEC again.

Renumber the late stations

NSEC = NSEC-1

DO 10 L = NPL1, NSEC
    X(L) = X(L+1)
    Y(L) = Y(L+1, I)
    Z(L) = Z(L+1, I)

10 CONTINUE

CONTINUE

RETURN

END
DO 290 I=1,EDIM
   Y(I,E)=YINT(E)
   Z(I,E)=ZINT(E)
290 CONTINUE

DO 293 E=1,NBP
   NPB=(NPP+1)/2
   YINT(E)=YIN(M,E)
   ZINT(E)=ZIN(M,E)
   CONTINUE

CALL DISTANCE(YINT,ZINT,NBP,NPP,YDIST,KTPP,0.05,1)
293 CONTINUE

DO 295 E=1,NPP
   ZINT(E)=ZIN(M,NPP+1-E)
   CONTINUE

CALL DISTANCE(YINT,ZINT,NPP,NPP,YDIST,KTPP,0.05,0)
295 CONTINUE

DO 297 E=1,NPP
   YINT(E)=YIN(M,NPP+1-E)
   CONTINUE

RETURN
END
SUBROUTINE WGRID

This subroutine reads the fuselage grid and combines it with the wing grid.

DIMENSION D1(NP1), D2(NP1), E(NP1)

COMMON /REF/ BROOT, STIP, ARCORE

Read surfgrid dimensions

XIN, TH, ZIN - read in grid
X, Y, Z - output grid
THETA, ZETA, ZETA - dummy vectors
EDIM = # of points of the whole body in the warp-around direction
E1 = E2 in the case of the nose top part (circumferential direction)
F1 = FL in the tail part (circumferential direction)

SCALE = Scale factor for the wing-body
NAMELIST /REAL/ EDIM, E1, E2, F1, F2, T1, T2, AZERIFORM, BOFF

SCALE = Number points along the trailing wake

Read the fuselage geometry

CALL FUSE

Take this initial grid

1) convert it into area distribution
2) add shape functions
3) convert it back to grid

CALL SURF

BL = XIN(NP)
ECOS = E1*BL
ETIP = (EDIM+1)/2

Find number of streamwise stations. We keep the stations of the fuselage; As to the common area of the wing and fuselage we use the stations of the wing.

M1 is the station the nose ends
M2 is the station the wing ends
M1 = 0
M2 = 0

IF (ABS(XIN(1)) GT XOUT(1,1)) .AND. XIN(NP), GT XOUT(NSEC,1)
M1 = M + 1

IF (M1 EQ 0) THEN
DO 20 M = 1, NP
20 CONTINUE

IF (M2 EQ 0) THEN
DO 30 M = 1, NP
30 CONTINUE

ENDIF

ENDIF

We divide the configuration into four parts:

The nose, the wing-body, the tail, and the extended wake. And for each part we distribute points for the lower and upper part separately.

From the nose to the leading edge
CALL NOSE(WBROF, WBTOP)

From the leading edge to the trailing edge (the whole wing)
CALL WBNAME
CALL WING_BODY

CALL TAIL(-1, WAC)

Redistribute each streamwise station, so that the grid is closer near the wing tip.

IF (CUSTER, GT 0) CALL EQSPACE

Redistribute the streamwise stations, so that it is equally spaced.

IF (CUSTERN, GT 0) CALL EQSPACE

Add the extended wake

L2 = L - 1
XLAST = X(L2) + BL*BWAKE
XTOLE = XLAST - X(L2)
We might like to scale the whole thing by a reference length

Boeing Baseline Configuration
Before writing out the grid, this Boeing wing needed to be fixed
for some stupid reasons.

Langley's Low-Boom Configuration

Write out new surfgrid in plotid format.

Write the wing-body grid \((x,y,z)\) to \((XOUT,YOUT,ZOUT)\)
SUBROUTINE WFMATCH
CALL LININT(X1, X2, Z1, Z2, XI, XINT(3), ZZ)
RETURN
END
SUBROUTINE WING_BODY
include "agrid.com"
DIMENSION DI(NP1),D2(NP1)
COMMON /REF/ EGRID,STIP,ARCOR
!

The wing-body section, itself has NSEC sections

LW is the section index for YIN,XIN

LD = NL-1
LE = NL + NSEC -1
DO 200 L=LD,LE
!

Calculate the points (YINT,ZINT) on the fuselage at XLOCAL.

0.5 sections is this section, each section has same number

of points in rope-around direction.

DO 10 NW=M,NE
IF(YIN(M).GE.XLOCAL).OR.

ABS(YIN(M)-XLOCAL).LE.1.E-7) THEN

MW = M
GO TO 15
ENDIF
10 CONTINUE

There are NPTS points in the wing area, need to check out

every point is in the fuselage section.

EI is the inters pt. / the fuselage & wing at bottom in old grid

EJ is the inters pt. / the fuselage & wing at top in old grid

MID is the inters pt. / the fuselage & wing in bottom in new grid

There are NT points from 1 to NT

!MW=(MW+NE)/2
MID=(EDIM-NPTS)/2
!

Find EI

DO 60 E=ET,-1,-1

IF(YINT(E).LT.YOUT(LW,1) .AND. YINT(E).NE.YINT(E-1)) THEN

EI = E
GO TO 62
ENDIF
60 CONTINUE

Find EJ

DO 70 E=ET,+1,1

IF(YINT(E).GE.YOUT(LW,NPTS) .AND. YINT(E).NE.YINT(E+1)) THEN

EJ = E
GO TO 72
ENDIF
70 CONTINUE

For arrow-wing type, EI needed to be relocated

IF(ABS(YOUT(LW,1)-YOUT(LW,NPTS)).LE.1.E-7) THEN

EI = EI + ERS
EJ = EI
YINT(EJ)=YINT(EI)
ZINT(EJ)=ZINT(EI)
ENDIF

Calculate the points at the bottom (from seg. Y to Y at Xlocal)

EI=EI
EX=EX
DO 82 E=EX,EX

EE=E+E
D1(EE) = YINT(E)
D2(EE) = ZINT(E)
82 CONTINUE

CALL DISTARC(D1,D2,EX,YNEW(1),ZDIST,MID,-10.,1)
!

DO 100 K=I,MID

Y(K+MID,NPTS+1) = YNEW(K)
Y(K+MID,NPTS) = IDIST(K)
X(K+MID) = XLOCAL
100 CONTINUE

! For arrow-wing type, make sure wake points ok
! IF(ABS(YOUT(LW,1)-YOUT(LW,NPTS)).LE.1.E-7) THEN
! YOUT(LW,1) = YOUT(LW,1)
! YOUT(LW,NPTS) = YOUT(LW,NPTS)
! ENDIF

The points of the wing section

DO 110 K=1,NPTS

Y(K,MID+I) = YOUT(LW,K)
Y(K,MID+2) = YOUT(LW,K)
X(K) = XOUT(LW,K)
110 CONTINUE

Calculate the points at the top

EX=EX
DO 115 E=EX,EX

EE=E+E
D1(EE) = YINT(E)
D2(EE) = ZINT(E)
115 CONTINUE

CALL DISTARC(D1,D2,EX,YNEW(1),ZDIST,MID,-10.,0)
!

DO 120 K=I,MID

Y(K+MID,NPTS+1) = YOUT(K)
Y(K,MID+2) = YOUT(K)
X(K) = XOUT(K)
120 CONTINUE

! For arrow-wing type, make sure wake points ok
! IF(ABS(YOUT(LW,1)-YOUT(LW,NPTS)).LE.1.E-7) THEN
! YOUT(LW,1) = YOUT(LW,1)
! YOUT(LW,NPTS) = YOUT(LW,NPTS)
! ENDIF

UN

2252 \( Z(L, M\text{ID}) = Z(L, M\text{ID} + N\text{PTS} + 1) \)

2253 ENDIF

2254 C

2255 C Fill the unsmooth part by FILT

2256 C First of all, find the set of points needed to be rearrange

2257 DO 125 K = 1, EDIM

2258 \( YINT(K) = Y(L, K) \)

2259 \( ZINT(K) = Z(L, K) \)

2260 125 CONTINUE

2261 125 CONTINUE

2262 RFILT = RFIL

2263 IF (X(L) .GE. XSTART .AND. X(L) .LE. XOFF)

2264 \& CALL FILT(YINT, ZINT, EDIM, MB1, MB2, MT1, MT2, RFILT)

2265 DO 140 K = 1, EDIM

2266 \( Y(L, K) = YINT(K) \)

2267 \( Z(L, K) = ZINT(K) \)

2268 140 CONTINUE

2269 CONTINUE

2270 CONTINUE

2271 RETURN

2272 END
SUBROUTINE WINGHARE(XLOCAL,XT,NUL,IFLAT)
include "agrid.com"

C This subroutine creates the data when the station is in the plane.
C where some points are on the wing, some are not.
C
At the x=station, we put 10 points in the x=order.

C ET = 0.
IF(ARDING .GT. 0.) THEN
C The wing is swept backwards
DO 75 J=2,NC
J1=J-1
J2=J
IF(LOCAL .LT.XLE(J)+CHORD(J)) GOTO 75
X1=XLE(J)+CHORD(J)
X2=XLE(J)+CHORD(J)
S1=BASE(NUL,IFLAT,J)
S2=BASE(NUL,IFLAT,J)
ZTIP=S1+(XLOCAL-X1)*(X2-X1)
IFLAT=ZTIP

C
KLEG = NC
DO MUN=2,NC
IF(LOCAL .LT.BASE(1,IFLAT,MUN)) THEN
KLEG = LEAD-1
1
X1 = BASE(1,IFLAT,MUN-1)
X2 = BASE(1,IFLAT,MUN-1)
Y1 = BASE(1,IFLAT,MUN-1)
Z1 = BASE(1,IFLAT,MUN-1)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
YINT(1) = YY
ZINT(1) = ZZ
GOTO 40
ENDIF
ENDDO
75 CONTINUE

C ELSE
DO 65 M=NLEDG,32,-1
DO 64 I=J,L
TF(I)=LOCAL.LE.XBASE(I,IFLAT,M)
XI = XBASE(I,IFLAT,M)
X2 = XBASE(I,IFLAT,M)
Y1 = YBASE(I,IFLAT,M)
Y2 = YBASE(I,IFLAT,M)
Z1 = ZBASE(I,IFLAT,M)
Z2 = ZBASE(I,IFLAT,M)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
IF(NLEDG .EQ.NC) THEN
M=M-NLEDG-M-1
ELSE
M=M-NLEDG-M-2
ENDIF
YINT(M) = YY
ZINT(M) = ZZ
GOTO 65
ENDIF
64 CONTINUE
65 CONTINUE
C ELSE
ZROOT = ZBASE(NUL,IFLAT,31)
Y1=YBASE(NUL,IFLAT,31)
Y2=YBASE(NUL,IFLAT,32)
Z1=ZBASE(NUL,IFLAT,31)
Z2=ZBASE(NUL,IFLAT,32)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
DZ=(ZFLAT-ZROOT)/9.
DO 70 M=1,10
IF(NLEDG .EQ.NC) THEN
M=NLEDG-J2+1-M
ELSE
M=NLEDG-J2+1-M+1
ENDIF
YINT(M) = YY+FLOAT(M-1)*DZ
CALL LININT(ZROOT,ZZ,YOUT(L-1),YY,ZINT(M),YYY)
70 CONTINUE
IF(NLEDG .EQ.NC) THEN
ET = (NLEDG-J21+1)+1
ELSE
ET = (NLEDG-J21)+1+1
ENDIF
GOTO 200
75 CONTINUE
C ELSE
DO 105 J=2,NC
J1=J-1
J2=J
IF(LOCAL .LT.XLE(J)+CHORD(J)) GOTO 105
X1=XLE(J)+CHORD(J)
X2=XLE(J)+CHORD(J)
S1=BASE(NUL,IFLAT,J)
S2=BASE(NUL,IFLAT,J)
ZTIP=S1+(XLOCAL-X1)*(X2-X1)
IFLAT=ZTIP

CHEUNG-SIMP
ZTIP=BASE(NUL,IFLAT,M)
CHEUNG
ZTIP=BASE(NUL,IFLAT,M)
C CHEUNG
ADJ 10 points for the wake.
Y1=BASE(NUL,IFLAT,J)
Y2=BASE(NUL,IFLAT,J)
Y3=BASE(NUL,IFLAT,J)
Y4=BASE(NUL,IFLAT,J)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
DO = (ZTIP-FLAT)/9.
DO 80 M=1,10
ZINT(M) = ZTIP-10*(M-1)*DZ
CALL LININT(ZTIP,ZINT(M),YOUT(L-1),YY,ZINT(M),YYY)
80 CONTINUE
CHEUNG
ZTIP=BASE(NUL,IFLAT,M)
C CHEUNG
CHEUNG
A 10 points for the wake.
Y1=BASE(NUL,IFLAT,J)
Y2=BASE(NUL,IFLAT,J)
Y3=BASE(NUL,IFLAT,J)
Y4=BASE(NUL,IFLAT,J)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
DO = (ZTIP-FLAT)/9.
DO 80 M=1,10
ZINT(M) = ZTIP-10*(M-1)*DZ
CALL LININT(ZTIP,ZINT(M),YOUT(L-1),YY,ZINT(M),YYY)
80 CONTINUE
CHEUNG
ZTIP=BASE(NUL,IFLAT,M)
C CHEUNG
<table>
<thead>
<tr>
<th>LINE #</th>
<th>SOURCE TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2392</td>
<td>TINT(M) = YY</td>
</tr>
<tr>
<td>2393</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>2394</td>
<td>DO 100 M=1,1,-1</td>
</tr>
<tr>
<td>2395</td>
<td>DO 90 I=2,NUL</td>
</tr>
<tr>
<td>2396</td>
<td>IF(XBASE(I,IFLAT,M) .LE. XLOCAL) THEN</td>
</tr>
<tr>
<td>2397</td>
<td>X1 = XBASE(I-1,IFLAT,M)</td>
</tr>
<tr>
<td>2398</td>
<td>X2 = XBASE(I,IFLAT,M)</td>
</tr>
<tr>
<td>2399</td>
<td>Y1 = YBASE(I-1,IFLAT,M)</td>
</tr>
<tr>
<td>2400</td>
<td>Y2 = YBASE(I,IFLAT,M)</td>
</tr>
<tr>
<td>2401</td>
<td>Z1 = ZBASE(I-1,IFLAT,M)</td>
</tr>
<tr>
<td>2402</td>
<td>Z2 = ZBASE(I,IFLAT,M)</td>
</tr>
<tr>
<td>2403</td>
<td>CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)</td>
</tr>
<tr>
<td>2404</td>
<td>CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)</td>
</tr>
<tr>
<td>2405</td>
<td>TINT(I) = TINT(I) + YY</td>
</tr>
<tr>
<td>2406</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>2407</td>
<td>GOTO 100</td>
</tr>
<tr>
<td>2408</td>
<td>ENDIF</td>
</tr>
<tr>
<td>2409</td>
<td>90 CONTINUE</td>
</tr>
<tr>
<td>2410</td>
<td>100 CONTINUE</td>
</tr>
<tr>
<td>2411</td>
<td>X1 = X1 + X1</td>
</tr>
<tr>
<td>2412</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>2413</td>
<td>GOTO 200</td>
</tr>
<tr>
<td>2414</td>
<td>105 CONTINUE</td>
</tr>
<tr>
<td>2415</td>
<td>C ENDIF</td>
</tr>
<tr>
<td>2416</td>
<td>200 CONTINUE</td>
</tr>
<tr>
<td>2417</td>
<td>RETURN</td>
</tr>
<tr>
<td>2418</td>
<td>END</td>
</tr>
</tbody>
</table>
SUBROUTINE WININ

include "sgrid.com"

C NC = # of sections in the spanwise direction
C XBASE(I) = X value of Ith section
C YBASE(I) = Y value of Ith section
C XBASE(I) = chord length of the Ith section
C Note: the end points of upper and lower sections are same physical pts

C MACC2 configuration

READ(10,930) NC,NU
NL = NU

K=1
CONTINUE

READ(10,940) XBASE(1,1),YBASE(1,1),EBASE(1,1)

READ(10,950) XBASE(1,1),YBASE(1,1),EBASE(1,1)

READ(10,960) XBASE(1,1),YBASE(1,1),EBASE(1,1)

READ(10,970) XBASE(1,1),YBASE(1,1),EBASE(1,1)

READ(10,980) XBASE(1,1),YBASE(1,1),EBASE(1,1)

FORMAT(2SX, I 5,9X, Z 5/)
FORMAT(I )
FORMAT(2X, I 5,9X, Z 5/)
FORMAT(I )
FORMAT(2X, I 5,9X, Z 5/)
FORMAT(I )
RETURN
END
SUBROUTINE FUSEIN

include 'egrid.com'

DIMENSION YT(NPI),ZT(NPI)

NF = # of section in the fuselage
NFP = # of points in nth section

Read the fuselage geometry

M = M + 1

Call EXPAND(YT,EN,NFP,YT,NFP,0.,0.,0.)

DO 110 K = 1,NP
    IF(M.LT.NM) GOTO 40

Write the fuselage geometry into PLOTID Planar format.

WRITE(51),XK,EN,M

DO 400 L = 1,EN
    WRITE(51),XIN(L),XK,EN,M

400       FORMAT(1X)

DO 800 L = EN + 1,EN
    WRITE(51),ZIN(L),YK,EN,M

800       FORMAT(1X)

RETURN

END
SUBROUTINE SACIN
include "sgrid.com"
DIMENSION TT(NPI),ET(NPI)
COMMON /SACIN/ ENAC(NPI),THAC(NPI,NPI),ZHAC(NPI,NPI)
COMMON /NDIM/ NAC,NACF,NACP,INAC(4)
C
C  NAC = # of section in the sacell
C  NACF = # of points in nth section
C
C Read the sacelle geometry
READ(10,810)
READ(10,820)
READ(10,830)
C
M=0
40 CONTINUE
C
READ(10,840)
M=M+1
C
DO 100 E=1,NACF
READ(10,850) ENAC(E),THAC(E,NPI),ZHAC(E,NPI)
100 CONTINUE
C
DO 110 B=1,NACBP

C Write the sacelle geometry into PILOT3D Planar format
C
C  B=1
C  E=1
C  WR(E),SW,B,SW,M
C  DO 800 L=1,M
C     WRITE(20)[(THAC(L,E),E=1,EE),]}
C     [ZAC(L,E),E=1,EE]}
C
C800 CONTINUE
C
C Call flush (32)
C
C810 FORMAT(1X)
C820 FORMAT(1X)
C830 FORMAT(1X,15.E15/)
C840 FORMAT(1X)
C850 FORMAT(1X,3F16.7)
C
RETURN
END
SUBROUTINE WINGMAKER

C
C include 'sgrid.com'
C
C This program generates a 'clipped' delta wing with no twist
C based on airfoil coordinates read in from fort. 90. To use as
C part of samgrid, WINGMA is not necessary (nor is VARINSE).
C
C Written by Donovan L. Mathias
C
C July 1982

C
C Whenever possible the same variables are used as in samgrid.f
C
C Fort. 90 airfoil coordinates
C
C Description of the wing

C Declarations

C REAL XTE(LE),SLOPE1,SLOPE2,SCALE,XAF(150)
C REAL YU(150),YI(150)
C REAL SPAN

C INTEGER L,I,J,K

C Initialization

C read(77,*)NC
C read(77,*)XLE(I),XTE(I)
C read(77,*)YBASE(I,1,1),YTE(I)
C read(77,*)SCALE(1,1,1),BASE(1,1,1)

C Need in airfoil coordinates (L is # of X coords.)
C
C READ(90,*)
C READ(90,*)NU
C READ(90,*)L
C L=NU
C DO I=1,NU
C READ(90,19)XAF(I),YU(I),YI(I)
C END DO

C format(3x, f9.7, f9.7, f9.7)

C Establish f distance (Spanwise)
C SPAN = BASE(1,1,NC) - BASE(1,1,1)

C DO K=0,NC-1
C DO I=1,NU
C BASE(1,1,K+1) = (K*SPAN/(NC-1))
C BASE(1,2,K+1) = (K*SPAN/(NC-1))
C END DO
C END DO

C Establish sweep (1 FOR LE, 2 FOR TE)
C
C SLOPE1 = (XLE(1,1)-XLE(NC))/BASE(1,1,1)-BASE(1,1,1)
C SLOPE2 = (XTE(1,1)-XTE(NC))/BASE(1,1,1)-BASE(1,1,1)

C Generate leading and trailing edges
C DO K=1,NC
C XLE(K) = XLE(1) + SLOPE1*BASE(1,1,K)
C XTE(K) = XTE(1) + SLOPE2*BASE(1,1,K)
C END DO

C Distribute grid points

C DO K=1,NC
C SCALE = XTE(K)-XLE(K)
C DO I=1,NU
C BASE(1,1,K) = XLE(K) + SCALE*XAF(I)
C BASE(1,2,K) = YU(I)*SCALE
C BASE(1,3,K) = XLE(K) + SCALE*XAF(I)
C BASE(1,4,K) = YI(I)*SCALE
C END DO
C END DO

C Return values to original names
C DO K=1,NC
C CHORD(K) = ABS(XLE(K)-XTE(K))
C YLE(K) = YBASE(1,1,K)
C END DO

RETURN
END
Manual of PVM

Samson Cheung

Merritt Smith
# Table of Contents

1 Preface.................................................................................. 1

2 Introduction
   Software Package................................................................. 2
   Definitions ........................................................................... 2
   Structure of PVM................................................................. 3
   Directory Setup ..................................................................... 3

3 Programming Concepts
   Master and Slaves ............................................................... 4
   Single Program-Multiple Data (SPMD) ................................. 6

4 PVM Daemon
   Console Commands ............................................................ 10
   Console Usage ....................................................................... 11

5 PVM Library
   Process Control ................................................................. 12
   Dynamic Configuration ....................................................... 13
   Message Buffers ................................................................. 13
   Packing and Unpacking ....................................................... 13
   Sending and Receiving ......................................................... 14

6 Tutorial
   Golden Section ................................................................. 16
   Serial Program ................................................................. 17
   PVM Master Guideline/Master Program ............................... 18
   PVM Slave Guideline/Slave Program ................................. 22
   Compilation and Running .................................................... 24
7 Problems and Tips
Problems ............................................................. 26
Host File .............................................................. 27

8 Appendix
ARCH Names...................................................... 29
Error Codes ....................................................... 30
Preface

This manual serves as a supplementary document for the official reference manual of a relatively new research software, PVM, which has been developed at Oak Ridge National Laboratory. A beginner, who has no previous experience with PVM, would find this manual useful.

We would like to thank you in advance that if you find any problems in PVM or this manual, please contact one of us.

Mr. Merritt Smith
NASA Ames Research Center, MS 258-1
Moffett Field, CA 94035
e-mail: mhsmith@nas.nasa.gov
phone: (415) 604-4462

Dr. Samson Cheung
MCAT Institute
NASA Ames Research Center, MS 258-1
Moffett Field, CA 94035
e-mail: cheung@nas.nasa.gov
phone: (415) 604-4462
This manual provides you with an introduction to PVM and provides the fundamentals necessary to write FORTRAN programs in the PVM environment through a tutorial sample. This manual is designed for those who have no previous experience with PVM. However, you should know basic FORTRAN programming and UNIX. If you are ready for an advanced PVM application, please consult the official PVM Reference Manual.

Software Package

PVM stands for Parallel Virtual Machine. It is a software package that allows a heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource. PVM allows you to link up all or some of the computational systems on which you have accounts, to work as a single distributed-memory (parallel) machine. We call this a Virtual Machine.

PVM is useful for the following reasons. Unlike large mainframe computers or vector supercomputers, workstations spend most of the time idle. The idle time on a workstation represents a significant computational resource. PVM links these workstations up to become a powerful multi-processor computational machine. With PVM, the lack of supercomputer resources should not be an obstacle to number crunching computational programs. Furthermore, the annual maintenance costs of a vector supercomputer is often sufficient to purchase the equivalent computing resource in the form of workstation CPU's.

Definitions

Here are some terms we use throughout this document:

- **Virtual Machine**: PVM links different user-defined computers together to perform as one large distributed-memory computer. We call this computer the Virtual Machine.
- **Host**: Individual computer (member) in the virtual machine.
- **Process**: Individual program operating on different computers or hosts.
- **Processor**: The processing unit in computers. A virtual machine can be viewed as a multi-processor computer.
INTRODUCTION

Task
The unit of computation handled by the virtual machine. You may want to think of one processor handling one task.

Tid
Task identification number which is a unique number used by the daemon and other tasks.

Console
A program from which you can directly interact with the virtual machine. (Add hosts, kill processes,...)

Structure of PVM
The PVM software is composed of two parts. The first part is a daemon. We call it pvmd. This is the control center of the virtual machine. It is responsible for starting processes, establishing links between processes, passing messages, and many other activities in PVM. Since the daemon runs in the background, you have to use PVM console to directly interact with the virtual machine.

The second part of the system is a library of PVM interface routines located in libpvm3.a. This library contains user callable routines for message passing, spawning processes, coordinating tasks, and modifying your machine. In writing your application, you will need to call the routines in this library.

Directory Setup
This setup is for NAS system. Before you use PVM, you need to set up the following directories on all the machines that you want PVM to link:

• Make a directory $HOME/pvm3/bin/ARCH in all the hosts of the virtual machine.

Note ARCH is used throughout this manual to represent the architecture name that PVM uses for a given computer. The table in the Appendix lists all the ARCH names that PVM supports. For example, for Silicon Graphic IRIS workstations, you should make a directory $HOME/pvm3/bin/SGI.

• Make a directory $HOME/pvm3/include, and copy the file fpv3.h from /usr/nas/pkg/pvm3.2/include. (If you are on different system from NAS, please consult your system consultant.)

• Make a directory $HOME/pvm3/codes, and write your application programs in this directory. You can actually put your programs anywhere you like as long as the correct “include” files are included. The current setup is for clarity.
Programming Concept

Unlike graphical software or a word-processor, you cannot see PVM working by clicking your mouse buttons. In fact, a virtual machine is quite an abstract concept because you don’t physically have a multi-processor machine! In this chapter, you will learn a simple concept, which will help you to visualize how PVM works.

A common way to work with PVM is a Master/Slave relationship. A Master process starts Slave routines and distributes work. However, a Master does not actively participate in the computation. A Master process most often resides on the originating host (user’s computer), while the Slave programs are distributed to the hosts of the virtual machine.

You need to distribute executables of Slave programs to the directory $HOME/pvm3/bin/ARCH on every host. You can locate this Master program anywhere you like.

Since the Master program spawns Slave programs on each of the hosts to do jobs, it is important to understand the communication (message passing) among the hosts in PVM.

Typically, a Master and a Slave have the following logic:

<table>
<thead>
<tr>
<th>Master</th>
<th>Slave</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Enroll itself to PVM</td>
<td>1 Enroll itself to PVM</td>
</tr>
<tr>
<td>2 Spawn slave processes</td>
<td>2 Receive message from master</td>
</tr>
<tr>
<td>3 Initialize buffer, pack, and send message to all slaves.</td>
<td>3 ...do something useful...</td>
</tr>
<tr>
<td>4 ...wait for slaves to finish...</td>
<td>4 Initialize buffer, pack, and send message to master</td>
</tr>
<tr>
<td>5 Receive message from slave(s)</td>
<td>5 Exit PVM</td>
</tr>
<tr>
<td>6 Exit PVM</td>
<td></td>
</tr>
</tbody>
</table>

The figure on the opposite page graphically describes a Master/Slave relationship and shows the exchange of information.
FIGURE 1. Communication in Master/Slave programs.
SPMD

Another common way to work with PVM is the SPMD, Single Program Multiple Data model. There is only a single program, and there is no Master program directing the computation. The user starts the first copy of the program and using the routine `pvmfparent()`, this copy can determine that it was not spawned by PVM, and thus must be the first copy (parent). It then spawns multiple copies (children) of itself and passes them the array of `tids`. At this point each copy is equal and can work on its partition of the data in collaboration with the other processes.

Typically, a SPMD program has the following logic:

1. **Enroll in pvm**
2. **If I am the first copy (parent)**
   a) Spawn child processes
   b) Initialize buffer, pack, and send message out
3. **If I am a secondary copy (child)**
   Receive messages
4. **Work!...Work!...Work!**
5. **Exit PVM**

The program on the opposite page describes a SPMD logic and shows the exchange of information. Please spend some time to study the program.

In the next chapter we will introduce the PVM daemon and the fundamentals of message passing.
SPMD
Program

program spmd
include '../include/fpvm3.h'
PARAMETER( NPROC=4 )
iconteger mytid, me, i
tinteger tids(0:NPROC)

Enroll in pvm
call pvmfmytid( mytid )

Find out if I am parent or child

call pvmfparent( tids(0) )
if( tids(0) .lt. 0 ) then
  tids(0) = mytid
  me = 0

start up copies of myself

call pvmfspawn('spmd', PVMDEFAULT, '*', '
  * NPROC-1, tids(1), info)

multicast tids array to children

call pvmfinitsend( PVMDEFAULT, info )
call pvmfpack( INTEGER4, tids, NPROC-1, info )
call pvmfmcast( NPROC-1, tids(1), 0, info )
else

receive the tids array and set me

call pvmfrecv( tids(0), 0, info )
call pvmfunpack( INTEGER4, tids, NPROC-1, info)
do 30 i=1, NPROC-1
  if( mytid .eq. tids(i) ) me = i
  continue
30 continue
endif

all NPROC tasks are equal now
and can address each other by tids(0) thru tids(NPROC-1)
for each process me = process number [0-(NPROC-1)]

print*, 'me = ', me, ' mytid = ', mytid
call dowork( me, tids, NPROC )

program finished exit pvm

call pvmfexit( info)
stop
end
Notes
subroutine dowork( me, tids, nproc )
include '../include/fpvm3.h'

C Simple subroutine to pass a token around a ring
C
integer me, nproc
integer tids( 0:nproc)

integer token, dest, count, stride, msgtag

count = 1
stride = 1
msgtag = 4

if( me .eq. 0 ) then  
token = tids(0)
call pvmfinitsend( PVMDEFAULT, info )
call pvmfpack( INTEGER4,token,count,stride,info)
call pvmfsend( tids(me+1), msgtag, info )
call pvmfrecev( tids(nproc-1), msgtag, info )
print*, 'token ring done'
else
  call pvmfrecev( tids(me-1), msgtag, info )
call pvmfunpack( INTEGER4,token,count,stride,info)
call pvmfinitsend( PVMDEFAULT, info )
call pvmfpack( INTEGER4,token,count,stride,info)
dest = tids(me+1)
if( me .eq. nproc-1 ) dest = tids(0)
call pvmfsend( dest, msgtag, info )
endif

return
end
The PVM daemon is the control center of the virtual machine. You can activate the PVM daemon by starting the PVM console or by invoking the daemon directly with a list of hosts. The latter will be discussed in chapter 6. To start the console, enter `pvm` at UNIX prompt on your local machine. The PVM console prints the prompt

```
pvm>
```

and accepts commands from standard input. The console allows interactive adding and deleting of hosts to the virtual machine as well as interactive starting and killing of PVM processes. Even if the daemon is started directly, the console can be used to modify the virtual machine.

### Console Commands

Here are the commands available in the PVM console:

- **ADD**: add other computers (hosts) to PVM
- **ALIAS**: define and list command aliases you set
- **CONF**: show members in virtual machine
- **DELETE**: remove hosts from pvm
- **ECHO**: echo arguments
- **HALT**: stop all pvm processes and exit daemon
- **HELP**: print this information
- **ID**: print console task identity
- **JOBS**: display list of running jobs
- **KILL**: terminate tasks
- **MSTAT**: show status of hosts
- **PS**: list tasks
- **PSTAT**: show status of tasks
- **QUIT**: exit PVM console, but PVM daemon is still activated
- **RESET**: kill all tasks
- **SETENV**: display or set UNIX environment variables
- **SIG**: send signal to task
- **SPAWN**: spawn task
- **UNALIAS**: remove alias commands you previous set
- **VERSION**: show PVM version
Console Usage

Suppose the console is running on workstation *win210*. This computer will automatically be a host in your virtual machine. Here are some examples of using PVM console:

1. **Activate PVM console**
   
   ```
   win210> pvm
   ```

2. **Add amelia and fred to your virtual machine**
   
   ```
   pvm> add amelia
   1 successful
   HOST     DTID
   amelia   c0000
   
   pvm> add fred
   1 successful
   HOST     DTID
   fred     100000
   ```

3. **Check the configuration of your virtual machine**
   
   ```
   pvm> conf
   3 host, 1 data format
   HOST     DTID     ARCH     SPEED
   win210   40000     SGI      1000
   amelia   c0000     SGI      1000
   fred     100000    SGI      1000
   ```

4. **Delete amelia**
   
   ```
   pvm> delete amelia
   1 successful
   HOST     STATUS
   amelia   deleted
   ```

5. **Exit PVM console, but PVM daemon is still running**
   
   ```
   pvm> quit
   pvmd still running
   win210>
   ```
This chapter introduces the PVM library. In writing your application programs, you need to call the subroutines in the library to instruct PVM to control processes, send information, pack/unpack data, and send/receive messages. Many subroutines have pre-defined option values for some arguments. These are defined in the include file `epvm3.h` and are listed in the Appendix.

### Process Control

**call pvmfmytid( tid )**

This routine enrolls this process with the PVM daemon on its first call, and generates a unique `tid`. You call this routine at the beginning of your program.

**call pvmfexit( info )**

This routine tells the local PVM daemon that this process is leaving PVM. You call this routine at the end of your program. Values of `info` less than zero indicate an error.

**call pvmfkill( tid, info )**

This routine kills a PVM task identified by `tid`. Values of `info` less than zero indicate an error.

**call pvmfspawn( pname, flag, where, ntask, tids, numt )**

This routine starts up `ntask` instances of a single process named `pname` on the virtual machine. Here are the definition of the other arguments:

<table>
<thead>
<tr>
<th>flag</th>
<th>Option Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PVMDEFAULT (0)</td>
<td>PVM can choose any machine to start task</td>
<td></td>
</tr>
<tr>
<td>PVMHOST (1)</td>
<td>where specifies a particular host</td>
<td></td>
</tr>
<tr>
<td>PVMARCH (2)</td>
<td>where specifies a type of architecture</td>
<td></td>
</tr>
<tr>
<td>PVMDEBUG (4)</td>
<td>start up processes under debugger</td>
<td></td>
</tr>
<tr>
<td>PVMTRACE (8)</td>
<td>processes will generate PVM trace data</td>
<td></td>
</tr>
</tbody>
</table>

`where` is where you want to start the PVM process. If `flag` is `0`, `where` is ignored. `tids` contains identification numbers of PVM processes started by this routine. `numt` indicates how many processors started; negative values indicate an error.

**Note**

You should always check `tids` and `numt` to make sure all processes started correctly.
PVM Library

call pvmfparent (tid)
This routine returns the uid of the process that spawned this task. If the calling process was not created with pvm.spawn, then tid = PvmNoParent.

Dynamic Configuration

call pvmfaddhost( host, info )
call pvmfdelhost( host, info )
These routines add and delete hosts to the virtual machine respectively. Values of info less than zero indicate an error.

Note Both routines are expensive operations that require the synchronization of the virtual machine.

Message Buffers

call pvmfinitsend( encoding, build )
This routine clears the send buffer, and creates a new one for packing a new message.

encoding

<table>
<thead>
<tr>
<th>Encoding Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PVMDEFAULT (0)</td>
<td>XDR encoding if virtual machine configuration is heterogeneous</td>
</tr>
<tr>
<td>PVMRAW (1)</td>
<td>no encoding is done. Messages are sent in their original format.</td>
</tr>
<tr>
<td>PVMINPLACE (2)</td>
<td>data left in place. Buffer only contains sizes and pointers to the sent items.</td>
</tr>
</tbody>
</table>

build contains the message buffer identifier. Values less than zero indicate an error.

call pvmffreebuf( build, info)
This routine disposes the buffer with identifier build. You use it after a message has been sent, and is no longer needed. Values of info less than zero indicate an error.

Packing and Unpacking

call pvmfpack( what, xp, nitem, stride, info )
call pvmfunpack( what, xp, nitem, stride, info )
These routines pack/unpack your message xp, which can be a number or a string. You can call these routines multiple times to pack/unpack a single message. Thus a message can contain several arrays, each with a different data type.
Note: There is no limit to the complexity of the packed messages, but you must unpack them exactly as they were packed.

Indicates what type of data xp is:

- STRING (0)
- REAL (4)
- BYTE1 (1)
- COMPLEX8 (5)
- INTEGER2 (2)
- REAL8 (6)
- INTEGER4 (3)
- COMPLEX16 (7)

is number of items in the pack/unpack. If xp is a vector of 5, nitem is 5.

stride is the stride to use when packing.

info is status code returned by this routine. Values less than zero indicate an error.

### Sending and Receiving

**call pvmfsend** (tid, msgtag, info)

This routine labels the message with an integer identifier msgtag, and sends it immediately to the process tid. Values of info less than zero indicate an error.

**call pvmfmcast** (ntask, tids, msgtag, info)

This routine labels the message with an integer identifier msgtag, and broadcasts the message to all ntask number of tasks specified in the integer array tids. Values of info less than zero indicate an error.

**call pvmfrecv** (tid, msgtag, bufid)

This routine blocks the flow of your program until a message with label msgtag has arrived from tid. A value of -1 in msgtag or tid matches anything (wildcard). This routine creates a new active receive buffer, and puts the message in it. Values of bufid identify the newly created buffer; values less than zero indicate an error.

**call pvmfrecv** (tid, msgtag, bufid)

This routine performs in the same way as pvmrecv, except that it does not block the flow of your program. If the requested message has not arrived, this routine returns bufid=0. This routine can be called multiple times for the same message to check if it has arrived, while performing useful work between calls. When no more useful work can be performed, the blocking receive pvmrecv can be used for the same message.

**call pvmfprobe** (tid, msgtag, bufid)

This routine checks if a message has arrived; however, it does not receive the message. If the requested message has not arrived, this routine returns bufid=0. This routine can
be called multiple times for the same message to check if it has arrived, while performing useful work between calls.

call pvmfbufinfo (bufid, bytes, msgtag, tid, info)
This routine returns information about the message in the buffer identified by bufid. The information returned is the actual msgtag, source tid, and message length in bytes. Values of info less than zero indicate an error.
This chapter shows you how PVM may be applied to your application programs through a simple example. The example chosen is the Golden Section rule for finding the maximum of a function. You may remember it from Math class in high school. Let us review the method and the algorithm.

Suppose we want to find the maximum of a curve \( y = f(x) \); where \( x \) is between the interval \( a_1 \) and \( a_2 \). The points \( a_3 \) and \( a_4 \) are symmetrically placed in this interval, so that

\[
a_3 = (1-\alpha) a_1 + \alpha a_2 \quad \text{(EQ 1)}
\]
\[
a_4 = \alpha a_1 + (1-\alpha) a_2 \quad \text{(EQ 2)}
\]

See Figure 1 at left. Golden Section rule requires \( \alpha \) to be 0.382.

The algorithm of finding the maximum is as follow:

- If \( f(a_4) < f(a_3) \):
  1. Consider new interval \((a_1, a_4)\)
  2. Apply EQ. (1) and (2) again
  3. Until maximum is reached

- If \( f(a_4) > f(a_3) \):
  1. Consider new interval \((a_3, a_2)\)
  2. Apply EQ. (1) and (2) again
  3. Until maximum is reached

If \( f(a_3) = f(a_4) \), the maximum is found.

The FORTRAN program (Serial Program) on the opposite page is the Golden Section rule that a programmer would write on a normal serial computer. Please spend a few minutes to study the flow of the program. This simple program consists of two parts, the main (calling) program and the function subroutine. The latter has only four lines.

Note: Notice that for each interval \((a_1, a_2)\), we need to call the function evaluation four times to find \( f(a_1) \), \( f(a_2) \), \( f(a_3) \), and \( f(a_4) \).
Serial Program

Linear optimization:
Search for maximum of a x-y curve.

DIMENSION A(4), FN(4)

Initial interval
L = 0
TOL = 1.E-3
A(1) = 0.4
A(2) = 1.6

ALPHA = 0.382

CONTINUE

Loop begins:
L = L + 1

Four function evaluations

FUNCTION F(X)
F = TANH(X)/(1.+X*X)
RETURN
END

1. If $f(a4) > f(a3)$
   - $B1 = A(3)$
   - $B2 = A(2)$
   - $A(1) = B1$
   - $A(2) = B2$
   - GOTO 10

2. If $f(a4) < f(a3)$
   - $B1 = A(1)$
   - $B2 = A(4)$
   - $A(1) = B1$
   - $A(2) = B2$
   - GOTO 10
PVM Master Guideline

Recall that in the procedure of finding a new interval, the program calls the function evaluation four times serially to get \( f(a_1), f(a_2), f(a_3), \) and \( f(a_4). \) We would like to assign four processors to perform the four function evaluations simultaneously on the virtual machine. Therefore, we modify the Serial Program by writing the main (calling) program as a Master program, and the function subroutine as a Slave program.

The following steps are general guidelines to writing a Master program. Please study the steps, and compare them with the program on the opposite page. Also compare it with the Serial Program.

1. **Include fpvm3.h**
   Include this file in your program, you are able to use the PVM preset variables; such as PVMDFAULT, REAL4, and more, mentioned in Chapter 4 and the Appendix.

2. **Enroll Master to PVM**
   Use `pvmfmytid(mytid)` to enroll.

3. **Assign virtual processors**
   Use the following call to spawn `nproc` function processes.
   ```c
   pvmfspawn(pname, PVMDFAULT, where, nproc, tids, numt)
   ```
   Also tell PVM the name of the Slave program (`pname`). PVM returns `tids`, the identifier of the `nproc` processors.

4. **Initialize buffer and pack data**
   Use `pvmfinitsend` to clear buffer.
   Use the following routine to pack a real array `A` of dimension `m`.
   ```c
   pvmfpack(REAL4, A, m, 1, info)
   ```

5. **Send message**
   Use the following call to send the packed message to the Slave process identified by `tids`.
   ```c
   pvmfmcast(nproc, tids, msgtag, info)
   ```
Master Program

C Linear optimization:
C Search for maximum of a x-y curve.
PROGRAM MASTER
C
C include '../include/fpvm3.h'
DIMENSION A(4),FN(4)
integer tids(0:32),who
color*8 where
color*12 pname

Enroll this program in PVM
call pmfmytid(mytid)
C Start up the four processors
nproc = 4
where = '*
pname = 'function'
call pmfspawn(pname,PVMDFAULT,where,nproc,tids,numt)
do 20 i=0,nproc-1
   write(*,*) 'tid', i, tids(i)
20 continue
C
C Initial interval
L = 0
A(1) = 0.4
A(2) = 1.6
ALPHA = 0.382
TOL = 1.E-3
ERR = 1.
C
CONTINUE
C
C Loop begins:
L = L + 1

Equations (1) and (2)
A(3) = (1.-ALPHA)*A(1) + ALPHA*A(2)
A(4) = ALPHA*A(1) + (1.-ALPHA)*A(2)

Broadcast data to all node programs
first pack them, then send them
call pmfinitsend(PVMDEFAULT,info)
call pmfpack(INTEGER4,nproc,1,1,info)
call pmfpack(INTEGER4,tids,nproc,1,info)
call pmfpack(REAL4,A,4,1,info)
call pmfpack(REAL4,ERR,1,1,info)

Pack nproc, tids, A, and ERR
msgtype = 1
call pmfmcast(nproc,tids,msgtype,info)
6. **Wait until messages come from Slaves**
   Use `pvmrecv()` to block until Slaves return function values. Make sure value of `msgtype` matches values coming from Slaves.

7. **Receive and Unpack data**
   The sequence of unpacking is the same as the packing in the Slave.

8. **Exit PVM**
   Use `pvmexit(info)` to exit PVM.
6

msgtype value matches the one sent from Slave program

7

Receive/unpack FN and 'who' from the 4 processors one by one

![Python code]

```
msgtype = 2
DO 100 I = 1, NPROC
   CALL PVMRECV(-1, MSGTYPE, INFO)
   CALL PVMFUNPACK(INTEGER4, WHO, 1, 1, INFO)
   CALL PVMFUNPACK(REAL4, FN(WHO), 1, 1, INFO)
   CONTINUE

WRITE(10, *) 'A ', A(1), A(2), A(3), A(4)
WRITE(10, *) 'F ', FN(1), FN(2), FN(3), FN(4)
WRITE(10, *) ' ' ERR = ABS(FN(2) - FN(3))
IF(ERR .LE. TOL) GOTO 999

IF(FN(4) .GT. FN(3)) THEN
   B1 = A(3)
   B2 = A(2)
   A(1) = B1
   A(2) = B2
   GOTO 10
ELSEIF(FN(4) .LT. FN(3)) THEN
   B1 = A(1)
   B2 = A(4)
   A(1) = B1
   A(2) = B2
   GOTO 10
ENDIF

Program finished leave PVM before exiting
```

8

![Python code]

```
CALL PVMEXIT(INFO)
STOP
END
```
The Slave program is basically the function evaluation program. In order to do the function evaluation, it needs information from Master. For example, it needs the identity numbers \( \text{tids}(1), \ldots, \text{tids}(4) \) that PVM assigns, and the values of \( a_1, \ldots, a_4 \).

The following steps are general guidelines to writing a Slave program. Please study the steps, and compare them with the program on the opposite page. Also try to find the connection with the Master Program. You may find Figure 1 helpful.

1. **Include fpvm3.h**
   Include this file in your program, you are able to use the PVM preset variable names; such as `PVMDFAULT`, `REAL4`, and more, mentioned in all tables in Chapter 4 and the Appendix.

2. **Enroll Slave with PVM**
   Use `pvmfmytid(mytid)` to enroll.

3. **Identify the parent of this process**
   Use the following call to obtain the task identifier \( \text{mtid} \) of parent process. This is useful for returning solutions to the Master.
   
   ```
   pvmfparent(mtld)
   ```

4. **Receive and Unpack data**
   Make sure the value of `msgtype` matches the one from Master. The sequence of unpacking is the same as the packing in Master.

5. **Perform function evaluation**

6. **Initialize buffer and pack data**
   Use `pvmfinitsend` to clear buffer.
   Use the following call to pack a real array \( F \) of dimension \( n \).
   
   ```
   pvmfpack(REAL4,F,n,1,info)
   ```

7. **Send data**
   Use the following call to send the packed message to Master.
   
   ```
   pvmfsend(mtld,msgtag,info)
   ```

8. **Exit PVM**
   Use `pvmfexit(info)` to exit PVM.
Slave Program

1. Program function
   ```c
   include '../include/fpvm3.h'
   ```
   ```c
   integer tids(0:32),who
   real a(32)
   tor = 1.e-3
   ```

2. Enroll this program in PVM
   ```c
   call pvmfmytid(mytid)
   ```
   ```c
   Get the parent's task id
   call pvmfparent(ztid)
   ```

3. Continue
   ```c
   Receive data from host
   msgtype = 1
   call pvmfrecv(mtid,msgtype,info)
   call pvmfunpack(INTEGER4,nproc,l,l,lnfo)
   call pvmfunpack(INTEGER4,tids,nproc,l,info)
   call pvmfunpack(REAL4,A,4,1,info)
   call pvmfunpack(REAL4,ERR,1,1,info)
   ```
   ```c
   if(err.le.tor) go to 99
   ```

4. Determine which processor I am
   ```c
   do 5 i=0,nproc-1
      if(tids(i).eq.mytid) me = i
      continue
   who = me + 1
   ```

5. Calculate the function
   ```c
   X = A(who)
   f = TANH(X)/(1.+X*X)
   ```

6. Send the result to Master
   ```c
   call pvmfinitsend(PVMDEFAULT,info)
   call pvmfpack(INTEGER4,who,l,l,info)
   call pvmfpack(REAL4,f,l,1,info)
   msgtype = 2
   call pvmfsend(mtid,msgtype,info)
   go to 3
   ```

7. Program finished. Leave PVM before exiting
   ```c
   Program finished. Leave PVM before exiting
   continue
   ```

8. Stop
   ```c
   call pvmfexit(info)
   stop
   end
   ```
Compilation and Running

After you finish your program, it is time to compile and run. Follow the steps below to compile your programs.

1. **Make sure you have the correct directory setup**
   Follow the advice from *Directory Setup* in Chapter 1.

2. **Compile the program**
   Use the sample Makefile on the opposite page to compile your programs.

   **Note** The Makefile links the PVM library, *libfpvm3.a*.

3. **Copy executables to all the hosts**
   Follow the advice from *Directory Setup* in Chapter 1, and distribute the executables to $HOME/pvm3/bin/ARCH.

4. **Activate PVM**
   Activate PVM by entering `pvm` at UNIX prompt.

5. **Decide the configuration of the virtual machine**
   Add or delete hosts to the virtual machine. (Chapter 3)

6. **Quit PVM console**
   Leave PVM console (don't halt daemon) by entering `quit` at the `pvm` prompt.
Makefile

`# Custom section
# Set PVM_ARCH to your architecture type (SUN4, HP9K, RS6K, SGI, etc.)
# if PVM_ARCH = BSD386 then set ARCHLIB = -lrpc
# if PVM_ARCH = SGI then set ARCHLIB = -lsun
# if PVM_ARCH = I860 then set ARCHLIB = -lrpc -lsocket
# if PVM_ARCH = IPSC2 then set ARCHLIB = -lrpc -lsocket
# otherwise leave ARCHLIB blank
#
# PVM_ARCH and ARCHLIB are set for you if you use 'aimk'.
#
PVM_ARCH = SGI
ARCHLIB = -lsun
# END of custom section - leave this line here
#
PVM_DIR = /amd/fs02/pub/iris4d_irix4/nas/pkg/pvm3.2
PVMLIB = $(PVM_DIR)/lib/$(PVM_ARCH)/libpvm3.a
SDIR =
BDIR = /u/wk/cheung/pvm3/bin
XDIR = $(BDIR)/$(PVM_ARCH)

CFLAGS = -g -I../include
LIBS = $(PVMLIB) $(ARCHLIB)

F77 = f77
FFLAGS = -g
FLIBS = $(PVM_DIR)/lib/$(PVM_ARCH)/libfpv3.a $(LIBS)

default: master function

$(XDIR):
  - mkdir $(BDIR)
  - mkdir $(XDIR)

clean:
  rm -f *.o bfgs quadfunt

master: $(SDIR)/master.f $(XDIR)
  $(F77) $(FFLAGS) -o master $(SDIR)/master.f $(FLIBS)
  mv master $(XDIR)

function: $(SDIR)/function.f $(XDIR)
  $(F77) $(FFLAGS) -o function $(SDIR)/function.f $(FLIBS)
  mv function $(XDIR)`
Problems and Tips

PVM is a relatively new piece of software. It is not advanced enough to warn you ahead of time before problems come. Here are a couple of cases that you may encounter as a beginner.

Problems

Can’t activate PVM

• If the message you get, after entering `pv_` at UNIX prompt, is `libpvm [pid-1]: Console: Can’t start pmd`, it is possible that the last time you halted PVM daemon, the daemon created a residual file `/tmp/pvmd_xxxxx`, where `xxxx` is an unique number for you. Delete this file, and start PVM again.

• If the daemon is running but the PVM console will not start, it is possible that you have too many processes running. You have to kill all the processes before you re-activate PVM console.

Note Use `ps -ef | username` at UNIX prompt to locate your running processes.

Can’t add hosts

It is possible that there are no links between your local computer and the other hosts. Check the following two things:

• Make sure each of your hosts has a `.rhosts` file in the `$HOME` directory, and this file points to your local computer.

• Make sure the `.rhosts` file is “read” and “write” protected from others users.
You can create the following file to build the virtual machine without activating the PVM console. The addresses must be recognizable by your system.

```
computer1.address
computer2.address
computer3.address
computer4.address
```

Note The first machine listed must be the initiating host.

Note If tasks are to be spawned on specific systems, the system name contained in where (routine pvmspawn) must match the name in the host file exactly.

Note If spawning tasks are on the initiating host, use the truncated host name. For example, if the full address is `win210.nas.nasa.gov`; use `win210` instead. This is a bug in PVM v3.2.

Having the host file ready, enter the following at UNIX prompt,

```
win210> pvmd3 host
```
Notes

Place to jot down problems.

If encounter problems, please contact:

Merritt Smith: mhsmith@nas.nasa.gov

or

Samson Cheung: cheung@nas.nasa.gov
Appendix

TABLE 1. ARCH names used in PVM.

<table>
<thead>
<tr>
<th>ARCH</th>
<th>Machine</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFX8</td>
<td>Alliant FX 8</td>
<td></td>
</tr>
<tr>
<td>ALPHA</td>
<td>DEC Alpha</td>
<td>DEC OSF-1</td>
</tr>
<tr>
<td>BAL</td>
<td>Sequent Balance</td>
<td>DYNIX</td>
</tr>
<tr>
<td>BFLY</td>
<td>BBN Butterfly TC2000</td>
<td></td>
</tr>
<tr>
<td>BSD386</td>
<td>80386/486 Unix box</td>
<td>BSDI</td>
</tr>
<tr>
<td>CM2</td>
<td>Thanking Machines CM2</td>
<td>Sun front-end</td>
</tr>
<tr>
<td>CM5</td>
<td>Thanking Machines CM5</td>
<td></td>
</tr>
<tr>
<td>CNVX</td>
<td>Convex C-series</td>
<td></td>
</tr>
<tr>
<td>CNVXN</td>
<td>Convex C-series</td>
<td>native mode</td>
</tr>
<tr>
<td>CRAY</td>
<td>C-90, YMP, Cray-2</td>
<td>UNICOS</td>
</tr>
<tr>
<td>CRAYSM</td>
<td>Cray S-MP</td>
<td></td>
</tr>
<tr>
<td>DGAV</td>
<td>Data General Aviion</td>
<td></td>
</tr>
<tr>
<td>HP300</td>
<td>HP-9000 model 300</td>
<td>HPUX</td>
</tr>
<tr>
<td>HPPA</td>
<td>HP-9000 PA-RISC</td>
<td></td>
</tr>
<tr>
<td>I860</td>
<td>Intel iPSC/860</td>
<td>link-lprc</td>
</tr>
<tr>
<td>IPSC2</td>
<td>Intel iPSC/860 host</td>
<td>SysV</td>
</tr>
<tr>
<td>KSR1</td>
<td>Kendall Square KSR-1</td>
<td>OSF-1</td>
</tr>
<tr>
<td>NEXT</td>
<td>NeXT</td>
<td></td>
</tr>
<tr>
<td>PGON</td>
<td>Intel Paragon</td>
<td>link-lprc</td>
</tr>
<tr>
<td>PMAX</td>
<td>DECstation 3100,5100</td>
<td></td>
</tr>
<tr>
<td>RS6K</td>
<td>IBM/RS6000</td>
<td>Ultrix</td>
</tr>
<tr>
<td>RT</td>
<td>IBM RT</td>
<td>AIX</td>
</tr>
<tr>
<td>SGI</td>
<td>Silicon Graphics IRIS</td>
<td>link-lsun</td>
</tr>
<tr>
<td>SUN3</td>
<td>Sun 3</td>
<td>SunOS</td>
</tr>
<tr>
<td>SUN4</td>
<td>Sun 4, SPARCstation</td>
<td></td>
</tr>
<tr>
<td>SYMM</td>
<td>Sequent Symmetry</td>
<td></td>
</tr>
<tr>
<td>TITN</td>
<td>Staedent Titan</td>
<td></td>
</tr>
<tr>
<td>UVAX</td>
<td>DEC Micro VAX</td>
<td></td>
</tr>
</tbody>
</table>
#### TABLE 2. Error codes returned by PVM routines

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PvmOK (0)</td>
<td>All right</td>
</tr>
<tr>
<td>PvmBadParam (-2)</td>
<td>Bad parameter</td>
</tr>
<tr>
<td>PvmMismatch (-3)</td>
<td>Barrier count mismatch</td>
</tr>
<tr>
<td>PvmNoData (-5)</td>
<td>Read past end of buffer</td>
</tr>
<tr>
<td>PvmNoHost (-6)</td>
<td>No such host</td>
</tr>
<tr>
<td>PvmNoFile (-7)</td>
<td>No such executable</td>
</tr>
<tr>
<td>PvmNoMem (-10)</td>
<td>Can’t get memory</td>
</tr>
<tr>
<td>PvmBadMsg (-12)</td>
<td>Can’t decode received massage</td>
</tr>
<tr>
<td>PvmSysErr (-14)</td>
<td>Pvmd not responding</td>
</tr>
<tr>
<td>PvmNoBuf (-15)</td>
<td>No current buffer</td>
</tr>
<tr>
<td>PvmNoSuchBuf (-16)</td>
<td>Bad message identifier</td>
</tr>
<tr>
<td>PvmNukkGroup (-17)</td>
<td>Null group name is illegal</td>
</tr>
<tr>
<td>PvmDupGroup (-18)</td>
<td>Already in group</td>
</tr>
<tr>
<td>PvmNoGroup (-19)</td>
<td>No group with that name</td>
</tr>
<tr>
<td>PvmNotInGroup (-20)</td>
<td>Not in group</td>
</tr>
<tr>
<td>PvmNoInst (-21)</td>
<td>No such instance in group</td>
</tr>
<tr>
<td>PvmHostFail (-22)</td>
<td>Host failed</td>
</tr>
<tr>
<td>PvmNoParent (-23)</td>
<td>No parent task</td>
</tr>
<tr>
<td>PvmNoImpl (-24)</td>
<td>Function not implemented</td>
</tr>
<tr>
<td>PvmDSysErr (-25)</td>
<td>Pvmd system error</td>
</tr>
<tr>
<td>PvmBadVersion (-26)</td>
<td>Pvmd-pvmd protocol mismatch</td>
</tr>
<tr>
<td>PvmOutOfRes (-27)</td>
<td>Out of resources</td>
</tr>
<tr>
<td>PvmDupHost (-28)</td>
<td>Host already configured</td>
</tr>
<tr>
<td>PvmCantStart (-29)</td>
<td>Fail to execute new slave pvmd</td>
</tr>
<tr>
<td>PvmAlready (-30)</td>
<td>Slave pvmd already running</td>
</tr>
<tr>
<td>PvmNoTask (-31)</td>
<td>Task does not exist</td>
</tr>
<tr>
<td>PvmNoEntry (-32)</td>
<td>No such (group.instance)</td>
</tr>
<tr>
<td>PvmDupEntry (-33)</td>
<td>(Group.instance) already exists</td>
</tr>
</tbody>
</table>