Supersonic Civil Airplane Study and Design:
Performance and Sonic Boom

Samson Cheung

January 1995
NCC2-617


Unclas
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 opti-
mization tool suitable for parallel computers. A nonlinear unconstrained optimizer for Par-
allel 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 aerody-
namic 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 base-
line configuration. This model was tested in June 1993. The test results were used to vali-
date 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) con-
cept. 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-tun-
nel 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
- HYPGEN: 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 and published in Journal of Aircraft Vol. 32, No. 1, Jan/Feb. 1995.

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.
2.4 Oblique-All Wing (OAW) Computation and Design

Oblique flying-wing 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. 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
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
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 personnels 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 (IIOWA)

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 concept 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 IIOWA (parallel Optimizer With Aerodynamics). The figure below is a demonstration of IIOWA. 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 optimizer 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 Lighthill F-function on body surface,
   a) input parameters
2) the overpressure signature at given distance R1
3) the loudness level of the sonic boom at R1

INPUT:
LHF.in(3) : input parameter
area.in(3) : equivalent area distribution, (INAREA=0)
coef.dat[3] : F-function due to lift, (LIFT=1)
area.out(12) : Equivalent area distribution and its derivative.
                       F-functions on the body surface and at distance R1.
                       Pressure signature at distance R1.
                       Integral curve of the shifted F-function.

Output:

By Sanam Chang [Tel: (415)-604-4662]
NCAT Institute
NASA Ames Research Center
W3 355-81
 Moffett Field, CA 94035
Date: 1/20/92 Varion 2.1

PARAMETER (MAX=220,MIN=1,MAXX=351,MAXX=900)
DIMENSION X(MAX,MAX,MAX),Y(MAX,MAX,MAX),Z(MAX,MAX,MAX),W(MAX,MAX,MAX),T(MAX,MAX,MAX),F(MAX,MAX,MAX)
REAL X(MAX,MAX,MAX),Y(MAX,MAX,MAX),Z(MAX,MAX,MAX),W(MAX,MAX,MAX),T(MAX,MAX,MAX),F(MAX,MAX,MAX)
COMMON/PAR,PAR,MFAC,PFAC
LOGICAL BOPT

OPEN(UNIT=1,FILE='lhf.in',STATUS='OLD')
Read the input parameter

IF(TORI>0, sonic boom varies time, else varies distance
   IF TORI>0, is the surface grid contains the whole configuration, or
   only half-plane or only quarter-plane?
   PPAC = 1. Whole plane
   PPAC = 2. Half-plane
   PPAC = 3. Quarter-plane
   R1 will be the distance where the signature is captured.

IF read in area distribution, INAREA = 0

IF read the grid
   INAREA = 1
   INAREA = 2
   F-function

IF read a signature at R0
   INAREA = 0
   R0 = 0.

PI = 4.*ATAN(1.)

C COMMENT

C READ IN AREA DISTRIBUTION

C IF(INAREA.EQ.0) THEN
OPEN(UNIT=2,FILE='area.in')
DO 50 J=1,MAX
READ(2,*) END=75 TAU(J),S(J)
S(J) = S(J)+PFAC
50 CONTINUE
CLOSE(2)
C IF(INAREA.EQ.1) THEN
CALL WORKD(JDIN,STAU)
C IF(INAREA.EQ.2) THEN
C ENDIF

C READ IN THE GIVEN AREA DISTRIBUTION

C IF(INAREA.EQ.0) THEN
OPEN(UNIT=3,FILE='area.in')
DO 100 J=1,MAX
READ(3,*) TAU(J),S(J)
100 CONTINUE
CLOSE(3)
C IF(INAREA.EQ.1) THEN
C ENDIF

C READ IN THE F-FUNCTION OR DEFINE A F-FUNCTION BY CALLING FUNC

C and integrate the equivalent area by calling IAREA

C IF(INAREA.EQ.2) THEN
CALL IAREA(TAU,PFAC,JA)
OPEN(UNIT=4,FILE='f.dat')
DO 110 J=1,MAX
READ(4,*,END=110) TAU(J),S(J)
110 CONTINUE
C ENDIF

C READ IN THE GIVEN AREA DISTRIBUTION

C IF(INAREA.EQ.0) THEN
OPEN(UNIT=5,FILE='area.in')
DO 120 J=1,MAX
READ(5,*) TAU(J),S(J)
120 CONTINUE
C ENDIF

C READ IN THE F-FUNCTION OR DEFINE A F-FUNCTION BY CALLING FUNC

C and integrate the equivalent area by calling IAREA

C IF(INAREA.EQ.2) THEN
CALL IAREA(TAU,PFAC,JA)
OPEN(UNIT=6,FILE='f.dat')
DO 130 J=1,MAX
READ(6,*,END=130) TAU(J),S(J)
130 CONTINUE
C ENDIF

C READ IN THE GIVEN AREA DISTRIBUTION

C IF(INAREA.EQ.0) THEN
OPEN(UNIT=7,FILE='area.in')
DO 140 J=1,MAX
READ(7,*) TAU(J),S(J)
140 CONTINUE
C ENDIF

C READ IN THE F-FUNCTION OR DEFINE A F-FUNCTION BY CALLING FUNC

C and integrate the equivalent area by calling IAREA

C IF(INAREA.EQ.2) THEN
CALL IAREA(TAU,PFAC,JA)
OPEN(UNIT=8,FILE='f.dat')
DO 150 J=1,MAX
READ(8,*,END=150) TAU(J),S(J)
150 CONTINUE
C ENDIF

C READ IN THE GIVEN AREA DISTRIBUTION

C IF(INAREA.EQ.0) THEN
OPEN(UNIT=9,FILE='area.in')
DO 160 J=1,MAX
READ(9,*) TAU(J),S(J)
160 CONTINUE
C ENDIF

C READ IN THE F-FUNCTION OR DEFINE A F-FUNCTION BY CALLING FUNC

C and integrate the equivalent area by calling IAREA

C IF(INAREA.EQ.2) THEN
CALL IAREA(TAU,PFAC,JA)
OPEN(UNIT=10,FILE='f.dat')
DO 170 J=1,MAX
READ(10,*,END=170) TAU(J),S(J)
170 CONTINUE
C ENDIF

C READ IN THE GIVEN AREA DISTRIBUTION

C IF(INAREA.EQ.0) THEN
OPEN(UNIT=11,FILE='area.in')
DO 180 J=1,MAX
READ(11,*) TAU(J),S(J)
180 CONTINUE
C ENDIF

C READ IN THE F-FUNCTION OR DEFINE A F-FUNCTION BY CALLING FUNC

C and integrate the equivalent area by calling IAREA

C IF(INAREA.EQ.2) THEN
CALL IAREA(TAU,PFAC,JA)
OPEN(UNIT=12,FILE='f.dat')
DO 190 J=1,MAX
READ(12,*,END=190) TAU(J),S(J)
190 CONTINUE
C ENDIF

C READ IN THE GIVEN AREA DISTRIBUTION

C IF(INAREA.EQ.0) THEN
OPEN(UNIT=13,FILE='area.in')
DO 200 J=1,MAX
READ(13,*) TAU(J),S(J)
200 CONTINUE
C ENDIF

C READ IN THE F-FUNCTION OR DEFINE A F-FUNCTION BY CALLING FUNC

C and integrate the equivalent area by calling IAREA

C IF(INAREA.EQ.2) THEN
CALL IAREA(TAU,PFAC,JA)
OPEN(UNIT=14,FILE='f.dat')
DO 210 J=1,MAX
READ(14,*,END=210) TAU(J),S(J)
210 CONTINUE
C ENDIF
CALL D2ARC(TAU,FTAU,J-DIM,TAU,FTAU,JDIM,IO.,O)
GOTO 270
ENDIF
READ(UNIT,10,FILE-'grid.in',FORM-'UNFORMATTED')
READ(11) EDIM,JDIM,JDIM
DO 100 J=1,JDIM
READ(11) ((X(X,L,J),E-1,EDIM,I,J,EDIM),L,J,EDIM),
        ((Y(X,L,J),E-1,EDIM,I,J,EDIM),L,J,EDIM),
        ((Z(X,L,J),E-1,EDIM,I,J,EDIM),L,J,EDIM)
100 CONTINUE
CLOSE(11)
CALL SQUARE(EDIM,JDIM,EDIM,X,Y,Z,JDIM,IMAX,IMAX,LMAX,LMAX)

IF(LF(L.D)) CALL BFWM(C01JD1M,S,TAU)

B-IDIM = JDIM-1
DO 100 J=1,JDIM
   A1=(X(J)-X(J-1))/(X(J)-X(J-1))
   A2=(X(J)-X(J-1))/(X(J)-X(J-1))
   A3=(X(J)-X(J-1))/(X(J)-X(J-1))
   SP(J)=A1*(X(J)-X(J-1))
   A2=(X(J)-X(J-1))/(X(J)-X(J-1))
   A3=(X(J)-X(J-1))/(X(J)-X(J-1))
   SP(J)=A1*(X(J)-X(J-1))
   A3=(X(J)-X(J-1))/(X(J)-X(J-1))
   A2=(X(J)-X(J-1))/(X(J)-X(J-1))
   A1=(X(J)-X(J-1))/(X(J)-X(J-1))
   SP(J)=A1*(X(J)-X(J-1))
   A2=(X(J)-X(J-1))/(X(J)-X(J-1))
   A3=(X(J)-X(J-1))/(X(J)-X(J-1))
100 CONTINUE

DO 200 J=1,JDIM
   WRITE(13,580)
   WRITE(13,555)
   WRITE(13,555)
   CLOSE(13)

CALL FFLN(FTAU,TAU,FMACH,JDIM,RO,E1,TORX)
CLOSE(3)
STOP
END
SUBROUTINE LIGHT1(TAU,R,SP,N,FTAU)

DIMENSION R(N),SP(N),TAU(N),FTAU(N)

PI= 4./kT_1()
BET_ = SORT (PR&CR * 2.0)
TKU(I)=0.
PR_U(I)=0.
95 FTAU(I)=0.
DO 100 J=1,N
DO 102 I=2,N
IF (ABS(R(I)) .LE. 1.E-10) THEN
21 = 1.E+10
F4 = 0.
GOTO 98
ENDIF
AB=2.Q/BETA*R(I))
ABI=ABS(ABI)
F1=SQRT(ABI)
F2=SP(1)-SP(1-1)
F3=F1+F2
F4=F3/(2.0+F1)
ZL=TAU(I)/TAU(I)/BETA*R(I))
XLO=1.0
IF (ZL.LT.XLO) GO TO 96
IF (ZL.LT.4.0) GO TO 97
IF (ZL.GE.4.0) GO TO 98
96 BZ1=0.
97 FTAU(I)=FTAU(I)*BZ1+F4
GO TO 99
98 BZ1=.09937*11.*Z1-.2175*Z1+.7531
FTAU(I)=FTAU(I)+BZ1+F6
GO TO 99
99 CONTINUE
100 CONTINUE
RETURN
END
SUBROUTINE SQUAREA(EDIM, LDIM, JDIM, X, Y, Z, EMAX, LMAX, JMAX, S)

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 approximated by trapezoidal rule.

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

REAL S(JDIM)

DO 10 J=1, JDIM
   DAREA = 0.
   DO 5 K=2, EDIM
      X = Y(K, LDIM, J) - Y(K-1, LDIM, J)
      ADD = X*(Z(K, LDIM, J) + Z(K-1, LDIM, J))
      DAREA = DAREA + 0.5*ADD
   CONTINUE
3

The unwrapped base area:

K = Y(1, LDIM, J) - Y(EDIM, LDIM, J)
ADD = K*(Z(1, LDIM, J) + Z(EDIM, LDIM, J))
BASE = ABS(0.5*ADD)

The area surrounded by half of the plane

S(J) = ABS(DAREA) - BASE

CONTINUE
10
RETURN
END
SUBROUTINE PNT(F, I, PFACE, NP, RO, R1, TMAX)

This program uses F-function theory to predict the pressure

signature at far field when an initial pressure signature is given

PARAMETER (NMAX=1400)

DIMENSION F(NP), X(NP), Y(NMAX), P(NMAX)

DIMENSION YSTP(NMAX)

DIMENSION DRV(V, 3)

OPEN(UNIT=14, FILE='p.out')

Input of initial parameters and pressure signal

IF (RO > 0) THEN
  OPEN(UNIT=4, FILE='p.r0')
  DO 15 I=1,NMAX
   ROW(I) = SQRT(2) * SQRT(I(2) * B/I(I))
   15 CONTINUE
ENDIF

TMAX = TMAX - 1.84
THIN = THIN - 1.84
DO 55 I=1,NP
  IF (I(I)) THEN
   X(I) = Y(I) + TMAX - (TMAX - Y(I))
   55 CONTINUE

Plot the integral curve of the shifted F-function

CALL INTYP(NP, T, F)

Need to march in Y-direction, define the step

YSTP(I) = YTHIN
TDIS = TTHIN + TTHIN
DY = TDIS/FMAX(NMAX-1)
DO 80 J=1,NMAX
  YSTP(I) = YSTP(J-1) + DY
  CALL MARCH(NMAX, NP, Y, YSTP, F)

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

Obtain the solution

NOTE: IF TMAX<5, the sonic boom is in the form (F-Plat) vs time

as in it is. As in the form (F-Plat)/Plat vs distance.

DO 150 I=1,NP
  P(I) = GAMMA*PFACE*F(I)/SQT(2.*B/R0)
  X(I) = Y(I) + R0
  150 CONTINUE

Make the data points evenly distributed manner and
scale the sonic if desired

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

Atmospheric aspect

ALT = Altitude

As = speed of sound at ground level, in ft/sec

F0 = reference pressure in lb/ft^2 = SQRT(Pa*Ps)

Ps = pressure at the ground

Ps = pressure at flight altitude

Ps = SQRT(Pa*Ps)

VEL = FFACE*As

TREF = T(I)/VEL

IF (TMAX < 5), THEN
  DO 250 J=1,NP
    X(I) = X(I) + TREF
  250 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.
Note: Unit used is still the stupid English unit!

CALL WISPERM(FNACE, P, X, NP, ALT, ZERSE)

Obtain the noise level

CALL NOISE(DBVL, X, P, NP)

Write the dB(FL) value out

WRITE(14,500)DBVL(1),DBVL(2),DBVL(3)

500 FORMAT('Noise level ',F10.4,'dB',3X,F10.4,'dB(A)',3X,F10.4,'dB(C)')

ENDIF

Write the sonic boom

WRITE(14,555)R1

555 FORMAT(28DFFree pressure signal at R1= ,F10.4)

DO 670 I=1,NP

WRITE(14,700)X(I),P(I)

670 CONTINUE

700 FORMAT(3X,E20.8,2X,E16.6)

CLOSUE(14)

RETURN
SUBROUTINE INTF(NP,Y,F)

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

DIMENSION F(NP),Y(NP)

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

SUMF = 0

DO 100 J=2,NP
   DY = Y(J)-Y(J-1)
   SUMF = SUMF + 0.5*DY*(F(J)+F(J-1))

WRITE(34,130)Y(J),SUMF

100 CONTINUE

CLOSE(34)

RETURN

END
SUBROUTINE SHEET(NMAX,NP,T,YSTP,F,INDEX,F5,Y5,INSCT)

DIMENSION Y(NP),T(NP)
DIMENSION YSTP(NMAX)
DIMENSION INDEX(NMAX),FS(40)

COMMON/SHOC/ INSCT

YEND = Y(NP)
FIRST = 1.
DO 500 J=1,NMAX
TS = YSTP(J)

CALL POINT(NP,T,F,INDEX,FS,YSTP,INSCT)

IF(INST.GT.2) THEN
  INSCT = 3
  IE = INDEX(INST)
  CALL AREA(NP,T,F,TS,F5,YS,INDEX)
ELSE
  The tail shock is already formed, leave program
  FIRST = 1.
  RETURN
ENDIF

IF(FIRST.GT.0.) IFLAT1 = IFLAT2
IF(IFLAT2.EQ.0) RETURN
FIRST = -1.

IF(IFLAT = 0) TS is the point that have area balanced,
IF(IFLAT2 and IFLAT are in different signs, i.e.,
  Use bisection method to find the correct point(T(START)

IF(IFLAT1+IFLAT2 LT 0.) THEN
  Y1 = YSTP(J-1)
  Y2 = YSTP(J)
  NC = 500
  DO 300 IC=1,NC
    YS = 0.5*(Y2+Y1)
    CALL POINT(NP,T,F,INDEX,FS,Y5,INSCT)
    IF(INSCT.EQ.3) INSCT = 3
    IE = INDEX(INST)
    IE = INDEX(INST)
    CALL AREA(NP,T,F,TS,F5,YS,INDEX)
    IF(IFLAT0 EQ 0) RETURN
    IF(IFLAT0+IFLAT1 LT 0.) THEN
      Y2 = YS
    IFLAT2 = IFLAT0
    ELSE
      Y1 = Y5
      IFLAT1 = IFLAT0
ENDIF
200  CONTINUE
WRITE(*,*)`After ',NC,' steps of bisection'
RETURN
ELSE
IF(IFLAT = IFLAT2
GOTO 500
ENDIF
FIRST = 1.
500  CONTINUE
RETURN
END
SUBROUTINE POINT(NP,Y,F.INDEX,F5,YS,INSCt)  
DIMENSION Y(NP),F(NP)  
DIMENSION INDEX(NP),FS(40)  
C Find the points F6 on the F-function when YS is given  
INDEX = the index runs from 1 to NP  
INSCt - f of points being intersect, at least) point to do integration  
IF(YS AT Y(1)) THEN  
INSCt = INSCt + 1  
FS(INSCt) = 0  
INDEX(INSCt) = 1  
ENDIF  
DO 100 J2,NP  
FAC2 = YS - Y(1)  
IF(FAC2 = 0) THEN  
ENDIF  
INSCt = INSCt + 1  
SL = (F(1)-F(1-1))/(Y(1)-Y(1-1))  
FS(INSCt) = F(1-1)+SL*(YS-Y(1-1))  
 INDEX(INSCt) = 1  
ENDIF  
CONTINUE  
IF(YS GT Y(NP)) THEN  
INSCt = INSCt + 1  
FS(INSCt) = Y(NP)  
INDEX(INSCt) = NP  
ENDIF  
RETURN  
END
SUBROUTINE AREAL \(NP,T,F,Y5,PS,INE,IFLAT)\)

DIMENSION Y(5P), F(5P), T5(45)

COMMON/SHOCK/ INE

C Find the integral of F by trapezoidal rule
C Integrating from \(I=I\) to \(I=I\). \(E1\) is area that from \(Y\) to \(Y\).
C and \(E2\) is the area that from \(Y(I)\) to \(Y(I)\). Thus \(E1\) should be
C subtracted out and \(E1\) should be added in.

E1 = 0.5*(Y(5P)-Y5)*(F(5P)+F5(5P))
E1 = 0.5*(Y(I)-Y5)*(F(I)+F5(INE))
AREAL = E1

DO 10 I = 1,5

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

10 CONTINUE

C A = AREAL - E2
IF(A.GT.0) IFLAT=1
IF(A.LT.0) IFLAT=-1
IF(NB(A).LT.1.E-7) IFLAT=0

C RETURN
END
SUBROUTINE WBODY(JDIM,S,TAU)

C This subroutine finds the area distribution of
C the wing-body configuration.

DIMENSION S(JDIM), TAU(JDIM)

PI = 4.*ATAN(1.)
ANG = 21.*PI/180.
ANG1 = 80.*PI/180.
DIM = 25.52/FLOAT(JDIM-1)

C

S(1) = 0.
DO 2 J = 2, JDIM
   TAU(J) = TAU(J-1)*DX
   TTT = TAU(J)-.01
   IF(TTT.GT.0.) TTT = 0.
   RR = 0.54-0.011*TTT**2
   S(J) = PI*RR*5.
      AA = 4.*0.5*0.05*TAN(ANG)*(TAU(J)-8.21)**2
      S(J) = S(J) + AA
   ENDIF
   IF(TAU(J).GT.12.25 .AND. TAU(J).LT.15.7768849) THEN
      B1 = 0.05*(16.29-TAU(J))
      B2 = 2.31-((TAU(J)-12.25)/(15.7768849-12.25))
      B3 = (TAU(J)-8.21)*TAN(ANG)-B2
      B4 = 0.05*B1*TAN(ANG)
      AA = 4.*0.5*B1*B2+0.5*(B1+B3)*B2
      S(J) = S(J) + AA
   ENDIF
   IF(TAU(J).GT.15.77688849 .AND. TAU(J).LT.16.29) THEN
      S(J) = S(J) + AA
   ENDIF
   IF(TAU(J).GT.16.29 .AND. TAU(J).LT.17.52) THEN
      SLOP = 0.15-0.05*TAN(ANG)*((16.29-TAU(J))**2)
      (RR = 0.54+SLOP*(TAU(J)-17.52)
      S(J) = PI*RR**2
   ENDIF
   IF(TAU(J).GT.17.52) THEN
      S(J) = PI*0.15*0.15
   ENDIF
   CONTINUE
RETURN
END
SUBROUTINE CONE(JDIM,5,TAU)

C This subroutine finds the area distribution of the cone-cylinder.
C with half-angle 3.24 degree and 8.8 units of length.

DIMENSION S(JDIM),TAU(JDIM)

PI=4. *ATAN(1.)
ANG = 3.24*PI/180.
DX = 16./PI/60.(JDIM-1)
S(1) = 0.

DO 2 J=2,JDIM
  TAU(J) = 0

   IF(TAU(J) .LE. g.6) THEN
      R = TAU(J) + TAN(ANG)
      ELSE
      R = 8.6*TAN(ANG)
   ENDIF

   S(J) = PI*R*R

2 CONTINUE
RETURN
END
SUBROUTINE SEARS(JDIM, S,TAD)
C
C This subroutine finds the area distribution of the Sears-Adams body
C
C with fineness ratio 23.5
C
C DIMENSION S(JDIM), TAD(JDIM)
C
C I of point on body + I of point on sting = JDIM
C
C JBDY = JDIM/2, JBDY-1
C
C P = 23.5
C
C DTETA = PI/FLOAT(JBDY-1)
C
C RX = BL/2 * PI
C
C S(1) = 0
C
C TAD(I) = 0
C
C Consists of Sears-Adams body
C
C write(4,'(2)*Input Abs/Amx')
C
C read(*,'(2)') AB
C
C write(4,'(2)*Input RXMAX')
C
read(4,'(2)') AA
C
C RXMAX = AA*BL
C
C CONST = AB/PI
C
C CI = 1/(2) * (2.0*RXMAX/BL - 1.0)
C
C DO 2 J=2,JBDY
C
C THETA = PI-DTETA*FLOAT(J-1)
C
C TAD(J) = FLOAT(J-1)+PI
C
C THETA = ACOS(2.0*THETA/(TAD(J))
C
C Sears-Adams body
C
C POS = SIN(THETA)**2
C
C Sears-Adams body
C
C POS = CONST* (1.0-SIN(THETA))
C
C (4.0/3.0)*CI* (SIN(THETA))**3
C
C B = RXMAX-SORT( ABS(POS) )
C
C J(J) = PI*R+B
C
C CONTINUE
C
C ADD a sting
C
C DX = (TOTAL-BL)/FLOAT(JBDY)
C
C DO 3 J=2,JBDY+1,JDIM
C
C TAD(J) = TAD(J-1) + DX
C
C S(J) = S(J-1) + DX
C
C CONTINUE
C
RETURN
C
END
SUBROUTINE BULLET(JDIM, S, TAU)

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

R = AT GAMS

DIMENSION S(JDIM), TAU(JDIM)

PI=4.*ATAN(1.)

RBASE = 0.25

A = RBASE/(BL**GAMA)

TOTLEN = BL + 2.*BL

DX = TOTLEN/FLOAT(JDIM-1)

S(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

S(J) = PI*R*R

END

CONTINUE

RETURN
SUBROUTINE WING(JDIM, S, TAU)
   DIMENSION S(JDIM), TAU(JDIM)

   PI = 4. * ATAN(1.)
   STING = 0.
   DX = 1. / FLOAT(JDIM - 1)
   S(1) = 0.
   TAU(1) = 0.
   DO 2 J = 2, JDIM
      TAU(J) = TAU(J - 1) + DX
   1   IF (TAU(J) LT 1) THEN
       S(J) = 0.5
   2   IF (TAU(J) GT 1.70897) THEN
       STING = PI * 0.0625 * 0.0625
       S(J) = PI + STING - 1 * 1.25
   3   ELSE
       STING = PI * 0.0625 * 0.0625
       S(J) = STING
   4   ENDIF
   CONTINUE
   RETURN
END
SUBROUTINE BFUNC(JDIM, JDIM)  
  C This subroutine obtains the B-function from fort.10 and add it  
  C into the equal-weight area.  
  C
  PARAMETER (NMAX=800)  
  DIMENSION S(JDIM), TAU(JDIM), B(NMAX), X(NMAX)  
  COMMON/PAR/ PARAC, PPAC
  C
  OPEN (UNIT=33, FILE='coe.dat', FORM='FORMATTED')  
  C
  READ (33,12)
  READ (33,12)
  READ (33,12)
  READ (33,12)
  READ (33,12)
  DO 19 I=1,NMAX
  READ (33,15,END=17) E(1), CL, CD, SLOD, B(1), CH
  READ (33,15,END=17) E(1), B(1)
  B(I) = B(I)+PPAC  
  10 CONTINUE
  11 CONTINUE
  12 CONTINUE
  13 FORMAT(6,13.5)
  14 CONTINUE
  15 CLOSE(33)
  16 NPOINT = I-1
  17 OPEN (UNIT=33, FILE='b(np.dat')
  18 DO 30 I=1,NPOINT  
  19 WRITE(33,*) A(I), B(I)
  20 CONTINUE
  C
  ISTART=1
  DO 50 J=1,JDIM
  READ (33,17,END=50) X(J), TAU(J), PROX
  IF(ABS(X(J)-TAU(J)) LE.1.E-10) THEN
    S(J)=S(J)+B(J)
  ELSE IF(X(J)-TAU(J) LE.1.E-10) THEN
    S(J)=S(J)+B(J)
  ELSE IF(X(J)-TAU(J) LE.1.E-10) THEN
    S(J)=S(J)+B(J)
  ENDIF  
  30 CONTINUE
  C
  IF(JJ1.LE.NMAX) THEN
    X(J) = X(J) - S(J) + B(J)
    IF(J.JE.1) THEN
      B(J) = S(J) + B(J)
    ELSE IF(J.GT.1) THEN
      B(J) = S(J) + B(J)
    ENDIF  
  50 CONTINUE
  C
  RETURN
  END
SUBROUTINE WB(JDIM, SP, TAU)
C This subroutine obtains the wing-body interference correction
C and adds it into the derivative of equivalent area
C
DIMENSION SP(JDIM), TAU(JDIM)
COMMON/PAR1, PMAC, PFAC

DO 10 J=1,JDIM
   IF(TAU(J).GE.8.2 .AND. TAU(J).LE.12.25)
      SP(J) = SP(J) + 4.*0.54
      SP(J) = SP(J) - 4.*0.54
      SP(J) = SP(J) - 4.*0.54
10 CONTINUE
RETURN
END
SUBROUTINE F_NC(TAU,FTAU,JDIM)

DIMENSION TAU(3),FTAU(3)

NAMELIST /FTNC/( TF, ELAM, C, B, D, E, BL, TR, DEL)

READ(1,FTNC)

WRITE(6,FTNC)

TAU(1)=0.

IF (TF.EQ.0.) GOTO 6

IF(TAU(2).LE.TF/2.) TAU(3)=TAU(3)+TF/2.

DO 10 J=2,JDIM

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

IF(TAU(J).GE.TF) GOTO 6

10 CONTINUE

WRITE (13,80)

DO 20 J=1,JDIM

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

20 CONTINUE

WRITE(13,75)

DO 21 J=1,JDIM

21 CONTINUE

WRITE(13,80)

FORMAT(2X, F8.4, IX, EL6.8 }

RETURN

END
SUBROUTINE AREA(S,FTAU,TAU,JDIM)

DIMENSION S(JDIM),TAU(JDIM),FTAU(JDIM)

A(X) = \int_{0}^{X} \frac{f(t)/\sqrt{t-y}}{dy}

S(I) = 0.
TAU(I) = 0.
DO 10 J=1,JDIM
SS = 0.
DO 9 J=1,J
DO 9 I=1,J
FTAU(I) = S(I)/TAU(I)
FINHEL = 0.
DO 9 I=1,J
DT = TAU(I+1)-TAU(I)
FINHEL = FINHEL + DT*FTAU(I)/SQR(TAU(I)-TAU(I))
9 CONTINUE
FINHEL = FINHEL + SS*X.*SQR(TAU(I+1)-TAU(I))*S(TAU(I+1)^)*DT
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
10 FORMAT(22,F8.4,F12.8)
RETURN
END
SUBROUTINE RISETIME(PMACH,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 units used are English units !!!
PMACH = Free-stream Mach number
P = Sonic boom
PSH = Shock strength
ALT = Altitude (ft)
P0 = Free-stream pressure (lb/ft2)
PR = Rise time (sec)
TEMP = Temperature K+459.67=(9/5)K
DIMENSION (HP,T(HP)
PS = 2116.3
TEMP = 518.69
ICOUNT = 0
CONTINUE
ICOUNT = ICOUNT + 1
CONTINUE
DO 30 I=1,HP
ENDIF
CONTINUE
IRIS = 1
IF(I(PSH.EQ.0.) RETURN
IF(I(IRS.GT.1)) THEN
Now calculate the rise time using Air Force data base
V1 = 2.9*PMACH - 7.38
V2 = V1 + (V1+PMACH) - 2 - 45.9*PMACH - 62.9)/**(1.5
AK1 = V2 * 1000.
AR = AK1 / ((AK1/1000. - 5.)*2117.
V10 = 100. * 0.5*(TEMP-410.)
BT = (AK-V12)/P0/PSH+TEMP
BT = BT/1000.
ELSEIF(IRS.EQ.2) THEN
Now calculate the rise time assuming lpsf has linear rise time
BT = 0.003/PSH
ENDIF
WRITE(14,80)BT
80 FORMAT(165 RISE time (sec) of the bow shock is,10.5)
Originally, T(IRS)=T(186) with infinite shock strength, now create a
signal with the rise time, between the index IRS to IRS
Also extend the signal by the amount of rise time
BRT = BT/FLOAT(186-IRS)
DO 200 I=IRS,IRS-1
T(I-1) = T(I) + BRT
200 CONTINUE
DO 300 I=IRS+1,HP
T(I) = T(I) + BT
300 CONTINUE
IF(ICOUNT.LT.10) GOTO 12
RETURN
END
SUBROUTINE DISTARC(X,Y,N,NNEW,THET,NNEW,FAC,IFLAT)

DIMENSION X(N),Y(N),NNEW(NNEW),THET(NNEW)

C This program redistributes the points (X,Y) by subroutine DISTRI
C based on the arc length. FAC is the final grid spacing. Note that
C the end points of the two sets are the same.
C IFLAT=0, grid points will connect near the first point, -1 near the end.
C Input array is (X(1),Y(1)), i=1,...,N
C Output array is (X(NNEW(1),NNEW(1))), i=1,...,NNEW

PARAMETER (MAX=2000, SMAX=50)
DIMENSION S(MAX),TOTARC(MAX),SMAX(MAX),TH(MAX)

C Maximum number of points allowed is MAX
IF(MAX.LE.N .OR. MAX.LE.NNEW) THEN
WRITE(*,'*') 'SUB DISTARC : MAX is less than N or NNEW'
STOP
ENDIF
C 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
C Apply subroutine DISTRI to obtain the stretching function S
IF(FAC.GT.1.) THEN
DELT=TARC(N)/FLOAT(NNEW-1)
CALL DISTRI(DELT,NNEW,S,IFLAT)
ELSE
S(1) = 0.
DO 20 K=2,NNEW
S(K) = S(K-1) + 1./FLOAT(NNEW-1)
20 CONTINUE
ENDIF

C Redistribution, put new array in a temporary array XM and YM
XM(1)=X(1)
YM(1)=Y(1)
XM(NNEW)=Y(N)
YM(NNEW)=X(N)

DO 40 J=2,NNEW
ARKEN = S(J)/TOTARC(N)
DO 50 K=1,J
IF(TOTARC(K).EQ.ARKEN) THEN
XM(J) = X(K)
YM(J) = Y(K)
50 GOTO 40
ENDIF

IF(TOTARC(K).GT.ARKEN) THEN
IX = X(K-1)
I2 = X(K)
I1 = Y(K-1)
T2 = Y(K)
IX = X1 + (X(K)-X(K-1)) * (ARKEN-TOTARC(K-1))/(TOTARC(K)-TOTARC(K-1))
CALL LIMIT(X1,X2,IX,T2,TT)
XM(J) = IX
YM(J) = TT
GOTO 60
60 CONTINUE
40 CONTINUE
55 CONTINUE
60 CONTINUE
C Write the temporary arrays into the output NNEW, THEN
C Do 70 J=1,NNEW
C XM(J) = XM(J)
C YM(J) = YM(J)
C THEN(J) = TH(J)
70 CONTINUE
RETURN
END
SOURCE PROGRAM
LHF.f

LINE # | SOURCE TEXT
--- | ---
1077 | 1078 | SUBROUTINE DISTRI(D,PC,5,FPC)
1079 | 1080 | PARAMETER (MAX=500)
1081 | 1082 | DIMENSION S(MAX),DUM(MAX)
1083 | 1084 | C Calculating the stretching function S when given
1085 | 1086 | the first spacing, FPC, and the number of points FPCS
1087 | 1088 | IF(FINE=1), distribution is clustering at outer grid
1089 | 1090 | IF(FINE.LE.FPCS) THEN
1091 | 1092 | WRITE(*,*) 'SUB DISTRI : MAX is less than FPCS'
1093 | 1094 | STOP
1095 | 1096 | ENDIF
1097 | 1098 | IF(FINE.EQ.1) THEN
1099 | 1100 | D1 = FPC
1101 | 1102 | EPS = EPS+1
1103 | 1104 | DIETA = 1.0/FLOAT(EPS)
1105 | 1106 | KD = 1.5
1107 | 1108 | CALL GEBE(D1, EPS, C, 0.0001, 100, KDBETA)
1109 | 1110 | CALL FILL(FPCS, KDBETA, DIETA, S)
1111 | 1112 | C IF (FINE.EQ.1) THEN
1113 | 1114 | DO 17 B=1,FPC
1115 | 1116 | D1 = 1.0/FLOAT(FPC)
1117 | 1118 | CONTINUE
1119 | 1120 | DO 18 B=1,FPC
1121 | 1122 | D1 = 1.0/FLOAT(FPC)
1123 | 1124 | CONTINUE
1125 | 1126 | ENDIF
1127 | 1128 | RETURN
1129 | 1130 | END
***UNIX FORTRAN Program***

**SOURCE PROGRAM**

**LHF.f**

**SOURCE TEXT**

```
113 SUBROUTINE F1(I1,THETA,DET,Z)
114 C COMPUTES NORMALIZED NORMAL DISTANCE, Z(i)
115 DIMENSION Z(250)
116 IF(TBETA.EQ.1.) THEN
117 DO 10 I=1,111
118 Z(I)=0.
119 10 CONTINUE
120 ELSE
121 ETA=-(I-1)*DET
122 R=TBETA-1.1/(TBETA-1.)
123 EEE=1.-ETA
124 RRETA=RR-EEE
125 Z(I)=(TBETA-1.)*RRETA+TBETA)/(TBETA+1.)
126 END IF
127 RETURN
128 END
```
SUBROUTINE LINT(X1,X2,Y1,Y2,YLOCAL,YLOCAL)

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

IF(X1.EQ.X2) THEN
   YLOCAL = (Y2-Y1)/2.
ENDIF

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

YLOCAL = SLOPE*(XLOCAL-X2) + Y2

100 CONTINUE

RETURN

END
SUBROUTINE MARCH(NMA,NP,Y,YSTP,F)
DIMENSION F(NP),Y(NP)
DIMENSION YSTP(NMAX)
DIMENSION INDEX(40),FS(40)
DIMENSION S(40)
COMMON/Shock, INST

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

KOUNT = 0
YEND = Y(NP)
DO INDO=1,40
INDEX(INDO)=0
INSCT = 0
ENDO

CALL SRFFT(NMAX,NP,Y,YSTP,F,INDEX,FS,YS,0)

For tall shock, we need to check the possible positions of shock.
IF(YS.GT.0) .AND. ((YSTP(NMAX)-YSTP(1))) GOTO 400

Only one possible location of shock
IF(INST.EQ.3) GOTO 400

More than one possible locations of shock
IF(INST.GE.5) THEN
YSHE = YS
CALL SEFFT(NMAX,NP,Y,YSTP,F,INDEX,FS,YS,0)
The wing shock overcomes
CALL SEFFT(NMAX,NP,Y,YSTP,F,INDEX,FS,YS,0)
GOTO 400
ELSE
There are two separated shocks
For the shock is actually locate on the turning edge of f-function
we need to relocate it.

Fix the Yl and Y3 of this small region
IF(YS.GT.INDEX(INST)) THEN
Y2 = YS
BIG = 0.
DO ITEST=INDEX(INST)+1,NP
IF(YS.GT.Y(I)) AND. ABS(YS-Y(I))G.T.BIG THEN
Y3 = Y(I)
BIG = ABS(YS-Y(I))
ENDIF
ENDIF
IF(Y(I).GT.YS) GOTO 300
300 CONTINUE

Find YS by bisecting Yl and Y3
NC = 500
DO 320 IC=1,NC
YS = 0.5*(Y2+Y1)
CALL POINT(NP,Y,F,INDEX,FS,YS,INSCT)
DO IT=INSCT+1,IC
IF(INDEX(IT).LE.INSCT) THEN
INSCT = IT
GOTO 310
ENDIF
310 CONTINUE
IS=INDEX(INSCT)
IF=INDEX(INST)
CALL AREA(NP,Y,F,FS,YS,IS,IE,IFLAT0)
IF(IFLAT0.EQ.0) GOTO 400
IF(IFLAT0.GT.0) THEN
Y2 = YS
ELSE
Y1 = YS
ENDIF
320 CONTINUE
WRITE(*,'(A)') 'After ' .NC. ' steps of bisection'
ENDIF
GOTO 400
ENDIF
ENDIF
400 CONTINUE
Form the shock
IF(INST.LT.1. AND. YS.GE.YEND) RETURN
IF(IS = INSCT-1) THEN
IF(IFLAT0.EQ.0) THEN
WRITE(*,'(A)') 'SDT: ZERO DIVISION HAPPEN IN MARCH'
RETURN
ENDIF
IF(IS.EQ.0) THEN
WRITE(*,'(A)') 'SDT: ZERO DIVISION HAPPEN IN MARCH'
RETURN
ENDIF
IF(IS.EQ.1) THEN
WRITE(*,'(A)') 'SDT: ZERO DIVISION HAPPEN IN MARCH'
RETURN
ENDIF

450 CONTINUE
IF(YEQT.EQ.20) THEN
WRITE(*,'(A)') 'EQUIT = 201'
RETURN
ELSE
EQUIT = EQUIT+1
GOTO 100
ENDIF
END
SUBROUTINE SIMPSON(X,F,N,X0,X1,SM)

C

N = N/2
SM = 0.
DO 10 J=1,N
   I = 2*J-1
   ODD = ODD + 2.*F(I)
   EVEN = EVEN + 4.*F(I+1)
10 CONTINUE
SM = F(1) + ODD + EVEN + F(N)
SM = SM*(X1-X0)/(6.*FLOAT(N))
RETURN
END
Appendix B

SAMGRID (Fortran Listing)
PROGRAM SANGRID
include 'sgrid.com'

Dr. Samuel Cheung
Revision: 1.1
Date: Dec., 1993
Version 3.0

This subroutine reads a surface grid in airfoil
sections and reformats it to produce
a surface grid of axisymmetric cross-sections.

10 Date: Dec., 1993
15 Read input geometry

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

110 ISEG = 1 of sections (streamwise stations) of the new grid
115 NPTS = # of pts in the circumferential direction (MUST be odd)
120 NPTX = max streamwise stations
125 FAC = the first grid spacing is DIST /
130 XL = X leading edge
135 ARENWW = 0. arrow wing
140 ETIP = number of points in the wing-around direction on one surface
145 MD = number of cut in the spanwise direction
150 NP = number of points in the upper part of the wing
155 NL = number of points in the lower part of the wing

160 Read surfgrid dimensions (nsec x npts x 1)
165 NAMELIST /WING/, NSEC, NPTS, FAC
170 NAMELIST /WING/, ARENW, ETIP
175 WRITE(*,*, WING)

180 Read the input grid

CALL WING

190 Setup distribution of cross-sections to be obtained (solid)
200 TIP=ETIP
205 ETDEX=ETDE+CHORD/2
210 XDEXT=XDE+CHORD/2
215 IF(ETDE>ETIP) THEN
220 ETDE=ETDE
225 ARENW=-1.
230 ELSE
235 ETDE=ETDE
240 ARENW=1.
245 ENDIF
250 DO 100 J=1,NSEC
255 DIST(J)=XDEXT-XDE*(XDE-XDEXT)/(XDE-XDEXT+1.0E-10)
260 CONTINUE
265 ETIP=(NPTX+1)/2
270 THE NOSE OF THE WING
275 DO 180 J=1,NPTS
280 XOUT(J,K)=EDIST(1)
285 YOUT(J,K)=YNEW(K)
290 ZOUT(J,K)=ZST(K)
295 CONTINUE
300 BEGIN MAIN LOOP FOR EACH X-STATION
310 DO 300 K=1,NSEC
320 LOCAL-EDIST(L)
330 redistribute the points from spanwise out to streamwise out.
340 the output EDIST and YNEW are from the root to the tip, therefore
350 LOCATE the lower surface used to rearrange the arguments.
360 the output (EDIST,YNEW) in both surfaces have ETIP # of pts in
370 the circumferential direction, their last point have the same physical
380 value for both surfaces.
390 Do the lower surface
400 CALL REDIST(LOCAL, EDIST, FAC, 1)
410 DO 300 I=1,ETIP
415 XOUT(I,L)=LOCAL
420 YOUT(I,L)=YNEW(I)
425 ZOUT(I,L)=ZST(I)
430 CONTINUE
435 Do the upper surface
440 CALL REDIST(LOCAL, ETIP, FAC, 1)
445 DO 400 I=ETIP-1,NPTS
450 XOUT(I,L)=LOCAL
455 YOUT(I,L)=YNEW(I)
460 ZOUT(I,L)=EDIST(I)
465 CONTINUE
470 FOR THE COMPUTATIONAL GRID OF GERRIS CODE
475 the wake has to have two different pts in same
480 physical location, such that (Y1,Y2)=(Y2,Y1)
485 FOR SAFETY'S SAKE, SET E=0
490 IF(LOCAL .LT. WAKE) GOTO 300
495 DO 300 K=1,ETIP-1
500 K=ETIP+K
505 ETIP=K
510 IF(ABS(YOUT(L,K1)-YOUT(L,K2)) .LE. 1.0E-4) THEN
515 IF(ABS(YOUT(L,K1)-YOUT(L,K2)) .LE. 1.0E-2) THEN
520 IF(ABS(YOUT(L,K1)-YOUT(L,K2)) .LE. 1.0E-5) THEN
525 ZOUT(L,K1)=ZOUT(L,K2)
530 ENDIF
535 CONTINUE
540 CONTINUE
545 CONTINUE
550 CONTINUE
555 CONTINUE
560 CONTINUE
565 CONTINUE
570 CONTINUE
575 CONTINUE
580 CONTINUE
585 CONTINUE
590 CONTINUE
595 CONTINUE
600 CONTINUE
605 CONTINUE
610 CONTINUE
615 CONTINUE
620 CONTINUE
625 CONTINUE
630 CONTINUE
635 CONTINUE
640 CONTINUE
645 CONTINUE
650 CONTINUE
655 CONTINUE
660 CONTINUE
665 CONTINUE
670 CONTINUE
675 CONTINUE
680 CONTINUE
685 CONTINUE
690 CONTINUE
695 CONTINUE
700 CONTINUE
705 CONTINUE
710 CONTINUE
715 CONTINUE
720 CONTINUE
725 CONTINUE
730 CONTINUE
735 CONTINUE
740 CONTINUE
745 CONTINUE
750 CONTINUE
755 CONTINUE
760 CONTINUE
765 CONTINUE
770 CONTINUE
775 CONTINUE
780 CONTINUE
785 CONTINUE
790 CONTINUE
795 CONTINUE
800 CONTINUE
805 CONTINUE
810 CONTINUE
815 CONTINUE
820 CONTINUE
825 CONTINUE
830 CONTINUE
835 CONTINUE
840 CONTINUE
845 CONTINUE
850 CONTINUE
855 CONTINUE
860 CONTINUE
865 CONTINUE
870 CONTINUE
875 CONTINUE
880 CONTINUE
885 CONTINUE
890 CONTINUE
895 CONTINUE
900 CONTINUE
905 CONTINUE
910 CONTINUE
915 CONTINUE
920 CONTINUE
925 CONTINUE
930 CONTINUE
935 CONTINUE
940 CONTINUE
945 CONTINUE
950 CONTINUE
955 CONTINUE
960 CONTINUE
965 CONTINUE
970 CONTINUE
975 CONTINUE
980 CONTINUE
985 CONTINUE
990 CONTINUE
995 CONTINUE
1000 END
LINE #  SOURCE TEXT
14 WRITE(50), NPT$ , EW , NSEC
15 DO 1334 I = 1 , NSEC
16 WRITE(50), ( YOUT(I,E), E = 1 , NPT$ )
17 ( YOUT(I,E), E = 1 , NPT$ )
18 1334 CONTINUE
19 WRITE(II), NI , NC , NW
20 WRITE(II), ( XBASE(I,2,M), I = 1 , NU ) , ( YBASE(I,2,M), I = NL, 1 , -1 )
21 ( XBASE(I,2,M), I = NL, 1 , -1 )
22 READ the fuselage grid and combine the fuselage with the wing grid to form a whole configuration.
23 CALL HKGRID
24 READ the aeroles grid and combine the aeroles with the wing-body grid.
25 CALL HKGRID
26 CLOSE(10)
27 CLOSE(30)
28 STOP
29 END
SUBROUTINE ADDGRID(NP11,NP12,NP2,NP1,NSEC,EDIM)

This subroutine allows us to add a grid line between streamwise section
NP11 and NP12, and the new dimension is NSEC axis.

PARAMETER (MAX=100)
DIMENSION YTEMP(MAX),XTEMP(MAX)
DIMENSION X(NP1),Y(NP1,NP1),Z(NP1,NP1)

IF(MAX LE NP1) THEN
  WRITE(",","SUB ADDGRID : MAX is less than NP1")
  STOP
ENDIF

IF(NP11 GE NP12) 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(NP1)
II = Y(NP1)
II = 0.5*(X(NP1)+X(NP11))
DO 10 X=1,EDIM
  Y1 = Y(NP11,E)
  Y2 = Y(NP12,E)
  Z1 = Z(NP11,E)
  Z2 = Z(NP12,E)
  CALL LINEP(X1,Y1,Z1,X2,Y2,Z2)
  CALL LINEP(X1,Y1,Z1,X2,Y2,Z2)
  YTEMP(E) = Y1
  XTEMP(E) = X1
10 CONTINUE

C Renumber the late stations
NSEC - NSEC1
DO 20 = 1,NSEC,NP2+1,-1
  X(L,E) = X(L-1,E)
  Y(L,E) = Y(L-1,E)
  Z(L,E) = Z(L-1,E)
20 CONTINUE

C Put the temporary array in the grid
DO 50 = 1,EDIM
  X(NP11,E) = X
  Y(NP11,E) = YTEMP
  Z(NP11,E) = ZTEMP
50 CONTINUE

RETURN
END
SUBROUTINE CIRCLES(EK, XI, MAX, X1, RFIL, ANCORR)

DIMENSION XI(MAX), X(MAX), Y(MAX), I, J, RFIL, ANCORR

C Given a set of pts (X(I),Y(I)) I=1,...,MAX, and radius of fillet RFIL,
C this subroutine replaces the points (X(I),Y(I)) I=ES,...,EK by the fillet
C points on fillet circle.

C Look for the center of the fillet circle (YC,SC)
TAY = (Y(ES) - Y(EK)) / (ES - EK)
TA = (X(ES) - X(EK)) / (ES - EK)

B = 2 * (X(ES) - X(EK)) + (Y(ES) + Y(EK))
C = 2 * (Y(ES) + Y(EK)) - (X(ES) + X(EK))

A = -B * B / 4 - 2 * RFIL

B = (X(ES) + X(EK)) / 2
C = (Y(ES) + Y(EK)) / 2

DET = B * B * B - 4 * A * C

IF (DET .LT. 0) THEN
  WRITE(15,*) 'Detemrnant is less than 0', 'DET
  GO TO 200
ENDIF

YC1 = TAY + (SQR(DET)) / (2 * A)
SC1 = BB + (2 * TAY + TB) * (B - B / 2) * C)

YC2 = TAY + (SQR(DET)) / (2 * A)
SC2 = BB + (2 * TAY + TB) * (B - B / 2) * C)

IF (YC1 .GE. YC2) THEN
  SC = SC1
  YC = YC1
ENDIF

GO TO 50

50 C Find the total arc length given
TWARC = 0.
DO 50 K = ES, EK - 1
  SORT( Y(K+1)-Y(K), K+1 - K)
  ARC = TWARC + (K+1 - K) * (Y(K+1)-Y(K))
  TWARC = ARC
ENDDO

50 C Find the arc length between two points.
ARC = TWARC + (EK - ES) * (Y(ES) - Y(EK))

GO TO 55

55 C Find the coordinates for each point
DO 100 K = ES, EK - 1
  TA = Y(K)
  ZA = X(K)
  TB = YC
  ZB = SC
  SY = TA - YB
  SZ = TA - ZB

  B = 2 * TAY - 2 * TB - BB / 2
  C = 2 * TAY + TB + (BB / 2) * SC

  A = 1 * B * B - 4 * A * C
  DET = B * B * B - 4 * A * C
  IF (DET .LT. 0) THEN
    WRITE(15,*) 'Detemrnant is less than 0', 'DET
    GO TO 200
  ENDIF

  YC1 = (B * SQR(DET)) / (2 * A)
  SC1 = BB + (2 * TAY + TB) * (B - B / 2) * C)
  YC2 = (B * SQR(DET)) / (2 * A)
  SC2 = BB + (2 * TAY + TB) * (B - B / 2) * C)

  GO TO 200

200 RETURN

END
SUBROUTINE SPLINE(N,NMAX,THEN,THEN0)

PARAMETER (NMAX=100)

REAL X(N), Y(N), DEL(NMAX), THEN(NMAX)

C

THIS SUBROUTINE PROCEDES A MONOTONE CUBIC SPLINE INTERPOLANT
TO THE DATA (X(I),Y(I)) 1<=I<=N AND COMPUTES VALUES AT
THE NEW POINTS THEN(I), 1<=I<=NMAX. THESE ARE RETURNED IN
ARRAY THEN(I). THE ALGORITHM USED IS THAT OUTLINED BY PRITZK AND

C

....BETWEEN BY JEFF CORDOVA 10/31/86

C

REAL D(NMAX), DEL(NMAX), R(NMAX)

C

MESH SPACING AND FIRST DIVIDED DIFFERENCE

DO 100 I=1,N-1
   M(I) = X(I+1) - X(I)
100 CONTINUE

C

DO 200 I=1,N-1
   DEL(I) = (Y(I+1) - Y(I)) / M(I)
200 CONTINUE

C

SPLINE COEFFICIENT EVALUATION

C

DO 300 I=1,N
   S(I) = 0
300 CONTINUE

C

DO 400 I=1,N
   IF (N .GE. 2) THEN
      S(I) = DEL(I)
      D(I) = DEL(I)
   GO TO 500
500 CONTINUE

C

ELSEIF (PCBST(D,I),DEL(I),DEL(2),LT.0.) THEN
   DEL(I) = 0.
600 CONTINUE

C

DO 400 I=1,N
   IF (ABS(D(I)) .GT. ABS(MAX)) D(I) = MAX
700 CONTINUE

C

INTERIOR POINTS (HERALD MODIFICATION OF BUTLAND FORMULA)

DO 100 I=1,N-1
   TOP = DEL(I) * DEL(I)
   IF (TOP .LE. 0.) THEN
      S(I) = S(I) - 1. / 3.
   ELSEIF (PCBST(D(I),DEL(I),DEL(2)),LT.0.0.) THEN
      S(I) = MAX(1.,DEL(I)) + 3.*DEL(I)
   ENDIF
100 CONTINUE

C

LAST BOUNDARY POINT (USE THREE POINT FORMULA ADJUSTED TO BE
SHAPE PRESERVING)

DO 500 I=1,N-1
   IF (I.EQ.1) THEN
      S(I) = (X(I) + 1.75*X(I+1) - 1.75*X(I-1)) / (2.*M(I))
   ELSEIF (PCBST(D(I),DEL(I)-1),LE.0.) THEN
      S(I) = S(I) - 0.178*DEL(I)
      D(I) = S(I) - 1. - 0.178*DEL(I)
   ELSEIF (PCBST(D(I),DEL(I)-1),GT.0.0.) THEN
      S(I) = S(I) - 0.178*DEL(I)
      D(I) = S(I) - 1. + 0.178*DEL(I)
   ENDIF
500 CONTINUE

C

DO 300 I=1,N
   D(I) = S(I) * M(I)
    S(I) = D(I) * M(I)
300 CONTINUE

C

END

FUNCTION PCHST(ARG1, ARG2)

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

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

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
   CONTINUE
   CONTINUE
11  CONTINUE
ENDIF
RETURN
END
**SUBROUTINE CURTER**

**SOURCE PROGRAM**

**samgrid.f**

```fortran
**SUBROUTINE CURTER**
**SOURCE PROGRAM**
**samgrid.f**

**DIMENSION** TWF(NPI),WT(NPI)

**LJ** = L - 1

**NFUS** = EDIM - NPTS

**NBOT** = NFUS/2 + 1

**NTOP** = NFUS/2 + 1

**C**

**DO** 900 I = 1, L3

**C**

**Note:** I am leaving the nose and the wake alone

**IF** (AMP(Y(I),NBOT) - Y(I,EDIM-NTOP+1)) LE 1.E-7 **THEN**

**RETURN**

**ENDIF**

**C**

**DO** I = 1, NBOT

**TINF(I) = S(I,LJ)**

**ZINF(I) = Y(I,LJ)**

10 **CONTINUE**

**FSUP = SQRT((S(I,NBOT)-S(I,NBOT+1))**2 +

**2 * (Y(I,NBOT)-Y(I,NBOT+1))**2 )

**CALL DISTANC(YINF,ZINF,NBOT,THK,LEN,NBOT,FSUP,1)**

**DO** 10 I = 1, NBOT

**S(I,LJ) = YINF(I)**

**Z(I,LJ) = ZINF(I)**

10 **CONTINUE**

80 **CONTINUE**

**C**

**DO** 80 I = 1, NTOP

**TINF(I) = Y(I,EDIM-NTOP)**

**ZINF(I) = S(I,EDIM-NTOP)**

100 **CONTINUE**

**W1 = (EDIM-NTOP)**

**W2 = (EDIM-NTOP)**

**FSUP = SQRT((2*W1)-S(I,W2))**2 +

**2 * (Y(I,W1)-Y(I,W2))**2 )

**CALL DISTANC(YINF,ZINF,NBOT,THK,LEN,W1,NBOT,FSUP,0)**

**DO** 80 I = 1, NTOP

**S(I,EDIM-NTOP) = YINF(I)**

**Z(I,EDIM-NTOP) = ZINF(I)**

80 **CONTINUE**

900 **CONTINUE**

**RETURN**

**END**
```
SUBROUTINE DISTARC(X,Y,N,XNEW,YNEW,FCS,IPLAT)
C
DIMENSION X(N),Y(N),XNEW(NNEW),YNEW(NNEW)
C
This program redistribute the points (X,Y) by subroutine DISTRI
based on the arc length. FCS is the first grid spacing. Note that
the end points of the two sets are the same.
C
IFPLAT=0, grid points will cluster near the first point, -1 near the end.
C
Input array is (X(I),Y(I)), I=1,...,N.
C
Output array is (XNEW(I),YNEW(I)), I=1,...,NNEW.
C
PARAMETER (MAX=400)
DIMENSION S(MAX),TOTARC(MAX),EN(MAX),TH(MAX)
C
Maximum number of points allowed is MAX
IF(MAX LE N OR MAX LE NNEW) THEN
WRITE(6,*) 'SUB DISTARC MAX is less than N or NNEW'
STOP
ENDIF
C
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
C
Apply subroutine DISTRI to obtain the stretching function S
Here, equal spacing is used
IF(NCS GT 0.) THEN
S = (S(I-1)+1.) /FLOAT(NNEW-1)
ELSE
S(I) = 0.
DO 25 I = 2,NNEW
S(I) = S(I-1) + 1. /FLOAT(NNEW-1)
25 CONTINUE
ENDIF
C
Redistribute, put new array in a temporary array XN and YN
XN(1) = XI(1)
YN(1) = YI(1)
XN(NNEW) = XI(N)
YN(NNEW) = YI(N)
C
DO 60 J = 2,NNEW
ARCNEW = S(J) /TOTARC(N)
DO 55 K = 2,N
IF(ABS(TOTARC(K)-ARCNEW) LE 1.E-7) THEN
XN(J) = X(K)
YN(J) = Y(K)
GOTO 60
ENDIF
55 CONTINUE
IF(TOTARC(K) GT ARCNEW) THEN
XI = X(K-1)
X2 = X(K)
Y1 = Y(K-1)
Y2 = Y(K)
XX = XI + (X(K)-X(K-1)) * (ARCNEW-TOTARC(K-1)) /TOTARC(K-1)
NL = 1
CALL LINTER(XI,X2,Y1,Y2,XX,YY)
ELSE
NL = 2
TH(NL) = TH(NL-1) + (ARCNEW-TOTARC(K-1))
ENDIF
56 CONTINUE
60 CONTINUE
C
Write the temporary arrays into the output XNEW, YNEW
DO 70 J = 1,NNEW
XNEW(J) = XN(J)
YN(J) = YN(J)
70 CONTINUE
RETURN
END
**SOURCE PROGRAM**

**samgrid.f**

---

**LINE #**

**SOURCE TEXT**

669 SUBROUTINE DISTRI(FANG, KPCS, S, IFINE)
670 PARAMETER (MAX=400)
671 DIMENSION S(MAX), DOM(MAX)
672 C calculating the stretching function & when given
673 C the first spacing, FANG, and the number of points KPCS
674 C if IFINE=1, distribution is culminating at outer grid
675 C
676 IF(MAX.LE.KPCS) THEN
677 WRITE(*,*)'SUB-DISTR : MAX is less than KPCS'
678 STOP
679 ENDIF
680 IF(KPCS.EQ.1) THEN
681 S(1) = 0.
682 GOTO 40
683 ENDIF
684 DII = FANG
685 IFM = IFCS-1
686 DIZETA = 1./FLOAT(KFCS)
687 CALL GBET(DII, IFM, 0.0001, 100, RDBETA)
688 CALL FZ1(KFCS, RDBETA, DIZETA, S)
689 DO 37 K=1,IFCS
690 S(K) = 1.-DUM(K)
691 CONTINUE
692 DO 38 K=1,IFCS
693 S(K) = 1.-DUM(K)
694 CONTINUE
695 ENDIF
696 CONTINUE
697 RETURN
698 END
SUBROUTINE ED_E(NC,NU,NL,XL,XBK,IBASE,YBASE,ZBASE_NPK,LS)

DIMENSION ZBASE(NPK,2,LS),XBASE(NP_,2,LS),YBASE(NPK,2,LS)

IZLE - ZBASE(I,1,1)

DO 200 K = 1,NC

IF(ZBASE(1,1,K).GT.XBK) THEN

I1 = ZBASE(1,1,K-1)
I2 = ZBASE(1,1,K)
I3 = ZBASE(1,1,K+1)
I4 = ZBASE(1,1,K+2)

CALL LININT(I1,I2,I3,I4,XBK,ZBK)
GOTO 210

ENDIF

200 CONTINUE

210 CONTINUE

IF(ZBASE(1,1,E).G_.ZB[E]) GO TO 700

CALL LININT(Z_E,XBK,XL,:XB_,_II,

XL_OLD = X_SE(I,E,E)
XTL = XBASE(NL,E,E)
DO 280 I = 1,NL

F = XBASE(I,E,E)-XL_OLD
E = XTL-XBASE(I,E,E)

XBASE(I,E,E) = (F*XTL + E*XL_OLD)/(F+E)
280 CONTINUE

XTL = XBASE(NU,1,E)
DO 300 I = 1,NE

f = XB(I,E,E)-XL_OLD

E = XTL-XBASE(I,1,E)

300 CONTINUE

RETURN

END
SUBROUTINE EQSPACE

include "agrid.com"

L2 = L-1
DO 130 L1=1,L2
  XIN(L1) = X(L1)
  DO 120 K=1,EDIM
    YIN(L1,K) = Y(L1,K)
  120 CONTINUE
  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,KDIM
    IF(ABS(XIN(KL)-X(JL)) .LE. 1.E-7) THEN
      DO 140 ,KDIM
        Z(KL,K) = ZIN(JL,K)
        Y(KL,K) = YIN(JL,K)
      140 CONTINUE
      GOTO 150
    ENDIF
    IF(XIN(KL) .GT. X(JL)) THEN
      DO 145 ,KDIM
        X1 = XIN(KL-1)
        Y1 = YIN(KL-1,K)
        Z1 = ZIN(JL,K)
        CALL LININT(X1,X2,Y1,Y2,XX,YY)
        Y(JL,K) = YY
        Z(JL,K) = ZZ
      145 CONTINUE
      GOTO 160
    ENDIF
  150 CONTINUE
  CONTINUE
  CONTINUE
  RETURN
END
SUBROUTINE FILET(Y,EDIM,MB1,MB2,MT1,MT2,RFIL)

PARAMETER (MB2=400)
DIMENSION Z(EDIM),Y(EDIM)
DIMENSION DI(MAX),D2(MAX)
COMMON /REP/ ZROOT,ATIP,ARCORE

C This subroutine takes a wing-fuselage section, \((Y(k),Z(k))\) \(k=1,EDIM\),
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 \((Y(k),Z(k))\), it
C extends to a segment of points, \((Y(k),Z(k))\) \(k\)=ATIP to ATIP, where
C ATIP-\(k\)-MB1, ATIP-\(k\)-MB2.
C
C Similar procedure for the top part.
C
C and then, call subroutine CIRCLE to replace the segment by a
C segment of a circle with radius RFIL.

IF(EDIM.EQ.0.) GOTO 735
IF(MAX.LE.EDIM) THEN
  WRITE(*,'(*)') 'SUBFILET : MAX is less than EDIM'
  STOP
END

C Bottom part of the aircraft
IF(MB1.EQ.0. AND MB1.EQ.0.) GOTO 135
DO 130 ATIP,EDIM
  IF (X.EQ.ATIP .AND. 1(X).GT.ZROOT) THEN
    EP1=E-MB1
    EP2=M-MB2
    CALL CIRCLE(EP1,EP2,EDIM,Y,Z,RFIL,ARCORE)
  END
C We have \(N=EP1-EP2+1\) pts in filet area, employ two more points
C from the original grid and redistribute them, the grid spacing looks
C smooth.
C
  EF1=EF1+1
  DO 220 ED-ED+1,EF1
    D1(ED-ED+1) = Y(ED)
    D2(ED-ED+1) = Z(ED)
    CONTINUE
C
  HF1=HF1+1
  CALL DISTARC(D1,D2,H,N,D1,D2,H,-10.,0)
  DO 330 KD,KFI,KF2
    Y(KD)-DI(KD-KFI+I)
    Z(KD)-D2(KD-KFI+I)
  CONTINUE
GOTO 135
END

C Top part of the aircraft
IF(MT1.EQ.0. AND. MT2.EQ.0.) GOTO 735
DO 730 X-ATIP,EDIM
  IF(XX.EQ.XX .AND. L(XX).LE.ZROOT) THEN
    EF1=E-MT1
    EF2=M-MT2
    CALL CIRCLE(EF1,EF2,EDIM,Y,Z,RFIL,ARCORE)
  END
C We have \(N=EF1-EP2+1\) pts in filet area, employ two more points
C from the original grid and redistribute them, the grid spacing looks
C smooth.
C
    EY1=EY1+1
    DO 420 ED-ED+1,EY1
      D1(ED-ED+1) = Y(ED)
      D2(ED-ED+1) = Z(ED)
    CONTINUE
C
    HF1=HF1+1
    CALL DISTARC(D1,D2,H,N,D1,D2,H,-10.,0)
    DO 530 KD,KFI,KF2
      Y(KD)-DI(KD-KFI+I)
      Z(KD)-D2(KD-KFI+I)
    CONTINUE
GOTO 735
END

RETURN
END
```fortran
SUBROUTINE FZ1(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 FZ1 : 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
    BBETA=BBETA+ETA
    ETA=(ETA/TBETA-1.0)**DET
    Z(L)=-(ETA/TBETA-1.0)**(ETA/TBETA-1.0)
  10 CONTINUE
  END IF
END
```

C SUBROUTINE GETR(DPM,NPT,FCC,ICC,BETA)
C BISECTION METHOD USED TO DETERMINE STRETCHING PARAMETER, BETA,
C WHICH GIVES DESIRED CR AT THE WALL
C
C PARAMETER (MAX=400)
C DIMENSION Z(MAX)
C IF(MAX.LE.NPT) THEN
C WRITE(6,'(A)') 'SUB GETR : MAX is less than NPT'
C STOP
C ENDIF
C
C ICC-ICC
873 FPCCL=FPCC-DPM
874 BETA=BETA
875 Z1=DPM
876 DET=1.NPT
877 BB=1
878 IT=11
879 ICC=ICC/10
880 GO TO 1,11
881 BP=BETA
882 BETA=0.5*(BP+BETA)
883 CALL FZI(2,BP,DET,Z)
884 FF=Z(2)-Z1
885 IF(FF.LT.0.) GO TO 15
886 CONTINUE
887 15 CONTINUE
888 DO 5 NIT=I,ICCL
889 CALL FZI(2,BETA,DET,Z)
890 FF=F-Z1
891 IF(FF.GT.0.) THEN
892 BP=BETA
893 ELSE
894 END IF
895 BETA=0.5*(BP+BR)
896 IF(ABS(F).LT.FPCC) GO TO 4
897 5 CONTINUE
898 WRITE(6,100) BETA,F
100 FORMAT(3E10.6)
101 CONTINUE
102 DO 4 NIT=I,ICCL
103 CALL FZI(2,BETA,DET,Z)
104 FF=F-Z1
105 IF(FF.GT.0.) THEN
106 BR=BP-1.
107 4 CONTINUE
108 END
SUBROUTINE LININT(X1,X2,Y1,Y2,YLOCAL,YLOCAL)
C     This subroutine linearly interpolate YLOCAL when given (X1,Y1) & (X2,Y2)
IF(XABS(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,JNUM,IPRN)

DIMENSION NPAIR(2,NPI),LNUM(4)

DIMENSION XENG(NPI),YENG(NPI,1),ZENG(NPI,1),XAC(NPI)

COMMON /XENG/,YENG/ZENG/,ZENG/,XAC/,YAC/,ZAC/2

C (XENG,YENG,ZENG) Coordinate of engine

C NAC(*) Number of stations in nacelle

C NACP(*) Number of points in each station

C (1,1) Inner nacelle, (2,1) outer nacelle

C MC = # of pts added in the grid (=* pts at nacelle)

IPRN = 0  no writing out

MC = 40

NPI = NPTS+MC

IF(LNUM(1).EQ.LNUM(3) .AND. LNUM(2).EQ.LNUM(4)) THEN

NPTS = NPTS + MC

ENDIF

NPI = (NPTS+1)/2

NPAIR(1,JN) = NAC(JN)

NAC = NAC(JN)

GOTO 105

C If(IPRNT.NE.0) WRITE(IPRTN)IPRN,JNUM,LNUM(1)-LNUM(1)+1

C Now the big job!

C Interpolate the points of the nacelle at XOUT

DO 100 JN=1,NAC

IFABS(XENG(JN,JN)-XOUT(L,1)).LE.1.E-7 THEN

DO K=1,NACP

YAC(K) = YENG(JN,K)

ZAC(K) = ZENG(JN,K)

ENDIF

GOTO 105

ELSEIF(XENG(JN,JN).GT.XOUT(L,K)) THEN

DO K=1,NACP

X1 = XENG(JN,K)

Y1 = YENG(JN,K)

Z1 = ZENG(JN,K)

ENDIF

CALL LININT(X1,Y1,Z1,KOUT)

ENDIF

CONTINUE

C Count the number of points in the wake if we are in the wake

KOUNT = 0

DO 120 K=1,NPB

E1=NPB-K

E2=NPB+K

IF(ABS(YOUT(L,E1)-YOUT(L,E2)).LE.1.E-7) THEN

ABS(SORT(E1,E2)) .GT. L.E-7 THEN

KOUNT = KOUNT + 1

NPAIR(1,KOUNT) = E1

NPAIR(2,KOUNT) = E2

ENDIF

120 CONTINUE

ENDIF

100 CONTINUE

RADNAC = SQRT((ZAC[1]-ZAC(NACP/2)**2) +
(AC[1]-ZAC(NACP/2)**2))

C Nacelle totally under the wing, INTRAIL = 0

C Nacelle over the wing, part in the wake, INTRAIL = 1

C Nacelle totally in the wake, INTRAIL = 2

IF(KOUNT.EQ.0) GOTO 415

IF(SORT(L,NPAIR(1,1))).GT.ZAC(NACP/2) INTRAIL=1

IF(SORT(L,NPAIR(1,1))).GT.ZAC(NACP/2) INTRAIL=2

WRITE(10,'(A)') 'INTRAIL=', INTRAIL

IF(INTRAIL.NE.2) THEN

GOTO 415

ELSE

WRITE(10,'(A)') 'We are in the wake region of the trailing edge

GOTO 415

ENDIF

130 CONTINUE

C Obtain all points under the wake line

DO 180 E1=1,KOUNT

IF(SORT(L,NPAIR(1,E1))).LT.ZAC(NACP)) THEN

KS = E1

r1 = abs(zout(L,NPAIR(1,E1))-zout(L,NPAIR(1,E1)+1))

r2 = abs(zout(L,NPAIR(1,E1)+1)-zout(L,NPAIR(1,E1)+2))

IF(r1 .LE. r2) ZS = E1

IF(zout(L,NPAIR(1,E1)).LT.ZAC(NACP)) THEN

NTEMP = NTEMP + 1

GOTO 130

ENDIF

180 CONTINUE

NS = NTEMP

IF(NS.LT.1) THEN

CALL LININT(YAC(NS-1),YAC(NS),ZAC(NS-1),ZAC(NS)

Y2 = YOUT(L,NPAIR(1,KS))

Z2 = ZAC(NS)

ENDIF

190 CONTINUE

NTEMP = NACP
CONTINUE DO 300 = 1,NPAIR(1,KE)
   YNG(K) = YOUT(L,K)
   ZNG(K) = ZOUT(L,K)
ENDO

320 CONTINUE
CONTINUE
IF(NTEMP .NE. NAC) THEN
   Y = YOUT(L,NPAIR(1,KE))
   YNE = YOUT(L,NPAIR(1,KE)+1)
ELSE CALL LIMIT(YOUT(L,NE-1),YNE,ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
   ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)+1),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

330 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

350 CONTINUE
CONTINUE
IF(YOUT(L,NPAIR(I,KE)) .GT. ZZE) THEN
   ZZE = YOUT(L,NPAIR(I,KE))
ENDIF

360 CONTINUE
CONTINUE
IF(YOUT(L,NPAIR(I,KE)) .LT. ZZE) THEN
   ZZE = YOUT(L,NPAIR(I,KE))
ENDIF

370 CONTINUE
CONTINUE
DO 200 = 1,NPAIR(1,KE)
   YNG(K) = YOUT(L,K)
   ZNG(K) = ZOUT(L,K)
ENDDO

200 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

210 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

230 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

250 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

270 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

290 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

310 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

330 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF

350 CONTINUE
CONTINUE
NE = NTEMP
IF(INTEMP .NE. NNACH) THEN
   CALL LIMIT(YOUT(L,NE-1),YNAC(NE-1),ZNAC(NE-1),ZNAC(NE),ZNE)
   YNE = YOUT(L,NPAIR(1,KE))
ELSE CALL LIMIT(YOUT(L,NPAIR(1,KE)),YNE,YOUT(L,NE-1),YOUT(L,NE),ZNE)
   YNE = ZNE
ENDIF
ENDIF
C Store wingroot grid
DO 500 X=1,ES
  YNAC(X) = YOUT(L,E)
  YNAC(X) = YOUT(L,E)
  CONTINUE
500
C Get the nacelle grid under the wake line ready
N2 = NTRIE-NFRI=+1 -2
YN(1) = YYS
YN(1) = ZYY
YN(2) = YVY
YN(2) = ZXY
DO 520 X=NFRI,NFRI
  YN(1) = YVY
  YN(1) = ZXY
  CONTINUE
520
C Place the nacelle grid above the wing
DO 520 X=1,NS-1
  YN(1) = YYY
  YN(1) = ZYY
  CONTINUE
520
C Put back the wingroot grid into the wing
KEEP = NFPI-NPAIR(K,E)+1
DO X=1,KEEP
  YN=NAC(E)+1 = YINT(E)
  YN=NAC(E)+1 = YINT(E)
  CONTINUE
ENDIF
DO 600 I=NP, NPTS
  IF(DOUT(L,E), LE, ZNAC(NFRIS)) THEN
  NN2 = 1
  CALL LININT(ZOUT(L,N2), ZOUT(L,N2-1), YOUT(L,N2),
  YOUT(L,N2-1), ZNAC(NFRIS), TT2)
  I = I + 1
  312 = ZNAC(NFRIS)
  GOTO 605
ENDIF
600 CONTINUE

DO 610 I=NP, N1
  ING(NPBN-K-NP) = YOUT(L,E)
  3N2 = ZNAC(NFRIS)
  GOTO &05
ENDIF
600 CONTINUE

C Redistribute the points above the escallo
  H2 = N2-NN1+1
  SW(1) = YT1
  SW(N) = YTT
  SW(N) = YTT
  DO X=NN1, N2-1
    INE(K) = YOUT(L,E)
    ZWE(K) = ZOUT(L,E)
  ENDDO

C Store the wingtip grid
DO K=1, NNPW
  YOUT(L,E) = XOUT(L,E)
  ZOUT(L,E) = YOUT(L,E)
  ZOUT(L,E) = YOUT(L,E)
ENDDO

700 CONTINUE

C Store the wingtip grid
DO K=NP2, NPTS
  ING(NPT+NNK+1) = YOUT(L,E)
  ING(NPT+NNK+1) = ZOUT(L,E)
ENDDO

700 CONTINUE

IF(IPRNT .NE. 0) THEN
  WRITE (IPRNT) (XOUT(L,E), K=1,NP),
  (YOUT(K), K=1, NNP),
  (ZOUT(K), K=1, NNP)
  CALL FLUSH(IPRNT)
ENDIF

DO 750 K=1, NPPW
  XOUT(L,E) = XOUT(L,E)
  YOUT(L,E) = YOUT(K)
  ZOUT(L,E) = ZOUT(K)
ENDDO

750 CONTINUE

C
800 CONTINUE

IF(LNUM(1), EQ., LNUM(1), .AND. LNUM(2). EQ., LNUM(4)) THEN
  NPTS = NPTS - NC
ENDIF

END
**SOURCE PROGRAM**

```fortran
SUBROUTINE MACGRID
   include "sgrid.com"
   COMMON /ENG/, XENG(2,NPI), YENG(2,NPI,NPI), ZENG(2,NPI,NPI)
   COMMON /HAC/, XHAC(NAC), YHAC(NAC), ZHAC(NAC,4)
   COMMON /NNACP/, NNAC, HNACP, NNAC3, XNNACP, YNNACP, ZNNACP

120 C (XENG, YENG, ZENG) Coordinates of engine
121 C NNAC(*) Number of stations in nacelle
122 C HNACP(*) Number of points in each station
123 C (1,*) inner nacelle, (2,*) outer nacelle
124 C
125 C Read the nacelles geometry:
126 C
127 C Inner nacelle geometry
128 CALL MACIN
129 XIN = NNAC1
130 XIN2 = XHAC(NNAC)
131 NNAC(1) = NNAC
132 NNAC(2) = NNAC
133 XENG(1,1) = XHAC(L)
134 DO 100 L = 1, NNAC
135 XENG(1,1) = XHAC(L)
136 XENG(1,1) = XHAC(L)
137 CT CONTINUE
138 C
139 C Outer nacelle geometry
140 CALL MACIN
141 XOUT1 = XHAC(NNAC)
142 XOUT2 = XHAC(NNAC)
143 NNAC(2) = NNAC
144 NNAC(2) = NNAC
145 XENG(1,1) = XHAC(L)
146 DO 200 L = 1, NNAC
147 XENG(1,1) = XHAC(L)
148 XENG(1,1) = XHAC(L)
149 CT CONTINUE
150 C
151 C In a case of 3 nacelles, three zones will be made.
152 C HNENG : Station(s) will be added to the wing at inlet and outlet
153 C of the nacelle
154 C MOUNT : Mount the nacelle under the wing and/or wake line
155 JN = 0
156 JN = 1
157 JN = 2
158 C The first zone, only one nacelle appears
159 WRITE(*,'(a)') 'zone 1'
160 JN = 1
161 CALL MOUNT(JN, LNUM(1), XOUT1)
162 CALL MOUNT(JN, LNUM(0), 0)
163 LNUM(3) = LNUM(1)
164 LNUM(4) = LNUM(2)
165 C
166 C The second zone consists two nacelles appear
167 WRITE(*,'(a)') 'zone 2'
168 JN = 1
169 CALL MOUNT(JN, LNUM(1), XOUT1, XOUT2)
170 LNUM(3) = LNUM(1) + 1
171 CALL MOUNT(JN, LNUM(0), 0)
172 LNUM(4) = LNUM(2)
173 C
174 C The third zone, only one nacelle appears
175 WRITE(*,'(a)') 'zone 3'
176 JN = 1
177 CALL MOUNT(JN, LNUM(1), XOUT1, XOUT2)
178 LNUM(3) = LNUM(1) + 1
179 CALL MOUNT(JN, LNUM(0), 0)
180 C
181 C RETURN
182 END
```
SUBROUTINE MEWING(LINUM,XX,XX)
  include "agrld.com"
  DIMENSION XWNG(NPI,NPI),YWNG(NPI,NPI),ZWNG(NPI,NPI)
  DIMENSION LNUM(4)

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

C Find out where the x-location of start and end of nsecell
LSTART = XX
END = XX

C Add two stations in the wing, these two stations lie exactly on
C INSTRT and ENNEND.
DO 55 NN=1,2
  IF(NN.EQ.1) X=INSTRT
  IF(NN.EQ.2) X=ENNEND
  GO TO 55

C The wing-section is very close to nsecell's station (INSTRT or ENNEND)
IF(ABS(XWNG(LN,1)-XX),LL,1,K-7) THEN
  DO 1=1,NPTS
    XWNG(LN,1) = XX
  END
  LNUM(NN) = LN
  GO TO 55

C Create an extra station in the wing
IF(XWNG(LN,1),LT,XX) THEN
  X(LW) = XX
  LNUM(NN) = LN
  DO 24 X=1,NPTS
    IF(X.GT.XWNG(LN,1)) GO TO 24
    XWNG(LN,1) = XX
  24 CONTINUE

C If we are in wake, make sure top pts intersect the bottom pts
DO 24 X=1-NPTS/2+1,NPTS
    IF(ABS(Y(LW,1)-Y(LN,1)) .LT. 1.E-6 .AND.
        .AND.
        ABS(Z(LW,1)-Z(LN,1)) .LT. 1.E-6) THEN
      Y(LW,1) = Y(LN,1)
      Z(LW,1) = Z(LN,1)
    END
  24 CONTINUE

C Put the rest of the station into (X,Y,Z)
DO 35 L=1-NSEC
  DO 35 K=1-NPTS
    X(L) = XWNG(L,NPI)
    Y(L) = YWNG(L,NPI)
    Z(L) = ZWNG(L,NPI)
  35 CONTINUE

C HSEC = NSEC - 1
DO 45 L=1-HSEC
  DO 45 K=1-NPTS
    X(L) = XWNG(L,NPI)
    Y(L) = YWNG(L,NPI)
    Z(L) = ZWNG(L,NPI)
  45 CONTINUE

C Continue
GO TO 55

END
**SOURCE PROGRAM**

```fortran
include "sgrid.com"

* SOURCE TEXT *

**DIMENSION D1(NP1),D2(NP1),S(NP1)**

**Ktip** = **(Kdim+1)/2**

**From the nose to the leading edge**

DO 12 **I=1,Kdim**

**Xi(I)** = **Xin(I)**

**Y(I)** = **Yin(I,1)**

**Z(I)** = **Zin(I,1)**

12 **CONTINUE**

**Loop for all stations, from station 1 to station N**

DO 50 **I=M+2,N**

**L = I**

**Xlocal** = **Xin(I)**

Store the input to dummy array.

DO 31 **I=1,NP**

**Yin(I)** = **Tin(M,I)**

**Zin(I)** = **Zin(M,I)**

31 **CONTINUE**

**Tref** is value of the first point of the wing.

**Tref** = **Tout(1,NP)**

**Options**:

Boeing Baseline Configuration

**DO 41 I=1,NP**

**If(Tint(I).GE.Yref)THEN**

**Yref=I**

**Goto 44**

**Endif**

**CONTINUE**

44 **CONTINUE**

**Kref=NFP**

**L=Kref**

**Xref=31**

**Options:**

Lower part of the nose (-Y to Yref)

**Ks=1**

**Ee=Eref**

**Ks=Ee-1**

**Do 74 I=Ks,Ee**

**Ee=Ee-1**

**D1(EE)** = **Yint(EE)**

**D2(EE)** = **Zint(EE)**

74 **CONTINUE**

**Call DistArc(D1,D2,EX,Ynew,EDist,Etip,Fnew,1)**

**Do 85 I=1,Etip**

**Y1(L,E)** = **Ynew(E)**

**Z1(L,E)** = **EDist(E)**

**X1(L)** = **Xlocal**

85 **CONTINUE**

**From Yref to pos Y**

**Ks=Yref**

**Ee=NFP**

**Ks=Ee-1**

**Do 100 I=Ks,Ee**

**Ee=Ee-1**

**D1(EE)** = **Yint(EE)**

**D2(EE)** = **Zint(EE)**

100 **CONTINUE**

**Call DistArc(D1,D2,EX,Ynew,EDist,Etip,Fnew,0)**

**Do 400 I=Etip,Kdim**

**Y1(L,E)** = **Ynew(E- Etip+1)**

**Z1(L,E)** = **EDist(E- Etip+1)**

**X1(L)** = **Xlocal**

400 **CONTINUE**

**RETURN**

**END**
```
**Samgrid.f**

**Type:** Source Text

**Dimension:** 610.0 x 792.0

**Page Number:** 24

**Source Program:**

```fortran
544 SUBROUTINE RESIST(XLOCAL, ETIP, FAC, IFLAT)
547 include "agrid.com"
562 PARAMETER(NINT=400)
569 DO 50 I-2,NUL
570 IF(XBASE(I,IFLAT,NC-M+I).GE.ILOCAL .OR.
571 A_S(IBASE(I,IFLAT,NC-M+I)-XI_3_AL).LE.1.E-7) T_I
572 XI " XBASE(I-I,IFLAT,NC-M+I)
573 X2 " X_BASE(I,IFLAT,NC-M+I)
574 Y1 " YBASE(I-I,IFLAT,NC-M+I)
575 Z1 - ZBASE(I-I,IFLAT,NC-M+I)
576 Z2 - ZBASE(I-1,IFLAT,NC-M+I)
577 YINT(M) - TT
578 ZZNT(_7) " ZZ
579 GOI_200
580 CONTINUE
583 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
584 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
585 YINT(KT) = YY
586 ZINT(KT) = ZZ
587 GO_200
589 DO 150 I = 2,NC
590 XL = XBASE(I,IFLAT,M-1)
591 XR = XBASE(I,IFLAT,M)
592 IF(XL*XR.LT.0.) THEN
593 XI = XBASE(I,IFLAT,M-1)
594 X2 = XBASE(I,IFLAT,M)
595 Y1 = YBASE(I,IFLAT,M-1)
596 Z1 = ZBASE(I,IFLAT,M-1)
597 Y2 = YBASE(I,IFLAT,M)
598 Z2 = ZBASE(I,IFLAT,M)
599 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
600 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
601 YINT(KT) = YY
602 ZINT(KT) = ZZ
603 ELSEIF (ABS(XL*XR).LT.1.E-7) THEN
604 XI = XI_3_AL
605 X2 = XLOCAL
606 Y1 = Y_LOCAL
607 Z1 - Y_BASE(I,IFLAT,M)
608 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
609 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
610 YINT(KT) = YY
611 ZINT(KT) = ZZ
612 GOO 200
614 ENDIF
617 ET = 0
620 DO 150 I = 1,NUL
623 ET = ET + 1
625 IF(XBASE(I,IFLAT,NC-M+I).GT.XLOCAL) THEN
626 XI = XBASE(I,IFLAT,NC-M+I)
627 X2 = XBASE(I,IFLAT,NC-M+I)
628 V1 = VBASE(I,IFLAT,NC-M+I)
629 V2 = VBASE(I,IFLAT,NC-M+I)
630 Z1 = ZBASE(I,IFLAT,NC-M+I)
631 Z2 = ZBASE(I,IFLAT,NC-M+I)
632 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
633 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
634 YINT(KT) = YY
635 ZINT(KT) = ZZ
636 GOO 200
639 ENDIF
642 DO 150 M = 2,NC
645 XL = XLOCAL-XBASE(I,IFLAT,NC-M+I)
646 XR = XLOCAL-XBASE(I,IFLAT,NC-M+I)
647 IF(XL.LT.0.) THEN
648 XI = XBASE(I,IFLAT,NC-M+I)
649 X2 = XBASE(I,IFLAT,NC-M+I)
650 Y1 = YBASE(I,IFLAT,NC-M+I)
651 Z1 = ZBASE(I,IFLAT,NC-M+I)
652 Y2 = YBASE(I,IFLAT,NC-M+I)
653 Z2 = ZBASE(I,IFLAT,NC-M+I)
654 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
655 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
656 YINT(KT) = YY
657 ZINT(KT) = ZZ
658 GOO 160
662 ELSEIF (ABS(XL*XR).LT.1.E-7) THEN
665 IF(X(XL) .LE. 1.E-7) THEN
```

**Source Program:**

```fortran
544 SUBROUTINE RESIST(XLOCAL, ETIP, FAC, IFLAT)
547 include "agrid.com"
562 PARAMETER(NINT=400)
569 DO 50 I-2,NUL
570 IF(XBASE(I,IFLAT,NC-M+I).GE.ILOCAL .OR.
571 A_S(IBASE(I,IFLAT,NC-M+I)-XI_3_AL).LE.1.E-7) T_I
572 XI = XBASE(I-I,IFLAT,NC-M+I)
573 X2 = X_BASE(I,IFLAT,NC-M+I)
574 Y1 = YBASE(I-I,IFLAT,NC-M+I)
575 Z1 = ZBASE(I-I,IFLAT,NC-M+I)
576 Z2 = ZBASE(I-1,IFLAT,NC-M+I)
577 YINT(M) = TT
578 ZZNT(_7) = ZZ
579 GOI_200
580 CONTINUE
583 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
584 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
585 YINT(KT) = YY
586 ZINT(KT) = ZZ
587 GOO 200
589 DO 150 I = 2,NC
590 XL = XBASE(I,IFLAT,M-1)
591 XR = XBASE(I,IFLAT,M)
592 IF(XL*XR.LT.0.) THEN
593 XI = XBASE(I,IFLAT,M-1)
594 X2 = XBASE(I,IFLAT,M)
595 Y1 = YBASE(I,IFLAT,M-1)
596 Z1 = ZBASE(I,IFLAT,M-1)
597 Y2 = YBASE(I,IFLAT,M)
598 Z2 = ZBASE(I,IFLAT,M)
599 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
600 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
601 YINT(KT) = YY
602 ZINT(KT) = ZZ
603 ELSEIF (ABS(XL*XR).LT.1.E-7) THEN
604 XI = XI_3_AL
605 X2 = XLOCAL
606 Y1 = Y_LOCAL
607 Z1 = Y_BASE(I,IFLAT,M)
608 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
609 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
610 YINT(KT) = YY
611 ZINT(KT) = ZZ
612 GOO 200
614 ENDIF
617 ET = 0
620 DO 150 I = 1,NUL
623 ET = ET + 1
625 IF(XBASE(I,IFLAT,NC-M+I).GT.XLOCAL) THEN
626 XI = XBASE(I,IFLAT,NC-M+I)
627 X2 = XBASE(I,IFLAT,NC-M+I)
628 V1 = VBASE(I,IFLAT,NC-M+I)
629 V2 = VBASE(I,IFLAT,NC-M+I)
630 Z1 = ZBASE(I,IFLAT,NC-M+I)
631 Z2 = ZBASE(I,IFLAT,NC-M+I)
632 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
633 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
634 YINT(KT) = YY
635 ZINT(KT) = ZZ
636 GOO 200
639 ENDIF
642 DO 150 M = 2,NC
645 XL = XLOCAL-XBASE(I,IFLAT,NC-M+I)
646 XR = XLOCAL-XBASE(I,IFLAT,NC-M+I)
647 IF(XL.LT.0.) THEN
648 XI = XBASE(I,IFLAT,NC-M+I)
649 X2 = XBASE(I,IFLAT,NC-M+I)
650 Y1 = YBASE(I,IFLAT,NC-M+I)
651 Z1 = ZBASE(I,IFLAT,NC-M+I)
652 Y2 = YBASE(I,IFLAT,NC-M+I)
653 Z2 = ZBASE(I,IFLAT,NC-M+I)
654 CALL LININT(X1, X2, Y1, Y2, XLOCAL, YY)
655 CALL LININT(X1, X2, Z1, Z2, XLOCAL, ZZ)
656 YINT(KT) = YY
657 ZINT(KT) = ZZ
658 GOO 160
662 ELSEIF (ABS(XL*XR).LT.1.E-7) THEN
665 IF(X(XL) .LE. 1.E-7) THEN
```
YINT(KT) = YBASE(1,IFLAT,M-1)
ELSE
YINT(KT) = YBASE(1,IFLAT,M)
ENDIF
GOTO 160
ENDIF
CONTINUE
1672 CONTINUE

Right now YINT and ZINT is from wing tip to the root. In order to
the cubic spline program need from root to tip.

DO 220 K=1,KT
Z(K) = YINT(K)
220 CONTINUE
DO 230 K=1,KT
YINT(K) = Z(K)
230 CONTINUE
DO 240 K=1,KT
Z(K) = ZINT(K)
240 CONTINUE
DO 250 K=1,KT
ZINT(K) = Z(K)
250 CONTINUE

OPTIONS:
Distribute coordinates (repeet around direction) from root to
leading edge and back (dist).
IT>1, grid points will cluster near the wing tip, -8 near the root.
For nseb, IT=-1, for bowing, IT=1
CALL DISTARC(YINT, YINT, ET, ZDIST, YNEW, KT, FAT, IT)
IF(LT. NE. NSEC)CALL CSPLINE(ZINT, YINT, ET, ZDIST, YNEW, ETIP)
RETURN
END
subroutine subtractgrid(npl1, x, y, z, nsec, kdim)

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

dimension x(npl1, y(npl1, nsec), z(npl1, nsec)

removery the late stations

nssec = nsec - 1

do 10 l = 1, npl1, nsec

y(l, t) = y(l+1, t)

z(l, t) = z(l+1, t)

10 continue

continue

return

end
SUBROUTINE TAIL(FAC1,FAC2)
include "egrid.com"
DIMENSION S(NPI)
COMMON REF,ROOT,KTIP,ARCOHER

XIP = (EDIM-1)/2
LI = L-1

The tail of the configuration

XAPEPT is the Y value that the wake is (in Y value of ROOT)

(To get the X, use XAPEPT = XAPEPT / YAPEPT)

\[ XAPEPT = \frac{\text{YAPEPT}}{\text{YAPEPT}} \]

DO 40 L = 1,L2
J = M2+1-LI
JI = M2+1-L2
I = (J+L2)/2
II = (J-L2)/2

(REDIST, NPTS)

DO 230 K = 1,KDIM
EC = I,KDIM
(EK+K)/2
KF = (NPTS-I)/2
(KE+K)/2

DO 223 L = 1,L2
E = X2-L2
EI = ZDIST(E)
E2 = ZDIST(E2)

DO 222 K = 1,KDIM
EK = I,KDIM
(EK+K)/2
KF = (NPTS-I)/2
(KE+K)/2

DO 210 L = 1,L2
E = X2-L2
EI = ZDIST(E)
E2 = ZDIST(E2)

ENDF

DO 230 K = 1,KDIM
EC = I,KDIM
(EK+K)/2
KF = (NPTS-I)/2
(KE+K)/2

DO 223 L = 1,L2
E = X2-L2
EI = ZDIST(E)
E2 = ZDIST(E2)

ENDF

Find X2 the point at old grid where YIN(M,EF) = XAPEPT

and calculate the points from seg Y to Y at E2

DO 222 E = 1,NHF
IF(YIN(M,EF) = XAPEPT, 1,EDIM-1,E) THEN
E = XAPEPT
GOTO 223
ENDIF

IF(YIN(M,EF) = XAPEPT, 1,EDIM-1,E) THEN
E = XAPEPT
GOTO 223
ENDIF

Find X2 the point at old grid where YIN(M,EF) = XAPEPT

and calculate the points from seg Y to Y at E2
1844  DO 290 K=1,EDIM
1845   Y(L,E)=TINT(K)
1846   Z(L,E)=ZINT(K)
1847 290  CONTINUE
1848
1849  C continues...

1850  DO 395 E=1,NPE
1851   YINT(E)=VINT(E,1)
1852   ZINT(E)=ZINT(E,1)
1853 395  CONTINUE

1854  CALL DISTAN(YINT,ZINT,NPE,THEN,EDIST,ETIP,0.05,1)
1855  DO 395 E=1,NPE
1856   2(L,E)=EDIST(E)
1857   Y(L,E)=THEN(E)
1858   395  CONTINUE

1859  DO 395 E=1,NPE
1860   4(L,E)=VINT(E,NPE+1-1)
1861   ZINT(E)=ZINT(E,NPE+1-1)
1862 395  CONTINUE

1863  CALL DISTAN(YINT,ZINT,NPE,THEN,EDIST,ETIP,0.05,0)
1864  DO 395 E=1,NPE
1865   5(L,E)=EDIST(E)
1866   Y(L,E)=THEN(E)
1867 395  CONTINUE

1868  C continues...

1870  C This allows the tail has equal thickness

1871  DO 313 E=1,EDIM
1872   6(L,E)=T(L-1,E)
1873   7(L,E)=T(L+1,E)
1874 313  CONTINUE

1875  RETURN
1876
1877  END
**SOURCE PROGRAM**

```
**SOURCE TEXT**

1880 SUBROUTINE WGRID
1881 INCLUDE 'GRID.COM'
1882 C This subroutine read the fuselage grid and combine it with the
1883 C wing grid.
1884 C
1885 DIMENSION D1(NP1),D2(NP1),L(NP1)
1886 COMMON /GRID/ D1,E2,EX,AR,ASS
1887 C Read surfgrid dimensions
1888 C
1889 XI,NI,NZ = read in grid
1890 X,1 = output grid
1891 Y1,Z1,BL,DI = dummy vectors
1892 EDIM = # of points of the whole body in the warp-around direction
1893 ESDIM = # of points in the nose bottom part (circumferential direction)
1894 FWDIM = # of points in the nose top part (circumferential direction)
1895 FDIM = # of points in the tail part (circumferential directions)
1896 RFIL = Fillet radius
1897 ARCGR - This make fillet looks smooth
1898 BLAKE = Distance of the trailing wake in body length
1899 BL = Body length
1900 UAVRE = Number points along the trailing wake
1901 SCALE = Scale factor for the wing-body
1902 NAMELIST /GRID/ EDIM,ESDIM,FDIM,FWDIM,RFIL,ARCGR
1903 & BLAKE,SCALE,CC,CE,CE,CE
1904 /OPN/ TOTLEN,XLAST = X(L2) ÷ BL,BLWAGE
1905 & READ(40,BODY)
1906 READ(40,FILE)
1907 READ(40,OPTN)
1908 WRITE(40,OUT)
1909 READ(40,FILE)
1910 WRITE(40,OPN)
1911 WRITE(*,FILE)
1912 WRITE(*,OPN)
1913 C Read the fuselage geometry
1914 CALL FUSEX
1915 C Take this initial grid
1916 C (1) convert it into area distribution
1917 C (2) add shape function
1918 C (3) convert it back to grid
1919 CALL FUSEX
1920 BL = X(IN)
1921 ICXS = ECH=BL
1922 ETIP = (EDIM(N))/3
1923 C Find number of streamwise station. We keep the stations
1924 C of the fuselage. As to the common area of the wing and fuselage
1925 C we use the stations of the wing.
1926 C M1 is the station the nose ends
1927 C M2 is the station the wing ends
1928 M1 = 0
1929 M2 = 0
1930 DO 10 M = 1,NF
1931 IF(ABS(XIN(M) - BOUT(1,1)) LE 1.0E-7) M1 = M
1932 IF(ABS(XIN(M) - BOUT(NSEC,1)) LE 1.0E-7) M2 = M+1
1933 10 CONTINUE
1934 IF(M1.EQ.0) THEN
1935 DO 20 M = 1,NF
1936 IF(IN(M).LE.BOUT(1,1),AND. XIN(M).GT.BOUT(1,1))
1937 M = M+1
1938 20 CONTINUE
1939 ENDIF
1940 IF(M1.EQ.0) THEN
1941 DO 30 M = 1,NF
1942 IF(IN(M).LE.BOUT(NSEC,1),AND. XIN(M).GT.BOUT(NSEC,1))
1943 M = M+1
1944 30 CONTINUE
1945 ENDIF
1946 C We divide the configuration into four parts:
1947 C (1) the nose, the wing-body, the tail, and the extended wake. And
1948 C for each part we distribute points for the lower and upper part
1949 C separately.
1950 C
1951 C From the nose to the leading edge
1952 CALL NOSE(FNBOE,FNUPE)
1953 C
1954 C From the leading edge to the trailing edge (the whole wing)
1955 CALL WINGFACE
1956 CALL WING
1957 C
1958 C From the trailing edge to the end of tail
1959 CALL TAIL(-1,PARS)
1960 C
1961 C Redistribute each streamwise station, so that the grid
1962 C is flatter near the wing tip.
1963 IF(CEXT,GT.0) CALL EQSPACE
1964 C
1965 C Redistribute the streamwise stations, so that
1966 C it is equally spaced.
1967 IF(CEXT,GT.0) CALL EQSPACE
1968 C
1969 C Add the extended wake
1970 L2 = L-1
1971 XLAST = X(L2) ÷ BL. + UAVRE
1972 TOTLEN = XLAST - X(L2)
```
SOURCE PROGRAM
samgrid.f

SOURCE TEXT

We might like to scale the whole thing by a reference length

We're interested in Boeing Baseline Configuration before we write out the grid, this Boeing wing needed to be fixed for some stupid reason.

Langley's Low-Boom Configuration

Write out new surfgrid in plotid format

Write the wing-body grid (x,y,z) to (XOUT,YOUT,ZOUT)

END
SUBROUTINE WFMATCH

include "sgr.d.d.h"

DIMENSION 01 (NPI), D2 (NPI)

C: The wing-body section, itself has NSEC sections

L1 = the section index for TIM, LIN

DZMAX = 0

DZMIN = 1.E+20

LE = M1-1

DO 200 L=1, LE

LIN = LIN +1

ELOCAL = ROUT (LIN)

C: Calculate the points [VINT, ZINT] on the fuselage at ELOCAL

C: Note: assumed that in this section, each station has same number

C: of points in range around direction.

DO 10 M=1,NF

IF (IN(M) .GE. ELOCAL .OR. 

ABS (IN(M) - ELOCAL) .LE. 1.E-7) THEN

MM = M

GOTO 15

ENDIF

10 CONTINUE

10 CONTINUE

DO 15 F=1, NFP

15 CONTINUE

C: The wing is in the x-direction (upward) to make sure

DO 30 T=1, NFP

30 CONTINUE

C: The wing is not inside or outside the fuselage

ZROOT = ZINT (NFP+1)/2

DO 40 T=1, NFP

40 CONTINUE

RETURN

END
SUBROUTINE WING_BODY
include "samgrid.com"

DIMENSION DI(NPI),D2(NPI)

COMMON /NWP/ ENWP,STIP,ARCSML

The wing-body section, itself has NSEC sections
LN is the section index for YIN,LIN

LB = M+1
LE = M + NSEC - 1
DO 200 L=LB,LE
LN=LN+1, start with second wing station
XLOCAL = XOUT(L,1)

200 IF(NW=11, LE, E) THEN
NW = M
GO TO 15
ENDIF
10 CONTINUE
15 CONTINUE
DO 30 K=1, NWP
X1=XOUT(L,M-1)
X2=XOUT(L,M)
Y1=YOUT(L,M-1,K)
Y2=YOUT(L,M,K)
Z1=ZOUT(L,M-1,K)
Z2=ZOUT(L,M,K)
CALL LIMIT(X1,X2,Y1,Y2,ZOUT,L,YM,K)
ZINT(K) = Z2
YINT(K) = Y2
30 CONTINUE
CONTINUE
35 CONTINUE

There are NPTS points in the wing area, used to check out
the new mesh point is in the fuselage section
EI is the inter pt. / the fuselage & wing at bottom in old grid
E2 is the inter pt. / the fuselage & wing at top in old grid
MID is the inter pt. / the fuselage & wing at bottom in new grid
There are NT points from 1 to EI
EI=(NWT+1)/2
MID=(NWT-NPTS+1)/2
EI=0
CONTINUE
60 CONTINUE
CONTINUE
DO 70 K=1, NWP
IF(YINT(K) .GE. YOUT(L,M,NPTS) ) AND. YINT(K) .GT. YINT(E-1) ) THEN
EI = K
GO TO 62
70 CONTINUE
72 CONTINUE
C For arrow-wing type, EI needed to be relocated
IF(ABS(YOUT(L,M,1)-YOUT(L,M,NPTS)) .LE. 1.E-7 ) THEN
EI = El + ELADS
E2 = EI
YINT(EI)=YINT(EI)
ZINT(EI)=ZINT(EI)
ENDIF
80 CONTINUE

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

ES=1
EE=EE-ES+1
DO 82 K=1,EE
DI(K) = YINT(K)
D2(K) = ZINT(K)
82 CONTINUE

CALL DISTARC(DI,D2,EE,YOUT,EDIST,MID,NWK-10,1)
DO 100 K=1,EDIST
Y(K,E) = YOUT(K,E)
Z(K,E) = ZOUT(K,E)
100 CONTINUE
CONTINUE

The points of the wing section
DO 110 K=1, NPTS
Y(L,MID+K) = YOUT(L,M,E)
Z(L,MID+K) = ZOUT(L,M,E)
110 CONTINUE
CONTINUE

Calculate the points at the top
ES=2
EE=EE-ES+1
DO 150 K=1,EE
DI(K) = YINT(K)
D2(K) = ZINT(K)
150 CONTINUE
CONTINUE

CALL DISTARC(DI,D2,EE,YOUT,EDIST,MID,NWK-10,0)
DO 200 K=1,EDIST
Y(L,MID+NPTS) = YOUT(K,E)
Z(L,MID+NPTS) = ZOUT(K,E)
200 CONTINUE
CONTINUE

C For arrow-wing type, make sure wake points ok
IF(ABS(YOUT(L,M,1)-YOUT(L,M,NPTS)) .LE. 1.E-7 ) THEN
Y(L,MID+NPTS+1) = YOUT(L,M)
Y(L,MID) = Y(L,MID+NPTS+1)
2252 \( Z(L,\text{MID}) = Z(L,\text{MID+NPTS+1}) \)

ENDIF

2255 \( \) C

2256 \( \) C Fill the unsmooth part by FILET

2257 \( \) C First of all, find the set of points needed to be rearrange

DO 125 \( \) K=1, KDIM

2259 \( \) YINT(K) = Y(L, K)

2260 \( \) ZINT(K) = Z(L, K)

2261 \( \) 125 CONTINUE

RFILT - RFILT

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

2264 \( \) & CALL FILET(YINT, ZINT, KDIM, MB1, MB2, MT1, MT2, RFILT)

DO 140 \( \) K = 1, KDIM

2265 \( \) Y(L, K) = YINT(K)

2266 \( \) Z(L, K) = ZINT(K)

2267 \( \) 140 CONTINUE

2268 \( \) 200 CONTINUE

2270 \( \) RETURN

2271 \( \) END
**SOURCE PROGRAM**

**samgrid.f**

**SOURCE TEXT**

```fortran
SUBROUTINE WINGGEN(XLOCAL, IFLAT, NUL)

  include "xgrid.com"

  This subroutine creates the data when the station is in the plane.
  where some points are on the wing, some are the wake.

  At the x-plane, we put 10 points in the wake part.

  ET = 0.

  IF(ARRIVER > 0.) THEN
    The wing is swept backwards
    DO 75 J=2,NC
      J1 = J-1
      J2 = J
      IF((XLOCAL(J1) > CHORD(J)) OR (XLOCAL(J2) > CHORD(J))) GOTO 75
      XI1 = XLE(J1) + CHORD(J1)
      XI2 = XLE(J2) + CHORD(J2)
      ZTIP = ((XI-X11)/(XI2-X11))*(Z2-Z11)
      IFLAT = STIP
      CALL LININT( XI, XI2, ZTIP, Z11, XLOCAL, YTIP)
      CALL LININT( XI1, XI2, ZTIP, Z21, XLOCAL, YTIP)
      YTIP = (YTIP - ZTIP)*10.0
      DD = 0.01
      DO 75 J=1,10
        YTIP = ZTIP + DD
        CALL LININT( XI, XI2, ZTIP, Z11, XLOCAL, YTIP)
        CALL LININT( XI1, XI2, ZTIP, Z21, XLOCAL, YTIP)
      END
      75 CONTINUE
  ELSE
    The wing is swept forwards
    DO 105 J=2,NC
      J1 = J-1
      J2 = J
      IF((XLOCAL(J1) > CHORD(J)) OR (XLOCAL(J2) > CHORD(J))) GOTO 105
      XI1 = XLE(J1) + CHORD(J1)
      XI2 = XLE(J2) + CHORD(J2)
      ZTIP = ((XI-X11)/(XI2-X11))*(Z2-Z11)
      IFLAT = STIP
      CALL LININT( XI, XI2, ZTIP, Z11, XLOCAL, YTIP)
      CALL LININT( XI1, XI2, ZTIP, Z21, XLOCAL, YTIP)
      YTIP = (YTIP - ZTIP)*10.0
      DD = 0.01
      DO 105 J=1,10
        YTIP = ZTIP + DD
        CALL LININT( XI, XI2, ZTIP, Z11, XLOCAL, YTIP)
        CALL LININT( XI1, XI2, ZTIP, Z21, XLOCAL, YTIP)
      END
      105 CONTINUE
  END
```

**END**
TINT(M) = YY
CONTINUE
DO 100 N=1,1,-1
    IF(ILOCAL.LT.XBASE(1,IFLAT,M)) THEN
        XI = XBASE(1,IFLAT,M)
        YI = YBASE(1,IFLAT,M)
        YI = YBASE(1,IFLAT,M)
        ZI = ZBASE(1,IFLAT,M)
        CALL LININT(X1,X2,YI,YY,X1CBL,ZZ)
        YINT(JI-N+I) = YY
        ZINT(JI-N+J) = ZZ
        GO TO 100
    ENDIF
100 CONTINUE
END
SUBROUTINE WINGIN

include "agrid.com"

C NC = # of sections in the spanwise direction
C XBASE(I) = X value of Ith section
C YBASE(I) = leading edge X value of Ith section
C YLE(I) = leading edge Y value of Ith section
C GBORD(I) = Chord length of the Ith section
C NL = # of points in the upper & lower sections are same physical pts
C Note : the end points of upper and lower sections are same physical pts
C MACH2 configuration:
C READ(10,920) NC, NU
C E = 1
C CONTINUE
C READ(10,930)
C READ(10,950)
C DO 12 I = 1, NU
C READ(I0,*) XBASE(I,2,K), YBASE(I,2,K), EBASE(I,2,K)
C READ(I0,940)
C READ(I0,950)
C DO 15 I = 1, NU
C READ(I0,*) XBASE(I,1,K), YBASE(I,1,K), EBASE(I,1,K)
C READ(I0,940)
C READ(I0,950)
C IF(K.EQ.MC) GO TO 11
C 11 FORMAT(I)
C END
SUBROUTINE FUSEIN

include 'egrid.com'

DIMENSION YT(NFI),ZT(NFI)

NF = 1 of section in the fuselage
NPF = 1 of points in nth section

C Read the fuselage geometry

C MACES configuration

READ(10,810)
READ(10,820)
READ(10,830)NF,NPF

C

40 CONTINUE

C READ (10,840)

M = M + 1

C DO 100 E=1,NPF

CALL DISPAC(TY,E,NPF,TT,ZT,NPF,-16,0)

100 CONTINUE

C WRITE fuselage geometry into PLOTID Planar format.

K=1

E=1

WRITE(51),E,K,M

DO 800 L=1,M

WRITE(51),XIM(L),E=1,EE.

WRITE(51),YIM(L),E=1,EE.

WRITE(51),ZIM(L),E=1,EE.

800 CONTINUE

C

810 FORMAT(1X)

820 FORMAT(1X)

830 FORMAT(1X)

840 FORMAT(1X)

850 FORMAT(1X)

RETURN

END
SUBROUTINE NACIN
  include "sgrid.com"
  DIMENSION TT(MPI),TE(MPI),
  COMMON /NAC/ NNAC(MPI),NNACP(MPI),ZNAC(MPI),ZNACP(MPI),
  COMMON /NDIN/ NNAC,NNACP,ZNAC,ZNACP
  NNAC = # of section in the nacelle
  NNACP = # of points in mth section

  Read the nacelle geometry
  READ(10,810)
  READ(10,830)NNAC,NNACP
  N=0
  40 CONTINUE
  READ(10,840)
  M = M + 1
  DO 100 E = 1,NNACP
    READ(10,850)ZNAC(M),ZNAC(M,E),ZNAC(M,E)
    ZT(E) = ZTAC(M,E)
  100 CONTINUE
  CALL DISTANCE(TT,TE,NNACP,TT,TE,NNACP,-10.,0.)

  Write the nacelle geometry into PLOT3D Plenar format
  DO 800 L = 1,M
    WRITE(32),(ZT(E),E=1,NNACP)
  800 CONTINUE
  CALL flush (32)

RETURN
END
SUBROUTINE WINGMAKER

include 'sgrid.com'

This program generates a 'clipped' delta wing with no twist based on airfoil coordinates read in from fort 80. To use as part of sgrid, WINGM is not necessary (nor is VARIBM).

Written by: Donovan L. Mathias
- July 1982

When possible the same variables are used as in sgrid.f

Fort 80 airfoil coordinates

Description of the wing

Declarations

REAL XTE(LE),SLOPE1,SLOPE2,SCALE,YU(150)
REAL YU(150),YU(150)
real SPAN
INTEGER I,1,J,K

Initialization

read(77,*)NC
read(77,*)XLE(1),XLE(1)
read(77,*)XTE(1),XTE(1)
read(77,*)XBASE(1,1,1),XBASE(1,1,1)

Need airfoil coordinates (L is # of A coords.)

READ(90,*)
READ(90,*)NU
READ(90,*)L
ENDO
READ(90,19)XAF(1),YU(1),YU(1)
ENDO
format(3x,f9.7,3x,f9.7,3x,f9.7)

Establish 1 distance (Spanwise)

SPAN = XBASE(1,1,NC) - XBASE(1,1)

DO K=1,NC-1
   DO I=1,NU
      XBASE(1,1,K) = (K*SPAN/(NC-1))
      XBASE(1,2,K) = (K*SPAN/(NC-1))
   ENDO
   END

Establish sweep (1 FOR LE, 2 FOR TE)

SLOPE1 = (XLE(1)-XLE(1))/XBASE(1,1,NC)-XBASE(1,1,1)
SLOPE2 = (XTE(1)-XTE(1))/XBASE(1,1,NC)-XBASE(1,1,1)

Generate leading and trailing edges

DO K=1,NC
   XLE(K) = XLE(1) + SLOPE1*XBASE(1,1,K)
   XTE(K) = XTE(1) + SLOPE2*XBASE(1,1,K)
ENDO

Distribute grid points

DO K=1,NC
   SCALE = XTE(K)-XLE(K)
   DO I=1,NU
      YLE(I) = SCALE*YU(I)
   ENDO
   END

Return values to original names

DO K=1,NC
   CHORD(K) = ABS(XLE(K)-XTE(K))
   YLE(I) = YBASE(1,1,K)
ENDO
RETURN
END
# 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 *pvm*. 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 *fpvm3.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.
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.
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 )
  integer mytid, me, i
  integer tids(0:NPROC)

  c Enroll in pvm
  call pvmfmytid( mytid )

  c Find out if I am parent or child
  call pvmfparent(tids(0))
  if( tids(0) .lt. 0 ) then
    tids(0) = mytid
    me = 0
  endif
  c start up copies of myself
  call pvmfspawn('spmd', PVMDEFAULT, '*', NPROC-1, tids(1), info)
  c multicast tids array to children
  call pvmfunitsend( PVMDEFAULT, info )
  call pvmfpack( INTEGER4, tids, NPROC, i, info)
  call pvmfmcast( NPROC-1, tids(1), 0, info )
  else
    c receive the tids array and set me
    call pvmfrecv( tids(0), 0, info )
    call pvmfununpack( INTEGER4, tids, NPROC, l, info)
    do 30 i=1, NPROC-1
      if( mytid .eq. tids(i) ) me = i
    continue
  endif

  c all NPROC tasks are equal now
  c and can address each other by tids(0) thru tids(NPROC-1)
  c for each process me => process number [0- (NPROC-1)]
  print*, 'me=', me, ' mytid=', mytid
  call dowork( me, tids, NPROC )

  c program finished exit pvm
  call pvmexit(info)
  stop
end
```
Notes
subroutine dowork( me, tids, nproc )
include '../include/fpvm3.h'

Simple subroutine to pass a token around a ring

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.

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
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 \texttt{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 \texttt{uid}. 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 \texttt{info} less than zero indicate an error.

**call pvmfkill(tid,info)**
This routine kills a PVM task identified by \texttt{uid}. Values of \texttt{info} less than zero indicate an error.

**call pvmfspawn(pname,flag,where,ntask,tids,numt)**
This routine starts up \texttt{ntask} instances of a single process named \texttt{pname} on the virtual machine. Here are the definition of the other arguments:

<table>
<thead>
<tr>
<th>\textbf{flag}</th>
<th>\textbf{Option Value}</th>
<th>\textbf{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>

*Note* You should always check \texttt{tids} and \texttt{numt} to make sure all processes started correctly.
call pvmfparent( tld )
This routine returns the "id of the process that spawned this task. If the calling process was not created with pvmfspawn, then tld=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, bufid )
This routine clears the send buffer, and creates a new one for packing a new message.

<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>

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

call pvmffreebuf( bufid, info)
This routine disposes the buffer with identifier bufid. 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.

what indicates what type of data xp is

<table>
<thead>
<tr>
<th>STRING (0)</th>
<th>REAL (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE1 (1)</td>
<td>COMPLEX8 (5)</td>
</tr>
<tr>
<td>INTEGER2 (2)</td>
<td>REAL8 (6)</td>
</tr>
<tr>
<td>INTEGER4 (3)</td>
<td>COMPLEX16 (7)</td>
</tr>
</tbody>
</table>

nitem 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 pvmfrecv, except that it does not block the flow of your program. If the requested message has not arrived, this routine returns bufid=-1. 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 pvmfrecv 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 pvmbufinfo (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.
5

Golden Section

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

\[
\begin{align*}
    a_3 &= (1-\alpha) \, a_1 + \alpha \, a_2 \\
    a_4 &= \alpha \, a_1 + (1-\alpha) \, a_2
\end{align*}
\]  

(EQ 1)  

(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:

<table>
<thead>
<tr>
<th>If ( f(a_4) &lt; f(a_3) )</th>
<th>If ( f(a_4) &gt; f(a_3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Consider new interval ( (a_1, a_4) )</td>
<td>1 Consider new interval ( (a_3, a_2) )</td>
</tr>
<tr>
<td>2 Apply EQ.(1) and (2) again</td>
<td>2 Apply EQ. (1) and (2) again</td>
</tr>
<tr>
<td>3 Until maximum is reached</td>
<td>3 Until maximum is reached</td>
</tr>
</tbody>
</table>

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

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

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

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

Function evaluation

Equations (1) and (2)

A(3) = (1. - ALPHA)*A(1) + ALPHA*A(2)
A(4) = ALPHA*A(1) + (1. - ALPHA)*A(2)
FN(1) = F(A(1))
FN(2) = F(A(2))
FN(3) = F(A(3))
FN(4) = F(A(4))
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', ERR
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

CONTINUE
STOP
END
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.
   
   ```
   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 \( m \).
   
   ```
   pvmfpack(REAL4, A, m, l, info)
   ```

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

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

PROGRAM MASTER

include '../include/fpvm3.h'
DIMENSION A(4), FN(4)
integer tids(0:32), who
character*8 where
character*12 pname

Enroll this program in PVM
call pvmfmytid(mytid)

Start up the four processors
nproc = 4
where = '*'
pname = 'function'
call pvmfspawn(pname, PVMDFAULT, where, nproc, tids, numt)
do 20 i=0, nproc-1
   write(*,*) 'tid', i, tids(i)
20 continue

Initial interval
L = 0
A(1) = 0.4
A(2) = 1.6
ALPHA = 0.382
TOL = 1.E-3
ERR = 1.

CONTINUE

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 pvmfinitsend(PVMDEFAULT, info)
call pvmfpack(INTEGER4, nproc, 1, 1, info)
call pvmfpack(INTEGER4, tids, nproc, 1, info)
call pvmfpack(REAL4, A, 4, 1, info)
call pvmfpack(REAL4, ERR, 1, 1, info)

msgtype = 1
call pvmfmcast(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.
```
c    Wait for results from processors

msgtype value matches the one sent from Slave program

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

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
```

```
call pvmfexit(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 \((tids(1), \ldots, tids(4))\) 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 `PVMDefault`, `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 \((mtid)\) of parent process. This is useful for returning solutions to the Master.
   ```
   pvmfparent(mtid)
   ```

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**

Program function

```c
#include '../include/fpvm3.h'

integer tids(0:32), who
real a(32)
tor = 1.e-3

Enroll this program in PVM
call pvmfmytid(mytid)

Get the parent's task id
call pvmfparent(mtid)

continue

Receive data from host
msgtype = 1
call pvmfrecv(mtid, msgtype, info)
call pvmfunpack(INTEGER4, nproc, 1, info)
call pvmfunpack(INTEGER4, tids, nproc, 1, info)
call pvmfunpack(REAL4, A, 4, 1, info)
call pvmfunpack(REAL4, ERR, 1, 1, info)

if(err.le.tor) go to 99

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

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

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

Program finished. Leave PVM before exiting
continue
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

```
Mkfile

# Custom section
# Set PVM_ARCH to your architecture type (SUN4, HP9K, RS6K, SGI, etc.)
# if PVM_ARCH is BSD386 then set ARCHLIB = -lrpc
# if PVM_ARCH is SGI then set ARCHLIB = -lsun
# if PVM_ARCH is I860 then set ARCHLIB = -lrpc -lsocket
# if PVM_ARCH is 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
#
PVMDIR    =  /amd/fs02/pub/iris4d_irix4/nas/pkg/pvm3.2
PVMLIB     =  $(PVMDIR)/lib/$(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      =  $(PVMDIR)/lib/$(PVM_ARCH)/libfPVm3.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)
```

PVM Library

Make appropriate changes for your own path
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 `pvm` at UNIX prompt, is `libpvm [pid-1]: Console: Can’t start pvm`, it is possible that the last time you halted PVM daemon, the daemon created a residual file `/tmp/pvmd.xxxx`, 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 `~/.rhosts` directory, and this file points to your local computer.

- Make sure the `.rhosts` file is “read” and “write” protected from others users.
Host File

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
```

host file

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 `pvmdspawn`) 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
```
Problems and Tips

Place to jot down problems.

Notes

If encounter problems, please contact:

Merritt Smith: mhsmith@nas.nasa.gov

or

Samson Cheung: cheung@nas.nasa.gov
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>Thinking Machines CM2</td>
<td>Sun front-end</td>
</tr>
<tr>
<td>CM5</td>
<td>Thinking 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>CRAYSMP</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>IB60</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>Ultrix</td>
</tr>
<tr>
<td>RS6K</td>
<td>IBM/RS6000</td>
<td>AIX</td>
</tr>
<tr>
<td>RT</td>
<td>IBM RT</td>
<td></td>
</tr>
<tr>
<td>SGI</td>
<td>Silicon Graphics IRIS</td>
<td>link -lun</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>
### Appendix

**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 message</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>