Supersonic Civil Airplane Study and Design: Performance and Sonic Boom

Samson Cheung

January 1995
NCC2-617

Surface Grid Generator

Upwind Parachuted
Navier-Stokes code
(finite volume scheme)

Isotropic Euler
Axisymmetric code
(method of characteristics)

NO
L/D optimized?
YES

OPTIMIZER

CFD Calculation
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$^6$; a sequential quadratic programming algorithm in which the search direction is the solution of a quadratic programming subproblem
- HYPGEN: hyperbolic grid generator$^7$; 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$^8$
- 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$^9$ 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 summaries the result of this study. The nose of the body was trimmed to reduce the wave drag. Since the total volume was constrained, volume was added near the end of body. Total wave drag reduction was by 6%. The results were presented in a AIAA meeting\textsuperscript{12} and published in Journal of Aircraft Vol. 32, No. 1, Jan/Feb. 1995.

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

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

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

The flow solver being used was Overflow code, a 3-D Navier-Stokes code using the diagonal with ARC3D algorithm \(^{15}\). One of the most challenging works of this project was to reduce the separation on the left wing (trailing wing). The separation on the upper surface of the wing and the corresponding vortices are shown in the left side of the figure below. It was found that bending of the wing could abate the separation, as well as improve the lift-to-drag ratio. The right side of the figure shows a weaker separation pattern on the ended
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 personnel 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 concepts is the network-based distributed computing. The Parallel Virtual Machine (PVM) is a software package that allows a heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource. PVM allows users to link up engineering workstations to work as a single distributed-memory (parallel) machine. Merritt Smith and I wrote a manual on PVM for beginning users. A copy of the manual is attached in Appendix C.

A parallel optimizer based on nonlinear Quasi-Newton method has been developed and coupled with an efficient CFD code for basic aerodynamic design and study. This optimizer is called 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 Lightbody F-function on body surface,
2) input data
3) define parameters
4) the overpressure signature at given distance R1
5) the loudness level of the sonic boom at R1

INPUT:
- LHF.in(3) (Input parameter
- area.@(3) (Equivalent area distribution (INAREA=0)
- data(4) (F-function distribution (INAREA=2)
- coef.dat(3) (F-function due to lift (LIFT=1)
- @.dat(3) (Surface grid - FGRID placer format. (INAREA=1)
- Default case Wing-body (INAREA=1)

OUTPUT:
- area.out(12) (Equivalent area distribution and its derivative,
- @.out(13) (F-functions on the body surface and at distance R1,
- Force (F), Pressure Signature at distance R1,
- Curve_F(14) (Integral curve of the shifted F-function.

Er. Seung Cheung [Tel: (415)-604-4462]
NASA Ames Research Center
M/S 215-01
Moffett Field, CA 94033
Date: 1992/7/4 Version 2.1

PARAMETER (INX=220, INY=350, INZ=550, INA=100)

DIMENSION X(INX, INY, INZ), Y(INX, INY, INZ), Z(INX, INY, INZ)
REAL H(INX, INY), Z(INX, INY), TAU(INX, INY, INZ), TAUH(INX, INY, INZ)
COMMON/PAR, FPAR, FPAR, JDI, INAREA, LIFT, TURTLE

LOGICAL W

W = TRUE

OPEN UNIT, FILE='LHF.in', STATUS='OLD'

Read the input parameters

PAREM LIST, PAR, FPAR, JDI, INAREA, LIFT, TURTLE

READ(1,PARA)

WRITE(1,PARA)

Input free-stream Mach number = FPAR
If FPAR > 0, sonic boom varies time, else varies distance

Is the surface grid contains the whole configuration, or
only half-plane or only quarter-plane?

FFPAR = 1: Whole-plane
FFPAR = 2: Half-plane
FFPAR = 3: Quarter-plane

RI will be the distance where the signature is captured.

If read in area distribution, INAREA = 0
If read the grid, INAREA = 1
If read the wing-body case, INAREA = 2
If read in F-function, INAREA = 3
If read a signature at R1, INAREA = 4

PI = 4. * ATAN(1.)

JDI = JMAX

If JDIM > 361, GOTO 790

If JDIM > 0, we read the pressure signature at R1 and extrapolate

GOTO 210

Find the area distribution of body configuration (sample case).

IF(INAREA EQ 0) THEN

CALL WGRID(JDI, S, TAU)

W = TRUE

CALL CORD(JDI, S, TAU)

CALL WIND(JDI, S, TAU)

CALL SEARCH(JDI, S, TAU)

CALL BULLEEP(JDI, S, TAU)

GOTO 270 ENDIF

Read the given area distribution

IF(INAREA EQ 0) THEN

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

DO 50 J=1, INMAX

READ(2, *, END=75) TAU(J), S(J)

50 CONTINUE

CLOSE(2)

IF(DI = 1, JDI = JMAX)

OPEN(UNIT=3, FILE='area.in')

DO 90 J=1, JDI

WRITE(2, *) TAU(J), S(J)

90 CONTINUE

GOTO 270 ENDIF

Read in the F-function or define a F-function by calling FUNC

IF(INAREA EQ 2) THEN

CALL FUNC(TAU, TAUH, JDI)

OPEN(UNIT=4, FILE='f.dat')

DO 100 J=1, INMAX

READ(4, *, END=110) TAU(J), TAUH(J)

100 CONTINUE

ENDIF

Read the F-function or define a F-function by calling FUNC

IF(INAREA EQ 2) THEN

CALL FUNC(TAUH, TAU, JDI)

OPEN(UNIT=4, FILE='f.dat')

DO 100 J=1, INMAX

READ(4, *, END=110) TAU(J), TAUH(J)

100 CONTINUE

ENDIF

END
CALL DISTARC(TAU,FTAU,J-1,TAU,FTAU,JDIM,10.,0)
CALL CAREA(S,FTAU,TAU,JDIM)
GOTO 270
ENDIF
DO 220 J=1,JDIM
S(J) = FFAC*S(J)
TAU(J) = TAU(J-1)
220 CONTINUE
DO 270 J=1,JDIM
OPEN(UNIT=12,FILE='grid.in',FORM='UNFORMATTED')
READ(12,1,FORMAT(2X,El6.0,1Z,El.e,8))
WRITE(12,1,FORMAT(2X,El6.0,1Z,El.e,8))
END
SUBROUTINE LIGHTI(TAU,R,SP,N,FACIa,FTAU)

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

C
P1= 4.*ATAN(1.)
BETA=SQR(T(NACH)**2-1.0)
TAU(1)=0.
FTAU(1)=0.

C
DO 95 N=1,N
95 FTAU(N)=0.

DO 100 J=1,N
100 CONTINUE
RETURN
END

C
DO 102 I=2,N
IF(ABS(R(I)) LE.1.E-10) THEN
Z1 = 1.E+10
F4 = 0.
GOTO 98
ENDIF
AB=2.Q/BETA*R(I))
ABI=ABS(AB)
F1=SQRF(ABI)
F2=SP(1)-SP(1-1)
F3=F1+F2
F4=F3/(2.0+F1)
Z1=TAU(I)/BETA*R(I))
XLO=1.0
IF (Z1.XLO) GO TO 96
IF (Z1.LT.XLO) GO TO 97
IF (Z1.GE.4.0) GO TO 98
IF (Z1.LE.4.0) GO TO 99
96 R2=0.
FTAU(3)=FTAU(3)+R2+P6
GO TO 99
97 R2=0.0/(2.0+XI)
ABI=ABS(R2)
R2=SQRT(R2)
FTAU(3)=FTAU(3)+R2+P4
GO TO 99
98 CONTINUE
100 CONTINUE
CONTINUE
RETURN
END
SUBROUTINE SQUAREA(EDIM,LDIM,JDIM,X,Y,Z,EMAX,LMAX,JMAX,K)

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

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

REAL S(3,JDIM)

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

S The unwanted base area.

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

S The area surrounded by half of the plane.

S(J) = ABS(SAREA)-BASE

10 CONTINUE
RETURN
END
SUBROUTINE F(N, F, FMACH, NP, RO, RI, TMAX)

PARAMETER (INMAX=1400)
DIMENSION F(NP), X(NP), Y(NMAX), P(INMAX)
DIMENSION YSTP(INMAX)
DIMENSION DRIVE(3)

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

Input of initial parameters and pressure signal

PMACHE = Free-stream Mach number
RO = Initial distance from the body (altitude)
WP = Initial distance from the body (altitude)
INMAX = Number of data points (WP < INMAX)
TMAX = This should be large enough to resolve the signature
NAMELIST /PSCALE, ESSCALE, PSCALE, A, P, P, IRISE

Read the input parameters
READ(14, EPSCALE)
WRITE(6, EPSCALE)

Define the parameters used in the F-function theory

GAMMA = 1.4
B = SQRT(3)*(GAMMA+2-1) / (GAMMA+4)
CAP = (GAMMA+1) / (GAMMA+4)
SRBL = SQRT(B)

IF(R0 > 0, extrapolate from R0 to R1. First calculate the F-func.
IF(R0 < 0.5, THEN
OPEN(UNIT=4, FILE='p.ro')
DO 15 I=1, INMAX
HEAD(R), ENDS(10)(I), P(I)
15 CONTINUE
DO 30 I=1, NP
HEA = SQRT(R)
DO 35 J=1, NP
P(I) = SQRT(1+B*R) * P(I) / (GAMMA+PMACHE+PMACHE)
35 CONTINUE
ENDIF

Y = transposed coordinates
WRITE(13, 49)
DO 40 I=1, NP
Y(I) = Y(I) - CAP*SRBL*P(I)
40 WRITE(13, 49), X(I), Y(I)
49 FORMULATE a transposed F-function at surface
CONTINUE

Find the largest and smallest values of Y

THIN = YMIN = 1.8+B
DO 50 I=1, NP
IF(Y(I) < 1.8+B*RO) THEN
THIN = Y(I)
50 CONTINUE

Print out the integral curve of the shifted F-function
CALL INTF(NP, T, Y)

Need to march in Y-direction, define the step

YSTP(I) = YMIN
DYDS = YMAX - YMIN
DY = DYDS/YSTP(NP)
DO 80 J=2, NP
YSTP(J) = YSTP(J-1) + DY
80 CONTINUE

March through the shifted F-function, check area-balance and place the shock.
CALL MARCH(INMAX, NP, Y, YSTP, T)

Obtain the solution
NOTE: IF TOR>=0, the sonic boom is in the form of (P-Plift) vs time
as in it is. As the form (P-Plift)/Plift vs distance.
DO 150 J=1, NP
P(I) = GAMMA*PMACHE*PMACHE*(P(I)/SQRT(2+B*R))
150 X(I) = X(I) + B*R1

150 CONTINUE

Make the data points in evenly distributed manner and scale the sonic if desired
DO 180 J=1, NP
X(I) = X(I)*PSCALE
P(I) = P(I)*PSCALE
180 CONTINUE

Atmospheric aspect
ALT = Altitude
A = speed of sound at ground level ft/sec
PS = reference pressure lb./ft.2 = SQRT(PA*P)

Cont pressure at the ground
PC = pressure at flight altitude
PO = SQRT(PA*P)
VEL = PMACHE*A
TREX = X(I)/VEL
IF(TREX,CY>.9, THEN
DO 260 I=1, NP
X(I) = X(I)/VEL - TREX
F(I) = P(I)*P0
260 CONTINUE

The signal (DP vs Time) is calculated, use a empirical program to calculate the rise time, and embed the rise time into the signature.
C Note: Unit used is still the stupid English unit!
CALL MISVIEW(FNAME,P,X,RY,ALT,THRSE)
C Obtain the noise level
CALL NOISE(DBIVAL,X,P,NP)
C Write the dB(PL) value out
WRITE(14,500)DBIVAL(1),DBIVAL(2),DBIVAL(3)
500 FORMAT('Noise level ',F10.4,'dB(PL)',3X,F10.4,'dB(A)',3X,F10.4,'dB(C)')
ENDIF
C Write the sonic boom
WRITE(14,555)X
555 FORMAT(20X'Take pressure signal at Ri = ',F10.4)
DO 670 I=1,NP
WRITE(14,700)X(I),P(I)
670 CONTINUE
700 FORMAT(3X,E20.8,2X,E16.6)
CLOSE(14)
C RETURN
END
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')
C
SUMF = 0
DO 100 J=2,NP
   DY = Y(J)-Y(J-1)
   SUMF = SUMF + 6.5*DY*(F(J)+F(J-1))
100 WRITE(34,120)Y(J),SUMF
C
CLOSE(34)
RETURN
END
SUBROUTINE SHPT(NMAX, NP, Y, YSTP, F, INDEX, FS, YS, INSCT)
DIMENSION Y(NP), YSTP(NP)
DIMENSION INDEX(40), FS(40)
COMMON/SHOC/K/

YEND = Y(NP)
FIRST = 1
DO 300 J=2, NMAX
YST = YSTP(J)

CALL POINT(NP, Y, F, INDEX, FS, YS, INSCT)

IF(INST.GT.2) THEN
  IF(INSCT.EQ.3) INST = 3
  IE = INDEX(INST)
  CALL AREA(NP, Y, F, FS, YS, IE, IFLAT2)
ELSE
  The tail shock is already formed, leave program
ENDIF

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

IF(IFLAT = 0, IE is the point that have area balanced.
IF IFLAT2 and IFLAT1 are in different sign, i.e.,
Use bisection method to find the correct point Y(ISTART)
ELSE
  IF(IFLAT1.GT.0) THEN
    Y1 = YSTP(J-1)
    Y2 = YSTP(J)
    NC = 500
    DO 200 IC = 1, NC
      Y = 0.5(Y1 + Y2)
      CALL POINT(NP, Y, F, INDEX, FS, YS, INSCT)
      IF(INSCT.EQ.3) INST = 3
      IE = INDEX(INST)
      CALL AREA(NP, Y, F, FS, YS, IE, IFLAT2)
      IF(IFLAT.EQ.0) RETURN
      IF(IFLAT.GT.0) THEN
        Y1 = Y
        IFLAT1 = IFLAT2
        ELSE
        Y1 = YS
      ENDIF
      CONTINUE
    WRITE(* ,*) 'After ', NC, ' steps of bisection'
    RETURN
    ELSE
      IFLAT1 = IFLAT2
      GOTO 300
    ENDIF
  ENDIF
  FIRST = -1
  CONTINUE
RETURN
END
SUBROUTINE POINT(NP,Y,F,INDEX,FS,YS,INRECT)
DIMENSION Y(NP),F(NP)
DIMENSION INDEX(40),FS(40)

C Find the points FS on the F-function when YS is given
INDEX = the index runs from 1 to NP
INRECT = f of points being intersected, at least point to do integration

INRECT = 0
IF(YS .LE. Y(1)) THEN
    INRECT = INRECT + 1
INDEX(INRECT) = 0
END IF

DO 100 J = 2,NP
    FAC1 = YS - Y(J)
    IF(FAC1 .LE. 0.0) THEN
        IF(ABS(Y(J-1)-Y(J)) .LE. 1.E-14) THEN
            WRITE(*,*)'ZEROOOOOOO',YS,Y(J),'
            INRECT = 0
            RETURN
        END IF
    END IF
    INRECT = INRECT + 1
    SL = (Y(J)-F(J-1))/(Y(J-1)-Y(J-1))
    FS(INRECT) = F(J-1)+SL*(YS-Y(J-1))
INDEX(INRECT) = J
END IF
100 CONTINUE

IF(YS .GT. Y(NP)) THEN
    INRECT = INRECT + 1
    FS(INRECT) = F(NP)
    Y(NP) = YS
INDEX(INRECT) = NP
END IF

RETURN
END
SUBROUTINE AREA(HF,Y,F,Y0,H,F0,Y00,ITYP,IFLAT)

DIMENSION Y(NP),F(NP),FS(NP)

COMMON/SNDRK/, INGCT

C Fix the integral of F by trapezoidal rule
C Integrating from -IN to IN. E1 is area that from Y0 to Y(NP)
C and E2 is the area that from Y(IN) to Y0. Thus E2 should be
C subtracted out and E1 should be added in

E1 = 0.5*(Y(IN)-Y0)*(F(IN)+FS(IN))
E2 = 0.5*(Y0-Y(IN))*(F(IN)+FS(IN))
Areal = E1

DO 10 IN = IN+1
SLAP = 0.5*(Y(IN)-Y(IN-1))*(F(IN-1)+F(IN))
Areal = area + SLAP
10 CONTINUE

A = AREAL - E1
IF(A.GT.0) IFLAT=1
IF(A.LT.0) IFLAT=-1

RETURN
END
SUBROUTINE BODY(JDIM, S, TAU)
C This subroutine finds the area distribution of
C the wing-body configuration.
C
DIMENSION S(JDIM), TAU(JDIM)
C
PI = 4. * ATAN(1.)
ANG = 80. * PI/180.
ANG1 = 80. * PI/180.

DO 2 J = 2, JDIM

TTT = TAU(J) - 7.01
IF (TTT .GT. 0.) TTT = 0.
RR = 0.54 - 0.011 * TTT**2
S(J) = PI * RR**2

IF (TTT .GT. 0.) TTT = 0.
RR = 0.54 - 0.011 * TTT**2
S(J) = PI * RR**2

IF (TTT .GT. 0.) TTT = 0.
RR = 0.54 - 0.011 * TTT**2
S(J) = PI * RR**2

IF (TTT .GT. 0.) TTT = 0.
RR = 0.54 - 0.011 * TTT**2
S(J) = PI * RR**2

CONTINUE
RETURN
END
SUBROUTINE CONE(JDIM, S, TAU)
C This subroutine finds the area distribution of the cone-cylinder
C with half-angle 3.24 degree and 8.8 units of length.
C
DIMENSION S(JDIM), TAU(JDIM)

PI = 4. * ATAN(1.)
ANG = 3.24 * PI / 180.
DX = 16. / PI / ANG / (DIM - 1)
S(1) = 0.
TAU(1) = 0.
DO 2 J = 2, DIM
   TAU(J) = TAU(J - 1) + DX
   IF (TAU(J) <= 8.8) THEN
      R = TAU(J) * TAN(ANG)
   ELSE
      R = 8.8 * TAN(ANG)
   ENDIF
   S(J) = PI * R * R
2 CONTINUE
RETURN
END
SUBROUTINE SHEX(A,JDIM,TAU)

C This subroutine finds the area distribution of a Sears-Hassack body
C with fineness ratio 23.5.
C
DIMENSION S(JDIM),TAU(JDIM)

C # of point on body + # of point on sting = JDIM

JDIM = JDIM + 2,

PI = 4.*ATAN(1.,'
BL = 1.,
TOUT = 1.9*BL
F = 23.5

DTHETA = PI/FLOAT(JBDY-1)

DX = BL/FLOAT(JBDY-1):

RMAX = BL/(2.*F)

S(1) = 0.

TAU(1) = 0.

C Consists of Sears-Adams body

write(*,'(A,A)')'Input Abase/Amex'
read(*,'(A,A)')AA
write(*,'(A,A)')'Input RMAX'
read(*,'(A,A)')AA

RMAX = AA*BL

CONST = AA/EE

C1 = 1./2.*(2.*RMAX/BL - 1.)

DO 2 J=2,JBDY

THETA = PI-DTHETA*FLOAT(J-1)

TAU(J) = (1.+COS(THETA))*BL/2.

TAU(J) = FLOAT(J-1)*DX

THETA = ACOS(2.*TAU(J)/BL - 1.)

Sears-Hassack body

POS = SIN(THETA)**2

Sears-Adams body

5

POS = const*(PI-SIN(THETA)**2)

(4./3.)*C1*(SIN(THETA)**2)

E = RMAX+SQRT(ABS(POS))

S(J) = PI*E+R

2 CONTINUE

C Add a sting

DX = (TOUT-BL)/FLOAT(JBDY)

DO 5 J=JBDY+1,JDIM

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

S(J) = S(J-1)

5 CONTINUE

RETURN

END
SUBROUTINE BULLET(JDIM,5,TAU)
C This subroutine find the area distribution of a bullet with a form
C R = 5.0 gms
C DIMENSION S(JDIM),TAU(JDIM)

PI=4.0*ATAN(1.)
BL = 4.
GAMA = 0.65
RBASE = 0.25
A = RBASE/(BL*GAMA)
TOTLEN = BL + 2.*BL
DX = TOTLEN/FLOAT(JDIM-1)
S(1) = 0.
TAU(1) = 0.
DO 2 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
   CONTINUE
RETURN
END
SUBROUTINE WING(JDIM, S, TAU)

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

   C
   C
   Pi=4.*ATAN(1.)
   STING = 0.
   DX = 1./FLOAT(JDIM-1)
   S(J) = 0.
   TAU(1) = 0.
   DO J=2,JDIM
       TAU(J)+TAU(J-1)+DX
       Z = (PI/12.5)*((TAU(J)-0.5*TAU(J)*TAU(J))
       S(J) = 1.
       IF(TAU(J) .GT. 1.7087) THEN
           STING=PI*0.0625*0.0625
           S(J) = Z + STING - Z*0.125
           ENDIF
       ELSE
           STING=PI*0.0625*0.0625
           S(J) = STING
           ENDIF
       END
   CONTINUE
   RETURN
END
SUBROUTINE BRUNC(JDIM, S, TAU)

PARAMETER (NMAX=800)
DIMENSION S(JDIM), TAU(JDIM), S(NMAX), X(NMAX)
COMMON/PHACK/ PHACK, PHAC

OPEN(UNIT=33, FILE='coef.dat', FORM='FORMATTED')
READ(33, 12)
READ(33, 12)
READ(33, 12)
READ(33, 12)
READ(33, 12)
DO 10 J=1, NMAX

READ(33, 15, END=17) X(I), CL, CD, SLOD, B(I), CH
READ(33, 15, END=17) X(I), B(I)

B(1) = B(1)*PPAC
10 CONTINUE

IPOINT = 1
OPEN(UNIT=33, FILE='b/a.dat')
DO 20 I=1, IPOINT
WRITE(33, *) A(I), B(I)
20 CONTINUE

ISTART = 1
DO 50 J=1, JDIM
DO 50 I=ISTART, IPOINT
IF(ABS(X(I)-TAU(J)) LE.1.E-10) THEN
S(J) = S(2) - B(I)
ISTART = I
GOTO 40
ENDIF
IF(X(I) GT TAU(J)) THEN
BP = 0.0
IF(X(I) EQ. 1.0) THEN
BP = 0.0
ELSE
BP = B(I-1)
ENDIF
SLOPE = (B(I) - BP)/(X(I) - XF)
BT = B(I) + SLOPE*(TAU(J) - X(I))
S(J) = S(J) + BT
ISTART = 1
ENDIF
IF(X(I) EQ. 1.0) THEN
ISTART = I
GOTO 40
ENDIF
ELSE
IF(X(I) LT. NPOINT) GOTO 30
ENDIF
30 CONTINUE
40 CONTINUE
50 CONTINUE
RETURN
END
SUBROUTINE WBJ(DIM, SP, TAU)
C add where the integral of equivalent area
C This is a test for wing-body case
DIMENSION SP(DIM), TAU(DIM)
COMMON/PAR, FMAC, PFAC
C
DO 10 J=1,DIM
IF(TAU(J).GE.8.2 .AND. TAU(J).LE.12.25)
     L SP(J) = SP(J) + 4.* .05 *.54
10 CONTINUE
RETURN
END
SUBROUTINE F_NC(TAU,FTAU,JDIM)

DIMENSION TAU(JDIM),FTAU(JDIM)

NAMELIST /FFUNC/ YF,ELAM,C,B,D,E,BL,YR,DEL

READ(3,FFUNC)
WRITE(6,FFUNC)

TAU(1)=0.
FTAU(1)=0.
DT=TR/FLOAT(JDIM-1)

DO 10 J=2,JDIM

TAU(J)=TAU(J-1)+DT
10 IF(TAU(J).GE.YF) GOTO 6

IF(TAU(J).LE.YF/2.) FTAU(J)=2.*TAU(J)/YF

DO 6 J=2,JDIM

TAU(J)=TAU(J-1)+DY
6

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

FTAU(J)=C*(TAU(J)/YF-1.) - B*(TAU(J)/YF-1.)

IF(TAU(J).GE.YF AND. TAU(J).LE.DEL)

FTAU(J)=C

IF(TAU(J).GE.YF. AND. TAU(J).LE.BL)

IF(TAU(J).GE.YF. AND. TAU(J).LE.BL)

DO 10 J=1,JDIM

WRITE(13,80) TAU(J), FTAU(J)
10 CONTINUE

CALL DISTANCE(TAU,FTAU,JDIM,TAU,FTAU,JDIM,16,0)

WRITE(13,75)

DO 20 J=1,JDIM

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

20 CONTINUE

FORMAT(2X,F8.4,IXE16.8)

RETURN

END
SUBROUTINE AREA(5,PTAU,TAU,JOIM)

DIMENSION 5(500),TAU(500),PTAU(500)

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

C

A(1) = \int_0^\infty \frac{1}{\sqrt{y-t}} f(t) \, dt \, dy

C

S(I) = 0

TAU(I) = 0.

DO 10 J=1,JOIM

SS = 0.

DO 7 I=1,J

DT = TAU(I) - TAU(I-1)

FINTL = 0.

DO 5 J=I+1,J

DT = TAU(I+1) - TAU(I)

FINTL = FINTL + DT*PTAU(I)/SRT(TAU(I)-TAU(I-1))

5 CONTINUE

6 CONTINUE

SS = SS + 2.*FINTL

6 CONTINUE

SS = SS + 4.*SRT(TAU(J)-TAU(J-1))*PTAU(J)*DT

10 CONTINUE

C

320 FORMAT(2E8.4,1X,E16.8)

RETURN

END
SUBROUTINE ELTIME(PMACH, P,T, HP, ALT, IRIS)

An empirical method to calculate the rise time of a sonic boom in flight test data. Good for N-wave type of signal, may be somewhat conservative (shorter rise time).

All units used are English units!!!

PMACH = Free-stream Mach number
T = Sonic boom
P = Shock strength
ALT = Altitude (ft)
PS = Free-stream pressure (lb/ft2)
R = Rise time (sec)
S = Temperature R = T(459.67) * (8/5)

DIMENSION HP, T(HP)

DO 211 HP = 2114.2
TEMP = 518.69

DO 12 COUNT = 0
DO 12 COUNT = COUNT + 1
COUNT = 12
CONTINUE

DO 12 1 = IHP, HP
IF (1 = P, HP EQ. (T(I+1)) THEN
IF (1 = P, HP EQ. (P, HP))
PS = ABS(P, HP - P, HP)
ELSE
IF (P, PS EQ. 0.) THEN
GOTO 30
ELSE
GOTO 40
ENDIF

CONTINUE

DO 30 I = 1, HP
T(I) = T(I) + DRT
200 CONTINUE

DO 300 I = IHP + 1, IVP
T(I) = T(I) + RT
300 CONTINUE

IF (1 = HP EQ. 10) GOTO 12
RETURN
END
SUBROUTINE DISTARC(X,Y,Z,NEW, I, NEW, N, MAX, NNEW, IFLAT)

      DIMENSION X(N),Y(N), NEW(NMAX), I, NEW(NMAX)

      IF(NEW(1).EQ.0) GOTO 40

      IF(NEW(1).GT.0) THEN
         CALL DISTRI(DELT, NNEW, 5, IFLAT)
      ELSE
         CALL DISTRK(DELTA, NNEW, 5, IFLAT)
      END IF

      NMAX = 1 + NNEW - 1
      DO 35 J = 1, N
         X(J) = X(J) + DELT
         Y(J) = Y(J) + DELT
      END DO
      CALL DISTRK(DELTA, NNEW, 5, IFLAT)

      35 CONTINUE

      40 CONTINUE

      30 CONTINUE

      20 CONTINUE

      10 CONTINUE

      00 CONTINUE

      END

      SUBROUTINE DISTRI(DELT, NNEW, 5, IFLAT)

      CALL DISTRK(DELTA, NNEW, 5, IFLAT)

      END

      SUBROUTINE DISTRK(DELTA, NNEW, 5, IFLAT)

      CALL DISTRK(DELTA, NNEW, 5, IFLAT)

      END
1077 **********************************************************************
1078 SUBROUTINE DISTRI(YANG,ETCS,5,IFINE)
1079 PARAMETER (MAX=500)
1080 DIMENSION S(MAX),DUN(MAX)
1081 C Calculating the stretching functions S when given
1082 C the first spacing, YANG, and the number of points ETCS
1083 C
1084 C IF IFINE=1, distribution is coaltering at outer grid
1085 C
1086 IF(MAX.LE.ETCS) THEN
1087 WRITE(*,*)'SUB DISTRI : MAX is less than ETCS.'
1088 STOP
1089 ENDIF
1090 IF(IFCS.EQ.1) THEN
1091 S(I) = 0.
1092 GOTO 40
1093 ENDIF
1094 C
1095 DO1 = YANG
1096 EPS = ETCS-1
1097 DIETA = 1./FLOAT(EPS)
1098 EDBETA = 1.5
1099 CALL GEBET(DO1,EPS,.0001,100,EDBETA)
1100 CALL FILL(ETCS,EDBETA,DIETA,S)
1101 C
1102 IF (IFINE.EQ.1) THEN
1103 DO 37 EPS = 1.,ETCS
1104 DUN(ETCS-EPS+1) = S(EPS)
1105 CONTINUE
1106 DO 38 EPS = 1.,ETCS
1107 S(EPS) = 1.-DUN(EPS)
1108 CONTINUE
1109 CONTINUE
1110 RETURN
1111 END
DIMENSION Z(250)

IF(TBETA.EQ.1.0)
   DO 10 L=1,LI
      Z(L)=0.0
   CONTINUE

ELSE
   DO 20 L=1,LI
      ETA = (L-1)*DET
      ER = (TBETA+I)/(TBETA-I)
      EEE = 1.0 - ETA
      RSETA = RR*EEE
      Z(L) = (TBETA-I)*(RR-RSETA)/(TBETA+I)
   CONTINUE

END IF

RETURN
SUBROUTINE GRED(DFM,NPT,FPCC,ICC,BETA)

DIMENSION T(350)
ICCL=ICC
FPCC=FPCC+DFM
BETA=BETA
DF=DFM
SS=1./DF
SR=1.
FF=21.
III=ICC/10
DO 10 II=1,III
BP=BETA
BETA=0.5*(BETA+1.)
CALL P2(1,BP,DF,ST)
FP=521.
IF(FF .GT. 0.) GO TO 15
BETA=BETA+2.*BETA-1.
10 CONTINUE
DO 15 CONTINUE
DO 5 II=1,ICCL
CALL P2(2,BETA,DF,ST)
F=521.
IF(F .GT. 0.) THEN
FP=F
BP=BETA
ELSE
FP=FP
BP=BETA
END IF
BETA=0.5*(BP+SR)
IF(ABS(F) .LT. FPCC) GO TO 4
5 CONTINUE
WRITE(6,100) BETA,F
100 FORMAT(180,3,E18.6) BETA,F
4 CONTINUE
CALL P2(2,BETA,DF,ST)
C
P=521.
B1=BETA-1.
B Len=BP-1.
B Len=BP-1.
RETURN
END
SUBROUTINE LININT(X1,X2,Y1,Y2,YLOC,YLOCAL)
IF(X1.EQ.X2) THEN
   YLOCAL=(Y2-Y1)/2.
ENDIF
SLOPE = (Y2-Y1)/(X2-X1)
YLOCAL = SLOPE*(XLOCAL-X2) + Y2
CONTINUE
RETURN
END
SUBROUTINE MARCH (NMA_, NP, Y, YSTP, F )

DIMENSION F(NP) ,Y(NP)
DIMENSION ySTP (NMAX)
DIMENSION INDEX ( 40 ), FS[40]

C bl if the areas ere bala.
and then the shock

KOUNT = 0
YEND = Y(NP)

CONTINUE
DO IND'=1,40
INDEX(IND') = 0,
ENDO

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

I_r(YS.GT:0.(YSTP._YS)):YS-(I) }

C iniNse_._. any
YSHK = YS
CALL SBKPT(NMA3 NP,Y,YSTP,F, INDEX, FS, YS, 3 )

 Else
C ,OYO
C Y xe

Y2 = YS
BIG = O.
DO ITEST-INDEX(INSCT)+I, NP
IF(YS.GT.Y(ITEST) .AND. ABS(YS-Y(ITEST)).GT.BIG}
Yl = Y(ITEST)
BIG = ABS(YS-Y(ITEST))
ENDIF
IF(Y(ITEST).GE.YS) GOTO 300
300 CONTINUE

NC = 500
DO 320 IC=I,NC
YS = O.5*(Y2+Y1)
CALL POINT(NP, Y, r, INDEX, FS, YS, INSCT )
DO II-INSCT, I ,-I
IF(INDEX (II) . LE . I TEST) THEN
INSCT - II
GOTO 310
ENDIF
ENDDO
310 CONTINUE
IS-INDEX(INSCT)
IE-INDEX(INSCT )
CALL AREA (NP ,Y, F, YS, FS,IS,IE,ILATO)
IF(IFSATO.EQ.0.0 ) GOTO 400
IF(ILAT0 .EQ. 0) THEN
Y2 = YS
ELSE
Y1 = YS
ENDIF
320 CONTINUE
WRITE(*,*) 'After ' ,NO ,'_tep_of_bisection'
ENDIF
GOTO 400
ENDIF
ENDIF
C 400 CONTINUE
IF(INSCT.LE.I AND. YS.GE.YEND}
RETURN
FDIS = FS(INSCT)-FS(1)
IF ( FLOAT(INDEX(INSCT) ) INDEX(1) ).EQ.0.0 ] then
WRITE(15,*) 'SDT: ZERO DIVISION A_Oq_T TO HAPPEN in MARCH'
ELSE
DF = FDIS/FLOAT(INDEX(INSCT) ) INDEX(1) )
ENDIF
F(IS(1) ) = FS(1)
Y(IS(1) ) = YS
IS = INDEX(1) + 1
DO 450 I = IS,INDEX(INSCT)
Y(I) = YS
Y(I) = Y(I-1) + DF
450 CONTINUE
IF(EQUANT.EQ.20) THEN
WRITE(*,*)'"EQUANT",'+
RETURN
ELSE
EQUANT - EQUANT+1
GOTO 100
ENDIF
END
SUBROUTINE SIMPSON(X,F,N,X0,X1,SM)
DIMENSION X(N),F(N)
C
K = 1 + n/2
SM = 0.
DO 10 I=1,N
   ODD = ODD + 2.*F(I+I)
   EVEN = EVEN + 4.*F(I)
10 CONTINUE
SM = F(1) + ODD + EVEN + F(N)
SM = SM*(X1-X0)/(6.*FLOAT(N))
RETURN
END
Appendix B

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

Dr. Samuel Cheung
Date: Dec., 1993
This subroutine reads a surface grid in airfoil
sections and formats it to produce
a surface grid of axysymmetric cross-sections.

Date: Dec., 1993 Version 3.0
read input geometry

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

NSEC = # of sections (streamwise stations) of the new grid
NPTS = # of pts in the circumferential direction (MUST be odd)
NWT= # streamwise stations
FAC = the first grid spacing is DIST
ELE = X leading edge
AERING = 0, arrow wing
ETIP = number of points in the wing-around direction on one surface
NC = number of cut in the spanwise direction
NCU = number of points in the upper part of the wing
NCB = number of points in the lower part of the wing

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

CALL WING

Setup distribution of cross-sections to be obtained (solid)
TIP=Z(Z+1)-CHORD(Z+1)
ITE=Z(4)+CHORD(4)
IF(ITE .GE. NWT) THEN
X1 = X2
AERING = -1.
ELSE
X1 = X2
AERING = 1.
ENDIF
XXX = MIN(X1, XXT)
DO 100 J=1, NSEC
DIST(J) = (X1-X2)/(FLOAT(J-1)/FLOAT(NSEC-1))
100 CONTINUE

The nose of the wing
DO 187 J=1, NPTS
YOUT(J,K) = YNEW(K)
YOUT(J,K) = YOUT(J,K) - ZDST(K)
187 CONTINUE

Begin main loop for each X-section
DO 1000 J=1, NSEC

LOCAL-EDIST(L)

Redistribute the points from spanwise out to streamwise out.
The output EDIST and YNEW are from the root to the tip, therefore
utilizes the lower surface, used to re-arrange the arguments.
The output (EDIST,YNEW) in both surfaces have ETIP # of pts in
the circumferential direction, their last point have the same physical
value for both surfaces.

Do the lower surface
CALL REDIST(LOCAL, ETIP, FAC, 1)
DO 300 K=1, ETIP
LOCAL = LOCAL - YOUT(J,K)
IF(LOCAL .LE. YNEW(K)) THEN
LOCAL = IDIST(K)
300 CONTINUE

Do the upper surface
CALL REDIST(LOCAL, ETIP, FAC, 2)
DO 400 K=ETIP-1, NPTS
LOCAL = LOCAL + YOUT(J,K)
IF(LOCAL .LE. YNEW(K)) THEN
LOCAL = IDIST(K)
400 CONTINUE

For the computational grid of UPGRID code
the zero has to have two different pts in same
physical location, such that (Y1,Y2)=(Y2,Y1) here
the calculation divided into upper and lower parts.

For safety sake, set ETIP=2
IF(LOCAL .LE. YNEW) GOTO 500
DO 500 K=1, ETIP-1
E1 = ETIP-1
E2 = ETIP
500 CONTINUE

*England model
IF(ABS(YOUT(J,K1)-YOUT(J,K2)) .LE. 1.0E-4) THEN
YOUT(J,K1) = YOUT(J,K2)
100 CONTINUE

*Pet-H
IF(ABS(YOUT(J,K1)-YOUT(J,K2)) .LE. 1.0E-2) THEN
YOUT(J,K1) = YOUT(J,K2)
110 CONTINUE

Proceed to next x coordinate

Write out new surfgrid in plotid format
EN=1
WRITE(50),NPTS, EW, NSEC
DO 1334 I=1, NSEC
  WRITE(50),(XOUT(L,E),X=1,NPTS),
  (YOUT(L,E),X=1,NPTS),
  (ZOUT(L,E),X=1,NPTS)
1334 CONTINUE
Write out original database in plotid format

NI=MH+1
WRITE(11) ((XBASE(I,1,N),I=1,N), (XBASE(I,2,N),I=M+1,N+1-1),
  (YBASE(I,1,N),I=1,N), (YBASE(I,2,N),I=M+1,N+1-1),
  (ZBASE(I,1,N),I=1,N), (ZBASE(I,2,N),I=M+1,N+1-1),
  (N+1,NC))

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

Read the nacelles grid and combine the nacelles with the
wing-body grid.
CALL WGRID

CLOSE(10)
CLOSE(30)
STOP
END
**SUBROUTINE ADDGRID(NPL1,NPL2,NPL3,NPL4,NSEC,RDIM)**

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

**PARAMETER (MAX=100)**

**DIMENSION YTEMP(MAX).XTEMP(MAX)**

**DIMENSION X(NPL1,Y(NP1,NPL2)).Y(NP1,NPL2)**

**IF(MAX.LE.NPI) THEN**

**WRITE('*,*)' SUB ADDGRID : MAX is less than NPI**

**STOP**

**ENDIF**

**IF(NPL2.GE.NPL2) THEN**

**WRITE('*,*)' No plane is added in the streamwise direction**

**STOP**

**ENDIF**

**C**

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

**XI = X(NPL1)**

**XJ = X(NPL2)**

**XI = 0.5*(X(NPL1)+X(NPL2))**

**DO 10 J=1,NPL2**

**XI = Y(NP1,J)**

**II = J**

**ZI = Y(NP1,J)**

**IIZ = Z(NP1,J)**

**CALL LINFIX(XI,X2,Y1,Y2,XX,YY)**

**CALL LINFIX(Y1,Y2,Z1,Z2,YY,ZZ)**

**YTEMP(I) = YY**

**XTEMP(I) = ZZ**

**CONTINUE**

**C**

**Re-number the last station**

**NSEC  = NSEC+1**

**DO 10 L = NSEC,NPL2+1,-1**

**X(L-1) = X(L)**

**DO 10 J=1,NPL2**

**X(L-1,J) = X(L,J)**

**CONTINUE**

**C**

**Put the temporary array in the grid**

**DO 10 J=1,NPL2**

**X(NPL2) = XJ**

**Y(NPL2,J) = YTEMP(J)**

**Z(NPL2,J) = ZTEMP(J)**

**CONTINUE**

**RETURN**

**END**
SUBROUTINE CIRCLES(X,EX,EMAX,Y,EMAX,R,RFL,ANCOR)

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

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

Look for the center of the fillet circle (YC,SC)

TA=X(EX)
FA=Y(EX)
FA=FA+RFL
 BB=(Y-A)/B
 R=B

IF(TA+R<.R2) THEN
  WRITE(15,'*')"Determinant is less than 0.,"DET
  GOTO 200
ENDIF

Y1 = (-B+SORT(DET))/2.,A)
 Y2 = (B-SORT(DET))/2.,A)

IF(Y1.Y2) THEN
  SC1 = (SCY+2*ANCOR)+(SCY+2*ANCOR)
  YC=SC1
  ENDIF

ELSE
  SC2
  YC=SC2
  ENDIF

IF(YC+R<.R2) THEN
  WRITE(15,'*')"Determinant is less than 0.,"DET
  GOTO 200
ENDIF

Find the total arc length given

T0PARC=0.

DO 50 E=EX,EX-1
  T0PARC=T0PARC+
  SORT((Y(X-I+1)-Y(X))**2+((X(X-I+1)-X(X))**2)

  CONTINUE

  ARC=T0PARC/2.

  ENDIF

Find the arc length between two points.

ARC = T0PARC/FLIGHT(EE-EX)

IF(YC+R<.R2) THEN
  WRITE(15,'*')"Determinant is less than 0.,"DET
  GOTO 200
ENDIF

IF(YC+R<.R2) THEN
  WRITE(15,'*')"Determinant is less than 0.,"DET
  GOTO 200
ENDIF

Find the coordinates for each point

DO 100 E=EX,EX-1
  TA=X(E)
  FA=Y(E)
  FB=FC

  SCY+TA-YB
  SCY+YB
  BB=(Y-A)/B
  R=BY

  A = 1.+R**2
  B = 2.*BY - 2.*FB - BB/R**2
  C = BB*BYB+BY + (BB/2.*SS)**2 - BB*BYB - RFL**2

  DET=A-B**4,-A**4 THEN
    WRITE(15,'*')"Determinant is less than 0.,"DET
    GOTO 200
  ENDIF

  YC1 = (-B+SORT(DET))/2.,A)
  YC2 = (B-SORT(DET))/2.,A)

  IF(YC1.YC2) THEN
    SC1 = (SCY+2*ANCOR)+(SCY+2*ANCOR)
    YC=SC1
    ENDIF

  ELSE
    SC2
    YC=SC2
    ENDIF

  ENDIF

100 CONTINUE

200 RETURN
END
SOURCE TEXT

SUBROUTINE CEPLINE(N,NX,NY,NW,THEN,THEN)

C
PARAMETER (NMAX=500)
C
REAL X(N), Y(N), Z(NMAX), NW(NMAX)
C
C THIS SUBROUTINE PRODUCES A MONOTONE CUBIC SPLINE INTERPOLANT
C FROM THE DATA (X(I),Y(I)) I=1,...,N AND COMPUTES VALUES AT
C THE NEW POINTS THEN(I), I=1,...,NW.  THESE ARE RETURNED IN
C ARRAYS Z(NMAX), NW(NMAX).  THE ALGORITHM USED IS THAT OUTLINED BY
C
C WRITTEN BY JEFF CORDOVA 16/3/86
C
REAL D(NMAX), DEL(NMAX), R(NMAX)
C
C MESH SPACING AND FIRST DIVIDED DIFFERENCE
DO 100 I=1,N-1
N1 = X(I+1) - X(I)
100 CONTINUE
C
DO 200 I=1,N-1
DEL(I) = (Y(I+1) - Y(I)) / N1
200 CONTINUE
C
C SPLINE COEFFICIENTS
C *** LINEAR INTERPOLATION FOR N=2 CASE ***
IF (N .LE. 2) THEN
D(N) = DEL(N)
D(N-1) = DEL(N-1)
GO TO 399
ELSEIF (PCBST(D(N),DEL(N)) .LT. 0.) THEN
D(N) = 0.
ENDIF

C *** MONOTONE SPLINE COEFFICIENTS FOR N > 2 CASE ***
C
C FIRST BOUNDARY POINT (USE THREE POINT FORMULA ALTHOUGH TO BE
C SHAPE PRESERVING)
BSIN = R(1) + R(2)
W1 = (R(1) + BSIN) / BSIN
W2 = (R(1) / BSIN
D(1) = W1*DEL(1) + W2*DEL(2)
IF (PCBST(D(1),DEL(1)) .LE. 0.) THEN
D(1) = 0.
ELSEIF (PCBST(D(1),DEL(1)) .LT. 0.) THEN
D(N-1) = 3.*DEL(N-1)
ENDIF
C
C INTERIOR POINTS (PIEGELS MODIFICATION OF BUTLAND FORMULA)
CONST = 1. / 3.
DO 300 I=1,N-1
TOP = DEL(N-1) - DEL(1)
TOP = TOP - 5.* (1. + SIGN(1.,TOP))
ALPHA = CONST - (R(I-1) + 2.*R(I)) / (R(I-1) + R(I))
BOT = ALPHA - DEL(I) - (1.-ALPHA) - DEL(I-1) + 1.E-20
D(I) = TOP / BOT
300 CONTINUE
C
C LAST BOUNDARY POINT (USE THREE POINT FORMULA ADJUSTED TO BE
C SHAPE PRESERVING)
BSIN = R(N-2) + R(N-1)
W1 = (R(N-1) / BSIN
W2 = (R(N-2) + BSIN) / BSIN
D(N) = W1*DEL(N-2) + W2*DEL(N-1)
IF (PCBST(D(N),DEL(N-1)) .LE. 0.) THEN
D(N) = 0.
ELSEIF (PCBST(D(N),DEL(N-1)) .LT. 0.) THEN
D(N) = 3.*DEL(N-1)
ENDIF
C
C END
C
C SPLINE EVALUATION
C
DO 400 J=1,N-1
CTREE = (D(J) + D(J+1) - 3.*DEL(J)) / (R(J)+R(J))
CTWO = (3.*DEL(J) - 2.*D(J) - D(J+1)) / R(J)
CRAY C = ISECBF(IXENN,NK,NX,NW,1,E(34)) 10C CRAY
C = ISECBF(IXENN,NK,NX,NW,1.E(34))
400 CONTINUE
C
C RETURN
END
FUNCTION PCBST(ARG1,ARG2)

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

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

RETURN

END
FUNCTION ISRCHGE(N,IX,INCR,FRTAEG)  
DIMENSION I(*)  
I = (N.LE.0) THEN  
ISRCHGE = 0  
RETURN  
ELSE  
IT = 1 + (N-1) * INCR  
ISRCHGE = 1  
DO 10 I=1,IT,INCR  
IF(X(I).GE.FTARGET) GOTO 11  
ISRCHGE = ISRCHGE + 1  
10 CONTINUE  
11 CONTINUE  
ENDF  
RETURN  
END
SUBROUTINE CURTER

DIMENSION TWF(NPI),ZWF(NPI)

LJ = L - 1
NFUS = EDM - NPTS
NBOT = NFUS/2 + 1
NTOP = NFUS/2 + 1

DO 900 LL=1,LJ

Note: I am leaving the nose and the wake alone

IF((ABY((LL,NBOT))-(LL,EDM-NTOP+1))/.LE.1.E-7 THEN
  RETURN
ENDIF

DO 90 E=1,NBOT
ZINT(K) = Z(LL,E)
CVN(K) = Y(LL,E)
CONTINUE

10 CONTINUE
FSP = SQRT((Z(LL,NBOT)-Z(LL,NBOT+1))**2 +
  (Y(LL,NBOT)-Y(LL,NBOT+1))**2 )

CALL DISTARC(VINT,ZINT,NBOT,TWF,ZWF,NBOT,FSP,1)
DO 80 E=1,NTOP
Z(LL,E) = ZWF(K)
Y(LL,E) = TWF(K)
CONTINUE

80 CONTINUE

DO 100 K=1,NTOP

900 CONTINUE

RETURN

END
**Source Program**: samgrid.f

**Source Text**

**LINE #** | **SOURCE TEXT**
--- | ---
486 | SUBROUTINE DISTARC(X,Y,N,NEW,THNEW,NNEW,NGS,IFLAT)
488 | !
489 | !
490 | !
491 | !
492 | !
493 | !
494 | !
495 | !
496 | !
497 | !
498 | !
499 | !
500 | !
501 | !
502 | !
503 | !
504 | !
505 | !
506 | !
507 | !
508 | !
509 | !
510 | !
511 | !
512 | !
513 | !
514 | !
515 | !
516 | !
517 | !
518 | !
519 | !
520 | !
521 | !
522 | !
523 | !
524 | !
525 | !
526 | !
527 | !
528 | !
529 | !
530 | !
531 | !
532 | !
533 | !
534 | !
535 | !
536 | !
537 | !
538 | !
539 | !
540 | !
541 | !
542 | !
543 | !
544 | !
545 | !
546 | !
547 | !
548 | !
549 | !
550 | !
551 | !
552 | !
553 | !
554 | !
555 | !
556 | !
557 | !
558 | !
559 | !
560 | !
561 | !
562 | !
563 | !
564 | !
565 | !
566 | !
567 | !

This is a subroutine in a FORTRAN program named `samgrid.f`. The subroutine is named `DISTARC` and takes several arguments: `X,Y,N,NEW,THNEW,NNEW,NGS,IFLAT`. It appears to be involved in some form of data redistribution, possibly for a grid or network of points. The specific details of the function are not fully visible, but it seems to involve calculations related to distances and arc lengths on a grid. The code structure suggests it might be part of a larger program, possibly for handling grid data or network analysis. The exact purpose of the subroutine would require a deeper understanding of the context in which this code is used.
**Subroutine** DISTRI(FANG, FPCS, $, IFINE)

**Parameters** (MAX=400)

**Dimension** ($MAX, DCN(MAX)

C...Calculating the stretching function $ when given FANG, and the number of points FPCS

C...If IFINE=1, distribution is consistent at outer grid

IF(MAX.LE.FPCS) THEN
  WRITE(*,*)'SUB-DISTRI : MAX is less than FPCS'
  STOP
ENDIF

IF(FPCS.EQ.1) THEN
  $ = 0.
  GOTO 40
ENDIF

C

D$ = FANG
D$ = FPCS-1
$ = 1./FLOAT(D$)
R$ = 1.5
CALL GRBET(D$, R$, 0.0001, 100, R$)
CALL PS1(FPCS, R$, $, R$)

DO 37 K = 1, FPCS - 1
  S(K) = 1. - DUM(K)
37 CONTINUE

DO 38 K = E_S, FPCS - 1
  S(K) = 1. - DUM(K)
38 CONTINUE

IF (IFINE.EQ.1) THEN
  DO 37 K = 1, FPCS
    S(K) = 1. - DUM(K)
37 CONTINUE

ENDIF

RETURN
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)

ZLE = ZBASE(I,I,1)
DO 200 K = 1,NC
IF(ZBASE(I,I,K).GT.XBK) THEN
   X1 = XBASE(I,I,K-1)
   X2 = XBASE(I,I,K)
   Z1 = ZBASE(I,I,K-1)
   Z2 = ZBASE(1,I,K)
   CALL LININT(X1,X2,Z1,Z2,XBK,ZBK)
   GOTO 210
ENDIF
200 CONTINUE
210 CONTINUE
IF(_BASE(1,1,E).GT.ZB[) GO TO 700
CALL LININT(_BASE(1,1,E),_BASE(1,1,E),_BASE(1,1,2),_BASE(1,1,2),_BASE(1,1,2),_BASE(1,1,2))
XL_OLD = XLE(I,E)
XTL = XBASE(NL,E)
DO 280 I = 1,NL
   IF(XBASE(I,E)-XL_OLD .GT. XLE(I,E)) THEN
      E = (1.*XLE(I,E) - E*XLE(I,E))/(F+E)
   ELSE
      E = XLE(I,E)
   ENDIF
280 CONTINUE
IF(_BASE(1,2,E).GT.ZB[) GO TO 700
CALL LININT(_BASE(1,2,E),_BASE(1,2,E),_BASE(1,2,2),_BASE(1,2,2),_BASE(1,2,2),_BASE(1,2,2))
XBASE(1,2,E) = (F*XTL + E*XLE)/(F+E)
300 CONTINUE
500 CONTINUE
700 CONTINUE
RETURN
END
SUBROUTINE EQSPACE
INCLUDE "sysgrid.com"
L2 = L-1
DO 130 L=1,L2
   XIN(LL) = X(LL)
   DO 120 K=1,EDIM
      ZIN(LL,K) = Z(LL,K)
      YIN(LL,K) = Y(LL,K)
   CONTINUE
   120 CONTINUE
130 CONTINUE
135 XTOT = X(L2)-X(1)
136 DX = XTOT/FLOAT(L2-1)
137 DO 140 J=1,L2
   X(JL) = X(JL-1)+DX
138 DO 150 K=1,EDIM
      IF(ABS(XIN(KL)-X(JL)) .LE. 1.E-7) THEN
         DO 140 _=1,EDIM
            ZIN(KL,_) = ZIN(3L,_) + YIN(KL,_) - YIN(3L, _)
         CONTINUE
         GOTO 1G0
      ENDIF
      IF(XIN(KL) .GT._T(3L)) THEN
         DO 145 _=1,EDIM
            X1 = XIN(KL-1)
            V1 = YIN(KL-1,_) + ZIN(KL-1, _)
            Z1 = ZIN(KL-1, _)
            X2 = XIN(KL)
            Y2 = YIN(KL,_) + ZIN(KL, _)
            Z2 = ZIN(KL, _)
            XX = ZIN(XL)
            CALL LININT(X1,X2,Y1,Y2,XX,YY)
            CALL LININT(X1,X2,Y1,Y2,ZZ)
            Y(JL,_) = YY
            Z(JL,_) = ZZ
         CONTINUE
         GOTO 1G0
      ENDIF
   CONTINUE
   150 CONTINUE
140 CONTINUE
RETURN
END
**SAMGRID.F**

The subroutine `FILET` takes the wing-fuselage intersection points, `(Y(K),E(K))`, for each segment to be interpolated between `(Y(K),E(K))` and `(Y(K+1),E(K+1))`. It also finds the intersection points for the top and bottom parts of the aircraft. The routine `CIRCLE` is called to replace the segment by a circle with radius `RFIL`. The routine `DISTARC` is used to connect the points of each segment to form the complete wing-fuselage model.
SUBROUTINE PZI(LI,TBETA,DET,Z)

COMPUTES NORMALIZED NORMAL DISTANCE, Z(I)

PARAMETER (MAX=400)

DIMENSION Z(MAX)

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

IF(TBETA.EQ.1.) THEN
    DO 10 I=1,LI
        Z(I)=0.
        CONTINUE
ELSE
    ETA=(L-1.)*DET
    RR=(TBETA-1.)/(TBETA-1.)
    RR=1.-ETA
    ETA=RR*ETA
    Z(I)=(TBETA-1.)*(RR-ETA)/(ETA-1.)
    CONTINUE
ENDIF
END
SUBROUTINE GRET(DFM,NPT,FPCC,ICC,BETA)

C BISECTION METHOD USED TO DETERMINE STRETCHING PARAMETER, BETA,
C WHICH GIVES DESIRED GY AT THE WALL

PARAMETER (MAX=400)
DIMENSION I(MAX)

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

ICC=ICC
FPCC=FPCC*DFM
BETA=BETA
Z1=DFM
DET=1./NPT
BR=1
IF=11
ITCC=ICC/10
GO TO 12
IF=11
IF(ICC.GT.11) GO TO 15

DO I0 I1,ICC
BF=BETA
BETA=0.5*(BF+BETI)
CALL FZI(BF,DET,Z)
FF-Z(2)
IF(FF.LT.0.) GO TO 15
BETI=2.*BETI
I0 CONTINUE

CONTINUE

DO I1,NIT
CALL FZI(2,BETA,DET,Z)
FR-FZI(BF,BETI,DET,Z)
ELSE
FF-FZI(BF,BETI,DET,Z)
END IF

IF(ABS(FF).GT.FPCC) GO TO 4
CONTINUE

WRITE(4,100) BETA,FR
100 FORMAT(10,H5,18.6) NIT=3G13.6

BETA=BETI-1.
BF=MIF+BF-1.
BETI-BF-1.
RETURN
END
**SOURCE TEXT**

SUBROUTINE LININT(X1,X2,Y1,Y2,YLOCAL)
  C This subroutine linearly interpolates YLOCAL when given (X1,Y1) & (X2,Y2)
  IF(XABS(X1-X2) LE.1.E-7) THEN
  YLOCAL = (Y2+Y1)/2.
  ELSE
  SLOPE = (Y2-Y1)/(X2-X1)
  YLOCAL = SLOPE*(XLOCAL-X2) + Y2
  ENDIF
  CONTINUE
  RETURN
END
UNITX™ FORTRAN Program

SOURCE PROGRAM samgrid.f

INCLUDE "sm10.in.c"

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

DIMENSION XENG(4),YENG(4),ZENG(4),INAC(NPI)

COMMON /ENG/ XENG(2,NPI),YENG(2,NPI),ZENG(2,NPI)

COORD = /ENG/ XENG(2,NPI),YENG(2,NPI),ZENG(2,NPI)

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

IPRT = 0 no writing out

MC = 40

IPRT = NPTS+MC

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

NPTS = NPTS + MC

END IF

NPRM = (NPTS+1)/2

NPAIR = (NPRM)+1/2

INAC = NAC(JK)

NACP = NAC(JK)

DO 100 L=NNUM(1),LNUM(2)

C Integrate the points of the nacelle at wout

DO 100 L=NUN(J),LNUM(1),1,1

IFABS(XENG(J,N),ZOUT(L,1)) LE 1.E-7 THEN

DI = XENG(J,N,1)

END IF

GO TO 105

ELSEIF(XENG(J,N,GT.ZOUT(L,1)) THEN

DI = XENG(J,N,L)

END IF

GO TO 105

C Do = INACP

X1 = XENG(J,N,1)

Y1 = YENG(J,N,L)

Z1 = ZENG(J,N,L)

X2 = XENG(J,N,L)

Y2 = YENG(J,N,L)

Z2 = ZENG(J,N,L)

XX = COUNT(L,1)

CALL LININT(X1,X2,Y1,Y2,Z1,Z2,XX)

YAC(R) = Y1

ZNAC(K) = Z1

END IF

GO TO 105

105 CONTINUE

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

CONTINUE

COUNT = 0

DO 100 K=1,NPRM

IF(YOUT(I,K) .GT. YAC(R) ) .OR. ZOUT(I,K) .GE. ZNAC(K) THEN

COUNT = COUNT + 1

NPAIR(K,ROUNT) = XI

NPAIR(K+1,ROUNT) = XI

END IF

100 CONTINUE

CONTINUE

C Nacelle totally under the wing, INTRAIL = 0

CONTINUE

INTRAIL = 0

IF(COUNT.EQ.0) GOTO 415

IF(ZOUT(L,NPAIR(I,K)).GT.ZNAC(K)) INTRAIL=1

IF(ZOUT(L,NPAIR(I,K)).GT.ZNAC(K)) INTRAIL=2

WRITE(*,'(15,T12)') 'INTRAIL = ', INTRAIL

IF(INTRAIL .LT. 2) THEN

CALL LININT(X1,X2,Y1,Y2,Z1,Z2,XX)

YAC(R) = Y1

ZNAC(R) = Z1

ENDIF

100 CONTINUE

END IF

CONTINUE

C Nacelle totally under the wing, INTRAIL = 1

100 CONTINUE

INTRAIL = 1

IF(COUNT.EQ.0) GOTO 415

IF(ZOUT(L,NPAIR(I,K)).GT.ZNAC(K)) INTRAIL=1

IF(ZOUT(L,NPAIR(I,K)).GT.ZNAC(K)) INTRAIL=2

WRITE(*,'(15,T12)') 'INTRAIL = ', INTRAIL

IF(INTRAIL .LT. 2) THEN

CALL LININT(X1,X2,Y1,Y2,Z1,Z2,XX)

YAC(R) = Y1

ZNAC(R) = Z1

ENDIF

100 CONTINUE

END IF

CONTINUE

C Nacelle totally under the wing, INTRAIL = 2

100 CONTINUE

INTRAIL = 2

IF(COUNT.EQ.0) GOTO 415

IF(ZOUT(L,NPAIR(I,K)).GT.ZNAC(K)) INTRAIL=1

IF(ZOUT(L,NPAIR(I,K)).GT.ZNAC(K)) INTRAIL=2

WRITE(*,'(15,T12)') 'INTRAIL = ', INTRAIL

IF(INTRAIL .LT. 2) THEN

CALL LININT(X1,X2,Y1,Y2,Z1,Z2,XX)

YAC(R) = Y1

ZNAC(R) = Z1

ENDIF

100 CONTINUE

END IF

CONTINUE

C We are in the wake region of the trailing edge

GOTO 415

ELSE

C We are in the wake region of the trailing edge

WRITE(*,'(15,T12)') 'We are in the wake region'

NTERM = 1

CONTINUE

C Obtain all points under the wake line

DO 180 I=1,KOUNT

IF(ZOUT(I,NPAIR(1,K))).GT.ZNAC(NTERM) THEN

IN = I

ENDIF

IF(COUNT(1) .LE. I .AND. COUNT(1) .GT. XI) THEN

Z2 = ZENG(JN,LN)

ZOUT(I,NPAIR(1,K))-ZOUT(I,NPAIR(1,K))

YOUT(I,NPAIR(1,K))-YOUT(I,NPAIR(1,K))

ZENG(JN,LN)

ZENG(JN,LN)

END IF

180 CONTINUE

CONTINUE

NTERM = NTERM + 1

END IF

GOTO 185

END IF

185 CONTINUE

NS = NTERM

IF(NS .LT. 11) THEN

CALL LININT(YAC(HS-1),YAC(HS),ZNAC(HS-1),ZNAC(HS),YAC(HS),

YOUT(I,NPAIR(1,K)),1,1)

YVS = YOUT(I,NPAIR(1,K),1,1)

ZVS = ZNAC(HS)

ENDIF

END IF

C END OF THE LISTING
**SOURCE TEXT**

```
1 500 DO E=1,NFRI
     YINT(E) = YOUT(E)
     ENDDO

500 CONTINUE
```

**SOURCE PROGRAM**

```
500 DO E=1,NFRI
     YINT(E) = YOUT(E)
     ENDDO
```
TF(ZOUT(L,K) .LE.ZNAC(NFRIS)) THEN
  NN2 = K
  CALL LININT(ZOUT(L,NN2),ZOUT(L,NN2-1),YOUT(L,NN2),
              YY2)
  ZZ2 = ZNAC(NFRIS)
  GOTO 605
END IF
600 CONTINUE
C Store the wingtip grid
DO 610 K=NPW,NN1
  YWNG(K) = YOUT(L,K)
  ZWNG(K) = ZOUT(L,K)
610 CONTINUE
C Redistribute the points above the sailplane
  WI = NN2-NN1+1
  SWI = WI
  SWF = NN2
  CALL LININT(ZOUT(L,NN2),ZOUT(L,NN2-1),YOUT(L,NN2),
              YY2)
  ZZ2 = ZNAC(NFRIS)
  GOTO &05
ENDIF
600 CONTINUE
C Store the wingroot grid
DO 640 K=NP2,NPTS
  YWNG(K) = YOUT(L,K)
  ZWNG(K) = ZOUT(L,K)
640 CONTINUE
C Redistribute the points above the sailplane
  WI = NN2-NN1+1
  SWI = WI
  SWF = NN2
  CALL LININT(ZOUT(L,NN2),ZOUT(L,NN2-1),YOUT(L,NN2),
              YY2)
  ZZ2 = ZNAC(NFRIS)
  GOTO &05
ENDIF
600 CONTINUE
C Store the wingtip grid
DO 650 K=NPW,NN1
  YWNG(K) = YOUT(L,K)
  ZWNG(K) = ZOUT(L,K)
650 CONTINUE
C Redistribute the points above the sailplane
  WI = NN2-NN1+1
  SWI = WI
  SWF = NN2
  CALL LININT(ZOUT(L,NN2),ZOUT(L,NN2-1),YOUT(L,NN2),
              YY2)
  ZZ2 = ZNAC(NFRIS)
  GOTO &05
ENDIF
600 CONTINUE
C Store the wingroot grid
DO 660 K=NP2,NPTS
  YWNG(K) = YOUT(L,K)
  ZWNG(K) = ZOUT(L,K)
660 CONTINUE
C Redistribute the points above the sailplane
  WI = NN2-NN1+1
  SWI = WI
  SWF = NN2
  CALL LININT(ZOUT(L,NN2),ZOUT(L,NN2-1),YOUT(L,NN2),
              YY2)
  ZZ2 = ZNAC(NFRIS)
  GOTO &05
ENDIF
600 CONTINUE
C Return
RETURN
END
SUBROUTINE MACGRID

INCLUDE "samgrid.com"

! This subroutine read the nacelle grid and combine it with the wing grid.
COMMON /ENG/ XENG(2,NIPI), YENG(2,NIPI), ZENG(2,NIPI,NIPI)
&/MAC/ XMAC(2), YMAC(2), ZMAC(2), NMAC(2), NMACP(2)
COMMON /MACP/ XMACP(2), YMACP(2), ZMACP(2), NMACP(2), NMACP(2)
COMMON /MACP1/ XMACP1(2), YMACP1(2), ZMACP1(2), NMACP1(2), NMACP1(2)

C (XENG,YENG,ZENG) Coordinates of engine
C NMAC(*) Number of stations in nacelle
C NMACP(*) Number of points in each station
C (1,*,*) inner nacelle, (2,*,*) outer nacelle
C
C Read the nacelles geometry:
C
C Inner nacelle geometry
CALL MACIN
XIN1 = XMAC(1)
YIN1 = YMAC(1)
ZIN1 = ZMAC(1)
DO 100 L=1,NMAC
XENG(1,L) = XMAC(L)
YENG(1,L) = YMAC(L)
ZENG(1,L) = ZMAC(L)
100 CONTINUE

C Outer nacelle geometry
CALL MACIN
XOUT1 = XMAC(2)
YOUT1 = YMAC(2)
ZOUT1 = ZMAC(2)
DO 100 L=1,NMACP
XENG(2,L) = XMACP(L)
YENG(2,L) = YMACP(L)
ZENG(2,L) = ZMACP(L)
100 CONTINUE

C In a case of 3 nacelles, three zone will be made.
C HENVING Station(s) will be added to the wing at inlet and outlet
C of the nacelles
C MOUNT Mount the nacelles under the wing and/or wake line
C JN = 1 inner nacelle
C JN = 2 outer nacelle
C JN = 3 third nacelle
C
C The first zone, only one nacelle appears
WRITE('***','zone 1')
JN = 1
CALL HENVING(JN,XIN1,YOUT1)
CALL MOUNT(JN,XIN1,1)
LNUM(1) = LNUM(1) + 1
LNUM(4) = LNUM(4) + 1
LNUM(5) = LNUM(5) + 1
46 C The second zone consists two nacelles appear
WRITE('***','zone 2')
JN = 2
CALL HENVING(JN,XIN1,YOUT1)
CALL MOUNT(JN,XIN1,0)
LNUM(1) = LNUM(1) + 1
LNUM(4) = LNUM(4) + 1
LNUM(5) = LNUM(5) + 1
46 C The third zone, only one nacelle appears
WRITE('***','zone 3')
JN = 3
CALL HENVING(JN,XIN1,YOUT1)
CALL MOUNT(JN,XIN1,1)
LNUM(1) = LNUM(1) + 1
LNUM(4) = LNUM(4) + 1
46 C RETURN
SUBROUTINE MEMNG(LNUM,XX,XX)
  include "afrlgd.co"
  DIMENSION XNANG(NPT,NPT),YNANG(NPT,NPT),ZNANG(NPT,NPT)
  DIMENSION LNAM(4)

  C Rewrite the coordinate of the wing
  DO 10 L=1,NSEC
    DO 20 K=1,NPTS
      XNANG(L,K) = XOUT(L,K)
      YNANG(L,K) = YOUT(L,K)
      ZNANG(L,K) = ZOUT(L,K)
    20 CONTINUE
  10 CONTINUE

  C Find out where the x-location of start and end of ncell
  INSTR = XX
  IXEND = XX

  C Add two stations in the wing, these two stations lie exactly on
  INSTR and IXEND.
  DO 30 N=1,1
    IF(MEQ.EQ.1) X1=INSTRT
    IF(MEQ.EQ.2) X2=IXEND
    DO 40 K=1,NPTS
      XNANG(K,1) = X1
    40 END
    GO TO 50
  30 CONTINUE

  C Create an extra station in the wing
  IF(XNANG(N1,1).GE.XT) THEN
    X(N1) = XX
    LNAM(N1) = LW
    GO TO 50
  40 CONTINUE

  C If we are in wake, make sure top pts intersect the bottom pts
  DO 24 E1=1,LH,NSEC
    DO 25 E2=1,NPTS
      IF (EQ.EQ.1) Y1=YT1
      IF (EQ.EQ.2) Y2=YT2
      IF (EQ.EQ.3) Y3=YT3
      CALL LIMIT(XX,X1,X2,Y1,Y2,Y3)!
      V(XX,E1) = Y
      Z(XX,E1) = Z
    25 CONTINUE
    GO TO 24
  24 CONTINUE

  20 CONTINUE

  C Put the rest of the station into (X,Y,Z)
  DO 35 L=1,NSEC
    DO 36 E1=1,LH,NSEC
      X(E1) = XNANG(L,E1)
      Y(E1) = YNANG(L,E1)
      Z(E1) = ZNANG(L,E1)
    36 CONTINUE
  35 CONTINUE

  30 CONTINUE

  C Generate the node coordinate of the wing
  DO 45 L=1,LH,NSEC
    DO 46 E=1,NPTS
      X(L,E) = X(E+1,E)
      Y(L,E) = Y(E+1,E)
      Z(L,E) = Z(E+1,E)
    46 CONTINUE
  45 CONTINUE

  40 CONTINUE

  GO TO 55

  END

  50 CONTINUE

  55 CONTINUE

  DO 910 L=1,NSEC
    DO 910 E=1,NPTS
      XOUT(L,E) = XNANG(L,E)
      YOUT(L,E) = YNANG(L,E)
      ZOUT(L,E) = ZNANG(L,E)
  910 CONTINUE

  RETURN
END
**SOURCE PROGRAM**

```
LINE # | SOURCE TEXT
**------------------------**
1461 | include *sgrid.com*
1462 | DIMENSION D1(NP1),D2(NP1),S(NP1)
1463 | KTIP = (KDIM+1)/2
1464 | **From the nose to the leading edge**
1465 | DO 12 K=1,KDIM
1466 | X(K) = XIM(K)
1467 | Y(K) = YIM(K)
1468 | Z(K) = ZIM(K)
1469 | 12 CONTINUE
1470 | **Loop for all stations, from station 1 to M1**
1471 | DO 550 N=1,M1
1472 | L = N
1473 | **STORE the input to dummy array**
1474 | DO 31 K=1,NFP
1475 | TINT(K) = XIM(M,K)
1476 | ZINT(K) = ZIN(M,K)
1477 | 31 CONTINUE
1478 | **TREEP is the value of the first point of the wing**
1479 | **TREEP is the corresponding index of each station**
1480 | TREEP=TOOT(1,NFPT)
1481 | **OPTIONS**
1482 | Boeing Baseline Configuration
1483 | IF(TINT(K) .GE.YREF) THEN
1484 | YREF=1
1485 | GOTO 44
1486 | ENDIF
1487 | 44 CONTINUE
1488 | **Treep=MV/2 = KREFP**
1489 | **Laegley's Low-Boom Configuration**
1490 | YREF = 31
1491 | **Lower part of the nose (-Y to +YREF)**
1492 | ES=1
1493 | EE=KREF
1494 | EX=EX=EX=1
1495 | DO 74 E=EX,EE
1496 | EX = EX+1
e1771 | D1(E) = TINT(E)
1497 | D2(E) = ZINT(E)
1498 | 74 CONTINUE
1499 | **CALL DISTMC(D1,D2,EX,YNEW,EDIST,ETIP,FNPB0T,1)**
1500 | **From Y=REF to pos Y**
1501 | KY=REF
1502 | KY=KY=KY=1
1503 | DO 100 K=KY,KE
1504 | KE = KE+1
e1771 | D1(K) = TINT(K)
1505 | D2(K) = ZINT(K)
1506 | 100 CONTINUE
1507 | **CALL DISTMC(D1,D2,EX,YNEW,EDIST,ETIP,FNPB0T,0)**
1508 | **RETURN**
1509 | END
```
**Source Program**

**samgrid.f**

**Source Text**

```fortran
DO 50 I=2,NUL
   XI XBASE(I-1,IFLAT,NC-M+I)
   X2 XBASE(I,IFLAT,NC-M+I)
   Y1 YBASE(I-1,IFLAT,NC-M+I)
   Y2 YBASE(I,IFLAT,NC-M+I)
   Z1 ZBASE(I-1,IFLAT,NC-M+I)
   Z2 ZBASE(I,IFLAT,NC-M+I)
   CALL LININT(X1,X2,Y1,Y2,XLOCBL,YY)
   CALL LININT(X1,X2,Z1,Z2,XLOCY,ZZ)
   YINT(KT) = YY
   ZINT(KT) = ZZ
   GOTO 200

50 CONTINUE

DO 150 I = 2,NC
   XL = XLOCY-XBASE(I-1,IFLAT,NC)
   XR = XLOCY-XBASE(I,IFLAT,NC)
   IF(XL*XR.LT.0.) THEN
      XI = XBSE(I,IFLAT,NC-I)
      X2 = XBSE(I,IFLAT,NC+I)
      Y1 = YBASE(I,IFLAT,NC-I)
      Y2 = YBASE(I,IFLAT,NC+I)
      Z1 = ZBASE(I,IFLAT,NC-I)
      Z2 = ZBASE(I,IFLAT,NC+I)
      CALL LININT(X1,X2,Y1,Y2,XLOCY,YY)
      CALL LININT(X1,X2,Z1,Z2,XLOCY,ZZ)
      YINT(KT) = YY
      ZINT(KT) = ZZ
   ELSEIF ABS(XL*XR).LT.1.E-7 THEN
      IF(ABS(XL).LE.1.E-7) THEN
         CALL WINGWAKE(ILOCAL,ET,NUL,IFLAT)
      ELSE
         CALL WINGWAKE(ILOCAL,ET,NUL,IFLAT)
      ENDIF
   ENDIF
150 CONTINUE

DO 160 I = 1,N_L
   IT = KT+1
   XI = XT(I)
   X2 = XBSE(I-1,IFLAT,NC)
   Y1 = YBASE(I,IFLAT,NC)
   Y2 = YBASE(I+1,IFLAT,NC)
   Z1 = ZBASE(I,IFLAT,NC)
   Z2 = ZBASE(I+1,IFLAT,NC)
   CALL LININT(X1,X2,Y1,Y2,XLOCY,YY)
   CALL LININT(X1,X2,Z1,Z2,XLOCY,ZZ)
   YINT(KT) = YY
   ZINT(KT) = ZZ
   GOTO 200

160 CONTINUE
```

**Notes**

- The subroutine `RESIST` is used to redistribute the points from the spanwise cut to streamwise cut.
- When `CHESSP-SIMP` is used, the output is for `SIMP` code.
- The subroutine includes input parameters and dimension statements.
- The subroutine handles the leading edge and trailing edge of the wing.
- Streamwise distances between leading and trailing edges are calculated.
- Checks are made to ensure that the calculated points are within the valid range.
- If points are outside the valid range, a call to `WINGWAKE` is made.
- The subroutine includes calls to `LININT` for line interpolation.
YINT(KT) = YBASE(1,IFLAT,M-1)
ELSE
YINT(KT) = YBASE(1,IFLAT,M)
ENDIF
ZINT(KT) = ZBASE(1,IFLAT,M)
GOTO 160
ENDIF
CONTINUE

DO 220 KE=1,KT
Z(KE) = YINT(KE)
220 CONTINUE
DO 230 KE=1,KT
YINT(KE) = S(KE-KE-1)
230 CONTINUE
DO 240 KE=1,KT
S(KE) = ZINT(KE)
240 CONTINUE
DO 250 KE=1,KT
ZINT(KE) = S(KE-KE-1)
250 CONTINUE

OPTIONS :-
Distribute & coordinates (rove around direction) from root to
leading edge and back (xdist);
IT=1, grid points will cluster near the wing tip, =0 near the root.
For HESS, IT=0; For Boeing, IT=1
CALL DISTARC(ZINT,YINT,XT,ZDIST,YNEW,KT,PAC,ET)
IF(L,NE,NSEC)CALL CSPLINE(ZINT,YINT,XT,ZDIST,YNEW,ETIP)
RETURN
END
SUBROUTINE SUBTRACGRID(NPL1,X,Y,Z,NSEC,KDIM)

C This subroutine allows us to subtract a grid line NPL1 in
C streamwise section, and the new dimension is NSEC again.
C
DIMENSION X(NPL1),Y(NPL1),Z(NPL1)

C Remember the late stations
NSEC = NSEC-1

DO 10 L = NPL1,NSEC
   X(L) = X(L-1)
   Y(L) = Y(L-1,1)
   Z(L) = Z(L-1,1)
10 CONTINUE

10 CONTINUE

RETURN
END
SUBROUTINE TAIL(FAC1,FAC2)

DIMENSION S(NPI)
COMMON /RF/ DROOT,KTIP,ARCOER

KTI = (EDIM+1)/2
LE = L-1

PTI - (TOUT(NSEC,1)+TOUT(NSEC,NPTS))/2.
PT2 = (TIN(NF,NFP)+TIN(NF,1))/2.
X1 = X(L)
X2 = X(IN(NF))

L1 = L=1
L2 = L=1+(NF-M2)-1
DO 360 L=L1,L2
K=1+L-M2
4(L) = XIN(M)
END

C OPTIONS
C
C Design the let point (EF) will be the off fuselage side
C It is also the number of pts in the upper or lower fuselage, therefore
C It depends on the previous station.
C (no. of pts in this station) / (no. of pts in previous) =
C (radius of this station) / (radius of previous)
C
C (NPTS+1)/ABS((TIN(NF,NFP)+TIN(NF,1))/ABS((TIN(NK,N,1)-TIN(N-1,1)-TIN(N-1,NFP)))

F=EDIM-NPTS/2+1
EFR=EDIM-NPST/2

IF(RATIO.LT.6.5 OR RATIO.GE.1.1) THEN
IF(RATIO.FLT.(RFIL)/2) E=1
ENDIF

C Find EF the point at old grid where YIN(N,F,1)=NAKEEP
C and calculate the points from seg Y to Y at EF
DO 222 EF=1,EFP
IF(ABS(YIN(M,E)-NAKEEP).LE.1.E-7) THEN
ST=K
GOTO 223
ENDIF

IF(YIN(M,E).LE.1.E-7) THEN
YTI=K
Y1=TIN(M,K,1)
Y2=TIN(M,K,1)
Y3=SN(M,K,1)
Y4=SN(M,K,1)
TY=NAKEEP
CALL LININT(Y1,Y2,Y3,Y4,Y5)
ENDIF

222 CONTINUE

223 CONTINUE

DO 230 EF=1,EF
230 CONTINUE

C Find EF the point at old grid where YIN(N,F,1)=NAKEEP
C and calculate the points from seg Y to Y at EF
C
DO 240 EF=1,EF
Y(L,E)=YIN(Y)
Y3(L,E)=DIST(Y)
240 CONTINUE

C These are points in off fuselage side
C E=No. of pts in the off-fuselage side
E=EDIM-2*EF
KRN=KRN+1/2
RSPAN=ABS((2*(LE,EKIP)-LE,EK) )
IF(FAC3.LT.0.21) E=15
DFT=FAC2*(RSPAN)/FLOAT(KRN)
CALL DISTO(DFT,KRN+1.5,0)

C CALL LININT(X1,X2,TOUT(NSEC,NPTS)/2),PT2,ZLOCAL,TOB
C
DO 270 X=1,EK
270 CONTINUE

C Smooth the fuselage-wake part
C
DO 280 X=1,EDIM
280 CONTINUE

CALL FILE(TINT,YINT,EDIM,MB1,MB2,MT1,MT2,RFILT)
DO 290 I=1,EDIIM
   Y(L.E)+YINT(E)
   Z(L.E)+ZINT(E)
290 CONTINUE

DO 295 E=1,NPE
   YINT(E)=YIN(M,E)
   ZINT(E)=ZIN(M,E)
295 CONTINUE

CALL DISTANC(YINT,ZINT,NPE,THEN,EDIST,RTIP,0.05,1)

DO 290 E=1,RTIP
   Z(E)=EDIST(E)
290 CONTINUE

RETURN
SUBROUTINE WGRID

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

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

COMMON /REF/ SROOT,STIP,ARCOR

C Read surfgrid dimensions

C XIN,YN,ZIN - read in grid
X, Y, Z = output grid

C SROOT,S1,S2 = dummy vectors

C EDIM = # of points of the whole body in the warp-around direction
C BROOT = 0 if the nose bottom part (circumferential direction)
C FWIND = 0 in the nose top part (circumferential direction)
C TASC = 0 in the tail part (circumferential direction)

C RFL = Fillet radius
ARCOVER - This makes fillet looks smooth

C D1 = Distance of the trailing wake in body length

C NWAKE = Number points along the trailing wake

C SCALE = Scale factor for the wing-body

NAMELIST /BOD/ EDIM,FROOT,FWIND,FWIND,TASC,FROOT

&
& BLAKE,FNAXE,CFUSE,SCALE
& /FIL/ RFL,ARCOVER,MS1,MS2,MT1,MT2,ASTART,BODF

&
& /OPTN/ TREFAND,STAND

READ(40,BODY)
READ(40,FIL)
READ(40,OPN)
WRITE(*,BODY)
WRITE(*,*OPTN)
WRITE(*,TREF)
WRITE(*,STAND)

C Read the fuselage geometry

C CALL FUSE

C Take this initial grid
1) convert it into area distribution
2) add shape function
3) convert it back to grid

CALL FUSE

BI = XIN(NF)
BOL = XIN(NF)
ETIP = (EDIM1)/2

C Find number of streamwise stations. We keep the stations of the fuselage. As to the common area of the wing and fuselage we use the stations of the wing.
M1 is the station the nose ends
M2 is the station the wing ends
M1+1
M2+1

DO 10 M = 1,NF
IF (ABS(XIN(M)-BOL(1,1)) .LT. 1.E-7) M = M+1
IF (ABS(XIN(M)-BOL(NSEC,1)) .LT. 1.E-7) M = M+1
10 CONTINUE

IF (M1 .LT. 0) THEN
DO 20 M = 1,NF
IF (XIN(M) .LT. BOL(NSEC,1) AND XIN(M) .GT. BOL(1,1)) M = M+1
20 CONTINUE

ENDIF

IF (M1 .LT. 0) THEN
DO 30 M = 1,NF
IF (XIN(M) .LT. BOL(NSEC,1) AND XIN(M) .GT. BOL(1,1)) M = M+1
30 CONTINUE

ENDIF

C The configuration into four parts:
C The nose, the wing-body, the tail, and the extended wake, and for each part we distribute points for the lower and upper part separately.

C From the nose to the leading edge
CALL NOSE(MBOT,MTOP)

C From the leading edge to the trailing edge (the whole wing)
CALL WING

C From the trailing edge to the end of tail
CALL TAIL(-1,PACK)

C Redistribute each streamwise station, so that the grid is clustered near the wing tip
IF (CFUSE .LT. 0.0) CALL EQSPACE

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

C Add the extended wake
L2 = L1
XLAST = X(L2) + BI*NWAKE

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

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

Write out new surfgrid in plotld format.

Write the wing-body grid (X,Y,Z) to (XOUT,YOUT,ZOUT)
SUBROUTINE WFMATCH

INCLUDE 'SGR.D.'

DIMENSION O1(NPI), O2(NPI)

C

DIM IN, ZROOT, KTIP, ARCORR

C

DZMIN = 1.E-20

LB = M1+1

LE = M1*(NSEC-1)

LW = M1+1

LOCAL = IOUT(M1)

C

CALCULATE THE POINTS [TINT,JINT] ON THE FUSelage AT LOCAL

C

NOTE: ASSUMED THAT IN THIS SECTION, EACH STATION HAS SAME NUMBER

C

OF POINTS IN L-R AND D-R DIRECTIONS.

DO 100 M=M1,NF

IIF ([IN(M)] .GE. LOCAL .OR.

ABS([IN(M)]-LOCAL).LE.1.E-7) THEN

MN = M

GOTO 15

ENDIF

10 CONTINUE

15 CONTINUE

DO 10 E=1,NPP

I1=IN(MN-1)

I2=IN(MN)

Y1=IN(MN-1,E)

Y2=IN(MN,E)

Z1=IN(MN-1,E)

Z2=IN(MN,E)

CALL LININT(X1,X2,Z1,Z2,XIN(M),ZROOT,XT,YY)

CALL LININT(X1,X2,Z1,Z2,XIN(M),ZROOT,XY,YY)

ZINT(K) = ZY

YINT(K) = YY

10 CONTINUE

CONTINUE

35 CONTINUE

C

MOVE THE WING IN THE Z-DIRECTION [SPACING] TO MAKE SURE

C

THE WING IS NOT INSIDE OR OUTSIDE THE FUSelage.

ZROOT = ZINT([NPP+1]/2)

DO E1 = 1,NPP

IIF ([E1].GT.ZROOT) ZROOT-ZINT(E1)

ENDDO

DS = IOUT(LW,1)-ZROOT

IF ([ABS(DS)].GT.[ABS(DMAX)]) DIMAX+DS

IF ([ABS(DS)].LT.[ABS(DMIN)]) DIMIN+DS

C

200 CONTINUE

IF ([DIMAX].GT.0.) DZ=DIMIN

IF ([DZMIN].LT.0.) DZ=DIMAX

DO 400 E=1,NPP

DO 400 I=1,NSEC

IIF ([ZOUT(I,E)].GE.[ZOUT(L,E)]-DZ

400 CONTINUE

RETURN

END
**SUBROUTINE WING_BODY**

*include "samgrid.com"*

**DIMENSION DI(NPT),D2(NPT)**

**COMMON /REF/ XOUT, YOUT, ARCORS**

The wing-body section, itself has NSEC sections

**LW** is the section index for YIN, LEM

**LB = M+1**

**LE = M + NSEC - 1**

**DO 200 L=LB,LE**

**LWL=M+1**

**XLOCAL = XOUT(LW,1)**

**200 CONTINUE**

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

Note: assumed that in this section, each station has same number of points in space around direction.

**DO 10 M=NL,NP**

**IF(XIN(M).GE.ILOCAL).OR. & AB$**

**GOTO 20**

**10 CONTINUE**

**IF(YINT(K).LT.YOUT(LN,NPTS).AND.YINT(K).NE.YINT(K-I)) THEN**

**K1=K**

**GOTO 30**

**ENDIF**

**60 CONTINUE**

**IF(YINT(K).GT.YOUT(LW,K).AND.YINT(K).NE.YINT(K+I)) THEN**

**K2=K**

**GOTO 70**

**ENDIF**

**70 CONTINUE**

**IF(K_S (YO_T(LW,1)-YOUT(LN,NPTS) ).LE. 1.E-7) THEN**

**KS=K**

**KE=K+S**

**DO 82 K=K,KE**

**DI(K) = YINT(K)**

**D2(K) = ZINT(K)**

**82 CONTINUE**

**CALL DISTARC(D1,D2, KN,YNEW,ZDIST,MID,-10., 1)**

**DO 100 K=1,NID**

**Y(L,K) = YNEW(K)**

**Z(L,K) = ZDIST(K)**

**100 CONTINUE**

**CALL DISTARC(D1,D2, KN,YNEW,ZDIST,MID,0., 0)**

**DO 120 K=1,MID**

**Y(L,K+MID+NPTS) = YNEW(K)**

**Z(L,K+MID+NPTS) = ZDIST(K)**

**120 CONTINUE**

**C For arrow-wing type, make sure wake points ok**

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

**K1 = K + ELADD**

**E1 = E**

**TINT(E2)=TINT(E1)**

**ZINT(E2)=ZINT(E1)**

**ENDIF**

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

**ES=1**

**EE=EE+ES**

**EN=EN+EE**

**DI(EE) = YINT(E)**

**D2(EE) = ZINT(E)**

**82 CONTINUE**

**CALL DISTARC(D1,D2, EN,YNEW,ZDIST,MID,10.,1)**

**DO 100 K=1,MID**

**Y(L,K) = YNEW(K)**

**Z(L,K) = ZDIST(K)**

**100 CONTINUE**

**C The points of the wing section**

**DO 110 K=1,NPTS**

**Y(L,MID) = YOUT(LM,E)**

**Z(L,MID) = ZOUT(LM,E)**

**110 CONTINUE**

**C Calculate the points at the top**

**ES=2**

**EE=EE+ES**

**EN=EN+EE**

**DI(EE) = YINT(E)**

**D2(EE) = ZINT(E)**

**82 CONTINUE**

**CALL DISTARC(D1,D2, EN,YNEW,ZDIST,MID,-10.,0)**

**DO 100 K=1,MID**

**Y(L,K) = YNEW(K)**

**Z(L,K) = ZDIST(K)**

**100 CONTINUE**

**C For arrow-wing type, make sure wake points ok**

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

**Y(L,MID+NPTS) = YOUT(LW,1)**

**Z(L,MID+NPTS) = ZOUT(LW,1)**
2252       Z(L,MID) = Z(L,MID+NPTS+1)
2253       ENDIF
2254       
2255       C       Fill the unsmooth part by FILT
2256       C       First of all, find the set of points needed to be rearrange
2257       DO 125 K=1,KDIM
2258           YINT(K)=Y(L,K)
2259           ZINT(K)=Z(L,K)
2260       125      CONTINUE
2261       
2262       IF(X(L).GE.XSTART .AND. X(L).LE.XOFF) THEN
2263           CALL FILT(YINT, ZINT, KDIM, MB1, MB2, MT1, MT2, RFILT)
2264           DO 140 K=1,KDIM
2265               Y(L,K)=YINT(K)
2266               Z(L,K)=ZINT(K)
2267           140      CONTINUE
2268       200      CONTINUE
2269       RETURN
2270       END
SUBROUTINE WINGHOLE(XLOCAL,ET,NUL,IFLAT)

include "grid.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(ARRING.GT.0.) THEN

The wing is swept backwards
DO 75 J=2,NC
J1=J-1
J2=J+1
IF(XLOCAL.GT.XLE(J1)+CHORD(J1)) GOTO 75
XI1=XLE(J1)+CHORD(J1)
Y1=YBASE(NUL,IFLAT,J1)
Z1=ZBASE(NUL,IFLAT,J1)
Z2=ZBASE(NUL,IFLAT,J2)
ZTIP=(XI1*(XLOCAL-XI1)/(XI1-XI2))**2
ZFLAT=ZTIP

IF(NLEDG.EQ.NC) THEN
NLEDG=NC
DO MUN=2,NC
IF(XLOCAL.EQ.XBASE(MUN)) THEN

IF(XLOCAL.LE.XLE(J)) THEN
XI=XLE(J)
X2=XLE(J)+CHORD(J)
Y2=YBASE(NUL,IFLAT,J)
Z2=ZBASE(NUL,IFLAT,J)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
ENDIF
ENDIF
CONTINUE
GO TO 75
ELSE

ELSE
C The wing is swept forwards
DO 105 J=2,NC
J1=J-1
J2=J+1
IF(XLOCAL.LE.XLE(J1)+CHORD(J1)) GOTO 105
X1=XLE(J1)+CHORD(J1)
Y1=YBASE(NUL,IFLAT,J1)
Z1=ZBASE(NUL,IFLAT,J1)
Z2=ZBASE(NUL,IFLAT,J2)
ZTIP=(XI1*(XLOCAL-XI1)/(XI1-XI2))**2
ZFLAT=ZTIP

IF(NLEDG.EQ.NC) THEN
NLEDG=NC
DO MUN=2,NC
IF(XLOCAL.GT.XBASE(MUN)) THEN

IF(XLOCAL.LE.XLE(J)) THEN
XI=XLE(J)
X2=XLE(J)+CHORD(J)
Y2=YBASE(NUL,IFLAT,J)
Z2=ZBASE(NUL,IFLAT,J)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
ENDIF
ENDIF
CONTINUE
GO TO 75
ELSE

ELSE
C

C Add 10 points for the wake.
ZROOT=ZBASE(NUL,IFLAT,J)
Y1=YBASE(NUL,IFLAT,J)
Y2=YBASE(NUL,IFLAT,J)
Z1=ZBASE(NUL,IFLAT,J)
Z2=ZBASE(NUL,IFLAT,J)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,ZROOT,ZZ)
ENDIF
ENDIF
CONTINUE
GO TO 75
ELSE

ELSE
C The wing is swept backwards
DO 105 J=2,NC
J1=J-1
J2=J+1
IF(XLOCAL.LE.XLE(J1)+CHORD(J1)) GOTO 105
X1=XLE(J1)+CHORD(J1)
Y1=YBASE(NUL,IFLAT,J1)
Z1=ZBASE(NUL,IFLAT,J1)
Z2=ZBASE(NUL,IFLAT,J2)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,ZROOT,ZZ)
DO = (ZTIP-ZFLAT)/9.
DO 80 M=1,10
ZINT(M)=ZTIP+FLOAT(M-1)*DO
CALL LININT(ZTIP,ZZ,YOUT(1,NPTS/2+M),YY,ZINT(M),YYY)
ENDIF
ENDIF
CONTINUE
GO TO 75
ELSE

ELSE
C Add 10 points for the wake.
Y1=YBASE(NUL,IFLAT,J)
Y2=YBASE(NUL,IFLAT,J)
Z1=ZBASE(NUL,IFLAT,J)
Z2=ZBASE(NUL,IFLAT,J)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,ZROOT,ZZ)
DO = (ZTIP-ZFLAT)/9.
DO 100 M=1,10
ZINT(M)=ZTIP+FLOAT(M-1)*DO
CALL LININT(ZTIP,ZZ,YOUT(1,NPTS/2+M),YY,ZINT(M),YYY)
ENDIF
ENDIF
CONTINUE
GO TO 75
ELSE

ELSE
C The wing is swept forwards
DO 105 J=2,NC
J1=J-1
J2=J+1
IF(XLOCAL.LE.XLE(J1)+CHORD(J1)) GOTO 105
X1=XLE(J1)+CHORD(J1)
Y1=YBASE(NUL,IFLAT,J1)
Z1=ZBASE(NUL,IFLAT,J1)
Z2=ZBASE(NUL,IFLAT,J2)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,ZROOT,ZZ)
DO = (ZTIP-ZFLAT)/9.
DO 200 M=1,10
ZINT(M)=ZTIP+FLOAT(M-1)*DO
CALL LININT(ZTIP,ZZ,YOUT(1,NPTS/2+M),YY,ZINT(M),YYY)
ENDIF
ENDIF
CONTINUE
GO TO 75
ELSE

ELSE
C

C Add 10 points for the wake.
Y1=YBASE(NUL,IFLAT,J)
Y2=YBASE(NUL,IFLAT,J)
Z1=ZBASE(NUL,IFLAT,J)
Z2=ZBASE(NUL,IFLAT,J)
CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
CALL LININT(X1,X2,Z1,Z2,ZROOT,ZZ)
DO = (ZTIP-ZFLAT)/9.
DO 300 M=1,10
ZINT(M)=ZTIP+FLOAT(M-1)*DO
CALL LININT(ZTIP,ZZ,YOUT(1,NPTS/2+M),YY,ZINT(M),YYY)
ENDIF
ENDIF
CONTINUE
GO TO 75
ELSE

ELSE
C
TINT(M) = YTY
DO 100 N=1,1,-1
   DO 90 I=2,N+1
      IF(LOCAL.LE.XBASE(I,IFLAT,M)) THEN
         X1 = XBASE(I-1,IFLAT,M)
         Y1 = YBASE(I-1,IFLAT,M)
         Z1 = ZBASE(I-1,IFLAT,M)
      ENDIF
      CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
      CALL LININT(X1,Z1,Z1,Z1_CBL,ZZ)
      YINT(JI-N+I10) = YY
      ZINT(JI-N+I +10) = ZZ
      GOTO 100
   CONTINUE
100   CONTINUE
105   CONTINUE
ENDF
GOTO 200
END
SUBROUTINE WINGIN
    include "sgrid.com"

    NC = # of sections in the spanwise direction
    XBASE = X value of 1st section
    YBASE = Y value of 1st section
    YLD = leading edge Y value of 1st section
    Y endanger (X) = chord length of the 1st section
    N = # of points in the upper and lower sections are same physical pts

    MAC configuration:
    READ(10,900)
    READ(10,910)
    READ(10,920)NC,NU
    NL = NU
    READ(10,930)
    READ(10,940)
    READ(10,950)
    READ(10,960)
    DO 12 I=1,NU
    READ(IO,* )EBASE(I,2,K),YBASE(I,2,K),BASE(I,2,K)
    READ(IO,*) data
    ... 
    READ(IO,*) data
    DO 15 I=1,NU
    READ(IO,* )BASE(I,1,K)
    READ(IO,*) data
    ... 
    READ(IO,*) data
    IF(I.LE.NC)GOTO111
    ... 
    CONTINUE

    K=1
    DO 12 I=1,NU
    READ(IO,* )EBASE(I,1,K),YBASE(I,1,K),BASE(I,1,K)
    READ(IO,*) data
    ... 
    READ(IO,*) data
    DO 15 I=1,NU
    READ(IO,* )BASE(I,1,K)
    READ(IO,*) data
    ... 
    READ(IO,*) data
    IF(I.LE.NC)GOTO111
    ... 
    CONTINUE

END
SUBROUTINE FUSEIN

include 'egrid.com'
DIMENSION TT(NP1),ST(NP1)

C NF = # of sections in the fuselage
C NPF = # 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 N=0
CONTINUE
DO 40 K = 1, NPF
READ(10,840) M - M + 1
C DO 100 I=1,NPF
READ(15,*), XIN(M), YIN(M), ZIN(M)
TT(I) = YIN(M)
ST(I) = ZIN(M)
CONTINUE
DO 100 I = 1, NPF
C CALL DISPAC(TT, ST, NPF, TT, ST, -15., 0)
CONTINUE
DO 100 I = 1, NPF
C Write the fuselage geometry into PLOTID Planar format.
M = O
DO 100 I = 1, NPF
WRITE(51), XIN(M), YIN(M)
WRITE(51), (XIN(L), L = 1, EE)
WRITE(51), (YIN(L), L = 1, EE)
WRITE(51), (ZIN(L), L = 1, EE)
CONTINUE
DO 100 I = 1, NPF
CONTINUE
DO 100 I = 1, NPF
RETURN
END
SUBROUTINE HACIN
  include "agrid.com"
  DIMENSION TT(NPI),ST(NPI)
  COMMON /HAC/, NAC(NPI,3),NACP(NPI,NPI)
  COMMON /HIN/, NAC,NACP,INHIN(4)
C
C NAC  = # of section in theocsels
C NACP  = # of points in ith section
C
C Read the nacelle geometry
C
READ(10,840) M = M + 1
C
DO 100 E=1,NACP
  READ(10,850) TT(M,E),ST(M,E)
C
100 CONTINUE
C
C Write the nacelle geometry into PILOT3D Planar format
C
C E=1
C
WRITE(32)[(NACP)]
C
DO 800 I=1,E
  WRITE(32)[(NAC(I,E),E=1,E), (DONC(I,E),E=1,E)]
C
800 CONTINUE
C
C Call flush (32)
C
810 FORMAT(13)
820 FORMAT(15)
830 FORMAT(31X,15.8E,15/)
840 FORMAT(13)
850 FORMAT(3X,F16.7)
C
RETURN
END
SUBROUTINE WINGMAKER

C include 'sgrid.com'

C This program generates a 'clipped' delta wing with no twist
dependent on airfoil coordinates read in from fort 90. To use as
part of sgrid, WINGMAKER is not necessary (nor is VARIWIND).

    Written by Donovan L. Mathias
    - July 1982

    Whenever possible the same variables are used as in sgrid.f
    Fort 90 airfoil coordinates

C Description of the wing

C Declarations

REAL XLE(L),SLOPE1,SLOPE2,SCALE,XAF(150)
REAL YU(150),YF(150)
REAL SPAN
INTEGER I,J,K

C Initialization

DO 19 I=1,NU
    READ(90,*) XI(I)
19 CONTINUE
ENDDO

C Read in airfoil coordinates (L is # of X coords.)

DO 20 I=1,NU
    READ(90,*) XI(I),YU(I),YF(I)
20 CONTINUE

C Establish Spans (Spanwise)

SPAN = ZBASE(1,1,NC) - ZBASE(1,1,1)
DO 30 K=1,NC
    XI(K) = XI(1) + SPAN*(K/(NC-1))
30 CONTINUE

C Establish sweep (1 FOR LE, 2 FOR TE)

SLOPE1 = (XI(2)-XI(1))/(ZBASE(1,1,NC)-ZBASE(1,1,1))
SLOPE2 = (XI(NC)-XI(1))/(ZBASE(1,1,NC)-ZBASE(1,1,1))

C Generate leading and trailing edges

DO 50 K=1,NC
    XLE(K) = XI(K) + SLOPE1*ZBASE(1,1,K)
DO 50 K=1,NC
    XTE(K) = XI(K) + SLOPE2*ZBASE(1,1,K)

C Distribute grid points

DO 60 K=1,NC
    SCALE = XTE(K)-XLE(K)
DO 60 K=1,NU
    ZBASE(1,1,K) = XLE(K) + SCALE*XAF(I)

DO 60 K=1,NU
    ZBASE(1,1,K) = YU(I)*SCALE + XLE(K)

DO 60 K=1,NU
    ZBASE(1,1,K) = YF(I)*SCALE + XLE(K)

C Return values to original names

C WRITE(3,100)
CA (1,1), I, N, = 100
END

C END SUBROUTINE WINGMAKER
# 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
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 `pvmmd`. 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.so`. 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.
Programming Concept

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

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

You need to distribute executables of Slave programs to the directory $HOME/pvma3/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

```fortran
program spmd
  include '../include/fpvm3.h'
  parameter( nproc=4 )
  integer mytid, me, i
  integer tids(0:nproc)

  enroll in pvm
  call pvmfmytid( mytid )

  find out if I am parent or child
  call pvmfparent(tids(0))
  if( tids(0) .lt. 0 ) then
    tids(0) = mytid
    me = 0
    start up copies of myself
    call pvmfspawn('spmd', pvmdefault, '*', nproc-1, tids(1), info)
    multicast tids array to children
    call pvmfinitsend( pvmdefault, info )
    call pvmfpack( integer4, tids, nproc, 1, info )
    call pvmfmcast( nproc-1, tids(1), 0, info )
  else
    receive the tids array and set me
    call pvmfrecv( tids(0), 0, info )
    call pvmfunpack( integer4, tids, nproc, 1, info )
    do 30 i=1, nproc-1
      if( mytid .eq. tids(i) ) me = i
    continue
  endif

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

  program finished exit pvm
  call pvmexit(info)

end
```

7
Notes
subroutine dowork(me, tids, nproc)
include '../include/fpvm3.h'

C Simple subroutine to pass a token around a ring

C
integer me, nproc
integer tids(0:nproc)

integer token, dest, count, stride, msgtag

count = 1
stride = 1
msgtag = 4

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

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

```
pvm>
```

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

### Console Commands

Here are the commands available in the PVM console:

- **ADD**: add other computers (hosts) to PVM
- **ALIAS**: define and list command aliases you set
- **CONF**: show members in virtual machine
- **DELETE**: remove hosts from pvm
- **ECHO**: echo arguments
- **HALT**: stop all pvm processes and exit daemon
- **HELP**: print this information
- **ID**: print console task identity
- **JOBS**: display list of running jobs
- **KILL**: terminate tasks
- **MSTAT**: show status of hosts
- **PS**: list tasks
- **PSTAT**: show status of tasks
- **QUIT**: exit PVM console, but PVM daemon is still activated
- **RESET**: kill all tasks
- **SETENV**: display or set UNIX environment variables
- **SIG**: send signal to task
- **SPAWN**: spawn task
- **UNALIAS**: remove alias commands you previous set
- **VERSION**: show PVM version
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**
   
   ```plaintext
   win210> pvm
   ```

2. **Add amelia and fred to your virtual machine**
   
   ```plaintext
   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**
   
   ```plaintext
   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**
   
   ```plaintext
   pvm> delete amelia
   1 successful
   HOST STATUS
   amelia deleted
   ```

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

**Process Control**

**call pvmfmytid( tid )**
This routine enrolls this process with the PVM daemon on its first call, and generates a unique 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 `info` less than zero indicate an error.

**call pvmfkill( tid, info )**
This routine kills a PVM task identified by uId. Values of `info` less than zero indicate an error.

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

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

*where* is where you want to start the PVM process. If *flag* is 0, *where* is ignored.

*tids* contains identification numbers of PVM processes started by this routine.

*numt* indicates how many processors started; negative values indicate an error.

**Note** You should always check *tids* and *numt* to make sure all processes started correctly.
call pvmfparent ( tld )
This routine returns the uid of the process that spawned this task. If the calling process was not created with pvmspawn, 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</th>
<th>REAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0)</td>
<td>(4)</td>
</tr>
<tr>
<td>BYTE1</td>
<td>COMPLEX8</td>
</tr>
<tr>
<td>(1)</td>
<td>(5)</td>
</tr>
<tr>
<td>INTEGER2</td>
<td>REAL8</td>
</tr>
<tr>
<td>(2)</td>
<td>(6)</td>
</tr>
<tr>
<td>INTEGER4</td>
<td>COMPLEX16</td>
</tr>
<tr>
<td>(3)</td>
<td>(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 pvmfnrecv( 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=0. This routine can be called multiple times for the same message to check if it has arrived, while performing useful work between calls. When no more useful work can be performed, the blocking receive 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 pvmfbufinfo (bufid, bytes, msgtag, tid, info)
This routine returns information about the message in the buffer identified by bufid. The information returned is the actual msgtag, source tid, and message length in bytes. Values of info less than zero indicate an error.
This chapter shows you how PVM may be applied to your application programs through a simple example. The example chosen is the Golden Section rule for finding the maximum of a function. You may remember it from Math class in high school. Let us review the method and the algorithm.

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

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

(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

Loop begins:
L = L + 1

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

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 evaluation
Recall that in the procedure of finding a new interval, the program calls the function evaluation four times *serially* to get f(a₁), f(a₂), f(a₃), and f(a₄). We would like to assign four processors to perform the four function evaluations *simultaneously* on the virtual machine. Therefore, we modify the Serial Program by writing the main (calling) program as a Master program, and the function subroutine as a Slave program.

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

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

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

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

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

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

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 pvmfisend(PVMDFAULT,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)

Pack nproc, tids, A, and ERR

msgtype = 1
  call pvmfmcast(nproc,tids,msgtype,info)
6. **Wait until messages come from Slaves**
   Use `pvmfrecv()` 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 `pvmfexit(info)` to exit PVM.
Wait for results from processors

\[ \text{msgtype value matches the one sent from Slave program} \]

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

\[ \text{msgtype} = 2 \]
\[ \text{do 100 i=1,nproc} \]
\[ \text{call pvmrecv(-1,msgtype,info)} \]
\[ \text{call pvmunpack(INTEGER4,who,1,1,info)} \]
\[ \text{call pvmunpack(REAL4,FN(who),1,1,info)} \]
\[ \text{continue} \]
\[ \text{WRITE(10,*,'A ',A(1),A(2),A(3),A(4))} \]
\[ \text{WRITE(10,*,'F ',FN(1),FN(2),FN(3),FN(4))} \]
\[ \text{WRITE(10,*,'')} \]
\[ \text{ERR = ABS(FN(2)-FN(3))} \]
\[ \text{IF(ERR.LE.TOL) GOTO 999} \]

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

\[ \text{call pvmexit(info)} \]
\[ \text{continue} \]
\[ \text{STOP} \]
\[ \text{END} \]
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 \((t_1, \ldots, t_4)\) 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 \texttt{PVMDFAULT, REAL4}, and more, mentioned in all tables in Chapter 4 and the Appendix.

2. **Enroll Slave with PVM**
   
   Use \texttt{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.
   
   \texttt{pvmfparen(t(mtld))}

4. **Receive and Unpack data**
   
   Make sure the value of \texttt{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 \texttt{pvmfinitsend} to clear buffer.
   
   Use the following call to pack a real array \(F\) of dimension \(n\).
   
   \texttt{pvmfpack(REAL4,F,n,1,info)}

7. **Send data**
   
   Use the following call to send the packed message to Master:
   
   \texttt{pvmfsend(mtid,\text{msg}tag,\text{info})}

8. **Exit PVM**
   
   Use \texttt{pvmfexit(info)} to exit PVM.
Slave Program

1. program function
2. include '../include/fpvm3.h'
3. integer tids(0:32), who
4. real a(32)
5. tor = 1.e-3
6. Enroll this program in PVM
7. call pvmfmytid(mytid)
8. Get the parent's task id
9. call pvmfparent(mtid)
10. continue
11. Receive data from host
12. msgtype = 1
13. call pvmfrecv(mtid,msgtype,info)
14. call pvmfunpack(INTEGER4,nproc,1,1,info)
15. call pvmfunpack(INTEGER4,tids,nproc,1,info)
16. call pvmfunpack(REAL4,a,4,1,info)
17. call pvmfunpack(REAL4,ERR,1,1,info)
18. if(err.le.tor) go to 99
19. Determine which processor I am
20. do 5 i=0,nproc-1
21. if(tids(i).eq.mytid) me = i
22. continue
23. who = me + 1
24. Calculate the function
25. X = A(who)
26. f = TANH(X)/(1.+X*X)
27. send the result to Master
28. call pvmfinitsend(PVMDEFAULT,info)
29. call pvmfpack(INTEGER4,who,1,1,info)
30. call pvmfpack(REAL4,f,1,1,info)
31. msgtype = 2
32. call pvmfsend(mtid,msgtype,info)
33. go to 3
34. Program finished. Leave PVM before exiting
35. continue
36. call pvmfexit(info)
37. stop
38. end
Compilation and Running

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

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

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

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

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

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

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

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

# Custom section
# Set PVM_ARCH to your architecture type (SUN4, HP9K, RS6K, SGI, etc.)
# if PVM_ARCH = BSD386 then set ARCHLIB = -lrpc
# if PVM_ARCH = SGI then set ARCHLIB = -lsun
# if PVM_ARCH = I860 then set ARCHLIB = -lrpc -lsocket
# if PVM_ARCH = IPSC2 then set ARCHLIB = -lrpc -lssocket
# 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_irlx4/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 quadfunct

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

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

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

Problems

Can't activate PVM

- If the message you get, after entering `pv_` at UNIX prompt, is `libpvm [pid=1]: Console: Can't start pmad`, 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 `$HOME` 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.

\[
\text{computer1.address} \\
\text{computer2.address} \\
\text{computer3.address} \\
\text{computer4.address}
\]

\[\text{host file}\]

**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

Notes

Place to jot down problems.

If encounter problems, please contact:

Merritt Smith: mhsmith@nas.nasa.gov

or

Samson Cheung: cheung@nas.nasa.gov
## Appendix

### TABLE 1. ARCH names used in PVM.

<table>
<thead>
<tr>
<th>ARCH</th>
<th>Machine</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFX8</td>
<td>Alliant FX 8</td>
<td></td>
</tr>
<tr>
<td>ALPHA</td>
<td>DEC Alpha</td>
<td>DEC OSF-1</td>
</tr>
<tr>
<td>BAL</td>
<td>Sequent Balance</td>
<td>DYNIX</td>
</tr>
<tr>
<td>BFLY</td>
<td>BBN Butterfly TC2000</td>
<td></td>
</tr>
<tr>
<td>BSD386</td>
<td>80386/486 Unix box</td>
<td>BSDI</td>
</tr>
<tr>
<td>CM2</td>
<td>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>IR60</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 -lrun</td>
</tr>
<tr>
<td>SUN3</td>
<td>Sun 3</td>
<td>SunOS</td>
</tr>
<tr>
<td>SUN4</td>
<td>Sun 4, SPARCstation</td>
<td></td>
</tr>
<tr>
<td>SYMM</td>
<td>Sequent Symmetry</td>
<td></td>
</tr>
<tr>
<td>TITN</td>
<td>Staedent Titan</td>
<td></td>
</tr>
<tr>
<td>UVAX</td>
<td>DEC Micro VAX</td>
<td></td>
</tr>
</tbody>
</table>
### Table 2. Error codes returned by PVM routines

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PvmOK (0)</td>
<td>All right</td>
</tr>
<tr>
<td>PvmBadParam (-2)</td>
<td>Bad parameter</td>
</tr>
<tr>
<td>PvmMismatch (-3)</td>
<td>Barrier count mismatch</td>
</tr>
<tr>
<td>PvmNoData (-5)</td>
<td>Read past end of buffer</td>
</tr>
<tr>
<td>PvmNoHost (-6)</td>
<td>No such host</td>
</tr>
<tr>
<td>PvmNoFile (-7)</td>
<td>No such executable</td>
</tr>
<tr>
<td>PvmNoMem (-10)</td>
<td>Can't get memory</td>
</tr>
<tr>
<td>PvmBadMsg (-12)</td>
<td>Can't decode received massage</td>
</tr>
<tr>
<td>PvmSysErr (-14)</td>
<td>Pvm 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>