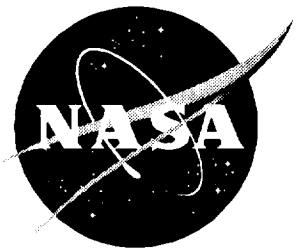
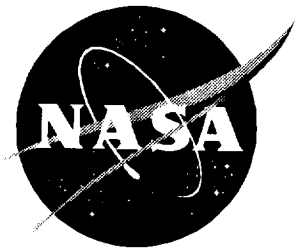
NASA Technical Memorandum 4674



User's Manual for the Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA)

F. McNeil Cheatwood and Peter A. Gnoffo

April 1996



NASA Technical Memorandum 4674

User's Manual for the Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA)

F. McNeil Cheatwood
ViGYAN, Inc. • Hampton, Virginia

Peter A. Gnoffo
Langley Research Center • Hampton, Virginia

National Aeronautics and Space Administration
Langley Research Center • Hampton, Virginia 23681-0001

April 1996

The use of trademarks or names of manufacturers in this report is for accurate reporting and does not constitute an official endorsement, either expressed or implied, of such products or manufacturers by the National Aeronautics and Space Administration.

Available electronically at the following URL address: http://techreports.larc.nasa.gov/ltrs/ltrs.html

Printed copies available from the following:

NASA Center for AeroSpace Information
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

Contents

Figures

Tables

Nomenclature

a	frozen sound speed, nondimensionalized by $\mathcal{V}_\infty$
E	total energy per unit mass of mixtures, nondimensionalized by $\mathcal{V}_\infty$
ϵ	internal energy, nondimensionalized by $\mathcal{V}_\infty^2$
I, J, K	total number of cells in i, j, k direction, respectively
K_{eq}	equilibrium constant
L	reference length defined via *iunit*
n	number of unknowns at cell
$\tilde{n}$	number density
p	pressure, nondimensionalized by $\rho_\infty \mathcal{V}_\infty^2$
R_N	body nose radius, in units specified via *iunit*
s	arc length, in units specified via *iunit*
T	translational-rotational temperature, K
T_V	vibrational-electron-electronic excitation temperature, K
T_w	surface (wall) temperature, K
$\mathcal{V}$	total velocity, m/s
x, y, z	Cartesian coordinates, in units specified via *iunit*
β	In general, $\beta = \partial p / \partial(\rho E)$; for perfect gas, this reduces to $\beta = \gamma - 1$
γ	ratio of specific heats
ϵ_0	parameters for defining minimum eigenvalue
μ	mixture viscosity, nondimensionalized by $\rho_\infty \mathcal{V}_\infty L$
ξ, η, ζ	computational coordinates
ρ	mixture density, nondimensionalized by ρ_∞
ρ_s	density of species s, nondimensionalized by ρ_∞
θ	body half-angle for conic geometry, deg

Subscripts:

b	body
e	electron
max (MAX)	maximum
s	species
∞	free stream

Chapter 1

Introduction

The National Aeronautics and Space Administration's interest in viscous, hypersonic flow field simulation has grown in recent years in anticipation of the design needs for space transportation and exploration over the next three decades, e.g., Walberg (ref. 1). Proposed aeroassisted space transfer vehicles will use the upper layers of planetary atmospheres in hypersonic aerobraking maneuvers. Supersonic combustion ramjet engines are being designed to propel vehicles at hypersonic speeds through the Earth's atmosphere to orbit. Various concepts for a single-stage-to-orbit (SSTO) vehicle are now being considered. The external flow field surrounding such vehicles, as well as the internal flow field through the scramjet engine and nozzle, can be significantly influenced by thermochemical nonequilibrium processes in the flow. Accurate simulations of these phenomena would provide designers valuable information concerning the aerodynamic and aerothermodynamic character of these vehicles.

This user's manual provides detailed instructions for the installation and the application of version 4.1 of the Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA) (refs. 2 and 3), which is a program for obtaining the simulations discussed above. Earlier versions of LAURA were predominantly research codes, and they had minimal (or no) documentation. This manual describes UNIX-based utilities for customizing the code for special applications that also minimize system resource requirements. The algorithm is reviewed, and the various program options are related to specific equations and variables in the theoretical development.

Two major challenges exist to the simulation of flow fields in thermochemical nonequilibrium around vehicles traveling at hypersonic velocities through the atmosphere. First, these simulations require modeling of the nonequilibrium processes in the flow; these processes frequently occur at energies in which the models currently lack sufficient experimental or analytic validation. Second, because of the large number of unknowns associated with chemical species and energy modes and because of disparate time scales within the flow field, these simulations require algorithmic innovations to maintain numerical stability and fully exploit supercomputer resources.

Nonequilibrium processes occur in a flow when the time required for a process to accommodate itself to local conditions within some region is of the same order as the transit time across the region. The equations and the models used in this manual for nonequilibrium flow have been documented in reference 4, and they were substantially derived from the work of Park (refs. 5 and 6) and Lee (ref. 7). Calibration and validation of the physical models intrinsic to this code were first discussed in reference 8. Other code development and calibration programs (e.g. GASP (refs. 9 to 11), Candler (ref. 12), Candler and MacCormack (ref. 13), Park and Yoon (ref. 14), Netterfield (ref. 15), and Coquel et. al (ref. 16)) are now in progress within the area of viscous, hypersonic, reacting gas flow field simulations.

Numerical stability is maintained through an implicit treatment of the governing equations. A great variety of implicit treatments is possible. For problems in which only the steady-state solution is required, one is free to evaluate any element of the difference stencil at any iteration (pseudo-time) level which facilitates the relaxation process. In the most rigorous implicit treatment, all variables in all cells are simultaneously solved at an advanced iteration level, thus requiring the solution of a linearized equation set involving $(n \times I \times J \times K)$ equations where n is the number of unknowns at a cell and I, J, and K are the number of computational cells in the three respective coordinate directions. The various forms of factored implicit schemes and line relaxation methods sequentially solve equation sets involving $(n \times I)$, $(n \times J)$, and/or $(n \times K)$ variables. The point-implicit schemes, as utilized in the present work, sequentially solve equation sets involving n simultaneous, linearized equations. Further simplification is possible in chemical kinetic problems by linearizing contributions to the residual from only the source terms to alleviate problems of disparate chemical time scales, thus resulting in methods which involve no matrix operations.

The essence of the point-implicit strategy is to treat the variables at the cell center of interest implicitly at the advanced iteration level and to use the latest available data from neighbor cells in defining the "left-hand-side" numerics. The success of this approach is made possible by the robust stability characteristics of the underlying upwind difference scheme. Even simulations of thermochemical nonequilibrium flows in a near-equilibrium state can be handled by this approach (ref. 17). The algorithm requires only a single pseudo-time level of storage and is efficiently implemented on vector or parallel processors (ref. 18). Details of the relaxation algorithm, including effects of a gas in thermal and chemical nonequilibrium, are presented herein.

As noted above, there is no requirement to synchronize the evolution of the solution at neighboring points in the single-level-storage point-implicit relaxation strategy. Consequently, algorithm parallelization can be implemented on a subroutine level across several domains without the need to synchronize tasks or restrict parallel code to a "do loop" level. Scalar code and conditional logic do not inhibit parallel efficiency. Dynamic allocation of resources to domains that are slow to converge is enabled in this environment. These capabilities are exploited on CRAY class computers and are discussed in greater detail within this manual.

The code and the user interface are structured to make liberal use of `FORTRAN include` statements that tailor the resource requirements for each case to a minimum. System requirements vary from standard workstations for many perfect-gas applications to 128 Mw (megaword) in-core memory, 128 Mw of "fast disk" (SSD) memory, and more than 100 central processing unit (CPU) hours to obtain a converged solution on a CRAY YMP for thermochemical nonequilibrium flow (seven species) over the Space Shuttle with the thin-layer Navier-Stokes equations using a grid of $150 \times 109 \times 60$.

This manual is designed to guide the user through the application, beginning with the installation of the source code on a given machine. Chapter 2 contains an outline of this manual, and provides an overview of the LAURA algorithm.

Chapter 2

Overview

This chapter provides an overview of the LAURA algorithm, and it can be skipped by the experienced user. Chapter 3 is a checklist (a brief outline of the entire procedure) which is useful for the experienced user who needs a quick review. Chapter 4 is a quick reference guide to the utilities and commands of LAURA. Chapter 5 details the source-code installation procedure. The setup procedure for a particular application is given in chapter 6, and the menus of the startup routine are given in chapter 7. Chapters 8 and 9 discuss the compilation and execution of LAURA, respectively, and chapter 9 describes the resultant output files. Approaches for a number of advanced applications are presented in Chapter 11. The various appendixes provide

- a sample case (appendix A)
- the equations for conic geometries (appendix B)
- the details of the various script files employed by LAURA (appendixes C to N)
- an in-depth discussion of the LAURA algorithm (appendix O)
- the `FORTRAN` variables discussed in this document (appendix P)
- the `FORTRAN` flags changed through `data` (appendix Q)
- the `FORTRAN` flags changed through `stArt` (appendix R)
- the `FORTRAN` flags changed through file edits (appendix S)

Although not a requirement, in hypersonic blunt-body applications, the origin of the coordinate system generally sits at or near the stagnation point on the body, with the z-axis pointing out from the body toward the oncoming flow, as shown in figure 2.1. The $y = 0$ plane defines the symmetry plane. Lifting-body applications retain this orientation in which the origin of the coordinate system is at or near the vehicle stagnation point, the z-axis points out from the nose, against the flow, and the negative z-axis typically runs through the interior of the vehicle.

Computational coordinates (ξ, η, ζ) run in the direction of increasing i-, j-, and k-indices, respectively, as shown in figure 2.2. The vehicle surface grid is usually defined by the $k = 1$ plane to enhance convergence through the use of over-relaxation of the viscous terms and special limiters when sweeping across the boundary layer, although other orientations are permitted.

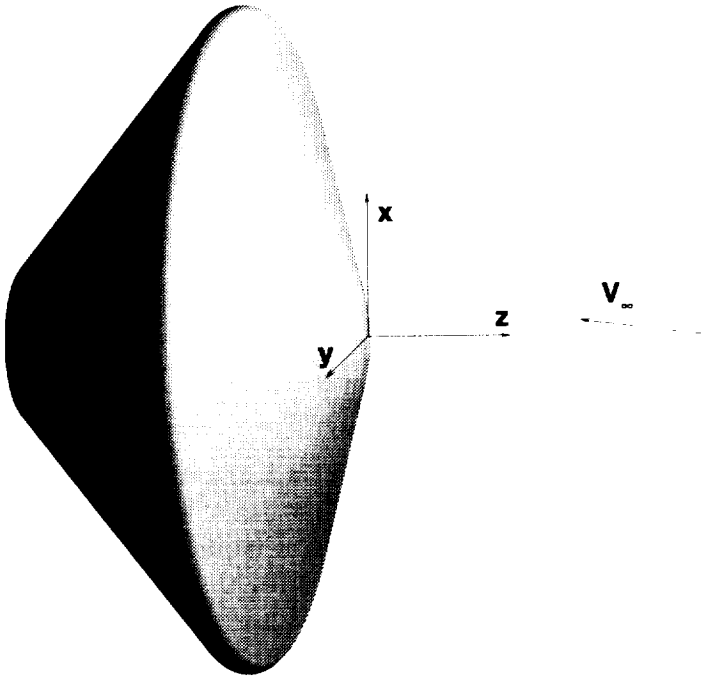


Figure 2.1. Typical orientation of Cartesian coordinate system with respect to blunt body in LAURA.

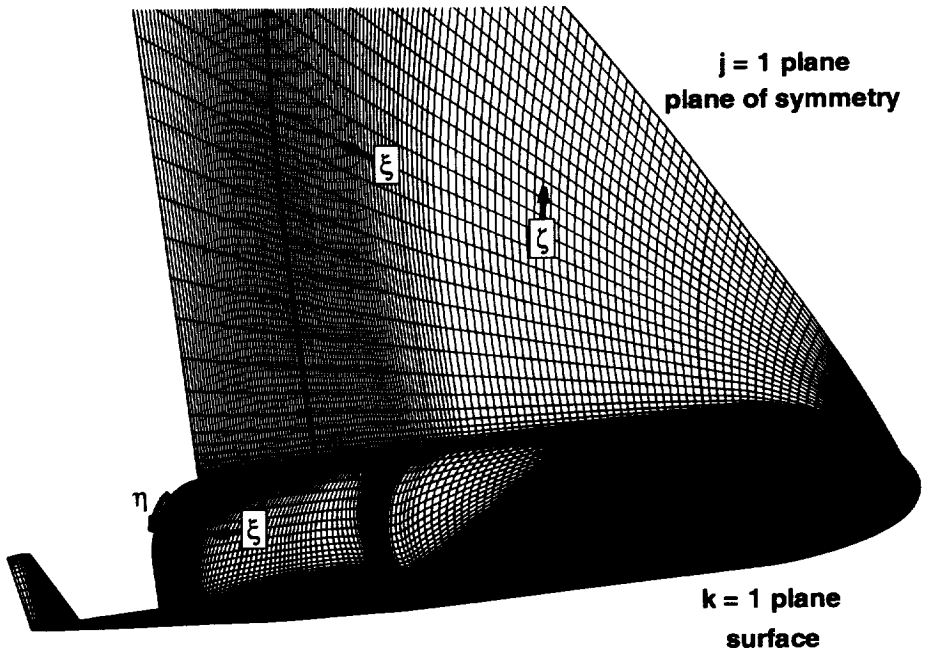


Figure 2.2. Typical orientation of computational coordinates over winged vehicle showing two computational planes.

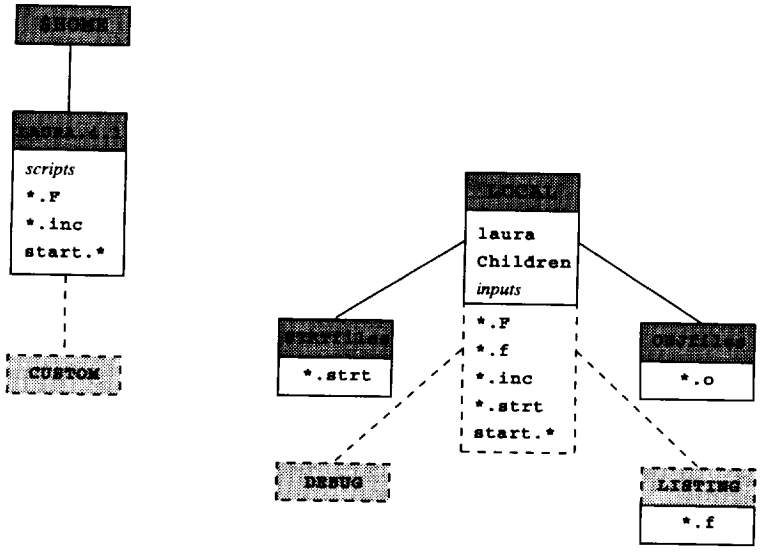


Figure 2.3. Subdirectory layout for LAURA.

NOTE: Options for automatically aligning the grid with the captured bow shock and resolving the boundary layer require this orientation with $k = 1$ on the surface.

The first step in applying LAURA to a given problem involves installation of the source code on the computer where the computations will be performed (chapter 5). LAURA has been successfully tested on SUN, CRAY, SGI, and CONVEX architectures. The LAURA package is distributed as two files, `LAURA.4.1.tar.Z` and `INSTALL_LAURA.4.1`, which should be placed in the user's `$HOME` directory. File `LAURA.4.1.tar.Z` is a `tarfile` (tape archival file) which contains the LAURA source code. The file `INSTALL_LAURA.4.1` is a script that executes the installation procedure.

NOTE: LAURA requires a UNIX operating system, and assumes the user has some familiarity with UNIX protocol.

LAURA is based on the premise that the bulk of the LAURA source code does not change from one application to the next. These files, which should rarely require modification, are kept in `$HOME/LAURA.4.1`, which is a read-only directory.

By default, LAURA uses the `$HOME/LAURA.4.1` source files, in conjunction with application-specific coding created through user-inputs to `PRELUDE` (the LAURA preprocessor). For further tailoring, the user can create a `LOCAL` version of any LAURA source file. This `LOCAL` file will be used for compilations from this working (`LOCAL`) directory only. This `LOCAL` file can be converted to a `CUSTOM` file if the user wishes to make it the default (in lieu of the `$HOME/LAURA.4.1` version) from any working directory. If a `CUSTOM` version of a file is encountered during compilation, it is used in lieu of the `$HOME/LAURA.4.1` version. If a `LOCAL` version of a file is encountered during compilation, it is used in lieu of the `$HOME/LAURA.4.1` version and the `CUSTOM` version (if one exists).

Figure 2.3 shows the relationship between (and roles of) the various directories employed by LAURA. The headers on each box are directory names, and the contents of each box are file types located in that directory. Solid lines indicate required files and directories, and the dashed lines indicate files and directories whose existence is dependent on which advanced features have been utilized.

To run LAURA interactively, type the command

```
laura < data > lfn
```

This command sends the output to file **lfn**. User control of a given run is provided through the input files and the include files with the **.strt** suffix (chapter 9).

A typewriter style is used to denote file names and directories, as well as **FORTRAN** coding. Italics are used for variable names and for units of measure. Shaded boxes with thin borders and sharp corners are used to denote contents of a file. Shaded boxes with thick borders and rounded corners are used to denote screen prompts. Commands that the user types are contained in smaller shaded boxes with thin borders and rounded corners.

The term **LOCAL** refers to files in the present working (or **LOCAL**) directory, which will be used in lieu of the installed files within this directory only. The term **CUSTOM** refers to tailored files that are used in lieu of the installed files from any working directory.

Chapter 3

LAURA Quick Reference Guide

This chapter discusses the installation, the specialization, and the application of LAURA and is intended as a quick reference guide.

3.1. Installation

To install LAURA, complete the following steps:

- Place files `LAURA.4.1.tar.Z` and `INSTALL_LAURA.4.1` in the `$HOME` directory (chapter 5). File `LAURA.4.1.tar.Z` should be read-only; if not, type

  `chmod 400 LAURA.4.1.tar.Z`

  The file `INSTALL_LAURA.4.1` should be executable; if not, type

  `chmod 500 INSTALL_LAURA.4.1`

- Type the command

  `INSTALL_LAURA.4.1`

  to install the LAURA source code on this machine (chapter 5).

3.2. Specialization

For specialization, do the following:

- Create (and change to) a working (`LOCAL`) directory.
- Type

  `PRELUDE`

  to create the subdirectories that LAURA requires (chapters 6 and 7). For some advanced applications, the `stArt` executable may require tailoring. In such cases, the desired changes should be made (via `LOCAL` or `CUSTOM` source files for `stArt`) before running `PRELUDE`.
- Type

  `make`

  to compile `laura`, the LAURA executable (chapter 8). For advanced applications, further tailoring of LAURA may be required (chapter 11) before executing `make`. If so, use `LOCALIZE` to create `LOCAL` versions of files for necessary modifications. If `CUSTOM` files containing the desired changes already exist, this step is not required.

NOTE: The `laura` executable must be recompiled after changes to any LAURA `FORTRAN` files (`.F`, `.FOR`, `.f`, `.inc`, or `.strt` suffixes).

3.3. Application

For LAURA application, complete the following steps:

- Review LAURA input files (chapter 9) and modify, if necessary.
- To run interactively, type the command

```
laura < data > lfn
```

  to run LAURA and send the output to file `lfn`. Individual runs are controlled via the LAURA input files (chapter 9).

- The LAURA output files are discussed in chapter 10.

Chapter 4

Summary of LAURA Utilities

A summary of LAURA utilities is given below. The utility is listed in the left column, and its function is given in the right column.

ARCHIVE	saves the LOCAL, CUSTOM, and input files for the present working (LOCAL) directory
BLOX	exchanges data between the working files (RESTART.in and TWALL.in) and their master files (RESTART.MASTER and TWALL.MASTER)
CUSTOMIZE	moves the LOCAL file to the CUSTOM directory; the CUSTOM file will be used in lieu of the $HOME/LAURA.4.1 version in future LAURA applications
INITIALIZE	accepts a grid from a file in the PLOT3D format, initializes the flow field to the free-stream values, and creates RESTART.in
KEEPER	makes backup copies of files RESTART.in, RESTART.MASTER, TWALL.in, and TWALL.MASTER
LOCALIZE	creates a LOCAL version of the $HOME/LAURA.4.1 file; it is used in lieu of the $HOME/LAURA.4.1 version in this LOCAL directory
RESTORE	restores the archived files to the present working (LOCAL) directory
SIZEIT	estimates the memory requirements of a given application, based on the values specified in the files parameter.strt and assign_tasks
XCUSTOM	eliminates a customized file

Chapter 5

Source Code Installation

LAURA has been successfully tested on SUN, CRAY, SGI, and CONVEX architectures. The first step in applying LAURA to a given problem involves the installation of the source code on the computer where the computations will be performed. The LAURA package consists of two files, `LAURA.4.1.tar.Z` and `INSTALL_LAURA.4.1`, which should be placed in the user's `$HOME` directory. File `LAURA.4.1.tar.Z` should be read-only; if not, type

```
chmod 400 LAURA.4.1.tar.Z
```

The file `INSTALL_LAURA.4.1` should be executable; if not, type

```
chmod 500 INSTALL_LAURA.4.1
```

The source code for LAURA is contained in `LAURA.4.1.tar.Z`, which is a `tarfile` (tape archival file). Typing the command

```
INSTALL_LAURA.4.1
```

executes the relatively straightforward procedure (appendix C), which is outlined below.

1. A subdirectory named `LAURA.4.1` is created in the user's `$HOME` directory.
2. The LAURA source code, contained in the following files, is extracted from `LAURA.4.1.tar.Z` and placed in directory `$HOME/LAURA.4.1`. These files include
   (a) Files with the `start.` root, which contain the source code for `stArt`, the LAURA start-up routine (discussed in chapters 6 and 7)
   (b) The script `PRELUDE` that serves as the front-end to `stArt`
   (c) Files with a `.F` suffix, which contain LAURA subroutines; these files have a `.FOR` suffix on CONVEX architectures
   (d) Files with a `.inc` suffix, which contain additional `FORTRAN` coding (such as `COMMON` blocks); the information in these files is accessed by various LAURA subroutines through `include` statements
   (e) The file `vinokur.data`, which supplies the coefficients for Vinokur's curve-fits
   (f) The file `mAch+prOc.c`, which is a C program used to determine the machine architecture and number of available processors
   (g) The data file `DEFAULTS`, which supplies default values to `stArt`

(h) The script files **Makedep**, **Makedep.awk**, and **SYMLINKS**, which are used by the **Makefile** (appendix E); these files must be executable

(i) The following utilities (which are detailed in the appendixes); these script files must be executable:

ARCHIVE	INITIALIZE	RESTORE
BLOX	KEEPER	SIZEIT
CUSTOMIZE	LOCALIZE	XCUSTOM

(j) The following **FORTRAN** source files:

array.f	flowinit.f	makeblk.f
exchange.f	flowinit.inc	sizeit.f

3. The C program **mAch+prOc** is compiled and run to determine the machine architecture and number of available processors.

4. The **ARCHIVE**, **CUSTOMIZE**, **KEEPER**, **LOCALIZE**, **PRELUDE**, **SYMLINKS**, and **XCUSTOM** scripts are tailored to this architecture.

5. The program **start.f** is compiled to create the executable **stArt**. Also, the executables **ArrAy**, **flOwInIt**, **mAkEblk**, and **sIzEIt** are created from the **FORTRAN** source files listed above.

6. The data file **DEFAULTS** is tailored to this machine.

7. The user's **.cshrc** file is checked to see if aliases for the supporting script files of LAURA have been established. If not, the user is given the option to add aliases for these scripts (**ARCHIVE**, **CUSTOMIZE**, **KEEPER**, **LOCALIZE**, **PRELUDE**, and **XCUSTOM**). These aliases allow execution of these commands from any working (**LOCAL**) directory in the user's account without having to enter the full path name.

At this point the installation procedure is complete.

NOTE: Type

```
source .cshrc
```

to activate the new aliases for this shell.

Chapter 6

Setup for LAURA Application

After installation, the first step in applying LAURA to a specific problem involves creating a working (`LOCAL`) directory, either in scratch space or in a permanent subdirectory. The files in the working directory, are referred to as `LOCAL` files in this manual. Within this directory, `PRELUDE` is executed to create the required infrastructure for LAURA. `PRELUDE` also executes `stArt`, which allows the user to tailor LAURA to an application by selecting items from a series of menus (chapter 7). Defaults for these menu prompts are supplied by the file `DEFAULTS` (section 6.2), while the user inputs are written to the file `INPUTS` (section 6.3). The user-defined choices tailor LAURA through `FORTRAN parameter` statements, compile directives, and `include` files. The file `Makefile` is also constructed based on these selections. As a result, only the code and memory required for this application are activated during compilation. A thorough discussion of `PRELUDE` is presented in section 6.1. Its anatomy is given in appendix D.

For most applications, a successful run of `PRELUDE` is followed by the `make` command, which compiles the LAURA source code to create `laura` (the LAURA executable). The features of the LAURA `Makefile` are discussed in chapter 8. Its anatomy is given in appendix E, section E.1. For certain advanced applications, additional tailoring (chapter 11) may be required after running `PRELUDE`. This involves modifying `LOCAL` copies of LAURA files before compilation, as described in section 6.4.

Chapter 9 discusses actually running LAURA. Specifically, this chapter provides details on the input files that give the user control of a given run. The LAURA output files are described in chapter 10.

6.1. Workspace Layout

The command `PRELUDE` serves as the front-end for `stArt`, which performs the following functions:

- Case-specific include files (with `.strt` suffixes) containing `FORTRAN` coding are created.
- Up to three input data files are created by `stArt` for LAURA, depending on the following user inputs:
  - `data`: This file, which provides control of LAURA, is always generated by `stArt`. The roles of its various entries are discussed in section 9.1.3.
  - `RESTART.in`: This is the restart file for LAURA, which allows the current run to pick up the solution where the previous one left off. For externally generated grids, this file

must be supplied by the user. Otherwise, **stArt** provides a "cold-start" **RESTART.in** file (section 9.1.1).

- **TWALL.in**: This is the restart file for the wall temperature distribution (section 9.1.5). It is only used by LAURA when the "radiative equilibrium wall temperature" option is active, and only created by **stArt** when that particular option is selected by the user.

- The user inputs from this run of **stArt** are saved in files **DEFAULTS** and **INPUTS**.

In addition to activating **stArt**, several other tasks are performed by **PRELUDE**.

- Subdirectories **OBJfiles** and **STRTfiles** are created in the working (**LOCAL**) directory. The **OBJfiles** will ultimately contain the object files and executable file, which are created by **make**. **STRTfiles** will contain the **FORTRAN** files (**.strt** suffixes) created by **stArt**. **PRELUDE** executes **stArt** and then moves the resultant files to the **STRTfiles** directory.

- By default, the **$HOME/LAURA.4.1 stArt** executable is used. However, if any **LOCAL** versions of **stArt** source files exist, compilation is performed to create a **LOCAL** version of **stArt** (if it is outdated). If no **LOCAL stArt** source files exist, the existence of **CUSTOM stArt** source files is considered. If they do exist, a **CUSTOM stArt** executable is compiled (if it requires updating).

- Before **stArt** is executed, if any **LOCAL** case-specific include files (**.strt** suffixes) already exist, the user is given the option to retain them.

- If file **data**, **RESTART.in**, or **TWALL.in** already exists in the working (**LOCAL**) directory, the user is given the option to keep the old file or update it.

With each **PRELUDE** run, **DEFAULTS** supplies the default values (which are echoed to the screen) for the user inputs to **stArt** (section 6.2). The **INPUTS** file reflects only those values selected in the last execution of **PRELUDE** (section 6.3). If for any reason (file corruption or deletion, for example) the user needs to repeat this initialization, the command

PRELUDE INPUTS

reproduces all of the files produced by **stArt** without further user input.

6.2. DEFAULTS File

The default values for the variables of the **stArt** menus are supplied through the **LOCAL** file **DEFAULTS**. These values are echoed to the screen for their respective menus. Initially, this file is copied from **$HOME/LAURA.4.1**. With each execution of **PRELUDE**, **DEFAULTS** is updated to reflect user inputs. This feature provides the user with an on-line reminder of previous values.

> **NOTE:** Acceptance of the default values defined in the **$HOME/LAURA.4.1** version of the **DEFAULTS** file will generate the initial grid and flow field for the case discussed in appendix A.

The first section of **DEFAULTS** contains the LAURA version number as well as the machine time and number of available processors (screen 1). This section is determined during the installation of the code on a given machine. The next section of **DEFAULTS** contains general information, including free-stream conditions.

```
 1            nprocs ..................... number of processors to be used
 2            newjob ........... 0=externally generated, 1=conic, 2=aerobrake
 1            ndim ..................... flow:  1=axisymmetric, 2=2-D, 3=3-D
 1            igovern ................. fluid eqns:  0=Euler, 1=TL N-S, 2=N-S
5000.00       vinfb ..................................... velocity [m/s]
0.100000e-02  rinfb ..................................... density [kg/m^3]
200.000       tinf ............................ freestream temperature [K]
0.            attack ................................. angle of attack [deg]
0.            yaw ..................................... angle of yaw [deg]
 0            tempbc . Tw BC:  0=constant, 1=variable, 2=radiative equilibrium
500.000       twall ................................... wall temperature [K]
0.            ept ....................... wall temperature relaxation factor
```

Screen 1.

NOTE: In version 4.1 of LAURA, the yaw angle is set to $yaw = 0$.

The next section (screen 2) concerns the gas model.

```
 0          ngas ........................... gas model:  0=PG, 1=EQ, 2=NONEQ
 0          icrv ........................ EQ model:  1=Vinokur, 2=Tannehill
 0          itherm ............. 1=equilibrium (1-T), 2=nonequilibrium (2-T)
n           answern .................. species:  atomic nitrogen (y/n)?....
n           answero .................. species:  atomic oxygen.............
n           answern2 ................. species:  molecular nitrogen........
n           answero2 ................. species:  molecular oxygen..........
n           answerno ................. species:  nitric oxide..............
n           answernp ................. species:  ionized atomic nitrogen...
n           answerop ................. species:  ionized atomic oxygen.....
n           answern2p ................ species:  ionized molecular nitrogen
n           answero2p ................ species:  ionized molecular oxygen..
n           answernop ................ species:  ionized nitric oxide......
 0          jtype ................................ catalytic nature of wall
 0          nturb ....... turbulence:  0=no, 1=Cebeci-Smith, 2=Baldwin-Lomax
```

Screen 2.

The third section (screen 3) contains default values to the controls for the computational domain.

```
 1          nblocks ..................... number of computational blocks
30          iblk( 1) ..................... block  1:  cells in i-direction
 1          jblk( 1) .....................        1:  cells in j-direction
64          kblk( 1) .....................        1:  cells in k-direction
 4          itype(1, 1) ............. block  1:  nature of i =   1 boundary
 1          itype(2, 1) .............        1:  nature of i =  30 boundary
 5          itype(3, 1) .............        1:  nature of j =   1 boundary
 5          itype(4, 1) .............        1:  nature of j =   1 boundary
 0          itype(5, 1) .............        1:  nature of k =   1 boundary
 3          itype(6, 1) .............        1:  nature of k =  64 boundary
```

Screen 3.

NOTE: Initially, defaults are assigned for a single computational block. If more than one block is specified in **stArt**, the updated **DEFAULTS** file will contain the dimensions and boundary conditions for these additional blocks.

The final section (screen 4) contains the geometry information.

```
 0          iunit ..................... units:  0=m, 1=cm, 2=ft, 3=in, 4=__
m           unit ........... 1- or 2-character abbreviation for custom units
1.00000     rflngth ..................... conversion factor for custom units
0.          xcg ..................................................... x-cg [m ]
0.          zcg ..................................................... z-cg [m ]
3.14159     refarea ................................. reference area [m ^2]
2.00000     reflen ................................... reference length [m ]
 1          ndimb ............ body dimension:  1=axisymmetric, 2=2-D, 3=3-D
20          ic ..................................... number of cells on cap
1           konic .... {1=hyper, 2=para}boloid, {3=elliptic-, 4=sphere-}cone
0.          thc ............................. half-angle [deg] of asymptote
1.00000     b ............................... axial shape parameter for nose
1.00000     rnose ........................................ nose radius [m ]
1.00000     rxz ........................ nose radius [m ] in symmetry plane
1.00000     zmax ........................................ body length [m ]
1.00000     axfac ........................ axial stretching factor for grid
 1          iafe .......... aerobrake option:  0=AFE, 1=hemisphere, 2=custom
1.00000     scale ................................... aerobrake scale factor
 0          thetaxy ................................. body half-angle [deg]
90.0000     delta ....................................... rake angle [deg]
0.          tau ............................. shoulder turning angle [deg]
0.          radius ..................................... shoulder radius [m ]
1.00000     epsib ..................................... eccentricity of nose
1.00000     rbase ................................... base plane radius [m ]
```

Screen 4.

6.3. INPUTS File

Each time PRELUDE is successfully run, an INPUTS file is created which contains the user inputs of that session. As such, it is a subset of the DEFAULTS file, since no single run will reset all of the default values. (For example, if perfect gas flow is chosen, the user does not choose an equilibrium curve-fit.) To illustrate, the INPUTS file (screen 5) for appendix A is shown below.

```
2              newjob ..... 0=externally generated, 1=conic, 2=aerobrake
1              ndim ..... flow:  1=axisymmetric, 2=2-D, 3=3-D
1              igovern ..... fluid eqns:  0=Euler, 1=TL N-S, 2=N-S
5000.00        vinfb ..... velocity [m/s]
0.100000e-02   rinfb ..... density [kg/m^3]
200.000        tinf ..... freestream temperature [K]
0              tempbc . Tw BC:  0=constant, 1=variable, 2=radiative equilibrium
500.000        twall ..... wall temperature [K]
0              ngas ..... gas model:  0=PG, 1=EQ, 2=NONEQ
0              nturb ..... turbulence:  0=no, 1=Cebeci-Smith, 2=Baldwin-Lomax
30             iblk( 1) ..... cells in streamwise/axial direction
64             kblk( 1) ..... cells in normal direction (maximum)
 1             iafe ..... aerobrake option:  0=AFE, 1=hemisphere, 2=custom
1.00000        scale ..... aerobrake scale factor
```

Screen 5.

This example employs the self-starting capability for axisymmetric flow about a sphere-cone (specified in ft). The thin-layer Navier-Stokes equations will be solved. The perfect gas, the laminar flow, and a constant wall temperature are specified. The command

```
PRELUDE INPUTS
```

instructs PRELUDE to accept user inputs from file INPUTS, rather than from the screen.

> **NOTE:** Some changes to file INPUTS can result in a different question and answer sequence in stArt. For example, to change from *icrv* = 1 to *icrv* = 2, a user can elect to simply change its value in file INPUTS and type the command
>
> ```
> PRELUDE INPUTS
> ```
>
> However, this approach will not work for switching from *ndim* = 1 to *ndim* = 2, because PRELUDE prompts for angle of attack with two-dimensional flows, but it does not prompt for axisymmetric flows. Therefore, it is recommended that only the experienced user (i.e., one familiar with the prompting sequence of stArt) attempt to modify file INPUTS, followed by the command
>
> ```
> PRELUDE INPUTS
> ```
>
> All others should type the command
>
> ```
> PRELUDE
> ```
>
> and provide inputs to PRELUDE directly from the screen.

6.4. Tailoring the LAURA Algorithm

For most applications, the necessary tailoring of the LAURA algorithm for a given case can be accomplished through PRELUDE. As mentioned earlier, however, PRELUDE may not provide sufficient tailoring of LAURA for some advanced applications. For such instances, there are several ways to further specialize the LAURA algorithm (without altering the $HOME/LAURA.4.1 files); these methods are to

- Create LOCAL versions of the include files created by stArt (.strt suffixes) using the LOCALIZE command (appendix K)
- Create LOCAL versions of the LAURA source files located in $HOME/LAURA.4.1 by using the LOCALIZE command (appendix K)
- Create LOCAL pure-FORTRAN versions of source files (section 8.3) by using the LOCALIZE command (appendix K)
- Create CUSTOM files from LOCAL versions of source files by using the CUSTOMIZE command (appendix H)

These approaches are discussed below.

A number of LAURA's FORTRAN parameters are automatically assigned values by stArt, without user input, and are written to files with a .strt suffix. Any of these parameters can be changed by modifying a LOCAL copy of the appropriate lfn file, which is created using the command

```
LOCALIZE lfn
```

For example, stArt defaults to first-order extrapolations at the body surface and outflow boundaries. To switch from first to second order (via parameter *nordbc* in file nordbc_assn.strt), type the command

```
LOCALIZE nordbc_assn.strt
```

and a copy of file nordbc_assn.strt will be created in the working directory (appendix K). Now edit the LOCAL file, and change "*nordbc*= 1" to "*nordbc*= 2". When make is executed, this LOCAL file will be used in lieu of the STRTfiles version. Moreover, in future runs of PRELUDE within this working directory, the user will be given the option to save this LOCAL file or to overwrite it with the file created by stArt.

In other cases, FORTRAN coding can be added to, deleted from, or modified in the LAURA source files. Tailoring these LAURA source files (.F, .FOR, and .inc suffixes) is done in the same manner as above: simply use the LOCALIZE command on a given file from the $HOME/LAURA.4.1 directory and gain the desired modifications to this LOCAL file before compilation.

Pure-FORTRAN versions (.f suffixes) of the baseline subroutine files (.F suffixes) can be created by the command

```
make fortran
```

This command first creates the directory FORTRAN (if it does not already exist), and then preprocesses each of the subroutine files (from the $HOME/LAURA.4.1, CUSTOM, or LOCAL directories). Each of the preprocessed files is placed in the FORTRAN directory (with a .f suffix instead of a .F or .FOR suffix). As before, tailoring of these files is accommodated through the LOCALIZE command; this is followed by modification of the resultant LOCAL file.

The CUSTOMIZE command allows a LOCAL file in one working directory to be used for any LAURA application (without having to be present in that LOCAL directory) in lieu of the

`$HOME/LAURA.4.1` file. A `CUSTOM` version of the file `lfn` is created in the following manner. After making the modifications to `LOCAL` version of `lfn`, type the command

```
CUSTOMIZE lfn
```

The file will be moved to subdirectory `CUSTOM` in directory `$HOME/LAURA.4.1` (appendix H). Future executions of `make` from any directory will use this customized version of `lfn` in lieu of the original LAURA coding.

> **NOTE:** To defeat this customization in a given working directory, simply copy the baseline version of `lfn` from `$HOME/LAURA.4.1` before executing `make`. This `LOCAL` copy of the baseline `lfn` will be used instead of the customized version.

The command

```
XCUSTOM lfn
```

deletes the `CUSTOM` version of `lfn` (appendix N).

> **NOTE:** Do not use `CUSTOMIZE` on `LOCAL` versions of files created by `stArt` (`.strt` suffix).

Chapter 7

Menus of stArt

In this section, the various menus of **stArt** are presented and annotated. The menus are presented in their order of appearance during **stArt**. Each menu is enclosed within a shaded box to mimic what the user will see on the computer screen. The default value for each variable, which reflects the choice from the last **PRELUDE** session, is also shown.

> **NOTE:** The user can enter a comma (",") at any prompt to accept the default value.

7.1. Number of Processors

For CRAY architectures, the upper limit of processors that will be used during this LAURA run is specified through the variable *nprocs*:

```
Enter the upper limit of processors to be used during this LAURA run.

Enter choice (1 <= nprocs <= nprocmx) {default}:
```

where *nprocmx* is the number of available processors on this machine, as determined by **mAch+prOc** (appendix C, section C.2).

> **NOTE:** On CRAY machines, a single processor (*nprocs* = 1) should be used for LAURA runs where grid adjustments will be made. (See NOTE in section 11.5.3.) This is required because the multiprocessing is asynchronous.

7.2. Type of Initial Grid and Solution

The user has a number of options for automatic generation of a surface and volume grid, including an initialized solution (section 9.1.1), for specific parameterized body shapes. Alternately, the user can supply an externally generated restart file to LAURA (section 9.1.1). The type of initial grid and solution is specified through *newjob* (screen 6):

```
Select initialization:

   0) use existing "RESTART.in" file,
   1) create conic (cone/wedge, paraboloid, etc.),
   2) create generic aerobrake
      (includes AFE without axis singularity).

Enter choice {default}:
```

Screen 6.

NOTE: When *newjob* = 0 is selected, `stArt` will issue a warning if file `RESTART.in` does not exist in the `LOCAL` directory. Since `stArt` itself does not use this file, the user can continue with this `PRELUDE` session and create `RESTART.in` afterwards. The message simply serves as a reminder that `RESTART.in` must exist before executing `laura`.

7.3. Flow Dimensionality

The flow dimensionality is specified through *ndim* (screen 7):

```
Select flow dimensionality:

   1) axisymmetric flow,
   2) two-dimensional flow,
   3) three-dimensional flow.

Enter choice {default}:
```

Screen 7.

7.4. Governing Equations

LAURA provides inviscid and viscous flow options. The viscous flow options include thin-layer and full Navier-Stokes equations (which require more memory for treatment of the cross derivatives and associated metrics). Select the governing equations through *igovern* (screen 8):

```
Select governing equations:

   0) inviscid flow,
   1) thin-layer Navier-Stokes,
   2) full Navier-Stokes.

Enter choice {default}:
```

Screen 8.

The thin-layer option includes only the viscous terms defined by gradients in the coordinate directions emanating from wall boundaries. The viscous directions are set automatically as boundary conditions and are input with the variables $ivis_{nblk}$, $jvis_{nblk}$, and $kvis_{nblk}$ equal to 0 for off and 1 for on in computational block *nblk* (section 9.1.3.1). The user can change the default values in the file `data` (which is created by `stArt`).

> **NOTE:** If *igovern* = 1, defining *ivis* = *jvis* = *kvis* = 1 for a given block still omits the cross derivative terms from the full Navier-Stokes equations. The user must specify *igovern* = 2 in `stArt` to include the full Navier-Stokes terms in the compilation of `laura`.

7.5. Free-stream Conditions

Enter the free-stream conditions (screen 9) (in mkgs units):

```
Enter velocity [m/s] {default}:

Enter density [kg/m^3] {default}:

Enter freestream temperature [K] {default}:
```

Screen 9.

For two- or three-dimensional flow, the angle of attack (*attack*) must be specified (screen 10):

```
Enter angle of attack [deg] {default}:
```

Screen 10.

7.6. Surface Boundary Conditions

The surface boundary conditions are set internally for the momentum and global continuity equations.

> **NOTE:** For nonequilibrium flows (where the global continuity equation is not solved explicitly), the wall catalysis must be chosen to define the surface boundary conditions for the species continuity equations (section 7.7.2.3).

> **NOTE:** The inviscid boundary conditions are crude; they extrapolate pressure, temperature, and tangential velocity components to the wall and reflect a normal velocity component.

For viscous flow ($igovern \neq 0$), a wall boundary condition on the energy equation must be specified. Prescribe this wall temperature boundary condition through *tempbc* (screen 11):

```
Select wall temperature BC:

   0) constant
   1) specified variation          (ndim ≠ 3)
   2) radiative equilibrium

Enter choice {default}:
```

Screen 11.

The follow-up prompts for $tempbc = 0$, 1, and 2 are discussed in sections 7.6.1, 7.6.2, and 7.6.3, respectively.

7.6.1. Constant Wall Temperature

When $tempbc = 0$, input the wall temperature (screen 12), as follows:

```
Enter wall temperature [K] {default}:
```

Screen 12.

7.6.2. Specified Wall Temperature Variation

When *tempbc* = 1, the wall temperature distribution is provided in file **variabletw**. This allows the surface temperature to be specified by a fixed distribution (from experimental data, for example). The entries in file **variabletw** are the streamwise surface distance (in *units*) and temperature (K) for each location. They are read in free format (one entry per line) by **stArt** (and later by LAURA). In **reload.F**, these discrete values are linearly interpolated to provide values at cell face centers along the surface.

> **NOTE:** If file **variabletw** is not found by **stArt**, the free-stream temperature is used to initialize the flow field temperature distribution. This "fix" allows this **stArt** session to continue, but the initial temperature distribution is less than ideal. As a result, the user is strongly urged to create the file **variabletw** and repeat the start-up procedure using the command
>
> PRELUDE INPUTS
>
> **NOTE:** The specified wall temperature variation (*tempbc* = 1) option is currently not available for three-dimensional flows.

7.6.3. Radiative Equilibrium Wall Temperature

When *tempbc* = 2, an initial guess for the wall temperature must be supplied, along with the relaxation factor, *epl* (screen 13):

```
Enter initial wall temperature [K] {default}:

Enter wall-temperature relaxation factor {default}:
```

Screen 13.

A value of $epl < 1$ is an under-relaxation (with $epl = 0$ fixing T_w at the input value) and $epl = 1$ is a straight substitution. In the early stages of convergence, it is advisable to keep $epl = \mathcal{O}(0.01)$. This value can be increased (in file **data**) as the equilibrium value of T_w is approached.

7.7. Gas Model

LAURA has perfect gas, equilibrium, and nonequilibrium flow capabilities for air. Define the nature of the gas through *ngas* (screen 14):

```
Select the gas model:

   0) perfect gas,
   1) equilibrium,
   2) chemical nonequilibrium.

Enter choice {default}:
```

Screen 14.

If $ngas = 0$ is specified, no other user input is required for the gas model.

> **NOTE:** The perfect gas is assumed to be air, but constants appropriate for other gases can be specified in the include files `gas_model_vars.strt` and `sthrlnd_vars.strt`.

Subsequent screen prompts for equilibrium and nonequilibrium gas model are discussed in sections 7.7.1 and 7.7.2, respectively.

7.7.1. Equilibrium

For equilibrium flow ($ngas = 1$), choose which thermodynamic curve fit to use through *icrv* (screen 15):

```
Select a thermodynamic curve fit:

   1) Vinokur,
   2) Tannehill.

Enter choice {default}:
```

Screen 15.

Details of these curve-fits can be found in references 19 and 20, respectively.

7.7.2. Nonequilibrium

For nonequilibrium flows, the thermal state, constituent species, and wall catalysis must be specified. These constraints are discussed in the subsections that follow.

7.7.2.1. Thermal State

For nonequilibrium flow ($ngas = 2$), choose the thermal state of the gas through *itherm* (screen 16):

```
Select the thermal state of the gas:

   1) equilibrium    (one-temperature),
   2) nonequilibrium (two-temperature).

Enter choice {default}:
```

Screen 16.

7.7.2.2. Constituent Species

Next, select which species will be included in the air model (screen 17):

```
Will atomic nitrogen be considered?  (y/n) {default}:

Will atomic oxygen be considered?  (y/n) {default}:

Will molecular nitrogen be considered?  (y/n) {default}:

Will molecular oxygen be considered?  (y/n) {default}:

Will nitric oxide be considered?  (y/n) {default}:

Will ionized atomic nitrogen be considered?  (y/n) {default}:

Will ionized atomic oxygen be considered?  (y/n) {default}:

Will ionized molecular nitrogen be considered?  (y/n) {default}:

Will ionized molecular oxygen be considered?  (y/n) {default}:

Will ionized nitric oxide be considered?  (y/n) {default}:
```

Screen 17.

An answer of "y" activates the species, while an answer of "n" deactivates it. Thus, any subset of the total 11 species for air (N, O, N_2, O_2, NO, N^+, O^+, N_2^+, O_2^+, NO^+, e^-) can be specified.

7.7.2.3. Wall Catalysis

Physical models within LAURA are appropriate for weak ionization. Free-stream mass fractions appropriate for undissociated, low-temperature air are set in block data file `air.F` and can be adjusted as required. The catalysis of the wall is defined through the variable *jtype* (screen 18):

```
Select the catalytic nature of the wall boundary:

   0) non-catalytic;
   1) "super-catalytic";
   2) catalytic to ions, non-catalytic to neutrals;
   3) catalytic to ions, Stewart's finite-catalysis;
   4) catalytic to ions, Zoby's finite-catalysis;
   5) catalytic to ions, Scott's finite-catalysis;
   6) catalytic to ions, homogeneous recombination of all atoms.

Enter choice {default}:
```

Screen 18.

As shown, options for noncatalytic, finite-catalytic, and "super-catalytic" wall conditions are available.

> **NOTE:** The super-catalytic condition (*jtype* = 1) sets the mass fractions at the wall to their free-stream values. This should not be confused with a fully catalytic condition where the surface mass fractions are defined by the equilibrium composition at the given wall temperature.

Details of the *jtype* = 3, 4, and 5 options can be found in references 21, 22, and 23, respectively.

> **NOTE:** In theory, for inviscid flow, an extrapolation of the near-wall mass fractions to the wall provides the proper catalytic nature of this boundary. In the absence of this option, selecting the noncatalytic boundary condition (*jtype* = 0) is the best choice.

7.8. Turbulence

Algebraic models are employed within LAURA to provide a turbulence capability for perfect gas, equilibrium, and nonequilibrium flows. Control this option through *nturb* (screen 19):

```
Select laminar flow or turbulence model:

   0) laminar flow,
   1) Cebeci-Smith,
   2) Baldwin-Lomax

Enter choice {default}:
```

Screen 19.

Details of the Cebeci-Smith and Baldwin-Lomax algebraic models can be found in references 24 and 25, respectively.

> **NOTE:** A well-defined boundary-layer edge is required before these algebraic models can be implemented. Thus, for a given case, a laminar-flow solution is necessary as a starter for turbulent flow. The laminar solution does not need to be fully converged, but the shock layer should be well-developed.

> **NOTE:** A value of 0.9 is assumed for the turbulent Prandtl number. For nonequilibrium flows, a turbulent Schmidt number is required as well, and a value of unity is used.

Specification of transition is discussed in section 9.1.4.

7.9. Flow Field Grid and Initialization

LAURA has an internal grid generation and initialization algorithm that provides the user with a self-starting capability. This feature is limited to select bodies that can be described analytically. However, LAURA will accept externally generated grids and flow field initializations. This option allows computations over arbitrary bodies and the use of more sophisticated gridding techniques. Use of such external grids is discussed in section 7.9.1. Self-starting grids are the topic of section 7.9.2.

7.9.1. Externally Generated Grid and Initialized Flow

When *newjob* = 0 is specified, the user must supply the file `RESTART.in` for `laura`. The format of this restart file is given in section 9.1.1. This file does not need to exist before running `PRELUDE`, but it must be present before executing `laura`. Required inputs for an externally generated `RESTART.in` are discussed in the subsections below.

In `laura`, the grid must be oriented such that $y = 0$ defines the plane of symmetry. In the assumed orientation, the z-axis originates at the vehicle nose and is directed toward the oncoming flow (as shown in fig. 2.1). Such an orientation is common for blunt-body applications. The angles of attack and yaw are referenced to the negative z-axis, which yields the following definitions for the free-stream velocity components:

$$
\begin{aligned}
uinf &= \sin(attack) * \cos(yaw) \\
vinf &= \sin(yaw) \\
winf &= -\cos(attack) * \cos(yaw)
\end{aligned}
$$

NOTE: In `laura`, the yaw angle is set to $yaw = 0$.

The user will probably encounter externally generated grids with other orientations. In such cases, it is recommended that the user re-orient the grid to conform to the examples shown in figures 2.1 and 2.2. Afterwards, the utility `INITIALIZE` can be used to initialize the flow field and create the LAURA `RESTART.in` file (appendix I).

If the user prefers not to re-orient the grid, then the definitions of the free-stream velocity components can be modified. This approach involves creating `LOCAL` versions of files `setup.F` and `flowinit.f` with the `LOCALIZE` utility (appendix K). The appropriate modifications to the definitions of *uinf*, *vinf*, and *winf* can be made in these `LOCAL` files. When the utility `INITIALIZE` is executed, it will detect the `LOCAL` version of `flowinit.f`, compile it, and use the resultant `LOCAL flOwInIt` executable to initialize the given `PLOT3D` grid file.

> **NOTE:** The `laura` executable must be recompiled to get the new definitions from the `LOCAL setup.F`.

Alternately, the user can employ the default definitions for the free-stream velocity components in conjunction with other grid orientations as long as the angle of attack is referenced to the negative z-axis. Consequently, a value of $attack = 0$ in the default orientation can correspond to a value of -90, 90, or 180 *deg* (specified in file "`data`") for this alternative orientation.

7.9.1.1. Number of Computational Blocks

The flow field domain can be divided into multiple computational blocks. The number of blocks is defined through the variable *nblocks*:

```
Input the number of computational blocks ( 1 <= nblocks <= 6 ) {default}:
```

7.9.1.2. Dimensions for Blocks

For each block *nblk* ($1 \leq nblk \leq nblocks$), input the variables $iblk_{nblk}$, $jblk_{nblk}$ (3-D only), and $kblk_{nblk}$, which define the number of computational cells in this block (screen 20):

```
For block nblk, input the number of cells ...

   ...  in the i-direction {default}:

   ...  in the j-direction {default}:                    (3-D flow only)

   ...  in the k-direction {default}:
```

Screen 20.

NOTE: In each direction, the number of cells is one less than the number of grid points (cell walls).

7.9.1.3. Boundary Condition Types for Block Faces

If *newjob* = 0, the boundary type (such as a solid surface or a free stream) of the faces of each block must be specified by the user. The boundary type for each of the six sides of a given block is defined through the variable *itype* (screen 21):

```
Select the nature of the i = 1 boundary in block nblk:
```

```
Select the nature of the i = iblk_nblk boundary in block nblk:
```

```
Select the nature of the j = 1 boundary in block nblk:
```

```
Select the nature of the j = jblk_nblk boundary in block nblk:
```

```
Select the nature of the k = 1 boundary in block nblk:
```

```
Select the nature of the k = kblk_nblk boundary in block nblk:

   0) solid surface,
   1) outflow,
   2) symmetry across y = 0 plane (3-D flow),
   3) freestream,
   4) symmetry across x = 0 plane (axisymmetric or 2-D flow),
   5) symmetry across y = 0 plane (axisymmetric or 2-D flow),
   7) axis (for 3-D flow),
   9) boundary shared with another block.

Enter choice {default}:
```

Screen 21.

NOTE: For brevity, the option list is shown on the $k = kblk_{nblk}$ screen only.

If a shared boundary is specified, the block with which the boundary is common must be specified:

```
Enter the number of that block {default}:
```

NOTE: If $newjob \neq 0$, a single block is created, and default values for the boundary types of the faces of this block are defined automatically. No user specification is required.

7.9.2. Self-Starting Grid and Flow Initialization

When $newjob \neq 0$, `stArt` generates the `RESTART.in` file. Required user inputs for this option are discussed in the subsections that follow.

7.9.2.1. Number of Computational Blocks

The geometry and flow field grids are generated by using a single computational block, which is based on user inputs that are discussed later.

7.9.2.2. Dimensions for Computational Block

First, the size of the grid is defined through *iaq*, *jaq* (3-D only), and *kaq* (screen 22):

```
Input iaq, the number of cells in the streamwise direction
along the body.

Enter choice (1 <= iaq <= iaqm) {default}:


Input jaq, the number of cells in the circumferential direction
around the body.

Enter choice (1 <= jaq <= jaqm) {default}:              (3-D flow only)


Input kaq, the maximum number of cells normal to the body.

Enter choice (1 <= kaq <= kaqm) {default}:
```

Screen 22.

where $iaqm = 200$, $jaqm = 100$, and $kaqm = 128$.

> **NOTE:** If these values are too restrictive, the user can create a `LOCAL` copy of the file `start.inc` (using the `LOCALIZE` command) and modify these limits. When `PRELUDE` is executed, this `LOCAL` file will be used to create a `LOCAL stArt` executable, which is then used by `PRELUDE` in lieu of the default executable. `PRELUDE` accommodates `LOCAL` copies of any of the `stArt` source files (located in the `$HOME/LAURA.4.1` directory), if the user desires to modify `stArt` to produce tailored case-specific `include` files (`.strt` suffixes).

A single computational block (`Block A`) is utilized, where $iblk_1 = iaq$, $jblk_1 = jaq$, and $kblk_1 = kaq$. For two-dimensional and axisymmetric flows, $jblk_1 = 1$.

If options for grid adaption and alignment with the captured bow shock are to be exercised, then $kblk_1$ must be the number of cells between the body and the free-stream inflow boundary. Furthermore, $kblk_1$ should be divisible by 4 if options for coarse-grid initialization of the solution

are implemented, as is the case when any of the self-start geometry initialization packages are employed.

> **NOTE:** For *newjob* = 2 and *ndim* = 3, the number of circumferential cells (*jaq*) is defined as one-half of the number of streamwise cells (*iaq*). If the input value for *iaq* is not a multiple of 2, it is overwritten to *iaq* = 2 × *jaq*. The resultant values of *iaq* and *jaq* are then echoed to the screen.

7.10. Geometry Definition

LAURA is written using mkgs units. However, other units of length can be accommodated through the conversion factor *rflngth*. The value of *rflngth* (and hence, *units*) for several common choices is controlled through *iunit* (screen 23):

```
Select units for the geometry:

   0) meters,
   1) centimeters,
   2) feet,
   3) inches,
   4) other units.

Enter choice {default}:
```

Screen 23.

If *iunit* ≠ 4, the conversion factor from grid units to meters (*rflngth*) is automatically loaded. For instance, if the surface geometry and volume grid are defined in inches, then *units* = in. and *rflngth* = 0.0254 (since 1 in. = 0.0254 m).

For *iunit* = 4, the user must explicitly define *units* and *rflngth* (screen 24):

```
Enter 1- or 2-character abbreviation for this custom unit
of length {default}:

The conversion factor "rflngth" is defined such that:

      1 units = "rflngth" meters

Enter "rflngth" {default}:
```

Screen 24.

This feature allows the user to define a volume grid based on a nonstandard metric. For example, a generic blunted-cone might be defined in units of nose radius (R_N). If R_N = 2 ft and the volume grid was created in units of nose radius, then *units* = *RN* and *rflngth* = 0.6096 (since 1 *RN* = 2 ft = 0.6096 m). Proper specification of this factor is important because it is used to define the Reynolds number per unit grid length in the solution.

Required geometry inputs for externally supplied grids are described in section 7.10.1. Self-starting conic geometries (such as cones, paraboloids, and hyperboloids) are discussed in section 7.10.2. Self-starting aerobrakes (such as the Aeroassist Flight Experiment (AFE) and spheres) are the focus of section 7.10.3.

7.10.1. Externally Generated Grid

When *newjob* = 0, the volume grid (and initial flow field) must be provided by the user. Section 11.1 gives restrictions on the grid orientation for two-dimensional and axisymmetric flows.

> **NOTE:** The self-starting feature of LAURA can be employed to produce grids for flows over simple analytic shapes (section 7.10.2).

Several metrics of the geometry must be supplied by the user when the *newjob* = 0 option is chosen. The variables *xcg* and *zcg* define the location of the reference center for aerodynamic moments in grid coordinates (screen 25):

```
Enter x-coordinate of moment center [units] {default}:

Enter z-coordinate of moment center [units] {default}:
```

Screen 25.

NOTE: It is implicitly assumed that the flow has $y = 0$ as its plane of symmetry with $ycg = 0$.

The variables *refarea* and *reflen* define the reference area and length, respectively, for the evaluation of aerodynamic coefficients (screen 26):

```
Enter reference area for aerodynamics [units^2] {default}:

Enter reference length for aerodynamics [units] {default}:
```

Screen 26.

NOTE: For self-starting grids ($newjob \neq 0$), the values of *xcg*, *zcg*, *refarea*, and *reflen* are defined by `stArt` (along with $ycg = 0$). The user can change these values in file `data`.

7.10.2. Self-Starting Grid (Conic Geometry)

When $newjob = 1$, the conic geometry must be defined through a series of user inputs. Based on these values, *xcg*, *zcg*, *refarea*, and *reflen* are loaded by `stArt` (along with $ycg = 0$). For $ndim < 3$ (axisymmetric or two-dimensional flow), the body dimensionality is equal to the flow dimensionality ($ndimb = ndim$). If $ndim = 3$, then the dimensionality of the body must be specified through the variable *ndimb* (screen 27):

```
Select body dimensionality:

   1) axisymmetric,
   3) 3-D.

Enter choice {default}:
```

Screen 27.

NOTE: The option of $ndimb = 2$ is eliminated in this case because the configuration cannot be two-dimensional when $ndim = 3$.

The type of conic section is prescribed through the variable *konic*. The options for *konic* are a function of *ndimb*. The next three sections show the options for *konic* based on the value of the body dimensionality (*ndimb*).

7.10.2.1. Axisymmetric Geometry

For *ndimb* = 1, the options for *konic* (screen 28) are:

```
Select axisymmetric body:

   1) hyperboloid,
   2) paraboloid,
   3) ellipsoidally-blunted cone,
   4) spherically-blunted cone.

Enter choice {default}:
```

Screen 28.

For *konic* > 2, the value of *ic* must be specified (screen 29):

```
Input the number of cells (axial direction) to be used on the cap.

Enter choice (2 <= ic <= iaq) {default}:
```

Screen 29.

Unless *konic* = 2, the variable θ must be specified. For *konic* = 1, this is the half-angle of the asymptote of the hyperboloid:

```
Enter half-angle [deg] of asymptote {default}:
```

For *konic* = 3 and *konic* = 4, θ is the body half-angle:

```
Enter body half-angle [deg] {default}:
```

For *konic* = 3, the axial shape parameter of the nose (b), which is a function of the eccentricity, must be defined (appendix B):

```
Enter axial shape parameter for nose, b
( b > 0, where b = 1 is circle ) {default}:
```

NOTE: For *konic* = 4, $b = 1$ automatically.

The next step is to specify the nose radius and total body length of the geometry:

```
Enter nose radius [units] {default}:

Enter body length [units] {default}:
```

7.10.2.2. Two-Dimensional Geometry

For *ndimb* = 2 (only possible for *ndim* = 2), the options for *konic* are:

```
Select 2-D body:

   1) hyperbola,
   2) parabola,
   3) blunted wedge.

Enter choice {default}:
```

For *konic* = 3, the value of *ic* must be specified:

```
Input the number of cells (axial direction) to be used on the cap.

Enter choice (2 <= ic <= iaq) {default}:
```

For *konic* = 1 or *konic* = 3, the variable θ must be specified. For *konic* = 1, this is the half-angle of the asymptote of the hyperbola:

```
Enter half-angle [deg] of asymptote {default}:
```

For *konic* = 3, θ is the body half-angle, and must be specified. Also for *konic* = 3, the axial shape parameter for the nose (b) must be defined (appendix B):

```
Enter body half-angle [deg] {default}:

Enter axial shape parameter for nose, b
( b > 0, where b = 1 is circle ) {default}:
```

The next step is to specify the nose radius and total body length of the geometry:

```
Enter nose radius [units] {default}:

Enter body length [units] {default}:
```

7.10.2.3. Three-Dimensional Geometry

For *ndimb* = 3, the options for *konic* are as follows:

```
Select 3-D body:

   1) hyperboloid,
   2) paraboloid,
   3) blunted cone.

Enter choice {default}:
```

For *konic* = 3, the value of *ic* must be specified:

```
Input the number of cells (axial direction) to be used on the cap.

Enter choice (2 <= ic <= iaq) {default}:
```

Unless *konic* = 2, the variable θ must be specified. For *konic* = 1, this is the half-angle of the asymptote of the hyperboloid:

```
Enter half-angle [deg] of side-plane asymptote {default}:
```

For *konic* = 3, θ is the body half-angle, and must be specified. Also for *konic* = 3, the axial shape parameter for the nose (b) must be defined (appendix B):

```
Enter body half-angle [deg] in side plane {default}:

Enter axial shape parameter for nose, b, in side plane.
( b > 0, where b = 1 is sphere ) {default}:
```

The next step is to specify the nose radii, in both the symmetry and side planes, along with the total body length of the geometry:

```
Enter nose radius [units] in side plane {default}:

Enter nose radius [units] in symmetry plane {default}:

Enter body length [units] {default}:
```

7.10.2.4. Axial Stretching Factor

As a final step for *konic* ≥ 3 (for all values of *ndimb*), the axial stretching factor (*axfac*) for the grid must be defined. A message to the screen provides the user with the minimum value of *axfac* required to reach the end of the geometry, based on the previously specified values for *ic*, *iaq*, and *zmax*. This value is the default answer to the following prompt:

```
Enter axial stretching factor {default}:
```

A value of *axfac* = 1 yields no axial stretching. To prevent overstretching, values larger than approximately *axfac* = 1.2 should not be used.

7.10.3. Self-Starting Grid (Generic Aerobrake)

When *newjob* = 2, the aerobrake geometry must be defined through a series of user inputs. Based on these values, *xcg*, *zcg*, *refarea*, and *reflen* are loaded by `stArt` (along with *ycg* = 0). First, the type of aerobrake geometry is selected through variable *iafe*:

```
Select aerobrake geometry option:

   0) AFE aerobrake,
   1) hemisphere,
   2) customized aerobrake.

Enter choice {default}:
```

7.10.3.1. AFE Aerobrake

When *iafe* = 0, the aerobrake parameters are automatically set to the AFE configuration (ref. 26) values.

NOTE: Because the AFE geometry is three-dimensional, a specification of *iafe* = 0 can only be made if *ndim* = 3.

7.10.3.2. Hemisphere

When *iafe* = 1, the nose radius must be specified (in meters):

```
Enter radius [m] {default}:
```

7.10.3.3. Customized Aerobrake

When *iafe* = 2, the parameters that are defined implicitly for *iafe* = 0 must be explicitly supplied by the user (screen 30).

NOTE: Accepting the echoed default values will yield the AFE geometry.

A graphical representation of a typical body in the plane of symmetry, including explanations of the various parameters, is provided in figure 7.1.

```
Enter body half-angle [deg] in symmetry plane {default}:

Enter rake angle [deg] of base plane relative to body axis
(enter "90" for axisymmetric body) {default}:          (3-D flow only)


Enter turning angle [deg] of shoulder
(usually same as body half-angle) {default}:


Enter shoulder radius [units] {default}:


Enter eccentricity of nose {default}:                  (3-D flow only)


Enter nose radius [units] of blunted cone in symmetry plane {default}:


Enter base plane radius [units] {default}:
```

Screen 30.

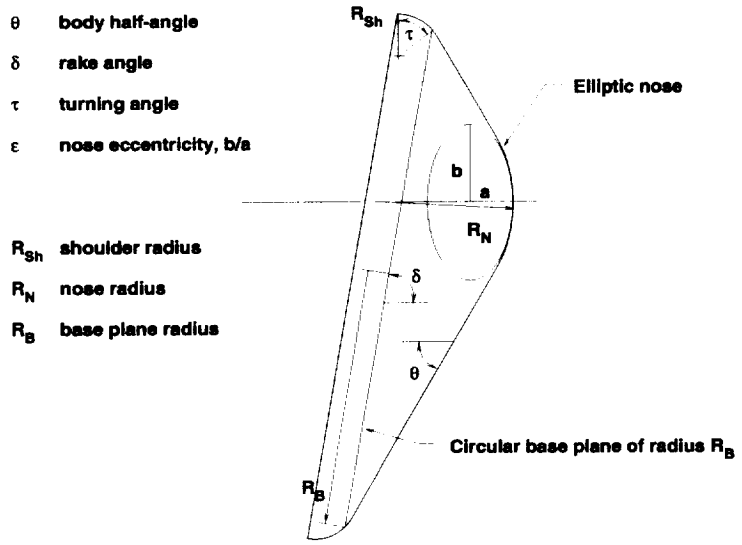


Figure 7.1. Parameters for defining generic probe shape in LAURA.

Chapter 8

Compiling LAURA

In an effort to minimize computer memory requirements, LAURA makes extensive use of `include` statements that allow the code to be tailored to a given application. This tailoring is facilitated through the execution of **stArt**, a preprocessor for LAURA, which is used to define array dimensions, governing equations, and gas models according to user inputs. Additional customization is possible through direct file editing. In the past, this meant that a local copy of the complete LAURA code needed to be present before compilation. This arrangement left the working directories cluttered with source files (many of which had undergone no modification) and object files. As a result, keeping track of the changes in the source files from case to case was a cumbersome task.

LAURA is based on the premise that the bulk of the LAURA source code does not change from one application to the next. These files, which should rarely require modification, are kept in a read-only directory. The relationship between (and roles of) the various directories employed by LAURA is shown in figure 2.3. Through **stArt**, the user defines parameters, compile directives, and other quantities that communicate the user's instructions to the `$HOME/LAURA.4.1` files during compilation. The capabilities of **stArt** have been expanded, and as a result, the need for direct editing of LAURA source files has been minimized (eliminated, in fact, for many applications).

For the most part, the tailoring is controlled through the files produced by executing **stArt** (denoted by `.strt` suffixes). To reduce clutter in the working (`LOCAL`) directory, subdirectories are created by `PRELUDE` (fig. 2.3). The files produced by **stArt** are stored in subdirectory `STRTfiles`, while `OBJfiles` holds the object files produced during compilation.

Should a LAURA file (`.F`, `.FOR`, `.inc`, or `.strt` suffix) require further tailoring, a `LOCAL` copy (`lfn`) is created with the command

```
LOCALIZE lfn
```

and then edited.

> **NOTE:** All tailoring of LAURA should be performed in the working (`LOCAL`) directory, thus allowing the installed version to remain intact.

A pure-`FORTRAN` version (`.f` suffix) of a subroutine file (`.F` or `.FOR` suffix) is also acceptable. A subsequent compilation uses this `LOCAL` file in lieu of the `$HOME/LAURA.4.1` version.

> **NOTE:** The `LOCAL` directory should contain no files with a `.f`, `.F`, `.FOR`, `.inc`, or `.strt` suffix which are not LAURA source files. Otherwise, extraneous warning messages may be displayed on the screen when **make** is executed.

The command

```
CUSTOMIZE lfn
```

moves a LOCAL file (.F, .FOR, .f, or .inc suffix only) to directory CUSTOM in $HOME/LAURA.4.1, and makes it the default instead of the $HOME/LAURA.4.1 version.

8.1. Using make

When the command

```
make
```

is executed, the actual compilation takes place in subdirectory OBJfiles.

> **NOTE:** On some machines, the environment variable $HOME is not exported to make. In such cases, the command
>
> ```
> make HOME=$HOME
> ```
>
> is required to provide make with the value of $HOME.

This fairly sophisticated Makefile is created by stArt, which tailors it not only to the machine architecture, but also to the specific case being run. For instance, if perfect gas flow is specified, the nonequilibrium and equilibrium gas routines are not included in the compilation. In fact, the stArt inputs of *igovern*, *machine*, *ndim*, *ngas*, and *nturb* determine which source files make is dependent upon. This treatment yields further reductions in computer memory requirements. Since this Makefile does more than simply compile the code, the various steps (appendix E, section E.1) are discussed below.

First, the script SYMLINKS (appendix E, section E.2) is executed to tell make where to find the source files it needs. Initially, symbolic links are established for the files in $HOME/LAURA.4.1 and STRTfiles. These symbolic links are redefined to the LOCAL and CUSTOM directories for any files that exist there. In other words, LOCAL and CUSTOM files are used if they exist; otherwise, the files in $HOME/LAURA.4.1 and STRTfiles are used.

> **NOTE:** If make has been executed previously, a check of these subdirectories is conducted to see that all LOCAL and CUSTOM files used in that compilation still exist. If not, the appropriate object files are removed before compilation so that the new executable reflects this change.

Next, the script Makedep (appendix E, section E.4) is executed to establish the dependencies of each subroutine file (.F, .FOR, or .f suffix) on the include files. For each source file, Makedep determines which include files (.inc and .strt suffixes) it depends upon. These dependencies are saved in file CHILDREN, which is in the OBJfiles directory. The existence of this file allows make to recognize when an include file is newer than an object file whose source file depends upon it. When this is the case, the source file is recompiled.

Now that make knows which source files it needs, and in turn, which include files they each depend upon, the actual compilation begins. More exactly, each subroutine file (.F, .FOR, or .f suffix) is first preprocessed according to the compile directives supplied by stArt. After preprocessing, the file is compiled, and the object (.o suffix) file is saved in OBJfiles. When all of the object files have been created, they are linked to create laura, the LAURA executable. Although this executable resides in OBJfiles, it is symbolically linked to the LOCAL directory to facilitate its execution.

The final step performed by make is the removal of the symbolic links located in OBJfiles. This step serves to reduce clutter in the OBJfiles subdirectory.

8.2. Using make debug

For running LAURA in conjunction with the `dbx` debugger (`cdbx` on CRAY architectures), the code must first be compiled with the debugger option. Also, the symbolic links should be retained so that `dbx` can locate the source files. To address these needs, the command

```
make debug
```

is employed. This command creates a subdirectory `DEBUG`, establishes the proper symbolic links, and compiles LAURA using the debugger flag (`-g`). Further, links are created for files `data`, `RESTART.in`, and `TWALL.in` (if it exists), which are in the `LOCAL` directory. For turbulent flows, links are also created to the `LOCAL` files `transition` and `variabletw`. To run the debugger, move down to the `DEBUG` directory.

8.3. Using make fortran

The command

```
make fortran
```

preprocesses each of the files with a `.F` or `.FOR` suffix without actually compiling and linking them to produce the LAURA executable. As with the standard `make`, symbolic links to the proper files are established. In addition to processing the compile directives, this command also pipes in the `include` files. Each resultant pure-`FORTRAN` file (which has the root name of the `.F` file and a `.f` suffix) is placed in subdirectory `FORTRAN`. As in section 8.1, after preprocessing, the symbolic links are removed.

The `make fortran` feature is useful for testing new code, since such new coding can be added to a file that is already tailored to the desired application. For example, in an equilibrium flow application, `make fortran` produces subroutine files (`.f` suffixes) in which the perfect gas and nonequilibrium flow coding has been omitted. Without these extraneous lines of `FORTRAN`, the user can more easily focus attention on the active (in this case equilibrium flow) sections of the various routines. Files of interest can be copied to the `LOCAL` directory, modified, and used in subsequent compilations since, as mentioned earlier, a `LOCAL` file with a `.f` suffix is recognized by `make` as a replacement for the default subroutine file (`.F` or `.FOR` suffix).

Chapter 9

Controlling LAURA

The LAURA input and case-specific include files (`.strt` suffixes) are the files through which the user can most readily control the options of LAURA. Typically, in fact, except for some advanced applications, these are the only files that the user should have to alter for a given case. These files can be grouped in the following two classes:

- Control via execution. The `LOCAL` mandatory files (`data`, `RESTART.in`) and application-dependent files (`assign_tasks`, `transition`, `TWALL.in`, `variabletw`) are read by `laura` (the LAURA executable) at run time. Therefore, `laura` does not need to be recompiled when these files are altered. These files are discussed in section 9.1. To run LAURA interactively, type the command

  `laura < data > lfn`

  This command sends the output to file `lfn`. User control of a given run is provided through the input files and the include files with the `.strt` suffix (chapter 9).

- Control via compilation. Changes to the files created by `stArt` are communicated during compilation of `laura`. The files consist of compile directives (in `HEADER.strt`) and `FORTRAN parameter` statements (in a number of files with `.strt` suffixes). The files are discussed in section 9.2. As with the LAURA source files (`.F`, `.FOR`, and `.inc` suffixes), it is recommended that any changes to these files be made on a `LOCAL` copy (using the `LOCALIZE` command), rather than the `STRTfiles` version. Subsequent `PRELUDE` runs will preserve these tailored files, rather than simply overwriting them in `STRTfiles`. Remember, to reflect a change in any of these files, `laura` must be recompiled.

First consider those files that provide user control during execution. The files `data` and `RESTART.in` are required inputs for any LAURA run. The file `data` contains a set of parameters (free-stream conditions included) which give the user a certain degree of control over LAURA without the need for recompilation. As its name implies, `RESTART.in` is the file that gives LAURA a restart capability. A third file, `TWALL.in`, supplies the surface temperature distribution when the radiative equilibrium wall temperature (*tempbc* = 2) option is exercised. If the specified wall temperature variation (*tempbc* = 1) option is exercised, the surface distribution is provided through file `variabletw`. When turbulent flows are calculated, the onset of transition is supplied through file `transition`. The file `assign_tasks` can be used to control processor allocations to computational blocks, as well as sweeping directions. Details of these files are given in sections 9.1.1 through 9.1.6. Both `RESTART.in` and `TWALL.in` are binary files. The other input files are ASCII format and can be modified through direct editing. The contents

of assign_tasks, data, transition, and variabletw are echoed to standard output with each execution of laura. This documents which LAURA options are active for a given run.

Now consider those files that provide user control during compilation. In general, the safest way to modify a file with a .strt suffix is through PRELUDE. However, for some advanced applications (chapter 11), direct editing of LOCAL versions of these files (created via the LOCALIZE command) is necessary. Some specific recommendations for modifying these files are given in sections 9.2.1 through 9.2.10.

> **NOTE:** There is actually another option for tailoring these files with .strt suffixes. As mentioned in section 7.9.2, PRELUDE can accommodate LOCAL or CUSTOM copies of stArt source files, which in turn gives the user the ability to produce tailored case-dependent include files (.strt suffixes). If any LOCAL stArt source files exist, PRELUDE will use them in conjunction with any CUSTOM files to create a LOCAL stArt executable. If CUSTOM stArt source files exist, but no LOCAL stArt source files are present, PRELUDE creates a CUSTOM stArt executable. This approach is especially useful for any modifications that will be repeated on a regular basis. This option is useful when changing the flow field initialization for a "cold-start" solution, for example.

The contents of HEADER.strt, as well as files

algnshk_vars.strt	iupwind_assn.strt	parameter.strt
gas_model_vars.strt	mtaska_assn.strt	source_vars.strt
issd_assn.strt	nordbc_assn.strt	sthrlnd_vars.strt

are repeated in LOCAL file ECHOSTRT (appendix E, section E.2). This file is created by make, and serves to document the LAURA options that are active for a given compilation. During subsequent executions of laura, the contents of ECHOSTRT are echoed to standard output.

> **NOTE:** The file ECHOSTRT should not be mistakenly edited in an attempt to effect changes to the laura executable. Modifications should be made via PRELUDE or through a LOCAL version of the above files, followed by a recompilation of laura.

9.1. Control Via Execution

9.1.1. File RESTART.in

The LAURA restart capability requires that the binary file RESTART.in be preserved between runs. This file is unformatted to minimize memory requirements. The file RESTART.in contains the grid, along with the primitive variables, for the entire flow field. As a result, in subsequent runs, LAURA can pick up the previous solution where it left off. The RESTART.in file is overwritten at the conclusion of each successful LAURA run.

> **NOTE:** Files RESTART.in and TWALL.in are overwritten at the end of each execution of laura. A backup capability exists through the command
>
> (KEEPER)
>
> (appendix J), which simply copies RESTART.in to RESTART.backup and TWALL.in to TWALL.backup.

LAURA always expects to find `RESTART.in`, even for cold starts. If *newjob* $\neq 0$ (conic or aerobrake geometry) is specified when running `PRELUDE`, then `RESTART.in` is automatically generated by `stArt`. Otherwise, the user must supply an externally generated restart file for `laura`. This file does not need to exist before running `PRELUDE`, but it must be present before executing the command

```
laura < data
```

The flow field domain can be divided into multiple computational blocks, and this is reflected in the restart file. LAURA obtains information from `RESTART.in` for each block. For nonequilibrium flow, the sequence for a given block (screen 31) is:

```
read (24) ix, jx, kx, ls

read (24) (((    u(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
&         (((    v(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
&         (((    w(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
&         ((( temp(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
&         (((tempv(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
&         ((( ri(i,j,k,s), i=1,ix  ), j=1,jx  ), k=1,kx  ),
&                          s=1,ls ),
&         (((    x(i,j,k)  , i=1,ix+1), j=1,jx+1), k=1,kx+1),
&         (((    y(i,j,k)  , i=1,ix+1), j=1,jx+1), k=1,kx+1),
&         (((    z(i,j,k)  , i=1,ix+1), j=1,jx+1), k=1,kx+1)
```

Screen 31.

where `ix`, `jx`, `kx` are *iblk*, *jblk*, and *kblk*, respectively, for block x, and `ls` is the number of species. Further, `temp` is T, `tempv` is T_V, and `ri(s)` is ρ_s; `u`, `v`, and `w` are velocity components; and `x`, `y`, and `z` are Cartesian coordinates. This read sequence is repeated for each block.

> **NOTE:** For each block, `laura` checks the values of `ix`, `jx`, `kx`, and `ls` against the maximum dimensions allowed (as specified in `parameter.strt`). If these upper limits are exceeded, an error message is issued, and the job is terminated.

For perfect gas and equilibrium flows, the sequence for a given block (screen 32) is as follows:

```
      read (24) ix, jx, kx, ls

      read (24) (((    u(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
     &          (((    v(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
     &          (((    w(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
     &          ((( temp(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
     &          (((    e(i,j,k)  , i=1,ix  ), j=1,jx  ), k=1,kx  ),
     &          (((   ri(i,j,k,1), i=1,ix  ), j=1,jx  ), k=1,kx  ),
     &          (((    x(i,j,k)  , i=1,ix+1), j=1,jx+1), k=1,kx+1),
     &          (((    y(i,j,k)  , i=1,ix+1), j=1,jx+1), k=1,kx+1),
     &          (((    z(i,j,k)  , i=1,ix+1), j=1,jx+1), k=1,kx+1)
```

Screen 32.

For perfect gas and equilibrium flows, air is treated as a single species (`ls` = 1), which is reflected above in the read sequence. As a result, $\rho_s = \rho_1 = \rho$. This read sequence is repeated for each block.

As mentioned above, `RESTART.in` is automatically generated for *newjob* $\neq 0$ (conic or aerobrake geometry). This cold-start file is a function of a number of user specifications (such as geometry or free-stream conditions). In `stArt`, the flow field variables are initialized as follows (screen 33):

```
      efactor = 8314.3 / ( bgas * wgas * vinfb**2 )

      do 50 k=1,ka

        etal = ( k - 1. )  / ka

      do 50 j=1,ja
      do 50 i=1,ia

        u   (i,j,k) = etal * uinf
        v   (i,j,k) = etal * vinf
        w   (i,j,k) = etal * winf
        temp(1,j,k) = etal * tinf + ( 1. - etal ) * twall
        e   (i,j,k) = temp(i,j,k) * efactor

   50 continue
```

Screen 33.

where *uinf*, *vinf*, and *winf* are the u-, v-, and w-components of the free-stream velocity, respectively, *tinf* is the free-stream temperature, and *twall* is the wall temperature. In addition to the above assignments, $T_V = T$ and $\rho_s = \rho_{s,\infty}$ initially.

NOTE: The above quantities are normalized in the following manner. The velocities (u, v, w, *uinf*, *vinf*, and *winf*) are nondimensionalized by the free-stream total velocity,

while the internal energy (e) is divided by this quantity squared. The species densities (ri) are normalized by the free-stream density. All temperatures are in degrees K, while x, y, and z are in grid units.

NOTE: As seen above and mentioned earlier, the self-starting feature of `stArt` employs a single computational block with the k-direction normal to the body.

9.1.2. File `assign_tasks`

The ASCII file **`assign_tasks`** gives the user control over several facets (which follow) of the LAURA algorithm via five parameters (*nbk*, *mbk*, *lstrt*, *lstop*, and *mapcpu*):

- Partitioning of computational blocks
- Number of processors assigned to individual blocks
- Sweep direction for each block

These five parameters, discussed in section 11.5.2, are entered on a single line in free format. A sample file is shown below (screen 34) for a five-block, eight-processor case:

```
1   3    1   20   1
2   3    1   20   2
3   3    1   20   3
4   3    1   20   4
5   3    1   20   5
1   3   21   40   6
2   3   21   40   7
3   3   21   40   8
4   3   21   40   1
5   3   21   40   2
1   3   41   60   3
2   3   41   60   4
3   3   41   60   5
4   3   41   60   6
5   3   41   60   7
```

Screen 34.

Here $kblk_n = 60$ for each block n, with *lstrt* and *lstop* defined as shown in the third and fourth columns, respectively. A k-directional sweep ($mbk = 3$) is specified for all tasks in the second column.

NOTE: The solution for a given block may be "frozen," while others are advanced by simply omitting that block from **`assign_tasks`**. For instance, in the above example, if the assignments for **`Block C`** ($nbk = 3$) were not included, then computations would be performed on the other four blocks while the **`Block C`** flow field remained fixed.

9.1.3. File data

The most direct user control for a given LAURA run is through the ASCII file **data**, which contains a number of parameters. Each entry is followed by a brief comment (i.e., variable name, short description, or acceptable values). LAURA receives these instructions through a series of free-formatted **read** files from **data**. The values for parameters that appear in **data** can be modified by the user through direct editing of the file. However, as a rule, lines should not be added to, or deleted from, file **data** to run different cases. Rather, the **PRELUDE** session should be repeated to create a new file **data**.

A sample **data** file is shown below (screen 35).

```
                          VERSION=LAURA.4.1
 1          nord ..................... 1(st-) or 2(nd-) order spatial accuracy
 1          ntrnsprt ............. iterations between transport property updates
 1          njcobian ..................... iterations between jacobian updates

 0  0  1    {i,j,k}vis . 0=off/1=on for {i,j,k} TL N-S viscous terms in block 1

.50000E+04  vinfb ................................... freestream velocity [m/s]
.10000E-02  rinfb ................................... freestream density [kg/m^3]
.20000E+03  tinf ..................................... freestream temperature [K]

 0          tempbc ....... {0=constant, 1=variable, 2=radiative equilibrium} Tw
.50000E+03  twall ........................ if tempbc=0:  wall temperature [K]
0.000       ept ..... if tempbc=2:  temperature relaxation factor (0 < ept < 1)

1.0000      rflngth ..... conversion:  grid units ==> meters (1 m  =  1.0000 m)
0.000000E+00 zcg ..................................... axial cg location [m ]
0.000000E+00 xcg ................................... vertical cg location [m ]
0.314159E+01 refarea ............................ reference area of body [m ^2]
0.200000E+01 reflen ............................ reference length of body [m ]

  100       iterg ............................ maximum iterations for this run
   20       movegrd ............................ frequency of grid adjustments
    0       maxmoves .......... maximum number of grid adjustments (0=no limit)
    1       iabseig ............................. {0=normal, 1=scaled} limiter
0.300       epsa ..................................... eigenvalue limiter
0.010       errd ............................ error criteria for grid doubling
10.00       hrs ................................ time limit for this run [hr]

2.00        rfinv ................... inviscid relaxation factor, (rfinv > 1.5)
1.00        rfvis ................... viscous  relaxation factor, (rfvis > 0.5)
```

Screen 35.

The above file is actually the initial **data** file for the sample case presented in appendix A.

The first line of **data** is a header that can be tailored by the user. Since this file is echoed in the LAURA output, the header serves as an identifier. The contents and length of (number of entries in) **data** are application-specific and thus will differ from case to case. The variable *ifrozen* is only included for nonequilibrium flow. Values for *ivis*, *jvis*, and *kvis* are defined for

Table 9.1. Dependence of `data` on *newjob*

newjob	*nord*	*ntrnsprt*	*njcobian*	*iterg*	*movegrd*
0	2	20	20	9000	0
1, 2	1	1	1	100	20

each computational block. The variable *attack* is only included for three-dimensional flow. The variables *tempbc*, *twall*, and *cpt* are excluded for inviscid flow (since a boundary condition for the energy equation is not required). In other words, the choices for *ngas*, *ndim*, and *igovern* within `stArt` will determine which entries are present in `data`.

NOTE: The exclusion of inactive variables serves as a safeguard to prevent the user from wasting CPU time with changes that have no effect on the solution.

NOTE: Any entry in file `data` can be modified by editing the file directly. For example, changes in free-stream conditions are easily made in this manner. However, if a particular entry is not present in the current file `data`, then the user is advised to rerun `PRELUDE` to make the changes. For example, suppose after converging a solution for inviscid flow, the user wishes to use that solution to initialize a viscous flow. As mentioned above, the variables *tempbc*, *twall*, and *cpt* are not present in the file data for inviscid flow, so the user should rerun `PRELUDE` to specify these values.

9.1.3.1. Initialization

An initial version of `data` is produced by `stArt`. The following variables are defined directly through user inputs:

attack	*rinfb*	*tempbc*	*cpt*
vinfb	*tinf*	*twall*	

The variable *rflngth* is supplied by `stArt`, based on the units specified by the user. The following initializations are supplied by `stArt`:

$ifrozen = 1$	$epsa = 0.3$	$rfinv = 2$
$maxmoves = 0$	$errd = 0.01$	$rfvis = 1$
$iabscig = 1$	$hrs = 10$	

Changes to these variables must be made by editing file `data`.

Based on the *newjob* specification, `stArt` makes the initializations shown in table 9.1. If $newjob = 0$, the user must supply the *cg*-location (*xcg*,*zcg*), reference area (*refarea*), and reference length (*reflen*) for the geometry. These values are provided by `stArt` for $newjob \neq 0$. Changes to variables listed in table 9.1 must be made by editing file `data`.

The parameters *ivis*, *jvis*, and *kvis* toggle (0 = off/1 = on) the viscous terms in the i-, j-, and k-direction, respectively. For increased flexibility, file `data` contains a set of these toggles for each computational block. If Euler equations are specified, $ivis = jvis = kvis = 0$. For full Navier-Stokes equations, $ivis = jvis = kvis = 1$. For thin-layer Navier-Stokes equations, viscous terms are included only in the body-normal direction. The self-starting feature of LAURA assumes the k-direction is normal to the body and therefore sets $ivis = 0$, $jvis = 0$, and $kvis = 1$. The values of *ivis*, *jvis*, and *kvis* can be modified by editing file `data`.

NOTE: If thin-layer Navier-Stokes equations (*igovern* = 1) have been specified, simply defining *ivis* = *jvis* = *kvis* = 1 DOES NOT yield the full Navier-Stokes equations because the cross derivative terms are omitted. The user must specify *igovern* = 2 in `stArt` to activate (during compilation of `laura`) the full Navier-Stokes terms.

NOTE: If thin-layer Navier-Stokes equations (*igovern* = 1) have been specified, setting *ivis* = *jvis* = *kvis* = 0 yields the Euler equations. However, the boundary conditions are still those of viscous flow. Further, past investigations have shown that if a viscous solution is used as the initialization for inviscid calculations, the boundary layer does not completely disappear unless the grid stretching in the body-normal direction is redefined according to inviscid parameters (which is accomplished through a call to `algnshk.F`).

9.1.3.2. Guide to File `data`

Typically, after LAURA is tailored to a given application, the path from initialization to convergence requires no additional compilations. Modifications to file `data` are the user's primary control over LAURA runs. Some "rules of thumb" for these parameters are given below.

nord:

For a cold-start initialization, the first-order accurate (*nord* = 1) scheme is more robust than the second-order finite-difference representation. However, after just a few hundred iterations, the switch to second order (*nord* = 2) should be made.

NOTE: Although a first-order solution is an improvement over the cold-start initialization, it is computationally wasteful (and, in fact, counter-productive) to fully converge the first-order solution before switching to *nord* = 2.

ifrozen:

This toggle controls the chemical and thermal source terms for nonequilibrium flow. By default *ifrozen* = 1 so that the source terms are turned on. With *ifrozen* = 0, the chemically and thermally frozen flow is calculated. In such a case, there is no dissociation, but the fluid is not treated as a perfect gas (constant γ).

ntrnsprt, and *njcobian*:

As the solution is driven toward convergence, key flow field quantities vary less from iteration to iteration. Thus, a reduction in the frequency of updates to these variables can yield a savings in computational costs. For a cold-start, the Jacobian and transport properties are updated for each iteration (*njcobian* = *ntrnsprt* = 1). As the residual drops, these numbers, which specify the number of iterations between updates, can be increased.

ivis, *jvis*, and *kvis*:

These are the toggles for the thin-layer viscous terms in the i-, j-, and k-directions, respectively. Each computational block has its own toggles. Setting *ivis* = *jvis* = *kvis* = 1 for a given block DOES NOT yield the full Navier-Stokes equations (section 9.1.3).

attack, *vinfb*, *rinfb*, and *tinf*:

The free-stream conditions can be altered at any time along the path to convergence.

tempbc, *twall*, and *cpt*:

The wall temperature boundary condition can be altered at any time along the path to convergence. For a constant wall temperature, set *tempbc* = 0 and define *twall*. For a specified wall temperature variation, set *tempbc* = 1 and supply the file `variabletw` (section 9.1.6). For the radiative equilibrium wall temperature distribution (*tempbc* = 2), the user must supply file `TWALL.in` (section 9.1.5).

rflngth, *zcg*, *xcg*, *refarea*, and *reflen*:

The reference quantities for the geometry can be altered at any time along the path to convergence.

iterg:

This parameter provides a maximum iteration limit for the current run.

> **NOTE:** The current job is terminated when either *iterg* iterations have been completed or the *hrs* time limit is reached, whichever comes first.

movegrd, and *maxmoves*:

For a given run, the flow field grid is realigned after every *movegrd* iteration (*movegrd* = 0 yields no adjustments) up to *maxmoves* times. If *maxmoves* = 0, the grid will be adjusted every *movegrd* iteration for the duration of the run.

iabseig:

This parameter controls the eigenvalue limiter scaling option. Although the default is *iabseig* = 1 (scaled), a switch to *iabseig* = 0 (normal) can be required for some problems early in the relaxation process to survive difficult transients as the solution evolves.

epsa:

This parameter controls the fraction of the local maximum eigenvalue which is used as a lower limit for defining the upwind dissipation. For *iabseig* = 1, the fraction is further reduced in directions where *ivis* = 1, *jvis* = 1, or *kvis* = 1 to substantially eliminate adverse effects on the computed heating and skin friction levels.

errd:

If the self-starting feature of LAURA is employed, the initial grid contains only one-quarter of the cells specified for the k-direction. As the shock layer develops, and the residual drops, the number of cells in this direction is doubled twice. These enrichments are performed automatically when the residual drops below the value of *errd*. In general, the timing of this doubling is not critical, so the default value is satisfactory. If the solution stagnates at a residual level above this value, however, the user can increase *errd* to hasten the adjustment.

hrs:

This parameter provides the time limit for the current run. Its value should be matched to any time limit dictated by the machine or queue since it determines when the restart files are created. If the writing of these files is not completed when the limit is reached, the run will have to be repeated.

> **NOTE:** The current job is terminated when either *iterg* iterations have been completed or the *hrs* time limit is reached, whichever comes first.

rfinv and *rfvis*:

These factors multiply the inviscid matrix $\mathbf{M}_{L,\mathrm{INV}}$ and the viscous matrix $\mathbf{M}_{L,\mathrm{VIS}}$, which are the Jacobians of the inviscid and viscous flux vectors, respectively. (Eq. (O.76) in appendix O.) These factors have no influence on the converged solution (in the sense that they do not alter the expression for the residual vector), but they do affect the path to a converged solution. The change in the solution vector caused by changes in the inviscid contributions to the residual is inversely proportional to *rfinv*. In a similar manner, changes to the solution vector because of changes in the viscous contributions to the residual are inversely proportional to *rfvis*. Their lower bounds are *rfinv* = 1.5 and *rfvis* = 0.5. Although the defaults are *rfinv* = 2 and *rfvis* = 1, increasing these to *rfinv* = 3 and *rfvis* = 2 (or possibly more) can be required early in the relaxation process for some problems to survive difficult transients as the solution evolves. The relaxation factors can also be increased in cases in which the convergence stalls because of a limit cycle induced by the use of the minmod function in the TVD scheme. Values significantly higher than these should not be required. As a general guide, low values tend to accelerate convergence after very large initial transients have passed. Larger values are appropriate if large initial transients are destabilizing or if the convergence has stalled because of limit cycles (probably in the vicinity of a captured shock) which are induced by the minmod function.

9.1.4. File `transition`

For turbulent flows (*nturb* = 1, 2), the location of the onset of transition (*str*) is specified through the ASCII file `transition`.

> **NOTE:** This location (surface distance from the nose of the geometry) should be defined as *str* = 0 for the calculation of fully turbulent flows.

> **NOTE:** The present coding assumes that the i- and j-coordinates are measured in the streamwise and circumferential directions, respectively, along the body. Further, it assumes that
>
> - the nose point ($s = 0$) lies in `Block A`
> - `Block A` is active

A sample file is shown below for transition at *str* = 1.0 *grid unit*:

```
1.0
```

9.1.5. File TWALL.in

For *tempbc* = 0 (constant wall temperature), the wall temperature value is provided through file **data** (section 9.1.3). For *tempbc* = 1 (specified wall temperature variation), the wall temperature distribution is defined through file **variabletw** (section 9.1.6). For *tempbc* = 2 (radiative equilibrium wall temperature), the wall temperature distribution is initialized through binary file **TWALL.in**. The size of this file is a function of the number of blocks (*nblocks*), the number of surfaces (*nsrf*), and the array dimensions. The read sequence for this unformatted file is:

```
    read (45) lsrf

    do 35 nn=1,lsrf
      read (45) imax, jmax
      read (45) (( twallv(i,j,nn), i=1,imax), j=1,jmax )
35 continue
```

where **lsrf** is the number of surfaces and **imax** and **jmax** are the grid dimensions along a given surface. This distribution is adjusted with each global iteration in LAURA. The adjustment is not a straight substitution, but rather it is a relaxation controlled by parameter *cpt*, which is read from file **data**. With the completion of a LAURA run, **TWALL.in** is overwritten to reflect the current distribution.

> **NOTE:** The file **TWALL.in** must exist when *tempbc* = 2. However, its overwrite only occurs if *cpt* $\neq$ 0. This allows the wall boundary condition to be frozen at any time during the solution procedure. This specification can also allow a flow field distribution with a fixed wall boundary condition to be used to initialize a *tempbc* = 2 case. For example, suppose that a fixed wall distribution (*tempbc* = 0, 1; *cpt* = 0) is specified for the early stages of flow field development. In a later run, set *cpt* $\neq$ 0 to create a **TWALL.in** file. In the subsequent run, set *tempbc* = 2 (since **TWALL.in** now exists), and the solution picks up where it left off, except that the wall temperature is iteratively determined as part of the solution.

9.1.6. File variabletw

When a specified wall temperature variation is desired (for comparison with experimental data, for instance), the wall temperature distribution must be supplied in the ASCII file **variabletw**. The surface distance and the temperature are specified at discrete locations down the body (which is assumed to be in the i-direction). A free format is used to read in this distribution (one ordered pair [s, T] per line) from file **variabletw**. LAURA interpolates these values to provide the proper distribution for the grid specified in **RESTART.in**.

> **NOTE:** Currently, this option (*tempbc* = 1) is not available for three-dimensional flows.

> **NOTE:** The present coding assumes the i-coordinate is measured in the streamwise direction along the body.

A sample file is shown below for a streamwise temperature distribution that varies from T_w = 313.9 K to T_w = 288.3 K as a function of the surface distance from s_b = 0.0 to s_b = 12.5 *grid units* (screen 36):

```
 0.0      313.9
 0.5225   310.0
 1.0475   313.9
 1.5700   329.4
 2.0950   331.7
 2.6175   316.1
 3.1425   298.9
 3.5775   290.6
 3.9775   288.9
 4.9775   287.8
 5.9750   288.3
 6.9725   287.8
 7.9700   287.8
 9.4675   287.2
10.4650   287.2
11.4625   288.3
12.5      288.3
```

Screen 36.

9.2. Control Via Compilation

9.2.1. File HEADER.strt

During **make**, each file in the LAURA source code is preprocessed before it is passed to the **FORTRAN** compiler. This preprocessing eliminates those sections of code which will be inactive for this case (as determined by user inputs). File **HEADER.strt** consists of a series of "**#define**" statements for variables that control the compile directives. The number of definitions and their values are controlled by the user through **stArt**. The options for these definitions are given below, along with the **FORTRAN** variable that gives the user control over them.

During the **INSTALL_LAURA.4.1** procedure, **mAch+prOc** identifies the machine architecture, and this is in turn reflected in the first definition in **HEADER.strt**, as follows:

$$\left.\begin{array}{l}\texttt{\#define CRAY\_ARCHITECTURE}\\ \texttt{\#define SUN\_ARCHITECTURE}\\ \texttt{\#define SGI\_ARCHITECTURE}\\ \texttt{\#define CONVEX\_ARCHITECTURE}\end{array}\right\} \text{for } machine = \left\{\begin{array}{l}0\\1\\2\\3\end{array}\right.$$

The next definition is dependent on *ndim*:

$$\left.\begin{array}{l}\texttt{\#define AXISYMMETRIC\_FLOW}\\ \texttt{\#define TWO\_DIMENSIONAL\_FLOW}\\ \texttt{\#define THREE\_DIMENSIONAL\_FLOW}\end{array}\right\} \text{for } ndim = \left\{\begin{array}{l}1\\2\\3\end{array}\right.$$

This definition reflects the value of *ngas*:

$$\left.\begin{array}{l}\texttt{\#define PERFECT\_GAS}\\ \texttt{\#define EQUILIBRIUM}\\ \texttt{\#define NONEQUILIBRIUM}\end{array}\right\} \text{for } ngas = \left\{\begin{array}{l}0\\1\\2\end{array}\right.$$

For equilibrium flow, the value of *icrv* provides a definition:

$$\left.\begin{array}{l}\texttt{\#define VINOKUR}\\ \texttt{\#define TANNEHILL}\end{array}\right\} \text{for } icrv = \begin{cases} 1 \\ 2 \end{cases}$$

For nonequilibrium flow, *itherm* is reflected in a definition:

$$\left.\begin{array}{l}\texttt{\#define ONE\_TEMPERATURE}\\ \texttt{\#define TWO\_TEMPERATURE}\end{array}\right\} \text{for } itherm = \begin{cases} 1 \\ 2 \end{cases}$$

The user also has an option for laminar or turbulent flow (through *nturb*):

$$\left.\begin{array}{l}\texttt{\#define LAMINAR\_FLOW}\\ \texttt{\#define TURBULENT\_FLOW}\end{array}\right\} \text{for } nturb = \begin{cases} 0 \\ 1,2 \end{cases}$$

If turbulent flow is chosen, there is presently a choice of two models (also available through *nturb*):

$$\left.\begin{array}{l}\texttt{\#define CEBECI\_SMITH}\\ \texttt{\#define BALDWIN\_LOMAX}\end{array}\right\} \text{for } nturb = \begin{cases} 1 \\ 2 \end{cases}$$

If $ngas = 2$ and $ns = 1$, then an additional definition is created:

```
#define ONE_SPECIES
```

The last definition is a function of *igovern* and specifies the governing equations:

$$\left.\begin{array}{l}\texttt{\#define INVISCID}\\ \texttt{\#define THIN\_LAYER}\\ \texttt{\#define NAVIER\_STOKES}\end{array}\right\} \text{for } igovern = \begin{cases} 0 \\ 1 \\ 2 \end{cases}$$

NOTE: Although this file can be modified directly, the user is strongly urged to make all changes by rerunning `PRELUDE` and modifying the variable that controls these definitions. This prevents misspellings, but more importantly, the definitions of certain variables are interrelated. For example, if "`EQUILIBRIUM`" is defined, then either "`VINOKUR`" or "`TANNEHILL`" must also be defined. Remember, if file `HEADER.strt` is modified, `make` must be run again to reflect the new specifications.

9.2.2. File `algnshk_vars.strt`

The file `algnshk_vars.strt` contains a number of parameters that control the implementation of `algnshk.F` (section 11.1.1). The default `algnshk_vars.strt` file is shown below (screen 37):

```
      parameter ( recell = 1. )          ! cell Re at wall
      parameter ( jumpflag = 1 )         ! flag for jump property
      parameter ( fctrjmp = 1.5 )        ! factor for jump property
      parameter ( fsh = .8 )             ! fraction of grid within shock layer
      parameter ( fstr = .5 )            ! fraction of cells in stretch region

      parameter ( betagrd = 0. )         ! stretching function control

C NOTE:  ep0 not used if betagrd < 1
      parameter ( ep0 = 0. )             ! no shock clustering
C     parameter ( ep0 = 25. / 4. )       ! recommended max. value
```

Screen 37.

These parameters should be changed by editing a LOCAL copy of `algnshk_vars.strt` and running `make` again.

9.2.3. File `gas_model_vars.strt`

The file `gas_model_vars.strt` contains a number of parameters that reflect LAURA defaults. The default `gas_model_vars.strt` file for the sample case presented in appendix A is shown below (screen 38):

```
      parameter ( bgas = .4 )            ! gamma - 1 (used for PG, EQ)
      parameter ( wgas = 28.86 )         ! air mol weight (used for PG, EQ)

C::   PERFECT GAS VARIABLE:

      parameter ( prandtl = .71 )        ! Prandtl Number
```

Screen 38.

For nonequilibrium flows, the perfect gas Prandtl number assignment is replaced with the following flags in `gas_model_vars.strt` (screen 39):

```
C::  NONEQUILIBRIUM FLAGS:

      parameter ( kmodel = 3 )   ! kinetic model option (1-5)
      parameter ( jtype = 0 )    ! !  wall catalysis option (0-6)

      parameter ( nsz = 11 )     ! number of species defined in air.f
      parameter ( nsp = 16 )     ! number of collision partners (kinetic)
      parameter ( nrz = 27 )     ! number of allowed reactions (kinetic)
```

Screen 39.

The value of *jtype* is defined by the user in `stArt`. This value can be modified by rerunning `PRELUDE` or by editing a `LOCAL` copy of `gas_model_vars.strt`. The other parameters should be changed by editing a `LOCAL` copy of `gas_model_vars.strt`. Remember, if file `gas_model_vars.strt` is modified, `make` must be run again to reflect the new specifications.

9.2.4. File `issd_assn.strt`

The file `issd_assn.strt` contains the flag that toggles the solid-state device (SSD) on CRAY architectures. The default `issd_assn.strt` file is shown below:

```
      parameter ( issd = 0 )            ! 0=off/1=on for SSD
```

The value of *issd* should be changed by editing a `LOCAL` copy of `issd_assn.strt` and running `make` again.

9.2.5. File `iupwind_assn.strt`

The file `iupwind_assn.strt` contains the flag that controls which TVD limiter is used. The default `iupwind_assn.strt` file is shown below:

```
      parameter ( iupwind = 0 )         ! option (0,1,2) for TVD limiter
```

The value of *iupwind* should be changed by editing a `LOCAL` copy of `iupwind_assn.strt` and running `make` again.

9.2.6. File `mtaska_assn.strt`

The file `mtaska_assn.strt` contains the flag that toggles the adaptive partitioning of tasks. The default `mtaska_assn.strt` file is shown below:

```
      parameter ( mtaska = 0 )          ! adaptive partitioning of tasks
```

The value of *mtaska* should be changed by editing a LOCAL copy of mtaska_assn.strt and running make again.

9.2.7. File nordbc_assn.strt

The file nordbc_assn.strt contains the flag that controls the spatial accuracy of surface and outflow boundary conditions. The default nordbc_assn.strt file is shown below:

```
parameter ( nordbc = 1 )          ! 1st- or 2nd-order BC
```

The value of *nordbc* should be changed by editing a LOCAL copy of nordbc_assn.strt and running make again.

9.2.8. File parameter.strt

This file parameter.strt contains a number of parameters that reflect user inputs during the compilation of laura. The structure of parameter.strt is shown below (screen 40).

```
parameter ( ns = 1 )      ! air is treated as 1 species for PG, EQ
parameter ( neq = 5 )     ! ...thus, there are 5 governing eqns
parameter ( nblocks = 1 ) ! number of computational blocks
parameter ( iaq =  30, ias = iaq + 1 )    !  block  1:   "i"-cells
parameter ( jaq =   1, jas = jaq + 1 )    !          1:   "j"-cells
parameter ( kaq =  64, kas = kaq + 1 )    !          1:   "k"-cells
parameter ( isjs = iaq * jas )    ! max dimension of sweep plane
parameter ( nsrf = 1 )            ! number of surfaces
parameter ( isrf =  30, jsrf =   1 )      ! max surface dimensions
```

Screen 40.

The contents and length of (number of entries in) parameter.strt is application-specific and thus will differ from case to case. In addition to the above entries, the dimensions *iaqf* and *jaqf* are included for full Navier-Stokes calculations. Also, the dimensions *maxi*, *maxj*, and *maxk* are defined for turbulent flow.

> **NOTE:** The dimensions of each computational block must be specified (e.g., *iaq*, *jaq*, and *kaq* for Block A and *ibq*, *jbq*, and *kbq* for Block B).

The parameter *nsrf* defines the total number of solid surfaces in all computational blocks. The parameters *isrf* and *jsrf* give the maximum values for the first and second indices, respectively, of any active wall boundary. These parameters, along with *iaqf*, *jaqf*, *neq*, and *ns*, are defined by stArt based on user inputs. Direct modification by the user via editing a LOCAL version of parameter.strt is unnecessary and discouraged. Changes to these parameters are best accomplished through running PRELUDE.

The parameter *isjs* is also defined by `stArt`, based on user inputs. A k-directional sweep is assumed. If an i- or j-directional sweep will be employed, the value of *isjs* can be changed by editing a `LOCAL` copy of `parameter.strt` (section 11.3).

`PRELUDE` should be rerun when the number of computational blocks (*nblocks*) is to be changed. This allows `stArt` to create the appropriate parameters for the block dimensions. The block dimensions themselves (such as *iaq* and *jaq*) also can be changed by editing a `LOCAL` copy of `parameter.strt`.

Remember, if file `parameter.strt` is modified, `make` must be run again to reflect the new specifications.

9.2.9. File `source_vars.strt`

The file `source_vars.strt` contains flags used in file `source.F` (nonequilibrium flows only). The default `source_vars.strt` file is shown below:

```
parameter ( imptemp = 1 )      !  0=ex/1=implicit T dependence
parameter ( icharge = 1 )      !  toggle for el continuity eqn
```

These parameters should be changed by editing a `LOCAL` copy of `source_vars.strt` and running `make` again.

9.2.10. File `sthrlnd_vars.strt`

The file `sthrlnd_vars.strt` contains the coefficients for `sthrlnd.F`, which is Sutherland's law (used for perfect gas flows only). The default `sthrlnd_vars.strt` file is shown below:

```
parameter ( v1gas = 1.4643e-6, v2gas = 112.222 )
```

These parameters should be changed by editing a `LOCAL` copy of `sthrlnd_vars.strt` and running `make` again.

NOTE: Remember to use constants that are appropriate for temperature in K.

Chapter 10

Output From LAURA

In addition to the standard (screen) output, a number of files are generated during and at the end of a successful `laura` run. Information is written to the files `algnshk.out` and `conv.out` during the `laura` run. At the conclusion of the run, the files `RESTART.in` and `TWALL.in` are updated. Next, the file `grid.out` is created. The information contained in these files is discussed in this chapter. Output from the sample case (appendix A) is shown.

10.1. Screen Output

The LAURA algorithm is executed interactively with the command

```
laura < data
```

The progress of an interactive job can be monitored through the output that is written to the screen (standard output). This output can be redirected to file `lfn` with the command:

```
laura < data > lfn
```

The standard output for the initial run of the sample case is displayed and discussed in the pages that follow.

NOTE: In actuality, the contents of a number of files are echoed to the screen at the start of a run. The following information is contained in this preamble:

- The contents of file `data`
- The contents of file `variabletw` (*tempbc* = 1 only)
- The contents of file `transition` (*nturb* > 0 only)
- The contents of file `ECHOSTRT` (section 9)
- A list of any `LOCAL` or `CUSTOM` files used in the compilation

In the interest of brevity, this preamble is not shown here.

The free-stream pressure and similarity parameters (which are calculated based on user inputs) and the value of the Courant number for this run are output (screen 41):

```
pressure              0.57618E+02  N/m^2

Mach number           17.605
Reynolds number       0.37693E+06
Knudsen number        0.78917E-04

Courant number        0.1E+07
```

Screen 41.

A header identifies elements of the main screen output as follows:

`tsk`	task number
`blk`	block number for this task
`ms`	sweep direction (*mbk*) for this task
`iter`	iteration number
`L2 norm`	running total of $L2$ norms for all tasks
`tsk norm`	$L2$ norm for this task
`inf norm`	maximum residual for this task
`i, j, k, m`	index of cell (i,j,k), and equation number (m), where the maximum residual for this task occurs
`time`	value from system timer
`strt`	*lstrt* for this task
`stop`	*lstop* for this task

In addition, as each processor comes on line, the message "CPU number -- starting" is issued to the screen (screen 42):

```
tsk blk ms iter   L2 norm  tsk norm  inf norm   i   j   k   m   time  strt stp

                          CPU number 01 starting

  1   1  3     1  1.113E+01 1.113E+01 4.060E-01   7   1   2   1    0.85   1  16
  1   1  3     2  3.800E+01 3.800E+01 1.050E+00   6   1   1   1    1.32   1  16
  1   1  3     3  1.200E+01 1.200E+01 7.042E-01   5   1   1   1    1.80   1  16
  1   1  3     4  2.107E+01 2.107E+01 6.712E-01   7   1   1   1    2.27   1  16
  1   1  3     5  9.462E+00 9.462E+00 5.502E-01   5   1   1   1    2.76   1  16
  1   1  3     6  1.459E+01 1.459E+01 4.923E-01   9   1   1   1    3.24   1  16
  1   1  3     7  7.563E+00 7.563E+00 4.525E-01   7   1   1   1    3.71   1  16
  1   1  3     8  1.095E+01 1.095E+01 3.884E-01  10   1   1   1    4.19   1  16
  1   1  3     9  6.207E+00 6.207E+00 3.740E-01   7   1   1   1    4.66   1  16
  1   1  3    10  8.557E+00 8.557E+00 3.164E-01  11   1   1   1    5.13   1  16
  1   1  3    11  5.223E+00 5.223E+00 3.130E-01   8   1   1   1    5.61   1  16
  1   1  3    12  6.853E+00 6.853E+00 2.647E-01  12   1   1   1    6.08   1  16
  1   1  3    13  4.490E+00 4.490E+00 2.659E-01   9   1   1   1    6.56   1  16
  1   1  3    14  5.584E+00 5.584E+00 2.265E-01  13   1   1   1    7.03   1  16
  1   1  3    15  3.924E+00 3.924E+00 2.290E-01  10   1   1   1    7.51   1  16
  1   1  3    16  4.612E+00 4.612E+00 1.976E-01  15   1   1   1    7.98   1  16
  1   1  3    17  3.478E+00 3.478E+00 1.994E-01  11   1   1   1    8.46   1  16
  1   1  3    18  3.860E+00 3.860E+00 1.750E-01  16   1   1   1    8.93   1  16
  1   1  3    19  3.125E+00 3.125E+00 1.753E-01  13   1   1   1    9.41   1  16
  1   1  3    20  3.277E+00 3.277E+00 1.568E-01  17   1   1   1    9.88   1  16

    ...Grid adjusted after iter =   20
```

Screen 42.

```
1  1  3    21  3.872E+00 3.872E+00 3.412E-01  18   1   8  1    10.50   1  16
1  1  3    22  2.514E+00 2.514E+00 2.371E-01  18   1   8  1    10.98   1  16
1  1  3    23  1.876E+00 1.876E+00 1.974E-01  10   1   9  1    11.46   1  16
1  1  3    24  2.058E+00 2.058E+00 1.461E-01  11   1   9  1    11.94   1  16
1  1  3    25  1.489E+00 1.489E+00 1.361E-01  10   1   9  1    12.42   1  16
1  1  3    26  1.880E+00 1.880E+00 1.148E-01  14   1   1  1    12.90   1  16
1  1  3    27  1.415E+00 1.415E+00 1.065E-01  14   1   1  1    13.38   1  16
1  1  3    28  1.820E+00 1.820E+00 1.297E-01  20   1   1  1    13.85   1  16
1  1  3    29  1.421E+00 1.421E+00 1.195E-01  20   1   1  1    14.33   1  16
1  1  3    30  1.681E+00 1.681E+00 1.432E-01  22   1   1  1    14.82   1  16
1  1  3    31  1.357E+00 1.357E+00 1.272E-01  22   1   1  1    15.29   1  16
1  1  3    32  1.444E+00 1.444E+00 1.474E-01  24   1   1  1    15.77   1  16
1  1  3    33  1.208E+00 1.208E+00 1.265E-01  24   1   1  1    16.25   1  16
1  1  3    34  1.169E+00 1.169E+00 1.428E-01  26   1   1  1    16.72   1  16
1  1  3    35  1.020E+00 1.020E+00 1.192E-01  25   1   1  1    17.20   1  16
1  1  3    36  9.127E-01 9.127E-01 1.314E-01  27   1   1  1    17.67   1  16
1  1  3    37  8.452E-01 8.452E-01 1.080E-01  27   1   1  1    18.14   1  16
1  1  3    38  7.073E-01 7.073E-01 1.174E-01  27   1   1  1    18.62   1  16
1  1  3    39  7.070E-01 7.070E-01 9.617E-02  27   1   1  1    19.09   1  16
1  1  3    40  5.588E-01 5.588E-01 1.011E-01  28   1   1  1    19.57   1  16

   ...Grid adjusted after iter =   40

1  1  3    41  2.646E+00 2.646E+00 2.781E-01  16   1  12  1    20.20   1  16
1  1  3    42  1.387E+00 1.387E+00 1.421E-01  17   1  10  1    20.67   1  16
1  1  3    43  1.082E+00 1.082E+00 1.146E-01  11   1  12  1    21.15   1  16
1  1  3    44  7.598E-01 7.598E-01 1.014E-01  28   1   1  1    21.62   1  16
1  1  3    45  6.750E-01 6.750E-01 8.246E-02  28   1   1  1    22.09   1  16
1  1  3    46  5.215E-01 5.215E-01 8.600E-02  29   1   1  1    22.56   1  16
1  1  3    47  5.348E-01 5.348E-01 7.471E-02   9   1  11  1    23.04   1  16
1  1  3    48  4.059E-01 4.059E-01 7.023E-02  29   1   1  1    23.51   1  16
1  1  3    49  4.587E-01 4.587E-01 7.235E-02   9   1  12  1    23.98   1  16
1  1  3    50  3.383E-01 3.383E-01 5.551E-02   9   1  12  1    24.46   1  16
1  1  3    51  4.222E-01 4.222E-01 7.845E-02   9   1  12  1    24.93   1  16
1  1  3    52  3.072E-01 3.072E-01 6.533E-02   9   1  12  1    25.41   1  16
1  1  3    53  4.148E-01 4.148E-01 8.061E-02   9   1  12  1    25.88   1  16
1  1  3    54  3.013E-01 3.013E-01 7.083E-02   9   1  12  1    26.35   1  16
1  1  3    55  4.249E-01 4.249E-01 9.626E-02   8   1  13  1    26.80   1  16
1  1  3    56  3.104E-01 3.104E-01 7.890E-02   8   1  13  1    27.27   1  16
1  1  3    57  4.442E-01 4.442E-01 1.093E-01   9   1  13  1    27.74   1  16
1  1  3    58  3.285E-01 3.285E-01 9.024E-02   9   1  13  1    28.21   1  16
1  1  3    59  4.687E-01 4.687E-01 1.139E-01   9   1  13  1    28.67   1  16
1  1  3    60  3.518E-01 3.518E-01 9.626E-02   9   1  13  1    29.14   1  16

   ...Grid adjusted after iter =   60
```

Screen 42. Concluded.

As shown, messages are also issued when **algnshk.F** is called. Here, shock adjustment has been requested after each 20 iterations. The confirmation message is "...Grid adjusted after iter = ___" (screen 43).

```
 1  1  3    61  8.506E-01 8.506E-01 2.808E-01  15   1  13  1    29.77   1  16
 1  1  3    62  5.473E-01 5.473E-01 1.369E-01  15   1  13  1    30.24   1  16
 1  1  3    63  6.346E-01 6.346E-01 1.604E-01  15   1  13  1    30.72   1  16
 1  1  3    64  4.670E-01 4.670E-01 1.063E-01  15   1  13  1    31.20   1  16
 1  1  3    65  6.021E-01 6.021E-01 1.322E-01  15   1  13  1    31.66   1  16
 1  1  3    66  4.598E-01 4.598E-01 1.017E-01  15   1  13  1    32.13   1  16
 1  1  3    67  6.027E-01 6.027E-01 1.198E-01  15   1  13  1    32.61   1  16
 1  1  3    68  4.678E-01 4.678E-01 9.944E-02  15   1  13  1    33.09   1  16
 1  1  3    69  6.136E-01 6.136E-01 1.114E-01  10   1  14  1    33.56   1  16
 1  1  3    70  4.816E-01 4.816E-01 9.485E-02  15   1  13  1    34.05   1  16
 1  1  3    71  6.289E-01 6.289E-01 1.164E-01  15   1  14  1    34.52   1  16
 1  1  3    72  4.976E-01 4.976E-01 9.597E-02  15   1  14  1    35.00   1  16
 1  1  3    73  6.457E-01 6.457E-01 1.228E-01  15   1  14  1    35.47   1  16
 1  1  3    74  5.137E-01 5.137E-01 1.032E-01  15   1  14  1    35.94   1  16
 1  1  3    75  6.616E-01 6.616E-01 1.228E-01  15   1  14  1    36.41   1  16
 1  1  3    76  5.283E-01 5.283E-01 1.052E-01  15   1  14  1    36.89   1  16
 1  1  3    77  6.745E-01 6.745E-01 1.180E-01  15   1  14  1    37.36   1  16
 1  1  3    78  5.400E-01 5.400E-01 1.032E-01  15   1  14  1    37.83   1  16
 1  1  3    79  6.831E-01 6.831E-01 1.098E-01  15   1  14  1    38.31   1  16
 1  1  3    80  5.477E-01 5.477E-01 9.786E-02  15   1  14  1    38.78   1  16

   ...Grid adjusted after iter =   80

 1  1  3    81  9.475E-01 9.475E-01 2.255E-01  24   1  14  1    39.41   1  16
 1  1  3    82  6.263E-01 6.263E-01 1.435E-01  24   1  14  1    39.88   1  16
 1  1  3    83  7.138E-01 7.138E-01 1.328E-01  23   1  13  1    40.35   1  16
 1  1  3    84  5.545E-01 5.545E-01 1.133E-01  23   1  13  1    40.83   1  16
 1  1  3    85  6.716E-01 6.716E-01 1.173E-01  23   1  13  1    41.30   1  16
 1  1  3    86  5.327E-01 5.327E-01 1.024E-01  23   1  13  1    41.78   1  16
 1  1  3    87  6.443E-01 6.443E-01 1.108E-01  16   1  14  1    42.25   1  16
 1  1  3    88  5.147E-01 5.147E-01 9.525E-02  17   1  14  1    42.73   1  16
 1  1  3    89  6.182E-01 6.182E-01 1.058E-01  17   1  14  1    43.20   1  16
 1  1  3    90  4.956E-01 4.956E-01 9.285E-02  17   1  14  1    43.68   1  16
 1  1  3    91  5.917E-01 5.917E-01 1.012E-01  23   1  14  1    44.16   1  16
 1  1  3    92  4.751E-01 4.751E-01 8.863E-02  18   1  14  1    44.63   1  16
 1  1  3    93  5.648E-01 5.648E-01 9.614E-02  23   1  14  1    45.11   1  16
 1  1  3    94  4.536E-01 4.536E-01 8.516E-02  18   1  14  1    45.60   1  16
 1  1  3    95  5.375E-01 5.375E-01 9.005E-02  23   1  14  1    46.08   1  16
 1  1  3    96  4.313E-01 4.313E-01 8.051E-02  19   1  14  1    46.55   1  16
 1  1  3    97  5.100E-01 5.100E-01 8.575E-02  16   1  15  1    47.02   1  16
 1  1  3    98  4.083E-01 4.083E-01 7.585E-02  19   1  14  1    47.53   1  16
 1  1  3    99  4.823E-01 4.823E-01 8.431E-02  16   1  15  1    48.00   1  16
 1  1  3   100  3.849E-01 3.849E-01 7.109E-02  16   1  15  1    48.48   1  16

   ...Grid adjusted after iter =  100
```

Screen 43.

The self-starting feature of LAURA is employed for this sample case. As mentioned in section 9.1.3, the initial grid contains only one-quarter of the cells specified for the k-direction. As the shock layer develops, and the residual drops, the number of cells in this direction is doubled twice. With each doubling, the message "...Grid doubled after iter = ___" is issued. After the specified k-direction is achieved, further calls to **algnshk.F** will be accompanied by a warning message if the criteria for the cell Reynolds number at the wall is not met. As stated in section 9.1.3.2, the variable *maxmoves* controls the number of times **algnshk.F** is called. When this value is exceeded, no further calls to **algnshk.F** are made, and the message "...Turning off algnshk after ___ adjustments" is issued. Each of these messages can be viewed in context in appendix A, where this sample case is discussed further.

As each task finishes, a confirmation message is issued:

```
CPU  1 terminated at    48.64 seconds (after   100 iterations).
```

Routine **taskit.F** outputs the integrated surface quantities (screen 44):

```
Aerodynamic coefficients cx, cy, cz, gy and
mass and energy flux through surfaces
for each of  1 tasks are presented below

No assumption of a half-body computation is
is made here; consequently, cx, cz, and gy
will need to be multiplied by 2 to obtain
the correct aerodynamics when only half of
the body is computed.


    integrated surface quantities:

                cx =  0.10319546E-01
                cy =  0.00000000E+00
                cz =  0.13448360E-01
                gy = -0.55733500E-02
           summdot =  0.32861283E-03
           sumheat = -0.10528496E-02
```

Screen 44.

Routine `wrapup.F` outputs several reference quantities (xcg, zcg, $refarea$, and $reflen$) as shown in screen 45:

```
                    x/rflngth   =  0.00000000E+00
                    z/rflngth   =  0.00000000E+00
                area/rflngth^2  =  0.31415901E+01
               length/rflngth   =  0.20000000E+01


END !  END !  END !  END !  END !  END !  END !  END !  END !  END
```

Screen 45.

NOTE: The call to routine `outputa.f` is currently "commented out." If this extended output is desired, simply reinstate this call in a `LOCAL` version of wrapup.

10.2. File `algnshk.out`

The file `algnshk.out` contains information from each call to `algnshk.F`. This file is overwritten with each `laura` run (provided `algnshk.F` is called). The output for the initial run of the sample case is displayed (screen 46) and discussed in the pages that follow.

```
...Grid adjusted after iter =   20

                                 norm.     wall       max.       location of
 i     j       hmin1             dist.   stretch    stretch       maximum

 1     1   .5489674E-05  .174746  2.212342  4.168002  .004896 ( 5)
 2     1   .5470985E-05  .174609  2.213022  4.169779  .004890 ( 5)
 3     1   .5527656E-05  .175025  2.210968  4.164412  .004908 ( 5)
 4     1   .5630268E-05  .175771  2.207308  4.154848  .004941 ( 5)
 5     1   .5780112E-05  .176844  2.202096  4.141229  .004988 ( 5)
 6     1   .5982895E-05  .178264  2.195280  4.123418  .005050 ( 5)
 7     1   .6246614E-05  .180060  2.186795  4.101244  .005129 ( 5)
 8     1   .6577184E-05  .182235  2.176711  4.074893  .005225 ( 5)
 9     1   .6984580E-05  .184810  2.165040  4.044396  .005339 ( 5)
10     1   .7481306E-05  .187808  2.151805  4.009813  .005473 ( 5)
11     1   .8081044E-05  .191242  2.137085  3.971347  .005626 ( 5)
12     1   .8803757E-05  .195146  2.120900  3.929052  .005803 ( 5)
13     1   .9672593E-05  .199547  2.103314  3.883098  .006003 ( 5)
14     1   .1071551E-04  .204471  2.084415  3.833714  .006228 ( 5)
15     1   .1196553E-04  .209941  2.064320  3.781201  .006481 ( 5)
16     1   .1346256E-04  .215980  2.043154  3.725892  .006762 ( 5)
17     1   .1525572E-04  .222617  2.021041  3.668108  .007074 ( 5)
18     1   .1740257E-04  .229871  1.998128  3.608233  .007418 ( 5)
19     1   .1997267E-04  .237766  1.974556  3.546637  .007796 ( 5)
20     1   .2304558E-04  .246317  1.950493  3.483757  .008210 ( 5)
21     1   .2671226E-04  .255529  1.926113  3.420050  .008660 ( 5)
22     1   .3106547E-04  .265379  1.901645  3.356112  .009146 ( 5)
23     1   .3620289E-04  .275833  1.877309  3.292519  .009667 ( 5)
24     1   .4221089E-04  .286822  1.853358  3.229931  .010220 ( 5)
25     1   .4915655E-04  .298248  1.830044  3.169010  .010801 ( 5)
26     1   .5708405E-04  .309998  1.807590  3.110333  .011404 ( 5)
27     1   .6600447E-04  .321944  1.786181  3.054391  .012023 ( 5)
28     1   .7591113E-04  .333975  1.765927  3.001463  .012650 ( 5)
29     1   .8692797E-04  .346152  1.746634  2.951048  .013291 ( 5)
30     1   .9837127E-04  .357733  1.729310  2.905780  .013904 ( 5)
```

Screen 46.

Each call to `algnshk.F` is annotated with the iteration count at the time of the call through the message "...Grid adjusted after iter = ___." A header identifies the quantities contained in this output (screen 46):

`i, j`	station index
`hmin1`	height (dimension in k-direction) of cell adjacent to wall
`norm. dist.`	normalized distance from surface to outer grid boundary
`wall stretch`	grid stretching factor at the wall
`max stretch`	maximum value of grid stretching factor used in the stretching region
`location`	normal distance from body to point of maximum grid stretching (k-index in parentheses)

NOTE: The target value for normalized distance is $ssl = 1$. A value of less than unity indicated that the criterion for *recell* could not be met. In such cases, `algnshk.F` stretches the grid by a factor of $1/ssl$ to encompass the bow shock. As a result, the actual value of *recell* is greater than unity. If *recell* is of order 10 or greater, then heating results may be suspect.

```
...Grid adjusted after iter =   40

 i    j       hmin1          norm.      wall       max.       location of
                             dist.     stretch    stretch      maximum

 1    1   .3131174E-05   .153732   2.328307   4.471033   .004002 ( 5)
 2    1   .3073462E-05   .153089   2.332290   4.481442   .003975 ( 5)
 3    1   .3053028E-05   .152860   2.333721   4.485180   .003966 ( 5)
 4    1   .3046536E-05   .152787   2.334177   4.486374   .003963 ( 5)
 5    1   .3000777E-05   .152267   2.337429   4.494869   .003941 ( 5)
 6    1   .2988069E-05   .152122   2.338341   4.497254   .003935 ( 5)
 7    1   .3026658E-05   .152562   2.335583   4.490047   .003953 ( 5)
 8    1   .3095165E-05   .153332   2.330782   4.477502   .003985 ( 5)
 9    1   .3187141E-05   .154347   2.324522   4.461143   .004027 ( 5)
10    1   .3320904E-05   .155786   2.315771   4.438275   .004087 ( 5)
11    1   .3528920E-05   .157941   2.302921   4.404697   .004177 ( 5)
12    1   .3828609E-05   .160890   2.285834   4.360046   .004302 ( 5)
13    1   .4074159E-05   .163183   2.272920   4.326299   .004399 ( 5)
14    1   .4354005E-05   .165677   2.259228   4.290522   .004505 ( 5)
15    1   .4711490E-05   .168700   2.243113   4.248410   .004634 ( 5)
16    1   .5175733E-05   .172387   2.224126   4.198796   .004794 ( 5)
17    1   .5786571E-05   .176890   2.201875   4.140651   .004990 ( 5)
18    1   .6567679E-05   .182173   2.176993   4.075630   .005222 ( 5)
19    1   .7450395E-05   .187625   2.152600   4.011889   .005464 ( 5)
20    1   .8442363E-05   .193224   2.128798   3.949692   .005716 ( 5)
21    1   .9649611E-05   .199434   2.103756   3.884253   .005997 ( 5)
22    1   .1117308E-04   .206523   2.076767   3.813727   .006323 ( 5)
23    1   .1305955E-04   .214403   2.048581   3.740074   .006688 ( 5)
24    1   .1532536E-04   .222863   2.020242   3.666021   .007085 ( 5)
25    1   .1806137E-04   .231969   1.991729   3.591513   .007518 ( 5)
26    1   .2145109E-04   .241988   1.962495   3.515120   .008000 ( 5)
27    1   .2605639E-04   .253949   1.930186   3.430694   .008582 ( 5)
28    1   .3138540E-04   .266064   1.900001   3.351817   .009180 ( 5)
29    1   .3727448E-04   .277881   1.872723   3.280536   .009770 ( 5)
30    1   .4341805E-04   .288897   1.849009   3.218566   .010326 ( 5)
```

Screen 46. Continued.

...Grid adjusted after iter = 60

i	j	hmin1	norm. dist.	wall stretch	max. stretch	location of maximum
1	1	.3508999E-05	.157739	2.304114	4.407815	.004169 (5)
2	1	.3354738E-05	.156143	2.313620	4.432655	.004102 (5)
3	1	.3344006E-05	.156030	2.314300	4.434431	.004098 (5)
4	1	.3339437E-05	.155982	2.314590	4.435188	.004096 (5)
5	1	.3350345E-05	.156097	2.313898	4.433381	.004100 (5)
6	1	.3404264E-05	.156661	2.310516	4.424542	.004124 (5)
7	1	.3503191E-05	.157680	2.304464	4.408728	.004167 (5)
8	1	.3640314E-05	.159058	2.296386	4.387619	.004224 (5)
9	1	.3796423E-05	.160581	2.287596	4.364650	.004289 (5)
10	1	.3991909E-05	.162426	2.277146	4.337342	.004367 (5)
11	1	.4272193E-05	.164960	2.263126	4.300707	.004474 (5)
12	1	.4685001E-05	.168482	2.244259	4.251406	.004625 (5)
13	1	.5097616E-05	.171784	2.227184	4.206786	.004767 (5)
14	1	.5487607E-05	.174731	2.212417	4.168198	.004895 (5)
15	1	.5779021E-05	.176836	2.202134	4.141326	.004987 (5)
16	1	.6047434E-05	.178709	2.193165	4.117891	.005069 (5)
17	1	.6394840E-05	.181045	2.182201	4.089240	.005172 (5)
18	1	.6905384E-05	.184318	2.167248	4.050166	.005317 (5)
19	1	.7591859E-05	.188455	2.148994	4.002467	.005501 (5)
20	1	.8422231E-05	.193115	2.129250	3.950871	.005711 (5)
21	1	.9356654E-05	.197980	2.109495	3.899251	.005931 (5)
22	1	.1032634E-04	.202675	2.091215	3.851483	.006146 (5)
23	1	.1139707E-04	.207506	2.073150	3.804276	.006368 (5)
24	1	.1272952E-04	.213086	2.053168	3.752060	.006627 (5)
25	1	.1439538E-04	.219506	2.031263	3.694821	.006927 (5)
26	1	.1631982E-04	.226297	2.009260	3.637323	.007248 (5)
27	1	.1871878E-04	.234010	1.985600	3.575498	.007616 (5)
28	1	.2174829E-04	.242811	1.960183	3.509079	.008039 (5)
29	1	.2533116E-04	.252170	1.934827	3.442820	.008495 (5)
30	1	.2964231E-04	.262271	1.909196	3.375843	.008992 (5)

Screen 46. Continued.

...Grid adjusted after iter = 80

i	j	hmin1	norm. dist.	wall stretch	max. stretch	location of maximum
1	1	.3443899E-05	.157072	2.308067	4.418145	.004141 (5)
2	1	.3344942E-05	.156040	2.314240	4.434276	.004098 (5)
3	1	.3335160E-05	.155936	2.314862	4.435899	.004094 (5)
4	1	.3350440E-05	.156098	2.313892	4.433365	.004100 (5)
5	1	.3385750E-05	.156468	2.311670	4.427559	.004116 (5)
6	1	.3459766E-05	.157235	2.307096	4.415607	.004148 (5)
7	1	.3586949E-05	.158526	2.299488	4.395726	.004202 (5)
8	1	.3762755E-05	.160256	2.289457	4.369513	.004275 (5)
9	1	.3981143E-05	.162326	2.277706	4.338807	.004362 (5)
10	1	.4236447E-05	.164644	2.264856	4.305227	.004461 (5)
11	1	.4544717E-05	.167311	2.250453	4.267592	.004575 (5)
12	1	.4961024E-05	.170713	2.232659	4.221092	.004721 (5)
13	1	.5386835E-05	.173984	2.216117	4.177866	.004863 (5)
14	1	.5902483E-05	.177705	2.197951	4.130398	.005025 (5)
15	1	.6528443E-05	.181919	2.178161	4.078684	.005211 (5)
16	1	.7167620E-05	.185932	2.160043	4.031340	.005389 (5)
17	1	.7747829E-05	.189357	2.145106	3.992305	.005542 (5)
18	1	.8422861E-05	.193118	2.129235	3.950834	.005711 (5)
19	1	.9293064E-05	.197661	2.110768	3.902576	.005917 (5)
20	1	.1031082E-04	.202602	2.091492	3.852207	.006142 (5)
21	1	.1144234E-04	.207703	2.072429	3.802392	.006377 (5)
22	1	.1268032E-04	.212888	2.053863	3.753877	.006617 (5)
23	1	.1402284E-04	.218118	2.035905	3.706950	.006862 (5)
24	1	.1551981E-04	.223546	2.018033	3.660248	.007118 (5)
25	1	.1731760E-04	.229596	1.998973	3.610441	.007405 (5)
26	1	.1948574E-04	.236328	1.978750	3.557596	.007727 (5)
27	1	.2071425E-04	.239910	1.968385	3.530511	.007899 (5)
28	1	.2373317E-04	.248119	1.945602	3.470978	.008297 (5)
29	1	.2726277E-04	.256834	1.922780	3.411340	.008724 (5)
30	1	.3143247E-04	.266164	1.899761	3.351189	.009185 (5)

Screen 46. Continued.

...Grid adjusted after iter = 100

i	j	hmin1	norm. dist.	wall stretch	max. stretch	location of maximum
1	1	.3029219E-05	.152591	2.335401	4.489572	.003955 (5)
2	1	.2933087E-05	.151489	2.342341	4.507707	.003909 (5)
3	1	.2936879E-05	.151533	2.342063	4.506979	.003911 (5)
4	1	.2960495E-05	.151806	2.340337	4.502469	.003922 (5)
5	1	.3003706E-05	.152301	2.337219	4.494321	.003943 (5)
6	1	.3081789E-05	.153183	2.331710	4.479927	.003979 (5)
7	1	.3205760E-05	.154550	2.323279	4.457895	.004036 (5)
8	1	.3374143E-05	.156346	2.312398	4.429461	.004111 (5)
9	1	.3584373E-05	.158500	2.299639	4.396121	.004201 (5)
10	1	.3842867E-05	.161026	2.285059	4.358021	.004307 (5)
11	1	.4162814E-05	.163986	2.268471	4.314673	.004433 (5)
12	1	.4581392E-05	.167620	2.248814	4.263307	.004588 (5)
13	1	.5011667E-05	.171112	2.230609	4.215737	.004738 (5)
14	1	.5448393E-05	.174442	2.213848	4.171937	.004883 (5)
15	1	.5972503E-05	.178192	2.195623	4.124314	.005047 (5)
16	1	.6650755E-05	.182708	2.174544	4.069232	.005246 (5)
17	1	.7315495E-05	.186823	2.156110	4.021062	.005429 (5)
18	1	.7972187E-05	.190633	2.139664	3.978085	.005599 (5)
19	1	.8791758E-05	.195083	2.121156	3.929722	.005800 (5)
20	1	.9810880E-05	.200221	2.100679	3.876214	.006033 (5)
21	1	.1100337E-04	.205769	2.079562	3.821031	.006288 (5)
22	1	.1234660E-04	.211527	2.058660	3.766412	.006554 (5)
23	1	.1386875E-04	.217536	2.037866	3.712074	.006835 (5)
24	1	.1561514E-04	.223879	2.016961	3.657447	.007133 (5)
25	1	.1764181E-04	.230639	1.995773	3.602080	.007454 (5)
26	1	.2002684E-04	.237925	1.974096	3.545436	.007804 (5)
27	1	.2206557E-04	.243682	1.957754	3.502732	.008082 (5)
28	1	.2543164E-04	.252418	1.934175	3.441117	.008507 (5)
29	1	.2930744E-04	.261525	1.911032	3.380641	.008955 (5)
30	1	.3366429E-04	.270807	1.888812	3.322577	.009416 (5)

Screen 46. Concluded.

10.3. File `conv.out`

The file `conv.out` contains a running convergence history for the `laura` runs within the `LOCAL` directory. With each `laura` run, new information is appended to this file. The output for the initial run of the sample case is displayed and discussed in the pages that follow.

A header identifies the quantities contained in this output:

`tsk`	task number
`iter`	iteration number
`L2 norm`	running total of $L2$ norms for all tasks
`time`	CPU time, s

In addition, surface pressure values at two stations are output for each iteration (screen 47).

NOTE: For viscous calculations, surface heating is output at these same stations.

tsk	iter	residual	time(sec)	body pressure (stag)	body pressure (end)	body heating (stag)	body heating (end)
1	1	.111251E+02	0.490	.576182E-02	.576182E-02	0.	0.
1	2	.380023E+02	0.960	.731007E-02	.577053E-02	-.123807E-05	-.382043E-07
1	3	.119971E+02	1.440	.182999E-01	.580280E-02	-.177536E-04	-.107455E-06
1	4	.210671E+02	1.910	.310216E-01	.584124E-02	-.248138E-04	-.175231E-06
1	5	.946205E+01	2.400	.522144E-01	.593069E-02	-.358498E-04	-.298696E-06
1	6	.145876E+02	2.880	.744622E-01	.602070E-02	-.357327E-04	-.413637E-06
1	7	.756324E+01	3.350	.106214	.617230E-02	-.486654E-04	-.597005E-06
1	8	.109528E+02	3.830	.137500	.631392E-02	-.477209E-04	-.760025E-06
1	9	.620669E+01	4.300	.180147	.652168E-02	-.611696E-04	-.997261E-06
1	10	.855677E+01	4.770	.219581	.670997E-02	-.596799E-04	-.120258E-05
1	11	.522298E+01	5.250	.271032	.696753E-02	-.727976E-04	-.148395E-05
1	12	.685348E+01	5.720	.316603	.719903E-02	-.706604E-04	-.172464E-05
1	13	.448989E+01	6.200	.374367	.750296E-02	-.825884E-04	-.204073E-05
1	14	.558383E+01	6.670	.423743	.777719E-02	-.797326E-04	-.231121E-05
1	15	.392410E+01	7.150	.485712	.812735E-02	-.898725E-04	-.265473E-05
1	16	.461184E+01	7.620	.536903	.844679E-02	-.865855E-04	-.295223E-05
1	17	.347849E+01	8.100	.600625	.884606E-02	-.946827E-04	-.331940E-05
1	18	.385990E+01	8.570	.651774	.921598E-02	-.914467E-04	-.364512E-05
1	19	.312473E+01	9.050	.713986	.966994E-02	-.975128E-04	-.403683E-05
1	20	.327746E+01	9.520	.762802	.100980E-01	-.947210E-04	-.439681E-05
1	21	.387234E+01	10.140	.819925	.106146E-01	-.320804	-.672383E-03
1	22	.251388E+01	10.620	.664872	.115725E-01	-.145658	-.908540E-03
1	23	.187591E+01	11.100	.625742	.111105E-01	-.108364	-.743926E-03
1	24	.205793E+01	11.580	.644236	.106750E-01	-.931990E-01	-.602291E-03
1	25	.148929E+01	12.060	.658604	.104534E-01	-.798007E-01	-.500528E-03

Screen 47.

NOTE: The pressure and heating output for this file is for Surface #1, assuming the following:

- $i = 1$ for the stagnation line

- $i = iaq$ at the aftmost point of the body
- $j = jaq$ is the leeside symmetry plane
- $k = 1$ is the body surface

If these assumptions are true, then the output contains leeside values at the stagnation and body-end locations. This output can be altered in a `LOCAL` version of file `swptask.F` (screen 48).

```
1   26 .187973E+01    12.540 .685238     .104241E-01 -.742116E-01 -.448847E-03
1   27 .141504E+01    13.020 .712985     .105541E-01 -.691393E-01 -.411082E-03
1   28 .182014E+01    13.490 .741987     .107443E-01 -.661469E-01 -.391369E-03
1   29 .142090E+01    13.970 .779123     .111720E-01 -.646882E-01 -.390305E-03
1   30 .168131E+01    14.460 .812621     .115689E-01 -.629977E-01 -.387193E-03
1   31 .135707E+01    14.930 .855967     .122886E-01 -.630703E-01 -.411887E-03
1   32 .144442E+01    15.410 .892370     .128973E-01 -.620696E-01 -.420021E-03
1   33 .120762E+01    15.890 .936180     .138827E-01 -.624471E-01 -.463032E-03
1   34 .116881E+01    16.360 .971330     .146829E-01 -.617338E-01 -.479173E-03
1   35 .102043E+01    16.840 1.00944     .158844E-01 -.617355E-01 -.534729E-03
1   36 .912701E+00    17.310 1.03906     .168356E-01 -.610563E-01 -.556667E-03
1   37 .845157E+00    17.780 1.06690     .181956E-01 -.603972E-01 -.620733E-03
1   38 .707287E+00    18.260 1.08814     .192521E-01 -.596093E-01 -.647104E-03
1   39 .706961E+00    18.730 1.10360     .207174E-01 -.582580E-01 -.717083E-03
1   40 .558819E+00    19.210 1.11545     .218388E-01 -.572949E-01 -.747088E-03
1   41 .264632E+01    19.840 1.11871     .233640E-01 -.149085     -.180440E-02
1   42 .138680E+01    20.310 .990941     .227179E-01 -.877278E-01 -.121301E-02
1   43 .108160E+01    20.790 .975642     .234899E-01 -.705461E-01 -.110537E-02
1   44 .759772E+00    21.260 .995075     .244906E-01 -.628068E-01 -.105261E-02
1   45 .675046E+00    21.730 1.01105     .261073E-01 -.582437E-01 -.113698E-02
1   46 .521515E+00    22.200 1.02706     .273711E-01 -.557855E-01 -.116241E-02
1   47 .534753E+00    22.680 1.03041     .290843E-01 -.525408E-01 -.128204E-02
1   48 .405861E+00    23.150 1.03411     .303302E-01 -.507998E-01 -.132489E-02
1   49 .458732E+00    23.620 1.02696     .319379E-01 -.477138E-01 -.144554E-02
1   50 .338284E+00    24.100 1.02335     .330659E-01 -.461257E-01 -.149167E-02
1   51 .422194E+00    24.570 1.01128     .344691E-01 -.432762E-01 -.160161E-02
1   52 .307171E+00    25.050 1.00451     .354251E-01 -.418525E-01 -.164569E-02
1   53 .414815E+00    25.520 .991727     .365875E-01 -.394433E-01 -.174167E-02
1   54 .301292E+00    25.990 .984499     .373586E-01 -.382823E-01 -.178186E-02
1   55 .424912E+00    26.440 .973628     .382930E-01 -.363805E-01 -.186485E-02
1   56 .310424E+00    26.910 .967772     .388980E-01 -.354510E-01 -.190109E-02
1   57 .444238E+00    27.380 .960182     .396481E-01 -.340157E-01 -.197423E-02
1   58 .328462E+00    27.850 .956560     .401247E-01 -.332922E-01 -.200740E-02
1   59 .468710E+00    28.310 .952188     .407444E-01 -.322725E-01 -.207419E-02
1   60 .351790E+00    28.780 .950565     .411335E-01 -.317431E-01 -.210541E-02
```

Screen 48.

NOTE: The evaluation of surface-property convergence histories at key stations provides the user a measure of overall solution quality.

```
1   61 .850596E+00    29.410 .948460     .416678E-01 -.286832E-01 -.344605E-02
1   62 .547339E+00    29.880 .948563     .417388E-01 -.284228E-01 -.339051E-02
1   63 .634605E+00    30.360 .948449     .410546E-01 -.280350E-01 -.303522E-02
1   64 .467012E+00    30.840 .949105     .409112E-01 -.278070E-01 -.287296E-02
1   65 .602136E+00    31.300 .948940     .413617E-01 -.274168E-01 -.281162E-02
1   66 .459770E+00    31.770 .949398     .418904E-01 -.271860E-01 -.277444E-02
1   67 .602692E+00    32.250 .948551     .425978E-01 -.267535E-01 -.280622E-02
1   68 .467766E+00    32.730 .948434     .431159E-01 -.265035E-01 -.281138E-02
1   69 .613552E+00    33.200 .946742     .436512E-01 -.260277E-01 -.285886E-02
1   70 .481572E+00    33.690 .945992     .439743E-01 -.257603E-01 -.287497E-02
1   71 .628930E+00    34.160 .943582     .442799E-01 -.252590E-01 -.291761E-02
1   72 .497649E+00    34.640 .942315     .444268E-01 -.249838E-01 -.293502E-02
1   73 .645697E+00    35.110 .939434     .445511E-01 -.244791E-01 -.296840E-02
1   74 .513746E+00    35.580 .937832     .445759E-01 -.242066E-01 -.298386E-02
1   75 .661562E+00    36.050 .934748     .445759E-01 -.237177E-01 -.300797E-02
1   76 .528288E+00    36.530 .933003     .445233E-01 -.234567E-01 -.302049E-02
1   77 .674542E+00    37.000 .929956     .444426E-01 -.229982E-01 -.303656E-02
1   78 .539965E+00    37.470 .928232     .443426E-01 -.227547E-01 -.304610E-02
1   79 .683141E+00    37.950 .925408     .442109E-01 -.223357E-01 -.305567E-02
1   80 .547736E+00    38.420 .923830     .440827E-01 -.221133E-01 -.306261E-02
1   81 .947471E+00    39.050 .921340     .439193E-01 -.196904E-01 -.288311E-02
1   82 .626324E+00    39.520 .920000     .438014E-01 -.195132E-01 -.289676E-02
1   83 .713803E+00    39.990 .918149     .436628E-01 -.192458E-01 -.290991E-02
1   84 .554506E+00    40.470 .917174     .435304E-01 -.190964E-01 -.291886E-02
1   85 .671584E+00    40.940 .915432     .433683E-01 -.188434E-01 -.292608E-02
1   86 .532733E+00    41.420 .914459     .432243E-01 -.187013E-01 -.293225E-02
1   87 .644253E+00    41.890 .912434     .430440E-01 -.184372E-01 -.293489E-02
1   88 .514715E+00    42.370 .911200     .428909E-01 -.182908E-01 -.293897E-02
1   89 .618159E+00    42.840 .909047     .426969E-01 -.180330E-01 -.293804E-02
1   90 .495591E+00    43.320 .907726     .425373E-01 -.178930E-01 -.294051E-02
1   91 .591651E+00    43.800 .906105     .423334E-01 -.176850E-01 -.293673E-02
1   92 .475097E+00    44.270 .905235     .421696E-01 -.175700E-01 -.293793E-02
1   93 .564757E+00    44.750 .904807     .419585E-01 -.174521E-01 -.293179E-02
1   94 .453562E+00    45.240 .904838     .417919E-01 -.173834E-01 -.293193E-02
1   95 .537530E+00    45.720 .905790     .415754E-01 -.173611E-01 -.292377E-02
1   96 .431251E+00    46.190 .906793     .414069E-01 -.173407E-01 -.292300E-02
1   97 .510013E+00    46.660 .908774     .411863E-01 -.173840E-01 -.291307E-02
1   98 .408316E+00    47.170 .910439     .410167E-01 -.173953E-01 -.291151E-02
1   99 .482306E+00    47.640 .912810     .407933E-01 -.174585E-01 -.290007E-02
1  100 .384926E+00    48.120 .914660     .406232E-01 -.174777E-01 -.289783E-02
```

Screen 48. Concluded.

10.4. File `grid.out`

The file `grid.out` contains information about grid resolution at each surface defined by the user. This file is overwritten at the conclusion of each `laura` run. The output for the initial run of the sample case is displayed and discussed here.

Each surface of each computational block has its own section, which is annotated with the number of cells normal to the body through the message "Block _, Surface _ (__ cells normal to body)." A header identifies the quantities contained in the following output (screen 49):

`i, j`	station index
`dh_w`	height (dimension in k-direction) of cell adjacent to wall
`Recell_w`	value of cell Reynolds number (*recell*) at the wall for this station
`max stretch`	maximum value of grid stretching factor used in the stretching region (k-index of location in parentheses)
`y+_w`	value of normal coordinate parameter (y^+) at the wall for this station (*nturb* $\neq 0$ only)

Block 1, Surface 1
(16 cells normal to body)

i	j	dh_w	Recell_w	max. stretch
1	1	.321195E-05	6.958525	4.49 (6)
2	1	.328374E-05	7.247622	4.50 (6)
3	1	.338243E-05	7.303569	4.51 (6)
4	1	.348394E-05	7.302690	4.50 (6)
5	1	.359983E-05	7.295987	4.49 (6)
6	1	.374088E-05	7.284156	4.48 (6)
7	1	.389317E-05	7.222345	4.46 (6)
8	1	.407609E-05	7.150728	4.43 (6)
9	1	.429236E-05	7.064906	4.39 (6)
10	1	.457386E-05	6.984239	4.36 (6)
11	1	.497868E-05	6.957453	4.31 (6)
12	1	.551343E-05	6.928688	4.26 (6)
13	1	.610366E-05	6.945080	4.21 (6)
14	1	.665580E-05	6.854008	4.17 (6)
15	1	.737871E-05	6.824148	4.12 (6)
16	1	.818040E-05	6.740974	4.07 (6)
17	1	.903207E-05	6.585194	4.02 (6)
18	1	.987771E-05	6.303515	3.97 (6)
19	1	.111265E-04	6.146283	3.93 (6)
20	1	.125892E-04	5.961880	3.87 (6)
21	1	.144564E-04	5.827547	3.82 (6)
22	1	.166364E-04	5.676118	3.76 (6)
23	1	.191056E-04	5.484815	3.71 (6)
24	1	.221322E-04	5.323339	3.65 (6)
25	1	.258567E-04	5.191367	3.60 (6)
26	1	.302151E-04	5.039758	3.55 (6)
27	1	.352713E-04	5.029430	3.49 (6)
28	1	.418457E-04	4.834682	3.44 (6)
29	1	.498752E-04	4.627432	3.38 (6)
30	1	.587542E-04	4.395599	3.33 (6)

Screen 49.

10.5. Post-Processing Files

Six files that allow graphical presentation of results are created at the conclusion of every LAURA run. These files are in `PLOT3D` format, but they can be easily converted to Tecplot^{TM} format (ref. 27). The files `flow1.g` and `surf1.g` contain the volume grid and surface grid, respectively, written in a multiblock format (even when only one block is present).

The file `flow1.q` contains the nondimensional values of density, u-, v-, and w-components of velocities (in the x-, y-, and z-directions, respectively), and pressure.

> **NOTE:** Although computations are performed at cell centers by `laura`, the values in the file `flow1.q` have been interpolated/extrapolated to cell corners for the entire volume grid.

> **NOTE:** If the file `RESTART.in` is produced by `PRELUDE`, the initial values of these flow field variables are written to files `flow0.g` and `flow0.q`. This is also the case if `INITIALIZE` is used to initialize an externally generated grid.

The file `flow2.q` contains the translational and vibrational temperatures [K], the nondimensional total enthalpy, the Mach number, and the ratio of frozen specific heats interpolated/extrapolated to cell corners for the entire volume grid. Any other computed quantity can be substituted in routine `plotprep.F`.

The first two entries of file `surf1.q` are p and q, the surface pressure [lbf/ft^2] and heating rate [Btu/ft^2-s], respectively. For viscous ($igovern \neq 0$), radiative equilibrium wall temperature ($tempbc = 2$) flows, the last three entries are the:

- surface temperature, K
- radiative equilibrium wall temperature, based on the local heating rate and an assumed emissivity of 0.9
- relative difference between the actual wall temperature and the radiative equilibrium wall temperature

Otherwise (for inviscid and/or fixed wall temperature flows), the last three entries are p/p_{max}, q/q_{max}, and p/p_∞. The first three entries of file `surf2.q` are the x-, y-, and z-components of the shear stress on the body surface. The fourth entry is the total shear stress, and the fifth is the coefficient of skin friction.

To convert these files to Tecplot^{TM} format, use the command

```
preplot infile -plot3d -f -m -3dw
```

along with the desired `-ip`, `-jp`, or `-kp` instructions (ref. 27), where

$$infile = \begin{cases} \texttt{flow1} \\ \texttt{flow2} \\ \texttt{surf1} \\ \texttt{surf2} \end{cases}$$

> **NOTE:** The command `preplot` is a preprocessor that is part of the Tecplot^{TM} package.

Chapter 11

Advanced Applications

11.1. Grid Orientation

Although not a requirement, in hypersonic blunt-body applications, the origin of the coordinate system generally sits at or near the stagnation point on the body, with the z-axis pointing out from the body toward the oncoming flow, as shown in figure 2.1. The $y = 0$ plane defines the symmetry plane. Lifting-body applications retain this orientation, with the origin of the coordinate system at or near the vehicle stagnation point, the z-axis pointing out from the nose, against the flow, and the negative z-axis typically running through the interior of the vehicle.

Computational coordinates (ξ, η, ζ) run in the direction of increasing i-, j-, and k-indices, respectively, as shown in figure 2.2. The vehicle surface grid is usually defined by the $k = 1$ plane because the shock alignment and cell-doubling features of LAURA require that the k-coordinate be in the body-normal direction.

Axisymmetric and two-dimensional flows are computed with three-dimensional grids, which contain a single j-plane of cell centers and two j-planes of cell walls. A representative axisymmetric surface grid ($k = 1$) is presented in figure 11.1. The pie-shaped cut is bounded by the $j = 1$ plane (negative y-coordinate) and the $j = 2$ plane (positive y-coordinate). The i index equals 1 at the axis ($[x, y] = [0, 0]$) and increases from there toward the end of the body. The j-planes are rotated ± 2.5 deg around the z-axis relative to the $y = 0$ plane for a total included angle of 5 deg. In the case of a two-dimensional configuration, the j-planes are offset from the $y = 0$ plane by ± 1 unit. The volume grid is constructed by defining rays normal to the body in the $y = 0$ plane and rotating ± 2.5 deg for axisymmetric geometries or offsetting ± 1 unit for two-dimensional geometries. This orientation is the only one permitted for axisymmetric and two-dimensional flows.

A representative three-dimensional surface grid ($k = 1$) is presented in figure 11.2. The $j = 1$ plane defines the upper (leeside) plane of symmetry. The $j = jblk_{nblk}$ plane defines the lower (windside) plane of symmetry. The $i = 1$ index defines the axis and increases from there toward the outflow boundary at $i = iblk_{nblk}$.

The two examples presented above contain an axis singularity (cell wall with zero area). Grids can be created which remove the singularity at the nose at the expense of creating skewed cells on the outflow boundary. A representative, singularity free surface grid for a blunted geometry is presented in figures 11.3 to 11.5. In this example, the $j = 1$ plane defines the plane of symmetry. The $j = jblk_{nblk}$ plane defines the outflow boundary from approximately 45 deg to 135 deg around the center of the base (fig. 11.4). The $i = 1$ plane defines the outflow boundary from the symmetry plane (0 deg) to the beginning of the $j = jblk_{nblk}$ plane. The $i = iblk_{nblk}$ plane defines the outflow boundary from the end of the $j = jblk_{nblk}$ plane to the symmetry plane

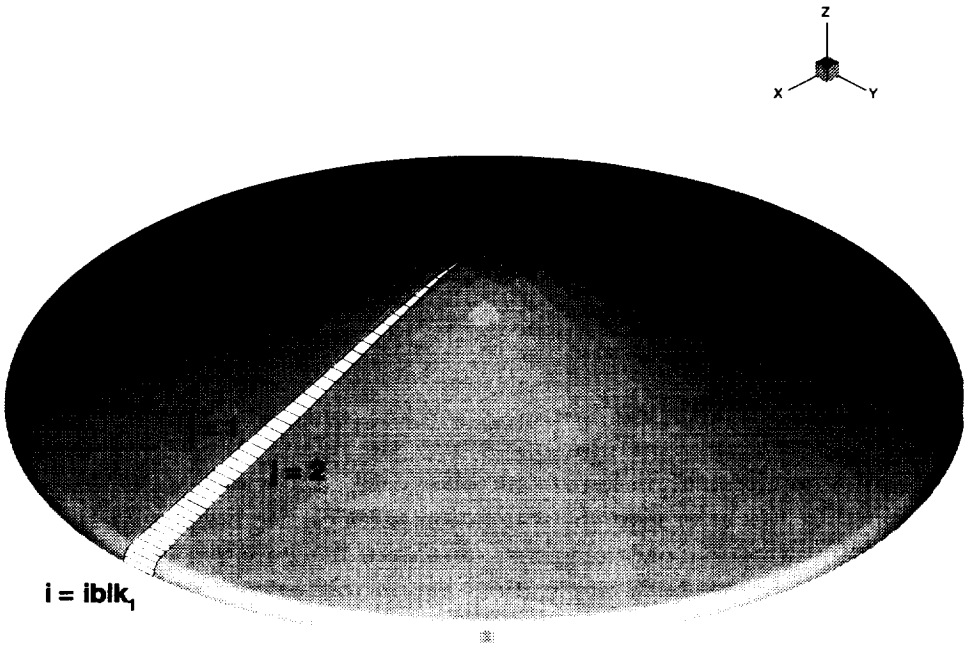


Figure 11.1. Surface grid on 70-deg spherically capped cone with rounded shoulder.

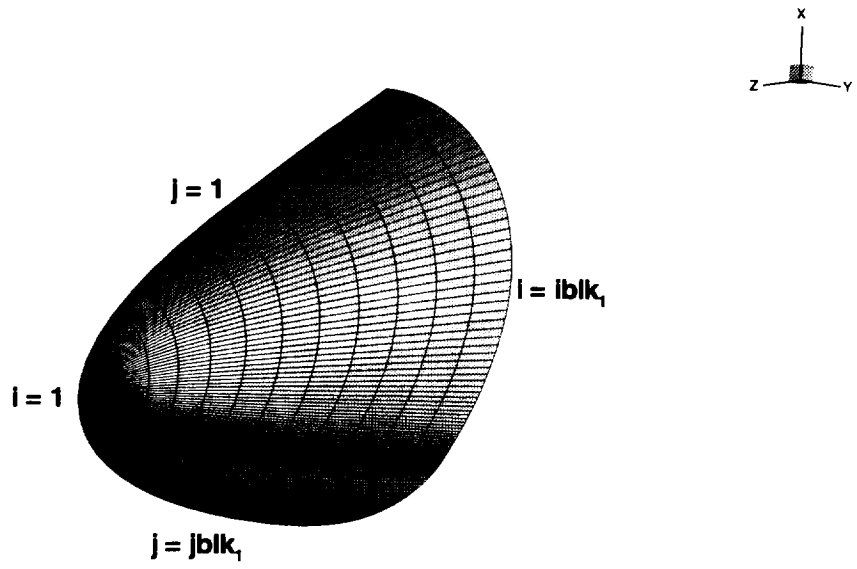


Figure 11.2. Surface grid on nose of Space Shuttle.

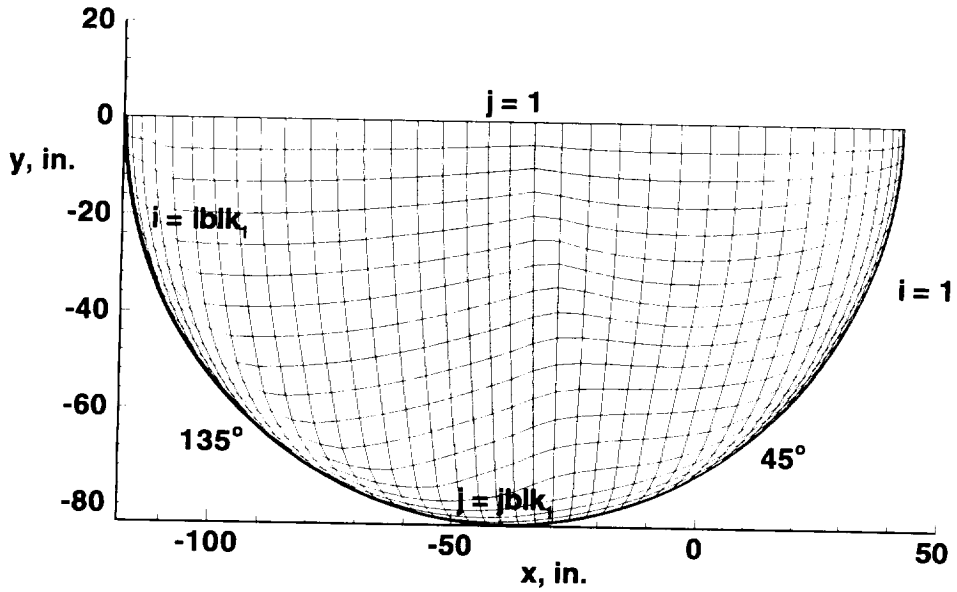


Figure 11.3. Projection of singularity free surface grid over blunt body on xy-plane.

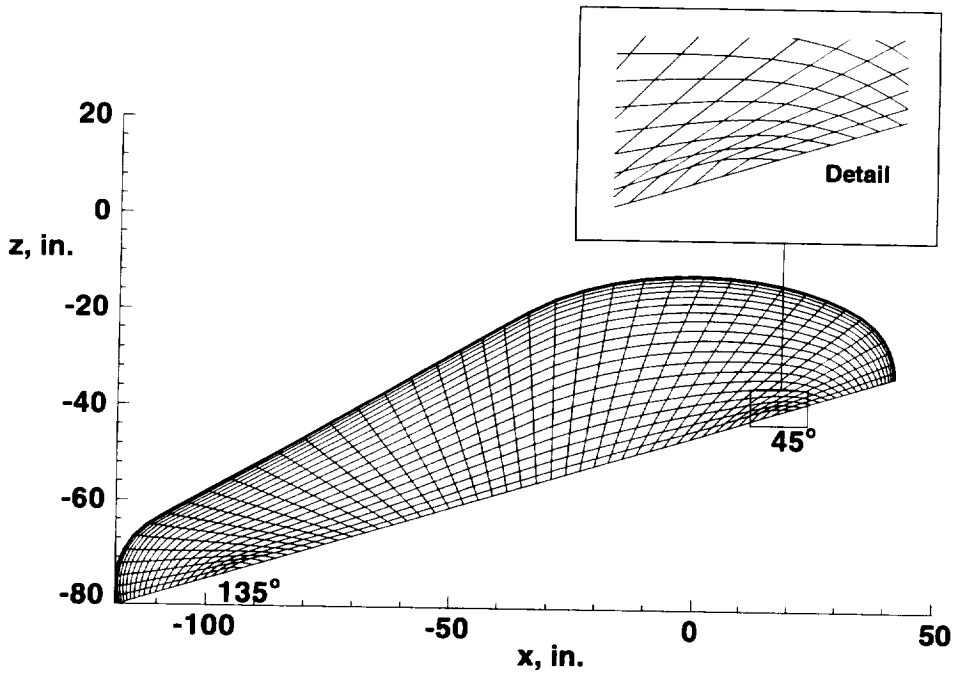


Figure 11.4. Projection of singularity free surface grid over blunt body on xz-plane.

180 deg around the base. The surface grid cells on the outflow boundary at the 45-deg and 135-deg locations appear triangular. (See the inset of fig. 11.4.) In fact, one face is composed of two families of coordinate lines.

11.1.1. Boundary-Layer and Shock Grid Adaption

Resolution of the captured bow shock is required to evaluate the influence of thermochemical nonequilibrium on radiative heating. Bow-shock resolution is potentially as important to

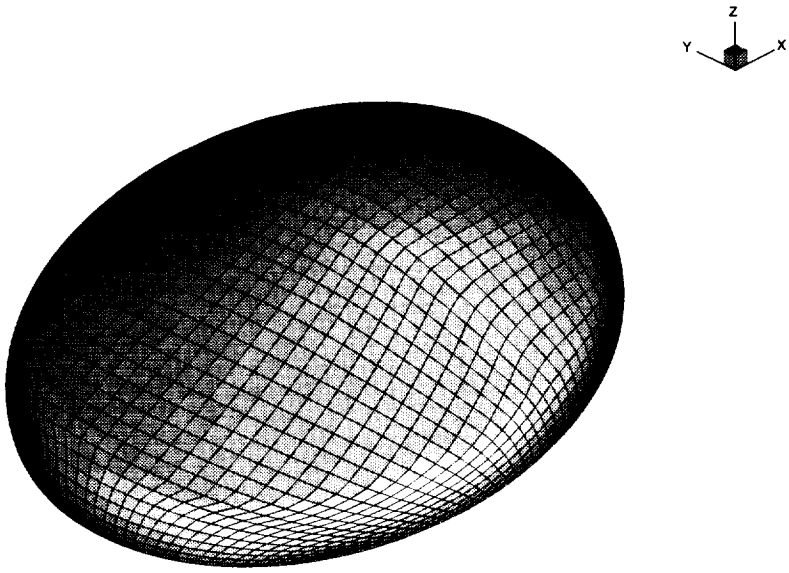


Figure 11.5. Singularity free surface grid over blunt body.

accurate radiative heating predictions as boundary-layer resolution is to convective heating predictions. Consequently, a simple algebraic grid adaption routine is used to distribute computational cells across the shock layer according to four quantities ($recell_w$, *fstr*, *cp0*, and *fsh*) which are discussed below.

First, the cell size at the body is set to yield a specified cell Reynolds number ($recell_w$) where

$$recell = \frac{\rho a \Delta s}{\mu}$$

and Δs is the cell height. Second, the fraction of cells available for boundary-layer resolution *fstr* is specified. Third, the concentration of cells at the shock front is controlled with the parameter *cp0*. Fourth, the outer boundary is adjusted to force the location of the captured shock in physical space to lie at a constant fraction *fsh* of the length of the ζ-coordinate line extending from the body, across the boundary layer and shock, and to the free-stream inflow boundary. The shock location is defined by the first point with a local property that exceeds its free-stream value by a specified amount.

Let $s^{[1]}_{i,j}(k)$ denote the present (superscript [1]) arc length from the body surface to the kth cell center along a ζ-coordinate line defined by K cells, where $k \leq K$. For simplicity, we drop the i, j subscripts in subsequent notation, but keep in mind that this procedure is followed for each ζ-coordinate line. Let

$$\hat{s}(k) = \frac{s^{[2]}(k)}{s^{[1]}(K)}$$

denote the nondimensional length of the adapted grid (superscript [2]) along the ζ-coordinate line, and let

$$\Delta\hat{s}(k) = \hat{s}(k_+) - \hat{s}(k_-)$$

where

$$k_+ = k + \frac{1}{2} \qquad \text{and} \qquad k_- = k - \frac{1}{2}$$

The fractional values $(k \pm \frac{1}{2})$ refer to cell edges.

First, a transformation to resolve the boundary layer is defined. Define the nondimensional height of the first cell by

$$\Delta\hat{s}(1) = \frac{recell_w \mu(1)}{\rho(1)a(1)s^{[1]}(K)} \tag{11.1}$$

The height of the next $kstr$ cells ($kstr = fstr\ K$) is defined by

$$\Delta\hat{s}(k) = \min \begin{cases} \left[1 + C' \sin\left(\dfrac{[k-1]\pi}{kstr - 1}\right)\right] \Delta\hat{s}(k-1) \\ \dfrac{1 - \hat{s}(k_-)}{K + 1 - k} \end{cases} \tag{11.2}$$

where

$$C' = \left[\frac{fstr}{\Delta\hat{s}(1)}\right]^{\frac{1}{kstr}} - 1 \tag{11.3}$$

This function provides a cell growth factor of 1 at $k = 1$ and $k = kstr$ and a maximum growth factor equal to $1 + C'$ at $k = kstr/2$. It also precludes the nondimensional arc length $\hat{s}(K_+)$ from exceeding 1 by limiting the continued application of cell growth factors, if necessary. The remaining cells extending past $k = kstr$ are equally spaced, thus

$$\Delta\hat{s}(k) = \Delta\hat{s}(k-1)$$

for $k > kstr$. The distribution in $\hat{s}$ is obtained by summation,

$$\hat{s}(k_+) = \sum_{l=1}^{k} \Delta\hat{s}(l) \tag{11.4}$$

The stretching function defined above is designed to yield a value of $\hat{s}(K_+) = O(1)$ with continuous first derivatives of the cell growth factor. An additional renormalization,

$$\hat{s}(k_+) - \frac{\hat{s}(k_+)}{\hat{s}(K_+)}$$

forces the distribution to span 0 and 1 even when the $\hat{s}(K_+) < 1$.

> **NOTE:** This renormalization will increase the effective cell Reynolds number, possibly beyond the range where boundary-layer resolution is adequate for heat transfer predictions. A warning message is written to standard output when the criteria for $recell_w$ is not satisfied. Cell Reynolds number information along the body is located in output file `grid.out`, which is created by `wrapup.F`. Values of $recell$ across the entire shock layer also can be viewed in the column labeled "`Re_cell`" of the output generated by `outputa.F`. This routine is not generally called, but can be engaged by removing the comment from the call statement in `wrapup.F`.

> **NOTE:** Users should rely on grid refinement studies as the primary tool for evaluating resolution quality.

A second transformation to $\tilde{s}(k_+)$ is designed to pull points toward the $\tilde{s} = fsh$ location to resolve the shock front

$$\tilde{s}(k_+) = [1 - cp(k_+)]\,\hat{s}(k_+) + fsh\,cp(k_+) \tag{11.5}$$

where

$$ep(k_+) = \hat{s}^2(k_+)[1 - \hat{s}(k_+)]\, ep0 \tag{11.6}$$

The weighting here is designed to give very little change in the near-wall grid distribution and preserve the domain $0 \leq \hat{s} \leq 1$. Because $\hat{s}^2 << \hat{s}$ when $\hat{s}$ is close to 0 (near the wall), the value of ep is also very small, and the grid in the near wall region is hardly disturbed by this new mapping. In a similar manner, grid points near the inflow boundary (where $\hat{s}$ is close to 1) are also protected from large movements by keeping the value of ep small with the factor $[1 - \hat{s}(k_+)]$. The magnitude of *ep0* is limited to keep the grid from folding back over itself in the vicinity of *fsh* as mesh points are mapped into this region.

The third and final transformation returns dimensionality to the distribution. A scale factor is applied which adjusts the outer boundary such that the captured shock lies a constant fraction *fsh* of the distance $s(K_+)$ between the body and the outer boundary. Thus,

$$s^{[2]}(k) = \frac{s^{[1]}(*)\hat{s}(k)}{fsh} \tag{11.7}$$

where $s^{[1]}(*)$ is the location on the original grid where the captured shock is first sensed according to criteria discussed earlier.

> **NOTE:** In some cases, it is advantageous to adjust the outer boundary to force a specified distance between it and the captured shock. A simple redefinition of *fsh* for each coordinate line can accomplish this goal.

Interpolation and extrapolation are used to map the values of $\vec{r}\left\{s^{[1]}_{i,j}(k_+)\right\}$ to $\vec{r}\left\{s^{[2]}_{i,j}(k_+)\right\}$, where $\vec{r}$ is the vector of the Cartesian coordinates.

The shock location keys on the first point in from the free-stream inflow boundary, which satisfies the criterion

$$fctrjmp \times proprty_\infty \; - \; proprty \; < \; -1 \times 10^{-6}$$

where

$$proprty = \begin{cases} p & (jumpflag = 1) \\ \rho & (jumpflag = 2) \\ T & (jumpflag = 3) \end{cases}$$

> **NOTE:** Specifying *jumpflag* = 0 fixes the outer boundary at its current position. With both the body surface and the outer grid boundaries fixed, **algnshk.F** simply adjusts the cell distribution between them.

By default, LAURA uses pressure (p) to determine the shock position. However, limited tests discussed in reference 17 indicate that pressure is not always the best choice, and other options are provided for the user's convenience through the parameter *jumpflag*.

Another option for controlling mesh spacing across the shock layer will be employed when the parameter *betagrd* is set greater than 1. In this case, equations 11.1 to 11.4 are replaced by a simpler stretching function defined by

$$\hat{s}(k_+) = 1 \; - \; betagrd\left[\frac{bz - 1}{bz + 1}\right]$$

where

$$bz = \left(\frac{betagrd + 1}{betagrd - 1}\right)^{\frac{K_+ - k_+}{K_+ - 1}}$$

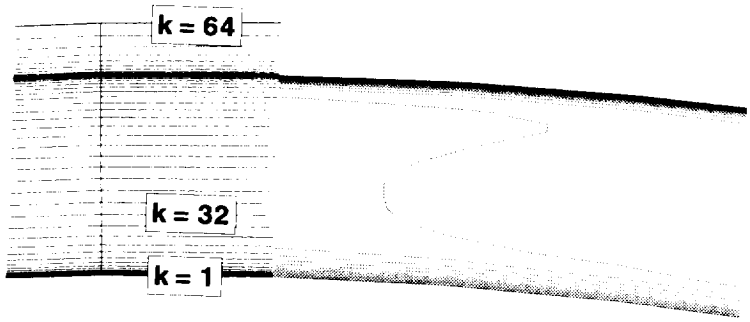


Figure 11.6. Detail of adapted grid and density contours in symmetry plane that shows enhanced resolution of the captured bow shock.

Table 11.1. Block Indices

block	A	B	C	D	E	G
nblk	1	2	3	4	5	6

The parameters for controlling grid distribution across the shock layer are defined in `algnshk_vars.strt`. Default values for LAURA are as follows:

$$\begin{aligned} recell_w &= 1.0 \\ jumpflag &= 1 \\ fctrjmp &= 1.5 \\ fsh &= 0.8 \\ fstr &= 0.5 \\ betagrd &= 0.0 \\ ep0 &= 0.0 \end{aligned}$$

These values can be changed in a `LOCAL` copy of `algnshk_vars.strt`, which can be created using the `LOCALIZE` command (appendix K).

NOTE: Since $ep0 = 0$, the default gridding yields no clustering at the shock.

In calculations discussed in reference 17, the best resolution of the shock front was obtained by keying on the first point in from the free stream where local density exceeded $1.5\rho_\infty$ ($jumpflag = 2$ and $fctrjmp = 1.5$) to sense shock location, in conjunction with setting $recell_w = 1.0$ and $ep0 = 25/4$ (which clusters cells about the shock). The values used for that case were $fstr = 0.5$ and $fsh = 0.8$. A representative grid that employs these mappings is shown in figure 11.6.

11.2. Multiple Computational Blocks

A computational block is a rectangularly ordered array of cells defining all or part of the solution domain. Six computational blocks are permitted in LAURA. Their designations are `A`, `B`, `C`, `D`, `E`, and `G`. As shown in table 11.1, these blocks have *nblk* values of 1, 2, 3, 4, 5, and 6, respectively.

NOTE: The natural designation of `F` for the sixth block is already occupied for other purposes.

Block-to-block connections assume simple extensions of the running index. For example, the $i = iblk_{nblk}$ plane of `Block X` (`X = A, B, C, D, E, G`) can be coincident with the $i = 1$ plane of any block. Further, the $i = 1$ plane of `Block Y` (`Y = A, B, C, D, E, G; X` $\neq$ `Y`) can be coincident with the $i = iblk_{nblk}$ plane of any block. More complex connections are not provided in this release. For example, the $i = 1$ plane cannot be coincident with the j- or k-plane of any other block nor can it coincide with the $i = 1$ plane of any other block. Also, any pair of blocks can share, at most, one boundary. Similar restrictions apply to j- and k-plane connections.

Multiple computational blocks can be used to define the flow field over complex configurations. In many instances the solution can be generated on a block-by-block basis; the nose region is solved first, and the windside and leeside blocks are solved subsequently. This approach reduces computational time and memory requirements by a factor of 10 or more. The solution state at the last, outflow plane of one block can be injected into the next downstream block for initialization. Further details on this approach are found in reference 28 and in appendix G.

The grid alignment capability in LAURA also can be used in the multi-block marching mode. A single block solution over the nose region can be grid aligned in the standard way. Grid alignment can be used in subsequent windside blocks under the special conditions that follow:

1. `Block A` is a single plane of cells ($iblk_1 = 1$) taken from the converged upstream block at or near its outflow boundary.

2. `Block B` defines the windside region to be solved and the inflow plane of `Block B` coincides with the outflow plane of `Block A`.

3. Only one active task, which is assigned to `Block B` exists.

Boundary conditions for `Block B` are as follows:

1. Shared boundary at $i = 1$ with `Block A`

2. Outflow boundary at $i = iblk_2$

3. Outflow (sideflow) boundary at $j = 1$

4. Plane of symmetry (no yaw) at $j = jblk_2$

5. Wall at $k = 1$

6. Free stream at $k = kblk_2$

LAURA will assume this configuration if $nblocks = 2$ (in `parameter.strt`) and $movegrd > 0$ (in `data`).

Grid alignment can be used in subsequent leeside blocks under the following special conditions:

1. `Block A` is a single plane of cells ($iblk_1 = 1$) taken from the converged upstream block at or near its outflow boundary.

2. `Block B` is a single plane of cells ($jblk_2 = 1$) taken from the converged windside block bounding the leeside block.

3. `Block C` defines the leeside region to be solved, and the inflow plane of `Block C` coincides with the outflow plane of `Block A` and the sideflow plane of `Block B`.

4. Only one active task, which is assigned to `Block C`, exists.

Boundary conditions for `Block C` are as follows:

1. Shared boundary at $i = 1$ with `Block A`

2. Outflow boundary at $i = iblk_3$

3. Plane of symmetry (no yaw) at $j = 1$

4. Shared sideflow boundary with `Block B` at $j = jblk_3$

5. Wall at $k = 1$

6. Free stream at $k = kblk_3$

LAURA will assume this configuration if $nblocks = 3$ (in `parameter.strt`) and $movegrd > 0$ (in `data`).

> **NOTE:** Users are cautioned that the very simplified, one-dimensional approach used for grid movement can introduce unacceptable grid skewness over complex configurations. Experienced users may find that it is possible to modify code in `algnshk.F` and/or `algnshk_vars.strt` to tailor alignment specifications and/or block orientations to suit their own applications.

11.3. Sweeping Options

The relaxation algorithm in LAURA updates the solution in a computational plane using the latest available data at neighboring planes and boundaries. For any cell within the computational plane, the relaxation process is a Jacobian function with respect to other cells in the plane and a Gauss-Seidel function with respect to cells in neighboring planes and boundaries. The global solution is obtained by sweeping across the domain, forward and backward, one plane at a time until the solution is converged.

In viscous hypersonic flow problems, the boundary layer is usually the slowest to converge, and sweeping strategies that pass across the boundary layer (using optimal viscous relaxation factors) provide the best convergence speeds in LAURA (ref. 18). A discussion of the viscous relaxation factor (*rfvis*) is presented at the end of section 9.1.3.2. As noted in section 11.1, the body surface is usually defined by the $k = 1$ plane so that the default sweep direction involves the sequential solution of k-planes. All the `FORTRAN parameter` definitions made with the execution of `PRELUDE` assume that only k-directional sweeps are required.

> **NOTE:** In certain advanced applications other sweep directions can be preferred (i.e., body surfaces defined on i- and/or j-planes). This can be most easily accomplished through file `assign_tasks` (section 9.1.2). In such cases, the value of *isjs* in a `LOCAL` version of file `parameter.strt` also can require modification.

The variable *isjs* (assigned in `parameter.strt`) must be greater than or equal to the maximum of the products

$$[L1 \times (L2 + 1)] \qquad \text{and} \qquad [(L1 + 1) \times L2]$$

where $L1$ and $L2$ are the number of cells in the two directions perpendicular to the sweep direction. LAURA compares these values with *isjs* for each block. If the maximum requirement for all blocks is less than *isjs*, a warning message is issued stating that *isjs* can be reduced to save memory. If any block requires more memory than *isjs* provides, an error message is issued and the job is terminated. In response to this situation, the user can increase the value of *isjs*, recompile with `make`, and repeat the LAURA execution. Sweeps in the i- and j-directions can be activated by overwriting the default k-directional sweep in the file `assign_tasks`, as described in the section 11.5.2.

> **NOTE:** If an axisymmetric solution is being calculated, specification of a circumferential sweeping direction yields a pure-Jacobian relaxation.

11.4. Solid-State-Device (SSD) Memory

The point-implicit relaxation strategy used in LAURA requires calculation of a Jacobian matrix at every computational cell. Because of the computational expense of evaluating and triangularizing this matrix every sweep, the LU-triangularization required for subsequent Gauss elimination is calculated once and stored (frozen) for *njcobian* sweeps before being updated. The memory overhead (in words) for this storage is large

$$neq^2 \times \sum_{n=1}^{nblocks} (iblk_n\, jblk_n\, kblk_n)$$

and can require the job to be run in a large memory queue with relatively poor turnaround time. CRAY computer systems are often equipped with fast, solid-state-device (SSD) memory that can be used to reduce the in-core memory requirements for the job. The process is equivalent to reading or writing the triangularized Jacobian to a fast disk drive once per solution sweep.

The SSD can be activated by setting *issd* = 1 (default: *issd* = 0) in a `LOCAL` copy of file `issd_assn.strt` (which can be found in the `STRTfiles` subdirectory after running `PRELUDE`). The executable will need to be recompiled by running `make`.

At this point, the triangularized Jacobian would be written to the disk on the executable directory. The working files now need to be assigned to the SSD. On some systems, space will need to be reserved in the SSD directory. One may calculate (in words) the required reservation, mem_{SSD}, using the formula

$$mem_{\mathrm{SSD}} = \left(isjs\; neq^2 \;+\; 512\right) \times \sum_{n=1}^{nblocks} LMAX_n \tag{11.8}$$

where $LMAX_n$ is the number of cells in the sweep direction for `Block X` (X = `A`, `B`, `C`, `D`, `E`, or `G`). The values of the other variables can be found in the file `parameter.strt`.

> **NOTE:** This reservation includes buffers needed for well-formed I/O and simplifications associated with multi-block `FORTRAN parameter` definition.

11.4.1. Interactive Jobs

The reservation for SSD memory in an interactive session is made on the Numerical Aerodynamic Simulation (NAS) facility CRAY C-90 with the command

```
srfs -r MEG_SSDMw $FASTDIR
```

where

$$MEG_{\mathrm{SSD}} = \mathtt{int}\left[\frac{mem_{\mathrm{SSD}}}{10^6}\right] + 1$$

NOTE: Typically, `$FASTDIR` is used in conjunction with $issd = 1$, while `$BIGDIR` is specified for $issd = 0$.

The working files are then assigned to the SSD with the command

```
assign -a $FASTDIR/scr80 -s u fort.80
```

where suffix 80 corresponds to `Block A`. If additional blocks are active, they would need to be assigned similarly, with suffixes 81 through 85 corresponding to `Blocks B` through `G` (excluding `F`, recall), respectively. The interactive session can now proceed as usual. When completed, the files on `$FASTDIR` should be removed, and the reservation concluded with the command

```
srfs -r0 $FASTDIR
```

11.4.2. Queued Jobs

The reservation for SSD memory in a queued job is made on the NAS CRAY C-90 with the command

```
# QSUB -lr '$FASTDIR,MEG_SSDMw'
```

placed at or near the beginning of the submit file. The variable MEG_{SSD} was defined in the previous subsection. The assignments are made via the following command lines:

```
/bin/rm -f .asgnf
setenv FILENV .asgnf
assign -a $FASTDIR/scr80 -s u fort.80
```

which are placed before the command

```
laura < data
```

Again, additional assignment statements are required when additional blocks are active, as discussed in the previous subsection.

NOTE: LAURA cannot accommodate different sweep directions in the same computational block when SSD memory is engaged.

11.5. Multitasking

Multitasking in a numerical algorithm refers to the capability of simultaneous (parallel) execution of different pieces of code on multiple central processing units (CPU's) for the purpose of decreasing elapsed wall clock time. The CRAY computers offer two options for multitasking.

The first option most often exploited by users is labeled microtasking. In `FORTRAN` codes, microtasking works on the "do-loop" level, thus spreading the work within a loop among several CPU's. The microtasking, in its simplest form (autotasking), can be implemented at compile

time with no changes required in the source code. Microtasking works well on the same loops that vectorize well, and in some situations (i.e. nested do-loops), compiler directives can be placed within the source code to further enhance parallelization and vectorization. (CRAY manuals should be consulted for more information on parallelization and vectorization.) LAURA makes extensive use of do-loops that can be vectorized and microtasked by the CRAY compiler in this way, although no special compiler directives are added to enhance microtasking.

> **NOTE:** Users who wish to invoke autotasking in LAURA on a CRAY computer should add the flag "`-Zp`" to the list of options for "`FFLAGS`" in the file `Makefile`, after running `PRELUDE` and before compiling with `make`.

The second option for multitasking is labeled macrotasking. Macrotasks can include large sections of code that are executed in parallel on multiple CPU's. Macrotasking requires the insertion of CRAY specific code to start tasks, synchronize tasks, and stop tasks as required in the algorithm. LAURA utilizes macrotasking by assigning pieces of the computational domain to individual tasks. Each task gathers and scatters data to a master copy of the solution which is saved in a shared memory, global common block. Subroutines are called by each task in parallel; consequently, task-specific data to be shared among subroutines are stored in "`task common`" blocks. The point-implicit relaxation used in LAURA makes it unnecessary to synchronize tasks after they are created so that very high levels of uninterrupted, parallel processing can be achieved. Sacrificing synchronization of tasks also means that the path to a converged solution is nondeterministic. Further discussion of asynchronous convergence is found in reference 18.

> **NOTE:** Users are cautioned against invoking both microtasking and macrotasking in the same job because more tasks can be created than the number of CPU's that are available, thus creating counterproductive contention among tasks for CPU's.

Macrotasking generally achieves higher levels of average concurrent CPU use than microtasking. However, there is a memory overhead per additional task (in words) approximately equal to

$$mem_{\text{task}} = \left(528 + 96ns + 6neq + 2neq^2\right) \times isjs \tag{11.9}$$

where neq is the number of governing equations being solved. This overhead (in words) is larger when the full Navier-Stokes equations are to be solved ($igovern = 2$):

$$mem_{\text{task}} = \left(538 + 98ns + 6neq + 2neq^2\right) \times isjs \tag{11.10}$$

11.5.1. Terminology

Some definitions are required before describing how macrotasking can be implemented in LAURA. Recall that a computational block is a rectangularly ordered array of cells defining all or part of the solution domain. A partition truncates a block in the computational sweep direction. For example, the recommended sweep direction starts at the body surface and moves across the boundary layer. In the default mode, this extends from $k = 1$ to $k = kblk$, where $kblk$ is the total number of cells in the k-direction of the active block (chapter 7). In this scenario, a k-partition is defined by specifying limits on the starting and stopping location of the sweep in the k-direction. An i-partition and a j-partition are defined similarly, by specifying limits on the starting and stopping location of the sweep in the i- and j-directions, respectively. Partitions can overlap in any computational block. Transverse sweep directions also can be accommodated.

> **NOTE:** A maximum of 100 partitions are allowed in LAURA.

One or more partitions are assigned to a task. It is convenient to think of a task as being assigned to a particular CPU. In practice, tasks get rolled in and out of execution in a multiuser environment, and there is no guarantee that the same CPU will always process the same task.

> **NOTE:** A maximum of 16 tasks can be accommodated in LAURA; however, it is counterproductive to assign more tasks than available CPU's on the computer.

11.5.2. Implementation

At present, macrotasking options exist only for the CRAY computers, and they can be implemented in two ways. In the default mode, the code assigns one k-partition per computational block, and each partition is assigned to its own task. As a result, a single block job will not macrotask in the default mode.

The user can overwrite the defaults by creating a file called `assign_tasks` in the executable directory. Each line of file `assign_tasks` defines a partition through the following five flags:

nbk	defines the computational block containing the partition (1 = `Block A`, 2 = `B`, 3 = `C`, 4 = `D`, 5 = `E`, and 6 = `G`)
mbk	defines the sweep direction and partition type (1 = i-, 2 = j-, and 3 = k-sweep)
lstrt	defines the starting index of the sweep ($1 \leq lstrt \leq lstop$)
lstop	defines the stopping index of the sweep ($lstrt \leq lstop \leq LMAX$)
mapcpu	defines the processor (by number) to which the partition is assigned ($1 \leq mapcpu \leq maxcpu$)

11.5.3. Load Balancing

The user can assign partition sizes and tasks in such a way as to concentrate CPU cycles in regions that are slow to converge. An option for dynamic load balancing exists for the case of a single block divided into k-partitions, with each partition assigned to its own task. The option can be switched on by setting $mtaska = 1$ (default: $mtaska = 0$) in the file `mtaska_assn.strt` after running `PRELUDE`. The executable will need to be recompiled by running `make`. The algorithm will dynamically change partition boundaries by comparing the error norms of adjacent partitions, decreasing the size of partitions with large error norms, and increasing the size of partitions with small error norms. While some very encouraging results have been obtained in test cases (ref. 18), this option is considered to be experimental and is only recommended for use by researchers interested in this specific topic.

Experience in a multiuser environment has shown that multiple tasks do not necessarily get equal access to CPU time. Typically, tasks one through four of an eight-CPU machine get nearly equal access, while subsequent tasks appear to be penalized, especially during peak load periods. It is believed this discrepancy derives from the manner in which the system is tuned to promote fair share access to all users. When this unequal access occurs, load balancing is disrupted, and CPU cycles can be wasted. Consequently, requests for more than four tasks are not recommended for small jobs run during peak activity periods. However, special multitasking queues can be created for very large jobs that run at off-peak times in a nearly dedicated, high-priority mode.

NOTE: The user should consult the system administrator for information on such queues.

In this mode, our experience shows that all tasks get equal access, and we routinely obtain average concurrent CPU use levels of greater than 7 on an 8-processor CRAY YMP and have achieved values greater than 14 (15.3, on one occasion in a multitasking queue run at night with minimal contention from other users) on a 16-processor CRAY C-90.

NOTE: Multitasking should not be run when the option for grid restructuring is active. Grid restructuring requires synchronization of tasks, which is not provided for in LAURA.

NOTE: Setup of the executable using **PRELUDE** assumes that only k-directional sweeps are required in setting up dimension statements. Defining i- and j-directional sweeps can result in an error message from `laura`, followed by an abort of the job request, if the required array dimensioning exceeds the prescribed value for *isjs* (section 11.3).

11.6. Radiative Transport

The treatment of radiation phenomena in a high-temperature flow requires a level of effort that is a significant fraction of the total effort required to express and solve the fluid equations. It is therefore often omitted in CFD applications. Under certain conditions, however, the effect of radiative heating on the flow and the heating of the vehicle surface is not negligible. The **LORAN** (Langley Optimized Radiative Nonequilibrium) algorithm (ref. 29) has been developed to allow LAURA to be applied to this class of problems. In cases in which radiative effects on the flow field are minor, **LORAN** can be used to post-process a LAURA solution. The details of this application are given in reference 30. For cases in which radiation significantly alters the flow field properties, **LORAN** may be iteratively coupled to the LAURA algorithm (ref. 31).

NASA Langley Research Center
Hampton, VA 23681-0001
October 10, 1995

Appendix A

Sample Case

An example case is included here to illustrate an application of the LAURA algorithm. This simple case is the axisymmetric flow over a sphere with a radius of 1 m. The thin-layer Navier-Stokes equations are solved for laminar, perfect gas flow. The free-stream conditions are $\mathcal{V}_\infty = 5000$ m/s, $\rho_\infty = 0.001$ kg/m^3, and $T_\infty = 200$ K. A constant wall temperature of $T_w = 500$ K is specified. The grid dimensions are 30 cells in the streamwise direction and 64 in the body-normal direction.

Files from the initial run for this sample case serve as examples throughout the manual. Files from the second run for this sample case are contained in this appendix. The results presented for this example can be replicated by creating a working directory and repeating the procedure described below.

Typing the command

```
PRELUDE
```

and accepting the default values for each prompt will yield the initialization of `RESTART.in` used here, as well as the file `data` shown in section 9.1.3. The file `INPUTS` is created by `PRELUDE` and is a record of the responses to the prompts of `stArt` during the most recent `PRELUDE` session in this `LOCAL` directory. Its contents for this sample case are shown in section 6.3. For subsequent `PRELUDE` sessions, typing the command

```
PRELUDE INPUTS
```

yields the initialization of `RESTART.in` used here, as well as the file `data` shown in section 9.1.3.

> **NOTE:** Since a constant wall temperature (*tempbc* = 0) is specified, the files `TWALL.in` and `transition` are not required. Further, the file `transition` is not required for laminar flow.

The results presented herein are from a SUN Sparcstation with a single processor. No `assign_tasks` file was incorporated. Since it is a simple case, no `LOCAL` or `CUSTOM` files were required. As mentioned above, the input and output files for the initial run are interspersed throughout this manual, and are not repeated here.

For the second run, `PRELUDE` does not need to be repeated. The adjustments are accomplished through a direct modification to file `data`. The resultant `data` file used for the second run is shown below (screen 50):

```
                          VERSION=LAURA.4.1
 2           nord ..................... 1(st-) or 2(nd-) order spatial accuracy
20           ntrnsprt ............ iterations between transport property updates
20           njcobian ..................... iterations between jacobian updates

 0  0  1     {i,j,k}vis . 0=off/1=on for {i,j,k} TL N-S viscous terms in block 1

.50000E+04   vinfb ................................. freestream velocity [m/s]
.10000E-02   rinfb ................................ freestream density [kg/m^3]
.20000E+03   tinf .................................. freestream temperature [K]

 0           tempbc ....... {0=constant, 1=variable, 2=radiative equilibrium} Tw
.50000E+03   twall ........................ if tempbc=0:  wall temperature [K]
0.000        ept ..... if tempbc=2:  temperature relaxation factor (0 < ept < 1)

1.0000       rflngth ..... conversion:  grid units ==> meters (1 m  =  1.0000 m)
0.000000E+00 zcg ....................................... axial cg location [m ]
0.000000E+00 xcg ..................................... vertical cg location [m ]
0.314159E+01 refarea ............................ reference area of body [m ^2]
0.200000E+01 reflen ............................. reference length of body [m ]

 1500        iterg ............................. maximum iterations for this run
  200        movegrd ............................ frequency of grid adjustments
    4        maxmoves .......... maximum number of grid adjustments (0=no limit)
    1        iabseig ............................... {0=normal, 1=scaled} limiter
0.300        epsa ........................................... eigenvalue limiter
0.010        errd ............................. error criteria for grid doubling
10.00        hrs ................................ time limit for this run [hr]

1.60         rfinv ................... inviscid relaxation factor, (rfinv > 1.5)
 .60         rfvis ................... viscous  relaxation factor, (rfvis > 0.5)
```

Screen 50.

A.1. Screen Output

The constituent elements of the standard output are discussed in section 10.1. That section also contains the standard output from the initial run of this sample case. The standard output for the second run (excluding the preamble) is shown on the pages which follow, beginning with the following iteration record (screen 51):

```
tsk blk ms iter   L2 norm  tsk norm  inf norm   i   j   k   m   time  strt stp

                          CPU number 01 starting

 1   1  3    20  1.737E-01 1.737E-01 4.920E-02  22   1  15   1    9.90   1  16
 1   1  3    40  6.422E-02 6.422E-02 2.914E-02  30   1  15   1   19.31   1  16
 1   1  3    60  1.680E-02 1.680E-02 1.626E-02  29   1   6   1   28.87   1  16
 1   1  3    80  7.106E-03 7.106E-03 1.473E-02  29   1   6   1   38.24   1  16

    ...Grid doubled after iter =   80

 1   1  3   100  3.304E-02 3.304E-02 1.841E-02   2   1   1   1   57.29   1  32
 1   1  3   120  2.697E-02 2.697E-02 1.922E-02  18   1   1   1   76.15   1  32
 1   1  3   140  2.167E-02 2.167E-02 2.068E-02  27   1   1   1   95.02   1  32
 1   1  3   160  9.398E-03 9.398E-03 1.478E-02  28   1   1   1  113.90   1  32

    ...Grid doubled after iter =  160

 1   1  3   180  3.031E-02 3.031E-02 1.641E-02  30   1  64   1  151.98   1  64
 1   1  3   200  3.173E-02 3.173E-02 1.440E-02   5   1   1   1  189.79   1  64

    ...Grid adjusted after iter =  200

       WARNING:  Recell_w criterion not satisfied in "algnshk" for a total
                 of   30 stations (see files "algnshk.out" & "grid.out").

 1   1  3   220  1.087E-01 1.087E-01 5.193E-02  30   1  57   1  228.12   1  64
 1   1  3   240  5.979E-02 5.979E-02 4.544E-02  30   1  58   1  265.90   1  64
 1   1  3   260  3.217E-02 3.217E-02 3.932E-02  30   1  59   1  303.71   1  64
 1   1  3   280  1.342E-02 1.342E-02 1.963E-02  30   1  59   1  341.52   1  64
 1   1  3   300  7.762E-03 7.762E-03 1.314E-02  30   1  60   1  379.33   1  64
 1   1  3   320  5.779E-03 5.779E-03 9.356E-03  21   1  56   1  417.29   1  64
 1   1  3   340  3.557E-03 3.557E-03 7.935E-03  27   1  59   1  455.12   1  64
 1   1  3   360  1.342E-03 1.342E-03 4.642E-03  30   1  61   1  492.97   1  64
 1   1  3   380  6.612E-04 6.612E-04 2.131E-03  28   1  55   1  530.79   1  64
 1   1  3   400  5.078E-04 5.078E-04 2.195E-03  24   1  22   1  568.64   1  64

    ...Grid adjusted after iter =  400

       WARNING:  Recell_w criterion not satisfied in "algnshk" for a total
                 of   30 stations (see files "algnshk.out" & "grid.out").
```

Screen 51.

NOTE: The frequency of output for the iteration record is controlled by the parameter *njcobian*. In other words, output is produced only for those iterations in which the Jacobian is updated. For the initial run, the update occurred for each iteration. For the second run, however, the solution advances 20 iterations (screen 52) between updates (as specified in file **data**).

```
1  1  3   420  6.391E-03 6.391E-03 1.384E-02  23   1  55  1   607.02   1  64
1  1  3   440  3.742E-03 3.742E-03 1.216E-02  25   1  55  1   644.82   1  64
1  1  3   460  2.631E-03 2.631E-03 1.041E-02  27   1  55  1   682.61   1  64
1  1  3   480  1.252E-03 1.252E-03 5.933E-03  30   1  55  1   720.36   1  64
1  1  3   500  4.885E-04 4.885E-04 2.682E-03  30   1  21  1   758.14   1  64
1  1  3   520  3.697E-04 3.697E-04 3.167E-03  25   1  56  1   795.94   1  64
1  1  3   540  3.619E-04 3.619E-04 4.036E-03  26   1  55  1   833.75   1  64
1  1  3   560  4.726E-04 4.726E-04 4.521E-03  28   1  55  1   871.52   1  64
1  1  3   580  4.010E-04 4.010E-04 4.339E-03  28   1  55  1   909.32   1  64
1  1  3   600  1.697E-04 1.697E-04 3.490E-03  30   1  55  1   947.08   1  64

   ...Grid adjusted after iter =  600

      WARNING:  Recell_w criterion not satisfied in "algnshk" for a total
                of   30 stations (see files "algnshk.out" & "grid.out").

1  1  3   620  5.297E-03 5.297E-03 1.710E-02  27   1  55  1   985.40   1  64
1  1  3   640  2.632E-03 2.632E-03 1.246E-02  29   1  55  1  1023.18   1  64
1  1  3   660  9.894E-04 9.894E-04 6.683E-03  30   1  55  1  1060.89   1  64
1  1  3   680  4.246E-04 4.246E-04 3.904E-03  30   1  54  1  1098.69   1  64
1  1  3   700  2.298E-04 2.298E-04 2.218E-03  21   1  55  1  1136.46   1  64
1  1  3   720  1.804E-04 1.804E-04 1.619E-03  21   1  55  1  1174.19   1  64
1  1  3   740  1.579E-04 1.579E-04 2.185E-03  18   1  55  1  1211.99   1  64
1  1  3   760  1.261E-04 1.261E-04 2.283E-03  19   1  55  1  1249.73   1  64
1  1  3   780  1.067E-04 1.067E-04 2.039E-03  21   1  55  1  1287.47   1  64
1  1  3   800  1.097E-04 1.097E-04 2.090E-03  22   1  55  1  1325.25   1  64

   ...Grid adjusted after iter =  800

      WARNING:  Recell_w criterion not satisfied in "algnshk" for a total
                of   30 stations (see files "algnshk.out" & "grid.out").

   ...Turning off algnshk after  4 adjustments

1  1  3   820  5.234E-03 5.234E-03 2.106E-02  24   1  55  1  1363.52   1  64
1  1  3   840  3.289E-03 3.289E-03 1.342E-02  26   1  55  1  1401.31   1  64
1  1  3   860  2.175E-03 2.175E-03 1.147E-02  28   1  55  1  1439.11   1  64
1  1  3   880  9.456E-04 9.456E-04 8.215E-03  30   1  55  1  1476.84   1  64
1  1  3   900  3.577E-04 3.577E-04 3.293E-03  30   1  55  1  1514.61   1  64
1  1  3   920  2.364E-04 2.364E-04 1.992E-03  25   1  55  1  1552.35   1  64
1  1  3   940  2.007E-04 2.007E-04 2.386E-03  27   1  55  1  1590.09   1  64
1  1  3   960  1.364E-04 1.364E-04 1.924E-03  20   1  56  1  1627.85   1  64
1  1  3   980  1.038E-04 1.038E-04 1.916E-03  18   1  55  1  1665.68   1  64
1  1  3  1000  1.142E-04 1.142E-04 2.600E-03  21   1  56  1  1703.47   1  64
```

Screen 52.

```
 1   1   3   1020  8.953E-05 8.953E-05 1.570E-03  25   1  55  1  1741.34   1  64
 1   1   3   1040  6.536E-05 6.536E-05 1.556E-03  27   1  55  1  1779.16   1  64
 1   1   3   1060  3.570E-05 3.570E-05 1.027E-03  28   1  55  1  1816.92   1  64
 1   1   3   1080  2.335E-05 2.335E-05 1.075E-03  18   1  56  1  1854.69   1  64
 1   1   3   1100  1.438E-05 1.438E-05 5.557E-04  28   1  55  1  1892.42   1  64
 1   1   3   1120  1.146E-05 1.146E-05 5.525E-04  28   1  55  1  1930.19   1  64
 1   1   3   1140  7.489E-06 7.489E-06 4.855E-04  30   1  55  1  1967.91   1  64
 1   1   3   1160  4.472E-06 4.472E-06 2.975E-04  30   1  55  1  2005.71   1  64
 1   1   3   1180  2.943E-06 2.943E-06 2.075E-04  18   1  55  1  2043.49   1  64
 1   1   3   1200  2.687E-06 2.687E-06 1.876E-04  18   1  55  1  2081.23   1  64
 1   1   3   1220  2.376E-06 2.376E-06 1.661E-04  18   1  55  1  2118.99   1  64
 1   1   3   1240  2.218E-06 2.218E-06 1.556E-04  22   1  55  1  2156.81   1  64
 1   1   3   1260  2.234E-06 2.234E-06 1.579E-04  23   1  55  1  2194.51   1  64
 1   1   3   1280  2.121E-06 2.121E-06 1.525E-04  26   1  55  1  2232.27   1  64
 1   1   3   1300  1.961E-06 1.961E-06 1.331E-04  28   1  55  1  2270.04   1  64
 1   1   3   1320  1.818E-06 1.818E-06 1.193E-04  28   1  55  1  2307.80   1  64
 1   1   3   1340  1.731E-06 1.731E-06 1.177E-04  15   1  55  1  2345.58   1  64
 1   1   3   1360  1.669E-06 1.669E-06 1.165E-04  15   1  55  1  2383.34   1  64
 1   1   3   1380  1.605E-06 1.605E-06 1.143E-04  15   1  55  1  2421.25   1  64
 1   1   3   1400  1.569E-06 1.569E-06 1.159E-04  21   1  56  1  2459.11   1  64
 1   1   3   1420  1.725E-06 1.725E-06 3.111E-04  21   1  56  1  2496.90   1  64
 1   1   3   1440  2.098E-06 2.098E-06 5.100E-04  21   1  56  1  2534.65   1  64
 1   1   3   1460  3.145E-06 3.145E-06 7.075E-04  21   1  56  4  2572.44   1  64
 1   1   3   1480  5.455E-06 5.455E-06 1.044E-03  21   1  56  4  2610.22   1  64
 1   1   3   1500  6.254E-06 6.254E-06 1.258E-03  21   1  56  4  2648.03   1  64

      CPU  1 terminated at  2648.08 seconds (after  1500 iterations).
```

Screen 52. Concluded.

The integrated surface quantities for this second run are as follows (screen 53):

```
                    cx =  0.99331755E-02
                    cy =  0.00000000E+00
                    cz =  0.12230405E-01
                    gy = -0.50450885E-02
               summdot = -0.67376419E-07
               sumheat = -0.27692626E-03
```

Screen 53.

A.2. File algnshk.out

The constituent elements of file **algnshk.out** are discussed in section 10.2. That section also shows the contents of this file from the initial run of this sample case. Its contents are shown for the second run on the pages that follow (screen 54):

```
...Grid adjusted after iter =  200

                                 norm.     wall      max.      location of
 i     j       hmin1             dist.   stretch   stretch      maximum

 1     1   .2290770E-05  .300507  1.045911  1.468397  .001765 (17)
 2     1   .2246804E-05  .298642  1.045998  1.469286  .001751 (17)
 3     1   .2228229E-05  .297847  1.046035  1.469667  .001746 (17)
 4     1   .2231236E-05  .297976  1.046029  1.469605  .001747 (17)
 5     1   .2254972E-05  .298990  1.045982  1.469120  .001754 (17)
 6     1   .2317622E-05  .301634  1.045859  1.467862  .001773 (17)
 7     1   .2418150E-05  .305778  1.045668  1.465916  .001803 (17)
 8     1   .2550057E-05  .311046  1.045429  1.463485  .001841 (17)
 9     1   .2711846E-05  .317265  1.045154  1.460674  .001886 (17)
10     1   .2894426E-05  .323996  1.044863  1.457703  .001934 (17)
11     1   .3057070E-05  .329760  1.044619  1.455215  .001977 (17)
12     1   .3195636E-05  .334511  1.044421  1.453200  .002011 (17)
13     1   .3383573E-05  .340744  1.044167  1.450607  .002057 (17)
14     1   .3653929E-05  .349320  1.043826  1.447127  .002120 (17)
15     1   .3977254E-05  .359043  1.043451  1.443297  .002192 (17)
16     1   .4367052E-05  .370094  1.043038  1.439087  .002275 (17)
17     1   .4820189E-05  .382152  1.042603  1.434654  .002365 (17)
18     1   .5307045E-05  .394301  1.042181  1.430346  .002457 (17)
19     1   .5822294E-05  .406384  1.041776  1.426211  .002549 (17)
20     1   .6406782E-05  .419266  1.041359  1.421953  .002648 (17)
21     1   .7086295E-05  .433308  1.040920  1.417481  .002756 (17)
22     1   .7877459E-05  .448583  1.040462  1.412800  .002874 (17)
23     1   .8809709E-05  .465340  1.039978  1.407871  .003005 (17)
24     1   .9902239E-05  .483560  1.039475  1.402737  .003148 (17)
25     1   .1117809E-04  .503242  1.038955  1.397434  .003304 (17)
26     1   .1265588E-04  .524281  1.038425  1.392022  .003472 (17)
27     1   .1440414E-04  .547192  1.037874  1.386405  .003657 (17)
28     1   .1664086E-04  .574000  1.037263  1.380165  .003874 (17)
29     1   .1975544E-04  .607668  1.036539  1.372785  .004151 (17)
30     1   .2242953E-04  .633914  1.036007  1.367350  .004368 (17)
```

Screen 54.

...Grid adjusted after iter = 400

i	j	hmin1	norm. dist.	wall stretch	max. stretch	location of maximum
1	1	.2416946E-05	.305729	1.045670	1.465938	.001802 (17)
2	1	.2408198E-05	.305373	1.045686	1.466105	.001800 (17)
3	1	.2430496E-05	.306279	1.045645	1.465682	.001806 (17)
4	1	.2472090E-05	.307954	1.045569	1.464905	.001818 (17)
5	1	.2509823E-05	.309459	1.045501	1.464212	.001829 (17)
6	1	.2553366E-05	.311175	1.045424	1.463425	.001841 (17)
7	1	.2607570E-05	.313285	1.045329	1.462465	.001857 (17)
8	1	.2660392E-05	.315314	1.045240	1.461549	.001871 (17)
9	1	.2747521E-05	.318604	1.045095	1.460078	.001895 (17)
10	1	.2874395E-05	.323272	1.044894	1.458019	.001929 (17)
11	1	.3007800E-05	.328036	1.044691	1.455954	.001964 (17)
12	1	.3136876E-05	.332513	1.044504	1.454043	.001997 (17)
13	1	.3253230E-05	.336447	1.044342	1.452389	.002026 (17)
14	1	.3417432E-05	.341842	1.044123	1.450156	.002065 (17)
15	1	.3615713E-05	.348134	1.043873	1.447602	.002112 (17)
16	1	.3819755E-05	.354374	1.043630	1.445121	.002158 (17)
17	1	.4083910E-05	.362135	1.043334	1.442104	.002216 (17)
18	1	.4412589E-05	.371342	1.042992	1.438620	.002284 (17)
19	1	.4749112E-05	.380311	1.042669	1.435320	.002352 (17)
20	1	.5142228E-05	.390274	1.042320	1.431757	.002427 (17)
21	1	.5639128E-05	.402172	1.041916	1.427636	.002517 (17)
22	1	.6257911E-05	.416061	1.041461	1.422998	.002623 (17)
23	1	.6992771E-05	.431429	1.040978	1.418069	.002741 (17)
24	1	.7835977E-05	.447807	1.040484	1.413033	.002868 (17)
25	1	.8794738E-05	.465080	1.039986	1.407946	.003003 (17)
26	1	.9871012E-05	.483058	1.039489	1.402875	.003144 (17)
27	1	.1110448E-04	.502147	1.038984	1.397723	.003295 (17)
28	1	.1262878E-04	.523910	1.038434	1.392115	.003469 (17)
29	1	.1465093E-04	.550279	1.037802	1.385669	.003682 (17)
30	1	.1654355E-04	.572884	1.037287	1.380418	.003865 (17)

Screen 54. Continued.

...Grid adjusted after iter = 600

i	j	hmin1	norm. dist.	wall stretch	max. stretch	location of maximum
1	1	.2371600E-05	.303874	1.045755	1.466806	.001789 (17)
2	1	.2382643E-05	.304327	1.045734	1.466593	.001792 (17)
3	1	.2399836E-05	.305032	1.045702	1.466264	.001797 (17)
4	1	.2422148E-05	.305940	1.045660	1.465840	.001804 (17)
5	1	.2455893E-05	.307305	1.045598	1.465206	.001814 (17)
6	1	.2499914E-05	.309064	1.045519	1.464393	.001826 (17)
7	1	.2560883E-05	.311470	1.045410	1.463291	.001844 (17)
8	1	.2623653E-05	.313906	1.045302	1.462184	.001861 (17)
9	1	.2697156E-05	.316711	1.045178	1.460922	.001882 (17)
10	1	.2798099E-05	.320481	1.045014	1.459245	.001909 (17)
11	1	.2914756E-05	.324729	1.044832	1.457384	.001940 (17)
12	1	.3043477E-05	.329286	1.044639	1.455417	.001973 (17)
13	1	.3176427E-05	.333861	1.044448	1.453474	.002007 (17)
14	1	.3331251E-05	.339032	1.044237	1.451314	.002045 (17)
15	1	.3516904E-05	.345028	1.043996	1.448856	.002089 (17)
16	1	.3728334E-05	.351607	1.043737	1.446215	.002137 (17)
17	1	.3966779E-05	.358736	1.043462	1.443416	.002190 (17)
18	1	.4261349E-05	.367164	1.043146	1.440189	.002253 (17)
19	1	.4593823E-05	.376227	1.042815	1.436812	.002321 (17)
20	1	.4930324E-05	.384968	1.042504	1.433641	.002387 (17)
21	1	.5328680E-05	.394824	1.042163	1.430164	.002461 (17)
22	1	.5795001E-05	.405761	1.041796	1.426420	.002544 (17)
23	1	.6345261E-05	.417948	1.041401	1.422382	.002637 (17)
24	1	.6983392E-05	.431239	1.040984	1.418129	.002740 (17)
25	1	.7737298E-05	.445953	1.040539	1.413593	.002854 (17)
26	1	.8585244E-05	.461415	1.040090	1.409007	.002974 (17)
27	1	.9535036E-05	.477589	1.039638	1.404394	.003101 (17)
28	1	.1070106E-04	.496063	1.039142	1.399340	.003247 (17)
29	1	.1226339E-04	.518857	1.038559	1.393393	.003429 (17)
30	1	.1372809E-04	.538561	1.038079	1.388489	.003587 (17)

Screen 54. Continued.

...Grid adjusted after iter = 800

i	j	hmin1	norm. dist.	wall stretch	max. stretch	location of maximum
1	1	.2375307E-05	.304026	1.045748	1.466735	.001790 (17)
2	1	.2388167E-05	.304554	1.045724	1.466487	.001794 (17)
3	1	.2411492E-05	.305507	1.045680	1.466042	.001801 (17)
4	1	.2436916E-05	.306539	1.045633	1.465562	.001808 (17)
5	1	.2470399E-05	.307886	1.045572	1.464937	.001818 (17)
6	1	.2515616E-05	.309688	1.045490	1.464107	.001831 (17)
7	1	.2574941E-05	.312019	1.045386	1.463040	.001848 (17)
8	1	.2639533E-05	.314516	1.045275	1.461908	.001866 (17)
9	1	.2708183E-05	.317128	1.045160	1.460736	.001885 (17)
10	1	.2781182E-05	.319857	1.045041	1.459522	.001904 (17)
11	1	.2871940E-05	.323184	1.044898	1.458058	.001929 (17)
12	1	.3009501E-05	.328095	1.044689	1.455928	.001964 (17)
13	1	.3178134E-05	.333919	1.044446	1.453449	.002007 (17)
14	1	.3350795E-05	.339674	1.044210	1.451048	.002049 (17)
15	1	.3535329E-05	.345611	1.043972	1.448620	.002093 (17)
16	1	.3749227E-05	.352243	1.043712	1.445963	.002142 (17)
17	1	.3987594E-05	.359345	1.043439	1.443180	.002195 (17)
18	1	.4253068E-05	.366932	1.043155	1.440277	.002251 (17)
19	1	.4578935E-05	.375830	1.042829	1.436957	.002318 (17)
20	1	.4962142E-05	.385774	1.042476	1.433353	.002393 (17)
21	1	.5374729E-05	.395931	1.042126	1.429780	.002470 (17)
22	1	.5814821E-05	.406213	1.041782	1.426268	.002548 (17)
23	1	.6326546E-05	.417545	1.041414	1.422513	.002634 (17)
24	1	.6914565E-05	.429846	1.041027	1.418568	.002729 (17)
25	1	.7596157E-05	.443273	1.040619	1.414407	.002833 (17)
26	1	.8388739E-05	.457924	1.040190	1.410027	.002947 (17)
27	1	.9304709E-05	.473769	1.039743	1.405468	.003071 (17)
28	1	.1044293E-04	.492094	1.039247	1.400408	.003216 (17)
29	1	.1198605E-04	.514955	1.038657	1.394390	.003397 (17)
30	1	.1344289E-04	.534835	1.038168	1.389400	.003557 (17)

Screen 54. Concluded.

A.3. File conv.out

The constituent elements of file conv.out are discussed in section 10.3. That section also shows the contents of this file from the initial run of this sample case. Its contents are shown for the second run (screen 55):

				body pressure		body heating	
tsk	iter	residual	time (sec)	(stag)	(end)	(stag)	(end)
1	20	.173680E+00	9.480	1.02787	.362811E-01	-.207159E-01	-.181072E-02
1	40	.642220E-01	18.890	.930974	.343741E-01	-.168117E-01	-.148686E-02
1	60	.167974E-01	28.450	.925715	.352460E-01	-.152770E-01	-.127062E-02
1	80	.710615E-02	37.820	.923494	.342261E-01	-.146595E-01	-.101474E-02
1	100	.330401E-01	56.870	.958593	.360259E-01	-.148522E-01	-.902765E-03
1	120	.269717E-01	75.730	.891812	.337655E-01	-.121135E-01	-.697635E-03
1	140	.216679E-01	94.600	.928096	.344724E-01	-.125548E-01	-.627335E-03
1	160	.939811E-02	113.480	.923566	.375377E-01	-.122950E-01	-.649356E-03
1	180	.303121E-01	151.560	.924323	.350256E-01	-.118011E-01	-.546651E-03
1	200	.317269E-01	189.370	.989561	.340480E-01	-.135799E-01	-.501591E-03
1	220	.108663E+00	227.700	.932868	.344543E-01	-.978973E-02	-.453082E-03
1	240	.597945E-01	265.480	.900657	.353153E-01	-.985254E-02	-.466547E-03
1	260	.321750E-01	303.290	.921168	.356246E-01	-.996108E-02	-.477142E-03
1	280	.134248E-01	341.100	.911631	.397506E-01	-.968400E-02	-.554359E-03
1	300	.776205E-02	378.910	.922587	.356448E-01	-.978836E-02	-.455140E-03
1	320	.577924E-02	416.870	.924712	.359903E-01	-.970618E-02	-.472353E-03
1	340	.355717E-02	454.700	.921260	.374134E-01	-.956439E-02	-.499765E-03
1	360	.134166E-02	492.550	.922012	.378405E-01	-.950948E-02	-.503842E-03
1	380	.661229E-03	530.370	.922229	.378553E-01	-.944654E-02	-.500433E-03
1	400	.507751E-03	568.220	.922666	.377150E-01	-.939521E-02	-.495071E-03
1	420	.639128E-02	606.600	.922256	.377506E-01	-.880919E-02	-.508537E-03
1	440	.374177E-02	644.400	.932637	.386944E-01	-.904346E-02	-.523425E-03
1	460	.263072E-02	682.190	.921975	.386469E-01	-.867947E-02	-.513020E-03
1	480	.125234E-02	719.940	.912888	.380557E-01	-.857411E-02	-.495906E-03
1	500	.488538E-03	757.720	.918442	.387008E-01	-.863178E-02	-.512426E-03
1	520	.369721E-03	795.520	.922170	.385519E-01	-.866181E-02	-.499670E-03
1	540	.361910E-03	833.330	.924905	.380903E-01	-.866811E-02	-.488361E-03
1	560	.472611E-03	871.100	.922862	.380120E-01	-.859700E-02	-.485701E-03
1	580	.401026E-03	908.900	.920210	.380712E-01	-.852818E-02	-.485206E-03
1	600	.169720E-03	946.660	.919402	.380940E-01	-.849003E-02	-.484321E-03
1	620	.529670E-02	984.980	.919623	.380117E-01	-.859598E-02	-.534466E-03
1	640	.263227E-02	1022.760	.920270	.382168E-01	-.856932E-02	-.537372E-03
1	660	.989424E-03	1060.470	.915966	.384135E-01	-.847342E-02	-.538888E-03
1	680	.424563E-03	1098.270	.919373	.383122E-01	-.851422E-02	-.533952E-03
1	700	.229767E-03	1136.040	.920236	.380983E-01	-.849428E-02	-.528916E-03
1	720	.180376E-03	1173.770	.919915	.383076E-01	-.846965E-02	-.534570E-03
1	740	.157903E-03	1211.570	.919846	.383422E-01	-.844631E-02	-.534464E-03
1	760	.126059E-03	1249.310	.919509	.382220E-01	-.842193E-02	-.531712E-03
1	780	.106714E-03	1287.050	.919529	.381634E-01	-.840353E-02	-.531835E-03
1	800	.109687E-03	1324.830	.919393	.383610E-01	-.838295E-02	-.537666E-03

Screen 55.

```
1  820 .523377E-02  1363.100 .919520     .382898E-01 -.837367E-02 -.547506E-03
1  840 .328934E-02  1400.890 .923097     .380419E-01 -.843978E-02 -.542679E-03
1  860 .217466E-02  1438.690 .928951     .380200E-01 -.849208E-02 -.543875E-03
1  880 .945589E-03  1476.420 .920814     .379861E-01 -.833722E-02 -.544010E-03
1  900 .357652E-03  1514.190 .919130     .381321E-01 -.830372E-02 -.549716E-03
1  920 .236392E-03  1551.930 .919473     .381633E-01 -.829368E-02 -.550581E-03
1  940 .200728E-03  1589.670 .920706     .381185E-01 -.829438E-02 -.551737E-03
1  960 .136390E-03  1627.430 .920990     .382616E-01 -.827712E-02 -.556893E-03
1  980 .103828E-03  1665.260 .920645     .382532E-01 -.825276E-02 -.557438E-03
1 1000 .114162E-03  1703.050 .920348     .381129E-01 -.823038E-02 -.555300E-03
1 1020 .895274E-04  1740.920 .920323     .380537E-01 -.821366E-02 -.555913E-03
1 1040 .653588E-04  1778.740 .920534     .382821E-01 -.820079E-02 -.563803E-03
1 1060 .357031E-04  1816.500 .920714     .384910E-01 -.818702E-02 -.569678E-03
1 1080 .233482E-04  1854.270 .920818     .384012E-01 -.817254E-02 -.568314E-03
1 1100 .143805E-04  1892.000 .920894     .382780E-01 -.815781E-02 -.566835E-03
1 1120 .114591E-04  1929.770 .920953     .382768E-01 -.814307E-02 -.568475E-03
1 1140 .748877E-05  1967.490 .921009     .383443E-01 -.812840E-02 -.571179E-03
1 1160 .447210E-05  2005.290 .921050     .383280E-01 -.811360E-02 -.571622E-03
1 1180 .294252E-05  2043.070 .921067     .382910E-01 -.809853E-02 -.571864E-03
1 1200 .268724E-05  2080.810 .921073     .383039E-01 -.808342E-02 -.573241E-03
1 1220 .237628E-05  2118.570 .921075     .382835E-01 -.806835E-02 -.573524E-03
1 1240 .221764E-05  2156.390 .921079     .382661E-01 -.805342E-02 -.574031E-03
1 1260 .223404E-05  2194.090 .921081     .382659E-01 -.803862E-02 -.574852E-03
1 1280 .212072E-05  2231.850 .921084     .382639E-01 -.802396E-02 -.575542E-03
1 1300 .196057E-05  2269.620 .921085     .382577E-01 -.800939E-02 -.576105E-03
1 1320 .181849E-05  2307.380 .921087     .382552E-01 -.799499E-02 -.576733E-03
1 1340 .173148E-05  2345.160 .921090     .382553E-01 -.798074E-02 -.577367E-03
1 1360 .166904E-05  2382.920 .921090     .382558E-01 -.796656E-02 -.577962E-03
1 1380 .160503E-05  2420.830 .921091     .382564E-01 -.795257E-02 -.578527E-03
1 1400 .156905E-05  2458.690 .921090     .382567E-01 -.793868E-02 -.579043E-03
1 1420 .172481E-05  2496.480 .921094     .382570E-01 -.792504E-02 -.579533E-03
1 1440 .209843E-05  2534.230 .921095     .382571E-01 -.791145E-02 -.579988E-03
1 1460 .314499E-05  2572.020 .921094     .382566E-01 -.789797E-02 -.580401E-03
1 1480 .545472E-05  2609.800 .921094     .382557E-01 -.788466E-02 -.580773E-03
1 1500 .625405E-05  2647.610 .921094     .382533E-01 -.787147E-02 -.581082E-03
```

Screen 55. Concluded.

A.4. File grid.out

The constituent elements of file grid.out are discussed in section 10.4. That section also shows the contents of this file from the initial run of this sample case. Its contents are shown here for the second run (screen 56):

Block 1, Surface 1
(64 cells normal to body)

i	j	dh_w	Recell_w	max. stretch
1	1	.132615E-05	3.307665	1.47 (18)
2	1	.134121E-05	3.311790	1.47 (18)
3	1	.136424E-05	3.313079	1.47 (18)
4	1	.139045E-05	3.308474	1.47 (18)
5	1	.142768E-05	3.307920	1.46 (18)
6	1	.146510E-05	3.282688	1.46 (18)
7	1	.151384E-05	3.253656	1.46 (18)
8	1	.158717E-05	3.245656	1.46 (18)
9	1	.165289E-05	3.193581	1.46 (18)
10	1	.172637E-05	3.135483	1.46 (18)
11	1	.182799E-05	3.102324	1.46 (17)
12	1	.198404E-05	3.116151	1.45 (18)
13	1	.211008E-05	3.034105	1.45 (18)
14	1	.234000E-05	3.055790	1.45 (18)
15	1	.248702E-05	2.927279	1.45 (18)
16	1	.278297E-05	2.934100	1.45 (17)
17	1	.309083E-05	2.895659	1.44 (18)
18	1	.337977E-05	2.790136	1.44 (17)
19	1	.385189E-05	2.781113	1.44 (18)
20	1	.423707E-05	2.657829	1.43 (18)
21	1	.479993E-05	2.596962	1.43 (18)
22	1	.543809E-05	2.517761	1.43 (18)
23	1	.616728E-05	2.427922	1.42 (18)
24	1	.708869E-05	2.356264	1.42 (18)
25	1	.814689E-05	2.268806	1.41 (18)
26	1	.939657E-05	2.180056	1.41 (18)
27	1	.109298E-04	2.098696	1.40 (18)
28	1	.129232E-04	2.023678	1.40 (18)
29	1	.152506E-04	1.912261	1.39 (18)
30	1	.177170E-04	1.835846	1.39 (18)

Screen 56.

Appendix B

Conic Geometry

This appendix provides the definitions for the conic geometry inputs ($newjob = 1$).

Nomenclature

b	axial shape parameter
B	cross-sectional shape parameter; ratio of principal radii of curvature
ϵ	axial eccentricity of nose
r, z, ϕ	cylindrical coordinates, in units specified via *iunit*
r_{axi}	radius for axisymmetric conic body
R_N	body nose radius, in units specified via *iunit*
R_{xz}	radius of curvature in symmetry plane, in units specified via *iunit*
R_{yz}	radius of curvature in side plane, in units specified via *iunit*
x, y, z	Cartesian coordinates, in units specified via *iunit*
θ	body half-angle for conic geometry, deg

Subscript:

junct	juncture between blunted forebody and cone afterbody

For many configurations of interest, the body shape can be defined by a conic equation. Consider an axisymmetric conic body rotated about the z-axis, whose nose is at ($z = 0, r = 0$):

$$r_{\text{axi}}^2 = r^2 = 2R_N z - bz^2 \tag{B.1}$$

The slope for this body is

$$\frac{dr}{dz} = \frac{R_N - bz}{r} \tag{B.2}$$

so that at the nose, the slope is infinite.

The character of the conic nose is dictated by the axial shape parameter (b) which is related to its eccentricity (ϵ). The relationship for $b \leq 1$ is

$$\epsilon = \sqrt{1 - b} \qquad \text{or} \qquad b = 1 - \epsilon^2$$

while for $b > 1$ it is

$$\epsilon = \sqrt{1 - \frac{1}{b}} \qquad \text{or} \qquad b = \frac{1}{1 - \epsilon^2}$$

Table B.1. Character of Axisymmetric Conic Geometry

Shape	b	ϵ
Hyperbola	< 0	> 1
Parabola	0	1
Ellipse	> 0	< 1
Circle	1	0

These results are summarized in table B.1, which shows that b has a unique value for a parabola (where $b = 0$) and a sphere ($b = 1$). The value of b for a hyperbola is related to the half-angle (θ) of its asymptote by the following equation:

$$b = -\tan^2 \theta \tag{B.3}$$

An ellipse can be used to define the forebody of a blunted cone. The afterbody can be described by the simple equation

$$r_{\text{axi}} = r = r_{\text{junct}} + (z - z_{\text{junct}}) \tan \theta \tag{B.4}$$

where r_{junct} and z_{junct} are the coordinates of the juncture between the ellipse and cone. The slope of the cone is

$$\frac{dr}{dz} = \tan \theta \tag{B.5}$$

At the juncture between the ellipsoidal forebody and cone afterbody, both the position and slope of the body are continuous. The coordinates of this juncture can be determined by equating equations (B.2) and (B.5), substituting equation (B.1) for r, and solving for z. The resultant axial location of this juncture is

$$z_{\text{junct}} = \frac{R_N}{b}\left(1 - \frac{\tan\theta}{\sqrt{b + \tan^2\theta}}\right) = \frac{R_N}{b}\left(1 - \frac{\sin\theta}{\sqrt{b\cos^2\theta + \sin^2\theta}}\right) \tag{B.6}$$

The corresponding value for r_{junct} can be found from equation (B.1):

$$r^2_{\text{junct}} = 2R_N z_{\text{junct}} - b z^2_{\text{junct}} \tag{B.7}$$

NOTE: Substituting a value of $b = 1$ into equation (B.1) yields

$$r^2_{\text{axi}} = r^2 = 2R_N z - z^2$$

which is an equation for a circle. For axisymmetric flow, this gives a spherical forebody, with the sphere-cone juncture located at

$$z_{\text{junct}} = R_N[1 - \sin\theta] \qquad r_{\text{junct}} = R_N \cos\theta$$

NOTE: The above discussion can be applied to two-dimensional conics by simply replacing the variable r with y, the subscript axi with 2-D, and the word "cone" with "wedge."

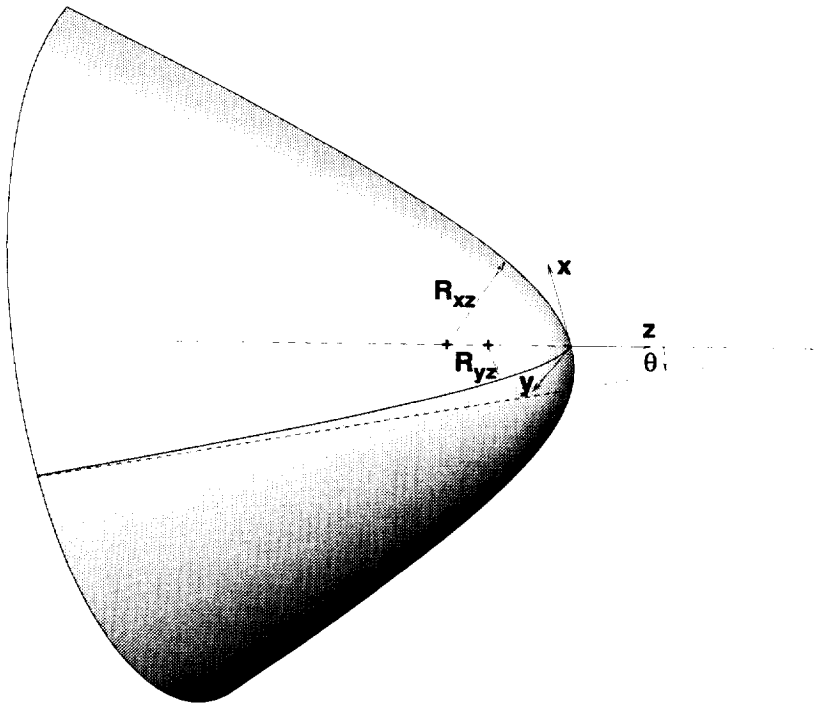


Figure B.1. Defining parameters for general conic geometry.

The above discussion can be extended to three-dimensional geometries (fig. B.1) with the following expression:

$$r_{3\text{-D}}^2 = r^2 = \frac{r_{\text{axi}}^2}{B\cos^2\phi + \sin^2\phi} \tag{B.8}$$

where $R_N = R_{yz}|_{z=0}$ in equation (B.1), and

$$\phi = \begin{cases} 0 & \text{(upper symmetry plane)} \\ \pi/2 & \text{(side plane)} \\ \pi & \text{(lower symmetry plane)} \end{cases}$$

The cross-sectional shape parameter is defined as

$$B = \left.\frac{R_{yz}}{R_{xz}}\right|_{z=0}$$

Values of $B > 0$ produce elliptical cross sections. The equation for an axisymmetric body is recovered by setting $B = 1$, which gives a circular cross section.

NOTE: For $b = B = 1$, equation (B.8) gives a sphere of radius R_N.

The cylindrical coordinates (r,ϕ) can be converted to Cartesian coordinates (x,y) using the following relations:

$$x = r\cos\phi \qquad y = r\sin\phi$$

Appendix C

Installation Procedure

This appendix covers the structure of `INSTALL_LAURA.4.1` and the structure of `mAch+prOc`.

C.1. Structure of `INSTALL_LAURA.4.1`

This script installs the LAURA code on this machine using the following procedures. Create a subdirectory for the LAURA source code (screen 57):

```
                                VERSION=LAURA.4.1

if [ -d $VERSION ]; then                # (if $VERSION exists already)...
        chmod 700 $VERSION
        chmod 700 $VERSION/*            2> /dev/null
        chmod 700 $VERSION/CUSTOM/*     2> /dev/null
        rm -r $VERSION
fi

mkdir $VERSION
```

Screen 57.

Extract the source code from `LAURA.4.1.tar.Z` (screen 58):

```
cd $VERSION

zcat ../LAURA.4.1.tar.Z | tar xf -
```

Screen 58.

Compile and run the C program `mAch+prOc` to determine the machine architecture and number of available processors (screen 59):

```
cc mach+proc.c -o mach+proc              # Determine architecture type
set `mach+proc` --                       # and number of processors.
```

Screen 59.

Compile the FORTRAN executables (ArrAy, flOwInIt, mAkEblk, sIzEIt, and stArt) as follows (screen 60):

```
if [ "$1" = "0" ]; then
        cf77 array.f -o ArrAy /dev/null
        cf77 flowinit.f -o flOwInIt /dev/null
        cf77 makeblk.f -o mAkEblk /dev/null
        cf77 sizeit.f -o sIzEIt /dev/null
        cf77 start.f -o stArt /dev/null
fi
if [ "$1" = "1" ]; then
        f77 array.f -o ArrAy 2> /dev/null
        f77 flowinit.f -o flOwInIt 2> /dev/null
        f77 makeblk.f -o mAkEblk 2> /dev/null
        f77 sizeit.f -o sIzEIt 2> /dev/null
        f77 start.f -o stArt 2> /dev/null
fi
if [ "$1" = "2" ]; then
        f77 array.f -o ArrAy 2> /dev/null
        f77 flowinit.f -o flOwInIt 2> /dev/null
        f77 makeblk.f -o mAkEblk 2> /dev/null
        f77 sizeit.f -o sIzEIt 2> /dev/null
        f77 start.f -o stArt 2> /dev/null
fi
if [ "$1" = "3" ]; then
        fc array.f -o ArrAy 2> /dev/null
        fc flowinit.f -o flOwInIt 2> /dev/null
        fc makeblk.f -o mAkEblk 2> /dev/null
        fc sizeit.f -o sIzEIt 2> /dev/null
        fc start.f -o stArt 2> /dev/null
        for FILE in *.F; do
                NOEXT=`basename $FILE .F`
                mv $FILE $NOEXT.FOR
        done
fi
```

Screen 60.

NOTE: For CONVEX architectures, the suffixes of the subroutine files are changed from .F to .FOR.

Tailor DEFAULTS to this machine (screen 61):

```
echo "  $1              machine ..................... 0=CRAY, 1=SUN, 2=SGI,
3=CONVEX" > temp
echo "  $2              nprocmx ..................... number of processors a
vailable" >> temp
cat DEFAULTS >> temp
mv temp DEFAULTS
```

Screen 61.

Remove write privileges for all files to discourage user alteration of the LAURA files in `$HOME/LAURA.4.1` (screen 62):

```
chmod 500 *
chmod 400 DEFAULTS TOP *.f mAch+prOc.c start* *.F* *.data *.inc

cd ..
chmod 500 $VERSION
```

Screen 62.

Add aliases for ARCHIVE, BLOX, CUSTOMIZE, INITIALIZE, KEEPER, LOCALIZE, PRELUDE, RESTORE, SIZEIT, and XCUSTOM to the user's `.cshrc` file (screen 63):

```
if [ ! "`grep PRELUDE .cshrc | grep $VERSION .cshrc`" ]; then
        echo ">> Do you wish to add aliases to .cshrc (y/n) {n}?"
        read ANSWER
        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                for FILE in ARCHIVE BLOX CUSTOMIZE INITIALIZE KEEPER LOCALIZE PR
ELUDE RESTORE SIZEIT XCUSTOM; do
                        echo "alias $FILE ~/$VERSION/$FILE" >> .cshrc
                done
        fi
fi
```

Screen 63.

NOTE: Type the command

`source .cshrc`

to activate the PRELUDE alias for this shell.

C.2. Structure of mAch+prOc

The C program **mAch+prOc** determines the architecture type, as well as available processors (*nprocmx*), for the host machine. This program is compiled and executed by **INSTALL_LAURA.4.1**. The information is used by **INSTALL_LAURA.4.1** to initialize the variables *machine* and *nprocmx* in **DEFAULTS**, which is later accessed by **stArt**. The operating system is also identified so that the various script files can be properly tailored to the host architecture.

> **NOTE:** On SGI systems, the existence of the R8000 CPU hardware is considered. If it exists, **start.f** is modified so that the appropriate flags are used in the compilation of **laura** via the **Makefile**.

The source code for **mAch+prOc.c** is as follows (screen 64):

```
#include <stdio.h>
#include <unistd.h>

#ifdef _CONVEX_SOURCE
#       include <sys/sysinfo.h>
#elif sun || sgi
#       include <sys/utsname.h>
#       include <string.h>
#       ifdef sgi
#               include <sys/sysmp.h>
#               include <invent.h>
#       endif
#endif

#ifdef sgi
        istfp()
        {
#               ifdef INV_IP21BOARD
                        inventory_t *entry;

                        while ((entry=getinvent())!=NULL) {
                                if (entry->inv_class==INV_PROCESSOR)
                                        if (entry->inv_type==INV_CPUBOARD)
                                                if (entry->inv_state==INV_IP21BO
ARD) {

                                                        endinvent();
                                                        return(1);
                                                }
                        }
                        return(0);
#               else
                        return(0);
#               endif

        }
#endif
```

Screen 64.

```
main()
{
        int machine=0, processors=0, ostype=0, tfp=0;

#       if sun || sgi
                struct utsname name;
                char *i;
#       elif _CONVEX_SOURCE
                struct system_information sysinfo;
#       endif

#       ifdef CRAY
                machine=0;
#               ifdef _SC_CRAY_NCPU
                        processors=(int)sysconf(_SC_CRAY_NCPU);
#               else
                        processors=1;
#               endif
#       endif

#       ifdef sun
                machine=1;
#               ifdef _SC_NPROCESSORS_CONF
                        processors=(int)sysconf(_SC_NPROCESSORS_CONF);
#               else
                        processors=1;
#               endif
                uname(&name);
                if ((i=strchr(name.release,'.'))!=NULL)
                        *i='\0';
                if (strcmp(name.release,"5")==0)
                        ostype=1;
#       endif
```

Screen 64. Continued.

```
#       ifdef sgi
                machine=2;
                processors=sysmp(MP_NPROCS);
                uname(&name);
                if ((i=strchr(name.release,'.'))!=NULL)
                        *i='\0';
                if (strcmp(name.release,"6")==0)
                        ostype=1;
                if (istfp()==1)
                        tfp=1;
#       endif

#       ifdef _CONVEX_SOURCE
                machine=3;
                getsysinfo(SYSINFO_SIZE, &sysinfo);
                processors=sysinfo.cpu_count;
#       endif

        printf("machine, processors, ostype, tfp);

        exit(0);
}
```

Screen 64. Concluded.

Appendix D

Structure of PRELUDE

In addition to serving as a front end for **stArt**, the script **PRELUDE** provides file management capability in the working (**LOCAL**) directory. Specifically, the user is given the option to remove or keep **LOCAL** versions of files that are created by **stArt**. By default, **PRELUDE** uses the **$HOME/LAURA.4.1** (baseline) version of **stArt**. However, if **CUSTOM** versions of any **stArt** source files exist, a **CUSTOM** version of **stArt** is used in lieu of the baseline version. Further, if **LOCAL** versions of any **stArt** source files exist, a **LOCAL** version of **stArt** is used in lieu of the baseline or **CUSTOM** version. After **stArt** is completed, **PRELUDE** creates subdirectory **STRTfiles** and places those include files created by **stArt** there (**.strt** suffixes). If **STRTfiles** already exists, then the files it contains are updated based on this latest execution of **stArt**.

The file **conv.out** contains the running residual history for a given case. If a new case is to be run in the same directory, the user may wish to remove the existing file so that the file begins with the new case (screen 65):

```
                                VERSION=LAURA.4.1

if [ -f conv.out ]; then

        echo "File \'conv.out' already exists."
        echo " Do you wish to remove it (y/n) {n}?"
        read ANSWER

        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                rm conv.out
                echo " File \'conv.out' removed."
        else
                echo " File \'conv.out' saved."
        fi

fi
```

Screen 65.

The user is also given the option to preserve or overwrite existing RESTART.in, TWALL.in, and data files each time PRELUDE is run (screen 66):

```
# if files RESTART.in TWALL.in data already exist...

for FILE in `ls RESTART.in TWALL.in data 2> /dev/null`; do

        echo "File \`$FILE' already exists."
        echo "  Do you wish to update it \c"
        echo "during this PRELUDE session (y/n) {n}?"
        read ANSWER

        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                rm $FILE
                echo " File \`$FILE' removed."
        else
                echo " File \`$FILE' saved."
        fi

done
```

Screen 66.

Next, PRELUDE checks for the existence of LOCAL files with a .strt suffix. If any exist, the user is given the option to save or discard them before stArt is executed (screen 67):

```
for FILE in `ls *.strt 2> /dev/null`; do

        echo "LOCAL \`$FILE' exists."
        echo "  Do you wish to preserve it (y/n) {n}?"
        read ANSWER

        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                echo "  LOCAL \`$FILE"' saved.
        else
                rm $FILE
                echo "  LOCAL \`$FILE"' removed.
        fi

done
```

Screen 67.

PRELUDE expects to find the file DEFAULTS in the LOCAL directory. If it is present, but empty, PRELUDE removes it. If it is present, but outdated, the user is given the option to update it. The file is removed, if an update is specified. After these possibilities are considered, a final check is made to see if it exists. If not, the $HOME/LAURA.4.1 version is copied to the LOCAL directory (screen 68):

```
if [ -f DEFAULTS ]; then

        if [ "`wc -l DEFAULTS | awk '{print $1}'`" -lt "`wc -l $HOMER/DEFAULTS |
 awk '{print $1}'`" ]; then
                rm DEFAULTS                     # rm DEFAULTS if empty or incomplete
        fi

fi

if [ -f DEFAULTS ]; then
        if [ "`grep nprocmx DEFAULTS | awk '{print $1}'`" = "0" ]; then
                rm DEFAULTS                     # rm DEFAULTS if corrupted
        fi
fi

if [ -f DEFAULTS ]; then
        if [ "`grep nprocs DEFAULTS | awk '{print $1}'`" = "0" ]; then
                rm DEFAULTS                     # rm DEFAULTS if corrupted
        fi
fi


if [ -f DEFAULTS ]; then

        if [ ! "`grep $VERSION DEFAULTS`" ]; then

                echo "WARNING:  File \'DEFAULTS' outdated.  Update (y/n) {n}?"
                read ANSWER

                if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                        rm DEFAULTS
                fi
                echo "WARNING:  File \'INPUTS' may be outdated as well."

        fi

fi

if [ ! -f DEFAULTS ]; then                      #

        cp $home/$VERSION/DEFAULTS .            # If LOCAL DEFAULTS file DNE,
        chmod 600 DEFAULTS                      # copy INSTALL'ed version.

fi                                              #
```

Screen 68.

If any LOCAL versions of stArt source files exist, a LOCAL stArt executable is created and used in lieu of the $HOME/LAURA.4.1 version. This compilation requires the creation of symbolic links to any stArt source files (in either $HOME/LAURA.4.1 or $HOME/LAURA.4.1/CUSTOM) which are not LOCAL. If a LOCAL executable already exists, PRELUDE checks to see if it needs to be recompiled (screen 69):

```
if [ "`ls -t st*rt* | awk '{print $1}' | head -1`" = "stArt" ]; then

        echo "  < LOCAL \'stArt' executable is up to date >"

else

        if [ -f stArt ]; then
                echo "  < Updating LOCAL \'stArt' executable >"
        else
                echo "  < Creating LOCAL \'stArt' executable >"
        fi

        if [ -d $HOMER/CUSTOM ]; then
                for FILE in `cd $HOMER/CUSTOM; ls -t start* 2> /dev/null`; do
                        if [ ! -f $FILE ]; then
                                ln -s $HOMER/CUSTOM/$FILE .
                        fi
                done
        fi

        for FILE in `cd $HOMER; ls -t start*`; do
                if [ ! -f $FILE ]; then
                        ln -s $HOMER/$FILE .
                fi
        done

        $(compile) start.f -o stArt

        for FILE in `ls -t start*`; do
                if [ "`ls -lt $FILE | grep LAURA`" ]; then
                        rm $FILE
                fi
        done

fi
```

Screen 69.

where

$$compile = \begin{cases} \texttt{CF} & \text{(CRAY architectures)} \\ \texttt{FC} & \text{(all others)} \end{cases}$$

If LOCAL stArt source files are not present, the possibility of CUSTOM files is considered. If they exist and the stArt executable is not up to date, it is recompiled. First, however, symbolic links are created to any stArt source files (in $HOME/LAURA.4.1), which are not in CUSTOM (screen 70):

```
( cd $HOMER/CUSTOM;

if [ "`ls -t st*rt* | awk '{print $1}' | head -1`" = "stArt ]; then

        echo "  <       CUSTOM \`stArt' executable is up>to date

else

        if [ -f stArt ]; then
                echo "  < Updating CUSTOM \`stArt' executable >";
        else
                echo "  < Creating CUSTOM \`stArt' executable >";
        fi

        chmod 700 $HOMER;
        chmod 700 $HOMER/CUSTOM;

        for FILE in `cd $HOMER; ls -t start*`; do
                if [ ! -f $FILE ]; then
                        ln -s $HOMER/$FILE .;
                fi
        done;

        $(compile) start.f -o stArt;

        for FILE in `ls -t start*`; do
                if [ "`ls -lt $FILE | grep LAURA`" ]; then
                        rm $FILE;
                fi
        done

        chmod 500 $HOMER;
        chmod 500 $HOMER/CUSTOM;

fi )

ln -s $HOMER/CUSTOM/stArt .     # link to CUSTOM stArt
```

Screen 70.

If no LOCAL or CUSTOM versions of **stArt** source files exist, create a symbolic link to the **$HOME/LAURA.4.1 stArt** executable and use the following:

```
ln -s $HOMER/stArt .                    # ...use DEFAULT stArt.
```

Now **stArt** is executed, and it reads its required inputs either from the screen or from file INPUTS (screen 71):

```
if [ "$1" ]; then                                    # Run start...
        stArt < $1                                   #       ...reading from INPUTS.
else                                                 #
        stArt                                        #       ...reading from screen.
fi                                                   #

if [ "`ls -lt stArt | grep LAURA`" ]; then           # Remove stArt if it's
        rm stArt                                     # just a sym-link
fi                                                   # to default stArt.

if [ -f variabletw ]; then
        if [ "`wc -l variabletw | awk 'print $1'`" = "0" ]; then
                rm variabletw                        # rm variabletw if it is empty
        fi
fi
```

Screen 71.

After **stArt** is executed, several subdirectories are created (if they do not already exist). Next, the old CHILDREN file is removed, and a new symbolic link is established (screen 72):

```
mkdir OBJfiles OBJfiles/LOCAL OBJfiles/CUSTOM STRTfiles 2> /dev/null

rm -f OBJfiles/CHILDREN CHILDREN        # remove old CHILDREN file & sym-link

touch OBJfiles/CHILDREN                 # create empty OBJfiles/CHILDREN file

ln -s OBJfiles/CHILDREN CHILDREN        # create new CHILDREN sym-link
```

Screen 72.

A reference copy of `Makefile` is created, and the `INPUTS` and `DEFAULTS` files are updated (screen 73):

```
if [ ! -f .Makefile ]; then
        cp Makefile .Makefile           # make a reference copy of Makefile
fi

mv INPUTS.active INPUTS                 # update INPUTS file for stArt

# update DEFAULTS file (including VERSION number) for stArt

echo " VERSION=$VERSION" > TeMP
tail +2 DEFAULTS.active >> TeMP
mv TeMP DEFAULTS
rm DEFAULTS.active
```

Screen 73.

Based on user specifications, up to three LAURA input files are created by `stArt`: `data`, `RESTART.in`, and `TWALL.in`. In actuality, these are initially written to temporary files (with `.TeMP` suffixes). Before `stArt` was executed by `PRELUDE`, the user was given the opportunity to save `data`, `RESTART.in`, and `TWALL.in`, if they were already in existence. If a file was saved then, its new version is removed now. If it was not saved, the new file takes its place (screen 74):

```
if [ -f data ]; then
        rm data.TeMP
else
        mv data.TeMP data
fi

for FILE in `ls TWALL.TeMP RESTART.TeMP 2> /dev/null`; do       #

        INPUT=`echo $FILE | awk -F. '{print $1}'`.in

        if [ -f $INPUT ]; then
                rm $FILE
        else
                if [ "`ls -lt $FILE | grep ' 0 '`" ]; then      # remove
                        rm $FILE                                # existing file
                fi                                              # (if empty).
                mv $FILE $INPUT
        fi

done                                                            #
```

Screen 74.

Next, the files created by **stArt** (**.TeMP** suffixes) are moved to **STRTfiles** (with a **.strt** suffix). However, if the file already resides in **STRTfiles**, and it is identical to the **LOCAL** file, the **LOCAL** file is simply removed (screen 75):

```
for FILE in *.TeMP; do                                  # for *.strt just created...

        STRT=`echo $FILE | awk -F. '{print $1}'`.strt

        if [ ! -f STRTfiles/$STRT ]; then               # if file DNE in STRTfiles,

                mv $FILE STRTfiles/$STRT                # move it there,
                chmod 400 STRTfiles/$STRT               # and make it read only

        else                                            #

                diff $FILE STRTfiles/$STRT > /dev/null 2>&1

                if [ $? -ne 0 ]; then                           # if files differ...
                        echo "        Updated file \`$STRT'"
                        mv -f $FILE STRTfiles/$STRT             # move to STRTfiles
                        chmod 400 STRTfiles/$STRT               # make it read only
                else                                            #
                        echo "   No change in file \`$STRT'"
                        rm $FILE                                # else...
                fi                                              # remove LOCAL file

        fi                                              #

done
```

Screen 75.

Any **LOCAL** files with a **.strt** suffix are compared with their new counterparts (now in **STRTfiles**). If they are identical, the **LOCAL** version is removed (screen 76):

```
if [ "`ls *.strt 2> /dev/null`" ]; then

        for FILE in *.strt; do

                diff $FILE STRTfiles > /dev/null 2>&1

                if [ $? -ne 0 ]; then   # if new & old files are identical...
                        rm -f $FILE
                        echo "LOCAL \'$FILE' identical to STRTfiles version"
                        echo "  Therefore, LOCAL \'$FILE' removed."
                fi

        done

fi
```

Screen 76.

An estimate for the memory required, based on values from **stArt**, is calculated through execution of the LAURA utility **SIZEIT** (appendix M). If an external grid (*newjob* = 0) was specified in **stArt**, the user is given the option to initialize this grid now. This initialization requires that the user input the name of the file that contains the grid (in **PLOT3D** format). The LAURA utility **INITIALIZE** (appendix I) is then executed for this file (screen 77):

```
$HOME/$VERSION/SIZEIT

if [ "`grep newjob DEFAULTS | awk '{print $1}'`" = "0" ]; then

        if [ ! -f RESTART.in ]; then            # if RESTART.in DNE...

                echo "Do you wish to initialize grid file now (y/n) {n}?"
                read ANSWER

                if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                        echo "Enter name of grid file:  "
                        read GRIDIN
                        $HOME/$VERSION/INITIALIZE $GRIDIN
                fi

        fi

fi
```

Screen 77.

Appendix E

Makefile and Its Supporting Files

When PRELUDE is run, **stArt** creates a sophisticated **Makefile** for LAURA based on the user inputs and the machine architecture. When the resultant **Makefile** is activated, it in turn executes a number of script files. The structure of the **Makefile** and its supporting cast are discussed in the subsections of this appendix.

E.1. Structure of Makefile

The **Makefile** contains a preamble that is common to all cases and architectures (screen 78):

```
PROG=   laura

SHELL=  /bin/sh

SOURCE= $(HOME)/LAURA.4.1
OBJDIR= OBJfiles
STRTDR= STRTfiles
DBGDIR= BUGOUT
FORDIR= FORTRAN
```

Screen 78.

The source files are supplied through the **SRCS** list. The contents of this list are case dependent as shown below (screen 79):

```
SRCS=   aaa.F abseig.F abseig1.F algnshk.F asave.F atime.F \
        blkout.F bmat.F bndr.F bndrfmn.F bndrmn.F bound.F \
        blturb.F csturb.F parab.F prabola.F stretch.F \              (turbulence)
        boundf.F boundr.F boundu.F boundv.F consrv.F defbod.F \
        defmom.F dirswp.F double.F dropone.F drv.F efg.F \
        etherm.F prand.F tannehill.F vinokur.F vintabl.F visc.F \  (equilibrium)
        gatdf1.F gatdf2.F gatdf3.F gatgeof.F nsbnd.F \       (full Navier-Stokes)
        gatface.F gatfgmn.F gatfmn.F gatgeo.F gatgeoa.F gatgmn.F \
        gatmn.F gatscat.F gatscta.F gauss.F invflx.F limiter.F \
        inv2d.F \                                                  (ndim ≠ 3)
        kinetic.F source.F air.F \                            (nonequilibrium)
        metric.F minmod.F moment.F outputa.F plotprep.F prpavg.F \
        prpaxj.F q4iuniv.F reload.F rmat.F saveblk.F \
        sample_handler.F \                                            (SUN)
        setup.F swptask.F taskit.F thermo.F trnsprt.F viscflx.F \
        sthrlnd.F \                                          (perfect gas)
        wrapup.F

OBJS=   $(SRCS:.F=.o)
```

Screen 79.

NOTE: The LAURA algorithm uses compile directives to activate and deactivate coding based on user specifications for a given application. These directives are used by the host machine's preprocessor. On most architectures, the preprocessor is applied automatically to any file with a `.F` suffix during its **FORTRAN** compilation. For CONVEX architectures, however, the preprocessor is applied automatically to any file with a `.FOR` suffix. This anomaly is handled during the installation process (chapter 5) and by **stArt**. However, notice the `.FOR` suffixes if the **Makefile** is edited on CONVEX architectures.

The compilation flags (which are architecture dependent) are supplied next. For CRAY machines, they are as follows (screen 80):

```
NPROC= nprocs

#PROFLIB=lprof                          # Profile library

#FFLAGS=-Wf"-a stack -ez"               # profile
#FFLAGS=-Wf"-a stack -ez" -F            # flowtrace
FFLAGS= -Wf"-a stack -o aggress"        # optimize
DFLAGS= -Wf"-a stack -g"                # debug
```

Screen 80.

For SUN machines, the flags are as follows (screen 81):

```
#FFLAGS=-C -p                    # check bounds & profile
FFLAGS= -O3                      # optimize
DFLAGS= -C -g                    # check bounds & debug
```

Screen 81.

For SGI machines, they are as follows (screen 82):

```
#FFLAGS=-check_bounds -p         # check bounds & profile
FFLAGS= -O2                      # optimize
DFLAGS= -check_bounds -g         # check bounds & debug
```

Screen 82.

For CONVEX machines, they are as follows (screen 83):

```
#FFLAGS=-cs -p                   # check bounds & profile
FFLAGS= -O3                      # optimize
DFLAGS= -cs -g                   # check bounds & debug
```

Screen 83.

The epilogue to the **Makefile** is

```
include CHILDREN
```

which provides the **Makefile** with a list of included files (**.inc** and **.strt** suffixes) that each source file (**.F**, **.FOR**, or **.f** suffix) is dependent upon. This allows **make** to recognize whether a file (or any **include** file it depends upon) has been modified since the last **make**. If not, the file is not recompiled, thus saving compilation time and costs.

E.1.1. Command: make

The command

make

executes the following "default" procedure (screen 84):

```
default:
        @echo ""; \
        @echo "<Estimating memory requirements for executable   >"; \
        @echo ""; \
        @$(SOURCE)/SIZEIT

        @echo ""; \
        @echo "<Building symbolic links in $(OBJDIR)            >"; \
        @echo ""; \
        @$(SOURCE)/SYMLINKS $(SOURCE) $(OBJDIR) $(STRTDR) $(PROG) $(DBGDIR)

        @if [ "$(FORTDR)" = "$(FORDIR)" ]; then \
                echo ""; \
                echo "< Preprocessing .F files                 >"; \
                echo ""; \
                ( cd $(FORDIR); \
                  for FILE in $(SRCS); do \
```

Screen 84.

The structure of the script **SIZEIT** is given in appendix M. The structure of the script **SYMLINKS** is given in appendix E, section E.2. The preprocessing sequence is machine dependent. For CRAY and SGI machines, it is as follows:

```
$(compile) -P ../$(OBJDIR)/$$FILE; \
mv `basename $$FILE .F`.i `basename $$FILE .F`.f; \
```

where

$$compile = \begin{cases} \texttt{CF} & \text{(CRAY architectures)} \\ \texttt{FC} & \text{(all others)} \end{cases}$$

For SUN machines, it is as follows:

```
$(FC) -F ../$(OBJDIR)/$$FILE; \
```

For CONVEX machines, it is as follows:

```
NOEXT=`basename ../$(OBJDIR)/$$FILE .FOR`; \
cpp ../$(OBJDIR)/$$FILE > $$NOEXT.f; \
```

The remainder of the default procedure is shown below (screen 85):

```
                done ); \
        else \
                $(SOURCE)/CHECKERS $(OBJDIR) $(DBGDIR); \
                echo ""; \
                echo "< Building object files and executable    >"; \
                echo ""; \
                ( cd $(OBJDIR); \
                  make FFLAGS='$(FFLAGS)' $(PROG) ); \
        fi

        @if [ ! "$(OBJDIR)" = "$(DBGDIR)" ]; then \
                echo ""; \
                echo "< Removing symbolic links in $(OBJDIR)    >"; \
                echo ""; \
                ( cd $(OBJDIR); \
                  rm -f Make* start* *.F* *.f *.inc *.strt *.trace; \
                  exit 0 ); \
        fi

$(PROG):$(OBJS)
        $(compile) $(FFLAGS) -o $@ $(OBJS)
```

Screen 85.

where

$$make = \begin{cases} \texttt{pmake} - Jnprocs, & \text{(SGI architectures)} \\ \texttt{make}, & \text{(all others)} \end{cases}$$

The structure of the script **CHECKERS** is given in appendix E, section E.3. The following sequence is required for SGI architectures only:

```
$(OBJS):$(@:.o=.F)
        $(FC) $(FFLAGS) -c $(@:.o=.F)
```

The following sequence is required for CONVEX architectures only:

```
$(OBJS):$(@:.o=.FOR)
        $(FC) -pp=cpp $(FFLAGS) -c $(@:.o=.FOR)
```

E.1.2. Command: make debug

The command

```
make debug
```

executes the following procedure (which, in turn, executes the default **make** procedure, (screen 86)):

```
debug:
        @if [ ! -d $(DBGDIR) ]; then \
                echo ""; \
                echo "< Creating $(DBGDIR) to hold required files       >"; \
                echo ""; \
                mkdir $(DBGDIR) $(DBGDIR)/CUSTOM $(DBGDIR)/LOCAL; \
                for FILE in data RESTART.in transition TWALL.in variabletw; do \
                        if [ -f $$FILE ]; then \
                                ln -s ../$$FILE $(DBGDIR)/$$FILE; \
                        fi \
                done \
        fi
        @make FFLAGS='$(DFLAGS)' OBJDIR=$(DBGDIR)
```

Screen 86.

E.1.3. Command: make fortran

The command

```
make fortran
```

executes the following procedure (which, in turn, executes the default **make** procedure (screen 87)):

```
fortran:
        @if [ ! -d $(FORDIR) ]; then \
                echo ""; \
                echo "< Creating $(FORDIR) to hold preprocessed files   >"; \
                echo ""; \
                mkdir $(FORDIR) ; \
        fi
        @make FORTDR=$(FORDIR)
```

Screen 87.

E.1.4. Command: `make clean`

The command

`make clean`

removes all existing object files (`.o` suffix) from the `OBJfiles` subdirectory. The procedure is as follows (screen 88):

```
clean:
        @if [ "`cd $(OBJDIR); ls *.o 2> /dev/null`" ]; then \
                echo ""; \
                echo "< Removing object files from $(OBJDIR)            >"; \
                echo ""; \
                rm $(OBJDIR)/*.o; \
        else \
                echo ""; \
                echo "< ERROR:  No object files exist in $(OBJDIR)      >"; \
                echo ""; \
        fi
```

Screen 88.

E.2. Structure of SYMLINKS

The script **SYMLINKS** is executed by the **Makefile** during its compilation of the LAURA code. This script establishes symbolic links to the source files necessary for this compilation. As a result, the **Makefile** knows when to use a **LOCAL** or **CUSTOM** version of a file in lieu of the installed (**$HOME/LAURA.4.1**) version. The first step in this linking process is to remove any links that might remain from a previously aborted compilation (screen 89):

```
                                VERSION=LAURA.4.1

SOURCE=$1
OBJECT=$2
START=$3
PROG=$4
DEBUG=$5

cd $OBJECT

rm -f Make* *.data *.F* *.f *.inc *.strt 2> /dev/null
```

Screen 89.

Next, the default links are reestablished (screen 90):

```
ln -s ../$START/*        . 2> /dev/null
ln -s ../Makefile        . 2> /dev/null

ln -s $SOURCE/Makedep*   . 2> /dev/null
ln -s $SOURCE/*.F*       . 2> /dev/null
ln -s $SOURCE/*.inc      . 2> /dev/null
ln -s $SOURCE/*.data     . 2> /dev/null

if [ "$OBJECT" != "$DEBUG" ]; then
        rm -f ../$PROG 2> /dev/null
        ln -s $OBJECT/$PROG ../$PROG
fi

rm -f start* CUSTOM/*.rm LOCAL/*.rm 2> /dev/null
```

Screen 90.

If any CUSTOM files exist, they are used instead of the $HOME/LAURA.4.1 versions. To accomplish this, the $HOME/LAURA.4.1 symbolic link is replaced with a link to the CUSTOM file. The possibility that CUSTOM files from the last compilation will no longer exist is also considered. First, the status of CUSTOM subroutine files (.F, .FOR, and .f suffixes) is checked (screen 91):

```
rm CUSTOM/*.rm 2> /dev/null

for FILE in `cd CUSTOM; ls *.F* *.f 2> /dev/null`; do
        echo "CUSTOM \`$FILE' used in last compilation..."
        NOEXT=`echo $FILE | awk -F. '{print $1}'`
        if [ ! -f $SOURCE/CUSTOM/$FILE ]; then
                echo "but CUSTOM \`$FILE' does not exist."
                if [ -f $NOEXT.o ]; then
                        rm $NOEXT.o
                        rm CUSTOM/$FILE
                fi
        else
                echo "  CUSTOM \`$FILE' still exists, and will be used in this c
ompilation."
                if [ ! "`grep $VERSION $SOURCE/CUSTOM/$FILE`" ]; then
                        echo "  WARNING:  VERSION mismatch.  Still use (y/n) {n}
?"
                        read ANSWER
                        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                                rm $NOEXT.F*
                                ln -s $SOURCE/CUSTOM/$FILE .
                        else
                                rm $NOEXT.o 2> /dev/null
                                touch CUSTOM/$FILE.rm
                        fi
                else
                        rm $NOEXT.F*
                        ln -s $SOURCE/CUSTOM/$FILE .
                fi
        fi
done
```

Screen 91.

Next, the status of CUSTOM include files (.inc suffixes) is reviewed (screen 92):

```
for INC in 'cd CUSTOM; ls *.inc 2> /dev/null'; do
        echo "CUSTOM \'$INC' used in last compilation..."
        if [ ! -f $SOURCE/CUSTOM/$INC ]; then
                echo "but CUSTOM \'$INC' does not exist."
                rm CUSTOM/$INC
                for DIR in $SOURCE $SOURCE/CUSTOM ..; do
                        for FILE in 'cd $DIR; grep -l $INC *.F* *.f 2> /dev/null
'; do
                                NOEXT='echo $FILE | awk -F. '{print $1}''
                                if [ -f $NOEXT.o ]; then
                                        rm $NOEXT.o
                                fi
                        done
                done
        else
                echo "  CUSTOM \'$INC' still exists, and will be used in this co
mpilation."
                if [ ! "'grep $VERSION $SOURCE/CUSTOM/$INC'" ]; then
                        echo "WARNING:  VERSION mismatch.  Still use (y/n) {n}?"
                        read ANSWER
                        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                                rm $INC
                                ln -s $SOURCE/CUSTOM/$INC .
                        else
                                touch CUSTOM/$INC.rm
                                for DIR in $SOURCE $SOURCE/CUSTOM ..; do
                                        for FILE in 'cd $DIR; grep -l $INC *.F*
*.f 2> /dev/null'; do
                                                NOEXT='echo $FILE | awk -F. '{pr
int $1}''
                                                if [ -f $NOEXT.o ]; then
                                                        rm $NOEXT.o
                                                fi
                                        done
                                done
                        fi
                else
                        rm $INC
                        ln -s $SOURCE/CUSTOM/$INC .
                fi
        fi
done
```

Screen 92.

New CUSTOM files may have been created since the last compilation. Therefore, the next step is to locate any CUSTOM files that were not used in the previous compilation and replace the $HOME/LAURA.4.1 link with a link to the CUSTOM file (screen 93):

```
for FILE in `cd $SOURCE/CUSTOM; ls *.F* *.f 2> /dev/null | grep -v start`; do
        if [ "`cd CUSTOM; ls $FILE 2> /dev/null`" ]; then
                if [ "`cd CUSTOM; ls $FILE.rm 2> /dev/null`" ]; then
                        rm CUSTOM/$FILE*
                fi
        else
                echo "CUSTOM \`$FILE' exists, and will be used in this compilati
on."
                if [ ! "`cd LOCAL; ls $FILE 2> /dev/null`" ]; then
                        NOEXT=`echo $FILE | awk -F. '{print $1}'`
                        if [ ! "`grep $VERSION $SOURCE/CUSTOM/$FILE`" ]; then
                                echo "  WARNING:  VERSION mismatch.  Still use (
y/n) {n}?"
                                read ANSWER
                                if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; the
n
                                        rm $NOEXT.* CUSTOM/$NOEXT.* 2> /dev/null
                                        ln -s $SOURCE/CUSTOM/$FILE .
                                        touch CUSTOM/$FILE
                                fi
                        else
                                rm $NOEXT.* CUSTOM/$NOEXT.* 2> /dev/null
                                ln -s $SOURCE/CUSTOM/$FILE .
                                touch CUSTOM/$FILE
                        fi
                fi
        fi
done
```

Screen 93.

```
for INC in `cd $SOURCE/CUSTOM; ls *.inc 2> /dev/null | grep -v start`; do
        if [ "`cd CUSTOM; ls $INC 2> /dev/null`" ]; then
                if [ "`cd CUSTOM; ls $INC.rm 2> /dev/null`" ]; then
                        rm CUSTOM/$INC*
                fi
        else
                echo "CUSTOM \`$INC' exists, and will be used in this compilatio
n."
                if [ ! "`cd LOCAL; ls $INC 2> /dev/null`" ]; then
                        if [ ! "`grep $VERSION $SOURCE/CUSTOM/$INC`" ]; then
                                echo "  WARNING:  VERSION mismatch.  Still use (
y/n) {n}?"
                                read ANSWER
                                if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; the
n
                                        rm $INC
                                        ln -s $SOURCE/CUSTOM/$INC .
                                        touch CUSTOM/$INC
                                else
                                        for DIR in $SOURCE $SOURCE/CUSTOM ..; do
                                                for FILE in `cd $DIR; grep -l $I
NC *.F* *.f 2> /dev/null`; do
                                                        NOEXT=`echo $FILE | awk
-F. '{print $1}'`
                                                        if [ -f $NOEXT.o ]; then
                                                                rm $NOEXT.o
                                                        fi
                                                done
                                        done
                                fi
                        else
                                rm $INC
                                ln -s $SOURCE/CUSTOM/$INC .
                                touch CUSTOM/$INC
                        fi
                fi
        fi
done
```

Screen 93. Concluded.

An analogous procedure is employed for LOCAL files. If a LOCAL file exists, it is used in place of the $HOME/LAURA.4.1 or CUSTOM versions through the creation of a link to the LOCAL file. The status of LOCAL subroutine files (.F, .FOR, and .f suffixes) is checked (screen 94):

```
rm LOCAL/*.rm 2> /dev/null

for FILE in `cd LOCAL; ls *.F* *.f 2> /dev/null`; do
        echo "LOCAL \`$FILE' used in last compilation..."
        NOEXT=`echo $FILE | awk -F. '{print $1}'`
        if [ ! -f ../$FILE ]; then
                echo "  but LOCAL \`$FILE' does not exist."
                if [ -f $NOEXT.o ]; then
                        if [ -f $SOURCE/CUSTOM/$FILE ]; then
                                touch CUSTOM/$FILE
                        fi
                        rm $NOEXT.o
                        rm LOCAL/$FILE
                fi
        else
                echo "  LOCAL \`$FILE' still exists, and will be used in this co
mpilation."
                if [ ! "`grep $VERSION ../$FILE`" ]; then
                        echo "          WARNING:  VERSION mismatch.  Still use (
y/n) {n}?"
                        read ANSWER
                        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                                rm $NOEXT.F* CUSTOM/$NOEXT.* 2> /dev/null
                                ln -s ../$FILE .
                        else
                                if [ -f $SOURCE/CUSTOM/$FILE ]; then
                                        touch CUSTOM/$FILE
                                fi
                                rm $NOEXT.o 2> /dev/null
                                touch LOCAL/$FILE.rm
                        fi
                else
                        rm $NOEXT.F* CUSTOM/$NOEXT.* 2> /dev/null
                        ln -s ../$FILE .
                fi
        fi
done
```

Screen 94.

Next, the status of LOCAL include files (.inc and .strt suffixes) is reviewed (screen 95):

```
for INC in `cd LOCAL; ls *.inc *.strt 2> /dev/null`; do
        echo "LOCAL \`$INC' used in last compilation..."
        if [ ! -f ../$INC ]; then
                echo "but LOCAL \`$INC' does not exist."
                if [ -f $SOURCE/CUSTOM/$INC ]; then
                        touch CUSTOM/$INC
                fi
                rm LOCAL/$INC
                for DIR in $SOURCE $SOURCE/CUSTOM ..; do
                        if [ -d $DIR ]; then
                                for FILE in `cd $DIR; grep -l $INC *.F* *.f 2> /
dev/null`; do
                                        NOEXT=`echo $FILE | awk -F. '{print $1}'
`
                                        if [ -f $NOEXT.o ]; then
                                                rm $NOEXT.o
                                        fi
                                done
                        fi
                done
        else
                echo "  LOCAL \`$INC' still exists, and will be used in this com
pilation."
                if [ ! "`grep $VERSION ../$INC`" ]; then
                        echo "WARNING:  VERSION mismatch.  Still use (y/n) {n}?"
                        read ANSWER
                        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                                rm $INC CUSTOM/$INC 2> /dev/null
                                ln -s ../$INC .
                        else
                                if [ -f $SOURCE/CUSTOM/$FILE ]; then
                                        touch CUSTOM/$FILE
                                fi
                                touch LOCAL/$INC.rm
                                for DIR in $SOURCE $SOURCE/CUSTOM ..; do
                                        if [ -d $DIR ]; then
                                                for FILE in `cd $DIR; grep -l $I
NC *.F* *.f 2> /dev/null`; do
                                                        NOEXT=`echo $FILE | awk
-F. '{print $1}'`
                                                        if [ -f $NOEXT.o ]; then
                                                                rm $NOEXT.o
                                                        fi
                                                done
                                        fi
                                done
                        fi
                else
                        rm $INC CUSTOM/$INC 2> /dev/null
                        ln -s ../$INC .
                fi
        fi
done
```

Screen 95.

Finally, any LOCAL files that were not used in the previous compilation are located, and the existing link is replaced with a link to the LOCAL file (screen 96):

```
for FILE in `cd ..; ls *.F* *.f 2> /dev/null | grep -v start`; do
        if [ "`cd LOCAL; ls $FILE 2> /dev/null`" ]; then
                if [ "`cd LOCAL; ls $FILE.rm 2> /dev/null`" ]; then
                        rm LOCAL/$FILE*
                fi
        else
                echo "LOCAL \`$FILE' exists, and will be used in this compilatio
n."
                NOEXT=`echo $FILE | awk -F. '{print $1}'`
                if [ ! "`grep $VERSION ../$FILE`" ]; then
                        echo "WARNING:  VERSION mismatch.  Still use (y/n) {n}?"
                        read ANSWER
                        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                                rm $NOEXT.* CUSTOM/$NOEXT.* 2> /dev/null
                                ln -s ../$FILE .
                                touch LOCAL/$FILE
                        else
                                if [ -f $SOURCE/CUSTOM/$FILE ]; then
                                        touch CUSTOM/$FILE
                                fi
                        fi
                else
                        rm $NOEXT.* CUSTOM/$NOEXT.* 2> /dev/null
                        ln -s ../$FILE .
                        touch LOCAL/$FILE
                fi
        fi
done
```

Screen 96.

```
for INC in `cd ..; ls *.inc *strt 2> /dev/null | grep -v start`; do
        if [ "`cd LOCAL; ls $INC 2> /dev/null`" ]; then
                if [ "`cd LOCAL; ls $INC.rm 2> /dev/null`" ]; then
                        rm LOCAL/$INC*
                fi
        else
                echo "LOCAL \`$INC' exists, and will be used in this compilation
."
                NOEXT=`echo $INC | awk -F. '{print $1}'`
                if [ ! "`grep $VERSION ../$INC`" ]; then
                        echo "WARNING:  VERSION mismatch.  Still use (y/n) {n}?"
                        read ANSWER
                        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                                rm $INC CUSTOM/$INC 2> /dev/null
                                ln -s ../$INC .
                                touch LOCAL/$INC
                                for DIR in $SOURCE $SOURCE/CUSTOM ..; do
                                        if [ -d $DIR ]; then
                                                for FILE in `cd $DIR; grep -l $I
NC *.F* *.f 2> /dev/null`; do
                                                        NOEXT=`echo $FILE | awk
-F. '{print $1}'`
                                                        if [ -f $NOEXT.o ]; then
                                                                rm $NOEXT.o
                                                        fi
                                                done
                                        fi
                                done
                        else
                                if [ -f $SOURCE/CUSTOM/$INC ]; then
                                        touch CUSTOM/$INC
                                fi
                        fi
                else
                        rm $INC CUSTOM/$INC 2> /dev/null
                        ln -s ../$INC .
                        touch LOCAL/$INC
                        for DIR in $SOURCE $SOURCE/CUSTOM ..; do
                                if [ -d $DIR ]; then
                                        for FILE in `cd $DIR; grep -l $INC *.F*
*.f 2> /dev/null`; do
                                                NOEXT=`echo $FILE | awk -F. '{pr
int $1}'`
                                                if [ -f $NOEXT.o ]; then
                                                        rm $NOEXT.o
                                                fi
                                        done
                                fi
                        done
                fi
        fi
done
```

Screen 96. Concluded.

The final task performed by SYMLINKS is to create the file ECHOSTRT. This file is a concatenation of those include files (with a .strt suffix) which contain FORTRAN parameter statements. As discussed in section 9.2, these files provide user control of LAURA during compilation. At laura run-time, this file is printed to standard output.

> **NOTE:** The file ECHOSTRT should not be mistakenly edited in an attempt to effect changes to the laura executable. Modifications should be made via PRELUDE, or through a LOCAL version of the appropriate source files, followed by a recompilation of laura (screen 97):

```
rm -f ECHOSTRT 2> /dev/null
cat $HOME/$VERSION/TOP > ECHOSTRT
echo "  File \'ECHOSTRT' was created by \'make' on 'date'." >> ECHOSTRT

for FILE in     HEADER.strt algnshk_vars.strt gas_model_vars.strt \
                issd_assn.strt iupwind_assn.strt mtaska_assn.strt \
                nordbc_assn.strt parameter.strt source_vars.strt \
                sthrlnd_vars.strt; do

        if [ -f $FILE ]; then
                echo "Contents of LOCAL file \'$FILE':" >> ECHOSTRT
                LINES='wc -l $FILE | awk '{print $1}''
                LINES='expr $LINES - 8'
                tail +6 $FILE | head -$LINES >> ECHOSTRT
        else
                if [ "'cd $START; wc -l $FILE | awk '{print $1}''" = "8" ]; then
                        echo "File \'$FILE' is not active." >> ECHOSTRT
                else
                        echo "Contents of file \'$FILE':" >> ECHOSTRT
                        LINES='wc -l $START/$FILE | awk '{print $1}''
                        LINES='expr $LINES - 8'
                        tail +6 $START/$FILE | head -$LINES >> ECHOSTRT
                fi
        fi

done
```

Screen 97.

> **NOTE:** As mentioned in section 10.1, the contents of file data (as well as files assign_tasks, transition, and variabletw, if they exist at run-time), are also echoed to standard output. However, since these files provide user control during execution rather than compilation, their contents are not included in file ECHOSTRT. Rather, as their contents are read in by LAURA, they are also echoed to the screen so that the values that are actually used in the current run are reflected.

The logic here checks for the presence of LOCAL versions of these files. If any are present, they are concatenated in lieu of the originals produced by stArt (which are located in STRTfiles, a subdirectory of the LOCAL directory). A complete list of LOCAL and CUSTOM files used in this

compilation is also included in ECHOSTRT. Thus, ECHOSTRT is a record of the user-defined tailoring that was used in the most recent compilation of LAURA (screen 98):

```
if [ -d $OBJECT/LOCAL ]; then
        if [ "`ls -t $OBJECT/LOCAL`" ]; then            # if dir is not empty...
                echo "The following LOCAL files were used in the last \`make':"
>> ECHOSTRT
                for FILE in `cd $OBJECT/LOCAL; ls * 2> /dev/null`; do
                echo " $FILE" >> ECHOSTRT
                done
        fi                                              #
fi

if [ -d $OBJECT/CUSTOM ]; then
        if [ "`ls -t $OBJECT/CUSTOM`" ]; then           # if dir is not empty...
                echo "The following CUSTOM files were used in the last \`make':"
>> ECHOSTRT
                for FILE in `cd $OBJECT/CUSTOM; ls * 2> /dev/null`; do
                        if [ ! -f $OBJECT/LOCAL/$FILE ]; then
                                echo "  $FILE" >> ECHOSTRT
                        fi
                done
        fi                                              #
fi

chmod 400 ECHOSTRT
```

Screen 98.

E.3. Structure of CHECKERS

CHECKERS executes **Makedep** to determine which included files each source file depends upon. The structure of the script **Makedep** is given in appendix E, section E.4. CHECKERS also compares the **Makefile** being executed with a reference copy (file **.Makefile**). If the compilation flags have been changed since the last **make**, the object files are removed. File **. Makefile** is initially created by PRELUDE. It is updated each time this check is positive (screen 99):

```
OBJECT=$1
DEBUG=$2

echo "< Building dependency list >"

( cd $OBJECT; Makedep > CHILDREN )

REMOVER=0

if [ "$OBJECT" = "$DEBUG" ]; then

        if [ -f $DEBUG/.Makefile ]; then

                if [ "`diff Makefile $DEBUG/.Makefile | grep DFLAGS=`" ]; then
                        REMOVER=1
                fi

                if [ "`diff Makefile $DEBUG/.Makefile | grep LFLAGS=`" ]; then
                        REMOVER=1
                fi

        else

                cp Makefile $DEBUG/.Makefile

        fi

else

        if [ "`diff Makefile .Makefile | grep FFLAGS=`" ]; then
                REMOVER=1
        fi

        if [ "`diff Makefile .Makefile | grep LFLAGS=`" ]; then
                REMOVER=1
        fi

fi
```

Screen 99.

```
if [ "$REMOVER" = "1" ]; then

        if [ "`cd $OBJECT; ls *.o 2> /dev/null`" ]; then

                echo "  <       Flags in \'Makefile' have changed      >"
                echo "  <       since last compilation, therefore      >"
                echo "  <       removing object files from $OBJECT.    >"

                rm $OBJECT/*.o

                if [ "$OBJECT" = "$DEBUG" ]; then
                        cp Makefile $DEBUG/.Makefile
                else
                        cp Makefile .Makefile
                fi

        fi

fi
```

Screen 99. Concluded.

E.4. Structure of Makedep

This script determines the **include** dependencies of each LAURA source file. **Makedep** is the front end for the "awk" (a UNIX utility) file **Makedep.awk** (screen 100):

```
for file in *.f *.F *.FOR; do

        awk -f Makedep.awk $file 2> /dev/null

done
```

Screen 100.

Makedep.awk establishes the file interdependencies by locating all **include** statements in each of the subroutine files (**.f**, **.F**, and **.FOR** suffixes). (See screen 101.)

```
BEGIN   {
        n=0;
        }

# With BEGIN finished, read the file.

/     include/ {

        m=split($0,nf,"\"")                 # This is the pattern to match.
        if (m==1) m=split($0,nf,"'");       # Try to split the line using the "
        if (m>1)                            # separator.  If line was not split
                {                           # into m > 1 parts, then try to split
                fc=substr(nf[1],1,1);       # using the ' separator.  If either
                if (fc=="#" || fc==" ")     # split works, then check whether the
                        {                   # character in column 1 is a # or space.
                        n++;                # If either is true, then the 2nd
                        fn[n]=nf[2]         # field on the line is the file name
                        }                   # to include.
                }
        }
```

Screen 101.

Although only "preprocessor" **include** statements are used in LAURA, this search also checks for **FORTRAN include** statements that might be introduced in tailored files.

These dependencies are output in a format that is usable in the **Makefile** (screen 102).

```
END     {
        split(FILENAME,inp,".");        # Extract the input filename.
        line=inp[1] ".o:  \t";          # Add .o extension and tab to name.
        if (n==0)                       # If no includes were found in
                {                       # this file, then just print
                print line;             # the filename and exit.
                exit ;
                }
        j=1;
        while (j<=n)                    # Loop though each included filename.
                {
                i=1;
                while (i<=3)            # Only put 3 filenames on one line.
                        {
                        line=line fn[j] "  ";          # Concatenate the names.
                        j++;
                        if (j-1==n) break;             # Determine if
                        if (i==3) line=line "\\";      # continuation
                        i++;                           # characters
                        }                              # are required.
                print line;             # Output the line.
                line="\t \t";
                }
        }
```

Screen 102.

Makedep is executed each time the **Makefile** is executed, and its output is directed to the file **CHILDREN**. The file **CHILDREN** is accessed by the **Makefile** to determine which include files (**.inc** and **.strt** suffixes) each source file (**.F**, **.FOR**, or **.f** suffix) is dependent upon. If any of these files have been modified more recently than the object file (**.o** suffix) was created, then the object file is recompiled. In other words, with each execution of **make**, only those object files that are outdated are recompiled.

Appendix F

Structure of ARCHIVE

The script **ARCHIVE** provides the user with an archival capability for the given working directory. In other words, a user can use this command to save key files for future use. These files can be restored at some future date using the **RESTORE** command (appendix L). After the restoration process, the solution can be picked up where it left off.

The command **ARCHIVE** is executed from the present working (**LOCAL**) directory. The procedure is as follows. First, a temporary subdirectory named **ARCHIVE** is created:

```
HOMER=$HOME/$VERSION

mkdir ARCHIVE 2> /dev/null
```

Within this directory, several subdirectories are created to hold various classes of files. First, the **DEFAULTS** and **INPUTS** files are saved in case the user would like to repeat **PRELUDE** in the future (screen 103):

```
if [ "`ls DEFAULTS INPUTS 2> /dev/null`" ]; then

        mkdir ARCHIVE/IN 2> /dev/null

        for FILE in DEFAULTS INPUTS; do

                if [ -f $FILE ]; then
                        cp $FILE ARCHIVE/IN
                fi

        done

fi
```

Screen 103.

Next, the input files for `laura` (`RESTART.in`, `assign_tasks`, `transition`, `TWALL.in`, and `variabletw`) are saved to provide a smooth restart capability (screen 104):

```
mkdir ARCHIVE/CONTROL 2> /dev/null

for FILE in RESTART.in assign_tasks data transition TWALL.in variabletw; do
        if [ -f $FILE ]; then
                cp $FILE ARCHIVE/CONTROL
        fi
done

if [ ! "`ls -t ARCHIVE/CONTROL`" ]; then
        rmdir ARCHIVE/CONTROL
fi
```

Screen 104.

If the master input files (`RESTART.MASTER` and `TWALL.MASTER`) exist, they are also saved, along with the file `conv.out`, which contains the convergence history for the run (screen 105):

```
if [ "`ls RESTART.MASTER TWALL.MASTER 2> /dev/null`" ]; then

        mkdir ARCHIVE/MASTER 2> /dev/null

        for FILE in RESTART.MASTER TWALL.MASTER; do
                if [ -f $FILE ]; then
                        cp $FILE ARCHIVE/MASTER
                fi
        done

fi

if [ -f conv.out ]; then
        mkdir ARCHIVE/CONV 2> /dev/null
        cp conv.out ARCHIVE/CONV
fi
```

Screen 105.

Copies of any CUSTOM files that might exist are saved as follows (screen 106):

```
if [ "`ls OBJfiles/CUSTOM`" ]; then
        mkdir ARCHIVE/CUSTOM 2> /dev/null
        for FILE in `ls OBJfiles/CUSTOM 2> /dev/null`; do
                cp $HOMER/CUSTOM/$FILE ARCHIVE/CUSTOM
        done
fi
```

Screen 106.

Any LOCAL files are also saved (screen 107):

```
if [ "`ls OBJfiles/LOCAL`" ]; then
        mkdir ARCHIVE/LOCAL 2> /dev/null
        for FILE in `ls OBJfiles/LOCAL 2> /dev/null`; do
                cp $FILE ARCHIVE/LOCAL
        done
fi
```

Screen 107.

Now the contents of the subdirectory **ARCHIVE** are ready to be packaged in a **tarfile**. First, the user is prompted for the desired name for this file. Next the **tarfile** is created. As a final step, the subdirectory **ARCHIVE** and its contents are removed (screen 108):

```
echo "  Enter name for ARCHIVE file:   "
read TARFILE

if [ -f $TARFILE ]; then

        echo "  File \'$TARFILE' already exists.  Overwrite it (y/n) {n}?  "
        read ANSWER

        if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                rm -f $TARFILE
        else
                echo " ARCHIVE procedure aborted"
                exit 0
        fi

fi

cd ARCHIVE

tar cf ../$TARFILE *
chmod 400 ../$TARFILE

cd ..

rm -r ARCHIVE
```

Screen 108.

Appendix G

Structure of BLOX

The script **BLOX** allows a user to assemble several computational blocks into a single-block, master copy of the solution. This utility also allows the user to partition a master copy of the solution into several blocks to reduce memory overhead associated with multitasking. Smaller working blocks can also be created from the master to concentrate relaxation cycles in critical regions or to implement a block marching strategy (from nose to tail and/or from windside to leeside) over vehicles. As solutions in the smaller working blocks are converged, they can be resaved in the master copy using the **BLOX** utility.

BLOX operates on the working copy restart file, **RESTART.in**, and, if necessary, on the associated wall temperature file, **TWALL.in**. The respective master files are called **RESTART.MASTER** and **TWALL.MASTER**. To use this utility, the user types

BLOX

which serves as a front-end to the **mAkEblk FORTRAN** executable. The user is prompted for information through a series of questions. When this interactive **mAkEblk** session is completed, the **BLOX** utility automatically compiles file **exchange.f** to create **ExchAngE**, which is the **FORTRAN** executable that implements the specified exchange from either the working file to the master file or vice versa.

The utility then attempts to execute **ExchAngE**. If the restart files are very large, local system defaults may not permit interactive execution of **ExchAngE**. In these cases, the user will need to submit **ExchAngE** in the working directory as a batch job according to local system protocol. A sample script file is shown in screen 109.

```
.

# QSUB -lT 100
# QSUB -lM 20mw

cd work_dir

ExchAngE
```

Screen 109.

Here, it is specified that the size of the executable is less than 20 megawords and requires less than 100 seconds of CPU time (screen 110).

> **NOTE:** The utility **BLOX** has no effect on the executable **laura**. Changes in boundary conditions or working block dimensions associated with reblocking, if necessary, are implemented with the utility **PRELUDE**.

```
                                VERSION=LAURA.4.1

HOMER=$HOME/$VERSION

rm mAkEblk exchange.f 2> /dev/null

ln -s $HOMER/mAkEblk .

mAkEblk

rm mAkEblk

if [ ! -f exchange.inc ]; then

        mv exchange.tmp exchange.inc
        ln -s $HOMER/exchange.f .
        f77 exchange.f -o ExchAngE
        rm exchange.f

else

        diff exchange.tmp exchange.inc > /dev/null 2>&1

        if [ $? -ne 0 ]; then   # if files differ...

                mv exchange.tmp exchange.inc
                ln -s $HOMER/exchange.f .
                f77 exchange.f -o ExchAngE
                rm exchange.f

        else

                rm exchange.tmp

        fi

fi
```

Screen 110.

```
if [ "`grep machine $HOMER/DEFAULTS | awk '{print $1}'`" = "0" ]; then
        ExchAngE > errout 2>&1
        if [ "`wc -l errout | awk '{print $1}'`" = "1" ]; then
                echo "  ABORT:  Job is too large to run interactively on this ma
chine."
                echo "          \'ExchAngE' must be submitted as a batch job."
        else
                cat echout
                echo "          ...Completed \'ExchAngE' procedure"
        fi
        rm echout 2> /dev/null
else
        ExchAngE
        echo "          ...Completed \'ExchAngE' procedure"
fi

for FILE in TWALL.in TWALL.MASTER; do
        if [ "`wc -c $FILE | awk '{print $1}'`" = "0" ]; then
                rm $FILE
        fi
done
```

Screen 110. Concluded.

Appendix H

Structure of CUSTOMIZE

As discussed in chapter 8, the philosophy behind the file directory structure of LAURA is that the bulk of the LAURA algorithm can be packaged such that the constituent files do not change from one application to the next. These files reside in the user's "$HOME/LAURA.4.1" directory. For advanced applications, LOCAL copies of these files can be created (appendix K) and modified. Subsequent compilations of LAURA will use these local files rather than the $HOME/LAURA.4.1 versions.

A LOCAL version of file lfn will only be used in the current working directory. There may be occasions when the user would like to use this tailored file in directories other than the current one. The obvious approach is to copy the LOCAL file to the other directory. Rather than copying a LOCAL file from directory to directory, however, the command

```
CUSTOMIZE lfn
```

can be used to place this LOCAL file in the $HOME/LAURA.4.1/CUSTOM directory. Any future compilations of LAURA (from any working directory on this machine) will use this $HOME/LAURA.4.1/CUSTOM file in lieu of the $HOME/LAURA.4.1 version.

Several steps must be performed by this command to create a CUSTOM file. First, the write-protection for "$HOME/LAURA.4.1" must be removed, and the subdirectory CUSTOM created as follows (screen 111):

```
                              VERSION=LAURA.4.1

FILE=$1

chmod 700 $HOME/$VERSION                # make CUSTOM read/write/execute

if [ ! -d $HOME/$VERSION/CUSTOM ]; then # if subdirectory CUSTOM DNE...
        mkdir $HOME/$VERSION/CUSTOM;            # ...create it
fi                                      #

chmod 700 $HOME/$VERSION/CUSTOM         # make CUSTOM read/write/execute
```

Screen 111.

Next, the LOCAL file is moved to CUSTOM. However, if a CUSTOM version of lfn already exists,

and the two files are identical, the LOCAL file is simply removed. On the other hand, if a CUSTOM version of lfn already exists, and the two files differ, the user is given the option to update the CUSTOM file (screen 112):

```
                                        # if file DNE in CUSTOM...
if [ ! -f $HOME/$VERSION/CUSTOM/$FILE ]; then

        mv $FILE $HOME/$VERSION/CUSTOM          # move it there, &
        chmod 400 $HOME/$VERSION/CUSTOM/$FILE   # make it read only

        echo "File \'$FILE' added to \$HOME/$VERSION/CUSTOM."
        echo "Future compilations will use this file in lieu"
        echo "of the \$HOME/$VERSION version of \'$FILE'."

else                                    #

        diff $FILE $HOME/$VERSION/CUSTOM/$FILE > /dev/null 2>&1

        if [ $? -ne 0 ]; then                   # if files differ...

                echo "File \'$FILE' already exists"
                echo "in \$HOME/$VERSION/CUSTOM. Update (y/n)?"
                read ANSWER

                if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then

                        mv -f $FILE $HOME/$VERSION/CUSTOM       # move to CUSTOM
                        chmod 400 $HOME/$VERSION/CUSTOM/$FILE   # make read only

                        echo "File \'$FILE' updated in \$HOME/$VERSION/CUSTOM."
                        echo "Future compilations will use this in lieu "
                        echo "of the \$HOME/$VERSION version of \'$FILE'."

                fi                                      #

        else                                    # else...

                rm -f $FILE                             # ...remove it
                echo " No change in $FILE in \$HOME/$VERSION/CUSTOM"

        fi                                      #

fi                                      #
```

Screen 112.

The final step is to reestablish write protection for the "$HOME/LAURA.4.1" files as follows:

```
chmod 500 $HOME/$VERSION/CUSTOM         # make read/execute
chmod 500 $HOME/$VERSION                # make read/execute
```

Appendix I

Structure of INITIALIZE

As discussed in section 9.1.1, the LAURA restart file (`RESTART.in`) consists of flow field properties (velocities, temperatures, and densities) for each cell, along with the grid. The `INITIALIZE` script allows the user to use an externally generated grid with LAURA. This utility creates a `RESTART.in` file from a grid file in `PLOT3D` format. The command

```
INITIALIZE lfn
```

takes a grid file `lfn`, initializes its flow field to free-stream values, and outputs the file `RESTART.in`.

> **NOTE:** `INITIALIZE` gleans information from several files that are created by `stArt`. Therefore, `PRELUDE` must be executed before an externally generated grid can be initialized.

In `laura`, the grid must be oriented such that $y = 0$ is the plane of symmetry. The user can encounter externally generated grids with other orientations. Before running `INITIALIZE`, it is recommended that the user reorient the grid to conform to the examples shown in figures 2.1 and 2.2. An alternative involves changing the definition of *uinf*, *vinf*, and *winf* for `laura` and `INITIALIZE` (section 7.9.1).

`INITIALIZE` uses the following procedure. First, it checks to see if the file `RESTART.in` exists. If so, the user is given the option to overwrite it as follows (screen 113):

```
                                VERSION=LAURA.4.1

GRIDIN=$1

if [ -f $GRIDIN ]; then

        if [ -f RESTART.in ]; then
                echo "  File \'RESTART.in' already exists."
                echo "          Do you wish to remove it (y/n) {n}?"
                read ANSWER
                if [ "$ANSWER" = "n" -o "$ANSWER" = "N" ]; then
                        exit 0
                fi
        fi

else
        echo "  ERROR: File \'$GRIDIN' not found."
        exit 0
fi
```

Screen 113.

The species indices are obtained from file `species_strt` (which is created by `stArt`). If a `LOCAL` version of `species_strt` exists, that file is used. This information is placed in file `flowinit.in`, as shown in screen 114:

```
HOMER=$HOME/$VERSION

rm flowinit.in 2> /dev/null

if [ "`ls STRTfiles/species.strt species.strt`" ]; then

        if [ ! -f species.strt ]; then
                ln -s STRTfiles/species.strt .
        fi

        grep "=" species.strt | awk -F= 'print $2' > flowinit.in

        if [ "`ls -lt species.strt | grep STRTfiles`" ]; then
                rm species.strt
        fi

else
        echo "ERROR:  File \'species.strt' not found."
fi
```

Screen 114.

The number of species is obtained from file **parameter_strt** (which is created by **stArt**). If a **LOCAL** version of **parameter_strt** exists, that file is used. This value is added to the file **flowinit.in** (screen 115):

```
if [ "`ls STRTfiles/parameter.strt parameter.strt`" ]; then

        if [ ! -f parameter.strt ]; then
                ln -s STRTfiles/parameter.strt .
        fi

        grep "ns =" $FILE | awk -F= 'print $2' >> flowinit.in

        if [ "`ls -lt parameter.strt | grep STRTfiles`" ]; then
                rm parameter.strt
        fi

else

        echo "ERROR:  File \'parameter.strt' not found."

fi
```

Screen 115.

The angle of attack, angle of yaw, free-stream temperature, and free-stream velocity are obtained from file **DEFAULTS** (which is updated by **stArt**). This information is added to file **flowinit.in** (screen 116):

```
if [ -f DEFAULTS ]; then

        grep attack     DEFAULTS >> flowinit.in
        grep yaw        DEFAULTS >> flowinit.in
        grep tinf       DEFAULTS >> flowinit.in
        grep vinfb      DEFAULTS >> flowinit.in

else

        echo "ERROR:  File \'DEFAULTS' not found."

fi
```

Screen 116.

The program **ArrAy** reads the prescribed computational block dimensions from file **lfn**. If a larger dimension is required than that specified in the **$HOME/LAURA.4.1** version of **flowinit.inc**, a **LOCAL** version of **flowinit.inc** is created, and **flowinit.f** is recompiled. Next, **flOwInIt** is executed to initialize the grid based on the values in **flowinit.in**. The resultant flow field is output to create **RESTART.in** (screen 117):

```
rm grid.in 2> /dev/null

ln -s $GRIDIN grid.in

$HOME/$VERSION/ArrAy    # create LOCAL flowinit.inc, if necessary.

if [ "`ls flowinit.* 2> /dev/null`" ]; then
        echo "  <               LOCAL \`flOwInIt' executable will be used
       >"
        if [ "`ls -t fl* | awk '{print $1}' | head -1`" = "flOwInIt" ]; then
                echo "  <               LOCAL \`flOwInIt' executable is up to da
te                    >"
        else
                echo "  <               Creating LOCAL \`flOwInIt' executable
               >"
                if [ -d $HOMER/CUSTOM ]; then
                        for FILE in `cd $HOMER/CUSTOM; ls flowinit.* 2> /dev/
null`; do
                                if [ ! -f $FILE ]; then
                                        ln -s $HOMER/CUSTOM/$FILE .
                                fi
                        done
                fi
                for FILE in `cd $HOMER; ls flowinit.*`; do
                        if [ ! -f $FILE ]; then
                                ln -s $HOMER/$FILE .
                        fi
                done
                FC flowinit.f -o flOwInIt
                for FILE in `ls flowinit.*`; do
                        if [ "`ls -lt $FILE | grep LAURA`" ]; then
                                rm $FILE
                        fi
                done
        fi
else
        rm flOwInIt 2> /dev/null
fi
```

Screen 117.

```
if [ ! -f flOwInIt ]; then
        if [ -d $HOMER/CUSTOM ]; then
                if [ "`cd $HOMER/CUSTOM; ls flowinit.* 2> /dev/null`" ]; then
                        echo "  <               CUSTOM \`flOwInIt' executable wi
ll be used                      >"
                        ( cd $HOMER/CUSTOM;
                        rm *trace 2> /dev/null;
                        if [ "`ls -t fl* | awk '{print $1}' | head -1`" = "flOwI
nIt" ]; then
                                echo "  <               CUSTOM \`flOwInIt' execu
table is up to date             >";
                        else
                                echo "  <               Creating CUSTOM \`flOwIn
It' executable                  >";
                                chmod 700 $HOMER;
                                chmod 700 $HOMER/CUSTOM;
                                for FILE in `cd $HOMER; ls flowinit.*`; do
                                        if [ ! -f $FILE ]; then
                                                ln -s $HOMER/$FILE .;
                                        fi
                                done;
                                FC flowinit.f -o flOwInIt;
                                for FILE in `ls flowinit.*`; do
                                        if [ "`ls -lt $FILE | grep LAURA`" ]; then
                                                rm $FILE;
                                        fi
                                done
                                chmod 500 $HOMER;
                                chmod 500 $HOMER/CUSTOM;
                        fi )
                        ln -s $HOMER/CUSTOM/flOwInIt .
                else
                        rm $HOMER/CUSTOM/flOwInIt 2> /dev/null
                        if [ !  "`ls $HOMER/CUSTOM`" ]; then
                                chmod 700 $HOMER
                                chmod 700 $HOMER/CUSTOM
                                rmdir $HOMER/CUSTOM
                                chmod 500 $HOMER
                        fi
                        ln -s $HOMER/flOwInIt .
                fi
        else
                ln -s $HOMER/flOwInIt .
        fi
fi
```

Screen 117. Continued.

```
flOwInIt

if [ "`ls -lt flOwInIt | grep LAURA`" ]; then   # Remove flOwInIt if
        rm flOwInIt                             # it's just a sym-link
fi                                              # to default flOwInIt.

rm grid.in 2> /dev/null

exit 0
```

Screen 117. Concluded.

Appendix J

Structure of KEEPER

This script allows the user to create backups of `RESTART.in` and `TWALL.in` by simply typing the command

`KEEPER`

Backups for the master files (`RESTART.MASTER` and `TWALL.MASTER`), which exist when multiple computational blocks are employed in the solution procedure, are also created. If a previous backup file is encountered, the user is given the option of updating it (screen 118):

```
BACKUP='backup'
echo "  Enter desired suffix for these backup files {$BACKUP}:"
read BACKUP

for FILE `ls in RESTART.in TWALL.in *.MASTER 2> /dev/null`; do

        if [ -f $FILE.$BACKUP ]; then

                echo "LOCAL file \`$FILE.$BACKUP' already exists.  Update (y/n) {
n}?"
                read ANSWER
                if [ "$ANSWER" = "y"  -o  "$ANSWER" = "Y" ]; then
                        rm $FILE.$BACKUP
                        cp $FILE $FILE.$BACKUP
                        echo "File \`$FILE' copied to \`$FILE.$BACKUP'"
                fi

        else

                cp $FILE $FILE.$BACKUP
                echo "File \`$FILE' copied to \`$FILE.$BACKUP'"
        fi

fi
```

Screen 118.

NOTE: These LAURA restart files (`RESTART.in` and `TWALL.in`) are only overwritten at the conclusion of a successful run, so "backing up" these files is not mandatory. In some cases, however, doing so can provide peace of mind for the user.

Appendix K

Structure of LOCALIZE

As discussed in chapter 8, the philosophy behind the file directory structure of LAURA is that the bulk of the LAURA algorithm can be packaged such that the constituent files do not change from one application to the next. These files reside in the user's `$HOME/LAURA.4.1` directory. For advanced applications, `LOCAL` copies of these files can be created with the command

```
LOCALIZE lfn
```

and modified. Subsequent compilations of LAURA will use these `LOCAL` files rather than the `$HOME/LAURA.4.1` versions.

First, `LOCALIZE` checks to see if a `LOCAL lfn` already exists. If so, the user has the option to abort the request or continue (and overwrite the existing `LOCAL` file) as shown in screen 119:

```
                                VERSION=LAURA.4.1

FILE=$1

if [ -f $FILE ]; then
        echo "  LOCAL file \'$FILE' already exists.  Overwrite (y/n) {n}?"
        read ANSWER
else
        ANSWER='y'
fi
```

Screen 119.

If the request is not aborted, `LOCALIZE` checks the following directories for the existence of `lfn`:

- `$HOME/LAURA.4.1`—the directory containing baseline files (`.F`, `.FOR`, and `.inc` suffixes)
- `STRTfiles`—the directory containing files created by `stArt(.strt` suffixes)
- `FORTRAN`—the directory containing pure-`FORTRAN` files (`.f` suffixes); this directory, and the files it contains, are created by the command

```
make fortran
```

which is discussed in section 8.3.

`LOCALIZE` copies the file `lfn` from its directory of residence and makes it user-writable (screen 120):

```
EXIST=0

if [ -f $HOME/$VERSION/$FILE ]; then

        if [ -d $HOME/$VERSION/CUSTOM ]; then
                if [ -f $HOME/$VERSION/CUSTOM/$FILE ]; then
                        echo "  WARNING: CUSTOM version of file \'$FILE' exists."
                fi
        fi

        EXIST=1

        cp $HOME/$VERSION/$FILE .       # copy installed version
                                        # to LOCAL directory,
        chmod 600 $FILE                 # and make it user-writable

fi

if [ -f STRTfiles/$FILE ]; then

        EXIST=1

        cp STRTfiles/$FILE .            # copy file from STRTfiles
                                        # to LOCAL directory,
        chmod 600 $FILE                 # and make it user-writable

fi

if [ -f FORTRAN/$FILE ]; then

        EXIST=1

        cp FORTRAN/$FILE .      # copy file from FORTRAN to LOCAL dir

fi
```

Screen 120.

With any future compilations of LAURA (from this working directory), this `LOCAL` file will be used in lieu of the `$HOME/LAURA.4.1` or `CUSTOM` (appendix H) versions.

If file `lfn` does not exist in any of these directories, the following error message is sent to the screen, and the request is aborted (screen 121):

```
if [ "$EXIST" = "0" ]; then

        echo "  ERROR:  File \'$FILE' does not exist in the "
        echo "        \$HOME/$VERSION, STRTfiles, or FORTRAN directories."

else
        echo "  LOCAL copy of the \$HOME/$VERSION version of \'$FILE' created."
        echo "  Future compilations will use this LOCAL file in lieu of the"
        echo "  \$HOME/$VERSION or CUSTOM versions of \'$FILE'."
fi
```

Screen 121.

Appendix L

Structure of RESTORE

The script RESTORE allows the user to reconstruct a working directory from a file `lfn` created with the ARCHIVE command (appendix F). First, create a new working directory. Next, move `lfn` (the `tarfile` created by ARCHIVE) to that directory. While in the new directory, type the command

`RESTORE lfn`

and the following procedure will be executed. The tarfile is unloaded as shown in screen 122:

```
TARFILE=$1

tar xf $TARFILE
```

Screen 122.

The DEFAULTS and INPUTS file are retrieved (screen 123):

```
if [ -d IN ]; then
        for FILE in `ls -t IN`; do
                mv IN/$FILE .
        done
        rmdir IN
fi
```

Screen 123.

The file `conv.out` is retrieved (screen 124):

```
if [ -d CONV ]; then
        mv CONV/conv.out .
        rmdir CONV
fi
```

Screen 124.

Any input files for `laura` (`RESTART.in`, `assign_tasks`, `transition`, `TWALL.in`, and `variabletw`) that were saved are retrieved (screen 125):

```
if [ -d CONTROL ]; then
        for FILE in 'ls -t CONTROL'; do
                mv CONTROL/$FILE .
        done
        rmdir CONTROL
fi
```

Screen 125.

If the master input files (`RESTART.MASTER` and `TWALL.MASTER`) were saved, they are restored (screen 126):

```
if [ -d MASTER ]; then
        for FILE in 'ls -t MASTER'; do
                mv MASTER/$FILE .
        done
        rmdir MASTER
fi
```

Screen 126.

If any LOCAL files were saved, they are reinstated (screen 127):

```
if [ -d LOCAL ]; then
        for FILE in `ls -t LOCAL`; do
                mv LOCAL/$FILE .
        done
        rmdir LOCAL
fi
```

Screen 127.

NOTE: If any CUSTOM files were saved, they are saved in a CUSTOM subdirectory below this LOCAL directory. They are not automatically reinstated in the $HOME/LAURA.4.1/CUSTOM directory to avoid the possibility of overwriting existing CUSTOM files. To restore them as CUSTOM files, the user should cd to CUSTOM and use the CUSTOMIZE command (appendix H) on the individual files.

The following steps are required to pick up the solution where it left off before archival:

- Type the command

  PRELUDE INPUTS

  to create the required subdirectories and source files (.strt suffixes).

  NOTE: Be sure to preserve the existing RESTART.in, data, and conv.out files when prompted.

- Type the command

  make

  to compile the LAURA source code and create the laura executable.

- Type the command

  laura < data

  to advance the solution.

Appendix M

Structure of SIZEIT

This script allows the user to estimate the memory requirements for the current application, by simply typing the command

SIZEIT

Both PRELUDE and Makefile execute this script as part of their procedures.

The script SIZEIT serves as the front end for sIzEIt, which is the executable for the FORTRAN routine sizeit.f. This routine solves equation (11.8), as well as equation (11.9) or (11.10), to determine memory allocations for each task defined in file assign_tasks. If file assign_tasks is not found, a single task with sweeping in the k-direction is assumed.

First, the size of the laura executable is determined (excluding the overhead associated with the use of multitasking and/or the solid-state-device), as shown in screen 128:

```
if [ -f laura ]; then
        if [ "`ls -t STRTfiles/*.strt *.strt *.inc *.F* *.f laura 2> /dev/null |
awk '{ print $1}' | head -1`" = "laura" ]; then
                size laura | tail -1 | awk '{print $4}' > sizeit.in
        else
                echo "0" > sizeit.in
        fi
else
        echo "0" > sizeit.in
fi
```

Screen 128.

Next, the information required by **sIzEIt** is gathered from files created by **PRELUDE**. Then **sIzEIt** is executed to determine any additional memory requirements for the current job. The results are saved in file **ECHOSIZE** and echoed to the screen (screen 129):

```
if [ "`ls -t STRTfiles/parameter.strt parameter.strt 2> /dev/null`" ]; then

        if [ ! -f DEFAULTS ]; then
                ln -s $HOME/$VERSION/DEFAULTS .
        fi

        if [ !  "`ls HEADER.strt 2> /dev/null`" ]; then
                ln -s STRTfiles/HEADER.strt .
        fi

        if [ "`grep NAVIER HEADER.strt`" ]; then
                echo "2" >> sizeit.in
        else
                echo "0" >> sizeit.in
        fi

        if [ "`ls -lt HEADER.strt | grep STRTfiles`" ]; then
                rm HEADER.strt
        fi

        if [ "`grep machine DEFAULTS | awk '{print $1}'`" = "0" ]; then

                ln -s $HOME/$VERSION/eval_param.f .

                for FILE in issd_assn.strt parameter.strt; do
                        if [ ! -f $FILE ]; then
                                ln -s STRTfiles/$FILE .
                        fi
                done

                FC eval_param.f -o EvAl_pArAm 2> /dev/null

                for FILE in issd_assn.strt parameter.strt; do
                        if [ "`ls -lt $FILE | grep STRTfiles`" ]; then
                                rm $FILE
                        fi
                done

                EvAl_pArAm >> sizeit.in
                rm eval_param.f EvAl_pArAm

      fi
```

Screen 129.

```
        rm -f ECHOSIZE 2> /dev/null

        $HOME/$VERSION/sIzEIt

        if [ "`wc -l ECHOSIZE | awk '{print $1}'`" = "0" ]; then
                rm ECHOSIZE             # rm ECHOSIZE if it is empty
        else
                chmod 400 ECHOSIZE
                cat ECHOSIZE
        fi

        rm sizeit.in

        if [ "`ls -lt DEFAULTS | grep LAURA`" ]; then   # Remove DEFAULTS if
                rm DEFAULTS                             # it's just a sym-link
        fi                                              # to default DEFAULTS.

else

        echo "ERROR:  File \'parameter.strt' not found."

fi
```

Screen 129. Concluded.

Appendix N

Structure of XCUSTOM

The command

XCUSTOM lfn

nullifies the

CUSTOMIZE lfn

command. In other words, it moves the file `lfn` from the `CUSTOM` directory (appendix H) to the `LOCAL` directory. This command performs the following tasks. First, the write-protection for `$HOME/LAURA.4.1`, the subdirectory `CUSTOM`, and the file `lfn` must be removed. Then the file itself is moved to the `LOCAL` directory. The final step is to reestablish write protection for the `$HOME/LAURA.4.1` files (screen 130):

```
FILE=$1

if [ -f $HOME/$VERSION/CUSTOM/$FILE ]; then

        chmod 700 $HOME/$VERSION                # make dir  read/write/execute
        chmod 700 $HOME/$VERSION/CUSTOM         # make dir  read/write/execute
        chmod 600 $HOME/$VERSION/CUSTOM/$FILE   # make file read/write

        mv $HOME/$VERSION/CUSTOM/$FILE .        # move file to LOCAL directory
        echo "CUSTOM \'$FILE' moved to LOCAL directory."
        echo "Unless removed, future compilations will use this
        echo "LOCAL version of \'$FILE'."
        if [ ! "`ls -t $HOME/$VERSION/CUSTOM`" ]; then  # if dir is now empty...
                rmdir $HOME/$VERSION/CUSTOM             # ...remove it
        else                                            # else...
                chmod 500 $HOME/$VERSION/CUSTOM         # ...make read/execute
        fi                                              #
        chmod 500 $HOME/$VERSION                # make dir read/execute
else
        echo "File \'$FILE' does not exist in \$HOME/$VERSION/CUSTOM."
fi
```

Screen 130.

Appendix O

LAURA Algorithm

Nomenclature

Note that boldface, lowercase symbols refer to vectors in parameter space. Boldface, uppercase symbols refer to matrices in parameter space. An arrow over a lowercase symbol refers to vectors in physical space, with (x, y, z)-coordinates. With the exception of the reaction rate coefficients and temperatures, all variables are nondimensional.

a	frozen sound speed, nondimensionalized by $\mathcal{V}_\infty$
$\mathbf{A}$	Jacobian matrix of $\mathbf{g}$ with respect to $\mathbf{q}$
$\mathbf{B}$	Jacobian matrix of $\mathbf{h}$ with respect to $\mathbf{q}$
$C_{v,tr}$	specific heat for translational-rotational energy
$C_{v,V}$	specific heat for vibrational-electronic energy
c_s	mass fraction of species s
c_s	average molecular speed of molecule s
D_s	effective diffusion coefficient of species s
E	total energy per unit mass of mixture, nondimensionalized by $\mathcal{V}_\infty^2$
e_s	energy per unit mass of species s
e_V	mixture vibrational-electronic energy per unit mass
$e_{V,s}$	vibrational-electronic energy per unit mass of species s
$\vec{\mathbf{f}}$	flux vector in Cartesian space
$\mathbf{g}$	inviscid component of flux vector relative to cell face
H	total enthalpy per unit mass of mixture
$\mathbf{h}$	viscous component of flux vector relative to cell face
h	enthalpy
h_s	enthalpy per unit mass of species s
$h_{V,s}$	vibrational-electronic enthalpy per unit mass of species s
$k_{b,r}$	backward reaction rate coefficient for reaction r
$k_{f,r}$	forward reaction rate coefficient for reaction r
$\vec{l}$	unit vector tangent to computational cell wall ($\vec{l} \cdot \vec{m} = 0$)
l_x, l_y, l_z	components of $\vec{l}$ in x-, y-, z-directions, respectively
$\mathbf{M}_L$	point-implicit Jacobian of flux terms
$\mathbf{M}_{L,\mathrm{INV}}$	point-implicit Jacobian of inviscid terms
$\mathbf{M}_{L,\mathrm{VIS}}$	point-implicit Jacobian of viscous terms
$\mathbf{M}_{L,\mathrm{SRC}}$	point-implicit Jacobian of source terms
M_s	molecular weight of species s
$\vec{m}$	unit vector tangent to computational cell wall ($\vec{l} \cdot \vec{m} = 0$)

m_x, m_y, m_z	components of $\vec{m}$ in x-, y-, z-directions, respectively
N	iteration level index
N_r	number of reactions
N_s	number of species
n	normal distance
$\vec{n}$	unit vector normal to computational cell wall ($\vec{n} \cdot \vec{l} = \vec{n} \cdot \vec{m} = 0$)
n_x, n_y, n_z	components of $\vec{n}$ in x-, y-, z-directions, respectively
$\tilde{n}$	number density
njcobian	number of iterations between Jacobian updates
ntrnsprt	number of iterations between transport property updates
p	pressure
Q_{rad}	divergence of radiative flux
$\mathbf{q}$	vector of conserved variables
$\bar{R}$	universal gas constant
$R_{b,r}$	backward reaction rate for reaction r
$R_{f,r}$	forward reaction rate for reaction r
$\mathbf{R}$	matrix of left eigenvectors of $\mathbf{A}$
$\mathbf{r}$	right-hand-side residual vector
r	reaction rate
rf_i	relaxation factor used with inviscid Jacobian matrices
rf_v	relaxation factor used with viscous Jacobian matrices
s	arc length
s_l, s_m, s_n	arc lengths in $\vec{l}$-, $\vec{m}$-, $\vec{n}$-directions, respectively
T	translational-rotational temperature
T_V	vibrational-electron-electronic excitation temperature
t	time
U, V, W	velocity component in the $\vec{n}$-, $\vec{l}$-, and $\vec{m}$-directions, respectively
u, v, w	velocity components in x-, y-, z-directions, respectively
$\mathcal{V}_\infty$	free-stream total velocity, m/s
x, y, z	Cartesian coordinates
y_s	mole fraction of species s
$\alpha_{s,r}$	stoichiometric coefficient for reactants in reaction r
β	In general, $\beta = \partial p / \partial (\rho E)$; for perfect gas, this reduces to $\beta = \gamma - 1$
$\beta_{s,r}$	stoichiometric coefficient for products in reaction r
γ	ratio of specific heats
$\tilde{\gamma}_r$	$\partial p / \partial \rho_r$
ϵ, ϵ_o	parameters for defining minimum eigenvalue
η	frozen thermal conductivity for translational-rotational energy
η_V	frozen thermal conductivity for vibrational-electronic energy
θ	0 for first-order; 1 for second-order approximation to inviscid flux
λ	eigenvalue of $\mathbf{A}$
$\tilde{\lambda}$	restricted eigenvalue of $\mathbf{A}$
$\mathbf{\Lambda}$	diagonal matrix of eigenvalues of $\mathbf{A}$
μ	mixture viscosity
μ_{sj}	reduced mass of species s and j
ξ, η, ζ	computational coordinates
ρ	mixture density
ρ_s	density of species s

σ	cell face area
σ_s	cross section of species s for translational-vibrational energy exchange
τ	shear stress
$< \tau_s >$	vibrational relaxation time defined in equation (O.56)
$\bar{\tau}_V$	relaxation time defined in equation (O.53)
ϕ	$\partial p / \partial (\rho e_V)$
λ	dummy variable for ξ, η, or ζ
$\dot{\omega}$	vector of source terms
$\dot{\omega}_s$	mass production rate of species s per unit volume
$\dot{\omega}_V$	vibrational-electronic energy source term
Ω	cell volume

Subscripts:

e	electron
I, J, K	indices of cell centers in ξ-, η-, ζ-directions, respectively
i, j, k	indices of cell walls in ξ-, η-, ζ-directions, respectively
L	dummy index for cell center
l	dummy index for cell wall
r	reaction r or species r
s	species s
V	vibrational-electronic

Superscripts:

n	iteration level index
$'$	relating to the thin-layer approximation

The following algorithm description is substantially the same as that provided in reference 32. Updates are provided as appropriate for LAURA.

O.1. Finite-Volume Fundamentals

The integral form of the conservation laws applied to a single cell in the computational domain is written

$$\int\int\int \frac{\partial \mathbf{q}}{\partial t}\, d\Omega + \int\int \vec{\mathbf{f}} \cdot \vec{n}\, d\sigma = \int\int\int \dot{\omega}\, d\Omega \tag{O.1}$$

In equation (O.1) the first term describes the time rate of change of conserved quantity $\mathbf{q}$ in the control volume; the second term describes convective and dissipative flux $\vec{\mathbf{f}}$ through the cell walls; and the third term accounts for sources or sinks of conserved quantities within the control volume. The third term is identically zero for perfect-gas flows, but it is required for flows in chemical or thermal nonequilibrium.

The finite-volume approximation to equation (O.1) for a general, unstructured grid is written

$$\left[\frac{\delta (\mathbf{q}\Omega)}{\delta t}\right]_L + \sum_{m=1}^{M_L} \vec{\mathbf{f}}_m \cdot \vec{n}_m \sigma_m = [\dot{\omega}\Omega]_L \tag{O.2}$$

where

$$\delta \mathbf{q} = \mathbf{q}^{n+1} - \mathbf{q}^{n} \qquad \text{and} \qquad \delta t = t^{n+1} - t^{n}$$

and Ω_L is constant with respect to time. Further, M_L is the number of faces of cell L having volume Ω, and subscript m refers to cell face m with surface area σ_m. The quantity $\vec{n}_m$ is a unit vector normal to cell face m in a direction facing away from the cell center. The dependent variable $\mathbf{q}$ is defined at cell centers. The independent variables x, y, and z are defined at cell corners.

The finite-volume approximation to equation (O.1) for a rectangularly ordered, structured grid is written

$$\begin{aligned} \left[\frac{\delta \mathbf{q}\Omega}{\delta t}\right]_{I,J,K} &+ \left[\vec{\mathbf{f}}_{i+1} \cdot \vec{n}_{i+1}\sigma_{i+1} - \vec{\mathbf{f}}_{i} \cdot \vec{n}_{i}\sigma_{i}\right]_{J,K} \\ &+ \left[\vec{\mathbf{f}}_{j+1} \cdot \vec{n}_{j+1}\sigma_{j+1} - \vec{\mathbf{f}}_{j} \cdot \vec{n}_{j}\sigma_{j}\right]_{I,K} \\ &+ \left[\vec{\mathbf{f}}_{k+1} \cdot \vec{n}_{k+1}\sigma_{k+1} - \vec{\mathbf{f}}_{k} \cdot \vec{n}_{k}\sigma_{k}\right]_{I,J} = [\dot{\omega}\Omega]_{I,J,K} \end{aligned} \tag{O.3}$$

A shorthand notation for equation (O.3) that will be used throughout this paper follows:

$$\left[\frac{\delta \mathbf{q}\Omega}{\delta t}\right]_{L} + \sum_{l=i,j,k} \left[\vec{\mathbf{f}}_{l+1} \cdot \vec{n}_{l+1}\sigma_{l+1} - \vec{\mathbf{f}}_{l} \cdot \vec{n}_{l}\sigma_{l}\right] = [\dot{\omega}\Omega]_{L} \tag{O.4}$$

Note in equations (O.2) to (O.4) that the uppercase integer variables I, J, K, and L denote computational coordinates at the cell centers, and the lowercase integer variables i, j, k, l, and m denote the cell faces or the cell corners. For example, $\sigma_{i,J,K}$ refers to the cell wall corresponding to indices $I - \frac{1}{2}, J, K$ (fig. O.1). In the shorthand notation of equation (O.4), the integer variable l is used as a generic index for i, j, or k. This notation is convenient because most of the formulations for quantities at the cell faces are independent of the coordinate direction. The geometric quantities Ω, σ, and $\vec{n}$ are easily derived given the Cartesian coordinates of the cell corners. Details are found in appendix A of reference 2.

The formulations that follow are based on a rectangularly ordered, structured grid. A first-order-accurate formulation of the inviscid equations on a structured grid is identical to the formulation on an unstructured grid. The modifications required to achieve second-order accuracy on an unstructured grid are not addressed in this paper. However, note that the formulations for obtaining second-order accuracy only involve modifications to the right-hand-side residual vector. The point-implicit relaxation procedure that will be defined by the formulation of the left-hand-side matrix is independent of grid structure. Consequently, much of the development that follows will carry over to unstructured grid formulations, as in the paper by Thareja et al. (ref. 33).

O.2. Conservation Equations

The inviscid, viscous, and source term contributions to the complete conservation laws are considered separately for convenience. Let

$$\vec{\mathbf{f}}_{l} \cdot \vec{n}_{l} = \mathbf{g}_{l} + \mathbf{h}_{l} \tag{O.5}$$

where $\mathbf{g}_l$ defines the inviscid terms and $\mathbf{h}_l$ defines the viscous terms. The finite-volume formulation of the conservation laws is now expressed as

$$\left[\frac{\delta \mathbf{q}\Omega}{\delta t}\right]_{L} + \sum_{l=i,j,k} \left[\mathbf{g}_{l+1}\sigma_{l+1} - \mathbf{g}_{l}\sigma_{l}\right] + \sum_{l=i,j,k} \left[\mathbf{h}_{l+1}\sigma_{l+1} - \mathbf{h}_{l}\sigma_{l}\right] = [\dot{\omega}\Omega]_{L} \tag{O.6}$$

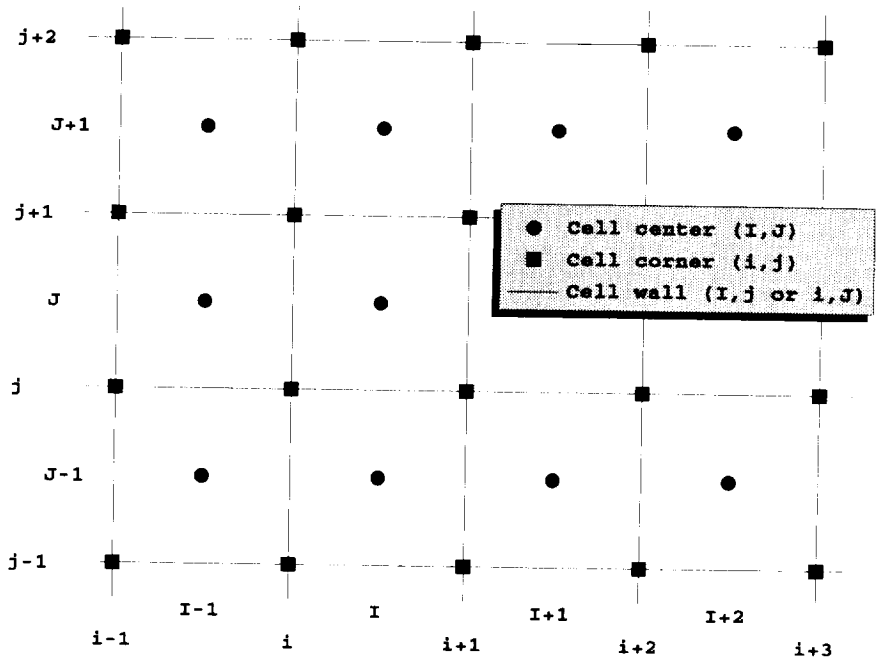


Figure O.1. Cell indexing system with cell corners defined by lowercase letters and cell centers defined by uppercase letters.

In the case of a reacting gas flow in which thermal nonequilibrium is modeled using a two-temperature approximation, the vectors $\mathbf{q}$, $\mathbf{g}$, $\mathbf{h}$, and $\dot{\omega}$ are defined as

$$\mathbf{q} = \begin{bmatrix} \rho_s \\ \rho u \\ \rho v \\ \rho w \\ \rho E \\ \rho e_V \end{bmatrix} \tag{O.7}$$

$$\mathbf{g} = \begin{bmatrix} \rho_s U \\ \rho U u + p n_x \\ \rho U v + p n_y \\ \rho U w + p n_z \\ \rho U H \\ \rho U e_V \end{bmatrix} \tag{O.8}$$

Table O.1. Species Indices

s	1	2	3	4	5	6	7	8	9	10	11
species	N	O	N_2	O_2	NO	N^+	O^+	N_2^+	O_2^+	NO^+	e^-

$$\mathbf{h} = \begin{bmatrix} -\rho D_s \dfrac{\partial y_s}{\partial s_n} \\ -\tau_{nx} \\ -\tau_{ny} \\ -\tau_{nz} \\ -u\tau_{nx} - v\tau_{ny} - w\tau_{nz} - \eta \dfrac{\partial T}{\partial s_n} - \eta_V \dfrac{\partial T_V}{\partial s_n} - \rho \displaystyle\sum_{s=1}^{N_s} h_s D_s \dfrac{\partial y_s}{\partial s_n} \\ -\eta_V \dfrac{\partial T_V}{\partial s_n} - \rho \displaystyle\sum_{s=1}^{N_s} h_{V,s} D_s \dfrac{\partial y_s}{\partial s_n} \end{bmatrix} \tag{O.9}$$

$$\dot{\omega} = \begin{bmatrix} \dot{\omega}_s \\ 0 \\ 0 \\ 0 \\ Q_{\text{rad}} \\ \dot{\omega}_V \end{bmatrix} \tag{O.10}$$

The first element of the vectors defined in equations (O.7) to (O.10) describes the species conservation; the next three elements describe x, y, and z momentum conservation; the fifth element describes total energy conservation; and the sixth element describes vibrational-electronic energy conservation. The present model considers the $N_s = 11$ species shown in table O.1.

Consequently, the vectors defined in equations (O.7) to (O.10) are composed of a total of 16 elements. Implicit in the use of a vibrational-electronic energy equation is the assumption that the partition of energy in the vibrational, bound electronic, and free-electron modes among all species can be described by a single temperature T_V. This approximation is based on rapid equilibration of vibrational and electronic energy and electron translational modes (refs. 6 and 34). The translational and rotational energy modes of heavy particles are assumed to be fully excited and described by temperature T.

The thermochemical nonequilibrium model is described in detail in reference 4. Some specifics on its formulation are given in a later section, but a brief overview is given below. The reactive source terms for the species conservation equations are denoted by $\dot{\omega}_s$. The radiative energy transport term Q_{rad} can be treated as a source term in the total energy equation. Although its effects are not included in the baseline LAURA algorithm, one approach for including it is discussed in section 11.6. Finally, the vibrational-electronic energy source term $\dot{\omega}_V$ accounts for the mechanisms by which vibrational-electronic energy is lost or gained caused by collisions among particles in the cell. These mechanisms include the energy exchange (relaxation) between

vibrational and translational modes caused by collisions within the cell, the vibrational energy lost or gained because of molecular depletion (dissociation) or production (recombination) in the cell, the electronic-translational energy exchange because of elastic collisions between electrons and heavy particles, the energy loss caused by electron impact ionization, the rate of energy loss caused by radiation caused by electronic transitions (a subset of Q_{rad} in the total energy equation (section O.5.3), and a term related to the work done on electrons by an electric field induced by the electron pressure gradient minus the flow work caused by electron pressure. The electron pressure flow work is normally considered as part of the electronic enthalpy in the inviscid (convective) portion of the flux balance. Moving the electron pressure from the convective term to the source term simplifies the expressions for eigenvalues and eigenvectors of the Jacobian of the inviscid flux vector.

O.3. Formulation of Inviscid Terms

The inviscid flux vector at cell face l is defined

$$\mathbf{g}_l = \left\{ \frac{1}{2} \left[\mathbf{g}_{L,l} + \mathbf{g}_{L-1,l} \right] \right\} - \left\{ \frac{1}{2(\setminus n)_l} \mathbf{R}_l \left| \mathbf{\Lambda}_l \right| \left[\mathbf{s}_l - \theta_l^{\text{lim}} \right] \right\} \tag{O.11}$$

where

$$\mathbf{g}_{L,l} = \left[\vec{\mathbf{f}}_L \cdot \vec{n}_l \right]_{\text{INV}}$$

The first term in braces is a second-order-accurate base approximation for $\mathbf{g}_l$. The second term in braces provides the upwind-biased numerical dissipation. It is a first-order dissipation when $\theta = 0$. It is a second-order dissipation when $\theta = 1$. The term $(\setminus n)_l$ is a shorthand notation for $\vec{\nabla} \setminus \cdot \vec{n}_l$, and can be thought of as the inverse of the projected distance between cell centers L and $L-1$ in a direction normal to cell face l. It is defined in equations (O.69) to (O.71) in section O.7. The variable $\setminus$ is a generic computational coordinate running in the direction of increasing generic index L.

The vector $\mathbf{s}_l$ is defined

$$\mathbf{s}_l = (\setminus n)_l \, \mathbf{R}_l^{-1} \left(\mathbf{q}_L - \mathbf{q}_{L-1} \right) \tag{O.12}$$

The matrix $\mathbf{R}_l^{-1}$ in equation (O.12) and the matrices $\mathbf{R}_l$ and $\mathbf{\Lambda}_l$ in equation (O.11) are related to the Jacobian of the inviscid flux vector $\mathbf{g}$ with respect to $\mathbf{q}$ in the following manner:

$$\mathbf{A} = \frac{\partial \mathbf{g}}{\partial \mathbf{q}} = \mathbf{R} \mathbf{\Lambda} \mathbf{R}^{-1} \tag{O.13}$$

The matrix $\mathbf{R}_l$ is the matrix of eigenvectors of $\mathbf{A}_l$, and $\mathbf{\Lambda}_l$ is a diagonal matrix containing the eigenvalues of $\mathbf{A}_l$. These matrices are defined in section O.3.1. Their elements, which are required at a cell face, are evaluated as appropriate averages of quantities at adjacent cell centers. This averaging procedure is discussed in section O.6. The matrix $|\mathbf{\Lambda}_l|$ is a diagonal matrix containing the absolute values of the eigenvalues of $\mathbf{A}_l$ with constraints on the minimum allowed magnitude of an eigenvalue given by

$$\tilde{\lambda}_l = \begin{cases} |\lambda_l|, & |\lambda_l| \geq 2\epsilon_l \\ \dfrac{\lambda_l^2}{4\epsilon_l} + \epsilon_l, & |\lambda_l| < 2\epsilon_l \end{cases} \tag{O.14}$$

The eigenvalue limiter ϵ_l was first used by Harten (ref. 35) to prevent the formation of expansion shocks across a sonic line where one eigenvalue equals zero. Its application is critical

in blunt body flows to prevent instabilities (often in the form of reversed flows at the stagnation point); these instabilities occur in the stagnation region where the eigenvalue U_l is near zero. The magnitude of ϵ_l, which is nondimensionalized by the free-stream velocity, is problem dependent. Yee et al. (ref. 36) has suggested a functional dependence of ϵ_l on the local values of sound speed and velocity. This relation has been adapted for use in the present work as follows:

$$\epsilon_l = \epsilon_o (a_l + |U_l| + |V_l| + |W_l|) \tag{O.15}$$

where ϵ_o is a user-defined constant that generally varies from 0.01 to 0.3. The larger values of ϵ_o are required for flows with extensive stagnation regions, as in the case of blunt-body flows. This limiter is called exclusively when the parameter $iabseig = 0$.

Experience shows that calculated convective heating levels are increased in the presence of a highly stretched grid across the boundary layer (cell growth factor greater than 1.2) and values of $\epsilon_o \geq 0.01$. However, small values of ϵ_o ($\epsilon_o < 0.01$) applied in all coordinate directions can cause instabilities. Several approaches to reduce the limiter across the boundary layer have been tried and documented in earlier versions of LAURA. The present approach, suggested by Netterfield (private communication, M. P. Netterfield, Fluid Gravity Engineering, Ltd., U.K., January 1993), changes the limiter for the following eigenvalue U_l:

$$\epsilon_l = \mathbf{max}[\epsilon_o (|U_l| + |V_l| + |W_l|), 0.001 a_l] \tag{O.16}$$

This limiter is called when the parameter $iabseig = 1$ and viscous terms are active across the cell face. Equation (O.15) is still used for the remaining eigenvalues. The parameter *iabseig* may need to be kept equal to 0 for some problems early in the relaxation process to survive difficult transients as the solution evolves.

The antidissipative flux correction for second-order accuracy is formulated for each element of $\mathbf{s}_l$ in a symmetric mode (parameter $iupwind = 0$, default) with

$$s_l^{\lim} = \mathbf{minmod}\left[2s_{l+1}\ ,\ 2s_l\ ,\ 2s_{l-1}\ ,\ \frac{(s_{l-1} + s_{l+1})}{2}\right] \tag{O.17}$$

and in an upwind biased mode ($iupwind = 1$) with

$$s_l^{\lim} = D_l^+ \ \mathbf{minmod}[s_{l-1}\ ,\ s_l] + D_l^- \ \mathbf{minmod}[s_{l+1}\ ,\ s_l] \tag{O.18}$$

where

$$D_l^+ = \frac{1}{2}\left[1 + \frac{\lambda_l}{\tilde{\lambda}_l}\right] \qquad \text{and} \qquad D_l^- = \frac{1}{2}\left[1 - \frac{\lambda_l}{\tilde{\lambda}_l}\right] \tag{O.19}$$

The **minmod** function returns the argument of smallest absolute magnitude when all the arguments are of the same sign, or it returns 0 if the arguments are of the opposite sign. The scheme reduces to first-order at cell faces where there is a sign change in the arguments of the **minmod** function. The symmetric limiter, derived by Yee (refs. 37 and 38) does not yield a strictly upwind biasing on the formulation of the flux vector. It involves symmetric functions of gradients in the neighborhood of the cell face, and algorithms based on these limiters are referred to as symmetric total variation diminishing (STVD) schemes. The STVD schemes involve little extra programming work over simple first-order algorithms because most of the quantities required in their implementation are already available. The upwind limiter, following the form of Wang and Richards (ref. 39), retains the proper zone of influence for the inviscid flux calculation. Marginally better convergence rates have been observed for the upwind limiter compared with the symmetric limiter; however, the symmetric limiter appears less diffusive in nonequilibrium flows. LAURA defaults to the symmetric limiter.

Equation (O.11) can be approximately linearized with respect to $\delta \mathbf{q}_L$ in the following manner. Define

$$\mathbf{g}^*_{l,L} = \frac{1}{2}\left[\mathbf{g}^{n+1}_{L,l} + \mathbf{g}^*_{L-1,l}\right] - \frac{1}{2(\backslash n)_l}\mathbf{R}_l\,|\mathbf{\Lambda}_l|\left[(\backslash n)_l\,\mathbf{R}_l^{-1}\left(\mathbf{q}^{n+1}_L - \mathbf{q}^*_{L-1}\right) - \theta \mathbf{s}_l^{\lim^*}\right] \qquad (O.20)$$

where superscript n refers to the current value at cell center L, superscript $n+1$ refers to the new value to be computed at cell center L, and superscript $*$ refers to the latest available value at neighbor cell $L-1$. The notation $\mathbf{g}^*_{l,L}$ refers to the inviscid flux through cell face l evaluated using the latest available data from cell center $L-1$ and the predicted data at cell center L. Elements of the vector $\mathbf{s}_l^{\lim^*}$ are also computed using current data at cell centers L and $L-1$. Substitute $\mathbf{g}^n_{L,l} + \mathbf{A}_{L,l}\delta\mathbf{q}_L$ for $\mathbf{g}^{n+1}_{L,l}$ in equation (O.20) to obtain

$$\mathbf{g}^*_{l,L} = \mathbf{g}_l + \frac{1}{2}\left(\mathbf{A}_{L,l} - |\mathbf{A}_l|\right)\delta\mathbf{q}_L$$

where

$$|\mathbf{A}_l| = \mathbf{R}_l\,|\mathbf{\Lambda}_l|\,\mathbf{R}_l^{-1}$$

In a similar manner, one can show that

$$\mathbf{g}^*_{l+1,L} = \mathbf{g}_{l+1} + \frac{1}{2}\left(\mathbf{A}_{L,l+1} + |\mathbf{A}_{l+1}|\right)\delta\mathbf{q}_L$$

The point-implicit discretization of the inviscid part of equation (O.6) can now be expressed by

$$\sum_{l=i,j,k}\left[\mathbf{g}^*_{l+1,L}\sigma_{l+1} - \mathbf{g}^*_{l,L}\sigma_l\right] = \sum_{l=i,j,k}\left[\mathbf{g}_{l+1}\sigma_{l+1} - \mathbf{g}_l\sigma_l\right] + \sum_{l=i,j,k}\left[\left(\mathbf{A}_{L,l+1} + |\mathbf{A}_{l+1}|\right)\sigma_{l+1} - \left(\mathbf{A}_{L,l} - |\mathbf{A}_l|\right)\sigma_l\right]\delta\mathbf{q}_L \qquad (O.21)$$

An application of Stokes theorem to the summation of $\mathbf{A}_{L,l}$ and $\mathbf{A}_{L,l+1}$ in equation (O.21) will show that

$$\sum_{l=i,j,k}\left[\mathbf{A}_{L,l+1}\sigma_{l+1} - \mathbf{A}_{L,l}\sigma_l\right]\delta\mathbf{q}_L = \sum_{l=i,j,k}\delta\vec{\mathbf{f}}_{L,\mathrm{INV}}\cdot\left[\vec{n}_{l+1}\sigma_{l+1} - \vec{n}_l\sigma_l\right] = 0$$

Therefore, equation (O.21) can be simplified as

$$\sum_{l=i,j,k}\left[\mathbf{g}^*_{l+1,L}\sigma_{l+1} - \mathbf{g}^*_{l,L}\sigma_l\right] = \sum_{l=i,j,k}\left[\mathbf{g}_{l+1}\sigma_{l+1} - \mathbf{g}_l\sigma_l\right] + \mathbf{M}_{L,\mathrm{INV}}\delta\mathbf{q}_L \qquad (O.22)$$

where

$$\mathbf{M}_{L,\mathrm{INV}} = \frac{1}{2}\sum_{l=i,j,k}\left[|\mathbf{A}_{l+1}|\,\sigma_{l+1} + |\mathbf{A}_l|\,\sigma_l\right] \qquad (O.23)$$

The definition of $\mathbf{A}$, $\mathbf{R}$, $\mathbf{R}^{-1}$, and $\mathbf{\Lambda}$ follow. The Jacobian of $\mathbf{g}$ with respect to $\mathbf{q}$ is

$$\mathbf{A} = \begin{bmatrix} U(\delta_{sr} - c_s) & c_s n_x & c_s n_y & c_s n_z & 0 & 0 \\ \tilde{\gamma}_r n_x - Uu & un_x(1-\beta) + U & -\beta v n_x + u n_y & -\beta w n_x + u n_z & \beta n_x & \phi n_x \\ \tilde{\gamma}_r n_y - Uv & -\beta u n_y + v n_x & v n_y(1-\beta) + U & -\beta w n_y + v n_z & \beta n_y & \phi n_y \\ \tilde{\gamma}_r n_z - Uw & -\beta u n_z + w n_x & -\beta v n_z + w n_y & w n_z(1-\beta) + U & \beta n_z & \phi n_z \\ \tilde{\gamma}_r U - UH & -\beta u U + H n_x & -\beta v U + H n_y & -\beta w U + H n_z & \beta U + U & \phi U \\ -U\epsilon_V & \epsilon_V n_x & \epsilon_V n_y & \epsilon_V n_z & 0 & U \end{bmatrix} \qquad (O.24)$$

The similarity transformation matrices $\mathbf{R}$ and $\mathbf{R}^{-1}$ are defined as

$$\mathbf{R} = \begin{bmatrix} \frac{\delta_{sr}}{a^2} & 0 & 0 & \frac{c_s}{2a^2} & \frac{c_s}{2a^2} & 0 \\ u/a^2 & l_x & m_x & \frac{u+an_x}{2a^2} & \frac{u-an_x}{2a^2} & 0 \\ v/a^2 & l_y & m_y & \frac{v+an_y}{2a^2} & \frac{v-an_y}{2a^2} & 0 \\ w/a^2 & l_z & m_z & \frac{w+an_z}{2a^2} & \frac{w-an_z}{2a^2} & 0 \\ \frac{\beta(u^2+v^2+w^2)-\tilde{\gamma}_r}{\beta a^2} & V & W & \frac{H+aU}{2a^2} & \frac{H-aU}{2a^2} & -\frac{\phi}{\beta a^2} \\ 0 & 0 & 0 & \frac{e_V}{2a^2} & \frac{e_V}{2a^2} & \frac{1}{a^2} \end{bmatrix} \tag{O.25}$$

$$\mathbf{R}^{-1} = \begin{bmatrix} a^2\delta_{sr} - c_s\tilde{\gamma}_r & \beta u c_s & \beta v c_s & \beta w c_s & \beta c_s & -\phi c_s \\ -V & l_x & l_y & l_z & 0 & 0 \\ -W & m_x & m_y & m_z & 0 & 0 \\ \tilde{\gamma}_r - Ua & -\beta u + an_x & -\beta v + an_y & -\beta w + an_z & \beta & \phi \\ \tilde{\gamma}_r + Ua & -\beta u - an_x & -\beta v - an_y & -\beta w - an_z & \beta & \phi \\ -e_V\tilde{\gamma}_r & \beta u e_V & \beta v e_V & \beta w e_V & -\beta e_v & a^2 - \phi e_V \end{bmatrix} \tag{O.26}$$

The diagonal matrix of eigenvalues of $\mathbf{A}$ is defined by

$$\mathbf{\Lambda} = \begin{bmatrix} U & 0 & 0 & 0 & 0 & 0 \\ 0 & U & 0 & 0 & 0 & 0 \\ 0 & 0 & U & 0 & 0 & 0 \\ 0 & 0 & 0 & U+a & 0 & 0 \\ 0 & 0 & 0 & 0 & U-a & 0 \\ 0 & 0 & 0 & 0 & 0 & U \end{bmatrix} \tag{O.27}$$

The variable c_s is the mass fraction of species s where

$$c_s = \frac{\rho_s}{\rho}$$

The variables β, ϕ, and $\tilde{\gamma}_r$ are related to the partial derivatives of pressure with respect to $\mathbf{q}$.

$$\beta = \frac{\partial p}{\partial(\rho E)} = \frac{\bar{R}}{\rho C_{v,tr}} \sum_{r=1, r\neq e}^{N_s} \frac{\rho_r}{M_r} \tag{O.28}$$

$$\phi = \frac{\partial p}{\partial(\rho e_V)} = \frac{\bar{R}}{\rho C_{v,V}} \frac{\rho_e}{M_e} - \beta \tag{O.29}$$

$$\tilde{\gamma}_s = \frac{\partial p}{\partial \rho_s} = \frac{RT_q}{M_s} + \frac{\beta}{2}\left(u^2 + v^2 + w^2\right) - \beta e_s - \phi e_{V,s} \tag{O.30}$$

The variable a is the frozen speed of sound.

$$a^2 = \sum_{s=1}^{N_s} c_s \tilde{\gamma}_s + \beta\left[H - u^2 - v^2 - w^2\right] + \phi e_V = (1+\beta)\frac{p}{\rho} \tag{O.31}$$

This definition of a^2 comes from the evaluation of the eigenvalues of **A**. The variable $\bar{R}$ is the universal gas constant, and M_s is the molecular weight of species s.

The variables n_x, n_y, and n_z are the x-, y-, and z-components of a unit vector normal to a computational cell face, and U is the normal component of velocity through the cell face, defined by

$$U = \vec{V} \cdot \vec{n} = un_x + vn_y + wn_z \tag{O.32}$$

The two unit vectors $\vec{l}$ and $\vec{m}$ are defined such that $\vec{n}$, $\vec{l}$, and $\vec{m}$ are mutually orthogonal (i.e., $n_i l_i = n_i m_i = l_i m_i = 0$). The velocity components in the $\vec{l}$- and $\vec{m}$-directions, tangent to the cell face, are then defined by

$$V = \vec{V} \cdot \vec{l} = ul_x + vl_y + wl_z \tag{O.33}$$

$$W = \vec{V} \cdot \vec{m} = um_x + vm_y + wm_z \tag{O.34}$$

In the matrices defined above, the first row and column correspond to the N_s species continuity equations. The subscript s refers to row s and species s, and the subscript r refers to column r and species r where both s and r vary from 1 to 11 in the present model. Note in equation (O.30) that $T_q = T_V$ when s is an electron; otherwise, $T_q = T$. Further details of the derivations may be found in reference 4.

O.4. Formulation of Viscous Terms

The viscous stresses on a cell face with unit normal $\vec{n}$ in the orthogonal directions $\vec{n}$, $\vec{l}$, and $\vec{m}$ are given by

$$\tau_{nn} = \lambda_l\left(\frac{\partial U}{\partial s_n} + \frac{\partial V}{\partial s_l} + \frac{\partial W}{\partial s_m}\right) + 2\mu_l\frac{\partial U}{\partial s_n} \tag{O.35}$$

$$\tau_{nl} = \mu_l\left(\frac{\partial V}{\partial s_n} + \frac{\partial U}{\partial s_l}\right) \tag{O.36}$$

$$\tau_{nm} = \mu_l\left(\frac{\partial W}{\partial s_n} + \frac{\partial U}{\partial s_m}\right) \tag{O.37}$$

where U, V, and W are velocity components, and s_n, s_l, and s_m are arc lengths in the $\vec{n}$-, $\vec{l}$-, and $\vec{m}$-directions, respectively. The variables μ and λ are the viscosity coefficients. All transport properties at cell face l are obtained as linear averages of properties at adjacent cell centers.

The component of shear stress acting in the s-direction (s being a dummy variable for x, y, or z) on a cell face with unit normal $\vec{n}$ can be expressed

$$\tau_{ns} = \tau_{nn}n_s + \tau_{nl}l_s + \tau_{nm}m_s \tag{O.38}$$

Substituting equations (O.35) to (O.37) into equation (O.38), collecting terms, and simplifying (ref. 2) yields the following relation for shear stress in the s-direction:

$$\tau_{ns} = \mu_l \left(\frac{\partial \dot{s}}{\partial \xi} \xi n + \frac{\partial \dot{s}}{\partial \eta} \eta n + \frac{\partial \dot{s}}{\partial \zeta} \zeta n + \frac{\partial U}{\partial \xi} \frac{\partial \xi}{\partial s} + \frac{\partial U}{\partial \eta} \frac{\partial \eta}{\partial s} + \frac{\partial U}{\partial \zeta} \frac{\partial \zeta}{\partial s} \right) + \lambda_l \left(\frac{\partial \vec{\mathbf{u}}}{\partial \xi} \cdot \vec{\nabla} \xi + \frac{\partial \vec{\mathbf{u}}}{\partial \eta} \cdot \vec{\nabla} \eta + \frac{\partial \vec{\mathbf{u}}}{\partial \zeta} \cdot \vec{\nabla} \zeta \right) n_s \tag{O.39}$$

where $\dot{s}$ is a dummy variable for u, v, or w corresponding to $s = x$, y, or z, and terms like ξn or ηl are shorthand notations for $\vec{\nabla} \xi \cdot \vec{n}$ or $\vec{\nabla} \eta \cdot \vec{l}$, respectively. A thin-layer approximation in the λ-coordinate direction ($\lambda = \xi$, η, or ζ) simplifies equation (O.39) by neglecting derivatives in the other two coordinate directions. Consequently,

$$\dot{\tau}_{ns} = \mu_l \left(\frac{\partial \dot{s}}{\partial \lambda} + \frac{1}{3} \frac{\partial U}{\partial \lambda} n_s \right) \lambda n \tag{O.40}$$

where n refers to the direction normal to a constant λ-surface, and the prime superscript refers to the thin-layer approximation. The Stokes relation, $\lambda = -\frac{2}{3}\mu$, and geometric identities have also been used in the simplification of equation (O.39). The viscous terms on the other two coordinate surfaces are also neglected in the thin-layer approximation because their contribution to the overall momentum and energy balance is small. These approximations are valid so long as the boundary layer is relatively thin and the λ-direction is approximately normal to the high gradient region.

Mass diffusion and energy conduction contributions to the viscous terms are functions of gradients normal to the cell face. For example, the gradient of T in the normal (n) direction is expressed

$$\frac{\partial T}{\partial s_n} = \vec{\nabla}(T) \cdot \vec{n} = \frac{\partial T}{\partial \xi} \xi n + \frac{\partial T}{\partial \eta} \eta n + \frac{\partial T}{\partial \zeta} \zeta n \tag{O.41}$$

The thin-layer approximation to equation (O.41) is expressed

$$\frac{\partial T}{\partial s_n} = \frac{\partial T}{\partial \lambda} \lambda n \tag{O.42}$$

Derivatives in the λ-direction are evaluated to second-order accuracy in computational space as follows:

$$\left(\frac{\partial u}{\partial \lambda} \right)^*_{l,L} = u^{n+1}_L - u^*_{L-1} = u^n_L + \delta u_L - u^*_{L-1} = \left(\frac{\partial u}{\partial \lambda} \right)_l + \delta u_L$$

$$\left(\frac{\partial u}{\partial \lambda} \right)^*_{l+1,L} = u^*_{L+1} - u^{n+1}_L = u^*_{L+1} - u^n_L - \delta u_L = \left(\frac{\partial u}{\partial \lambda} \right)_{l+1} - \delta u_L$$

For instance, the partial of u with respect to η in the ξ-direction is evaluated as follows (assuming a rectangular ordering of mesh points) :

$$\left(\frac{\partial u}{\partial \eta} \right)_{i+1,J,K} = \frac{u^*_{I,J+1,K} - u^*_{I,J-1,K} + u^*_{I+1,J+1,K} - u^*_{I+1,J-1,K}}{4}$$

The derivatives in the directions along the face (i.e., those derivatives neglected in the thin-layer approximation) have no functional dependence on the cell center. Therefore, the point-implicit treatment of the full Navier-Stokes equations is identical to the thin-layer Navier-Stokes equations.

Now, define $\mathbf{h}_l$ as a function of differences evaluated using currently available data, for example $\left(\frac{\partial u}{\partial \chi}\right)_l$, and define $\mathbf{h}^*_{l,L}$ as a function of differences using predicted values at cell center L, for example $\left(\frac{\partial u}{\partial \chi}\right)^*_{l,L}$. These definitions permit the linearization of the viscous terms to be expressed as follows:

$$\begin{aligned} \mathbf{h}^*_{l,L} &= \mathbf{h}_l - \mathbf{B}_{l,L}\delta\mathbf{q}_L \\ \mathbf{h}^*_{l+1,L} &= \mathbf{h}_{l+1} + \mathbf{B}_{l+1,L}\delta\mathbf{q}_L \end{aligned} \tag{O.43}$$

where

$$\begin{aligned} \mathbf{B}_{l,L} &= -\frac{\partial \mathbf{h}^*_{l,L}}{\partial \mathbf{q}_L} = -\frac{\partial \acute{\mathbf{h}}^*_{l,L}}{\partial \mathbf{q}_L} \\ \mathbf{B}_{l+1,L} &= \frac{\partial \mathbf{h}^*_{l+1,L}}{\partial \mathbf{q}_L} = \frac{\partial \acute{\mathbf{h}}^*_{l+1,L}}{\partial \mathbf{q}_L} \end{aligned} \tag{O.44}$$

The point-implicit implementation of the viscous terms follows the example set in the previous section on the inviscid terms

$$\sum_{l=i,j,k}\left[\mathbf{h}^*_{l+1,L}\sigma_{l+1} - \mathbf{h}^*_{l,L}\sigma_l\right] = \sum_{l=i,j,k}\left[\mathbf{h}_{l+1}\sigma_{l+1} - \mathbf{h}_l\sigma_l\right] + \mathbf{M}_{L,\mathrm{VIS}}\delta\mathbf{q}_L \tag{O.45}$$

where

$$\mathbf{M}_{L,\mathrm{VIS}} = \sum_{l=i,j,k}\left[\mathbf{B}_{l+1,L}\sigma_{l+1} + \mathbf{B}_{l,L}\sigma_l\right] \tag{O.46}$$

In the case of the thin-layer Navier-Stokes equations, the summation would only include one of the i-, j-, or k-directions, depending on the orientation of the computational coordinates with the body.

O.5. Formulation of Source Terms

O.5.1. Species Conservation

The mass rate of production of species s per unit volume is expressed as

$$\dot{\omega}_s = M_s \sum_{r=1}^{N_r} (\beta_{s,r} - \alpha_{s,r})\left[R_{f,r} - R_{b,r}\right] \tag{O.47}$$

where N_r is the number of reactions; $\alpha_{s,r}$ and $\beta_{s,r}$ are the stoichiometric coefficients for reactants and products in the r-reaction, respectively; and $R_{f,r}$ and $R_{b,r}$ are the forward and backward reaction rates for the r-reaction, respectively. These rates are defined by

$$R_{f,r} = k_{f,r}\prod_{s=1}^{N_s}\left(\frac{\rho_s}{M_s}\right)^{\alpha_{s,r}} \qquad \text{and} \qquad R_{b,r} = k_{b,r}\prod_{s=1}^{N_s}\left(\frac{\rho_s}{M_s}\right)^{\beta_{s,r}}$$

where $k_{f,r}$ and $k_{b,r}$ are the forward and backward reaction rate coefficients, respectively, defined in reference 4, and N_s is the number of chemical species. Five different chemical kinetic models for air chemistry are supported within LAURA (in files `kinetic.F` and `source.F`) through the definition of *kmodel* in `gas_model_vars.strt`. The default is *kmodel* = 3 in which the equilibrium constants are taken from reference 34 and which correspond to a number density

of 10^{16} cm^{-3}. The forward reaction rates are taken from reference 40. Numerical difficulties associated with chemical source terms are alleviated by limiting the minimum and maximum values of temperature used to compute the reaction rate coefficients. The parameters *tmin* and *tmax* are set to 1000 and 50 000, respectively, in `air.F`, which is a `block data` routine. As a converged solution is approached, the lower limit can usually be further diminished; however, experience with problems tested so far shows no significant effect on aerothermal loads. The upper limit is established to reflect the range of validity of the curve fits for thermodynamic properties.

The reaction rate coefficients are explicit functions of T and T_V. Consequently, the Jacobian of $\dot{\omega}_s$ with respect to $\mathbf{q}$ can be explicitly evaluated as follows:

$$\frac{\partial \dot{\omega}_s}{\partial q_j} = \left.\frac{\partial \dot{\omega}_s}{\partial q_j}\right|_{T,T_V} + \left.\frac{\partial \dot{\omega}_s}{\partial T}\right|_{T_V,\mathbf{q}} \frac{\partial T}{\partial q_j} + \left.\frac{\partial \dot{\omega}_s}{\partial T_V}\right|_{T,\mathbf{q}} \frac{\partial T_V}{\partial q_j} \tag{O.48}$$

where all derivatives with respect to q_j are evaluated at q_k $(k \neq j)$. The differential relations between T and $\mathbf{q}$ and between T_V and $\mathbf{q}$ can be expressed as

$$\begin{aligned} \rho C_{v,tr}\, dT &= \frac{u^2+v^2+w^2}{2} d\rho - \sum_{s=1}^{N_s} (\epsilon_s - \epsilon_{V,s})\, d\rho_s \\ &\quad - u\,d(\rho u) - v\,d(\rho v) - w\,d(\rho w) + d(\rho E) - d(\rho e_V) \end{aligned} \tag{O.49}$$

$$\rho C_{v,V}\, dT_V = d(\rho e_V) - \sum_{s=1}^{N_s} \epsilon_{V,s}\, d\rho_s \tag{O.50}$$

O.5.2. Total Energy Conservation

The radiative energy transport term Q_{rad} is treated in a purely explicit manner. Radiative energy transport has been calculated using the method of Hartung (ref. 29), which is based initially on converged, nonradiative, nonequilibrium flow field solutions. These radiative source terms are then held constant, while the governing equations are relaxed again. In cases of strong radiation, this relaxation process may require a slower introduction of the source through appropriate averaging of the old and new source terms (ref. 31). There is no point-implicit contribution from this term in the algorithm.

O.5.3. Vibrational-Electronic Energy Conservation

The vibrational-electronic energy source term $\dot{\omega}_V$ can be subdivided into three functionally distinct sets of terms.

$$\begin{aligned} \dot{\omega}_V &= \left\{ \sum_{s=\text{mol.}} \dot{\omega}_s \hat{D}_s - \sum_{r=\text{elec.imp.}} (R_{f,r} - R_{b,r})\, \hat{I}_r \right\} \\ &\quad + \left\{ \sum_{s=\text{mol.}} \rho_s \frac{\left(\epsilon^*_{V,s} - \epsilon_{V,s}\right)}{<\tau_s>} + 3\rho_e \bar{R}(T - T_V) \sum_{s \neq e} \frac{\nu_{es}}{M_s} \right\} \\ &\quad + \left\{ Q_{\text{rad}} - p_e \vec{\nabla} \cdot \vec{u} \right\} \end{aligned} \tag{O.51}$$

The first set, the reactive source terms in the first pair of braces of equation (O.51), is composed of terms that are proportional to either $\dot{\omega}_s$ or to $(R_{f,r} - R_{b,r})$. In the first case, the proportionality factor, $\hat{D}_s$, represents the average vibrational energy per unit mass created or

destroyed through recombination or dissociation of molecules. In the simplest approximation, it is set equal to the average vibrational-electronic energy, e_V, although more comprehensive treatments that model preferential dissociation of vibrationally excited molecules can be employed. In the second case, the proportionality factor, $\hat{I}_r$, represents the average translational energy per mole (mol.) lost by a free electron in freeing another electron from a neutral heavy particle in reaction r through the process of electron impact (elec.imp.) ionization. It is approximated by the ionization energy from an excited state of the target particle. Further details on these points are available in references 34 and 4. The point-implicit formulation of these terms treats the proportionality factor explicitly and the reaction rates implicitly according to equations (O.48) to (O.50).

The second set, the relaxation terms in the second pair of braces in equation (O.51), models the energy exchange between heavy particle translational-rotational modes and vibrational-electronic and electron translational modes. The first term in these braces, which models the exchange between vibrational and heavy particle translational modes, can be approximated by

$$\sum_{s=\text{mol.}} \rho_s \frac{e^*_{V,s} - e_{V,s}}{< \tau_s >} \approx \rho C_{v,V} \frac{T - T_V}{\tau_V} \tag{O.52}$$

where

$$\frac{1}{\tau_V} = \frac{\displaystyle\sum_{s=\text{mol.}} \frac{\rho_s}{M_s < \tau_s >}}{\displaystyle\sum_{s=\text{mol.}} \frac{\rho_s}{M_s}} \tag{O.53}$$

The approximations in equations (O.52) to (O.53) are made to reduce the number of thermodynamic and relaxation time variables to be carried through the calculation. Also, direct evaluation of the equilibrium value, $e^*_{V,s}$, is more cumbersome than working directly with the translational temperature T. This approximation degenerates as the differences between T and T_V get very large, but it is believed to be consistent within the total context of approximations made in the two-temperature model. The vibrational relaxation time $< \tau_s >$ is related to a number density weighted correlation of Millikan and White (ref. 41)

$$p\tau_s^{MW} = \frac{\displaystyle\sum_{j=1, j\neq e}^{N_s} \tilde{n}_j \exp\left[A_{s,j}\left(T^{-1/3} - 0.015\mu_{s,j}^{1/4}\right) - 18.42\right]}{\displaystyle\sum_{j=1, j\neq e}^{N_s} \tilde{n}_j} \tag{O.54}$$

and a high-temperature limiting correction of Park (ref. 34)

$$\tau_s^P = (\sigma_s \bar{c}_s \tilde{n})^{-1} \tag{O.55}$$

so that

$$< \tau_s > = \tau_s^{MW} + \tau_s^P \tag{O.56}$$

where

$$A_{s,j} = A_{s,s}\left(\frac{\mu_{s,j}}{\mu_{s,s}}\right)^{1/2}$$

and

$$A_{s,s} = \begin{cases} 220 & (N_2) \\ 129 & (O_2) \\ 168 & (NO) \end{cases}$$

The pressure p in equation (O.54) is in units of atmospheres. The two-temperature model should also have a corresponding term relating the energy exchange of translational and electronic energy. This transfer has not yet been formally included in the present work; however, it should be noted that the driving potential in the present approximation is already based on both the vibrational and electronic energies. The second term in these braces in equation (O.51) models the direct exchange of translational energy between electrons and heavy particles. This exchange rate is generally much slower than the previous term. Both terms in this set are now proportional to the difference between the translational and vibrational temperatures, $T - T_V$. Here again, the point-implicit formulation of these terms treats the proportionality factor explicitly and the driving potential $T - T_V$ implicitly according to equations (O.49) and (O.50).

The third set, the field-dependent terms in the third pair of braces of equation (O.51), is the functions of properties at the cell center and at the neighboring cells. (The first two sets are functions only of properties at the cell center.) These terms include radiative energy transport, which is work done by the electric field on electrons and electron pressure flow work, combined into a single term. Radiative energy transport is treated explicitly, if at all, as described before. The other contribution to the field-dependent terms is also treated explicitly. In fact, in the cases tested to date (where the maximum electron number densities were approximately 4 percent of the total number density), omission of this term has little effect on the flow field.

O.5.4. Point-Implicit Relaxation of Source Term

The source term in equation (O.6) can be approximately linearized in the following manner:

$$\dot{\omega}_L^{n+1} = \dot{\omega}_L^n + \mathbf{M}_{L,\mathrm{SRC}}\delta\mathbf{q}_L \tag{O.57}$$

where

$$\mathbf{M}_{L,\mathrm{SRC}} = \frac{\partial\dot{\omega}_L}{\partial\mathbf{q}_L} \tag{O.58}$$

and the elements of $\mathbf{M}_{L,\mathrm{SRC}}$ are calculated as described above.

O.6. Averaging Procedure

The variables at cell faces are evaluated as follows:

$$d_l = \frac{C_1 d_L + d_{L-1}}{C_1 + 1} \tag{O.59}$$

where d is a dummy variable representing u, v, w, c_s, H, ϵ_V, h_s, and $\epsilon_{V,s}$. (Only the averaged values of velocities and total enthalpy are required for perfect gas flows.) The weighting parameter is defined by

$$C_1 = \left(\frac{\rho_L}{\rho_{L-1}}\right)^{\frac{1}{2}}$$

The variables β, ϕ, and T are computed using a pressure-weighted average as defined below with dummy variable d.

$$d_l = \frac{d_L d_{L-1}\,(p_L + p_{L-1})}{p_L d_{L-1} + p_{L-1} d_L}$$

The variable $\epsilon_{s,l}$ required in the evaluation of $\tilde{\gamma}_{s,l}$ is defined as

$$\epsilon_{s,l} = h_{s,l} - \frac{RT_l}{M_s}$$

All quantities required for the evaluation of sound speed at the cell wall are now available. A lower limit on sound speed evaluated using cell wall averages, equal to the free-stream sound speed, has been applied to deal with difficult transients across strong shocks in equilibrium and nonequilibrium flows.

This simple averaging procedure does not exactly satisfy Roe's property U for equilibrium and nonequilibrium flows. The pressure weighting averaging on β can be shown to nearly satisfy Roe's property U for strong shocks and equilibrium flow. Vinokur (ref. 42), Liu and Vinokur (ref. 43), and Grossman and Cinnella (ref. 44) have proposed averaging schemes that enforce Roe's property U. These formulations have not been investigated here.

O.7. Geometrical Relations

The recommended, second-order-accurate formulations for face-centered metrics, which are required in the evaluation of the viscous dissipation terms across cell faces, are presented below.

$$\vec{\nabla}\xi_{i,J,K} = \frac{\Omega_{I,J,K}\left(\vec{\sigma}_{i-1,J,K}+\vec{\sigma}_{i,J,K}\right)+\Omega_{I-1,J,K}\left(\vec{\sigma}_{i,J,K}+\vec{\sigma}_{i+1,J,K}\right)}{4\Omega_{I,J,K}\Omega_{I-1,J,K}} \tag{O.60}$$

$$\vec{\nabla}\xi_{I,j,K} = \frac{\Omega_{I,J,K}\left(\vec{\sigma}_{i,J-1,K}+\vec{\sigma}_{i+1,J-1,K}\right)+\Omega_{I,J-1,K}\left(\vec{\sigma}_{i,J,K}+\vec{\sigma}_{i+1,J,K}\right)}{4\Omega_{I,J,K}\Omega_{I,J-1,K}} \tag{O.61}$$

$$\vec{\nabla}\xi_{I,J,k} = \frac{\Omega_{I,J,K}\left(\vec{\sigma}_{i,J,K-1}+\vec{\sigma}_{i+1,J,K-1}\right)+\Omega_{I,J,K-1}\left(\vec{\sigma}_{i,J,K}+\vec{\sigma}_{i+1,J,K}\right)}{4\Omega_{I,J,K}\Omega_{I,J,K-1}} \tag{O.62}$$

$$\vec{\nabla}\eta_{i,J,K} = \frac{\Omega_{I,J,K}\left(\vec{\sigma}_{I-1,j,K}+\vec{\sigma}_{I-1,j+1,K}\right)+\Omega_{I-1,J,K}\left(\vec{\sigma}_{I,j,K}+\vec{\sigma}_{I,j+1,K}\right)}{4\Omega_{I,J,K}\Omega_{I-1,J,K}} \tag{O.63}$$

$$\vec{\nabla}\eta_{I,j,K} = \frac{\Omega_{I,J,K}\left(\vec{\sigma}_{I,j-1,K}+\vec{\sigma}_{I,j,K}\right)+\Omega_{I,J-1,K}\left(\vec{\sigma}_{I,j,K}+\vec{\sigma}_{I,j+1,K}\right)}{4\Omega_{I,J,K}\Omega_{I,J-1,K}} \tag{O.64}$$

$$\vec{\nabla}\eta_{I,J,k} = \frac{\Omega_{I,J,K}\left(\vec{\sigma}_{I,j,K-1}+\vec{\sigma}_{I,j+1,K-1}\right)+\Omega_{I,J,K-1}\left(\vec{\sigma}_{I,j,K}+\vec{\sigma}_{I,j+1,K}\right)}{4\Omega_{I,J,K}\Omega_{I,J,K-1}} \tag{O.65}$$

$$\vec{\nabla}\zeta_{i,J,K} = \frac{\Omega_{I,J,K}\left(\vec{\sigma}_{I-1,J,k}+\vec{\sigma}_{I-1,J,k+1}\right)+\Omega_{I-1,J,K}\left(\vec{\sigma}_{I,J,k}+\vec{\sigma}_{I,J,k+1}\right)}{4\Omega_{I,J,K}\Omega_{I-1,J,K}} \tag{O.66}$$

$$\vec{\nabla}\zeta_{I,j,K} = \frac{\Omega_{I,J,K}\left(\vec{\sigma}_{I,J-1,k}+\vec{\sigma}_{I,J-1,k+1}\right)+\Omega_{I,J-1,K}\left(\vec{\sigma}_{I,J,k}+\vec{\sigma}_{I,J,k+1}\right)}{4\Omega_{I,J,K}\Omega_{I,J-1,K}} \tag{O.67}$$

$$\vec{\nabla}\zeta_{I,J,k} = \frac{\Omega_{I,J,K}\left(\vec{\sigma}_{I,J,k-1}+\vec{\sigma}_{I,J,k}\right)+\Omega_{I,J,K-1}\left(\vec{\sigma}_{I,J,k}+\vec{\sigma}_{I,J,k+1}\right)}{4\Omega_{I,J,K}\Omega_{I,J,K-1}} \tag{O.68}$$

In the case of the thin-layer Navier-Stokes equations, only the vectors defined by equations (O.60), (O.64), and (O.68) are required, depending on the orientation of the coordinate system. The dot product of these vectors with the corresponding unit normal to the cell face (recall, $\backslash n = \vec{\nabla}\backslash \cdot \vec{n}$) can be approximated as follows:

$$\xi n_{i,J,K} \approx \frac{\Omega_{I,J,K}\left(\sigma_{i-1,J,K}+\sigma_{i,J,K}\right)+\Omega_{I-1,J,K}\left(\sigma_{i,J,K}+\sigma_{i+1,J,K}\right)}{4\Omega_{I,J,K}\Omega_{I-1,J,K}} \tag{O.69}$$

$$\eta n_{I,j,K} \approx \frac{\Omega_{I,J,K}\left(\sigma_{I,j-1,K}+\sigma_{I,j,K}\right)+\Omega_{I,J-1,K}\left(\sigma_{I,j,K}+\sigma_{I,j+1,K}\right)}{4\Omega_{I,J,K}\Omega_{I,J-1,K}} \tag{O.70}$$

$$\zeta n_{I,J,k} \approx \frac{\Omega_{I,J,K}\left(\sigma_{I,J,k-1}+\sigma_{I,J,k}\right)+\Omega_{I,J,K-1}\left(\sigma_{I,J,k}+\sigma_{I,J,k+1}\right)}{4\Omega_{I,J,K}\Omega_{I,J,K-1}} \tag{O.71}$$

Another useful formulation in the programming of the thin-layer Navier-Stokes equations involves a geometric relation between the unit normal to a cell face and the gradient of the

computational coordinate that defines the cell face

$$\left(\frac{\partial \xi}{\partial s}\right)_{i,J,K} = (\xi n n_s)_{i,J,K} \tag{O.72}$$

$$\left(\frac{\partial \eta}{\partial s}\right)_{I,j,K} = (\eta n n_s)_{I,j,K} \tag{O.73}$$

$$\left(\frac{\partial \zeta}{\partial s}\right)_{I,J,k} = (\zeta n n_s)_{I,J,k} \tag{O.74}$$

where s is a dummy variable for x, y, or z.

O.8. Relaxation Algorithm

The governing relaxation equation is obtained by combining the results of equations (O.6), (O.23), (O.46), and (O.58) and taking the limit as time step δt goes to infinity. Thus,

$$\mathbf{M}_L \delta \mathbf{q}_L = \mathbf{r}_L \tag{O.75}$$

where $\mathbf{M}_L$ is the point-implicit Jacobian given by

$$\mathbf{M}_L = rf_i \mathbf{M}_{L,\mathrm{INV}} + rf_v \mathbf{M}_{L,\mathrm{VIS}} - \Omega_L \mathbf{M}_{L,\mathrm{SRC}} \tag{O.76}$$

and $\mathbf{r}$ is the right-hand-side solution (residual) vector given by

$$\mathbf{r}_L = -\sum_{l=i,j,k} \left[(\mathbf{g}_{l+1} + \mathbf{h}_{l+1})\, \sigma_{l+1} - (\mathbf{g}_l + \mathbf{h}_l)\, \sigma_l \right] + \dot{\omega}_L \Omega_L \tag{O.77}$$

Relaxation factors are used to control stability and convergence. Numerical tests in reference 18 indicate that underrelaxation is appropriate for the inviscid contribution to the residual, with $rf_i > 1.5$. Overrelaxation is appropriate for the viscous contribution to the residual with $rf_v > 0.5$ provided relaxation sweeps are across the boundary layer; otherwise, $rf_v \geq 1$. The viscous relaxation factor is automatically set to 1 for directions tangent to the sweep direction. The lower limits yield the fastest convergence rates, but can lead to instabilities if the solution is far from convergence or if the point-implicit Jacobian is "frozen," as discussed below, for too long. It is sometimes necessary to choose $rf_i \geq 3$ and $rf_v \geq 2$ to get past some difficult transients in the early stages of the relaxation process which defy linear analysis. When these transients pass, it is then advisable to switch to the lower limits of these parameters to get the best convergence rate. Convergence can eventually stall at some point because of limit cycles associated with the `minmod` function in equation (O.17). This stalling can be alleviated by again increasing the relaxation factors.

The solution vector $\mathbf{r}_L$ and the Jacobian $\mathbf{M}_L$ are evaluated using the latest available data. Consequently, the algorithm requires only a single level of storage. One can solve for $\delta \mathbf{q}_L$ using Gaussian elimination. Numerical experiments have shown that pivoting is not required, and so the algorithm is easily vectorized. An LU factorization of the Jacobian can be saved (frozen) over large blocks of iterations (10 to 50, as defined by *njcobian*) to further reduce computational costs as the solution converges. The Jacobian will generally need to be updated every iteration early in the calculation, when rapid adjustment of the solution occurs. There is a large cost in computational memory required for the Jacobian freezing; however, the use of solid-state memory (see section 11.4) essentially eliminates this problem on CRAY class computers.

One final scaling can be applied to $\delta \mathbf{q}_L$ before computing $\mathbf{q}_L^{n+1}$ to dampen potentially catastrophic perturbations in the evolving solution. Define κ_L by

$$\kappa_L = \mathbf{min} \left[1, \frac{safe\ \rho_L}{|\delta \mathbf{q}_L|_1} \right] \tag{O.78}$$

where $safe = 0.5$ is presently the hard-coded value, and the standard definition of an L_1 norm is employed. The value of κ_L is usually equal to 1; however, early in the relaxation process and occasionally in regions of very severe expansions, this parameter will engage to limit the computed solution as follows:

$$\mathbf{q}_L^{n+1} = \mathbf{q}_L^n + \kappa_L \delta \mathbf{q}_L \tag{O.79}$$

New values for T and T_V are obtained through a Newton-Raphson iteration based on equations (O.49) and (O.50). Thermodynamic properties and reaction rate coefficients are advanced every iteration based on these updated values of T and T_V. Transport properties are updated every *ntrnsprt* iterations.

The strategy used to drive the right-hand side of equation (O.77) to zero should take advantage of the host computer architecture and the physics of the problem. Generally, the solution is relaxed one plane at a time, and vector lengths are equal to the number of cells in a plane. Numerical tests indicate that relaxation sweeps which run from a wall across the boundary layer to the opposite boundary and then back again are the most efficient for the blunt-body problem. Effects of a perturbation at a wall are felt at the opposite wall after one sweep. Effects of a perturbation at one cell in a plane parallel to the wall require N iterations to be felt by a cell whose index differs from the source cell by N.

The ordering of the sweeps can be used to speed convergence, but in numerical tests performed to date, final, converged steady-state solution is not affected. Thus, one should be able to solve a large number of cells using a massively parallel processing computer in which each cell (or small group of cells) is relaxed semi-independently of its neighbor cells (cell groups) using its own processor. The expression "semi-independently" means that a cell (cell group) will need updated information from its neighbor cells (cell groups), but neither the order that it receives this information nor the lag time it takes for this information to arrive is critically important. As long as each processor has immediate access to some level of information from its neighbors (which could be stored locally), the execution stream could proceed uninterrupted in a parallel, asynchronous mode. A crude simulation of asynchronous iteration, discussed in reference 45, demonstrated that computational cells could be advanced in a random order without sacrifice of stability or convergence.

Asynchronous iteration has been tested on a four-processor CRAY 2 and an eight-processor CRAY Y-MP in reference 18. In these tests, the flow domains were subdivided into partitions with a single task assigned to each partition. Partition boundaries are dynamically adjusted to concentrate relaxation sweeps in the regions that are slowest to converge. Because no synchronization is required, all tasks (processors) can execute throughout the computation without interruption. A comparison of convergence histories for the solution of hypersonic flow in thermochemical nonequilibrium over an axisymmetric body is shown for a single task and a six-task, adaptive partition test in figure O.2. The symbols show the error norm for each individual task of the six-task run. The solid line shows the total error norm for the six-task run, and the dashed line shows the error norm for the single task run. Adaptive partitioning has allowed the six-task case to converge to a lower error norm than the single task case for the same amount of CPU time. Furthermore, the actual elapsed time for the six-task case would be a factor of 6 smaller than for the single task case on a dedicated machine.

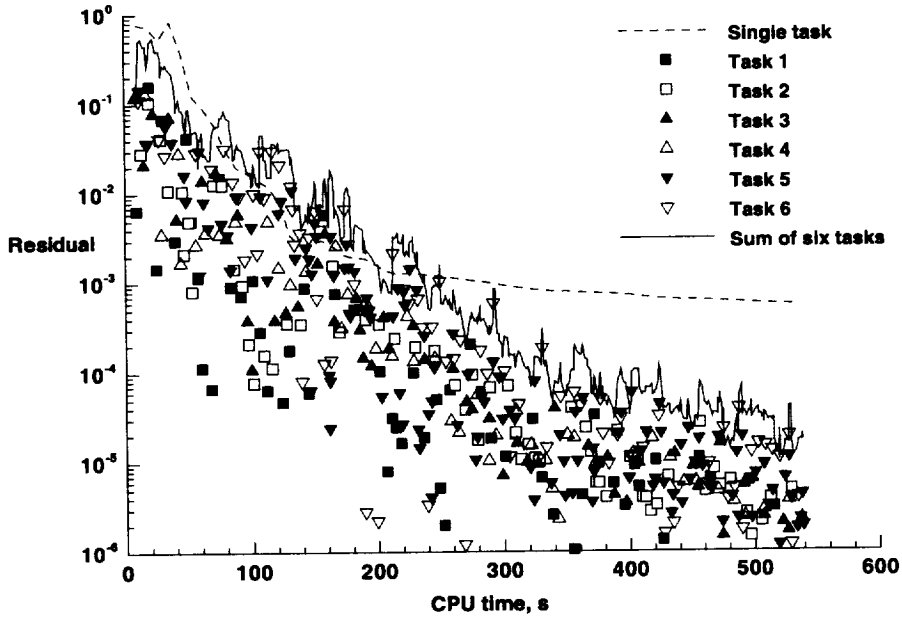


Figure O.2. Convergence histories for single-task and six-task, adaptive partitioned algorithms applied to problem of nonequilibrium, hypersonic flow over blunt, axisymmetric body.

Appendix P

FORTRAN Variables Discussed in This Manual

The following FORTRAN variables are discussed in this document:

attack	angle of attack, deg
axfac	axial stretching factor for grid (*konic* = 1)
b	axial shape parameter for conic geometry
betagrd	control parameter for grid alignment
bgas	perfect gas β, where $\beta = \gamma - 1$
epsa	coefficient for eigenvalue limiter (ϵ_o)
ept	wall temperature relaxation factor
ep0	control parameter for grid alignment (section 11.1.1)
errd	error criteria for doubling grid density in body-normal direction
fctrjmp	factor for property jump that identifies shock location (section 11.1.1)
fsh	fraction of grid within shock layer (section 11.1.1)
fstr	fraction of cells to be used in grid stretching region (section 11.1.1)
hrs	time limit for present **laura** run, hr
iabseig	flag for eigenvalue limiter scaling options
iafe	flag for aerobrake geometry options
iaq, jaq, kaq	number of cells in i-, j-, and k-directions, respectively, for **Block A**
iaqf, jaqf	dimensions for cross terms of full Navier-Stokes equations; required for *igovern* = 2 only
iblk, jblk, kblk	number of cells in i-, j-, and k-directions, respectively, for a given block
ic	number of cells used in axial direction for cap of blunted-cone or wedge.
icharge	toggle for electron continuity equation
icrv	flag for equilibrium air curve-fit options
ifrozen	toggle for chemical and thermal source term
igovern	flag for governing equation options
imptemp	toggle for temperature dependence of reaction rates

isjs	working array size (equal to maximum number of cell walls in planes perpendicular to sweep direction) defined as maximum of all blocks
isrf	maximum value for first index of any active wall boundary
issd	toggle for solid-state device (SSD) on CRAY architectures
iterg	maximum number of iterations for this run
itherm	flag for thermal state of nonequilibrium air
itype	boundary type of given side of given block
iunit	flag for geometry unit options
iupwind	flag for total variation diminishing (TVD) limiter options
ivis, jvis, kvis	i-, j-, and k-direction toggles for thin-layer viscous terms (0 = off, 1 = on)
jsrf	maximum value for second index of any active wall boundary
jtype	flag for wall catalysis options
jumpflag	flag for property used to determine shock location
kmodel	flag for kinetic model options
konic	flag for conic geometry options
kstr	number of cells to be used in grid stretching region (section 11.1.1)
lstrt	starting index (in sweep direction) of given partition ($1 \leq lstrt \leq lstop$)
lstop	stopping index (in sweep direction) of given partition ($lstrt \leq lstop \leq LMAX$)
LMAX	number of cells in sweep direction for given block
machine	flag for machine architecture options
mapcpu	task number to which partition is assigned ($1 \leq mapcpu \leq maxcpu$)
maxcpu	maximum number of processors available (hard-coded to $maxcpu = 16$ in file `btask_comm.inc`)
maxi, maxj, maxk	maximum of $iblk_n$, $jblk_n$, and $kblk_n$, respectively ($n = 1, nblocks$); used only for $nturb > 0$
maxmoves	maximum number of grid adjustments for this run
mbk	sweep direction for given partition
movegrd	number of iterations between grid adjustments
mtaska	toggle for multitasking with adaptive partitioning
nbk, nblk	computational block containing given partition
nblocks	number of computational blocks
ndim	flag for flow dimensionality options
ndimb	flag for body dimensionality options
neq	number of governing equations being solved
newjob	flag for initialization options
ngas	flag for gas model options
njcobian	number of iterations between Jacobian updates
nord	flag for governing equation spatial accuracy options
nordbc	flag for boundary condition spatial accuracy options
nprocmx	number of processors available on given machine

nprocs	number of processors available for this LAURA run
nrz	maximum number of reactions defined by any kinetic model in LAURA; currently, $nrz = 26$
ns	total number of active species (N_s) ($1 \leq ns \leq 11$); $ns = 1$ for $ngas \neq 2$
nsp	total number of species groups that serve as collision partners in LAURA; currently, $nsp = 16$
nsrf	total number of solid surfaces in all computational blocks
nsz	total number of species defined in LAURA; currently, $nsz = 11$
ntrnsprt	number of iterations between transport property updates
nturb	flag for turbulence options
prandtl	Prandtl number
recell	cell Reynolds number (section 11.1.1)
refarea	reference area for evaluation of aerodynamic coefficients, in units specified via *iunit*
reflen	reference length for evaluation of aerodynamic coefficients, in units specified via *iunit*
rfinv	inviscid relaxation factor (rf_i)
rfvis	viscous relaxation factor (rf_v)
rflngth	conversion factor, from units specified via *iunit* to meters
ri	species densities, nondimensionalized by ρ_∞
rinfb	free-stream density, kg/m^3
rnose	body nose radius of curvature (R_N), in units specified via *iunit*
rxz	body nose radius of curvature in symmetry plane (R_{xz}), in units specified via *iunit*
str	surface distance from nose to onset of transition ($nturb \neq 0$), in units specified via *iunit*
temp	translational-rotational temperature, K
tempbc	flag for wall temperature boundary condition options
tempv	vibrational-electron-electronic excitation temperature, K
thc	body half angle ($konic = 1$), deg
tinf	free-stream temperature, K
twall	wall temperature, K
u, *v*, *w*	velocity components in x-, y-, and z-directions, respectively, nondimensionalized by $\mathcal{V}_\infty$
uinf, *vinf*, *winf*	u-, v-, and w-components of free-stream velocity
vinfb	free-stream velocity, m/s
v1gas, *v2gas*	coefficients for Sutherland's law
wgas	molecular weight of air, kg/kg-mol
xcg, *ycg*, *zcg*	x-, y-, and z-location of reference center for aerodynamic moments, in units specified via *iunit*
yaw	yaw angle, deg
zmax	body length ($konic = 1$), in units specified via *iunit*

Appendix Q

FORTRAN Flags Changed Through data

Note that the default values are indicated for those parameters that are automatically defined by LAURA as follows:

Eigenvalue limiter scaling options:

$$iabseig = \begin{cases} 0 & \text{(normal limiter)} \\ 1 & \text{(scaled limiter } \{default\}) \end{cases}$$

Toggle for chemical and thermal source term ($ngas = 2$):

$$ifrozen = \begin{cases} 0 & \text{(chemically and thermally frozen flow)} \\ 1 & \text{(nonequilibrium flow } \{default\}) \end{cases}$$

Options for spatial accuracy of governing equations:

$$nord = \begin{cases} 1 & \text{(first-order accuracy)} \\ 2 & \text{(second-order accuracy)} \end{cases}$$

Appendix R

FORTRAN Flags Changed Through stArt

The following FORTRAN flags are changed through stArt:

Aerobrake ($newjob = 2$) geometry options:

$$iafe = \begin{cases} 0 & \text{(Aeroassist Flight Experiment (AFE) aerobrake (ref. 26))} \\ 1 & \text{(hemisphere)} \\ 2 & \text{(customized aerobrake)} \end{cases}$$

Equilibrium air ($ngas = 1$) curve-fit options:

$$icrv = \begin{cases} 1 & \text{(Vinokur in Liu and Vinokur (ref. 19))} \\ 2 & \text{(Tannehill in Srinivasan et al. (ref. 20))} \end{cases}$$

Options for governing equations:

$$igovern = \begin{cases} 0 & \text{(Euler)} \\ 1 & \text{(thin-layer Navier-Stokes (TL N-S))} \\ 2 & \text{(full Navier-Stokes (N-S))} \end{cases}$$

Thermal state of nonequilibrium air ($ngas = 2$):

$$itherm = \begin{cases} 1 & \text{(thermal equilibrium (1-T))} \\ 2 & \text{(thermal nonequilibrium (2-T))} \end{cases}$$

Options for geometry units:

$$iunit = \begin{cases} 0 & \text{(meters)} \\ 1 & \text{(centimeters)} \\ 2 & \text{(feet)} \\ 3 & \text{(inches)} \\ 4 & \text{(other units)} \end{cases}$$

Options for wall catalysis:

$$jtype = \begin{cases} 0 & \text{(non-catalytic)} \\ 1 & \text{("super-catalytic")} \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{cases} \begin{array}{l} \\ \\ \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \text{(catalytic to ions)} \end{array} \begin{cases} \text{(non-catalytic to neutrals)} \\ \left.\begin{array}{l} \text{(Stewart et al. (ref. 21))} \\ \text{(Zoby et al. (ref. 22))} \\ \text{(Scott et al. (ref. 23))} \end{array}\right\} \text{finite catalysis} \\ \text{(recombination of atoms)} \end{cases}$$

Conic options (for axisymmetric body):

$$konic = \begin{cases} 1 & \text{(hyperboloid)} \\ 2 & \text{(paraboloid)} \\ 3 & \text{(ellipsoidally-blunted cone)} \\ 4 & \text{(spherically-blunted cone)} \end{cases}$$

Conic options (for two-dimensional body):

$$konic = \begin{cases} 1 & \text{(hyperbola)} \\ 2 & \text{(parabola)} \\ 3 & \text{(blunted wedge)} \end{cases}$$

Conic options (for three-dimensional body):

$$konic = \begin{cases} 1 & \text{(hyperboloid)} \\ 2 & \text{(paraboloid)} \\ 3 & \text{(blunted cone)} \end{cases}$$

Flow dimensionality options:

$$ndim = \begin{cases} 1 & \text{(axisymmetric)} \\ 2 & \text{(two dimensional)} \\ 3 & \text{(three dimensional)} \end{cases}$$

Body dimensionality options:

$$ndimb = \begin{cases} 1 & \text{(axisymmetric)} \\ 2 & \text{(two dimensional)} \\ 3 & \text{(three dimensional)} \end{cases}$$

Initialization options:

$$newjob = \begin{cases} 0 & \text{(externally generated } \texttt{RESTART.in} \text{ file)} \\ 1 & \text{(flow about conic (cone/wedge, paraboloid))} \\ 2 & \text{(flow about generic aerobrake)} \end{cases}$$

Gas model options:

$$ngas = \begin{cases} 0 & \text{(perfect gas (}\texttt{PG}\text{))} \\ 1 & \text{(equilibrium air (}\texttt{EQ}\text{))} \\ 2 & \text{(chemical nonequilibrium (}\texttt{NONEQ}\text{))} \end{cases}$$

Turbulence options (for $ngas \neq 2$):

$$nturb = \begin{cases} 0 & \text{(none (laminar flow))} \\ 1 & \text{(Cebeci-Smith (in ref. 24) model)} \\ 2 & \text{(Baldwin-Lomax (in ref. 25) model)} \end{cases}$$

Options for wall temperature (T_w) boundary condition:

$$tempbc = \begin{cases} 0 & \text{(constant } T_w\text{)} \\ 1 & \text{(specified } T_w \text{ variation)} \\ 2 & \text{(radiative equilibrium } T_w\text{)} \end{cases}$$

Appendix S

FORTRAN Flags Changed Through File Edits

Note that the file of residence for each parameter is provided. Default values are indicated for those parameters that are automatically defined by LAURA as follows:

Toggle for electron continuity equation ($ngas = 2$), located in `source_vars.strt`:

$$icharge = \begin{cases} 0 & \text{(solution of electron continuity equation)} \\ 1 & (\tilde{n}_e = \sum_{s=\text{ions}} \tilde{n}_s \ \{default\}) \end{cases}$$

Toggle for temperature dependence of reaction rates ($ngas = 2$), located in `source_vars.strt`:

$$impltemp = \begin{cases} 0 & \text{(explicit treatment of dependence)} \\ 1 & \text{(implicit treatment of dependence } \{default\}) \end{cases}$$

Toggle for solid-state device (SSD) on CRAY architectures, located in `issd_assn.strt`:

$$issd = \begin{cases} 0 & \text{(no SSD } \{default\}) \\ 1 & \text{(SSD)} \end{cases}$$

Options for TVD limiter, located in `iupwind_assn.strt`:

$$iupwind = \begin{cases} 0 & \text{(symmetric limiter (eq. (3.8(c)), ref. 2) } \{default\}) \\ 1 & \text{(upwind-biased limiter (ref. 39))} \\ 2 & \text{(symmetric limiter (eq. (3.8(b)), ref. 2))} \end{cases}$$

Options for property used to determine shock location in `algnshk.F` (section 11.1.1), located in `algnshk_vars.strt`:

$$jumpflag = \begin{cases} 0 & \text{(fixed outer boundary)} \\ 1 & \text{(pressure } (p) \ \{default\}) \\ 2 & \text{(density } (\rho)) \\ 3 & \text{(temperature (T))} \end{cases}$$

Kinetic model options (for $ngas = 2$), located in `gas_model_vars.strt`:

$$kmodel = \begin{cases} 1 & \text{(Dunn and Kang (ref. 46) model)} \\ 2 & \text{(Park (ref. 6) model)} \\ 3 & \text{(Park (ref. 40) forward rates; } K_{eq} \text{ from reference 34 } \{default\}) \\ 4 & \text{(Kang and Dunn (ref. 46) forward rates; Gupta } K_{eq} \text{ (ref. 47))} \\ 5 & \text{(Park (ref. 40) forward rates; Gupta et al. } K_{eq} \text{ (ref. 47))} \end{cases}$$

Toggle for multitasking with adaptive partitioning, located in `mtaska_assn.strt`:

$$mtaska = \begin{cases} 0 & \text{(no adaptive partitioning } \{default\}) \\ 1 & \text{(adaptive partitioning)} \end{cases}$$

Options for spatial accuracy of surface and outflow boundary conditions, located in `nordbc_assn.strt`:

$$nordbc = \begin{cases} 1 & \text{(first-order accuracy } \{default\}) \\ 2 & \text{(second-order accuracy)} \end{cases}$$

References

1. Walberg, Gerald D.: A Survey of Aeroassisted Orbit Transfer. *J. Spacecr. & Rockets*, vol. 22, no. 1, Jan.–Feb. 1985, pp. 3–18.

2. Gnoffo, Peter A.: *An Upwind-Biased, Point-Implicit Relaxation Algorithm for Viscous, Compressible Perfect-Gas Flows*. NASA TP-2953, 1990.

3. Gnoffo, Peter A.: *Upwind-Biased, Point-Implicit Relaxation Strategies for Viscous, Hypersonic Flows*. AIAA-89-1972, June 1989.

4. Gnoffo, Peter A.; Gupta, Roop N.; and Shinn, Judy L.: *Conservation Equations and Physical Models for Hypersonic Air Flows in Thermal and Chemical Nonequilibrium*. NASA TP-2867, 1989.

5. Park, Chul: Problems of Rate Chemistry in the Flight Regimes of Aeroassisted Orbital Transfer Vehicles. *Thermal Design of Aeroassisted Orbital Transfer Vehicles*, H. F. Nelson, ed., Volume 96 *Progress in Astronautics and Aeronautics*, AIAA, 1985, pp. 511–537.

6. Park, Chul: Assessment of Two Temperature Kinetic Model for Ionizing Air. AIAA-87-1574, June 1987.

7. Lee, Jong-Hun: Basic Governing Equations for the Flight Regimes of Aeroassisted Orbital Transfer Vehicles. *Thermal Design of Aeroassisted Orbital Transfer Vehicles*, H. F. Nelson, ed., Vol. 96, Progress in Astronautics and Aeronautics, 1984, pp. 3–53.

8. Gnoffo, Peter A.: Code Calibration Program in Support of the Aeroassist Flight Experiment. *J. Spacecr. & Rockets*, vol. 27, no. 2, Mar.–Apr. 1990, pp. 131–142.

9. Walters, Robert W.; Cinnella, Pasquale; Slack, David C.; and Halt, David: Characteristic-Based Algorithms for Flows in Thermochemical Nonequilibrium, *AIAA J.*, vol. 30, no. 5, May 1992, pp. 1304–1313.

10. McGrory, William D.; Huebner, L. D.; Slack, David C.; and Walters, William D.: Development and Application of GASP 2.0. AIAA-92-5067, 1992.

11. McGrory, William D.; Slack, David C.; Applebaum, Michael P.; and Walters, Robert W.: *GASP Version 2.2—The General Aerodynamics Simulation Program*. AeroSoft Inc., 1993.

12. Candler, Graham: On the Computation of Shock Shapes in Nonequilibrium Hypersonic Flows. AIAA-89-0312, Jan. 1989.

13. Candler, Graham V.; and MacCormack, Robert W.: The Computation of Hypersonic Ionized Flows in Chemical and Thermal Nonequilibrium. AIAA-88-0511, Jan. 1988.

14. Park, Chul; and Yoon, Seokkwan: A Fully-Coupled Implicit Method for Thermo-Chemical Nonequilibrium Air at Sub-Orbital Flight Speeds. AIAA-89-1974, June 1989.

15. Netterfield, M. P.: Hypersonic Aerothermodynamic Computations Using a Point-Implicit TVD Method. *Aerothermodynamics for Space Vehicles*, ESA-SP-318, pp. 259–266.

16. Coquel, F.; Joly, V.; Marmignon, C.; and Flament, C.: Numerical Simulation of Thermochemical Non-Equilibrium Viscous Flows Around Reentry Bodies. *Aerothermodynamics for Space Vehicles*, ESA-SP-318, 1991, pp. 447–452.

17. Gnoffo, Peter A.; Hartung, Lin C.; and Greendyke, Robert B.: Heating Analysis for a Lunar Transfer Vehicle at Near-Equilibrium Flow Conditions. AIAA-93-0270, Jan. 1993.

18. Gnoffo, Peter A.: Asynchronous, Macrotasked Relaxation Strategies for the Solution of Viscous, Hypersonic Flows. AIAA-91-1579, June 1991.

19. Liu, Yen; and Vinokur, Marcel: Equilibrium Gas Flow Computations. I. Accurate and Efficient Calculation of Equilibrium Gas Properties. AIAA-89-1736, June 1989.

20. Srinivasan, S.; Tannehill, J. C.; and Weilmuenster, K. J.: *Simplified Curve Fits for the Thermodynamic Properties of Equilibrium Air*. NASA RP-1181, 1987.

21. Stewart, D. A.; Leiser, D. B.; Smith, M.; and Kolodziej, P.: Thermal Response of Integral, Multicomponent Composite Thermal Protection Systems, AIAA-85-1056, June 1985.

22. Zoby, E. V.; Gupta, R. N.; and Simmonds, A. L.: Temperature-Dependent Reaction Rate Expressions for Oxygen Recombination. *Thermal Design of Aeroassisted Orbital Transfer Vehicles*, H. F. Nelson, ed., AIAA, 1985, pp. 445–464.

23. Scott, Carl D.: Catalytic Recombination of Nitrogen and Oxygen on High-Temperature Reusable Surface Insulation. *Aerothermodynamics and Planetary Entry*, A. L. Crosbie, ed., Vol. 77, Progress in Astronautics and Aeronautics, 1980, pp. 192–212.

24. Cebeci, Tuncer: Behavior of Turbulent Flow Near a Porous Wall With Pressure Gradient. *AIAA J.*, vol. 8, no. 12, Dec. 1970, pp. 2152–2156.

25. Baldwin, Barret; and Lomax, Harvard: Thin-Layer Approximation and Algebraic Model for Separated Turbulent Flows. AIAA-78-257, Jan. 1978.

26. Cheatwood, F. McNeil; DeJarnette, Fred R.; and Hamilton, H. Harris, II: *Geometrical Description for a Proposed Aeroassist Flight Experiment Vehicle*. NASA TM-87714, 1986.

27. *Tecplot*™ *– Version 6 User's Manual*. Amteck Engineering Inc., 1993.

28. Gnoffo, Peter A.; Weilmuenster, K. J.; and Alter, Stephen J.: A Multiblock Analysis for Shuttle Orbiter Re-entry Heating From Mach 24 to Mach 12. AIAA-93-2813, July 1993.

29. Hartung, Lin C.: Development of a Nonequilibrium Radiative Heating Prediction Method for Coupled Flowfield Solutions. *J. Thermophys. & Heat Transf.*, vol. 6, no. 4, Oct.–Dec. 1992, pp. 618–625.

30. Chambers, Lin Hartung: *Predicting Radiative Heat Transfer in Thermochemical Nonequilibrium Flow Fields — Theory and User's Manual for the LORAN Code*. NASA TM-4564, 1994.

31. Hartung, Lin C.; Mitcheltree, Robert A.; and Gnoffo, Peter A.: Coupled Radiation Effects in Thermochemical Nonequilibrium Shock-Capturing Flowfield Calculations. AIAA-92-2868, July 1992.

32. Gnoffo, Peter A.: Point-Implicit Relaxation Strategies for Viscous, Hypersonic Flows. *Computational Methods in Hypersonic Aerodynamics*, Mech. Publ./Kluwer Acad. Publ., 1992, pp. 115–151.

33. Thareja, Rajiv R.; Stewart, James R.; Hassan, Obey; Morgan, Ken; and Peraire, Jaime: A Point Implicit Unstructured Grid Solver for the Euler and Navier-Stokes Equations. AIAA-88-0036, Jan. 1988.

34. Park, Chul: *Nonequilibrium Hypersonic Aerothermodynamics*. John Wiley & Sons, Inc., 1990.

35. Harten, Ami: High Resolution Schemes for Hyperbolic Conservation Laws. *J. Comput. Phys.*, vol. 49, no. 2, Feb. 1983, pp. 357–393.

36. Yee, H. C.; Klopfer, G. H.; and Montagné, J.-L.: *High-Resolution Shock-Capturing Schemes for Inviscid and Viscous Hypersonic Flows*. NASA TM-100097, 1988.

37. Yee, H. C.: *On Symmetric and Upwind TVD Schemes*. NASA TM-86842, 1985.

38. Yee, H. C.: Construction of Explicit and Implicit Symmetric TVD Schemes and Their Applications. *J. Comput. Phys.*, vol. 68, no. 1, Jan. 1987, pp. 151–179.

39. Wang, Z.; and Richards, B. E.: High Resolution Schemes for Steady Flow Computation. *J. Comput. Phys.*, vol. 97, Nov. 1991, pp. 53–72.

40. Park, Chul: Review of Chemical-Kinetic Problems of Future NASA Missions. I—Earth Entries. *J. Thermophys. & Heat Transf.*, vol. 7, no. 3, July–Sept. 1993, pp. 385–398.

41. Millikan, Roger C.; and White, Donald R.: Systematics of Vibrational Relaxation, *J. Chem. Phys.*, vol. 39, no. 12, Dec. 15, 1963, pp. 3209–3213.

42. Vinokur, Marcel: *Flux Jacobian Matrices and Generalized Roe Average for an Equilibrium Real Gas*. NASA CR-177512, 1988.

43. Liu, Yen; and Vinokur, Marcel: Upwind Algorithms for General Thermo-Chemical Nonequilibrium Flows. AIAA-89-0201, Jan. 1989.

44. Grossman, B.; and Cinnella, P.: The Development of Flux-Split Algorithms for Flows With Non-Equilibrium Thermodynamics and Chemical Reactions. AIAA-88-3596, July 1988.

45. Gnoffo, Peter A.; McCandless, Ronald S.; and Yee, H. C.: Enhancements to Program LAURA for Computation of Three-Dimensional Hypersonic Flow, AIAA-87-0280, Jan. 1987.

46. Dunn, Michael G.; and Kang, Sang-Wook: *Theoretical and Experimental Studies of Reentry Plasmas*. NASA CR-2232, 1973.

47. Gupta, Roop N.; Yos, Jerrold M.; Thompson, Richard A.; and Lee, Kam-Pui: *A Review of Reaction Rates and Thermodynamic and Transport Properties for an 11-Species Air Model for Chemical and Thermal Nonequilibrium Calculations to 30 000 K*. NASA RP-1232, 1990.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1996	Technical Memorandum

4. TITLE AND SUBTITLE
User's Manual for the Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA)

5. FUNDING NUMBERS
WU 232-01-04-04

6. AUTHOR(S)
F. McNeil Cheatwood and Peter A. Gnoffo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER
L-17419

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TM-4674

11. SUPPLEMENTARY NOTES
Cheatwood: ViGYAN, Inc., Hampton, VA; Gnoffo: Langley Research Center, Hampton, VA.
Research (in part) was supported by the National Aeronautics and Space Administration under Contract NAS1-19237.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category 34
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
This user's manual provides detailed instructions for the installation and the application of version 4.1 of the Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA) (refs. 2 and 3), which is a program for obtaining the simulations discussed above. Earlier versions of LAURA were predominantly research codes, and they had minimal (or no) documentation. This manual describes UNIX-based utilities for customizing the code for special applications that also minimize system resource requirements. The algorithm is reviewed, and the various program options are related to specific equations and variables in the theoretical development.

14. SUBJECT TERMS
Computational fluid dynamics; Hypersonic flow; Thermal nonequilibrium; Chemical nonequilibrium; Upwind schemes; Total variation diminishing

15. NUMBER OF PAGES
223

16. PRICE CODE
A10

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	