12/19/95

SANDIA REPORT

SAND95–1652 • UC–705
Unlimited Release
Printed December 1995

Trajectory Analysis and Optimization System (TAOS) User's Manual

David E. Salguero

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A13
Microfiche copy: A01

SAND95–1652
Unlimited Release
Printed December 1995

Distribution
Category UC–705

Trajectory Analysis and Optimization System (TAOS) User's Manual

(Version 96.0)

David E. Salguero
Aerospace Systems Development Center
Sandia National Laboratories
Albuquerque, NM 87185

Abstract

The Trajectory Analysis and Optimization System (TAOS) is software that simulates point–mass trajectories for multiple vehicles. It expands upon the capabilities of the Trajectory Simulation and Analysis Program (TSAP) developed previously at Sandia National Laboratories. TAOS is designed to be a comprehensive analysis tool capable of analyzing nearly any type of three degree–of–freedom, point–mass trajectory. Trajectories are broken into segments, and within each segment, guidance rules provided by the user control how the trajectory is computed. Parametric optimization provides a powerful method for satisfying mission–planning constraints. Although TAOS is not interactive, its input and output files have been designed for ease of use. When compared to TSAP, the capability to analyze trajectories for more than one vehicle is the primary enhancement, although numerous other small improvements have been made. This report documents the methods used in TAOS as well as the input and output file formats.

Acknowledgements

Over the years many people have contributed to the development of point–mass trajectory analysis software at Sandia. J. L. McDowell recognized the need for general–purpose trajectory and mission planning software and started development of the first point–mass trajectory code. This effort was completed by the author resulting in the Point–Mass Simulation Tool (PMAST). D. E. Outka carried on the effort, which evolved into the Trajectory Simulation and Analysis Program (TSAP), with major contributions involving the equations of motion, guidance, and optimization.

The use of parametric optimization in all of these codes is due to D. G. Hull from the University of Texas at Austin. McDowell, Outka, and the author were all students of Hull, and the trajectory codes reflect his teachings in flight mechanics and optimization.

The author also thanks B. R. Sturgis for independently verifying the mathematical formulation and methods and R. W. Greene for reviewing the document and testing the software.

Contents

List of Figures

List of Tables

Nomenclature

Symbols

$\vec{a}$	acceleration vector
$[A]$	transformation matrix
c	speed of sound
C_A	axial–force coefficient
C_D	drag coefficient
C_L	lift coefficient
C_N	normal–force coefficient
C_S	side–force coefficient
C_X	body x–axis force coefficient
C_Y	body y–axis force coefficient
C_Z	body z–axis force coefficient
e	earth eccentricity
f	earth flatness parameter
$\vec{F}$	force vector
g	acceleration of gravity
G	reference geopotential (9.80665 m/sec^2)
GM	gravity constant
h	altitude
H	geopotential altitude
$[I]$	moment of inertia matrix
J	gravity model zonal harmonics
$[J]$	Jacobian matrix
k	ratio of specific heats (1.4 for air)
L/D	lift–to–drag ratio
m	mass
M	Mach number
M_w	molecular weight
$\vec{M}$	moment vector
N	distance used in geodetic latitude calculations

N'	proportional navigation constant
p	pressure
P	power setting
q	dynamic pressure
r	range
$\vec{r}$	position vector
R	earth radius
R^*	universal gas constant (8314.32 N·m/(kmol·K)
R_N	Reynold's number per foot
$\vec{S}$	state vector
S_{ref}	aerodynamic reference area
S_{noz}	nozzle exit area
t	time
T	temperature
$\hat{u}$	unit vector in ECFC coordinates
U	acceleration of gravity potential function
V_c	closure velocity
$\vec{V}$	velocity vector
x	x coordinate or component
$\tilde{x}$	distance used in geodetic latitude calculations
$\hat{x}$	x–axis unit vector in ECFC coordinates
y	y coordinate or component
$\hat{y}$	y–axis unit vector in ECFC coordinates
z	z coordinate or component
$\hat{z}$	z–axis unit vector in ECFC coordinates
α	angle of attack
α_i	azimuth angle of i^{th} vehicle
α_{iip}	azimuth of the initial impact point
$\alpha_{L/D}$	angle of attack at maximum lift–to–drag ratio
α_r	radar azimuth angle
α_T	total angle of attack
β	sideslip angle

β_c	ballistic coefficient
β_{iip}	ballistic coefficient for the initial impact point calculations
β_E	Euler sideslip angle
γ	vertical flight path angle
δ	latitude
Δ	vector difference or increment
ε_i	elevation angle of i^{th} vehicle
ε_r	radar elevation angle
ε_1	angle between thrust vector and body x axis
ε_2	angle of thrust vector projected into body y–z plane
η_r	aspect angle relative to radar
Θ	pitch angle
λ	longitude
Λ_p	line of sight pitch angle
Λ_y	line of sight yaw angle
μ	bank angle
μ_f	coefficient of friction
ν	kinematic viscosity
ϱ	density
τ	molecular temperature
ϕ	meridional angle
ϕ_r	meridional angle relative to radar
ϕ_w	windward meridian
Φ	roll angle
ψ	horizontal flight path angle (heading)
Ψ	yaw angle
$\vec{\omega}$	angular velocity vector
Ω	rotation angle between ECFC and ECIC coordinate systems

Subscripts

0	initial, first, or sea level
1	temporary coordinate system, vehicle number, or first

2	temporary coordinate system, vehicle number, or second
aero	aerodynamics
b	body coordinate system
f	friction
gc	geocentric coordinate system
gd	geodetic coordinate system
grav	gravity
i	i^{th} vehicle or i^{th} table
iip	initial impact point
I	ECIC coordinate system
n	normal
p	inertial platform coordinates
P	polar
prop	propulsion
r	radar
rail	launch rail
sp	specific or per unit mass
S	surface of earth
tp	tangent plane coordinates
V	velocity coordinate system relative to $\vec{V}_{\oplus}$
V_w	velocity coordinate system relative to $\vec{V}_{w_{\oplus}}$
w	wind coordinate system
x	x component
y	y component
z	z component
$\oplus$	ECFC coordinate system or equatorial

Intentionally Left Blank

1. Introduction

Sandia National Laboratories tests high–speed flight vehicles, such as sounding rockets, reentry bodies, and guided missiles. Figure 1–1 illustrates a typical sounding rocket test flight. These vehicles often travel at hypersonic velocities and at high altitudes. The Trajectory Analysis and Optimization System (TAOS) has been developed to calculate trajectories for this type of high–speed vehicle; however, it can also be used for slower–speed vehicles, such as aircraft and cruise missiles.

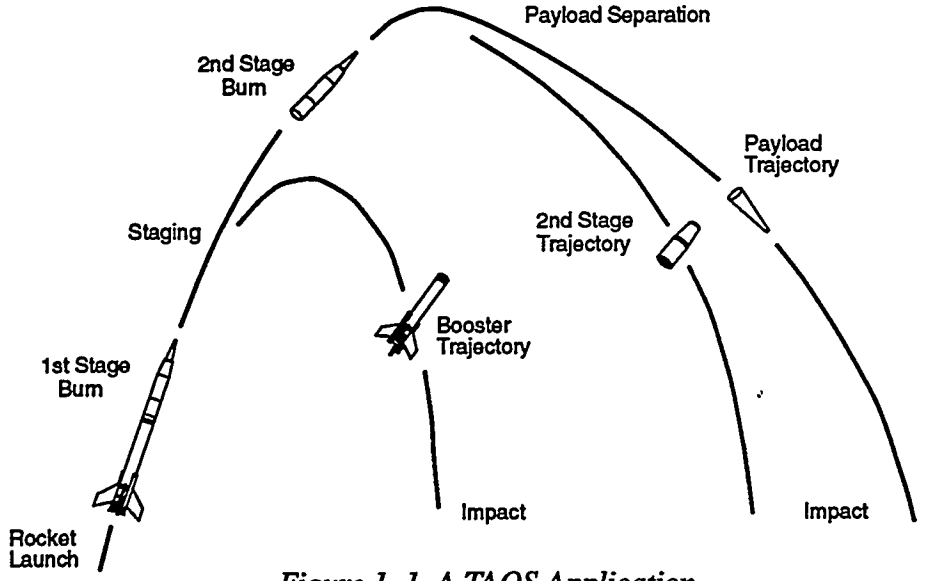

Figure 1–1. A TAOS Application.

TAOS predicts aircraft and missile trajectories from known vehicle characteristics. Given a vehicle's mass, its initial conditions, and the forces acting on it, TAOS computes its trajectory. A vehicle's trajectory is defined by its position, velocity, and acceleration as a function of time. This information is used to determine a vehicle's overall performance characteristics, such as its range, speed, time of flight, and altitude. Figure 1–2 shows typical trajectory information plotted as a function of time.

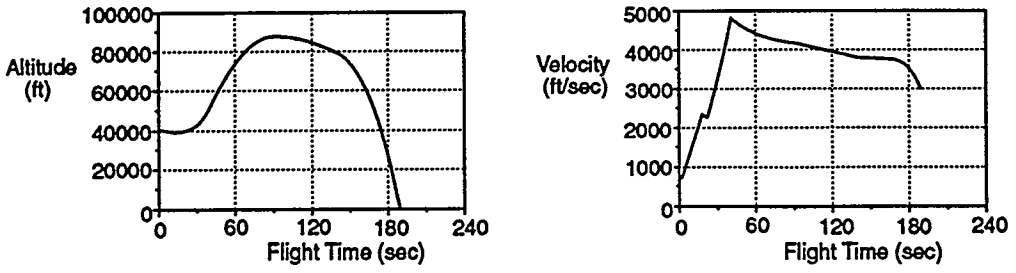

Figure 1–2. Typical Trajectory Information.

1.1. Trajectory Simulation

The motion of an object relative to a fixed point in space is obtained from a set of differential equations called the equations of motion. The equations of motion are based on Newton's second law as applied to rigid bodies of constant mass, that is,

$$\begin{aligned} \Sigma\vec{F} &= m \cdot \vec{a}_I \quad \text{Force Equation} \\ \Sigma\vec{M} &= [\, I \,] \cdot \dot{\vec{\omega}}_I \quad \text{Moment Equation} \end{aligned} \qquad (1\text{–}1)$$

These equations represent a set of second–order differential equations that is integrated with respect to time to compute an object's motion. The position and velocity of the object's center of mass are obtained from the force equation, and the object's angular orientation and rate are obtained from the moment equation. The two inertial acceleration vectors, $\vec{a}_I$ and $\dot{\vec{\omega}}_I$, each have three components, so there are six second–order differential equations, and this system is said to have six degrees of freedom (6–DOF).

TAOS uses a more simplistic approach that assumes the vehicle has a control system (automatic or human) capable of controlling the vehicle's angular orientation. Therefore, the vehicle's angular orientation is assumed to be known; it is an input quantity. This assumption eliminates the moment equation and three degrees of freedom, and it makes the problem easier to solve.

The problem reduces to

$$\Sigma\vec{F} = m \cdot \vec{a}_I \qquad (1\text{–}2)$$

which represents a set of three second–order differential equations for the motion of a point mass. Thus, TAOS is often called a point–mass trajectory simulation or a three degree–of–freedom (3–DOF) simulation. The vehicle's position, velocity, and acceleration depend on the forces acting on it and on its mass. Its angular orientation is input, and it is assumed that a control system is capable of maintaining it. The angular orientation is required to calculate the forces acting on the vehicle.

Although Equation (1–2) is for a constant mass rigid body, it can be used for variable mass bodies such as aircraft and rockets. Mass can change instantaneously, for example, when a weapon is dropped, and mass can change from the propulsive system burning and exhausting fuel.

An instantaneous mass change is not a problem because the trajectory can be divided into a trajectory before the change and a trajectory after the change. However, mass changes from a propulsive system produce propulsive and drag forces. In TAOS these forces must be accounted for in the $\Sigma\vec{F}$ term. An additional mass flow equation is added to the set of differential equations to provide user control over the vehicle's mass.

Over long time periods, control systems generally keep the angular accelerations near zero, which is called trimmed flight. In a point–mass simulation, such as TAOS,

the forces acting on the vehicle must correspond to this trimmed condition. Trim forces used to maintain the vehicle in various body orientations must be included as part of the total force acting on the vehicle.

Point–mass, 3–DOF trajectory simulations can be calculated much faster than full 6–DOF simulations. They also require less information about the vehicle than 6–DOF simulations. This makes them ideal for conceptual design, mission planning, and vehicle performance analysis. They are excellent for predicting trajectory characteristics of long duration and overall system performance.

Point–mass simulations cannot be used to study stability and control problems involving the dynamic response of the vehicle and its control system. They are also poor for trajectories with large angular accelerations.

Details of the mathematical formulation used in TAOS are given in Section 2.

1.2. Development History

The specific methods used in TAOS for flight path prediction have a long history. They began with the development of the Point–Mass Simulation Tool (PMAST).[1] The equations of motion, coordinate transformations, and some preliminary code for PMAST were derived by J. L. McDowell in 1984. This mathematical formulation was enhanced and converted into usable software by the author during 1985. Because of its capabilities and its user interface, PMAST became widely used at Sandia for trajectory prediction.

Parametric optimization methods vary a set of parameters to minimize an objective function subject to constraints. These methods can be applied to trajectory analysis to vary the shape of a trajectory or the timing of trajectory events to meet mission planning and flight path constraints. Parametric optimization was tested in PMAST, but it was unreliable because of singularities in the equations of motion and vehicle attitude definitions. These problems with optimization provided the motivation for the development of a new version of the code.

During 1987 and 1988, the equations of motion and mathematics in PMAST were modified by D. E. Outka resulting in the Trajectory Simulation and Analysis Program (TSAP).[2] The equations of motion in PMAST were derived in a wind coordinate system which is typical for aircraft trajectory simulations. Outka derived the equations of motion for TSAP in an earth–centered, earth–fixed coordinate system to avoid the numerical problems with singularities. This method is commonly used for missile trajectories. Outka used unit vectors for coordinate transforms rather than transformation matrices, which simplified parts of the code, and he also improved the reliability and efficiency of the optimization procedure. The TSAP user interface remained nearly identical to PMAST. TSAP quickly became the standard point–mass trajectory simulation software at Sandia.

From 1988 to 1993 capabilities were added to TSAP as required to support flight test projects at Sandia, and it was successfully used on many projects. However, problems involving multiple vehicles were awkward to solve with TSAP. Furthermore, TSAP was not easy to adapt to multiple vehicles because of the way it had been programmed, so a decision was made in 1993 to develop a new multi–vehicle, point–mass trajectory code.

The result of this effort is TAOS. The mathematical formulation in TAOS is nearly identical to TSAP. It has been proven over time, and there is no need to change it. The primary change from TSAP to TAOS is the extension to multiple vehicles. This is made possible by writing TAOS in the C programming language rather than Fortran. C provides data structures and dynamic memory allocation that make it easy to extend from a single vehicle to multiple vehicles.

Another difference between TSAP and TAOS is the user interface. The style is the same, but numerous changes were made to support the multi–vehicle capability, to increase flexibility, and to make it easier to use.

Numerical methods have been improved in TAOS often requiring less computer processing time than TSAP. Run time comparisons between the two programs vary greatly and are highly dependent on the problem. Single trajectories with no optimization require anywhere from about the same amount of processing time to 20 percent less time with TAOS than with TSAP. Long trajectories with optimization can run as much as 50 to 60 percent faster with TAOS.

TAOS is designed to run on UNIX systems; however, it is written in standard ANSI–C, so it can be run on any computer system with a 32–bit C compiler. For example, it has been tested on an IBM personal computer running Windows NT with the Borland C/C++ compiler. It does require a 32–bit operating system and compiler, such as Windows NT or VAX/VMS.

1.3. Program Capabilities

TAOS is a general purpose computer program that can simulate many different types of trajectories. One way it does this is to break the trajectory into pieces, such that each piece is relatively simple to describe. These pieces are called trajectory segments.

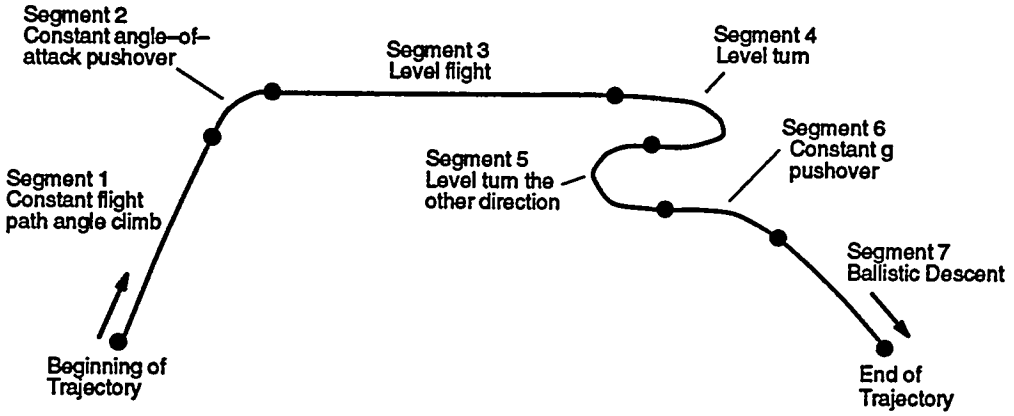

Figure 1–3. Segmented Trajectories.

Figure 1–3 shows a trajectory that has been divided into seven segments. Within each segment, the vehicle configuration and guidance rules stay the same. When either of these changes, a new segment is defined. The trajectories for each segment are combined to form the complete trajectory.

Guidance rules, for example "fly level" or "fly at a constant angle of attack," are used to determine the vehicle's angular orientation. Many guidance rules are available. Some directly specify the angular orientation of the vehicle, such as "fly a constant pitch angle," while others indirectly specify the angular orientation, such as "fly a level turn." These rules are input for each segment telling TAOS how to calculate the segment trajectory.

Because the guidance rules change instantaneously between segments, trajectories can have discontinuities in the vehicle angular orientation between segments. An assumption of this approach is that these discontinuities are small and that the control system can nearly instantaneously change from one orientation to another. This assumption is reasonable for many flight vehicles, such as missiles, aircraft, and reentry bodies.

Multiple trajectories are used to represent more than one vehicle or object. These are used to track spent boosters or deployed objects and to compute intercept trajectories. Figure 1–4 contains a plot with two trajectories that intercept.

During conceptual design studies, sets of trajectories are often computed where one or more of the input parameters is systematically varied. These trajectories are then

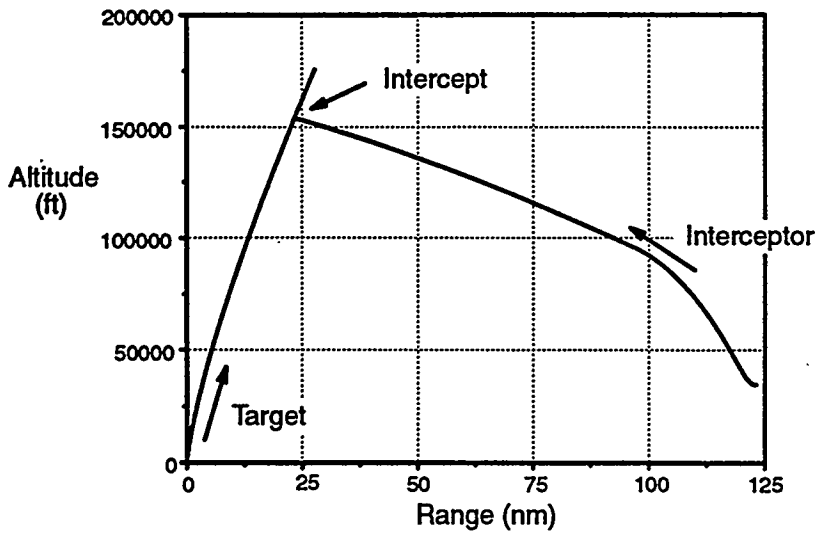

Figure 1–4. A Multiple Trajectory Example.

compared to study the effects of the trajectory parameters. TAOS provides a survey capability that can be used for these tradeoff studies.

Parameter Optimization

Mission planning and conceptual design studies often require trajectories satisfying many constraints. Parameter optimization can be used to vary a selected number of input values to minimize an objective function and to solve for these constraints. Optimization is the most powerful feature in TAOS and gives the software great flexibility to solve many types of problems.

TAOS uses a parameter optimization method called *vf02* originally developed by Powell.[3] This method, one of several recursive quadratic programming methods, solves the general nonlinear programming problem where a set of parameters is varied to minimize an objective function subject to equality and inequality constraints.

Typical optimization parameters are trajectory initial conditions, guidance rules, and segment final conditions. These parameters are varied to minimize or maximize an objective function, such as time, range, or altitude, subject to constraints.

Simple constraints specify a value or limit for a trajectory variable, such as a final altitude, velocity, range, or flight path angle. More complex constraints relate values between two trajectories. For example, an altitude in one trajectory can be constrained to be equal to an altitude in another trajectory. Optimization is used to calculate the intercept trajectory in Figure 1–4, and constraints are used to force the two trajectories to intercept.

Although most constraints apply to specific values at the end of segments, constraints can be set up that apply to the entire trajectory. For example, constraints can be used to keep the trajectory within a maximum altitude limit or within a minimum dynamic pressure limit.

1.4. Table and Problem Input Files

The current version of TAOS is a noninteractive computer program. All input data resides in one or more files created before running TAOS. As TAOS runs, it creates one or more output files. These output files cannot be inspected until TAOS has completed execution.

All of the input and output files are text files. They can be created, inspected, and modified with any text editor or word processor. On UNIX systems the standard text editor is *vi*; however, most UNIX workstations provide a window–based editor that is much easier to use. For example, the *jot* editor is available on Silicon Graphics workstations.

TAOS uses two types of input files: table files containing data such as thrust, mass flow, and aerodynamic coefficients, and problem files containing trajectory definition data. Table filenames are of type *.tbl*, and problem filenames are of type *.prb*.

Use of these file types is required; TAOS keys on the file type when processing the input files. Any number of table and problem files can be input; however, usually one or two table files are input, and one problem file is input.

file.tbl ⇒ Table File { Aero, Propulsion }

file.prb ⇒ Problem File { Initial & Final Conditions, Trajectory Description, Printout Control }

Table files are used to define the forces acting on the vehicle and other quantities that are complex functions of the vehicle's state. They contain information that is generally constant for a given vehicle, so the same table files are used for many different trajectories. These files are often large and complex, but they are seldom modified. Details of the table file format are given in Section 3.

Problem files contain information that describes how to compute the trajectories, for example, initial conditions, trajectory segment definitions, and output descriptions for each trajectory. Problem files are generally small and their contents change often. The problem file format is given in Section 4.

TAOS requires a problem file to run; the table files are optional. Constant aerodynamic and propulsive forces can be defined within a problem file so a table file is not always required.

1.5. Program Execution

TAOS is executed by entering a command from the UNIX shell or from DOS. On UNIX systems this command is entered from a terminal or a terminal window. On PC systems it is entered from a DOS window. The command has the form:

```
taos  file1.tbl  file2.tbl  ...  filex.prb
```

The program name, *taos*, is given first. This is the filename of the program, so it may require directory path names to fully specify it depending on the computer system. For example, the command

```
/usr/public/taos  file1.tbl  file2.tbl  ...  filex.prb
```

runs the program in the file */usr/public/taos*. The remainder of the command is a list of table and problem filenames separated by spaces. TAOS reads all of the table files first, and then it executes the problem files in the order given.

On the Silicon Graphics workstations used in the Aerospace Systems Development Center at Sandia, an icon has been set up for TAOS, as shown in Figure 1–5. TAOS can be executed by selecting all input table and problem files, dragging them to the TAOS icon, and dropping them on the icon. This submits a batch job that runs TAOS with the selected input files. The status of the batch job can be obtained by double clicking on the TAOS icon which opens a window containing a list of all batch jobs in the system. The window is updated every 5 seconds so it can be left open to continuously monitor job status.

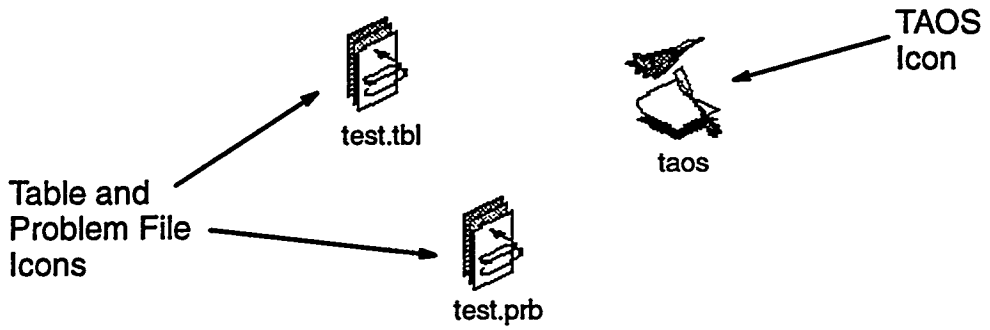

Figure 1–5. TAOS Icon on Silicon Graphics Workstations.

As TAOS calculates each trajectory, it saves all output data in memory. This can require large amounts of memory (4 to 16 Mb) depending on the length of the trajectory, the integration step size, and the print interval. Memory requirements may limit the trajectories that can be calculated on smaller PC systems. UNIX workstations generally do not have this limitation.

1.6. Output Files

TAOS automatically creates printout files. Printout files have the same filename as the input problem files except that the file type is *.out*. A separate printout file is created for each input problem file.

The printout file always contains a listing of the problem file and error messages. The remainder of the printout file is controlled by information in the problem file, such as a list of the variables to be printed and the print time interval. An example page from a printout file is shown in Figure 1–6. The column headings are names of output variables which are defined in the appendix.

```
Sandia National                          Trajectory Analysis & Optimization Software
Laboratories                                        (TAOS - Version 96.0)
-------------------------------------------------------------------------------------------

        Example of a ballistic reentry-body trajectory

        Problem (ballistic) / 1976 US Standard Atmosphere / Non-rotating WGS-84 Earth

        Trajectory for vehicle:  rv

  Time          Alt         Range         Long          Mach          Vel         Gamgd

 0.000     300000.0         0.000     20.00000       19.9642      18000.00      -25.000
 0.500     296194.1         1.324     20.02202       19.9715      18006.57      -25.023
 1.000     292383.5         2.648     20.04405       19.9788      18013.14      -25.046
 1.500     288568.3         3.973     20.06609       19.9861      18019.71      -25.069
 2.000     284748.3         5.298     20.08814       19.9933      18026.27      -25.092
 2.500     280923.7         6.623     20.11019       20.0006      18032.82      -25.115
 3.000     277094.4         7.949     20.13225       19.9008      18039.37      -25.138
 3.500     273260.5         9.276     20.15432       19.7897      18045.91      -25.161
 4.000     269421.8        10.603     20.17640       19.6805      18052.43      -25.184
 4.500     265578.5        11.930.    20.19848       19.5731      18058.93      -25.207
 5.000     261730.6        13.258     20.22057       19.4673      18065.42      -25.230
 5.500     257878.0        14.587     20.24267       19.3632      18071.89      -25.253
 6.000     254020.7        15.915     20.26478       19.2606      18078.32      -25.276
 6.500     250158.9        17.245     20.28690       19.1597      18084.73      -25.299
```

Figure 1–6. Example Printout.

TAOS also writes status information and error messages to standard output, which is the terminal or window used to enter the command that runs TAOS. On UNIX systems this information can be redirected to a file. When using the icon on Silicon Graphics workstations, the status information is written to the batch job output file.

Two other types of output files can be created: files containing trajectory values formatted for the Engineering Graphics System (EGS),[4] and files containing trajectory values arranged in columns.

EGS can be used to produce presentation–quality plots of the trajectory data. It is available on the Silicon Graphics workstations in the Aerospace Systems Development Center at Sandia. The trajectory plots in this report were created with EGS.

If EGS is not available, trajectory data arranged as columns of numbers in a file can be plotted with a variety of plotting packages. This file format is similar to the standard printout, but it is not formatted with page breaks and headers. This format is also useful for transmitting trajectory information to other people.

1.7. Future Plans

Future versions of TAOS will have the capability to interactively create and edit table and problem files. This will include interactive plotting of the tabulated data in the table files. Its purpose will be to help users avoid input errors.

Similarly, future versions of TAOS will have interactive plotting of trajectory output data. This will include plots of trajectory information during execution, so the trajectory can be observed during computation. This will be useful for optimization problems that take a long time to run because they can be stopped if they are not working correctly.

And finally, as with PMAST and TSAP, new output variables and guidance rules will be added to TAOS as required to support projects at Sandia.

Intentionally Left Blank

2. Methods

As mentioned in the previous section, TAOS uses the same methods as its predecessor TSAP,[2] so the description of the mathematical formulation given in the TSAP manual applies to TAOS as well. However, the mathematical formulation given in the TSAP manual is incomplete. This section includes information in the TSAP manual as well as additional information on the coordinate systems, equations of motion, atmosphere and gravity models, guidance methods, range calculations, searches, and optimization.

Notation

Although a list of symbols is included at the beginning of this report, some of the notation needs additional explanation. The large number of coordinate systems in TAOS is confusing so a notation has been adopted that states the mathematics precisely.

Vectors of arbitrary magnitude, used for position, velocity, and acceleration, are denoted with an arrow as shown below, and the magnitude of these vectors is denoted with double bars. Unit vectors with a magnitude of one are denoted with a caret, and matrices are denoted with square brackets. In summary

$\vec{x}$ = vector of arbitrary magnitude

$\|\vec{x}\|$ = magnitude of vector

$\hat{x}$ = unit vector

$[A]$ = matrix

A right subscript on a vector indicates a coordinate system association. For unit vectors this is the coordinate system the unit vectors represent. For other vectors it is the coordinate system dependence caused by vector differentiation with respect to time. For example, $\vec{V}_{\oplus}$ is the velocity vector relative to an earth–fixed coordinate system.

Left and right subscripts indicate the value is relative to two reference frames, so that ${}_{I}\vec{\omega}_{\oplus}$ is the rotational velocity between the inertial and the earth–fixed coordinate systems. This notation appears on vectors and matrices used to convert from one system to another.

Derivatives with respect to time are denoted with one or two dots above the symbol, for example,

$\dot{h}$ = first derivative of h with respect to time

$\ddot{h}$ = second derivative of h with respect to time

The dot product between two vectors can be interpreted as the projection of one vector onto another. Projecting a vector onto the unit vectors of a coordinate system gives the components of the vector in that coordinate system; thus, vector dot products can be used to denote the components of a vector in a coordinate system. For example,

$\vec{V}_{\oplus} \cdot \hat{x}_{gd}$ = component of earth–fixed velocity in the geodetic coordinate system

$\vec{V}_{\oplus} \cdot \hat{x}_{b}$ = component of earth–fixed velocity in the body coordinate system

Numerical Errors

The equations of motion are purposely written in a nonsingular form in an earth–fixed coordinate system to avoid numerical problems during integration, such as divide by zero. The only two conditions which cause problems are a vehicle at the center of the earth and a vehicle with zero mass. If either of these conditions occurs, the trajectory is ended and an error message is issued.

Although the equations of motion are nonsingular, many of the equations for output variables in TAOS have denominators that can go to zero in special situations creating a divide by zero. If this occurs, TAOS assumes a reasonable value, such as zero, and continues execution.

Program Modules

The TAOS source code is divided into modules containing related functions or subroutines as shown in Table 2–1.

Table 2–1. TAOS Program Modules.

Module	Description
atmos	Model atmospheres
coords	Coordinate systems
derivs	Equations of motion and guidance
errors	Error messages
in_prb	Problem input
in_seg	Trajectory segment input
in_trj	Trajectory input
main	Main program
numeric	Numerical analysis
output	Output files
path	Trajectory calculation & control
search	Search and optimization control
tables	Table input and interpolation
util	Utility functions
vf02	Optimization

Many of these modules, such as *errors*, *in_prb*, *output*, and *tables*, perform input or output functions; these are documented in Sections 3 and 4. The modules containing trajectory calculations, *atmos*, *coords*, *derivs*, *numeric*, *path*, and *search*, are documented in this section.

Units

Internally TAOS uses English units for everything except atmosphere calculations, which are in metric. However, the units of input and output variables can be changed from the default English system, given in the appendix, to the metric system as shown in Section 4.4.13.

The following unit conversion factors are used in TAOS:

0.017453293	degrees per radian
57.29577951	radians per degree
2.204622476	pounds mass per kilogram
0.45359240	kilograms per pound mass
32.1740485	pounds mass per slug
0.03108095	slugs per pound mass
3.280839895	feet per meter
0.3048	meters per foot
6076.1	feet per nautical mile
0.0001645792	nautical miles per foot
5280.0	feet per statute mile
0.0001893939	statute miles per foot
0.224808924	pounds force per Newton
4.448222	newtons per pound force
0.020885434	pounds per square foot per pascal
47.88026	pascals per pounds per square foot
9.806650	acceleration of gravity in meters per second squared

2.1. Coordinate Systems

When mathematically describing the motion of one object relative to others, it is necessary to define one or more reference frames or coordinate systems. The equations describing the object's motion are written in terms of one of these coordinate systems. When dynamics are involved, like in TAOS, the relationship of these coordinate systems to an inertial or nonaccelerating reference frame is also required.

Quantities defining a trajectory, such as position, velocity, and acceleration, are given in terms of these coordinate systems. For example, a position near the earth's surface can be descibed in terms of x, y, and z components from the earth's center, or as longitude, latitude, and altitude. The method selected depends on the problem being solved, so TAOS provides many coordinate systems for representing these quantities.

The following coordinate systems are used in TAOS:

- Earth–centered and fixed, cartesian (ECFC) system (equations of motion, unit vectors).
- Earth–centered, inertial, cartesian (ECIC) system (inertial).
- Geocentric system (spherical earth, longitude, latitude).
- Local geocentric horizon system (spherical earth, flight path angles).
- Geodetic system (ellipsoidal earth, longitude, latitude, altitude).
- Local geodetic horizon system (ellipsoidal earth, flight path angles).
- Body system (Euler angles, body angular orientation).
- Velocity system (bank angle).
- Wind system (aerodynamic angles).
- Inertial platform system (inertial alignment platforms).
- Tangent plane system (range safety, flat earth).

The equations of motion are written in terms of the ECFC coordinate system, so it is the most important in TAOS. All unit vectors are also written in terms of the ECFC system so they can be used for coordinate transforms between the systems.

The following sections precisely define each system, show how the unit vectors are calculated and show the relationships between the systems. All of the functions that calculate unit vectors or perform coordinate transforms are in the *coords* module.

2.1.1. Earth–Centered and Fixed, Cartesian (ECFC) Coordinate System

An ECFC coordinate system is used for the equations of motion; thus, it becomes the primary coordinate system in TAOS. Its origin is at the center of the earth, its x and y axes are in the equatorial plane, and its z axis points through the north pole as shown in Figure 2–1. The earth spins about the z axis. The x axis is aligned with the Greenwich meridian (longitude = 0°), and the y axis is aligned with the 90° meridian (longitude = 90°E). Unit vectors $\hat{x}_\oplus$, $\hat{y}_\oplus$, and $\hat{z}_\oplus$ are defined that are parallel to these axes.

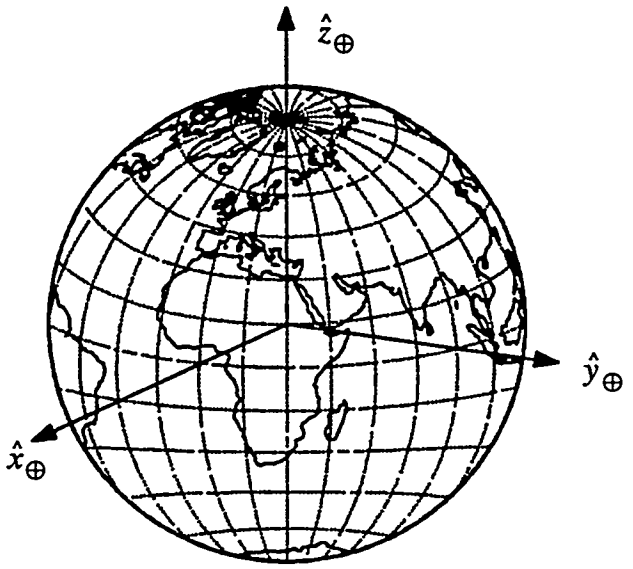

Figure 2–1. The ECFC Coordinate System.

The following output variables give position, velocity, and acceleration vector components in ECFC terms:

Position

xecfc	$\vec{r} \cdot \hat{x}_\oplus$	component of position vector $\vec{r}$ in the ECFC x–axis direction (the vector $\vec{r}$ extends from the earth center to the vehicle's center of mass).
yecfc	$\vec{r} \cdot \hat{y}_\oplus$	component of position vector $\vec{r}$ in the ECFC y–axis direction.
zecfc	$\vec{r} \cdot \hat{z}_\oplus$	component of position vector $\vec{r}$ in the ECFC z–axis direction.

Velocity

xecfcdt	$\vec{V}_{\oplus} \cdot \hat{x}_{\oplus}$	component of velocity vector $\vec{V}_{\oplus}$ in the ECFC x–axis direction (the vector $\vec{V}_{\oplus}$ is the earth–relative velocity of the vehicle's center of mass).
yecfcdt	$\vec{V}_{\oplus} \cdot \hat{y}_{\oplus}$	component of velocity vector $\vec{V}_{\oplus}$ in the ECFC y–axis direction.
zecfcdt	$\vec{V}_{\oplus} \cdot \hat{z}_{\oplus}$	component of velocity vector $\vec{V}_{\oplus}$ in the ECFC z–axis direction.

Acceleration

xecfcdt2	$\vec{a}_{\oplus} \cdot \hat{x}_{\oplus}$	component of acceleration vector $\vec{a}_{\oplus}$ in the ECFC x–axis direction (the vector $\vec{a}_{\oplus}$ is the earth–relative acceleration of the vehicle's center of mass).
yecfcdt2	$\vec{a}_{\oplus} \cdot \hat{y}_{\oplus}$	component of acceleration vector $\vec{a}_{\oplus}$ in the ECFC y–axis direction.
zecfcdt2	$\vec{a}_{\oplus} \cdot \hat{z}_{\oplus}$	component of acceleration vector $\vec{a}_{\oplus}$ in the ECFC z–axis direction.

Because the equations of motions are written in ECFC coordinates, these values are obtained directly from numerical integration as shown in Section 2.2.1.

2.1.2. Earth–Centered, Inertial, Cartesian (ECIC) Coordinate System

The center of the earth is assumed to be nonaccelerating, so an inertial coordinate system can be defined with the earth's center as its origin. The z axis of the inertial system $\hat{z}_I$ is aligned with the earth's spin axis $\hat{z}_\oplus$ so it is fixed with respect to the earth.

The x and y axes of the ECIC and ECFC coordinate systems are related by a rotation about the z axis by the angle

$$\Omega = \Omega_0 + \| {}_I\vec{\omega}_\oplus \| \cdot (t - t_0) \qquad (2\text{–}1)$$

where

$$\| {}_I\vec{\omega}_\oplus \| = {}_I\vec{\omega}_\oplus \cdot \hat{z}_\oplus = \text{Earth's rotation rate} \qquad (2\text{–}2)$$

because the earth's spin axis is aligned with the ECIC and ECFC z axes. The quantity Ω_0 is the initial rotation angle between the ECIC and ECFC coordinate systems at time t_0. The initial rotation angle Ω_0, the time t_0, and the earth's spin rate $\| {}_I\vec{\omega}_\oplus \|$ are input values (Sections 4.3.5 and 4.4.3).

Output variables that give position, velocity, and acceleration vector components in ECIC terms are computed in the function *ecic_coords* and given by

Position

$$\begin{aligned}
&\text{xecic} && \vec{r} \cdot \hat{x}_I = \vec{r} \cdot (\hat{x}_\oplus \cos\Omega - \hat{y}_\oplus \sin\Omega) \\
&\text{yecic} && \vec{r} \cdot \hat{y}_I = \vec{r} \cdot (\hat{x}_\oplus \sin\Omega + \hat{y}_\oplus \cos\Omega) \\
&\text{zecic} && \vec{r} \cdot \hat{z}_I = \vec{r} \cdot \hat{z}_\oplus
\end{aligned} \qquad (2\text{–}3)$$

Velocity

$$\begin{aligned}
&\text{xecicdt} && \vec{V}_I \cdot \hat{x}_I = \vec{V}_\oplus \cdot (\hat{x}_\oplus \cos\Omega - \hat{y}_\oplus \sin\Omega) \\
&&& \qquad - \| {}_I\vec{\omega}_\oplus \| \cdot \vec{r} \cdot (\hat{x}_\oplus \sin\Omega + \hat{y}_\oplus \cos\Omega) \\
&\text{yecicdt} && \vec{V}_I \cdot \hat{y}_I = \vec{V}_\oplus \cdot (\hat{x}_\oplus \sin\Omega + \hat{y}_\oplus \cos\Omega) \\
&&& \qquad + \| {}_I\vec{\omega}_\oplus \| \cdot \vec{r} \cdot (\hat{x}_\oplus \cos\Omega - \hat{y}_\oplus \sin\Omega) \\
&\text{zecicdt} && \vec{V}_I \cdot \hat{z}_I = \vec{V}_\oplus \cdot \hat{z}_\oplus
\end{aligned} \qquad (2\text{–}4)$$

Acceleration

$$\begin{aligned}
&\text{xecicdt2} && \vec{a}_I \cdot \hat{x}_I = \vec{F} \cdot (\hat{x}_\oplus \cos\Omega - \hat{y}_\oplus \sin\Omega) \;/\; m \\
&\text{yecicdt2} && \vec{a}_I \cdot \hat{y}_I = \vec{F} \cdot (\hat{x}_\oplus \sin\Omega + \hat{y}_\oplus \cos\Omega) \;/\; m \\
&\text{zecicdt2} && \vec{a}_I \cdot \hat{z}_I = \vec{F} \cdot \hat{z}_\oplus \;/\; m
\end{aligned} \qquad (2\text{–}5)$$

2.1.3. Local Geocentric Horizon Coordinate System

The local geocentric horizon coordinate system, shown in Figure 2–2, is defined to make it easy to compute the geocentric flight path angles. The origin is located at the vehicle's center of mass and is given by the position vector $\vec{r}$. The z axis is aligned with the position vector $\vec{r}$, but is in the opposite direction so it points towards the center of the earth.

The x and y axes are in a plane that is perpendicular to $\vec{r}$ called the local geocentric horizon plane. This plane is tangent to the earth's surface at the point where $\vec{r}$ intersects the earth's surface (point S in Figure 2–2). The x axis points toward the north pole, and the y axis points east making the y axis parallel to the equator.

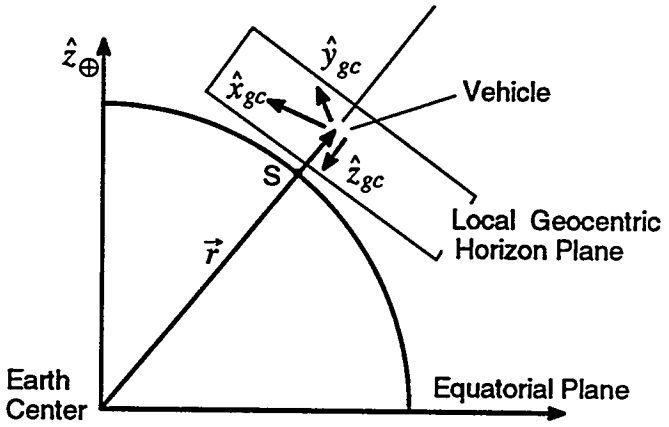

Figure 2–2. Local Geocentric Horizon Coordinates.

The unit vectors for the local geocentric horizon coordinate system, expressed in terms of the ECFC coordinate system, are computed in the function *geoc_unit_vectors* from

$$\begin{aligned} \hat{x}_{gc} &= -\sin\delta_{gc}\,(\cos\lambda\,\hat{x}_\oplus + \sin\lambda\,\hat{y}_\oplus) + \cos\delta_{gc}\,\hat{z}_\oplus \\ \hat{y}_{gc} &= -\sin\lambda\,\hat{x}_\oplus + \cos\lambda\,\hat{y}_\oplus \\ \hat{z}_{gc} &= -\cos\delta_{gc}\,(\cos\lambda\,\hat{x}_\oplus + \sin\lambda\,\hat{y}_\oplus) - \sin\delta_{gc}\,\hat{z}_\oplus \end{aligned} \qquad (2\text{–}6)$$

where δ_{gc} is the geocentric latitude and λ is the longitude (Section 2.1.4).

2.1.4. Geocentric Coordinate System

The geocentric coordinate system has its origin at the center of the earth like the ECFC system, but position and velocity vectors are defined in terms of their magnitude and orientation angles. The orientation angles used for the position vector, longitude λ and latitude δ_{gc}, form a spherical coordinate system at the earth's center as shown in Figure 2–3.

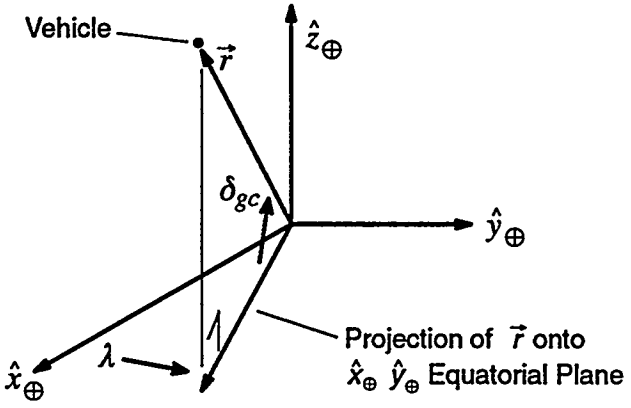

Figure 2–3. Geocentric Position Components.

Geocentric Position

The output variables for the position vector are defined as

rcm	$\|\vec{r}\|$	magnitude of position vector (distance from earth center).
long	λ	longitude, which is defined as the angle between the ECFC x axis and the projection of $\vec{r}$ onto the equatorial plane. The angle is positive when measured counterclockwise from the ECFC x axis or east.
latgc	δ_{gc}	geocentric latitude, which is defined as the angle between $\vec{r}$ and the equatorial plane. It is positive when pointed north.

In function *p_geocentric_to_ecfc*, the following equations relate the geocentric coordinate system to the ECFC system:

$$\begin{aligned} \vec{r} \cdot \hat{x}_\oplus &= \|\vec{r}\| \cos\delta_{gc} \cos\lambda \\ \vec{r} \cdot \hat{y}_\oplus &= \|\vec{r}\| \cos\delta_{gc} \sin\lambda \\ \vec{r} \cdot \hat{z}_\oplus &= \|\vec{r}\| \sin\delta_{gc} \end{aligned} \qquad (2\text{–}7)$$

These relationships are reversed in function *p_ecfc_to_geocentric* so the geocentric coordinates are a function of the ECFC coordinates, that is,

$$\tan\lambda = \frac{\vec{r} \cdot \hat{y}_{\oplus}}{\vec{r} \cdot \hat{x}_{\oplus}} \qquad (2\text{–}8)$$

$$\sin\delta_{gc} = \frac{\vec{r} \cdot \hat{z}_{\oplus}}{\|\vec{r}\|} \qquad (2\text{–}9)$$

Geocentric Velocity

The orientation angles used for the geocentric velocity components are with respect to the local geocentric horizon coordinate system described in Section 2.1.3. The angles, shown in Figure 2–4, are analogous to longitude and latitude except they are relative to the local geocentric horizon system rather than the ECFC system.

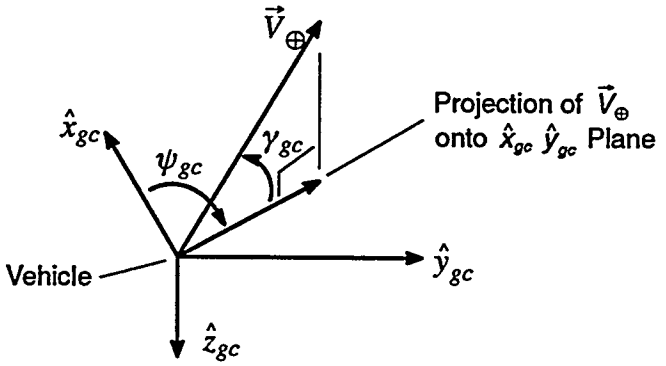

Figure 2–4. Geocentric Velocity Components.

The geocentric output variables defining velocity are

vel	$\|\vec{V}_{\oplus}\|$	magnitude of earth–relative velocity vector.
gamgc	γ_{gc}	geocentric vertical flight path angle, which is the angle between the velocity vector $\vec{V}_{\oplus}$ and the local geocentric horizon plane, that is, 90° minus the angle between $\vec{r}$ and $\vec{V}_{\oplus}$. It is positive when the vehicle is moving away from the earth's surface.
psigc	ψ_{gc}	geocentric horizontal flight path angle, which is the angle between the the local north vector $\hat{x}_{gc}$ and the projection of the velocity vector onto the local geocentric horizon plane. This angle is often called the heading angle and is positive when measured clockwise from north.

From the flight path angle definitions, the velocity vector components in local geocentric horizon coordinates are given by

$$\vec{V}_{\oplus} \cdot \hat{x}_{gc} = \| \vec{V}_{\oplus} \| \cdot \cos\gamma_{gc} \cos\psi_{gc} \quad (2\text{–}10)$$

$$\vec{V}_{\oplus} \cdot \hat{y}_{gc} = \| \vec{V}_{\oplus} \| \cdot \cos\gamma_{gc} \sin\psi_{gc} \quad (2\text{–}11)$$

$$\vec{V}_{\oplus} \cdot \hat{z}_{gc} = -\| \vec{V}_{\oplus} \| \cdot \sin\gamma_{gc} \quad (2\text{–}12)$$

These equations can be solved for the geocentric flight path angles in terms of the ECFC coordinate system resulting in

$$\sin\gamma_{gc} = \frac{-\vec{V}_{\oplus} \cdot \hat{z}_{gc}}{\| \vec{V}_{\oplus} \|} \quad (2\text{–}13)$$

$$\tan\psi_{gc} = \frac{\vec{V}_{\oplus} \cdot \hat{y}_{gc}}{\vec{V}_{\oplus} \cdot \hat{x}_{gc}} \quad (2\text{–}14)$$

The geocentric flight path angles are calculated in the function *v_ecfc_to_geocentric*. The unit vectors $\hat{x}_{gc}$, $\hat{y}_{gc}$, and $\hat{z}_{gc}$ in Equations (2–13) and (2–14) are defined in Equation (2–6). If $\vec{V}_{\oplus} = 0$, the flight path angles are undefined and TAOS sets $\gamma_{gc} = \psi_{gc} = 0$. If the velocity vector is straight up or down ($|\gamma_{gc}| = 90°$), then the heading angle is undefined and TAOS sets it to zero.

The reverse calculation, from geocentric to ECFC, is computed in function *v_geocentric_to_ECFC* from the equations

$$\vec{V}_{\oplus} \cdot \hat{x}_{\oplus} = -(\vec{V}_{\oplus} \cdot \hat{x}_{gc}) \cos\lambda \sin\delta_{gc} - (\vec{V}_{\oplus} \cdot \hat{y}_{gc}) \sin\lambda - (\vec{V}_{\oplus} \cdot \hat{z}_{gc}) \cos\lambda \cos\delta_{gc} \quad (2\text{–}15)$$

$$\vec{V}_{\oplus} \cdot \hat{y}_{\oplus} = -(\vec{V}_{\oplus} \cdot \hat{x}_{gc}) \sin\lambda \sin\delta_{gc} + (\vec{V}_{\oplus} \cdot \hat{y}_{gc}) \cos\lambda - (\vec{V}_{\oplus} \cdot \hat{z}_{gc}) \sin\lambda \cos\delta_{gc} \quad (2\text{–}16)$$

$$\vec{V}_{\oplus} \cdot \hat{z}_{\oplus} = (\vec{V}_{\oplus} \cdot \hat{x}_{gc}) \cos\delta_{gc} - (\vec{V}_{\oplus} \cdot \hat{z}_{gc}) \sin\delta_{gc} \quad (2\text{–}17)$$

where the terms $(\vec{V} \cdot \hat{x}_{gc})$, $(\vec{V} \cdot \hat{y}_{gc})$, and $(\vec{V} \cdot \hat{z}_{gc})$ are from Equations (2–10) through (2–12).

2.1.5. Local Geodetic Horizon Coordinate System

The local geodetic horizon coordinate system is similar to the local geocentric horizon system except that the horizon plane is relative to an ellipsoid representing the earth's surface rather than a sphere. The major axes of this ellipsoid or oblate spheroid are assumed to be in the equatorial plane as shown in Figure 2–5. Its shape is controlled by the equatorial earth radius $R_\oplus$ and one of three redundant terms: the earth polar radius R_p, the ellipsoid eccentricity e, or the ellipsoid flatness f. These parameters are related by

$$R_p = R_\oplus \sqrt{1 - e^2} = R_\oplus \cdot (1 - f) \qquad (2\text{–}18)$$

The eccentricity e and flatness f must be less than one, and the polar radius R_p must be less than the equatorial radius $R_\oplus$. These values, describing the shape of the earth, normally conform to the World Geodetic System WGS–72[5] or WGS–84[6] standard, but they can be changed as necessary as shown in Section 4.4.3.

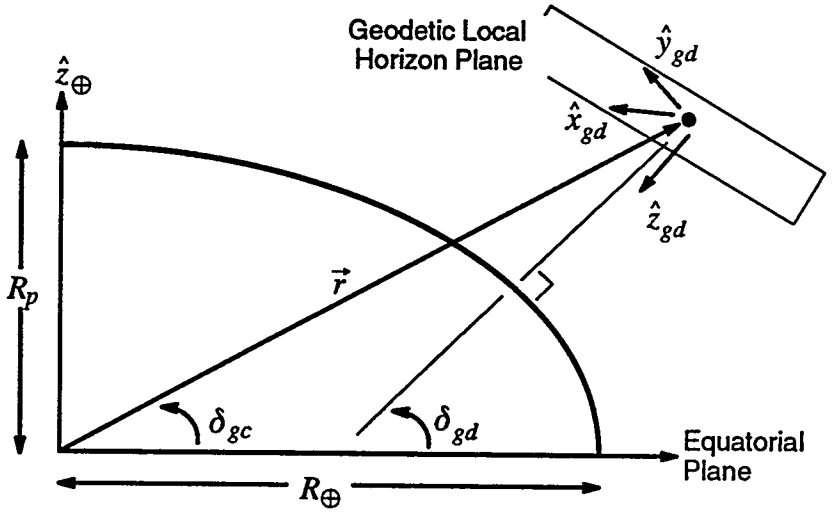

Figure 2–5. Local Geodetic Horizon Coordinates.

The local geodetic horizon coordinate system has its origin at the vehicle's center of mass. Its z axis is directed downward, perpendicular to the surface of the ellipsoid. The geodetic local horizon plane is normal to the z axis, and thus, it is parallel to a plane tangent to the earth's surface.

The x and y axes are located in the geodetic local horizon plane. The x axis is in the plane defined by $\vec{r}$ and $\hat{z}_\oplus$, and it points north. The y axis is perpendicular to the x axis and points east.

A comparison of the local geodetic horizon coordinate system with the local geocentric horizon system shows that the origins are at the same point, the vehicle's center of mass. The unit vectors $\hat{y}_{gc}$ and $\hat{y}_{gd}$ are also identical (parallel to the equator), and the $\hat{x}_{gc}$ and $\hat{z}_{gc}$ unit vectors are rotated about $\hat{y}_{gc}$ by an angle $\delta_{gd} - \delta_{gc}$.

The unit vectors for the local geodetic horizon coordinate system are given by

$$\hat{x}_{gd} = -\sin\delta_{gd}\,(\cos\lambda\,\hat{x}_{\oplus} + \sin\lambda\,\hat{y}_{\oplus}) + \cos\delta_{gd}\,\hat{z}_{\oplus} \qquad (2\text{–}19)$$

$$\hat{y}_{gd} = -\sin\lambda\,\hat{x}_{\oplus} + \cos\lambda\,\hat{y}_{\oplus} \qquad (2\text{–}20)$$

$$\hat{z}_{gd} = -\cos\delta_{gd}\,(\cos\lambda\,\hat{x}_{\oplus} + \sin\lambda\,\hat{y}_{\oplus}) - \sin\delta_{gd}\,\hat{z}_{\oplus} \qquad (2\text{–}21)$$

These are the same as Equations (2–6) except that the *gc* subscripts have been replaced with *gd* subscripts. The value of δ_{gd} required for Equations (2–19) through (2–21) is obtained from the iterative procedure given in Equations (2–34) through (2–41) in Section 2.1.6. The geodetic unit vectors are computed in the function *geod_unit_vectors*.

2.1.6. Geodetic Coordinate System

The geodetic coordinate system is similar to the geocentric system except that the position and velocity vector orientations are relative to the surface of an ellipsoid rather than a sphere. Use of the ellipsoidal earth model is more accurate than the spherical earth, so geodetic coordinates are used more often than geocentric. TAOS uses the approach shown in Reference 7 to handle the ellipsoidal earth geometry.

The earth's surface is modeled by an ellipsoid of revolution, such that in the equatorial plane, defined by $\hat{x}_\oplus$ and $\hat{y}_\oplus$, the ellipsoid forms a circle with a radius $R_\oplus$. The polar radius R_p is assumed to be less than the equatorial radius $R_\oplus$. Thus, points on the earth's surface must satisfy the equation

$$\frac{x_S^2}{R_\oplus^2} + \frac{y_S^2}{R_\oplus^2} + \frac{z_S^2}{R_p^2} = 1 \qquad (2\text{–}22)$$

The ellipsoid is symmetrical about the $\hat{z}_\oplus$ axis so its geometry can be analyzed in the plane formed by $\hat{x}_\oplus$ and $\hat{z}_\oplus$ as shown in Figure 2–6.

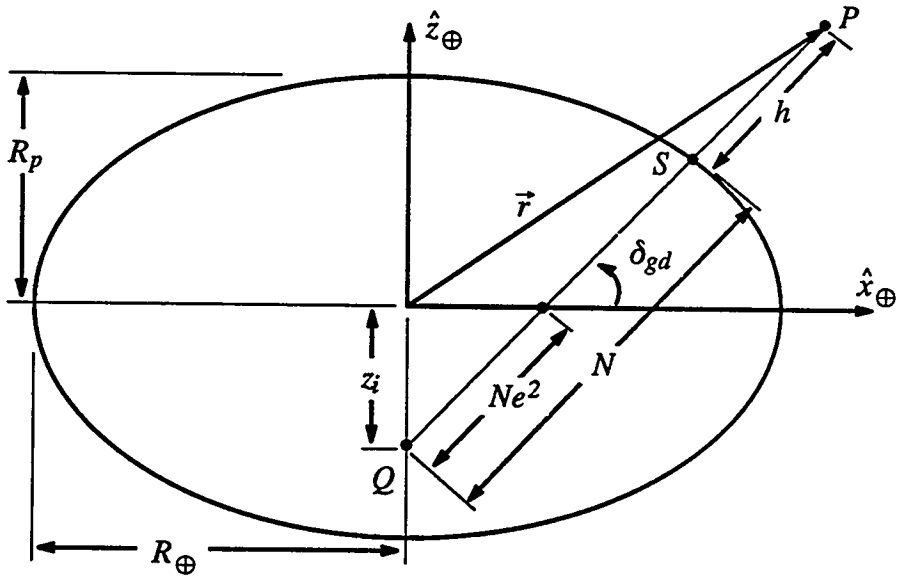

Figure 2–6. Ellipsoidal Earth Geometry.

The equation for the ellipse in the the $\hat{x}_\oplus\hat{z}_\oplus$ plane is

$$R_\oplus^2 = x_S^2 + \frac{z_S^2}{(1 - e^2)} \qquad (2\text{–}23)$$

where e is the eccentricity from Equation (2–18).

Points on the surface of the ellipse, such as point S in Figure 2–6, must satisfy this equation. Lines perpendicular or normal to the ellipse, such as the line between points S and Q, have a slope given by

$$\tan\delta_{gd} = -\frac{dx_S}{dz_S} \tag{2-24}$$

where δ_{gd} is the geodetic latitude. Equation (2–23) can be differentiated and substituted into Equation (2–24) giving

$$\frac{z_S}{x_S} = (1 - e^2)\tan\delta_{gd} \tag{2-25}$$

Equations (2–23) and (2–25) can be solved simulataneously for x_S resulting in

$$x_S = \frac{R_\oplus \cos\delta_{gd}}{\sqrt{1 - e^2\sin^2\delta_{gd}}} \tag{2-26}$$

From Figure 2–6 the coordinates of point S are also given by

$$x_S = N\cos\delta_{gd} \tag{2-27}$$

where N is normal to the ellipsoid at point S and is the distance from point S to the $\hat{z}_\oplus$ axis. From Equations (2–26) and (2–27), N is given by

$$N = \frac{R_\oplus}{\sqrt{1 - e^2\sin^2\delta_{gd}}} \tag{2-28}$$

Substituting the expression for x_S, Equation (2–27), into Equation (2–25) gives an equation for z_S, that is,

$$z_S = (N - e^2N)\sin\delta_{gd} = N(1 - e^2)\sin\delta_{gd} \tag{2-29}$$

This equation shows that the distance between the z–axis intercept of the surface normal, point Q, and its x–axis intercept is Ne^2 as shown in Figure 2–6.

Geodetic Position

These relationships, defining the geometry of the earth's surface as an ellipsoid, are used to locate the vehicle's position relative to the earth's surface. In Figures 2–6 and 2–7 the vehicle is located at point P given by the position vector $\vec{r}$. The point S, in both figures, is on the earth's surface directly below point P.

The output variables associated with the geodetic position vector are

alt	h	altitude, which is the distance from the vehicle's center of mass to the surface of the ellipsoid measured along the surface normal (the distance between points P and S).
long	λ	longitude, which is defined as the angle between the ECFC x axis and the projection of $\vec{r}$ onto the equatorial plane. The angle is positive when measured counter–clockwise from the ECFC x axis or east.
latgd	δ_{gd}	geodetic latitude, which is defined as the angle between $\hat{z}_{gd}$ and the equatorial plane, where $\hat{z}_{gd}$ is normal to the ellipsoid. Latitude is positive when directed north.

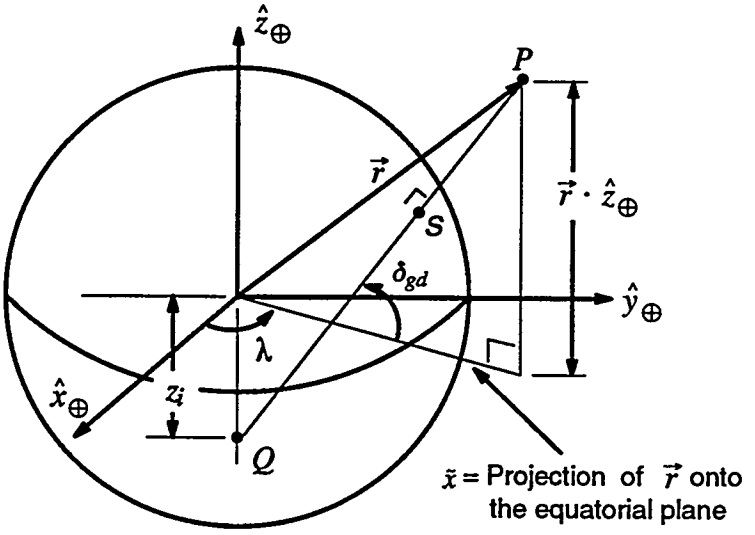

Figure 2–7. Geodetic Coordinate System Geometry.

The longitude is the same for both geocentric and geodetic coordinate systems and is computed from Equation (2–8). The relationship between ECFC coordinates and the remaining geodetic position terms is given by

$$\vec{r} \cdot \hat{x}_\oplus = \tilde{x}\cos\lambda \qquad (2\text{–}30)$$

$$\vec{r} \cdot \hat{y}_\oplus = \tilde{x}\sin\lambda \qquad (2\text{–}31)$$

$$\vec{r} \cdot \hat{z}_\oplus = \left[N \cdot (1 - e^2) + h\right] \cdot \sin\delta_{gd} \qquad (2\text{–}32)$$

$$\text{where } \tilde{x} = (N + h)\cos\delta_{gd} \qquad (2\text{–}33)$$

These equations are used in function *p_geodetic_to_ecfc* to compute the ECFC components of $\vec{r}$ given the geodetic position variables. However, in the reverse situation where the ECFC components are given, these equations cannot be directly solved for the geodetic position variables. An iterative solution is required to obtain geodetic altitude h and latitude δ_{gd} from the ECFC components because of the transcendental nature of δ_{gd} in these equations.

If the $\hat{z}_{gd}$ vector is extended to intercept the ECFC z axis, the intersection occurs at

$$z_i = Ne^2\sin\delta_{gd} = \frac{R_\oplus e^2 \sin\delta_{gd}}{\sqrt{1 - e^2\sin^2\delta_{gd}}} \qquad (2\text{–}34)$$

If the earth is assumed to be nearly spherical so that $e^2 \ll 1$, then

$$\sin\delta_{gd} \approx \sin\delta_{gc} = \frac{\vec{r} \cdot \hat{z}_{\oplus}}{\|\vec{r}\|} \tag{2–35}$$

When $\|\vec{r}\| \ll R_{\oplus}$ this is a poor assumption, but it is adequate to get the iteration started. With this assumption, z_i becomes

$$z_i = R_{\oplus} e^2 \sin\delta_{gc} \tag{2–36}$$

This z_i is only an initial estimate. The following sequence of equations is used repeatedly to refine this estimate, terminating when the difference between z_i values on successive iterations is less than 10^{-8}. Typical applications of this algorithm achieve an accuracy of 0.01 ft in less than three iterations, and in no cases are more than 25 iterations performed.

$$z_d = \vec{r} \cdot \hat{z}_{\oplus} + z_i \tag{2–37}$$

$$N + h = \sqrt{\vec{x}^2 + z_d^2} \tag{2–38}$$

$$\sin\delta_{gd} = \frac{z_d}{N + h} \tag{2–39}$$

$$N = \frac{R_{\oplus}}{\sqrt{1 - e^2 \sin^2\delta_{gd}}} \tag{2–40}$$

$$z_i = N e^2 \sin\delta_{gd} \tag{2–41}$$

The purpose of this iteration is to compute altitude h and geodetic latitude δ_{gd}. Altitude is obtained from Equation (2–38), and geodetic latitude is obtained from Equation (2–39).

Geodetic Velocity

The output variables for the geodetic velocity vector are

vel	$\|\vec{V}_{\oplus}\|$	magnitude of earth–relative velocity vector.
gamgd	γ_{gd}	geodetic vertical flight path angle, which is the angle between the velocity vector $\vec{V}_{\oplus}$ and the local geodetic horizon plane, that is, the angle between $\hat{z}_{gd}$ and $\vec{V}_{\oplus}$ minus 90°. It is positive when the vehicle is moving away from the earth's surface.
psigd	ψ_{gd}	geodetic horizontal flight path angle, which is the angle between the the local north vector $\hat{x}_{gd}$ and the projection of the velocity vector onto the local geodetic horizon plane. This angle is often called the geodetic

heading angle and is positive when measured clockwise from north so that 90° is east.

The magnitude of the earth–relative velocity is the same for both geocentric and geodetic coordinate systems. The geodetic flight path angles are defined in terms of the local geodetic horizon coordinate system as

$$\vec{V}_{\oplus} \cdot \hat{x}_{gd} = \| \vec{V}_{\oplus} \| \cdot \cos\gamma_{gd} \cos\psi_{gd} \qquad (2\text{–}42)$$

$$\vec{V}_{\oplus} \cdot \hat{y}_{gd} = \| \vec{V}_{\oplus} \| \cdot \cos\gamma_{gd} \sin\psi_{gd} \qquad (2\text{–}43)$$

$$\vec{V}_{\oplus} \cdot \hat{z}_{gd} = -\| \vec{V}_{\oplus} \| \cdot \sin\gamma_{gd} \qquad (2\text{–}44)$$

Solving these equations for the flight path angles gives

$$\sin\gamma_{gd} = \frac{-\vec{V}_{\oplus} \cdot \hat{z}_{gd}}{\| \vec{V}_{\oplus} \|} \qquad (2\text{–}45)$$

$$\tan\psi_{gd} = \frac{\vec{V}_{\oplus} \cdot \hat{y}_{gd}}{\vec{V}_{\oplus} \cdot \hat{x}_{gd}} \qquad (2\text{–}46)$$

These equations are used in the function *v_ecfc_to_geodetic* to compute the flight path angles from ECFC coordinates. The geodetic unit vectors are obtained from Equations (2–19) through (2–21). These equations are the same as those used for the geocentric flight path angles except they are referenced to the geodetic local horizon plane rather than the geocentric local horizon plane.

The reverse calculation, from geodetic to ECFC coordinates, is computed in function *v_geodetic_to_ecfc* from

$$\vec{V}_{\oplus} \cdot \hat{x}_{\oplus} = -(\vec{V}_{\oplus} \cdot \hat{x}_{gd}) \cos\lambda \sin\delta_{gd} - (\vec{V}_{\oplus} \cdot \hat{y}_{gd}) \sin\lambda - (\vec{V}_{\oplus} \cdot \hat{z}_{gd}) \cos\lambda \cos\delta_{gd} \qquad (2\text{–}47)$$

$$\vec{V}_{\oplus} \cdot \hat{y}_{\oplus} = -(\vec{V}_{\oplus} \cdot \hat{x}_{gd}) \sin\lambda \sin\delta_{gd} + (\vec{V}_{\oplus} \cdot \hat{y}_{gd}) \cos\lambda - (\vec{V}_{\oplus} \cdot \hat{z}_{gd}) \sin\lambda \cos\delta_{gd} \qquad (2\text{–}48)$$

$$\vec{V}_{\oplus} \cdot \hat{x}_{\oplus} = (\vec{V}_{\oplus} \cdot \hat{x}_{gd}) \cos\delta_{gd} - (\vec{V}_{\oplus} \cdot \hat{z}_{gd}) \sin\delta_{gd} \qquad (2\text{–}49)$$

Again these are the same equations as from geocentric to ECFC, Equations (2–15) through (2–17), except that the subscript *gc* has been replaced with *gd*.

2.1.7. Body–Fixed Cartesian Coordinate System

A body–fixed coordinate system is often used to define components of the aerodynamic and propulsive forces. For example, the aerodynamic coefficients C_A and C_N define one force component along a vehicle's centerline and another normal to it.

The origin of this system is at the vehicle's center of mass, and the x axis is the vehicle's longitudinal axis as shown in Figure 2–8. The y axis is oriented towards the right wing, and the z axis is oriented towards the bottom of the vehicle. The unit vectors $\hat{x}_b$, $\hat{y}_b$, and $\hat{z}_b$ are aligned with these axes.

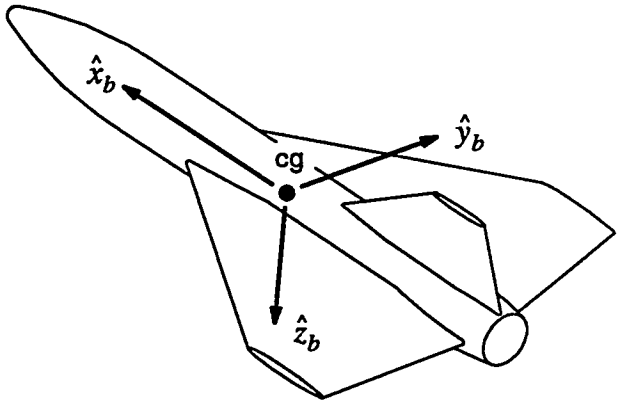

Figure 2–8. Body–Fixed Coordinates.

The body coordinate system is related to the local geodetic horizon coordinate system with three geodetic Euler angles: yaw Ψ_{gd}, pitch Θ_{gd}, and roll Φ_{gd}. The yaw and pitch angles are analogous to the horizontal and vertical flight path angles; the only difference is the vehicle longitudinal axis replaces the velocity vector. The rotation sequence to convert from the local geodetic horizon system to the body system is shown in Figure 2–9.

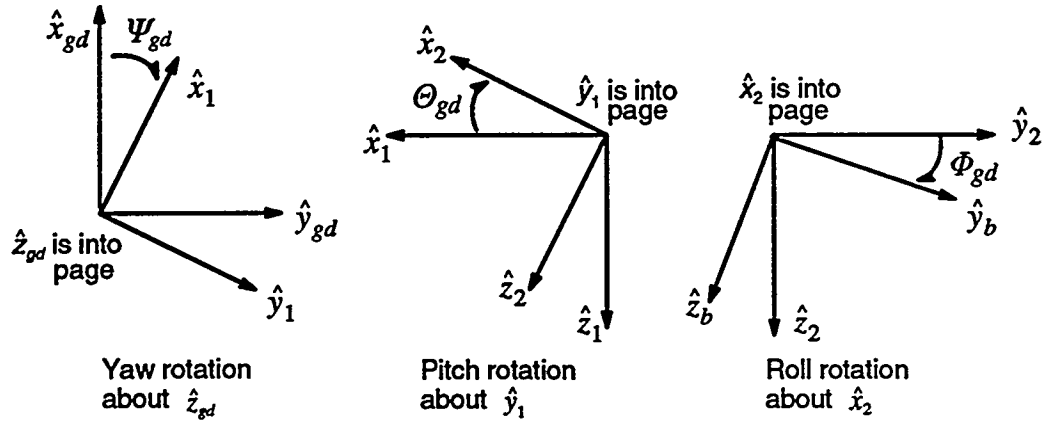

Figure 2–9. Geodetic Yaw, Pitch, and Roll Angles.

The first rotation, a yaw about the $\hat{z}_{gd}$ unit vector produces a temporary coordinate system shown as $\hat{x}_1$, $\hat{y}_1$, and $\hat{z}_1$ in Figure 2–9, where $\hat{z}_1$ and $\hat{z}_{gd}$ are the same vector. The second rotation, by the pitch angle, is about the unit vector $\hat{y}_1$. This forms another temporary coordinate system $\hat{x}_2$, $\hat{y}_2$, and $\hat{z}_2$, where $\hat{y}_2$ and $\hat{y}_1$ are the same. The final rotation, a roll, is about the $\hat{x}_2$ unit vector, and the vectors $\hat{x}_2$ and $\hat{x}_b$ are the same.

The same procedure is used to relate the body coordinate system to the local geocentric horizon system with the geocentric Euler angles, yaw Ψ_{gc}, pitch Θ_{gc}, and roll Φ_{gc}, and to the inertial platform coordinate system (Section 2.1.10) with the inertial platform Euler angles, yaw Ψ_p, pitch Θ_p, and roll Φ_p. These angles are defined as shown in Figure 2–9 except that the initial rotation is from the local geocentric horizon system or inertial platform system rather than the local geodetic horizon system.

These rotations, expressed as transformation matrices, are defined as

$$\begin{bmatrix} \hat{x}_b \\ \hat{y}_b \\ \hat{z}_b \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi_{gd} & \sin\Phi_{gd} \\ 0 & -\sin\Phi_{gd} & \cos\Phi_{gd} \end{bmatrix} \begin{bmatrix} \cos\Theta_{gd} & 0 & -\sin\Theta_{gd} \\ 0 & 1 & 0 \\ \sin\Theta_{gd} & 0 & \cos\Theta_{gd} \end{bmatrix} \begin{bmatrix} \cos\Psi_{gd} & \sin\Psi_{gd} & 0 \\ -\sin\Psi_{gd} & \cos\Psi_{gd} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_{gd} \\ \hat{y}_{gd} \\ \hat{z}_{gd} \end{bmatrix} \quad (2\text{–}50)$$

When expanded, these matrices give the following set of equations for the body unit vectors:

$$\hat{x}_b = \cos\Theta_{gd}\cos\Psi_{gd}\,\hat{x}_{gd} + \cos\Theta_{gd}\sin\Psi_{gd}\,\hat{y}_{gd} - \sin\Theta_{gd}\,\hat{z}_{gd} \quad (2\text{–}51)$$

$$\begin{aligned} \hat{y}_b = {} & (\sin\Phi_{gd}\sin\Theta_{gd}\cos\Psi_{gd} - \cos\Phi_{gd}\sin\Psi_{gd})\,\hat{x}_{gd} \\ & + (\sin\Phi_{gd}\sin\Theta_{gd}\sin\Psi_{gd} + \cos\Phi_{gd}\cos\Psi_{gd})\hat{y}_{gd} \\ & + \sin\Phi_{gd}\cos\Theta_{gd}\,\hat{z}_{gd} \end{aligned} \quad (2\text{–}52)$$

$$\begin{aligned} \hat{z}_b = {} & (\cos\Phi_{gd}\sin\Theta_{gd}\cos\Psi_{gd} + \sin\Phi_{gd}\sin\Psi_{gd})\,\hat{x}_{gd} \\ & + (\cos\Phi_{gd}\sin\Theta_{gd}\sin\Psi_{gd} - \sin\Phi_{gd}\cos\Psi_{gd})\hat{y}_{gd} \\ & + \cos\Phi_{gd}\cos\Theta_{gd}\,\hat{z}_{gd} \end{aligned} \quad (2\text{–}53)$$

The body unit vectors are computed in the function *euler_to_body*. They can be computed from geodetic Euler angles, as shown above, or they can be computed from geocentric or inertial platform Euler angles using similar equations. Equations (2–51) through (2–53) defining the body unit vectors assume the Euler angles are given, but this is not always true.

The body attitude, defined by the body unit vectors, is input directly or indirectly with the guidance rules. Euler angles are not always given in the guidance rules. The body attitude is often given in terms of aerodynamic angles which relate the body and wind coordinate systems. The relationship between the body and wind coordinate systems is given in Section 2.1.9 after the wind coordinate system has been defined.

If the body unit vectors are computed from known aerodynamic angles (Section 2.1.9), the corresponding Euler angles must still be calculated. The Euler angles are defined as a function of the body unit vectors as follows:

$$\cos\Psi_{gd} = \frac{\hat{x}_b \cdot \hat{x}_{gd}}{\sqrt{(\hat{x}_b \cdot \hat{x}_{gd})^2 + (\hat{x}_b \cdot \hat{y}_{gd})^2}} \tag{2–54}$$

$$\sin\Psi_{gd} = \frac{\hat{x}_b \cdot \hat{y}_{gd}}{\sqrt{(\hat{x}_b \cdot \hat{x}_{gd})^2 + (\hat{x}_b \cdot \hat{y}_{gd})^2}} \tag{2–55}$$

$$\cos(\Theta_{gd} + \tfrac{\pi}{2}) = \hat{x}_b \cdot \hat{z}_{gd} \tag{2–56}$$

$$\cos\Phi_{gd} = \hat{y}_1 \cdot \hat{y}_b \tag{2–57}$$

The yaw angle Ψ_{gd} is obtained by projecting the body $\hat{x}_b$ axis onto the local geodetic horizon plane defined by $\hat{x}_{gd}$ and $\hat{y}_{gd}$, and the pitch angle Θ_{gd} is obtained by projecting the body $\hat{x}_b$ axis onto the geodetic $\hat{z}_{gd}$ axis. Similarly, the roll angle Φ_{gd} is obtained by projecting the body $\hat{y}_b$ axis onto the $\hat{y}_1$ axis. This axis is perpendicular to $\hat{x}_1$ and the geodetic $\hat{z}_{gd}$ axis, that is,

$$\hat{y}_1 = \hat{z}_{gd} \times \hat{x}_1 \tag{2–58}$$

where the $\hat{x}_1$ axis is obtained by solving the linear equations

$$\begin{bmatrix} \hat{x}_{gd} \\ \hat{y}_{gd} \\ \hat{z}_{gd} \end{bmatrix} \cdot \hat{x}_1 = \begin{bmatrix} \cos\Psi_{gd} \\ \sin\Psi_{gd} \\ 0 \end{bmatrix} \tag{2–59}$$

The components of the unit vectors form the rows of the transformation matrix. Because this matrix is orthogonal, its inverse is its transpose, and $\hat{x}_1$ is given by

$$\hat{x}_1 = \begin{bmatrix} \hat{x}_{gd} & \hat{y}_{gd} & \hat{z}_{gd} \end{bmatrix} \cdot \begin{bmatrix} \cos\Psi_{gd} \\ \sin\Psi_{gd} \\ 0 \end{bmatrix} \tag{2–60}$$

where the components of the unit vectors now form the columns of the matrix. This method fails when the body is pointed straight up or straight down, that is, when $\hat{x}_b$ is aligned with $\hat{z}_{gd}$ because the yaw angle is undefined. When this occurs, TAOS sets the Euler angles to

$$\Psi_{gd} = \Phi_{gd} = 0$$
$$\Theta_{gd} = \frac{\pi}{2} \cdot (\hat{x}_b \cdot \hat{z}_{gd}) \qquad (2\text{–}61)$$

The equations above show how the geodetic Euler angles are computed from the geodetic and body unit vectors. The same procedure is used to calculate the geocentric and inertial platform Euler angles by substituting the appropriate unit vectors. These calculations occur in the function *euler_angles*.

The following output variables are used for the body coordinate system Euler angles:

yawgc	Ψ_{gc}	geocentric Euler yaw angle.
yawgd	Ψ_{gd}	geodetic Euler yaw angle.
yawi	Ψ_i	inertial platform Euler yaw angle.
pitchgc	Θ_{gc}	geocentric Euler pitch angle.
pitchgd	Θ_{gd}	geodetic Euler pitch angle.
pitchi	Θ_i	inertial platform Euler pitch angle.
rollgc	Φ_{gc}	geocentric Euler roll angle.
rollgd	Φ_{gc}	geodetic Euler roll angle.
rolli	Φ_{gc}	inertial platform Euler roll angle.

The following additional output variables give the vehicle's acceleration and velocity vectors in the body coordinate system:

accebx	$\vec{a}_\oplus \cdot \hat{x}_b$	component of earth–fixed acceleration in the body x–axis direction.
acceby	$\vec{a}_\oplus \cdot \hat{y}_b$	component of earth–fixed acceleration in the body y–axis direction.
accebz	$\vec{a}_\oplus \cdot \hat{z}_b$	component of earth–fixed acceleration in the body z–axis direction.
accibx	$\vec{a}_I \cdot \hat{x}_b$	component of inertial acceleration in the body x–axis direction.
acciby	$\vec{a}_I \cdot \hat{y}_b$	component of inertial acceleration in the body y–axis direction.
accibz	$\vec{a}_I \cdot \hat{z}_b$	component of inertial acceleration in the body z–axis direction.
velebx	$\vec{V}_\oplus \cdot \hat{x}_b$	component of earth–fixed velocity in the body x–axis direction.
veleby	$\vec{V}_\oplus \cdot \hat{y}_b$	component of earth–fixed velocity in the body y–axis direction.

velebz	$\vec{V}_{\oplus} \cdot \hat{z}_b$	component of earth–fixed velocity in the body z–axis direction.
velibx	$\vec{V}_I \cdot \hat{x}_b$	component of inertial velocity in the body x–axis direction.
veliby	$\vec{V}_I \cdot \hat{y}_b$	component of inertial velocity in the body y–axis direction.
velibz	$\vec{V}_I \cdot \hat{z}_b$	component of inertial velocity in the body z–axis direction.

2.1.8. Velocity Cartesian Coordinate Systems

Intermediate coordinate systems, called velocity coordinate systems, are used to convert from the geocentric or geodetic system to the wind coordinate system (See Section 2.1.9). These systems, one for converting from geocentric and one for converting from geodetic, are defined the same way except that one starts from geocentric coordinates and one starts from geodetic coordinates. This section defines the velocity coordinate system relative to the geodetic system; the same method is used relative to the geocentric system except that the *gd* subscripts are replaced with *gc* subscripts.

The origin of the velocity coordinate system is at the vehicle's center of mass. The x axis is aligned with the wind–corrected velocity vector $\vec{V}_{w_\oplus}$ given by

$$\vec{V}_{w_\oplus} = \vec{V}_\oplus - \Delta\vec{V}_{w_\oplus} \qquad (2\text{–}62)$$

where $\Delta\vec{V}_{w_\oplus}$ is the input wind velocity vector relative to the earth's surface (Section 4.4.14). The vector $\vec{V}_{w_\oplus}$ represents the velocity of the vehicle relative to the air or atmosphere. Its magnitude is the output variable

vair $\quad \| \vec{V}_{w_\oplus} \|$ airspeed.

The y axis is perpendicular to the plane formed by the velocity x axis and the geodetic $\hat{z}_{gd}$ axis, and the z axis is perpendicular to the other two axes and directed towards the earth as shown in Figure 2–10.

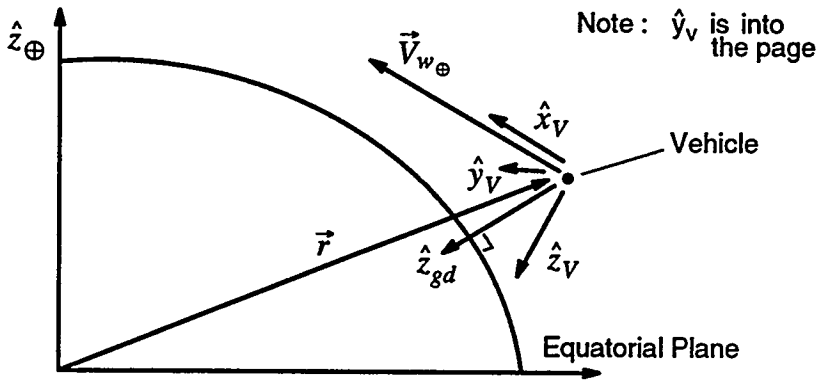

Figure 2–10. Velocity Coordinate System.

The velocity coordinate system unit vectors are computed in the function *wind_unit_vectors* from

$$\hat{x}_{V_w} = \frac{\vec{V}_{w_\oplus}}{\| \vec{V}_{w_\oplus} \|} \qquad (2\text{–}63)$$

$$\hat{y}_{V_w} = \frac{\hat{z}_{gd} \times \hat{x}_{V_w}}{\| \hat{z}_{gd} \times \hat{x}_{V_w} \|} \tag{2-64}$$

$$\hat{z}_{V_w} = \frac{\hat{x}_{V_w} \times \hat{y}_{V_w}}{\| \hat{x}_{V_w} \times \hat{y}_{V_w} \|} \tag{2-65}$$

The velocity coordinate systems shown above are used to define the wind coordinate system (Section 2.1.9). Another similar velocity coordinate system is used to relate the velocity vector to the geodetic coordinate system with flight path angles. This system is defined the same as above except it is aligned with the earth–relative velocity vector $\vec{V}_{\oplus}$ rather than the wind–corrected velocity vector $\vec{V}_{w_{\oplus}}$ so that

$$\hat{x}_V = \frac{\vec{V}_{\oplus}}{\| \vec{V}_{\oplus} \|} \tag{2-66}$$

The geodetic coordinate system is related to this velocity coordinate system by two rotations. The first rotation is about the geodetic z axis by the heading angle ψ_{gd}, and the second rotation is about the y axis by the vertical flight path angle γ_{gd}. The rotation matrices are

$$\begin{bmatrix} \hat{x}_V \\ \hat{y}_V \\ \hat{z}_V \end{bmatrix} = \begin{bmatrix} \cos\gamma_{gd} & 0 & -\sin\gamma_{gd} \\ 0 & 1 & 0 \\ \sin\gamma_{gd} & 0 & \cos\gamma_{gd} \end{bmatrix} \begin{bmatrix} \cos\psi_{gd} & \sin\psi_{gd} & 0 \\ -\sin\psi_{gd} & \cos\psi_{gd} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_{gd} \\ \hat{y}_{gd} \\ \hat{z}_{gd} \end{bmatrix} \tag{2-67}$$

When this is expanded it gives the unit vectors

$$\hat{x}_V = \cos\gamma_{gd}\cos\psi_{gd}\,\hat{x}_{gd} + \cos\gamma_{gd}\sin\psi_{gd}\,\hat{y}_{gd} - \sin\gamma_{gd}\,\hat{z}_{gd} \tag{2-68}$$

$$\hat{y}_V = -\sin\psi_{gd}\,\hat{x}_{gd} + \cos\psi_{gd}\,\hat{y}_{gd} \tag{2-69}$$

$$\hat{z}_V = \sin\gamma_{gd}\cos\psi_{gd}\,\hat{x}_{gd} + \sin\gamma_{gd}\sin\psi_{gd}\,\hat{y}_{gd} + \cos\gamma_{gd}\,\hat{z}_{gd} \tag{2-70}$$

These relations are used to derive the flight path angle rates in Section 2.2.5.

2.1.9. Wind Cartesian Coordinate System

The aerodynamic forces acting on the vehicle are usually functions of aerodynamic angles, such as angle of attack and sideslip. The wind coordinate system relates these aerodynamic angles to the body–fixed coordinate system. Its origin is at the vehicle's center of mass, and its x axis is aligned with the x axis of the velocity coordinate system and $\vec{V}_{w_\oplus}$.

The y and z axes are rotated by the geocentric bank angle μ_{gc} from the geocentric velocity coordinate system or by the geodetic bank angle μ_{gd} from the geodetic velocity coordinate system as shown in Figure 2–11. Although these two bank angles are slightly different to compensate for the differences between the geocentric and geodetic coordinate systems, the result is the same wind system.

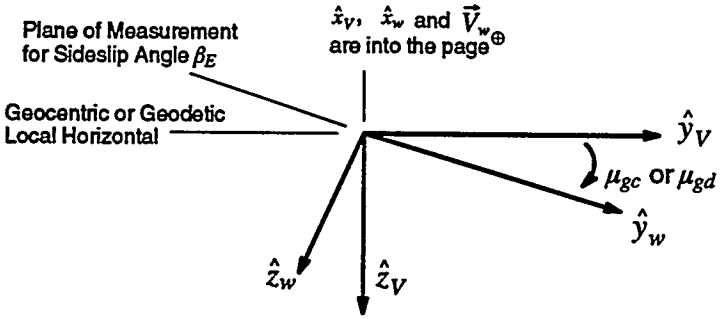

Figure 2–11. Bank Angle and Wind Coordinate System.

The wind coordinate system unit vectors are computed from

$$\begin{aligned} \hat{x}_w &= \hat{x}_V \\ \hat{y}_w &= \cos\mu\ \hat{y}_V + \sin\mu\ \hat{z}_V \\ \hat{z}_w &= -\sin\mu\ \hat{y}_V + \cos\mu\ \hat{z}_V \end{aligned} \qquad (2\text{–}71)$$

in the function *wind_unit_vectors*. The bank angle μ must correspond to the same coordinate system, geocentric or geodetic, as the velocity system unit vectors $\hat{x}_V$, $\hat{y}_V$, and $\hat{z}_V$ (Section 2.1.8).

The wind and body–fixed coordinate systems are related through aerodynamic angles as shown in Figures 2–12, 2–13, and 2–15. The following pairs of aerodynamic angles are used to describe the vehicle orientation relative to the wind coordinate system: angle of attack α and Euler sideslip angle β_E, angle of attack α and sideslip angle β, and total angle of attack α_T and windward meridian ϕ_w.

Angle of Attack and Euler Sideslip Angle

Figure 2–12 shows that a rotation about the body $-\hat{y}_b$ axis by the angle of attack α, followed by a rotation about the wind $\hat{z}_w$ axis by the Euler sideslip angle β_E results

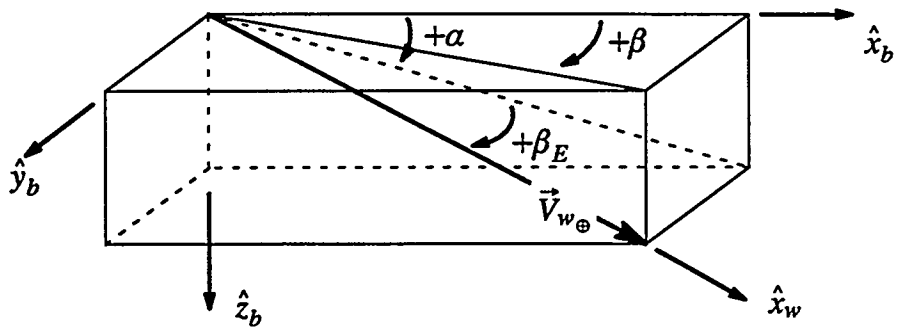

Figure 2–12. Angle of Attack and Sideslip Definitions.

in alignment with the wind axis. This definition of sideslip angle β_E, using rotations, is typically used for aircraft applications. Generally angle of attack values range from –180° to +180° and Euler sideslip values range from –90° to +90°, but the angles can take on any value because they are defined by rotations.

The body unit vectors are related to the wind unit vectors with the following rotation matrices:

$$\begin{bmatrix} \hat{x}_b \\ \hat{y}_b \\ \hat{z}_b \end{bmatrix} = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} \cos\beta_E & -\sin\beta_E & 0 \\ \sin\beta_E & \cos\beta_E & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \hat{x}_w \\ \hat{y}_w \\ \hat{z}_w \end{bmatrix} \qquad (2\text{–}72)$$

When expanded, this results in

$$\hat{x}_b = \cos\alpha\cos\beta_E\,\hat{x}_w - \cos\alpha\sin\beta_E\,\hat{y}_w - \sin\alpha\,\hat{z}_w \qquad (2\text{–}73)$$

$$\hat{y}_b = \sin\beta_E\,\hat{x}_w + \cos\beta_E\,\hat{y}_w \qquad (2\text{–}74)$$

$$\hat{z}_b = \sin\alpha\cos\beta_E\,\hat{x}_w - \sin\alpha\sin\beta_E\,\hat{y}_w + \cos\alpha\,\hat{z}_w \qquad (2\text{–}75)$$

Angle of Attack and Sideslip Angle

The other sideslip angle β is defined as the angle between $\hat{x}_b$ and the projection of the velocity vector $\vec{V}_{w_\oplus}$ onto the $\hat{x}_b\,\hat{y}_b$ plane. When this sideslip angle is used, angle of attack is defined as the angle between $\hat{x}_b$ and the projection of the velocity vector $\vec{V}_{w_\oplus}$ onto the $\hat{x}_b\,\hat{z}_b$ plane, that is,

$$\tan\alpha = \frac{\hat{x}_w \cdot \hat{z}_b}{\hat{x}_w \cdot \hat{x}_b} \qquad (2\text{–}76)$$

$$\tan\beta = \frac{\hat{x}_w \cdot \hat{y}_b}{\hat{x}_w \cdot \hat{x}_b} \qquad (2\text{–}77)$$

These aerodynamic angle definitions are commonly used for axisymmetric vehicles, such as reentry bodies and artillery shells, and they are associated with the total angle of attack and windward meridian. The angles are defined by projections instead of

rotations so not all combinations of angle of attack and sideslip are valid. There are three cases to consider: $\hat{x}_w \cdot \hat{x}_b > 0$, $\hat{x}_w \cdot \hat{x}_b < 0$, and $\hat{x}_w \cdot \hat{x}_b = 0$.

The first case with $\hat{x}_w \cdot \hat{x}_b > 0$, illustrated in Figure 2–12, results in both angle of attack and sideslip values between –90° and +90°, that is, $0° < |\alpha| < 90°$ and $0° < |\beta| < 90°$. Since $\hat{x}_w \cdot \hat{x}_b > 0$, the vehicle can be thought of as moving in a forward direction.

The second case with $\hat{x}_w \cdot \hat{x}_b < 0$, illustrated in Figure 2–13, results in $90° < |\alpha| < 180°$ and $90° < |\beta| < 180°$. In this case, the vehicle is moving in a backward direction.

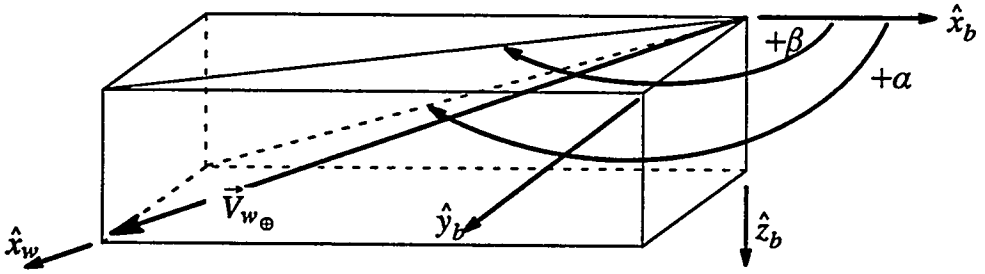

Figure 2–13. Angle of Attack and Sideslip Definitions.

With this definition angles of attack greater than 90° are not consistent with sideslip angles less than 90°. Both angle of attack and sideslip are defined with the same value of $\hat{x}_w \cdot \hat{x}_b$ restricting them to the values given above.

The third case with $\hat{x}_w \cdot \hat{x}_b = 0$ puts the wind vector in the $\hat{y}_b\,\hat{z}_b$ plane, and all such wind vectors according to Equations (2–76) and (2–77) have $\alpha = \pm 90°$ and $\beta = \pm 90°$. A wind vector in the $\hat{y}_b\,\hat{z}_b$ plane cannot be uniquely defined with these angle definitions so TAOS uses a special definition shown in Figure 2–14.

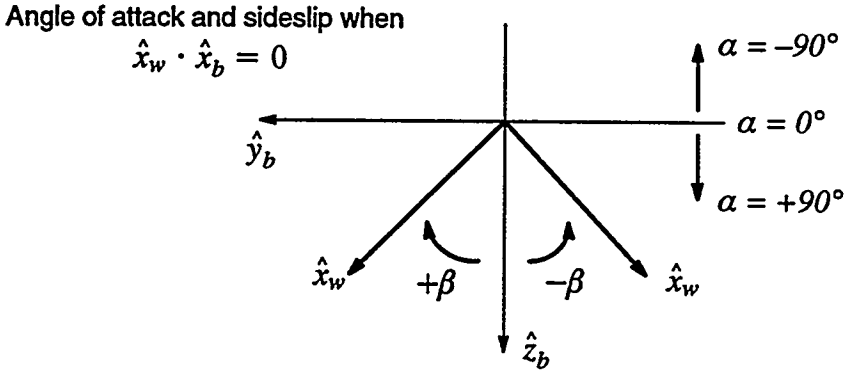

Figure 2–14. Aerodynamic Angles at 90° Angle of Attack.

When $\hat{x}_w \cdot \hat{x}_b = 0$, angle of attack is still given by Equation (2–76), but sideslip becomes

$$\tan\beta = \frac{\hat{x}_w \cdot \hat{y}_b}{\hat{x}_w \cdot \hat{z}_b} \qquad (2\text{–}78)$$

This definition is similar to the windward meridian. Because the basic angle definitions do not work for this case, use of angle of attack and sideslip are not recommended for trajectories with the wind vector in the $\hat{y}_b\ \hat{z}_b$ plane. Angle of attack and Euler sideslip or total angle of attack and windward meridian are recommended instead.

Total Angle of Attack and Windward Meridian

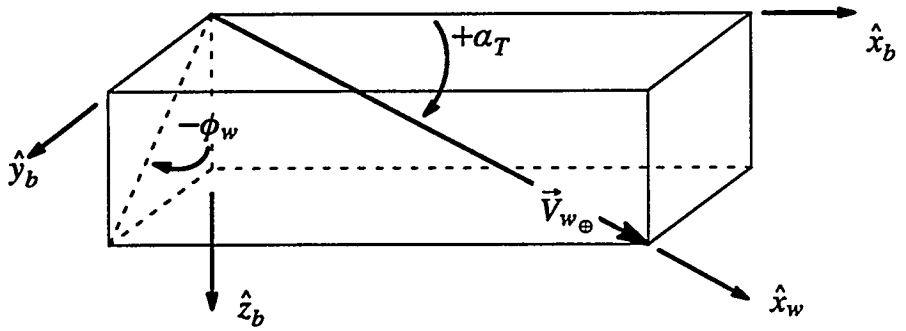

Figure 2–15. Total Angle of Attack and Windward Meridian Definitions.

The total angle of attack and windward meridian are defined in Figure 2–15. The total angle of attack α_T is the angle between the body $\hat{x}_b$ axis and the wind $\hat{x}_w$ axis, and the windward meridian ϕ_w is the angle between the body $\hat{z}_b$ axis and the projection of the velocity vector $\vec{V}_{w_\oplus}$ onto the $\hat{y}_b\ \hat{z}_b$ plane, that is,

$$\tan\alpha_T = \frac{\sqrt{(\hat{x}_w \cdot \hat{y}_b)^2 + (\hat{x}_w \cdot \hat{z}_b)^2}}{\hat{x}_w \cdot \hat{x}_b} \qquad (2\text{–}79)$$

$$\tan\phi_w = \frac{-\hat{x}_w \cdot \hat{y}_b}{\hat{x}_w \cdot \hat{z}_b} \qquad (2\text{–}80)$$

The total angle of attack is always positive and ranges from 0° to 180°. The windward meridian can take on any value, but it is generally given as an angle between 0° and 360°.

Aerodynamic Angle Relationships

The guidance rules in TAOS provide values for a pair of aerodynamic angles, and the remaining aerodynamic angles are computed from this pair of values. Relationships to convert from one angle pair to another can be derived from the angle definitions and the relationship of the angles to the wind coordinate system (Equations (2–73) through (2–80)).

If angle of attack α and Euler sideslip angle β_E are known, total angle of attack α_T, windward meridian ϕ_w, and sideslip angle β are obtained from

$$\tan\alpha_T = \frac{\sqrt{\sin^2\beta_E + \cos^2\beta_E \sin^2\alpha}}{\cos\alpha\cos\beta_E} \tag{2–81}$$

$$\tan\phi_w = \frac{-\sin\beta_E}{\sin\alpha\cos\beta_E} \tag{2–82}$$

$$\tan\beta = \frac{\sin\beta_E}{\cos\alpha\cos\beta_E} \tag{2–83}$$

For the special case when $\alpha = \pm 90°$, $\beta = -\phi_w$.

If angle of attack α and sideslip angle β are known, total angle of attack α_T, windward meridian ϕ_w, and Euler sideslip angle β_E are obtained from

$$\tan\alpha_T = \sqrt{\tan^2\alpha + \tan^2\beta} \tag{2–84}$$

$$\tan\phi_w = \frac{-\tan\beta}{\tan\alpha} \tag{2–85}$$

$$\sin\beta_E = -\sin\alpha_T \sin\phi_w \tag{2–86}$$

For the special case when $\alpha = \pm 90°$, $\phi_w = -\beta$.

Finally, if total angle of attack α_T and windward meridian ϕ_w are known, angle of attack α, sideslip angle β, and Euler sideslip angle β_E are obtained from

$$\tan\alpha = \frac{\sin\alpha_T \cos\phi_w}{\cos\alpha_T} \tag{2–87}$$

$$\tan\beta = \frac{-\sin\alpha_T \sin\phi_w}{\cos\alpha_T} \tag{2–88}$$

$$\sin\beta_E = -\sin\alpha_T \sin\phi_w \tag{2–89}$$

For the special case when $\alpha_T = 90°$, $\beta = -\phi_w$.

Aerodynamic Angle Calculations

When all of the aerodynamic angles have been computed, Equations (2–73) through (2–75) are used to obtain the body unit vectors. The aerodynamic angles and body unit vectors are computed in the function *body_attitude* in the *derivs* module.

If body unit vectors are computed from Euler angles, as shown in Section 2.1.7, then the total angle of attack and windward meridian can be computed from their definitions given by Equations (2–79) and (2–80). Once the total angle of attack and windward meridian are known, the remaining angles can be obtained from Equations (2–87) through (2–89).

The following output variables are used for the aerodynamic and bank angles:

alpha	α	angle of attack.
alphat	α_T	total angle of attack.
bankgc	μ_{gc}	geocentric bank angle.
bankgd	μ_{gd}	geodetic bank angle.
beta	β	sideslip angle.
betae	β_E	Euler sideslip angle.
phi	ϕ_W	windward meridian angle.

2.1.10. Inertial Platform Cartesian Coordinate System

An inertial platform coordinate system can be defined at any point in a trajectory. Its origin is at the vehicle's center of mass, and at a given time its axes can be aligned with any of the other coordinate systems. By default it is aligned with the geodetic coordinate system at the beginning of the trajectory. This means that the unit vectors $\hat{x}_p$, $\hat{y}_p$, and $\hat{z}_p$ are set equal to the geodetic unit vectors $\hat{x}_{gd}$, $\hat{y}_{gd}$, and $\hat{z}_{gd}$ at the start of the trajectory. Once the system is aligned, it is fixed in inertial space until it is re-aligned by the user (Section 4.2.6).

The following output variables, computed in *inertial_platform_coords*, give the vehicle's position, inertial velocity, and inertial acceleration relative to the inertial platform coordinate system:

Position

xip	$(\vec{r} - \vec{r}_p) \cdot \hat{x}_p$	component of position relative to the inertial platform in the x–axis direction ($\vec{r}_p$ is the vector from the center of the earth to the origin of the inertial platform).
yip	$(\vec{r} - \vec{r}_p) \cdot \hat{y}_p$	component of position relative to the inertial platform in the y–axis direction.
zip	$(\vec{r} - \vec{r}_p) \cdot \hat{z}_p$	component of position relative to the inertial platform in the z–axis direction.

Velocity

xipdt	$\vec{V}_I \cdot \hat{x}_p$	component of inertial velocity vector $\vec{V}_I$ in the inertial platform x–axis direction.
yipdt	$\vec{V}_I \cdot \hat{y}_p$	component of inertial velocity vector $\vec{V}_I$ in the inertial platform y–axis direction.
zipdt	$\vec{V}_I \cdot \hat{z}_p$	component of inertial velocity vector $\vec{V}_I$ in the inertial platform z–axis direction.

Acceleration

xipdt2	$\vec{a}_I \cdot \hat{x}_p$	component of inertial acceleration vector $\vec{a}_I$ in the inertial platform x–axis direction.
yipdt2	$\vec{a}_I \cdot \hat{y}_p$	component of inertial acceleration vector $\vec{a}_I$ in the inertial platform y–axis direction.
zipdt2	$\vec{a}_I \cdot \hat{z}_p$	component of inertial acceleration vector $\vec{a}_I$ in the inertial platform z–axis direction.

2.1.11. Tangent Plane Cartesian Coordinate System

The tangent plane coordinate system is fixed and tangent to the earth's surface. It is a cartesian coordinate system that is useful for range safety calculations and for approximating a flat earth model.

The origin of the tangent plane coordinate system is given as a geodetic position, that is, a latitude δ_{tp}, longitude λ_{tp}, and altitude h_{tp} (Section 4.3.7). If values are not input for the origin location, it is assumed to be located on the earth's surface directly below the vehicle's initial position.

The x and y axes are in a local geodetic horizon plane tangent to the earth's surface and passing through the origin. The direction of the tangent plane x axis is given by an azimuth or heading angle α_{tp} measured clockwise from north. The z axis points upward away from the earth's center, and the y axis is defined to give a right–handed coordinate system.

The tangent plane unit vectors $\hat{x}_{tp}$, $\hat{y}_{tp}$, and $\hat{z}_{tp}$ are computed from

$$\begin{aligned}\hat{x}_{tp} = &-(\sin\delta_{tp}\cos\lambda_{tp}\cos\alpha_{tp} + \sin\lambda_{tp}\sin\alpha_{tp})\hat{x}_{\oplus} \\ &-(\sin\delta_{tp}\sin\lambda_{tp}\cos\alpha_{tp} - \cos\lambda_{tp}\sin\alpha_{tp})\hat{y}_{\oplus} + \cos\delta_{tp}\cos\alpha_{tp}\hat{z}_{\oplus}\end{aligned} \qquad (2\text{–}90)$$

$$\begin{aligned}\hat{y}_{tp} = &-(\sin\delta_{tp}\cos\lambda_{tp}\sin\alpha_{tp} - \sin\lambda_{tp}\cos\alpha_{tp})\hat{x}_{\oplus} \\ &-(\sin\delta_{tp}\sin\lambda_{tp}\sin\alpha_{tp} + \cos\lambda_{tp}\cos\alpha_{tp})\hat{y}_{\oplus} + \cos\delta_{tp}\sin\alpha_{tp}\hat{z}_{\oplus}\end{aligned} \qquad (2\text{–}91)$$

$$\hat{z}_{tp} = \cos\delta_{tp}\cos\lambda_{tp}\hat{x}_{\oplus} + \cos\delta_{tp}\sin\lambda_{tp}\hat{y}_{\oplus} + \sin\delta_{tp}\hat{z}_{\oplus} \qquad (2\text{–}92)$$

These equations are similar to Equations (2–19) through (2–21) for the geodetic unit vectors except they have been rotated by the tangent plane azimuth angle α_{tp}. The unit vectors are used to compute the following output variables:

xtp	$\Delta\vec{r}_{tp} \cdot \hat{x}_{tp}$	x component of the vehicle's position relative to the tangent plane coordinate system.
ytp	$\Delta\vec{r}_{tp} \cdot \hat{y}_{tp}$	y component of the vehicle's position relative to the tangent plane coordinate system.
ztp	$\Delta\vec{r}_{tp} \cdot \hat{z}_{tp}$	z component of the vehicle's position relative to the tangent plane coordinate system.
xtpdt	$\vec{V}_{\oplus} \cdot \hat{x}_{tp}$	x component of the velocity relative to the earth in the tangent plane coordinate system.
ytpdt	$\vec{V}_{\oplus} \cdot \hat{y}_{tp}$	y component of the velocity relative to the earth in the tangent plane coordinate system.
ztpdt	$\vec{V}_{\oplus} \cdot \hat{z}_{tp}$	z component of the velocity relative to the earth in the tangent plane coordinate system.

xtpdt2	$\vec{a}_{\oplus} \cdot \hat{x}_{tp}$	x component of the acceleration relative to the earth in the tangent plane coordinate system.
ytpdt2	$\vec{a}_{\oplus} \cdot \hat{y}_{tp}$	y component of the acceleration relative to the earth in the tangent plane coordinate system.
ztpdt2	$\vec{a}_{\oplus} \cdot \hat{z}_{tp}$	z component of the acceleration relative to the earth in the tangent plane coordinate system.

The tangent plane relative position vector $\Delta\vec{r}_{tp}$ is the difference between the vehicle position vector $\vec{r}$ and the position vector of the tangent plane origin $\vec{r}_{tp}$.

The tangent plane unit vectors are calculated in the *tangent_plane_unit_vectors* function and the tangent plane output variables are calculated in the *tangent_plane_coords* function.

2.1.12. Coordinate Transformations

Unit vectors are used in TAOS for coordinate transformations rather than the more common transformation matrices because they are efficient to program. The two methods are identical, but use of unit vectors makes it obvious that the operations required to convert a vector from one coordinate system to another are equivalent to a dot product.

All coordinate systems in TAOS are defined with unit vectors expressed in the ECFC coordinate system. To convert a vector from the ECFC system to another system, it is projected onto the unit vectors of that system with dot products. For example, the dot products

$$\Sigma\vec{F}_{prop_b} = \begin{bmatrix} \hat{x}_b \cdot \Sigma\vec{F}_{prop_\oplus} \\ \hat{y}_b \cdot \Sigma\vec{F}_{prop_\oplus} \\ \hat{z}_b \cdot \Sigma\vec{F}_{prop_\oplus} \end{bmatrix} = \begin{bmatrix} \hat{x}_b{}^T \\ \hat{y}_b{}^T \\ \hat{z}_b{}^T \end{bmatrix} \cdot \Sigma\vec{F}_{prop_\oplus} = [\,{}_bA_\oplus] \cdot \Sigma\vec{F}_{prop_\oplus} \quad (2\text{–}93)$$

convert the thrust vector $\Sigma\vec{F}_{prop_\oplus}$ from the ECFC system to the body–fixed system $\Sigma\vec{F}_{prop_b}$. As shown above the dot products are equivalent to a transformation matrix $[{}_bA_\oplus]$ with rows consisting of the unit vector components.

Reversing the above procedure

$$\Sigma\vec{F}_{prop_\oplus} = [\,{}_\oplus A_b] \cdot \Sigma\vec{F}_{prop_b} = [\,{}_bA_\oplus]^{-1} \cdot \Sigma\vec{F}_{prop_b} \quad (2\text{–}94)$$

$$= \begin{bmatrix} \hat{x}_b{}^T \\ \hat{y}_b{}^T \\ \hat{z}_b{}^T \end{bmatrix}^T \cdot \Sigma\vec{F}_{prop_b} \quad (2\text{–}95)$$

$$= \begin{bmatrix} \hat{x}_b & \hat{y}_b & \hat{z}_b \end{bmatrix} \cdot \Sigma\vec{F}_{prop_b} \quad (2\text{–}96)$$

converts from the body–fixed coordinate system to the ECFC system. The reverse procedure is simplified because the transformation matrix is orthogonal so its inverse is its transpose. In this case the columns of the transformation matrix contain the components of the unit vectors.

Transformations to and from other coordinate systems in TAOS are handled in a similar manner.

2.2. Equations of Motion

The equations of motion are derived from Newton's second law applied to a point mass

$$\vec{a}_I = \Sigma\vec{F} \,/\, m \qquad (2\text{–}97)$$

This equation defines acceleration in an inertial coordinate system. It is convenient for trajectory analysis to define the acceleration in terms of an earth–fixed coordinate system, that is, $\vec{a}_\oplus$. This system rotates relative to the inertial system at the earth's spin rate ${}_I\vec{\omega}_\oplus$.

The acceleration in inertial coordinates is equivalent to taking the second derivative of the vehicle's position vector, that is,

$$\vec{a}_I = \frac{{}^I d^2\,\vec{r}}{dt^2} = \frac{{}^I d}{dt}\left(\frac{{}^I d\,\vec{r}}{dt}\right) \qquad (2\text{–}98)$$

where ${}^I d/dt$ means the time derivative with respect to inertial coordinates. If the inner derivative is differentiated relative to the rotating earth–fixed coordinate system, it results in

$$\vec{a}_I = \frac{{}^I d}{dt}\left[\frac{{}^\oplus d\,\vec{r}}{dt} + ({}_I\vec{\omega}_\oplus \times \vec{r})\right] \qquad (2\text{–}99)$$

where ${}^\oplus d/dt$ means the time derivative with respect to earth–fixed coordinates. Then, if the outer derivative is differentiated relative to the same earth–fixed coordinates, it gives

$$\vec{a}_I = \frac{{}^\oplus d}{dt}\left[\frac{{}^\oplus d\,\vec{r}}{dt} + {}_I\vec{\omega}_\oplus \times \vec{r}\right] + {}_I\vec{\omega}_\oplus \times \left[\frac{{}^\oplus d\,\vec{r}}{dt} + {}_I\vec{\omega}_\oplus \times \vec{r}\right] \qquad (2\text{–}100)$$

$$\vec{a}_I = \left[\frac{{}^\oplus d^2\,\vec{r}}{dt^2} + \frac{d\,{}_I\vec{\omega}_\oplus}{dt} \times \vec{r} + {}_I\vec{\omega}_\oplus \times \frac{{}^\oplus d\,\vec{r}}{dt}\right] \qquad (2\text{–}101)$$
$$+ {}_I\vec{\omega}_\oplus \times \frac{{}^\oplus d\,\vec{r}}{dt} + {}_I\vec{\omega}_\oplus \times ({}_I\vec{\omega}_\oplus \times \vec{r})$$

The second term in Equation (2–101) containing the derivative of the earth's spin rate is zero because the earth's spin rate ${}_I\vec{\omega}_\oplus$ is constant. Furthermore, the notation can be simplified with the substitutions

$$\vec{V}_\oplus = \frac{{}^\oplus d\,\vec{r}}{dt} \quad \text{and} \quad \vec{a}_\oplus = \frac{{}^\oplus d^2\,\vec{r}}{dt^2} \qquad (2\text{–}102)$$

This gives

$$\vec{a}_I = \left[\vec{a}_\oplus + {}_I\vec{\omega}_\oplus \times \vec{V}_\oplus\right] + \left[{}_I\vec{\omega}_\oplus \times \vec{V}_\oplus + {}_I\vec{\omega}_\oplus \times ({}_I\vec{\omega}_\oplus \times \vec{r})\right] \quad (2\text{–}103)$$

The terms ${}_I\vec{\omega}_\oplus \times \vec{V}_\oplus$ can be combined resulting in

$$\vec{a}_I = \vec{a}_\oplus + 2 \cdot ({}_I\vec{\omega}_\oplus \times \vec{V}_\oplus) + {}_I\vec{\omega}_\oplus \times ({}_I\vec{\omega}_\oplus \times \vec{r}) \quad (2\text{–}104)$$

Finally, combining Equation (2–97) with (2–104) gives an expression for acceleration in an earth–fixed coordinate system, that is,

$$\vec{a}_\oplus = \frac{\Sigma\vec{F}}{m} - 2 \cdot ({}_I\vec{\omega}_\oplus \times \vec{V}_\oplus) - {}_I\vec{\omega}_\oplus \times ({}_I\vec{\omega}_\oplus \times \vec{r}) \quad (2\text{–}105)$$

This is a second–order differential equation of motion, which can be written as the following set of first–order differential equations:

$$\begin{aligned} \dot{\vec{r}}_\oplus &= \vec{V}_\oplus \\ \dot{\vec{V}}_\oplus &= \frac{\Sigma\vec{F}}{m} - 2 \cdot ({}_I\vec{\omega}_\oplus \times \vec{V}_\oplus) - {}_I\vec{\omega}_\oplus \times ({}_I\vec{\omega}_\oplus \times \vec{r}) \end{aligned} \quad (2\text{–}106)$$

If an initial position $\vec{r}$ and velocity with respect to the earth $\vec{V}_\oplus$ are known at an initial time t_0, and if the vehicle's mass m and forces acting on it $\Sigma\vec{F}$ are known, these equations can be integrated with respect to time resulting in a trajectory. The functions used to integrate the equations of motion are in the *derivs* module.

2.2.1. Numerical Integration

For the purpose of numerical integration, the equations of motion are written in terms of a state vector defined as

$$\vec{S}(t) = \begin{bmatrix} \vec{r} \\ \vec{V}_\oplus \end{bmatrix} = \begin{bmatrix} \vec{r} \cdot \hat{x}_\oplus \\ \vec{r} \cdot \hat{y}_\oplus \\ \vec{r} \cdot \hat{z}_\oplus \\ \vec{V} \cdot \hat{x}_\oplus \\ \vec{V} \cdot \hat{y}_\oplus \\ \vec{V} \cdot \hat{z}_\oplus \end{bmatrix} = \begin{bmatrix} x_\oplus \\ y_\oplus \\ z_\oplus \\ \dot{x}_\oplus \\ \dot{y}_\oplus \\ \dot{z}_\oplus \end{bmatrix} \tag{2–107}$$

The state vector is a function of time and has a corresponding derivative vector. Given a time and state, the derivatives are

$$\dot{\vec{S}}(t, \vec{S}) = \begin{bmatrix} \dot{\vec{r}} \\ \dot{\vec{V}}_\oplus \end{bmatrix} = \begin{bmatrix} \vec{V}_\oplus \\ \vec{a}_\oplus \end{bmatrix} = \begin{bmatrix} \vec{V}_\oplus \cdot \hat{x}_\oplus \\ \vec{V}_\oplus \cdot \hat{y}_\oplus \\ \vec{V}_\oplus \cdot \hat{z}_\oplus \\ \vec{a}_\oplus \cdot \hat{x}_\oplus \\ \vec{a}_\oplus \cdot \hat{y}_\oplus \\ \vec{a}_\oplus \cdot \hat{z}_\oplus \end{bmatrix} = \begin{bmatrix} \dot{x}_\oplus \\ \dot{y}_\oplus \\ \dot{z}_\oplus \\ \ddot{x}_\oplus \\ \ddot{y}_\oplus \\ \ddot{z}_\oplus \end{bmatrix} \tag{2–108}$$

where

$$\ddot{x}_\oplus = \frac{\Sigma \vec{F} \cdot \hat{x}_\oplus}{m} + \omega_\oplus \cdot (2\, \dot{y}_\oplus + \omega_\oplus x_\oplus) \tag{2–109}$$

$$\ddot{y}_\oplus = \frac{\Sigma \vec{F} \cdot \hat{y}_\oplus}{m} + \omega_\oplus \cdot (-2\, \dot{x}_\oplus + \omega_\oplus y_\oplus) \tag{2–110}$$

$$\ddot{z}_\oplus = \frac{\Sigma \vec{F} \cdot \hat{z}_\oplus}{m} \tag{2–111}$$

and where

$$\omega_\oplus = \| {}_I\vec{\omega}_\oplus \| \tag{2–112}$$

These state and derivative vectors are augmented with equations for the mass m, the path length r, the ground range r_S, and any user–defined integral variables (Section 4.3.1).

Vehicle Mass

The initial mass is given as part of the initial conditions (Section 4.3.5), and the mass rate is given as a function of time and the vehicle's state (Section 4.2.9). The following output variables are associated with the vehicle's mass:

fuel	$\Delta m \cdot g$	fuel used (in English units $g = 32.174\ lb_m / slug$).
mass	m	vehicle mass.
mdt	$\dot{m}$	mass rate.
wt	$m \cdot g$	vehicle weight.

Path Length

The path length r represents the distance the vehicle travels through the air. It is defined as the time integral of the velocity magnitude $\| \vec{V}_{\oplus} \|$. Path length is often used to represent the length of a launch rail or sled track. The following output variables are associated with the path length:

plength	r	trajectory path length.
plmark	r	path length from a designated point in the trajectory.
plseg	r	segment path length.

Ground Range

The ground range r_S is the length of the projection of the flight path onto the earth's surface. Its derivative is the ground speed $\| \vec{V}_S \|$, the velocity of the point on the earth's surface directly below the vehicle. An expression for the ground speed is given in Equation (2–152) in Section 2.2.4.

Runge–Kutta Integration

The complete state and derivative vectors, calculated in the *compute_derivs* function, are given by

$$\vec{S}(t) = \begin{bmatrix} x_{\oplus} \\ y_{\oplus} \\ z_{\oplus} \\ \dot{x}_{\oplus} \\ \dot{y}_{\oplus} \\ \dot{z}_{\oplus} \\ m \\ r \\ r_S \end{bmatrix} \qquad \dot{\vec{S}}(t, \vec{S}) = \begin{bmatrix} \dot{x}_{\oplus} \\ \dot{y}_{\oplus} \\ \dot{z}_{\oplus} \\ \ddot{x}_{\oplus} \\ \ddot{y}_{\oplus} \\ \ddot{z}_{\oplus} \\ \dot{m} \\ \| \vec{V}_{\oplus} \| \\ \| \vec{V}_S \| \end{bmatrix} \tag{2–113}$$

These equations are numerically integrated with respect to time with a fixed step size 4th–order Runge–Kutta method. A fixed step size method is used for simplicity so numerical accuracy is controlled with the input step size Δt.

$$\vec{S}(t+\Delta t) = \vec{S}(t) + \frac{1}{6}(\dot{\vec{S}} + 2\dot{\vec{S}}' + 2\dot{\vec{S}}'' + \dot{\vec{S}}''')\,\Delta t \qquad (2\text{–}114)$$

where

$$\dot{\vec{S}} = \dot{\vec{S}}(t, \vec{S}) \qquad (2\text{–}115)$$

$$\dot{\vec{S}}' = \dot{\vec{S}}(t + 0.5\,\Delta t,\ \vec{S} + 0.5\,\Delta t \cdot \dot{\vec{S}}(t)) \qquad (2\text{–}116)$$

$$\dot{\vec{S}}'' = \dot{\vec{S}}(t + 0.5\,\Delta t,\ \vec{S} + 0.5\,\Delta t \cdot \dot{\vec{S}}'(t)) \qquad (2\text{–}117)$$

$$\dot{\vec{S}}''' = \dot{\vec{S}}(t + \Delta t,\ \vec{S} + \Delta t \cdot \dot{\vec{S}}'')) \qquad (2\text{–}118)$$

Each integration step requires four derivative evaluations and is computed in the function *integration_step*.

At each time step, TAOS computes a number of output variables from the state and derivative vectors including longitude and latitude rates and flight path angle rates. Equations for these values are derived in the following sections.

2.2.2. Rail Launches and Sled Tests

For rail–launched rockets or for sled tracks, the accelerations must be modified to restrict the motion to the direction of the rails. The launch rail or sled track orientation is given in the guidance rules as a body attitude. The rail or sled is assumed to be aligned with the body's $\hat{x}_b$ axis, so the vehicle can only move in this direction.

The accelerations required to keep the vehicle moving along the rail are given by

$$\begin{aligned} \ddot{x}_{\oplus} &= \| \vec{a}_{rail} \| \, \hat{x}_b \cdot \hat{x}_{\oplus} \\ \ddot{y}_{\oplus} &= \| \vec{a}_{rail} \| \, \hat{x}_b \cdot \hat{y}_{\oplus} \\ \ddot{z}_{\oplus} &= \| \vec{a}_{rail} \| \, \hat{x}_b \cdot \hat{z}_{\oplus} \end{aligned} \qquad (2\text{–}119)$$

where the magnitude of the acceleration along the rail is

$$\| \vec{a}_{rail} \| = \vec{a}_{\oplus} \cdot \hat{x}_b - \| \vec{a}_f \| \qquad (2\text{–}120)$$

The loss of acceleration from friction with the rail $\| \vec{a}_f \|$ is computed from the static or kinetic coefficient of friction μ_f and the total force normal to the rail $\| \Sigma \vec{F}_n \|$, that is,

$$\| \vec{a}_f \| = \frac{\mu_f \cdot \| \Sigma \vec{F}_n \|}{m} = \mu_f \cdot \| \vec{a}_n \| \qquad (2\text{–}121)$$

where

$$\| \vec{a}_n \| = \sqrt{(\vec{a}_{\oplus} \cdot \hat{y}_b)^2 + (\vec{a}_{\oplus} \cdot \hat{z}_b)^2} \qquad (2\text{–}122)$$

In other words, the acceleration normal to the rail is the vector difference between the total acceleration and the acceleration along the rail. The acceleration loss from friction is the product of the normal acceleration and the coefficient of friction μ_f. If the velocity is near zero (less than 0.001 ft/sec), the static coefficient of friction is used in Equation (2–121); otherwise, the kinetic coefficient of friction is used (Section 4.2.10).

When a rocket is mounted on a launch rail, the rail mechanism prevents it from moving backwards; it can only move forward in the positive $\hat{x}_b$ direction. Therefore, TAOS limits $\| \vec{a}_{rail} \| > 0$ for rail launches to keep the vehicle motionless before motor ignition.

For sled tests this limitation does not apply because the sled can both accelerate and decelerate. Sled tracks are normally oriented parallel to the earth's surface so the initial acceleration along the tracks before motor ignition is zero and the sled remains motionless.

2.2.3. Longitude, Latitude, and Altitude Rates

TAOS computes the longitude, latitude, and altitude rates as the following output variables in the function *compute_state_rates*:

longdt	$\dot{\lambda}$	longitude rate.
latgcdt	$\dot{\delta}_{gc}$	geocentric latitude rate.
latgddt	$\dot{\delta}_{gd}$	geodetic latitude rate.
altdt	$\dot{h}$	altitude rate or rate of climb.

Longitude Rate

Longitude λ is defined as

$$\sin\lambda = \frac{\vec{r} \cdot \hat{y}_{\oplus}}{\sqrt{(\vec{r} \cdot \hat{x}_{\oplus})^2 + (\vec{r} \cdot \hat{y}_{\oplus})^2}} \qquad (2\text{–}123)$$

$$\cos\lambda = \frac{\vec{r} \cdot \hat{x}_{\oplus}}{\sqrt{(\vec{r} \cdot \hat{x}_{\oplus})^2 + (\vec{r} \cdot \hat{y}_{\oplus})^2}} \qquad (2\text{–}124)$$

so

$$\lambda = \tan^{-1}\left[\frac{\vec{r} \cdot \hat{y}_{\oplus}}{\vec{r} \cdot \hat{x}_{\oplus}}\right] \qquad (2\text{–}125)$$

which is the same for both geocentric and geodetic coordinate systems. Longitude rate $\dot{\lambda}$ can be obtained by differentiating this equation as follows:

$$\dot{\lambda} = \frac{(\vec{V}_{\oplus} \cdot \hat{y}_{\oplus})(\vec{r} \cdot \hat{x}_{\oplus}) - (\vec{V}_{\oplus} \cdot \hat{x}_{\oplus})(\vec{r} \cdot \hat{y}_{\oplus})}{(\vec{r} \cdot \hat{x}_{\oplus})^2 + (\vec{r} \cdot \hat{y}_{\oplus})^2} \qquad (2\text{–}126)$$

$$\dot{\lambda} = \frac{\vec{V}_{\oplus} \cdot (-\sin\lambda\ \hat{x}_{\oplus} + \cos\lambda\ \hat{y}_{\oplus})}{\sqrt{(\vec{r} \cdot \hat{x}_{\oplus})^2 + (\vec{r} \cdot \hat{y}_{\oplus})^2}} \qquad (2\text{–}127)$$

The denominator is the projection of the position vector onto the equatorial plane. If it becomes zero, the longitude rate is undefined and TAOS sets it to zero.

Geocentric Latitude Rate

From Figure 2–2, the geocentric latitude rate $\dot{\delta}_{gc}$ is defined as

$$\dot{\delta}_{gc} = \frac{\vec{V}_{\oplus} \cdot \hat{x}_{gc}}{\|\vec{r}\|} \tag{2-128}$$

Geodetic Latitude Rate

The geodetic latitude rate $\dot{\delta}_{gd}$ cannot be determined as easily. A derivation of geodetic latitude rate begins by multiplying Equation (2–33) by the quantity $(1-e^2)\tan\delta_{gd}$ and subtracting Equation (2–32) to get

$$\tilde{x}(1-e^2)\tan\delta_{gd} - \vec{r}\cdot\hat{z}_{\oplus} = -he^2\sin\delta_{gd} \tag{2-129}$$

This equation can be differentiated with respect to time to get

$$\begin{aligned}\dot{\tilde{x}}(1-e^2)\tan\delta_{gd} + \dot{\delta}_{gd}\,\tilde{x}\,(1-e^2)\sec^2\delta_{gd} - \vec{V}_{\oplus}\cdot\hat{z}_{gd} \\ = -\dot{h}e^2\sin\delta_{gd} - \dot{\delta}_{gd}\,he^2\cos\delta_{gd}\end{aligned} \tag{2-130}$$

which can be solved for $\dot{\delta}_{gd}$ resulting in

$$\dot{\delta}_{gd} = \frac{\left\{\dot{z}\cos\delta_{gd} - \left[\dot{\tilde{x}}(1-e^2) + \dot{h}e^2\cos\delta_{gd}\right]\sin\delta_{gd}\right\}\cos\delta_{gd}}{\tilde{x}(1-e^2) + he^2\cos^3\delta_{gd}} \tag{2-131}$$

where

$$\tilde{x} = (N+h)\cos\delta_{gd} \tag{2-132}$$

$$\dot{\tilde{x}} = \vec{V}_{\oplus}\cdot(\cos\lambda\,\hat{x}_{\oplus} + \sin\lambda\,\hat{y}_{\oplus}) \tag{2-133}$$

$$\dot{z} = \vec{V}_{\oplus}\cdot\hat{z}_{\oplus} \tag{2-134}$$

$$\dot{h} = -\vec{V}_{\oplus}\cdot\hat{z}_{gd} \tag{2-135}$$

Equation (2–133) comes from differentiating Equations (2–31) and (2–32). Substituting Equations (2–132) through (2–135) into (2–131) and performing some algebra simplifies this expression to

$$\dot{\delta}_{gd} = \frac{\vec{V}_{\oplus}\cdot\hat{x}_{gd}}{N\cdot\left(\dfrac{1-e^2}{1-e^2\sin^2\delta_{gd}}\right)+h} \tag{2-136}$$

Finally, substituting Equation (2–28) for N gives

$$\dot{\delta}_{gd} = \frac{\vec{V}_{\oplus}\cdot\hat{x}_{gd}}{h + \dfrac{R_{\oplus}(1-e^2)}{\left(1-e^2\sin^2\delta_{gd}\right)^{1.5}}} \tag{2-137}$$

The geodetic latitude rate is singular when the denominator in Equation (2–137) is zero. When this condition occurs, the geodetic latitude rate is set equal to the geocentric value.

Altitude Rate and Acceleration

A result of the derivation for geodetic latitude rate is the altitude rate $\dot{h}$ given in Equation (2–133). The vertical acceleration $\ddot{h}$ can be obtained by differentiating this equation yielding

$$\ddot{h} = -\vec{a}_{\oplus} \cdot \hat{z}_{gd} - \vec{V}_{\oplus} \cdot \dot{\hat{z}}_{gd} \qquad (2\text{–}138)$$

where $\dot{\hat{z}}_{gd}$ is given by differentiating Equation (2–19)

$$\dot{\hat{z}}_{gd} = \frac{d}{dt}\left[-\cos\delta_{gd}\,(\cos\lambda\,\hat{x}_{\oplus} + \sin\lambda\,\hat{y}_{\oplus}) - \sin\delta_{gd}\,\hat{z}_{\oplus}\right] \qquad (2\text{–}139)$$

$$= \left[\sin\delta_{gd}\,(\cos\lambda\,\hat{x}_{\oplus} + \sin\lambda\,\hat{y}_{\oplus}) - \cos\delta_{gd}\,\hat{z}_{\oplus}\right]\dot{\delta}_{gd} + \cos\delta_{gd}(\sin\lambda\,\hat{x}_{\oplus} - \cos\lambda\,\hat{y}_{\oplus})\,\dot{\lambda} \qquad (2\text{–}140)$$

The vertical acceleration is required for one of the guidance rules (Section 2.5.1).

Dynamic Pressure and Mach Number Rates

Two other rates, the second derivative of dynamic pressure and the first derivative of Mach number, are required by some of the guidance rules. One definition of dynamic pressure is

$$q = 0.5\,\varrho\,\|\vec{V}_{w_{\oplus}}\|^2 \qquad (2\text{–}141)$$

Differentiating this twice gives

$$\ddot{q} = 0.5\,\ddot{\varrho}\,\|\vec{V}_{w_{\oplus}}\|^2 + 2\,\dot{\varrho}\,\|\vec{V}_{w_{\oplus}}\| \cdot \|\dot{\vec{V}}_{w_{\oplus}}\| + \varrho\,\|\dot{\vec{V}}_{w_{\oplus}}\|^2 \qquad (2\text{–}142)$$

where

$$\dot{\varrho} = \frac{\partial h}{\partial t} \cdot \frac{\partial \varrho}{\partial h} = \dot{h} \cdot \frac{\partial \varrho}{\partial h} \qquad (2\text{–}143)$$

and

$$\ddot{\varrho} = \frac{\partial^2 h}{\partial t^2} \cdot \frac{\partial \varrho}{\partial h} = \ddot{h} \cdot \frac{\partial \varrho}{\partial h} \qquad (2\text{–}144)$$

The terms with $\|\ddot{\vec{V}}_{w_{\oplus}}\|$ and $\partial^2\varrho/\partial h^2$ are assumed to be small, $\|\dot{\vec{V}}_{w_{\oplus}}\|$ is assumed to be equal to $\|\dot{\vec{V}}_{\oplus}\|$, and the partial derivative of atmospheric density with respect to altitude is obtained numerically from the model atmosphere.

Similarly, the Mach number, which is given by

$$M = \| \vec{V}_{w_\oplus} \| / \, c \tag{2-145}$$

can be differentiated resulting in

$$\dot{M} = \frac{\| \dot{\vec{V}}_{w_\oplus} \| \, c - \| \vec{V}_{w_\oplus} \| \, \dot{c}}{c^2} \tag{2-146}$$

where

$$\dot{c} = \frac{\partial h}{\partial t} \cdot \frac{\partial c}{\partial h} = \dot{h} \cdot \frac{\partial c}{\partial h} \tag{2-147}$$

Again $\| \dot{\vec{V}}_{w_\oplus} \|$ is assumed to be equal to $\| \dot{\vec{V}}_\oplus \|$, and the partial derivative of the speed of sound with respect to altitude is computed numerically from the model atmosphere.

2.2.4. Ground Speed and Range

As mentioned previously, the ground range r_S is the length of the projection of the flight path onto the earth's surface. Its derivative is the ground speed $\| \vec{V}_S \|$, the velocity of the point on the earth's surface nearest the vehicle. This point is where an extension of the geodetic $\hat{z}_{gd}$ axis intersects the earth's surface, that is, the point on the earth's surface directly below the vehicle.

The following output variables are computed in the function *compute_state_rates* for ground range and speed:

range	r_S	ground range for the entire trajectory.
grseg	r_S	ground range for a trajectory segment.
grmark	r_S	ground range from a designated point in the trajectory.
vgr	$\| \vec{V}_S \|$	ground speed.

The ground speed vector is defined as

$$\vec{V}_S = \vec{V}_\oplus + \Delta \vec{V}_S \qquad (2\text{–}148)$$

where $\Delta \vec{V}_S$ is the velocity of a point on the earth's surface below the vehicle relative to the vehicle. Another way of saying this is that $\Delta \vec{V}_S$ is the rate of change of the altitude vector, that is,

$$\Delta \vec{V}_S = \frac{d}{dt}\left(h\ \hat{z}_{gd}\right) \qquad (2\text{–}149)$$

$$= \dot{h}\ \hat{z}_{gd} + h\ \dot{\hat{z}}_{gd} \qquad (2\text{–}150)$$

The altitude rate $\dot{h}$ is defined in Equation (2–133), and the rate of change of $\hat{z}_{gd}$ is given in Equation (2–140). If Equation (2–150) is substituted into Equation (2–148) and if $\vec{V}_\oplus$ is expanded into its geodetic components, it becomes

$$\vec{V}_S = (\vec{V}_\oplus \cdot \hat{x}_{gd})\ \hat{x}_{gd} + (\vec{V}_\oplus \cdot \hat{y}_{gd})\ \hat{y}_{gd} + h\ \dot{\hat{z}}_{gd} \qquad (2\text{–}151)$$

Now the geodetic velocity components can be replaced with the definitions in Equation (2–49) resulting in

$$\vec{V}_S = \| \vec{V}_\oplus \| \cdot \cos\gamma_{gd}\ (\cos\psi_{gd}\ \hat{x}_{gd} + \sin\psi_{gd}\ \hat{y}_{gd}) + h\ \dot{\hat{z}}_{gd} \qquad (2\text{–}152)$$

Ground speed is the magitude of this vector, and ground range is the integral of ground speed over time.

2.2.5. Flight Path Angle Rates

The following flight path angle rates are computed as output variables in the function *compute_rates*:

psigcdt	$\dot{\psi}_{gc}$	geocentric heading rate.
psigddt	$\dot{\psi}_{gd}$	geodetic heading rate.
gamgcdt	$\dot{\gamma}_{gc}$	geocentric vertical flight path angle rate.
gamgddt	$\dot{\gamma}_{gd}$	geodetic vertical flight path angle rate.

Equations for the flight path angle rates are derived from the angular rate between the velocity coordinate system defined in Equations (2–68) through (2–70) and the geocentric or geodetic coordinate system. The derivation shown here is for the geodetic flight path angle rates; the same method can be used for the geocentric flight path angle rates.

The angular velocity of the velocity coordinate system can be expressed as

$$ {}_{\oplus}\vec{\omega}_V = {}_{\oplus}\vec{\omega}_{gd} + {}_{gd}\vec{\omega}_V \qquad (2\text{–}153) $$

which is the sum of the angular velocity between the ECFC and geodetic coordinate systems and the angular velocity between the geodetic and velocity coordinate systems.

From the coordinate system definitions, the angular velocity between the ECFC coordinate system and the geodetic coordinate system is given by

$$ {}_{\oplus}\vec{\omega}_{gd} = \dot{\lambda}\,\hat{z}_{\oplus} - \dot{\delta}_{gd}\,\hat{y}_{gd} \qquad (2\text{–}154) $$

Similarly, the angular velocity between the geodetic coordinate system and the velocity coordinate system is given by

$$ {}_{gd}\vec{\omega}_V = \dot{\psi}_{gd}\,\hat{z}_{gd} + \dot{\gamma}_{gd}\,\hat{y}_V \qquad (2\text{–}155) $$

Equations (2–153) through (2–155) can be combined to get

$$ {}_{\oplus}\vec{\omega}_V = \dot{\lambda}\,\hat{z}_{\oplus} - \dot{\delta}_{gd}\,\hat{y}_{gd} + \dot{\psi}_{gd}\,\hat{z}_{gd} + \dot{\gamma}_{gd}\,\hat{y}_V \qquad (2\text{–}156) $$

If this equation is broken into its components in the velocity coordinate system, the component in the $\hat{y}_V$ direction is

$$ {}_{\oplus}\vec{\omega}_V \cdot \hat{y}_V = (\dot{\lambda}\,\hat{z}_{\oplus} - \dot{\delta}_{gd}\,\hat{y}_{gd}) \cdot \hat{y}_V + \dot{\gamma}_{gd} \qquad (2\text{–}157) $$

because

$$ \dot{\psi}_{gd}\,\hat{z}_{gd} \cdot \hat{y}_V = 0 \qquad (2\text{–}158) $$

The definition of the heading angle ψ_{gd} given in Equation (2–69) can be substituted into Equation (2–157) yielding

$$ {}_{\oplus}\vec{\omega}_V \cdot \hat{y}_V = (\dot{\lambda}\, \hat{z}_{\oplus} \cdot \hat{y}_V) - \dot{\delta}_{gd}\cos\psi_{gd} + \dot{\gamma}_{gd} \qquad (2\text{–}159) $$

The same procedure can be used to get the component of Equation (2–156) in the $\hat{z}_V$ direction resulting in

$$ {}_{\oplus}\vec{\omega}_V \cdot \hat{z}_V = (\dot{\lambda}\, \hat{z}_{\oplus} - \dot{\delta}_{gd}\, \hat{y}_{gd} + \dot{\psi}_{gd}\, \hat{z}_{gd}) \cdot \hat{z}_V \qquad (2\text{–}160) $$

Equation (2–70) can be used to simplify this to

$$ {}_{\oplus}\vec{\omega}_V \cdot \hat{z}_V = \dot{\lambda}\, \hat{z}_{\oplus} \cdot \hat{z}_V - \dot{\delta}_{gd}\sin\gamma_{gd}\sin\psi_{gd} + \dot{\psi}_{gd}\cos\gamma_{gd} \qquad (2\text{–}161) $$

An expression for the angular velocity between the ECFC and velocity system can be obtained by writing the earth–fixed acceleration as the velocity differentiated with respect to the velocity coordinate system, that is,

$$ \vec{a}_{\oplus} = \frac{d\,\|\vec{V}_{\oplus}\|}{dt}\,\hat{x}_V + {}_{\oplus}\vec{\omega}_V \times \vec{V}_{\oplus} \qquad (2\text{–}162) $$

where

$$ \frac{d\,\|\vec{V}_{\oplus}\|}{dt} = \frac{\vec{a}_{\oplus} \cdot \vec{V}_{\oplus}}{\|\vec{V}_{\oplus}\|} = \vec{a}_{\oplus} \cdot \hat{x}_V \qquad (2\text{–}163) $$

The cross product in Equation (2–162) simplifies because the velocity vector is aligned with the $\hat{x}_V$ axis in the velocity coordinate system giving

$$ {}_{\oplus}\vec{\omega}_V \times \vec{V}_{\oplus} = \begin{vmatrix} \hat{x}_v & \hat{y}_v & \hat{z}_v \\ \omega_{V_x} & \omega_{V_y} & \omega_{V_z} \\ \|\vec{V}_{\oplus}\| & 0 & 0 \end{vmatrix} \qquad (2\text{–}164) $$

$$ = \|\vec{V}_{\oplus}\| \cdot \left[({}_{\oplus}\vec{\omega}_V \cdot \hat{z}_V)\, \hat{y}_V - ({}_{\oplus}\vec{\omega}_V \cdot \hat{y}_V)\, \hat{z}_V \right] $$

where ω_{V_x},ω_{V_y} and ω_{V_z} are the components of ${}_{\oplus}\vec{\omega}_V$ in the velocity coordinate system.

Equations (2–162) through (2–164) can be combined resulting in

$$ \vec{a}_{\oplus} = (\vec{a}_{\oplus} \cdot \hat{x}_V)\, \hat{x}_V + \|\vec{V}_{\oplus}\| \cdot \left[({}_{\oplus}\vec{\omega}_V \cdot \hat{z}_V)\, \hat{y}_V - ({}_{\oplus}\vec{\omega}_V \cdot \hat{y}_V)\, \hat{z}_V \right] \qquad (2\text{–}165) $$

If this equation is broken into its $\hat{y}_V$ and $\hat{z}_V$ components, it gives

$$ \vec{a}_{\oplus} \cdot \hat{y}_V = \|\vec{V}_{\oplus}\| \cdot ({}_{\oplus}\vec{\omega}_V \cdot \hat{z}_V) \qquad (2\text{–}166) $$

$$ {}_{\oplus}\vec{\omega}_V \cdot \hat{z}_V = \frac{\vec{a}_{\oplus} \cdot \hat{y}_V}{\|\vec{V}_{\oplus}\|} \qquad (2\text{–}167) $$

and

$$\vec{a}_{\oplus} \cdot \hat{z}_V = -\| \vec{V}_{\oplus} \| \cdot ({}_{\oplus}\vec{\omega}_V \cdot \hat{y}_V) \tag{2–168}$$

$${}_{\oplus}\vec{\omega}_V \cdot \hat{y}_V = \frac{-\vec{a}_{\oplus} \cdot \hat{z}_V}{\| \vec{V}_{\oplus} \|} \tag{2–169}$$

Equations (2–159) and (2–169) can be equated and solved for $\dot{\gamma}_{gd}$ giving

$$\dot{\gamma}_{gd} = -\frac{\vec{a}_{\oplus} \cdot \hat{z}_V}{\| \vec{V}_{\oplus} \|} - \dot{\lambda}\, \hat{z}_{\oplus} \cdot \hat{y}_V + \dot{\delta}_{gd} \cos\psi_{gd} \tag{2–170}$$

Similarly, Equations (2–161) and (2–167) can be equated and solved for $\dot{\psi}_{gd}$ giving

$$\dot{\psi}_{gd} = \frac{\frac{\vec{a}_{\oplus} \cdot \hat{y}_V}{\| \vec{V}_{\oplus} \|} - \dot{\lambda}\, \hat{z}_{\oplus} \cdot \hat{z}_V + \dot{\delta}_{gd} \sin\gamma_{gd} \sin\psi_{gd}}{\cos\gamma_{gd}} \tag{2–171}$$

The unit vectors $\hat{y}_V$ and $\hat{z}_V$ are obtained from Equations (2–68) through (2–70).

2.2.6. Specific Load Factors

One definition for the specific load vector is the inertial acceleration of the vehicle minus the acceleration from gravity, that is,

$$\vec{a}_{sp} = \vec{a}_I - \vec{a}_{grav} \qquad (2\text{–}172)$$

Another equivalent definition for specific load vector is the sum of the aerodynamic and propulsive forces acting on the vehicle divided by the vehicle mass, that is,

$$\vec{a}_{sp} = \frac{\Sigma\vec{F}_{aero} + \Sigma\vec{F}_{prop}}{m} \qquad (2\text{–}173)$$

Specific load factors, the components of the specific load vector, are useful for determining structural requirements and limitations, and they are often referred to as the g loading. TAOS computes the following specific load factor output variables in the function *compute_derivs*:

nx	$\vec{a}_{sp} \cdot \hat{x}_b$	specific load factor component along the body x–axis often called the axial g's.
ny	$\vec{a}_{sp} \cdot \hat{y}_b$	specific load factor component along the body y–axis.
nz	$\vec{a}_{sp} \cdot \hat{z}_b$	specific load factor component along the body z–axis.
ntotal	$\| \vec{a}_{sp} \|$	magnitude of the specific load vector often referred to as the total g's acting on the vehicle.

Another related quantity commonly used is called the normal g's or lateral g's. It can be computed from the specific load factors with

$$a_{sp_n} = \sqrt{(\vec{a}_{sp} \cdot \hat{y}_b)^2 + (\vec{a}_{sp} \cdot \hat{z}_b)^2} \qquad (2\text{–}174)$$

2.3. Contributions to Acceleration

The vehicle acceleration, given by the equations of motion, is

$$\vec{a}_{\oplus} = \frac{\Sigma\vec{F}}{m} - 2 \cdot ({}_I\vec{\omega}_{\oplus} \times \vec{V}_{\oplus}) - {}_I\vec{\omega}_{\oplus} \times ({}_I\vec{\omega}_{\oplus} \times \vec{r}) \qquad (2\text{–}175)$$

The term $\Sigma\vec{F}/m$ represents the contributions to acceleration from all of the external forces acting on the vehicle. This includes aerodynamic forces $\Sigma\vec{F}_{aero}$, propulsive forces $\Sigma\vec{F}_{prop}$, and gravitational forces $\Sigma\vec{F}_{grav}$. These forces and their corresponding accelerations are shown in Figure 2–16.

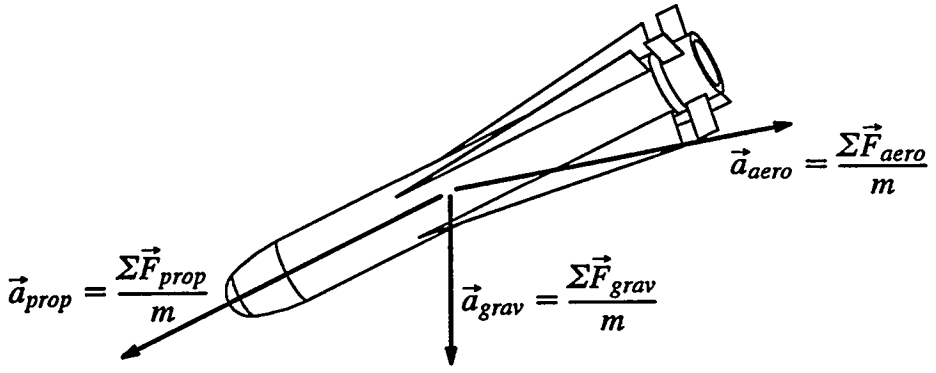

Figure 2–16. Accelerations from Forces Acting on the Vehicle.

Aerodynamic forces $\Sigma\vec{F}_{aero}$ are functions of the vehicle's shape, its state, and the earth's atmospheric properties, such as temperature, pressure, and density. TAOS contains several atmosphere models, given in Section 2.3.1, that compute these properties. Aerodynamic force calculations from input data are given in Section 2.3.2.

Propulsive forces $\Sigma\vec{F}_{prop}$, from rocket or jet engines, are defined with input data and discussed in Section 2.3.3.

TAOS assumes trajectories are near to the earth so that gravitational forces from the earth are predominant. Gravitational forces from the moon, sun, and planets are neglected. An earth gravity model is presented in Section 2.3.4 that computes acceleration from gravity $\vec{a}_{grav}$.

2.3.1. Atmosphere Model

The atmosphere models available in TAOS were originally developed by Acurex Corporation under contract to Sandia and subsequently corrected by Landrum in 1987.[12] The models include the 1976 U.S. standard atmosphere[13], 1966 supplemental atmospheres[14], Tonopah range data, and a mean annual Kwajalein atmosphere.[15]

The 1976 U.S. standard atmosphere is an idealized, mid–latitude representation of the atmosphere under year–round moderate solar activity. Other atmosphere models are similar in that they represent average conditions for a given location and season.

On any given day the actual atmospheric properties may significantly vary (as much as 50 percent for density) from the atmosphere model depending on the specific location and date. However, it is impossible to predict actual atmospheric conditions for a given location and date, so model atmospheres are used as a best approximation. The 1976 U.S. standard atmosphere is widely used for consistency when comparing trajectories.

All of the atmosphere models rely on the same basic assumptions to simplify analysis. From these assumptions, equations can be developed relating altitude, temperature, pressure, density, speed of sound, and viscosity. These equations are used to compute the following output variables in TAOS:

temp	T	temperature.
pres	p	pressure.
rho	ϱ	density.
sndspd	c	speed of sound.
nu	ν	kinematic viscosity.

Assumptions

The gas composition of the atmosphere is assumed to satisfy the aerostatic equation

$$dp = -\varrho \cdot g \cdot dh \qquad (2\text{–}176)$$

and the perfect gas law

$$p = \varrho \cdot \frac{R^*}{M_w} \cdot T \qquad (2\text{–}177)$$

where g is the acceleration due to gravity, h is the altitude above sea–level, R^* is the universal gas constant (8314.32 N·m/(kmol·K), and M_w is the molecular weight.

The atmosphere is assumed to be divided into concentric layers surrounding the earth. Within each layer the gradient of the molecular temperature τ with respect to the geopotential altitude H is assumed to be constant, that is,

$$\frac{d\tau}{dH} = \text{constant} \qquad (2\text{–}178)$$

$$\text{where} \quad \tau = \frac{M_{w_0}}{M_w} \cdot T \tag{2–179}$$

$$\text{and} \quad H = \frac{1}{G}\int_0^h g \cdot dh \tag{2–180}$$

The zero subscript indicates sea–level conditions. For example, M_{w_0} is the molecular weight at sea–level which is 28.9644 kg/kmol for the 1976 U.S. standard atmosphere model and 28.96 kg/kmol for all other atmosphere models. The reference geopotential G is 9.80665 m/sec^2. This corresponds to the sea–level acceleration due to gravity at a latitude of 45.5425° from Equation (2–183).

For atmospheric calculations the acceleration of gravity is assumed to vary with altitude according to the inverse square law

$$g = g_0 \cdot \left(\frac{R_\oplus{}'}{R_\oplus{}' + h}\right)^2 \tag{2–181}$$

where $R_\oplus{}'$ is the effective earth radius for geopotential calculations. When this is substituted into Equation (2–180) and integrated, the geopotential altitude becomes

$$H = \frac{g_0}{G} \cdot \left(\frac{R_\oplus{}' h}{R_\oplus{}' + h}\right) \tag{2–182}$$

The sea–level acceleration due to gravity in m/sec^2 is assumed to be a function of latitude and given by Lambert's equation

$$g_0 = 9.806160 \cdot (1 - 0.0026373\cos 2\delta + 0.0000059\cos^2 2\delta) \tag{2–183}$$

The latitude used in this equation is the latitude used to describe the atmosphere model. For example, a 60°N atmosphere uses a value of 60° for the latitude. This is inconsistent with the gravity model used for trajectory calculations, but it is consistent with the 1976 U.S. standard[13] and 1966 supplemental atmosphere models.[14]

The effective radius of the earth, used in Equation (2–182) for calculating the geopotential altitude, is different from the actual radius of the earth. It is obtained from

$$R_\oplus{}' = \frac{2 \cdot g_0}{3.085462 \times 10^{-6} + 2.27 \times 10^{-9}\cos 2\delta - 2.0 \times 10^{-12}\cos 4\delta} \tag{2–184}$$

which gives an earth radius of 6356.766 km at a latitude of 45.5425°.

The properties for each atmosphere layer are given in tables that are a function of altitude h or geopotential altitude H. Molecular temperature τ and molecular temperature gradient $d\tau/dH$ are given as a function of geopotential altitude H. Molecular weight M_w is given as a function of altitude h. These table values represent the properties at the base of each atmosphere layer.

Temperature

The geopotential altitude, molecular temperature, and gradient are obtained from the tables for the base of the i^{th} layer. The molecular temperature and weight at a specific geopotential altitude within this layer are

$$\tau = \tau_i + \frac{d\tau}{dH} \cdot (H - H_i) \qquad (2\text{–}185)$$

$$M_w = M_{w_i} + \frac{dM_w}{dH} \cdot (H - H_i) \qquad (2\text{–}186)$$

and the atmospheric temperature is

$$T = \frac{M_w}{M_{w_0}} \cdot \tau \qquad (2\text{–}187)$$

Pressure

The atmospheric pressure is obtained by writing Equations (2–176) and (2–177) as a function of geopotential altitude using Equations (2–179) and (2–180) resulting in

$$dp = -\varrho g_0 dH \qquad (2\text{–}188)$$

and

$$p = \frac{\varrho R^* \tau}{M_{w_0}} \qquad (2\text{–}189)$$

Dividing Equation (2–188) by Equation (2–189) gives the differential relation

$$\frac{dp}{p} = -\frac{g_0 M_{w_0}}{R^* \tau} dH \qquad (2\text{–}190)$$

The molecular temperature τ is a linear function of the geopotential altitude H as shown in Equation (2–185). Substituting this equation into Equation (2–190) and integrating gives the following two expressions for pressure depending on the value of the molecular temperature gradient:

$$p = p_i \, e^{-\left(\frac{g_0 M_{w_0}(H - H_i)}{R^* \tau_i}\right)} \quad \text{for} \quad \frac{d\tau}{dH} = 0 \qquad (2\text{–}191)$$

$$p = p_i \left(\frac{\tau_i}{\tau}\right)^{\frac{g_0 M_{w_0}}{R^* \cdot \frac{d\tau}{dH}}} \quad \text{for} \quad \frac{d\tau}{dH} \neq 0 \qquad (2\text{–}192)$$

These equations for atmospheric pressure are first used to compute pressure at the base altitudes of the atmospheric layers and then later to compute pressure at any geopotential altitude.

Density, Speed–of–Sound, and Viscosity

The equation for atmospheric density

$$\varrho = \frac{p \cdot M_{w_0}}{R^* \cdot \tau} \qquad (2\text{–}193)$$

is obtained from combining Equations (2–177) and (2–179).

The speed of sound is given by

$$c = \sqrt{\frac{k\,R^*\,T}{M_w}} = \sqrt{\frac{k\,R^*\,\tau}{M_{w_0}}} \qquad (2\text{–}194)$$

where k is the ratio of specific heats, which is equal to 1.4 for air.

And finally, kinematic viscosity ν is computed from Sutherland's formula

$$\nu = \frac{1}{\varrho} \cdot \frac{1.458 \times 10^{-6}\,\tau^{1.5}}{\tau + 110.4} \qquad (2\text{–}195)$$

Atmosphere Calculations

The 1976 U.S. standard atmosphere table is defined up to an altitude of 146 km. The other atmosphere models are defined up to altitudes ranging between 30 km and 117 km, and then they are extrapolated to 146 km such that at 146 km they are the same as the 1976 standard atmosphere. The function *atmos_setup* is executed prior to any atmosphere calculations to select the tables for the desired atmosphere model and load them into arrays for later use.

Atmosphere data for altitudes below 146 km is computed as shown above in function *atmos_properties*. Above 146 km a separate table containing temperature, pressure and density as a function of altitude for the 1976 U.S. standard atmosphere is used for all atmosphere models. The function *atmos_hi_alt* linearly interpolates temperature and exponentially interpolates pressure and density from this table for altitudes up to 1000 km.

All of the model atmosphere functions are in the *atmos* module. This module is separate so it can be easily used with other software.

2.3.2. Aerodynamic Forces

Aerodynamic force vectors are calculated from aerodynamic coefficients given in a table or problem file (Sections 3.1 and 4.2.1). The coefficients are converted to forces by multiplying them by the dynamic pressure q and the reference area S_{ref}.

Three different sets of aerodynamic coefficients can be input: the axial and normal force coefficients (C_A and C_N), the lift, drag, and side force coefficients (C_L, C_D, and C_S), and the body x, y, and z force coefficients (C_X, C_Y, and C_Z). Each set of coefficients defines an aerodynamic force vector. If more than one set of coefficients is given, they are summed to obtain $\Sigma\vec{F}_{aero}$.

The aerodynamic coefficients are often functions of the Mach number and Reynold's number. Both of these quantities are computed as output variables along with the dynamic pressure, that is,

dynprs	$q = 0.7pM^2$	dynamic pressure.
mach	$M = \|\vec{V}_{w_\oplus}\| / c$	Mach number.
reypft	$R_N = \|\vec{V}_{w_\oplus}\| / \nu$	Reynold's number per unit length.

These variables are functions of the atmospheric pressure p,the speed of sound c, and the kinematic viscosity ν (Section 2.3.1).

Axial and Normal Force Coefficients

The axial and normal force coefficients are often used for axisymmetric vehicles such as reentry bodies and missiles. The axial force acts in the opposite direction of the body x axis, and the normal force acts in the opposite direction of the windward meridian ϕ_w.

A unit vector representing the windward meridian $\hat{\phi}_b$ is obtained by projecting the x axis of the wind coordinate system onto the plane formed by the body $\hat{y}_b$ and $\hat{z}_b$ axes as shown in Figure 2–12 in Section 2.1.9, that is,

$$\hat{\phi}_b = \frac{(\hat{x}_w \cdot \hat{y}_b)\,\hat{y}_b + (\hat{x}_w \cdot \hat{z}_b)\,\hat{z}_b}{\sqrt{(\hat{x}_w \cdot \hat{y}_b)^2 + (\hat{x}_w \cdot \hat{z}_b)^2}} \qquad (2\text{–}196)$$

Now the axial and normal coefficients can be converted to an aerodynamic force with

$$\vec{F}_{aero} = -(C_A\,\hat{x}_b + C_N\,\hat{\phi}_b) \cdot q\,S_{ref} \qquad (2\text{–}197)$$

The first term represents the axial force and the second term represents the normal force. Equation (2–196) is singular when the wind x axis is aligned with the body x axis, in other words, when the total angle of attack is zero, so TAOS assumes the normal force is zero when this occurs.

Lift, Drag, and Side–Force Coefficients

Lift, drag, and side–force coefficients are often used for aircraft applications. The lift force is in the opposite direction to the wind z axis, the drag force is in the opposite

direction to the wind x axis, and the side force is in the same direction as the wind y axis. Thus, the aerodynamic force vector is given by

$$\vec{F}_{aero} = -(C_D\,\hat{x}_w - C_S\,\hat{y}_w + C_L\,\hat{z}_w)\cdot q\,S_{ref} \qquad (2\text{–}198)$$

Body–Axes Force Coefficients

The force coefficients C_X, C_Y, and C_Z, sometimes obtained from wind tunnel tests, represent forces aligned with the body–fixed coordinate system, so the aerodynamic force vector is

$$\vec{F}_{aero} = (C_X\,\hat{x}_b + C_Y\,\hat{y}_b + C_Z\,\hat{z}_b)\cdot q\,S_{ref} \qquad (2\text{–}199)$$

Aerodynamic Forces

Aerodynamic forces are calculated in the function *force_aero*. TAOS first converts all types of aerodynamic coefficients to lift, drag, and side forces using the definitions given above and coordinate transformations. If more than one set of coefficients is defined (with multiple **aero* data blocks as shown in Section 4.2.1), then the forces from each set of coefficients is added together giving the total lift, drag, and side force. Finally, the total aerodynamic force is transformed from the wind coordinate system to the ECFC coordinate system so it can be used in the equations of motion.

The following output variables are used in TAOS for the aerodynamic coefficients:

ca	C_A	axial aerodynamic force coefficient.
cn	C_N	normal aerodynamic force coefficient.
cl	C_L	total aerodynamic lift coefficient.
cd	C_D	total aerodynamic drag coefficient.
cs	C_S	total aerodynamic side–force coefficient.
cx	C_X	body x–axis aerodynamic force coefficient.
cy	C_Y	body y–axis aerodynamic force coefficient.
cz	C_Z	body z–axis aerodynamic force coefficient.

The output variables for the lift, drag, and side–force coefficients are always calculated. The remaining output variables are only defined when they correspond to an input set of coefficients. For example, if tables are input defining C_A and C_N, then the corresponding output variables are defined, but the output variables for C_X, C_Y, and C_Z are left undefined.

When multiple sets of aerodynamic coefficients are input, only the lift, drag, and side–force coefficients are guaranteed to represent the total forces acting on the vehicle. If all of the input aerodynamic coefficients are of the same type, then the corre-

sponding output variables are totalled correctly. However, if the input coefficients are not the same type, then the remaining coefficients are either undefined or represent partial totals.

2.3.3. Propulsive Forces

The magnitude and direction of propulsive force vectors are given in the table file as shown in Section 3.1 or in the problem file as shown in Section 4.2.9. If more than one force vector is given, they are summed to form $\Sigma\vec{F}_{prop}$. The equations of motion are for a point mass, so thrust vectors act through the vehicle's center of mass, and their direction is determined from two thrust vector angles, ε_1 and ε_2, defined in Figure 2–17.

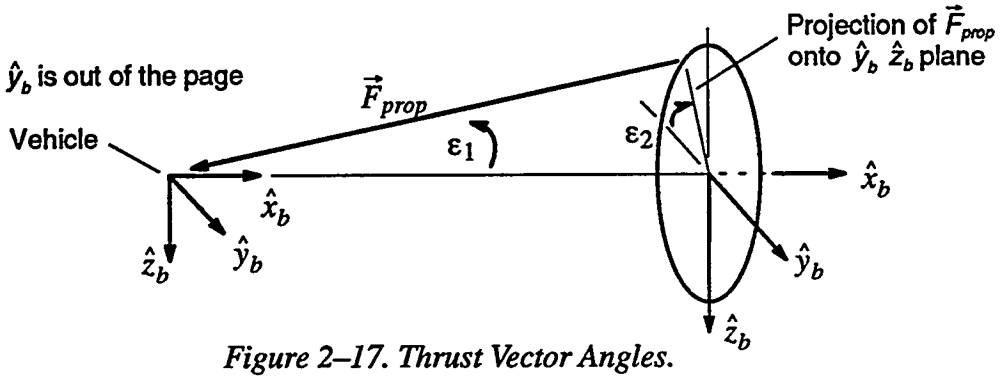

Figure 2–17. Thrust Vector Angles.

The thrust vector angles are analogous to the total angle of attack and windward meridian. The thrust vector angle ε_1 is the angle between the body x axis and the thrust vector and has values ranging from 0° to 180°. The thrust vector angle ε_2 is the angle between the projection of the thrust vector onto the plane formed by $\hat{y}_b$ and $\hat{z}_b$ and the negative $\hat{y}_b$ axis.

The thrust vector is defined as

$$\vec{F}_{prop} = \|\vec{F}_{prop}\| \cdot \cos\varepsilon_1 \, \hat{x}_b - \|\vec{F}_{prop}\| \cdot \sin\varepsilon_1 \cos\varepsilon_2 \, \hat{y}_b - \|\vec{F}_{prop}\| \cdot \sin\varepsilon_1 \sin\varepsilon_2 \, \hat{z}_b \qquad (2\text{–}200)$$

in function *force_prop*. It is converted from the body–fixed coordinate system to the ECFC coordinate system with a coordinate transform as shown in Section 2.1.12.

The following output variables are associated with the propulsive force vector:

thrust	$\|\vec{F}_{prop}\|$	total thrust vector magnitude.
ep1	ε_1	thrust vector angle 1.
ep2	ε_2	thrust vector angle 2.

If more than one propulsive force vector is given (more than one **prop* data block as shown in Section 4.2.9), then the *thrust* output variable is the sum of the thrust magnitudes. This only has meaning if all of the thrust vectors are aligned. The thrust vector angles, *ep1* and *ep2*, have values corresponding to the last thrust vector evaluated (the **prop* data blocks are evaluated in the same order as input).

2.3.4. Gravity Model

The gravitational field surrounding a planet is represented with a gravity potential function U. The gravitational acceleration vector, required for trajectory analysis, is the gradient of the potential function, that is,

$$\nabla U = \vec{a}_{grav} \tag{2–201}$$

When the gravity potential function is applied to the earth, it is called the geopotential. The geopotential is a function of an object's position relative to the center of the earth so it is written as a function of the geocentric coordinates $\|\vec{r}\|$, δ_{gc}, and λ. The geopotential is commonly expressed in terms of spherical harmonics as follows:[6]

$$U = \frac{GM}{\|\vec{r}\|}\left\{1 + \sum_{n=2}^{\infty}\sum_{m=0}^{n}\left[\frac{R_\oplus}{\|\vec{r}\|}\right]^n P_{n,m}(\sin\delta_{gc})\left[C_{n,m}\cos m\lambda + S_{n,m}\sin m\lambda\right]\right\} \tag{2–202}$$

One reason spherical harmonics are used for the geopotential is that the constants $C_{n,m}$ and $S_{n,m}$ can be related physically to regions on the earth's surface. Earth gravitational models, created from satellite tracking measurements, consist of values for the constants GM, $R_\oplus$, $C_{n,m}$, and $S_{n,m}$.

Since the center of the geocentric coordinate system coincides with the earth's center of mass, the first terms of the expansion, $C_{1,0}$ and $C_{1,1}$, are zero so they have been eliminated by summing from $n = 2$ rather than $n = 1$.

Geopotential

The geopotential function can be divided into three parts. The first part

$$\frac{GM}{\|\vec{r}\|} \tag{2–203}$$

corresponds to the potential derived from treating the earth as a point mass. When the spherical earth model is selected in TAOS, it consists of just this one term. All other terms are neglected.

The second part of the geopotential consists of the terms which are only functions of latitude, that is, the terms with $m = 0$. These terms are called the zonal harmonics. The second–degree zonal harmonic models the oblateness of the earth shape, and it is the largest of the spherical harmonic terms.

A common simplification of the geopotential is to include only the zonal harmonic terms, resulting in

$$U = \frac{GM}{\|\vec{r}\|}\left\{1 + \sum_{n=2}^{\infty}\left[\frac{R_\oplus}{\|\vec{r}\|}\right]^n P_{n,0}(\sin\delta_{gc})\ C_{n,0}\right\} \tag{2–204}$$

This equation is often written in terms of J_n, where $J_n = -C_{n,0}$ giving

$$U = \frac{GM}{\|\vec{r}\|}\left\{1 - \sum_{n=2}^{\infty}\left[\frac{R_\oplus}{\|\vec{r}\|}\right]^n P_{n,0}(\sin\delta_{gc})\ J_n\right\} \qquad (2\text{–}205)$$

The third part of the geopotential consists of the remaining terms that depend on longitude. The largest of these terms is usually the degree 2 and order 2 term which models the asymmetry of the earth about the equator. This term is significantly smaller than the second degree zonal harmonic term that models the oblateness of the earth.[10]

Legendre Functions

The $P_{n,m}(\sin\delta_{gc})$ terms in the geopotential function are Legendre functions of $\sin\delta_{gc}$ defined as[6]

$$P_{n,m}(\sin\delta_{gc}) = \cos^m\delta_{gc} \cdot \frac{d^m}{d(\sin\delta_{gc})^m}\left(P_{n,0}(\sin\delta_{gc})\right) \qquad (2\text{–}206)$$

where

$$P_{n,0}(\sin\delta_{gc}) = \frac{1}{(2^n)n!} \cdot \frac{d^n}{d(\sin\delta_{gc})^n}\left(\sin^2\delta_{gc} - 1\right)^n \qquad (2\text{–}207)$$

The notation

$$\frac{d^n}{d(\sin\delta_{gc})^n}\left(P(\sin\delta_{gc})\right) \qquad (2\text{–}208)$$

means the n^{th} derivative of P with respect to $\sin\delta_{gc}$. For example, the first four Legendre functions of order zero are

$$\begin{aligned} P_{1,0}(\sin\delta_{gc}) &= \sin\delta_{gc} \\ P_{2,0}(\sin\delta_{gc}) &= \tfrac{1}{2} \cdot (3\sin^2\delta_{gc} - 1) \\ P_{3,0}(\sin\delta_{gc}) &= \tfrac{1}{2} \cdot (5\sin^3\delta_{gc} - 3\sin\delta_{gc}) \\ P_{4,0}(\sin\delta_{gc}) &= \tfrac{1}{8} \cdot (35\sin^4\delta_{gc} - 30\sin^2\delta_{gc} + 3) \end{aligned} \qquad (2\text{–}209)$$

Normalization

The spherical harmonic coefficients in the geopotential functions given in Equations (2–202) and (2–204) are unnormalized. These coefficients become very small as the degree increases partly because of the nature of the gravity field, but mostly because the associated Legendre functions become large. Thus, it is common to normalize the coefficients and Legendre functions by a scale factor that is a function of the degree and order given by[10]

$$\overline{P}_{n,m} = \left[(2 - \delta_m)(2n + 1)\frac{(n-m)!}{(n+m)!}\right]^{\frac{1}{2}} \cdot P_{n,m} \qquad (2\text{–}210)$$

and to normalize the coefficients $C_{n,m}$ and $S_{n,m}$ by the inverse of this, for example,

$$\overline{C}_{n,m} = \left[\frac{1}{(2-\delta_m)(2n+1)}\frac{(n+m)!}{(n-m)!}\right]^{\frac{1}{2}} \cdot C_{n,m} \qquad (2\text{–}211)$$

where $\delta_m = 1$ if $m = 0$, and $\delta_m = 0$ if $m > 0$. Normalized Legendre functions and coefficients are denoted with a bar.

TAOS uses the unnormalized form of the Legendre functions and coefficients because it neglects all terms higher than 4th degree. However, published coefficients for the WGS–84[6] and GEM–T1[11] gravity models are normalized and must be unnormalized before they can be used in TAOS.

Gravity Acceleration Vector

As stated previously, the gravity acceleration vector is the gradient of the geopotential function so the components of the gravity acceleration vector in geocentric coordinates are

$$\vec{a}_{grav} \cdot \hat{x}_{gc} = \frac{1}{\|\vec{r}\|} \cdot \frac{\partial U}{\partial \delta_{gc}} \qquad (2\text{–}212)$$

$$\vec{a}_{grav} \cdot \hat{y}_{gc} = \frac{1}{\|\vec{r}\| \cos\delta_{gc}} \cdot \frac{\partial U}{\partial \lambda} \qquad (2\text{–}213)$$

$$\vec{a}_{grav} \cdot \hat{z}_{gc} = -\frac{\partial U}{\partial \|\vec{r}\|} \qquad (2\text{–}214)$$

The $\vec{a}_{grav} \cdot \hat{z}_{gc}$ term is negated because $\vec{r}$ is in the opposite direction of $\hat{z}_{gc}$. Taking the partial derivatives of Equation (2–202) and substituting them into Equations (2–212) through (2–214) results in

$$\vec{a}_{grav} \cdot \hat{x}_{gc} = \frac{GM}{\|\vec{r}\|^2} \sum_{n=2}^{\infty} \left[\frac{R_\oplus}{\|\vec{r}\|}\right]^n \sum_{m=0}^{n} \frac{\partial P_{n,m}(\sin\delta_{gc})}{\partial \delta_{gc}} \cdot \left[C_{n,m}\cos m\lambda + S_{n,m}\sin m\lambda\right] \qquad (2\text{–}215)$$

$$\vec{a}_{grav} \cdot \hat{y}_{gc} = \frac{GM}{\|\vec{r}\|^2} \sum_{n=2}^{\infty} \left[\frac{R_\oplus}{\|\vec{r}\|}\right]^n \sum_{m=0}^{n} m \frac{P_{n,m}(\sin\delta_{gc})}{\cos\delta_{gc}} \cdot \left[-C_{n,m}\sin m\lambda + S_{n,m}\cos m\lambda\right] \qquad (2\text{–}216)$$

$$\vec{a}_{grav} \cdot \hat{z}_{gc} = \frac{GM}{\|\vec{r}\|^2} \left\{1 + \sum_{n=2}^{\infty}(n+1)\left[\frac{R_\oplus}{\|\vec{r}\|}\right]^n \sum_{m=0}^{n} P_{n,m}(\sin\delta_{gc}) \cdot \left[C_{n,m}\cos m\lambda + S_{n,m}\sin m\lambda\right]\right\} \qquad (2\text{–}217)$$

These equations are used with $n = 4$ in function *accel_grav* to compute the gravity acceleration vector for the *wgs–84–full* and *gem–t1–full* earth models (Section 4.4.3). The gravitational effects of the sun and moon are neglected.

They are simplified to only include zonal harmonic terms through $n = 4$ for the *tsap–72* and *tsap–84* earth models. These earth models correspond to those used in TSAP[2]; however, they are somewhat inconsistent because the J_3 and J_4 terms are

roughly the same order of magnitude as some of the nonzonal harmonic terms that have been neglected.

The standard *wgs–72* and *wgs–84* earth models in TAOS include only the second degree zonal harmonic J_2 so the equations for the gravity acceleration vector simplify to

$$\vec{a}_{grav} \cdot \hat{x}_{gc} = -\frac{GM}{\|\vec{r}\|^2} \cdot \left[\frac{R_\oplus}{\|\vec{r}\|}\right]^2 \cdot (3\ J_2 \sin\delta_{gc}\ \cos\delta_{gc}) \qquad (2\text{–}218)$$

$$\vec{a}_{grav} \cdot \hat{y}_{gc} = 0 \qquad (2\text{–}219)$$

$$\vec{a}_{grav} \cdot \hat{z}_{gc} = \frac{GM}{\|\vec{r}\|^2} \cdot \left\{1 - \left[\frac{R_\oplus}{\|\vec{r}\|}\right]^2 \cdot \frac{3}{2}\ J_2\ (3\sin^2\delta_{gc} - 1)\right\} \qquad (2\text{–}220)$$

The earth models that include only the J_2 term provide adequate accuracy for a point–mass trajectory code like TAOS. The effects of higher–order terms on most trajectories is small and well within the accuracy of other features of the code such as the atmosphere models.

2.4. Output Variables

At each integration step TAOS computes the following output variables from the vehicle's state:

	Range and Distance	
crsrng	Δr_S	distance perpendicular to a reference azimuth.
dwnrng	Δr_S	distance along a reference azimuth.
east	Δr_S	distance along constant latitude.
north	Δr_S	distance along constant longitude.
	Radar Observations	
radrng	$\|\Delta\vec{r}_r\|$	radar range.
radrngdt	$\|\Delta\dot{\vec{r}}_r\|$	radar range rate.
radrngdt2	$\|\Delta\ddot{\vec{r}}_r\|$	radar range acceleration.
radaz	α_r	radar azimuth angle.
radazdt	$\dot{\alpha}_r$	radar azimuth rate.
radazdt2	$\ddot{\alpha}_r$	radar azimuth acceleration.
radelv	ε_r	radar elevation angle.
radelvdt	$\dot{\varepsilon}_r$	radar elevation rate.
radelvdt2	$\ddot{\varepsilon}_r$	radar elevation acceleration.
radasp	η_r	vehicle aspect angle from radar.
radmer	ϕ_r	vehicle meridional angle from radar.
	Relative Vehicle Variables	
relrng	$\|\Delta\vec{r}_i\|$	relative range of i^{th} vehicle.
relaz	α_i	relative azimuth angle of i^{th} vehicle.
relelv	ε_i	relative elevation angle of i^{th} vehicle.
relvel	$\|\Delta\dot{\vec{r}}_i\|$	closure velocity of i^{th} vehicle.
relxb	$\Delta\vec{r}_i \cdot \hat{x}_b$	x–position of i^{th} vehicle in body axis system.
relyb	$\Delta\vec{r}_i \cdot \hat{y}_b$	y–position of i^{th} vehicle in body axis system.
relzb	$\Delta\vec{r}_i \cdot \hat{z}_b$	z–position of i^{th} vehicle in body axis system.
	Initial Impact Point Variables	
iip_latgd	δ_{iip}	geodetic latitude of initial impact point.

iip_long	λ_{iip}	longitude of initial impact point.
iip_time	t_{iip}	time at initial impact point.
iip_rng	r_{iip}	range to initial impact point.
iip_azm	α_{iip}	azimuth to initial impact point.

Aerodynamic Variables

alpha_l/d	$\alpha_{L/D}$	angle of attack at maximum lift–to–drag ratio.
ballistic	β_c	ballistic coefficient.
l/d	L/D	lift–to–drag ratio.

These variables are computed only if they are used or referenced in the input problem file. For example, if one of these variables is listed as a printout variable or as a segment final condition, then it is computed.

2.4.1. Ranges and Distances

If the flight path is projected onto the earth's surface, the resulting curve gives the ground range as shown in Section 2.2.4. This is the distance traveled by the vehicle along the earth's surface, but it is not referenced to anything. The ranges and distances in this section use two–dimensional coordinate systems defined on the earth's surface to act as references for the trajectory ground track.

East and North Distances

An east/north coordinate system is defined on the earth's surface such that the east axis is aligned with lines of constant latitude and the north axis is aligned with lines of constant longitude as shown in Figure 2–18. The origin of the coordinate system is located at a known reference point, often the initial position of a vehicle. This two–dimensional coordinate system is useful for short–range trajectories where the earth's curvature is not a factor.

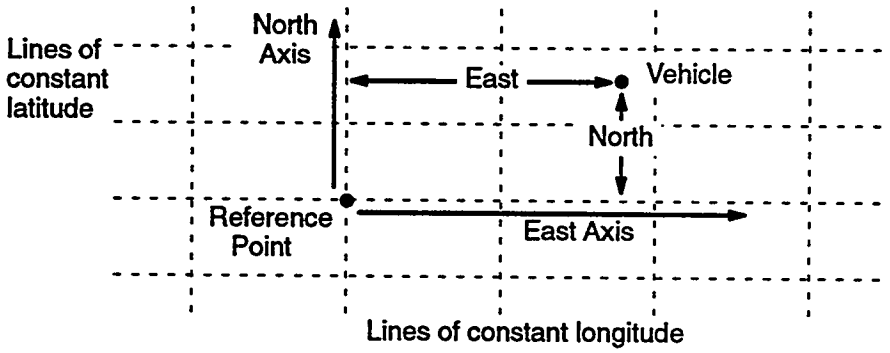

Figure 2–18. East and North Distances.

The east distance is the distance between the reference longitude and the vehicle's longitude along the reference latitude. Similarly, the north distance is the distance between the reference latitude and the vehicle's latitude along the reference longitude.

These calculations depend on methods that determine the distance between two known longitude/latitude points on the earth's surface. This is a well–known geodesy problem and many methods are available.

Sodano's Inverse Method

TAOS uses Sodano's inverse method because it is accurate, it is noniterative, and it works over both short and long distances.[16] The method is coded in function *sodano_inverse* in the *derivs* module. It requires the longitude and geodetic latitude of two points on the earth's surface, that is, $(\lambda_1, \delta_{gd_1})$ and $(\lambda_2, \delta_{gd_2})$. Then, the distance on the earth's surface Δr_S between these two points is given by

$$\frac{\Delta r_S}{R_p} = \left[1 + f + f^2\right]\phi + \sin\beta_1 \sin\beta_2 \left[(f + f^2)\sin\phi - \tfrac{1}{2} f^2 \phi^2 \csc\phi\right] \qquad (2\text{–}221)$$
$$- \tfrac{1}{2} m\left[(f + f^2)(\phi + \sin\phi\cos\phi) - f^2\phi^2\cot\phi\right]$$
$$- \tfrac{1}{2}\sin^2\beta_1 \sin^2\beta_2\, f^2 \sin\phi\cos\phi$$
$$+ \tfrac{1}{2}\sin\beta_1 \sin\beta_2\, mf^2\left[\sin\phi\cos^2\phi + \phi^2\csc\phi\right]$$
$$+ \tfrac{1}{16} f^2 m^2\left[\phi + \sin\phi\cos\phi - 2\sin\phi\cos^3\phi - 8\phi^2\cot\phi\right]$$

where

$$\cos\phi = \sin\beta_1 \sin\beta_2 + \cos\beta_1 \cos\beta_2 \cdot (\lambda_2 - \lambda_1) \qquad (2\text{–}222)$$

$$m = 1 - \cos^2\beta_1 \cos^2\beta_2 \sin^2(\lambda_2 - \lambda_1)\csc^2\phi \qquad (2\text{–}223)$$

and

$$\tan\beta_1 = (1 - f)\tan\delta_{gd_1} = \frac{R_p \sin\delta_{gd_1}}{R_\oplus \cos\delta_{gd_1}} \qquad (2\text{–}224)$$

$$\tan\beta_2 = (1 - f)\tan\delta_{gd_1} = \frac{R_p \sin\delta_{gd_2}}{R_\oplus \cos\delta_{gd_2}} \qquad (2\text{–}225)$$

The earth's equatorial radius $R_\oplus$, the earth's polar radius R_p, and the flatness parameter f are defined in Section 2.1.5.

Sodano's inverse method also gives azimuth angles between the two points. An azimuth angle is similar to a heading angle. It is measured clockwise from north so that a 0° azimuth angle points north and a 90° azimuth angle points east. The earth's surface is modeled as an ellipsoid so a constant azimuth angle defines a curve. A curve connecting two points on the earth's surface has an azimuth angle at each point called a forward azimuth ψ_{fwd} and a backward azimuth ψ_{back}.

Given two points, the forward azimuth corresponds to the azimuth of the connecting curve at the first point, and the backward azimuth corresponds to the azimuth at the second point. In many cases the difference between the angles is approximately 180°. From Sodano's inverse method, they are given by

$$\tan\psi_{fwd} = \frac{\cos\beta_2 \sin\Lambda}{\sin\Delta\beta + 2\cos\beta_2 \sin\beta_1 \sin^2\frac{\Lambda}{2}} \qquad (2\text{–}226)$$

$$\tan\psi_{back} = \frac{\cos\beta_1 \sin\Lambda}{\sin\Delta\beta - 2\cos\beta_1 \sin\beta_2 \sin^2\frac{\Lambda}{2}} \qquad (2\text{–}227)$$

where

$$\Lambda = \lambda_2 - \lambda_1 + \left(\frac{\cos\beta_1 \cos\beta_2 \sin(\lambda_2 - \lambda_1)}{\sin\phi}\right)\left[(f + f^2)\phi - \tfrac{1}{2}\sin\beta_1 \sin\beta_2 f^2(\sin\phi + 2\phi^2 \csc\phi) + \tfrac{1}{4}mf^2(\sin\phi\cos\phi - 5\phi + 4\phi^2\cot\phi)\right] \quad (2\text{–}228)$$

$$\Delta\beta = \delta_{gd_2} - \delta_{gd_1} + n(\sin 2\delta_{gd_1} - \sin 2\delta_{gd_2}) - \tfrac{1}{2}n^2(\sin 4\delta_{gd_1} - \sin 4\delta_{gd_2}) + \tfrac{1}{3}n^3(\sin 6\delta_{gd_1} - \sin 6\delta_{gd_2}) \quad (2\text{–}229)$$

$$= \delta_{gd_2} - \delta_{gd_1} + 2\sin(\delta_{gd_2} - \delta_{gd_1})[\sin\beta_1 \sin\beta_2(n + n^2 + n^3) - \cos\beta_1 \cos\beta_2(n - n^2 + n^3)] \quad (2\text{–}230)$$

and

$$n = \frac{R_\oplus - R_p}{R_\oplus + R_p} \quad (2\text{–}231)$$

Downrange and Crossrange

Another coordinate system can be defined on the earth's surface for downrange and crossrange calculations as shown in Figure 2–19. In this system downrange is measured from a reference point along a reference azimuth. Crossrange is measured perpendicular to the reference azimuth.

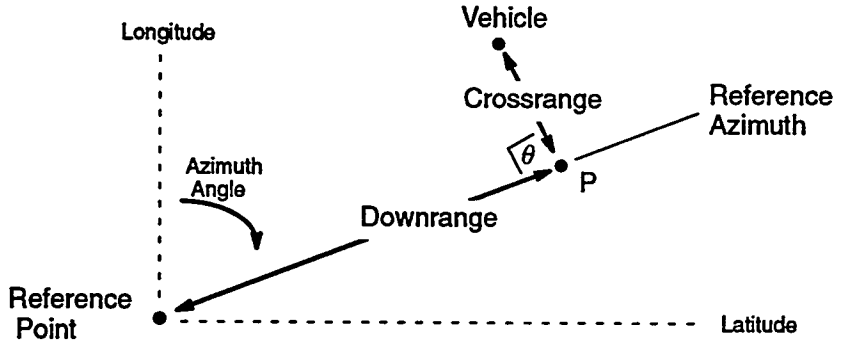

Figure 2–19. Downrange and Crossrange.

A Newton–Raphson iterative search is used to vary the downrange until the difference between the backward azimuth ψ_{back} from the point P to the reference point and the forward azimuth ψ_{fwd} from point P to the vehicle is 90°, that is,

$$\theta = \psi_{back} - \psi_{fwd} = 90 \quad (2\text{–}232)$$

In other words, the point P slides along the reference azimuth line until the downrange and crossrange lines are perpendicular.

The Newton–Raphson search method solves for a value x such that an error function $f(x) = 0$ (Section 2.6.2).[8] From an initial estimate of x_n, a new value of x_{n+1} is estimated from

$$x_{n+1} = x_n - f(x_n) \cdot \frac{\partial x}{\partial f(x_n)} \tag{2–233}$$

where n is the iteration counter. The partial derivative of $f(x_n)$ with respect to x is estimated numerically using a forward difference. In this application, x is the downrange and the error function $f(x)$ is the difference between the angle θ and 90°, that is,

$$f(x) = \frac{\pi}{2} - |\psi_{back} - \psi_{fwd}| \tag{2–234}$$

Sodano's Direct Method

The error function in the Newton–Raphson solution requires Sodano's direct method to calculate the backward azimuth ψ_{back}, the longitude λ_2, and latitude δ_{gd_2} of point P given a reference longitude λ_1, a reference latitude δ_{gd_1}, a reference azimuth ψ, and a distance Δr_S.[16]

Sodano's direct method begins by computing the angle ϕ from

$$\begin{aligned}\phi = \zeta &- ne'^2 \sin\zeta + \tfrac{1}{2}me'^2[\sin\zeta\cos\zeta - \zeta] + \tfrac{5}{2}n^2e'^4\sin\zeta\cos\zeta \\ &+ \tfrac{1}{16}m^2e'^4\left[11\zeta - 13\sin\zeta\cos\zeta - 8\zeta\cos^2\zeta + 10\sin\zeta\cos^3\zeta\right] \\ &+ \tfrac{1}{2}mne'^4\left[3\sin\zeta + 2\zeta\cos\zeta - 5\sin\zeta\cos^2\zeta\right]\end{aligned} \tag{2–235}$$

where

$$m = \tfrac{1}{2}\left[1 + \tfrac{1}{2}e'^2\sin^2\beta_1\right]\left[1 - \cos^2\beta_1\sin^2\psi\right] \tag{2–236}$$

$$n = \tfrac{1}{2}\left[1 + \tfrac{1}{2}e'^2\sin^2\beta_1\right]\left[\sin^2\beta_1\cos\zeta + \cos\beta_1\cos\psi\sin\beta_1\sin\zeta\right] \tag{2–237}$$

$$\zeta = \frac{\Delta r_S}{R_p} \tag{2–238}$$

and

$$e' = \frac{1}{\sqrt{1 - e^2}} = \sqrt{\frac{R_\oplus^2 + R_p^2}{R_p^2}} \tag{2–239}$$

The angle β_1, defined the same way as in Sodano's inverse method, is computed with Equation (2–224), and the earth's eccentricity e is defined in Section 2.1.5. After the angle ϕ is computed, the geodetic latitude of point P is given by

$$\tan\delta_{gd_2} = \frac{\tan\beta_2}{(1-f)} = \frac{R_\oplus \sin\beta_2}{R_p \cos\beta_2} \qquad (2\text{–}240)$$

where

$$\sin\beta_2 = \sin\beta_1 \cos\phi + \cos\beta_1 \cos\psi \sin\phi \qquad (2\text{–}241)$$

and

$$\cos\beta_2 = \sqrt{\cos^2\beta_1 \sin^2\psi + (\sin\beta_1 \sin\phi - \cos\beta_1 \cos\psi \cos phi)^2} \qquad (2\text{–}242)$$

The longitude of point P is calculated from

$$\lambda_2 = \lambda_1 + l + \tan^{-1}\left[\frac{\sin\phi \sin\psi}{\cos\beta_1 \cos\phi - \sin\beta_1 \sin\phi \cos\psi}\right] \qquad (2\text{–}243)$$

where

$$l = \cos\beta_1 \sin\psi\left[-f\zeta + 3f^2 n \sin\zeta + \tfrac{3}{2}f^2 m \cdot (\zeta - \sin\zeta\cos\zeta)\right] \qquad (2\text{–}244)$$

And finally the backward azimuth from point P to the reference point is obtained from

$$\tan\psi_{back} = \frac{-\cos\beta_1 \sin\psi}{\sin\beta_1 \sin\phi - \cos\beta_1 \cos\psi \cos\phi} \qquad (2\text{–}245)$$

Sodano's direct method is coded in the function *sodano_direct* in the *derivs* module.

Once the longitude and latitude of point P are known, the forward azimuth and distance from point P to the vehicle can be calculated with Sodano's inverse method. The two azimuth angles are used in Equation (2–234) to compute the error function for the downrange/crossrange Newton–Raphson search.

2.4.2. Radar Observations

Radar observations are simulated by tracking a vehicle's trajectory from an earth–fixed position representing the radar antenna. Trajectories can be observed from many radar locations.

For radar observations a local geodetic horizon coordinate system is defined with its origin at the radar antenna as shown in Figure 2–20. It is fixed with respect to the earth and a vector $\vec{r}_r$ can be defined that gives its position relative to the earth's center. This system has unit vectors $\hat{x}_r$, $\hat{y}_r$, and $\hat{z}_r$, aligned with the local geodetic horizon system, so they can be computed from Equations (2–19) through (2–21) in Section 2.1.5 using the station's position information.

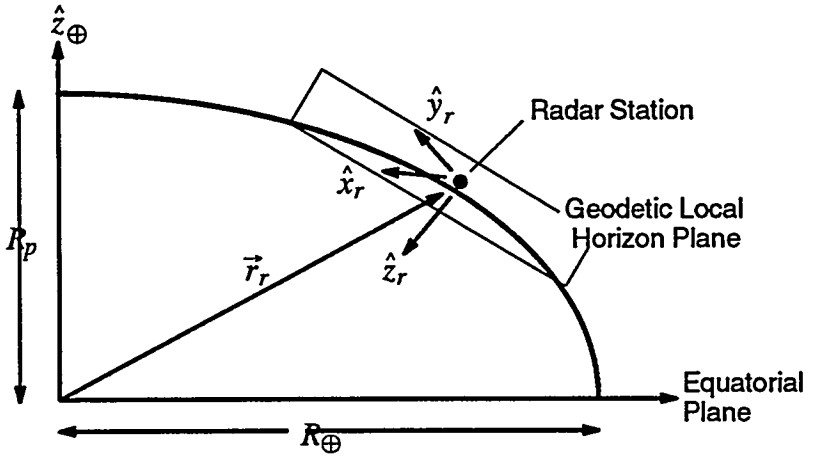

Figure 2–20. Radar Station Coordinates.

The position of the vehicle relative to the radar station is given by

$$\Delta\vec{r}_r = \vec{r} - \vec{r}_r \qquad (2\text{–}246)$$

and the radar range is the magitude of this vector, that is, $\|\Delta\vec{r}_r\|$.

A unit vector can be defined in this direction with

$$\hat{u}_r = \frac{\Delta\vec{r}_r}{\|\Delta\vec{r}_r\|} \qquad (2\text{–}247)$$

The azimuth α_r and elevation angle ε_r of the vehicle relative to the radar station, shown in Figure 2–21, are defined as

$$\tan\alpha_r = \frac{\Delta\vec{r}_r \cdot \hat{y}_r}{\Delta\vec{r}_r \cdot \hat{x}_r} \qquad (2\text{–}248)$$

$$\sin\varepsilon_r = -\hat{u}_r \cdot \hat{z}_r \qquad (2\text{–}249)$$

The azimuth angle α_r is measured in the geodetic local horizon plane containing the unit vectors $\hat{x}_r$ and $\hat{y}_r$. It is the angle from the vector $\hat{x}_r$ so that 0° is north and 90° is east. The elevation angle ε_r is measured from the geodetic local horizon plane and is positive when oriented away from the earth.

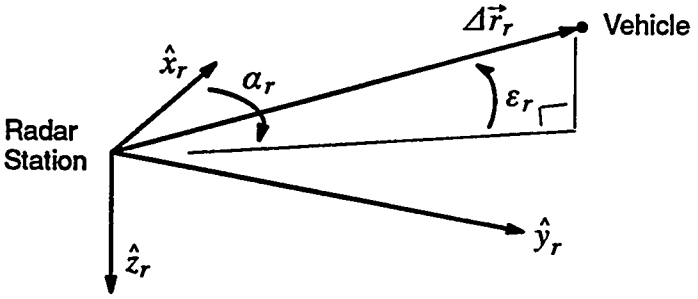

Figure 2–21. Radar Observations.

Expressions for the rates and accelerations of the radar range, azimuth angle, and elevation angle are obtained by differentiating the above equations. Range rate and acceleration are along the unit vector $\hat{u}_r$ and are given by

$$\|\dot{\Delta\vec{r}}_r\| = \vec{V}_\oplus \cdot \hat{u}_r \tag{2–250}$$

$$\|\ddot{\Delta\vec{r}}_r\| = \vec{a}_\oplus \cdot \hat{u}_r + \vec{V}_\oplus \cdot \dot{\hat{u}}_r \tag{2–251}$$

where the derivatives of the unit vector are

$$\dot{\hat{u}}_r = \frac{\vec{V}_\oplus - (\vec{V}_\oplus \cdot \hat{u}_r)\hat{u}_r}{\|\Delta\vec{r}_r\|} \tag{2–252}$$

$$\ddot{\hat{u}}_r = \frac{\vec{a}_\oplus - \|\ddot{\Delta\vec{r}}_r\|\,\hat{u}_r - 2(\vec{V}_\oplus \cdot \hat{u}_r)\dot{\hat{u}}_r}{\|\Delta\vec{r}_r\|} \tag{2–253}$$

The first two derivatives of the azimuth angle are

$$\dot{\alpha}_r = \frac{(\vec{V}_\oplus \cdot \hat{y}_r)(\Delta\vec{r}_r \cdot \hat{x}_r) - (\vec{V}_\oplus \cdot \hat{x}_r)(\Delta\vec{r}_r \cdot \hat{y}_r)}{(\Delta\vec{r}_r \cdot \hat{x}_r)^2 + (\Delta\vec{r}_r \cdot \hat{y}_r)^2} \tag{2–254}$$

$$\ddot{\alpha}_r = \frac{(\vec{a}_\oplus \cdot \hat{y}_r)(\Delta\vec{r}_r \cdot \hat{x}_r) - (\vec{a}_\oplus \cdot \hat{x}_r)(\Delta\vec{r}_r \cdot \hat{y}_r)}{(\Delta\vec{r}_r \cdot \hat{x}_r)^2 + (\Delta\vec{r}_r \cdot \hat{y}_r)^2} \tag{2–255}$$

$$- \frac{2\dot{\alpha}_r \cdot \left[(\vec{V}_\oplus \cdot \hat{x}_r)(\Delta\vec{r}_r \cdot \hat{x}_r) + (\vec{V}_\oplus \cdot \hat{y}_r)(\Delta\vec{r}_r \cdot \hat{y}_r)\right]}{(\Delta\vec{r}_r \cdot \hat{x}_r)^2 + (\Delta\vec{r}_r \cdot \hat{y}_r)^2}$$

and the derivatives of the elevation angle are

$$\dot{\varepsilon}_r = -\dot{\hat{u}}_r \cdot \hat{z}_r \sec\varepsilon_r \tag{2–256}$$

$$\ddot{\varepsilon}_r = \left[\dot{\varepsilon}_r^2 \sin\varepsilon_r - \ddot{\hat{u}}_r \cdot \hat{z}_r\right]\sec\varepsilon_r \tag{2–257}$$

The radar aspect angle η_r is the angle between the vehicle longitudinal $\hat{x}_b$ axis and the line of sight to the radar station $\hat{u}_r$ as shown in Figure 2–22. The meridional angle

ϕ_r is the angle produced by the radar line–of–sight vector $\hat{u}_r$ projected onto the vehicle plane formed by $\hat{y}_b$ and $\hat{z}_b$. These angles are analogous to the total angle of attack and the windward meridian.

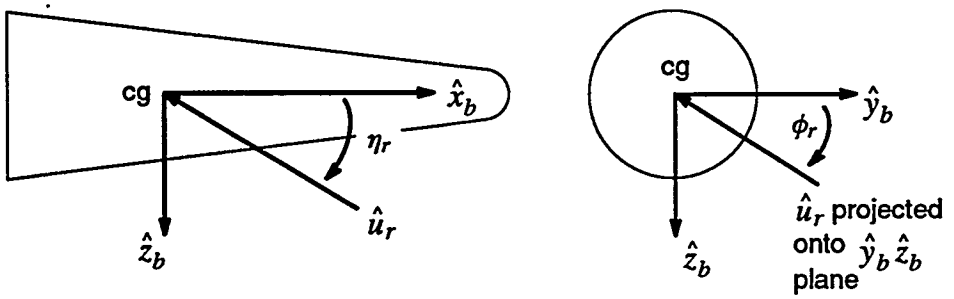

Figure 2–22. Radar Aspect and Meridional Angles.

The aspect angle is defined as

$$\sin\eta_r = \sqrt{(\hat{u}_r \cdot \hat{y}_b)^2 + (\hat{u}_r \cdot \hat{z}_b)^2} \qquad (2\text{–}258)$$

$$\cos\eta_r = -\hat{u}_r \cdot \hat{x}_b \qquad (2\text{–}259)$$

and the meridional angle is defined as

$$\sin\phi_r = \frac{-\hat{u}_r \cdot \hat{z}_b}{\sqrt{(\hat{u}_r \cdot \hat{y}_b)^2 + (\hat{u}_r \cdot \hat{z}_b)^2}} \qquad (2\text{–}260)$$

$$\cos\phi_r = \frac{-\hat{u}_r \cdot \hat{y}_b}{\sqrt{(\hat{u}_r \cdot \hat{y}_b)^2 + (\hat{u}_r \cdot \hat{z}_b)^2}} \qquad (2\text{–}261)$$

The radar observations are computed in the function *radar_out* in the *derivs* module.

2.4.3. Relative Vehicle Calculations

When trajectories are calculated for more than one vehicle, the position of each vehicle relative to the other can be computed. For a given vehicle, the position of other vehicles with respect to the vehicle's body coordinate system is illustrated in Figure 2–23. It is defined one of two ways: with a range, azimuth angle, and elevation angle, or with x, y, and z coordinates in the body coordinate system.

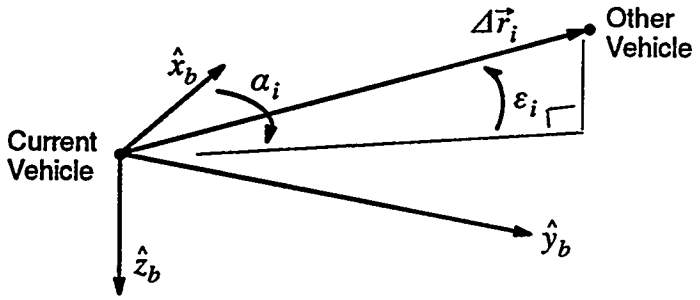

Figure 2–23. Relative Vehicle Calculations.

The orientation of the vehicle's body–fixed coordinate system is given from the guidance rules. It is an input value, and it can change instantaneously. Therefore, rates and accelerations in the body coordinate system cannot be computed.

The relative range $\|\Delta\vec{r}_i\|$, azimuth angle α_i, and elevation angle ε_i are defined the same way as for the radar observations, but they are referenced to the current vehicle's body coordinate system. Thus, the relative position of the i^{th} vehicle is given by

$$\Delta\vec{r}_i = \vec{r}_i - \vec{r} \qquad (2\text{–}262)$$

A unit vector can be defined in this direction with

$$\hat{u}_i = \frac{\Delta\vec{r}_i}{\|\Delta\vec{r}_i\|} \qquad (2\text{–}263)$$

and then the azimuth and elevation angles are defined as

$$\tan\alpha_i = \frac{\Delta\vec{r}_i \cdot \hat{y}_b}{\Delta\vec{r}_i \cdot \hat{x}_b} \qquad (2\text{–}264)$$

$$\sin\varepsilon_i = -\hat{u}_i \cdot \hat{z}_b \qquad (2\text{–}265)$$

The x, y, and z components of the relative position vector $\Delta\vec{r}_i$ are computed by projecting the relative position vector onto the body unit vectors with dot products.

The relative vehicle calculations are performed in the function *vehicle_obs* in the *path* module.

2.4.4. Initial Impact Point Calculations

The initial impact point (IIP) is calculated by integrating a ballistic trajectory from the vehicle's current state to impact with no propulsive forces and a simplified, constant drag coefficient, aerodynamic model. The drag coefficient, the aerodynamic reference area, and the vehicle's weight are given by an input ballistic coefficient β_{iip}. The IIP is an estimate of the impact point if the propulsive system fails during flight and is often required for range safety analysis.

The equations of motion, given in Section 2.2, are used for the IIP calculations. The contributions to acceleration include the acceleration from gravity $\vec{a}_{grav}$ and the acceleration from aerodynamic forces $\vec{a}_{aero}$. The acceleration from propulsive forces is zero.

The acceleration from gravity is computed as shown in Section 2.3.4. The acceleration from aerodynamic forces is computed from

$$\vec{a}_{aero} = \frac{\vec{F}_{aero}}{m} = \left(\frac{C_D\, q\, S_{ref}}{m}\right) \hat{x}_{V_w} = \left(\frac{C_D\, S_{ref}\, q\, g}{W}\right) \hat{x}_{V_w} \qquad (2\text{–}266)$$

$$= \left(\frac{q\, g}{\beta_{iip}}\right) \hat{x}_{V_w} \qquad (2\text{–}267)$$

$$= \frac{g\left(0.5\, \varrho \,\|\, \vec{V}_{W_\oplus} \|^2\right)}{\beta_{iip}}\, \hat{x}_{V_w} \qquad (2\text{–}268)$$

As shown in Section 2.1.8, the unit vector $\hat{x}_{V_w}$ is defined as

$$\hat{x}_{V_w} = \frac{\vec{V}_{W_\oplus}}{\| \vec{V}_{W_\oplus} \|} \qquad (2\text{–}269)$$

so the acceleration from aerodynamic forces simplifies to

$$\vec{a}_{aero} = 0.5 \cdot g \cdot \varrho \cdot \| \vec{V}_{W_\oplus} \| \cdot \vec{V}_{W_\oplus} \,/\, \beta_{iip} \qquad (2\text{–}270)$$

For the IIP calculation the equations of motion are integrated with an adaptive Runge–Kutta integration method from Fehlberg.[9] This method integrates with the 4th–order Runge–Kutta method shown in Section 2.2.1, but it uses results from a 5th–order Runge–Kutta integration to estimate the integration error and automatically adjust the integration step size. The adaptive step size method is ideal for the IIP calculation because only the final result is used; the trajectory is not divided into segments and results are not required at print intervals.

The integration step size is adjusted from 0.5 to 60 seconds such that the integration error is between 0.001 and 0.01 ft. Typical step sizes are 10 to 20 seconds. Even with

these large step sizes, the IIP calculations require a significant amount of processing time.

The impact time, longitude, and geodetic latitude are determined from the integration. The IIP range and azimuth are measured from the origin of the tangent plane coordinate system and are computed with Sodano's inverse method as shown in Section 2.4.1.

The function *iip_calc* in the *derivs* module performs the numerical integration for the IIP calculation. It calls the function *iip_derivs* to compute the derivatives of the equations of motion. The IIP output variables are computed in the function *iip_out.*

2.4.5. Aerodynamic Variables

The ballistic coefficient, defined by

$$\beta_c = \frac{W}{C_D \cdot S_{ref}} = \frac{m \cdot g}{C_D \cdot S_{ref}} \qquad (2\text{–}271)$$

is a useful design parameter for reentry vehicles. It is a measure of how fast the reentry vehicle decelerates as it enters the earth's atmosphere. Vehicles with low ballistic coefficients, such as 500 lb_f/ft^2, decelerate rapidly and have low aerodynamic heating, whereas vehicles with high ballistic coefficients, such as 4000 lb_f/ft^2, do not decelerate as fast and have high aerodynamic heating.

The lift–to–drag ratio *L/D*, given by

$$L/D = \frac{C_L \cdot q \cdot S_{ref}}{C_D \cdot q \cdot S_{ref}} = \frac{C_L}{C_D} \qquad (2\text{–}272)$$

is a measure of the aerodynamic efficiency of a vehicle. An unpowered vehicle glides furthest when flying at the angle of attack for maximum lift–to–drag ratio $\alpha_{L/D}$.

Figure 2–24 shows *L/D* as a function of angle of attack for a typical vehicle. The angle of attack for maximum *L/D* is found with a simple golden–section search (Section 2.6.2). This method is used because it converges despite possible discontinuities in the *L/D* function. The maximum L/D calculations are performed in the *maxlod* function in the *derivs* module.

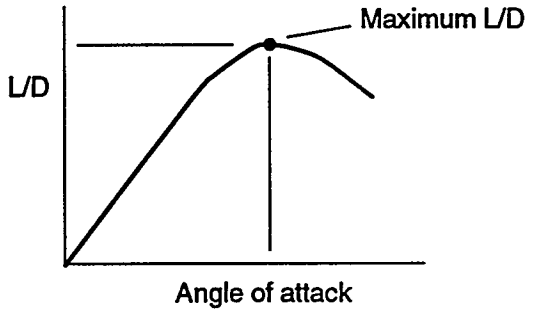

Figure 2–24. Lift–to–Drag Ratio.

2.5. Guidance Rules

Because TAOS is a 3–DOF point–mass trajectory simulation, the body attitude or angular orientation cannot be obtained from the equations of motion. Instead the body attitude is input in the form of guidance rules. It is assumed that the vehicle has a control system that can maintain this body attitude.

The body attitude, given by the body–fixed coordinate system, is defined relative to one of the other coordinate systems with three angles. If defined relative to the geocentric, geodetic, or inertial platform coordinate systems, three Euler angles must be provided. If defined relative to a velocity coordinate system, a pair of aerodynamic angles and a bank angle must be provided. Regardless of the method, the body attitude requires three angles. Table 2–2 contains a list of the valid combinations of angles that can be used to define the body attitude.

Table 2–2. Body–Attitude Control Variables.

Set	Body–Attitude Angles		
1	α Angle of attack	β_E Euler sideslip angle	μ_{gc} Geocentric bank angle
2	α Angle of attack	β Sideslip angle	μ_{gc} Geocentric bank angle
3	α_T Total angle of attack	ϕ_w Windward meridian	μ_{gc} Geocentric bank angle
4	α Angle of attack	β_E Euler sideslip angle	μ_{gd} Geodetic bank angle
5	α Angle of attack	β Sideslip angle	μ_{gd} Geodetic bank angle
6	α_T Total angle of attack	ϕ_w Windward meridian	μ_{gd} Geodetic bank angle
7	Ψ_{gc} Geocentric yaw angle	Θ_{gc} Geocentric pitch angle	Φ_{gc} Geocentric roll angle
8	Ψ_{gd} Geodetic yaw angle	Θ_{gd} Geodetic pitch angle	Φ_{gd} Geodetic roll angle
9	Ψ_p Inertial yaw angle	Θ_p Inertial pitch angle	Φ_p Inertial roll angle

Control Variables

The vehicle acceleration, and therefore, its motion, is determined from the forces acting on it. These forces are generally a function of the aerodynamic angles, which in turn are functions of the body attitude angles. These angles control the forces which control the trajectory, so these variables can be thought of as control variables.

Another way to think of this is to observe the functional dependency of the vehicle acceleration. Acceleration is a function of the total force acting on the vehicle which includes aerodynamic, propulsive, and gravitational forces. For a vehicle with fixed geometry, the aerodynamic forces are commonly functions of altitude, velocity, angle of attack, and sideslip angle. Propulsive forces are typically functions of altitude, velocity, and power setting. The altitude and velocity are known from the vehicle's state, but the angle of attack, sideslip, and power setting are independent variables or control variables whose values must be given in order to integrate the equations of motion.

TAOS uses four control variables. Three variables are from one of the sets of body attitude angles given in Table 2–2, and the fourth variable is power setting P. The control variables affect the forces acting on the vehicle only if the forces, defined in tables, are functions of them. For example, if the propulsion tables are not a function of power setting, then power setting has no effect on the propulsive force and no effect on the trajectory.

Guidance Rules

Values for the control variables are determined from guidance rules. These rules are input and they either directly or indirectly give values for the control variables. There are four control variables, so four guidance rules must be provided.

An example of a guidance rule that directly gives a value for a control variable is "fly at a 5° angle of attack." A value for angle of attack, a control variable in Table 2–2, has been directly specified, so no further action is required.

Values for any of the control variables in Table 2–2 can be given directly, but they must form a consistent set. All of the control variables directly specified must be in one of the sets in Table 2–2, that is, they must all be on the same row in the table.

An example of a guidance rule that indirectly specifies a control variable is "fly at a 0° vertical flight path angle," in other words, "fly level." To maintain level flight, the vehicle must be at some angle of attack, but this value is unknown. So TAOS must solve for the required angle of attack to maintain level flight.

Table 2–3 contains a list of guidance rules indirectly specifying control variables. Each guidance rule has one or more control variables associated with it. TAOS tries to solve for the value of one of these control variables that satisfies the guidance rule.

Table 2–3. Guidance Rules that Indirectly Specify Control Variables.

Guidance Rule	Control Variables	Solution Variable
Fly at a constant altitude	α, α_T, P	$\ddot{h}$
Fly at a constant lift coefficient	α, α_T	C_L
Fly at a constant side–force coefficient	β, β_E, ϕ_w	C_S
Fly at a constant dynamic pressure	P, α, α_T	$\ddot{q}$
Fly at a constant geocentric flight path angle	α, α_T, P	$\dot{\gamma}_{gc}$
Fly at a constant geodetic flight path angle	α, α_T, P	$\dot{\gamma}_{gd}$
Fly intercept or proportional nav in pitch	α, α_T	$\dot{\gamma}_{gd}$
Fly intercept or proportional nav in yaw	$\beta, \beta_E, \phi_w, \mu_{gc}, \mu_{gd}$	$\dot{\psi}_{gd}$
Fly at a constant lift–to–drag ratio	α, α_T	L/D
Fly at the maximum lift–to–drag ratio	α, α_T	$\alpha_{L/D}$

Fly at a constant Mach number	P, α, α_T	$\dot{M}$
Fly at a constant axial–g loading	P, α, α_T	$\vec{a}_{sp} \cdot \hat{x}_b$
Fly at a constant lateral–g loading	β, β_E, ϕ_w	$\vec{a}_{sp} \cdot \hat{y}_b$
Fly at a constant normal–g loading	α, α_T	$\vec{a}_{sp} \cdot \hat{z}_b$
Fly at a constant geocentric heading angle	$\beta, \beta_E, \phi_w, \mu_{gc}, \mu_{gd}$	$\dot{\psi}_{gc}$
Fly at a constant geodetic heading angle	$\beta, \beta_E, \phi_w, \mu_{gc}, \mu_{gd}$	$\dot{\psi}_{gd}$
Fly at a constant thrust	P	$\lVert \vec{F}_{prop} \rVert$
Fly up/down range insensitive axis in pitch	α, α_T	r_{iip}
Fly up/down range insensitive axis in yaw	$\beta, \beta_E, \phi_w, \mu_{gc}, \mu_{gd}$	α_{iip}
Fly at a constant velocity	P, α, α_T	$\lVert \dot{\vec{V}}_{\oplus} \rVert$

The solution variable in Table 2–3 is a function of the control variables and is related to the guidance rule. It is generally an acceleration value that can change instantaneously in the trajectory.

2.5.1. Control Variable Solution Process

TAOS analyzes the input guidance rules and determines which rules directly specify control variables and which do not. It compares the list of control variables directly specified with those in Table 2–2 and selects the control variable set with the lowest number that matches the ones input. If the input control variables do not match one of the sets, an inconsistent set of guidance rules has been input and an error occurs.

Control variables directly specified are set to the input values, and a list of the control variables and guidance rules indirectly specified is kept.

The solution process for the indirectly specified control variables is iterative. This procedure, shown in Figure 2–25, is called the guidance loop. It is part of the derivative calculation for the equations of motion.

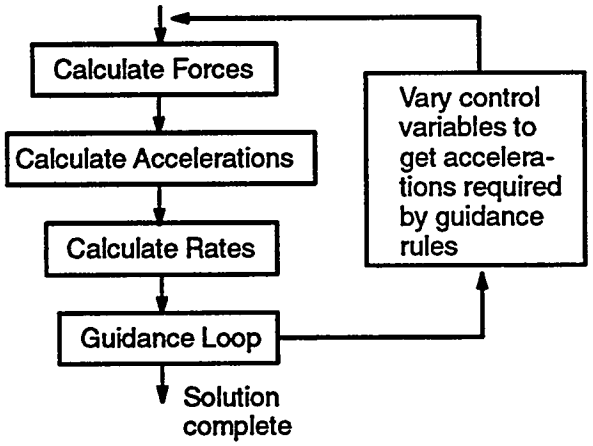

Figure 2–25. Guidance Loop.

Accelerations can change instantaneously along the trajectory so each guidance rule is associated with an acceleration or "solution" variable as shown in Table 2–3. A desired value for the acceleration variable is obtained from the guidance rule, and a control variable is varied until the acceleration variable is equal to the desired value. In other words, the indirectly specified control variables are varied until the difference between the computed and desired values of the acceleration variables is zero. This results in a set of nonlinear equations that can be solved using a multi–dimensional Newton–Raphson method.

In some cases, such as C_L and n_x, the value of the acceleration variable is directly given in the guidance rule. This is the desired value used in the guidance loop.

In other cases, such as altitude or flight path angle, the value of the acceleration variable is not given in the guidance rule. The guidance rule gives a desired value for the time integral or double integral of the solution variable. The guidance rule is converted into a desired acceleration by assuming that the guidance rule is a 2nd or 3rd order function of time that transitions from the current value to the desired value.

Parabolic Transition

The guidance rule "fly a constant geodetic vertical flight path angle equal to 5 degrees" gives a desired value for the acceleration variable $\dot{\gamma}_{gd}$ and its integral γ_{gd}, that is, $\gamma_{gd}' = 5$ and $\dot{\gamma}_{gd}' = 0$. The prime is used to indicate the desired value.

The current value of γ_{gd} may or may not be 5 degrees, but for this example, it is assumed to be greater than 5 degrees. TAOS assumes that γ_{gd} should be a 2nd order parabolic function of time that transitions from the current value to the desired value within a time interval Δt_{guid} as shown in Figure 2–26.

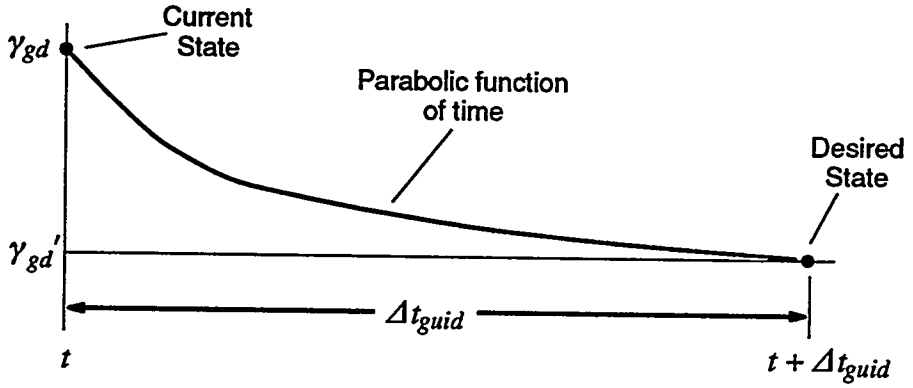

Figure 2–26. Parabolic Transition to Desired State.

The value of Δt_{guid}, input by the user, acts as a time constant. Large values of Δt_{guid} cause small corrections to the flight path, and it takes a long time for the trajectory to reach the desired conditions. Small values of Δt_{guid} cause large corrections and the trajectory reaches the desired conditions quickly. As values of Δt_{guid} get small, the corrections may be too large and the trajectory oscillates until the desired value is reached. Values of Δt_{guid} can be too small causing the trajectory to diverge from the desired value.

The desired value of the acceleration variable, $\dot{\gamma}_{gd}'$ in this case, is the derivative of the parabolic curve evaluated at the current time. The current time t and current value of γ_{gd} are known as well as the desired value γ_{gd}' and its derivative $\dot{\gamma}_{gd}'$ at time $t + \Delta t_{guid}$. A parabolic curve of the form

$$\gamma_{gd}'(t) = at^2 + bt + c \qquad (2\text{–}273)$$

is fit through these values. Its derivative is

$$\dot{\gamma}_{gd}'(t) = 2at + b \qquad (2\text{–}274)$$

where

$$a = \frac{\gamma_{gd} - \gamma_{gd}' + \dot{\gamma}_{gd}' \cdot \Delta t_{guid}}{\Delta t_{guid}^2} \qquad (2\text{–}275)$$

$$b = \dot{\gamma}_{gd}' - 2a(t + \Delta t_{guid}) \qquad (2\text{–}276)$$

This procedure, in function *parab_guidance_correction* in the *derivs* module, is used for all of the single–derivative acceleration variables, such as $\dot{\gamma}_{gd}$, $\dot{\psi}_{gd}$, and $\| \dot{\vec{V}} \|$.

Cubic Transition

A similar procedure is used for the double derivative acceleration variables, such as $\ddot{h}$, except that a 3rd order curve is fit through the known information in the function *cubic_guidance_correction* (Figure 2–27).

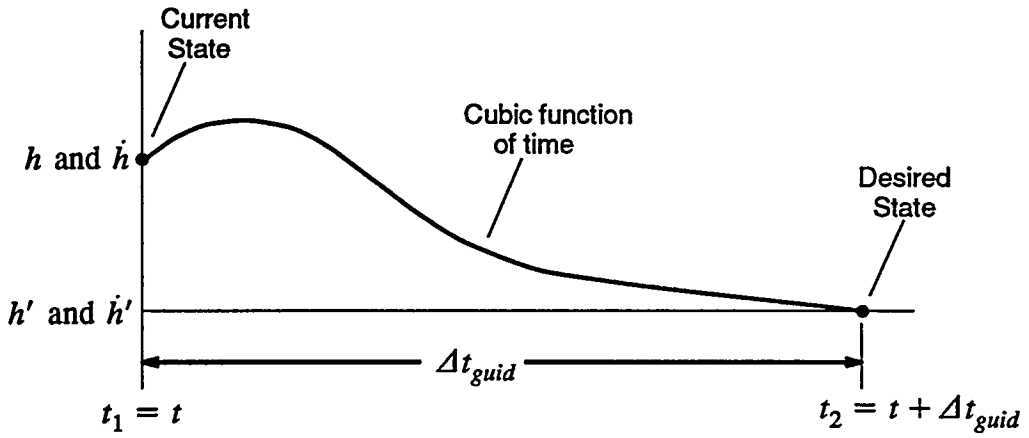

Figure 2–27. Cubic Transition to Desired State.

In this case, the current altitude and altitude rate are known along with the desired altitude and altitude rate. With four known values, a cubic curve of the form

$$h'(t) = at^3 + bt^2 + ct + d \qquad (2\text{–}277)$$

can be used to transition from the current state to the desired state. Differentiating this function twice gives

$$\ddot{h}'(t) = 6at + 2b \qquad (2\text{–}278)$$

where

$$b = \frac{(h - h')(t_2^2 - t_1^2) + \dot{h}(t_2 - t_1)(t_2^2 - t_1^2) + (\dot{h}' - \dot{h})(\frac{2}{3}t_1^3 - t_1^2 t_2 + \frac{1}{3}t_2^3)}{2(t_2 - t_1)(\frac{2}{3}t_1^3 - t_1^2 t_2 + \frac{1}{3}t_2^3) - (t_2^2 - t_1^2)(t_1^2 - 2t_1 t_2 + t_2^2)} \qquad (2\text{–}279)$$

$$a = \frac{\dot{h}' - \dot{h} - 2b(t_2 - t_1)}{3(t_2^2 - t_1^2)} \qquad (2\text{–}280)$$

Newton–Raphson Solution

The remaining part of the solution process solves for the values of the control variables giving the desired accelerations. The control variables are placed in a vector $\vec{x}$ and the solution functions, which are the differences between the desired acceleration and the actual acceleration $f_i = f - f'$, are placed in another vector $\vec{f}$.

A multi–dimensional Newton–Raphson method can be used to find the values $\vec{x}$ such that the vector $\vec{f}$ is zero.[8] This method starts from an initial estimate of the vector $\vec{x}_n$ and gets a new estimate from

$$\vec{x}_{n+1} = \vec{x}_n - [J]^{-1} \cdot \vec{f} \qquad (2\text{–}281)$$

where $[J]$ is the Jacobian matrix of $\vec{f}$ with respect to $\vec{x}$, that is,

$$[J] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_1} \cdots \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_2} \cdots \\ \frac{\partial f_1}{\partial x_3} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_3} \cdots \\ & \textit{etc.} & \end{bmatrix} \qquad (2\text{–}282)$$

The Jacobian matrix $[J]$ and the vector $\vec{f}$ are evaluated at $\vec{x}_n$, to get a new estimate of $\vec{x}_{n+1}$. The subscript n denotes the iteration number.

This procedure can be simplified by substituting $\vec{z}_{n+1} = \vec{x}_{n+1} - \vec{x}_n$, which results in

$$[J] \cdot \vec{z}_{n+1} = -\vec{f} \qquad (2\text{–}283)$$

This is a set of linear equations in $\vec{z}$, so it can be solved with Gaussian elimination.[8] Gaussian elimination is more efficient than the matrix inversion required to solve Equation (2–281).

The Newton–Raphson procedure converges faster if the initial estimates are close to the solution. For the first point in a trajectory, the initial estimates for the control variable values are halfway between their upper and lower limits. After the first trajectory point has been calculated, the initial estimates for the next trajectory point are set to the solutions from the previous point. This assumes the control variables are not changing rapidly during the trajectory which is normally true.

2.5.2. Intercepts and Proportional Navigation

Intercept trajectories can be computed three ways in TAOS: with optimization, with predictive guidance, and with proportional navigation guidance. When optimization is used, the trajectory final conditions are constrained to intercept the target. Optimization varies the trajectory shape to meet these constraints and to maximize or minimize another quantity such as time or velocity. Many trajectories are computed in an effort to find the best one. An example of an intercept trajectory computed with optimization is shown in Section 4.5.3.

The other two methods compute an intercept trajectory directly. These trajectories are not optimum, but no iteration is required. At each point along the intercept trajectory, the target state is observed and the vehicle trajectory is modified in an effort to hit the target.

The predictive guidance method assumes the target trajectory is nonaccelerating and estimates a straight line intercept point. Then it modifies the trajectory to fly towards the estimated intercept point.

The proportional navigation method tries to keep the line–of–sight angle from the vehicle to the target constant, in other words, it tries to keep the line–of–sight rate zero. It commands accelerations proportional to the line–of–sight rate.

In all cases the intercept trajectories are idealistic in that the guidance and control system is not modeled. TAOS also does not model the radar or sensors used to detect and track the target. Because of this, the intercept trajectories cannot be used to predict miss distances. But the trajectories are useful to predict overall performance capability, for example, time and range to intercept.

Predictive Guidance

The intercept geometry used for predictive guidance is shown in Figure 2–28, where vehicle 1 is the interceptor and vehicle 2 is the target. If the interceptor and target are assumed to be nonaccelerating and they are assumed to be at the same position at the intercept time t_i, then

$$\vec{r}_1(t) + (t_i - t) \cdot \vec{V}_1 = \vec{r}_2(t) + (t_i - t) \cdot \vec{V}_2 \qquad (2\text{–}284)$$

where t is the current time. This equation can be solved for the time to intercept giving

$$t_i - t = \frac{\| \vec{r}_2 - \vec{r}_1 \|}{\| \vec{V}_1 - \vec{V}_2 \|} \qquad (2\text{–}285)$$

so the predicted intercept point is

$$\vec{r}_i = \vec{r}_2(t) + \vec{V}_2 \cdot \frac{\| \vec{r}_2 - \vec{r}_1 \|}{\| \vec{V}_1 - \vec{V}_2 \|} \qquad (2\text{–}286)$$

Horizontal and vertical geodetic flight path angles are determined for a straight line path from the interceptor to the predicted intercept point given by $\Delta\vec{r}$. These flight

path angles are used as guidance rules for the solution method shown in the previous section. These calculations are performed in the function *intercept* in the *derivs* module.

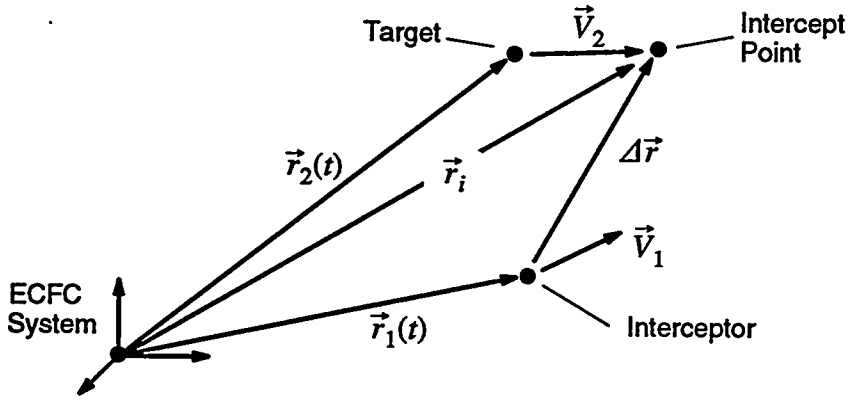

Figure 2–28. Intercept Geometry.

Proportional Navigation

Proportional navigation commands accelerations perpendicular to the line–of–sight angle and proportional to its rate.[17] The geometry is shown in Figure 2–29, where vehicle 1 is the interceptor and vehicle 2 is the target.

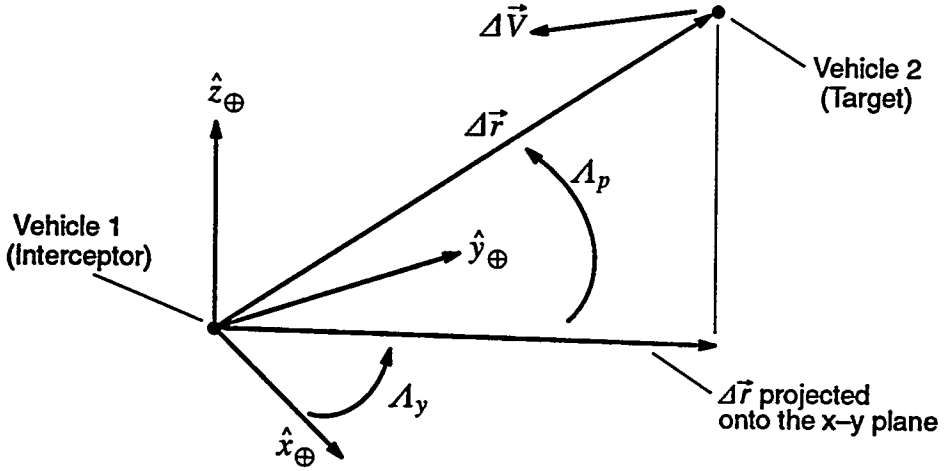

Figure 2–29. Proportional Navigation Geometry.

The line–of–sight vector $\Delta\vec{r}$ is the difference between the position vectors of the two vehicles. Its orientation with respect to the ECFC coordinate system is given by two angles: a line–of–sight yaw angle Λ_y and a line–of–sight pitch angle Λ_p. The line–of–sight yaw angle is defined as

$$\tan\Lambda_y = \frac{\Delta\vec{r} \cdot \hat{y}_\oplus}{\Delta\vec{r} \cdot \hat{x}_\oplus} \tag{2-287}$$

and the line–of–sight pitch angle is defined as

$$\tan\Lambda_p = \frac{\Delta\vec{r} \cdot \hat{z}_\oplus}{\sqrt{(\Delta\vec{r} \cdot \hat{x}_\oplus)^2 + (\Delta\vec{r} \cdot \hat{y}_\oplus)^2}} \tag{2-288}$$

These angles can be differentiated to get the line–of–sight angular rates

$$\dot{\Lambda}_y = \frac{(\Delta\vec{r} \cdot \hat{x}_\oplus) \cdot (\Delta\vec{V} \cdot \hat{y}_\oplus) - (\Delta\vec{r} \cdot \hat{y}_\oplus) \cdot (\Delta\vec{V} \cdot \hat{x}_\oplus)}{(\Delta\vec{r} \cdot \hat{x}_\oplus)^2 + (\Delta\vec{r} \cdot \hat{y}_\oplus)^2} \tag{2-289}$$

$$\dot{\Lambda}_p = \frac{(\Delta\vec{V} \cdot \hat{z}_\oplus)\left[(\Delta\vec{r} \cdot \hat{x}_\oplus)^2 + (\Delta\vec{r} \cdot \hat{y}_\oplus)^2\right] - (\Delta\vec{r} \cdot \hat{z}_\oplus)\left[(\Delta\vec{r} \cdot \hat{x}_\oplus)(\Delta\vec{V} \cdot \hat{x}_\oplus) + (\Delta\vec{r} \cdot \hat{y}_\oplus)(\Delta\vec{V} \cdot \hat{y}_\oplus)\right]}{\sqrt{(\Delta\vec{r} \cdot \hat{x}_\oplus)^2 + (\Delta\vec{r} \cdot \hat{y}_\oplus)^2}\left[(\Delta\vec{r} \cdot \hat{x}_\oplus)^2 + (\Delta\vec{r} \cdot \hat{y}_\oplus)^2 + (\Delta\vec{r} \cdot \hat{z}_\oplus)^2\right]} \tag{2-290}$$

The desired acceleration is proportional to these rates, that is,

$$n_y = N' \cdot V_c \cdot \dot{\Lambda}_y \tag{2-291}$$

$$n_p = N' \cdot V_c \cdot \dot{\Lambda}_p \tag{2-292}$$

The navigation constant N' is an input value, and it typically ranges from two to five. The closure velocity V_c is the projection of the relative velocity vector onto the line–of–sight vector, that is,

$$V_c = \frac{-\Delta\vec{V} \cdot \Delta\vec{r}}{\|\Delta\vec{r}\|} \tag{2-293}$$

The accelerations n_y and n_p are perpendicular to the line–of–sight vector, so they must be transformed to the ECFC coordinate system. A coordinate system with its x axis aligned with the line–of–sight vector is related to the ECFC by two rotations: (1) a rotation about the ECFC z axis by the angle Λ_y and (2) a rotation about the y axis of this system by the angle Λ_p. Thus, the rotation matrices are given by

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix}_{LOS} = \begin{bmatrix} \cos\Lambda_p & \sin\Lambda_p & 0 \\ -\sin\Lambda_p & \cos\Lambda_p & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Lambda_y & 0 & \sin\Lambda_y \\ 0 & 1 & 0 \\ -\sin\Lambda_y & 0 & \cos\Lambda_y \end{bmatrix} \begin{bmatrix} \hat{x}_\oplus \\ \hat{y}_\oplus \\ \hat{z}_\oplus \end{bmatrix} \tag{2-294}$$

(2-295)

The rotation matrices can be multiplied and then transposed to give the transformation for the accelerations

$$\begin{bmatrix} \vec{a}_{\oplus} \cdot \hat{x}_{\oplus} \\ \vec{a}_{\oplus} \cdot \hat{y}_{\oplus} \\ \vec{a}_{\oplus} \cdot \hat{z}_{\oplus} \end{bmatrix} = \begin{bmatrix} \cos\Lambda_p \cos\Lambda_y & -\sin\Lambda_p \cos\Lambda_y & -\sin\Lambda_y \\ \sin\Lambda_p & \cos\Lambda_p & 0 \\ \cos\Lambda_p \sin\Lambda_y & -\sin\Lambda_p \sin\Lambda_y & \cos\Lambda_y \end{bmatrix} \cdot \begin{bmatrix} 0 \\ n_p \\ n_y \end{bmatrix} \quad (2\text{–}297) \qquad (2\text{–}296)$$

The acceleration component aligned with the velocity vector can be ignored because the impact speed is not critical. The acceleration components normal to the velocity vector are important because these accelerations turn the trajectory so it hits the target.

The acceleration vector in ECFC coordinates $\vec{a}_{\oplus}$ can be converted to desired horizontal and vertical flight path angle rates with Equations (2–170) and (2–171) from Section 2.2.5. Values of the control variables that produce these accelerations can be obtained with the Newton–Raphson method as shown in Section 2.5.1.

The function *prop_nav* in the *derivs* module handles the proportional navigation calculations.

2.5.3. Range Insensitive Axis

If a vehicle is on a ballistic trajectory above the sensible atmosphere and its velocity is instantaeously incremented (a ΔV is applied), it normally changes the time and range to impact. However, if the ΔV is applied in a specific direction, called the range insensitive axis, only the time to impact changes, the range does not change as shown in Figure 2–30.

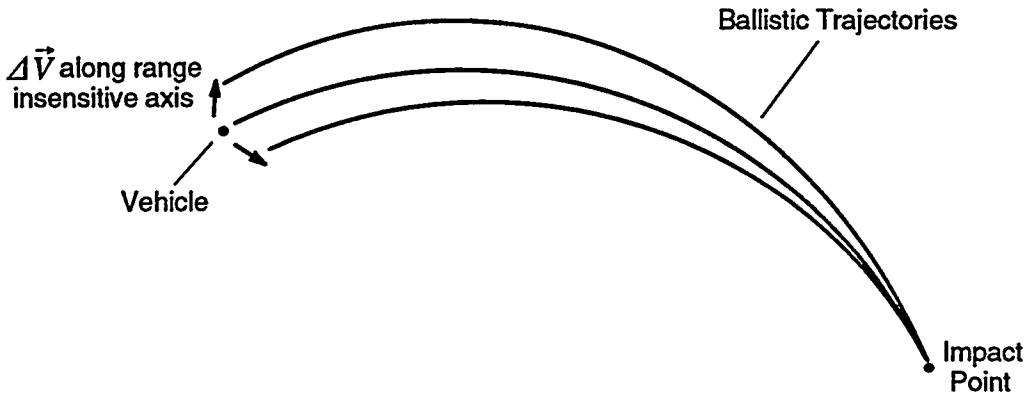

Figure 2–30. Range Insensitive Axis.

The range insensitive axis has two solutions: one that increases the time to impact, and one that decreases the time to impact. If a ΔV is applied along the range insensitive axis that increases the time to impact, the ΔV is said to be applied "up" the axis. If the ΔV decreases the time to impact, it is applied "down" the axis.

The range insensitive axis guidance rules align the body x axis with one of the range insensitive axes. The solution for the range insensitive axis begins by calculating the initial impact point range and azimuth from the current state with no ΔV applied (Section 2.4.4). Then a multi–dimensional Newton–Raphson search (Section 2.5.1) is used to vary the direction of a 10 ft/sec ΔV to give the same impact point. All impact points are computed with no aerodynamic forces.

The function *range_insensitive_axis* in the *derivs* module calculates the required pitch and yaw angles which define the body attitude.

2.5.4. Flight Path Limits

In addition to the guidance rules, limits or constraints can be imposed on the flight path. These constraints are usually applied directly to the control variables to act as upper and lower search boundaries for the Newton–Raphson method. But any guidance rule can be used as a flight path limit.

Limits on Specified Control Variables

If a limit is applied to a control variable that already has a specified value, the limit is ignored. For example, if a guidance rule says to "fly at a 25° angle of attack" and angle of attack has been limited to 15°, the limit is ignored and angle of attack is set to 25°.

Limits on Free Control Variables

If a limit is applied to one of the free control variables, that is, one of the controls that does not have a specific value, then the limit is applied during the Newton–Raphson search. The search procedure keeps the values of the control variables within default limits of $\pm 30°$ for the angle of attack and sideslip and within $\pm 180°$ for all other angles.

If the search fails to converge resulting in a flight path that does not follow the guidance rules, then one possible solution is to decrease the limits on the control variables to more reasonable values. Sometimes the aerodynamic tables do not extrapolate smoothly to angles as high as $\pm 30°$ so limiting the angles to less than $\pm 30°$ improves convergence.

Limits on Flight Conditions

If a limit is applied to a guidance rule that does not specify a control variable directly, then at each integration step, the trajectory values are compared to the limiting values. If a limit is exceeded, the limit becomes a new guidance rule replacing one of the existing guidance rules.

Each guidance rule is associated with a set of control variables. These control variables are prioritized in the order of the most likely to be used to satisfy the guidance rule. This list is used to obtain the control variable most likely associated with the exceeded limit. Then the guidance rule currently associated with that control variable is replaced with the limiting guidance rule.

For example, if a vehicle is climbing at an angle of attack and hits an altitude limit, the altitude limit replaces the guidance rule associated with angle of attack. The limiting guidance rule forces the trajectory to remain at a constant altitude equal to the limiting value.

2.6. Trajectory Calculations

The previous sections discuss different parts of the trajectory calculations, but they do not show how all the parts fit together. This section provides an overview of the trajectory calculations and how they are implemented in TAOS.

Main Program

When TAOS is executed, processing begins with the *main* function in the *main* module. Figure 2–31 is a flowchart showing the logic in this function. TAOS begins by creating two lists from the command line arguments: one containing a list of the table filenames and one containing a list of the problem filenames. Command line arguments are the filenames given in the command that runs TAOS (Section 1.5).

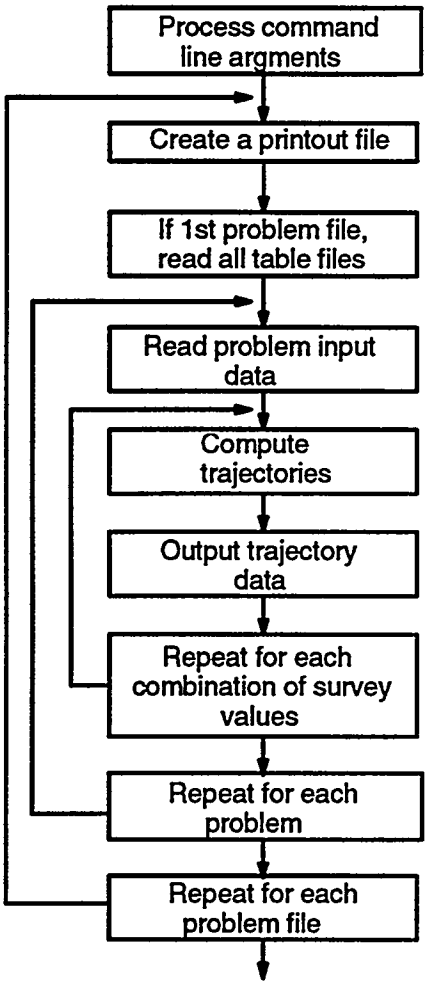

Figure 2–31. Main Flowchart.

A loop processes each input problem file. For each file, a corresponding printout file is created; however, this file remains empty until later in the function.

If it is the first problem file, then all table files are read and saved in memory. They remain in memory for all subsequent problem files.

Problem files can contain multiple problems so another loop is nested inside the problem file loop to process each problem. For each problem, the input data is read into memory and saved in a data structure.

This data structure begins with a linked list containing information for each vehicle or trajectory as shown in Figure 2–32. The data structure for each vehicle contains input and output data including input trajectory initial conditions, input segment definitions, and output trajectory variables at each integration time step. The information is organized so it is easy to retrieve. The data structure in TAOS contains many more records and lists than shown in Figure 2–32; Figure 2–32 shows only the major features of the data structure.

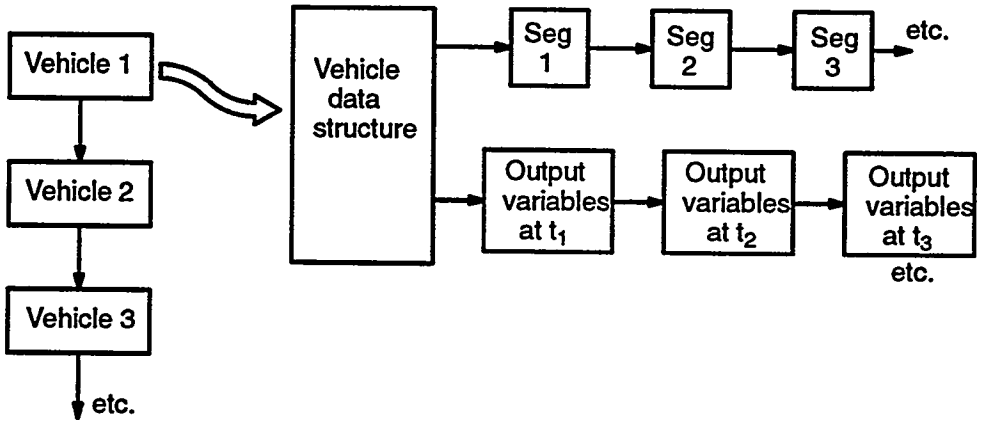

Figure 2–32. Vehicle Data Structure.

Use of dynamic memory allocation and linked lists for input and output data eliminates constraints on the amount of information that can be processed. In practice this means that there are few limitations on the number of vehicles that can be input, the number of segments for each vehicle, the amount of output data, or the length of the trajectory. These are only limited by the maximum amount of memory available on the computer.

After the problem input data has been read, trajectories are calculated for each vehicle and the trajectory data is written to the printout file. Other output files are also created at this point. No trajectory data is written to the printout file until all trajectories have been calculated so if the program ends prematurely, the output files are empty.

If surveys have been defined, the trajectories are recomputed for each combination of survey values.

Trajectory Calculations

The box in Figure 2–31 labeled "compute trajectories" can be expanded resulting in the flowchart in Figure 2–33. This flowchart represents the logic in function *compute_trajectories* in the *path* module.

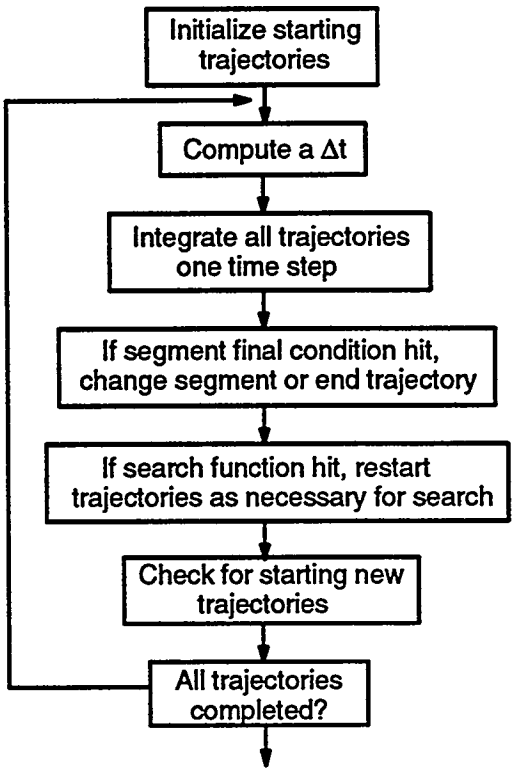

Figure 2–33. Trajectory Calculation Flowchart.

The trajectory calculations begin by locating the starting vehicle which is the vehicle with the minimum initial time. Trajectories can be initialized by directly specifying initial conditions or by using an initial state from an existing trajectory, but at least one trajectory must be initialized from directly specified initial conditions. A starting trajectory must exist.

The status for each trajectory, indicating whether a trajectory is active or inactive, is maintained throughout the trajectory calculations. A trajectory is active if it has been initialized and is currently being integrated, and a trajectory is inactive if it has not yet been initialized or if it has already been computed.

After the starting trajectories are initialized, the trajectory calculation loop is entered. It begins by selecting the minimum integration time step Δt for all vehicles. This time step is compared to the print time intervals and reduced if necessary so that calculations are performed at every print interval. If a guidance rule contains a table that is a function of time, the time step is adjusted to ensure that each table point is hit exactly.

Time is used to synchronize all trajectories, in other words, all trajectories are integrated with the same Δt step size. The time step calculations are in function *get_next_time_step* in the *path* module.

After the integration time step has been determined, the equations of motion for each active vehicle can be integrated one time step. Each vehicle is treated separately, that

is, each vehicle has its own equations of motion and is integrated separately. The vehicles are not combined into one large set of differential equations. The function *integration_step* in the *derivs* module contains the Runge–Kutta numerical integration procedure.

Segment Final Conditions

After each integration step, the segment final conditions on each trajectory are inspected in the function *check_final_conditions* in the *path* module. If a segment final condition boundary has been crossed, the last integration step is repeated while varying the integration time step until the final condition is met within a small tolerance. The secant method, a linear search method similar to Newton–Raphson, is used for this search process (Section 2.6.2).[8]

It is possible to have more than one segment final condition encountered in an integration step. There are two cases where this can happen: if two or more segments end at exactly the same time, or if two or more segments end at different times. Tests are made to check for the first case, and if it occurs, then both segments are ended simultaneously. If the second case occurs, the step size is repeatedly halved until only one final condition is hit. Then the secant search is used to find the step size that meets this final condition.

Search and Optimization Loops

Search and optimization loops, used to solve for trajectory constraints, can be defined in a problem. Both types of loops vary one or more input values to meet a goal. The goal is a simple function, often just the value of an output variable, that is evaluated at a specific point in the trajectory. In TAOS the search goals or objective functions can only be evaluated at the end of a segment.

At the end of each segment, the function *search_function_hit* in the *search* module is used to determine if a search has been defined with its objective function evaluated at this segment. If this occurs, then the input parameters for the search are modified according to the search or optimization algorithm and the trajectory is restarted. The parabolic search algorithms are given in Section 2.6.2, and an overview of the optimization algorithm is given in Section 2.6.3.

The trajectory is restarted just prior to the location of the modified input parameters; thus, only the part of the trajectory that changes is recomputed. This minimizes the amount of calculation required for the search and optimization loops.

Multiple Trajectories

When a final condition is encountered ending a segment, the trajectory may be continued with another segment or it may end. If the trajectory continues, a new segment is initialized and the trajectory calculation proceeds as shown above. During segment initialization inactive trajectories are inspected in the function *add_new_vehicles* in

the *path* module and if any use the current segment as a starting point, then they are initialized and added to the list of active trajectories.

If the end of the trajectory has been found, then the trajectory is marked as completed. The trajectory calculation process continues until all trajectories are completed.

2.6.1. Derivative Calculations

The derivatives for the equations of motion are calculated within the box labeled "integrate all trajectories one time step" in Figure 2–33. The function *integration_step* calculates the derivatives four times each time it propagates the trajectory forward one time step. The derivatives are calculated in the function *compute_derivs* in the *derivs* module as shown in the flowchart in Figure 2–34.

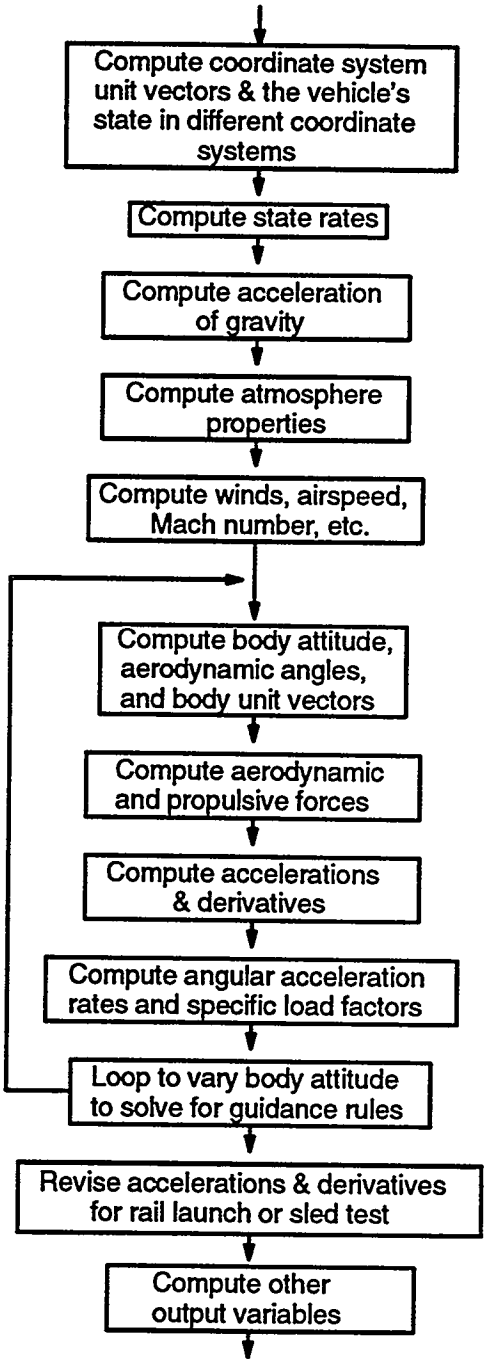

Figure 2–34. Derivative Calculation Flowchart.

Derivative calculations are a function of the time and the vehicle's state. The vehicle's state is maintained in ECFC coordinates so the first step in the derivative calculation is to compute the vehicle's state in geocentric and geodetic coordinates (Sections 2.1.4 and 2.1.6). These calculations must be first because many subsequent calculations require knowledge of the vehicle's state in these two coordinate systems.

Similarly the state rates, that is, the longitude, latitude, and altitude rates, are computed (Section 2.2.3). This is followed by computing the acceleration of gravity vector (Section 2.3.4), the atmospheric properties (Section 2.3.1), and the airspeed, Mach number, dynamic pressure, and Reynold's number (Sections 2.1.8 and 2.3.2).

An initial body attitude is assumed to start the guidance loop. From this body attitude, the aerodynamic angles and body coordinate system unit vectors are calculated (Sections 2.1.7 and 2.1.9). Once the body attitude is known, the aerodynamic and propulsive forces acting on the vehicle can be obtained from the tables (Sections 2.3.2 and 2.3.3), and then the accelerations and derivatives can be computed (Section 2.2.1).

The methods used to solve for the body attitude, given guidance rules, may require the acceleration vector in geodetic coordinates or the specific load factors so these are calculated just before the end of the guidance loop (Sections 2.2.5 and 2.2.6). The guidance loop modifies the body attitude to produce accelerations satisfying the input guidance rules (Section 2.5).

After the guidance loop is finished, the acceleration vector may be modified if a rail launch or sled is involved (Section 2.2.2). Finally the remaining output variables are calculated (Section 2.4).

2.6.2. Search Methods

Two types of one–dimensional search methods are used in TAOS. One type solves for a root of a function, that is, it varies a parameter to drive a function to zero. The other type varies a parameter to minimize a function. These one–dimensional searches can be nested to solve multi–dimensional problems if the functions are well behaved.

Within each type of search, several methods are used depending on the application. For example, a linear search is used to solve for segment final conditions because the search interval is small enough that most functions are nearly linear. Trajectory searches used to meet constraints use a parabolic method because it converges slightly faster in most cases.

Methods that solve for a root of a function vary a parameter x to solve for $f(x) = 0$. Parameter values are restricted to upper and lower bounds, that is, $x_{lo} < x < x_{hi}$. The function $f(x)$ is nonlinear, but it is assumed to be reasonably well–behaved within the interval from x_{lo} to x_{hi}, and it is assumed to have a single root in the interval.

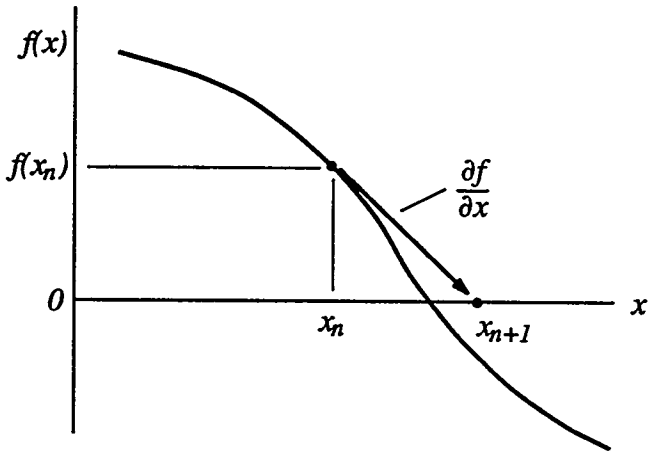

Figure 2–35. Newton–Raphson Search.

Newton–Raphson

The Newton–Raphson method is a simple technique that solves for a root of a function. It starts from an initial estimate of the root x_n and assumes the function is linear, that is,

$$f(x) = f(x_n) + (x - x_n) \cdot \frac{\partial f}{\partial x} \qquad (2\text{–}298)$$

Solving this equation for $f(x_{n+1}) = 0$ gives

$$x_{n+1} = x_n - f(x_n) \cdot \frac{\partial x}{\partial f(x_n)} \qquad (2\text{–}299)$$

Figure 2–35 shows how Equation (2–299) updates the estimate of the root from x_n to x_{n+1} using the slope $\partial f/\partial x$.

Equation (2–299) is used repeatedly to refine the estimate of x until $f(x)$ is within a given tolerance. The derivative $\partial f/\partial x$ is estimated numerically with a forward difference.

Secant Method

The secant method is related to Newton–Raphson because it also assumes the function $f(x)$ is linear. This method begins by evaluating the function at the upper and lower boundaries of the search interval, that is, at x_{lo} and x_{hi}. With the assumption that the function is linear, the value of x for $f(x) = 0$ is

$$x = x_{lo} - f(x_{lo}) \cdot \left[\frac{x_{hi} - x_{lo}}{f(x_{hi}) - f(x_{lo})} \right] \qquad (2\text{–}300)$$

as shown in Figure 2–36.

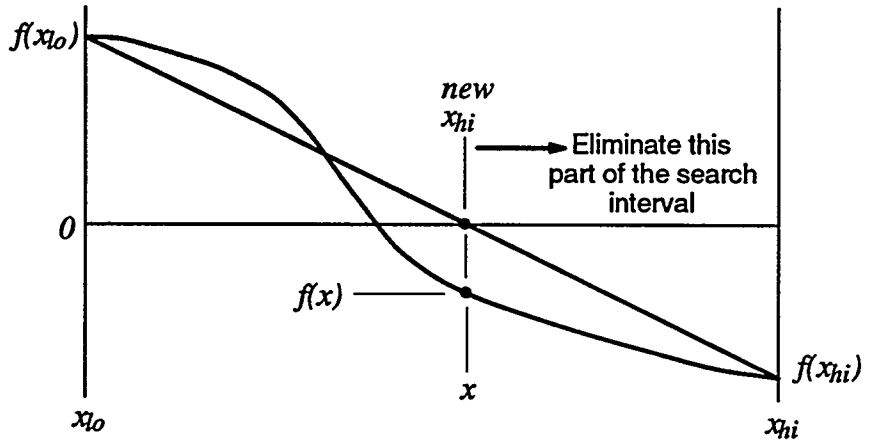

Figure 2–36. Secant Search.

The function is evaluated at this estimate for the root and a new search interval is defined with x as one of the search boundaries. The signs of $f(x_{lo})$ and $f(x_{hi})$ must be different for a root to exist in the interval, so the boundary that has the same sign as $f(x)$ is replaced with x. This procedure is repeated until the estimate of $f(x)$ is within a given tolerance.

Parabolic Root Search

The parabolic method assumes the function is a 2nd–order polynomial of the form

$$f(x) = ax^2 + bx + c \qquad (2\text{–}301)$$

This function has three unknowns so three function evaluations are required to calculate the coefficients a, b, and c. TAOS uses one procedure to obtain these function evaluations during the startup phase and another procedure after the startup phase.

The startup phase makes two function evaluations at $f(x-\Delta x)$ and $f(x+\Delta x)$ where x is an initial estimate and Δx is an estimate of the accuracy of x. For example, an initial

estimate of x might be 2.5 with an estimated accuracy Δx of ± 1.0. These two function evaluations are used in a linear approximation similar to Equation (2–300) to estimate an x value for the third function evaluation as shown in Figure 2–37. All of the x values must be within a search interval given by x_{lo} and x_{hi}.

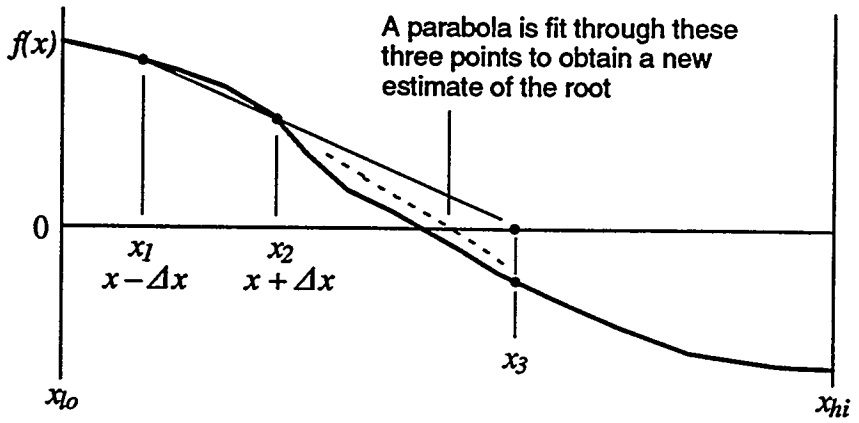

Figure 2–37. Parabolic Root Search.

With three values x_1, x_2, and x_3 and their function evaluations $f(x_1)$, $f(x_2)$, and $f(x_3)$, the coefficients of the 2nd–order polynomial can be computed from

$$b = \frac{(x_2^2 - x_3^2)[f(x_1) - f(x_3)] - (x_1^2 - x_3^2)[f(x_2) - f(x_3)]}{(x_2^2 - x_3^2)(x_1 - x_3) - (x_1^2 - x_3^2)(x_2 - x_3)} \qquad (2\text{–}302)$$

$$a = \frac{f(x_2) - f(x_3) - b(x_2 - x_3)}{x_2^2 - x_3^2} \qquad (2\text{–}303)$$

$$c = f(x_2) - ax_2^2 - bx_2 \qquad (2\text{–}304)$$

This polynomial has two roots at

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad (2\text{–}305)$$

Generally only one of the roots is in the search interval. If both roots are in the search interval the one closest to the last best estimate of the root is chosen. If this cannot be determined the lower value is chosen.

At this point there are four x values. Only three are required to solve for the polynomial on the next iteration, so the x value furthest from the estimate of the root is eliminated. Then the function is evaluated at the estimated root value and the procedure is repeated. This continues until $f(x)$ is less than a given tolerance.

Golden–Section Search

The search methods shown above are all used to find a root of a function. The following searches, such as the golden–section method, solve for the minimum of a function.

The golden–section search is one of the simplest methods requiring no derivatives of the function. It begins by evaluating two points in the search interval chosen such that they divide the interval into golden sections, that is,

$$x_1 = x_{hi} - a(x_{hi} - x_{lo}) \qquad (2\text{–}306)$$

$$x_2 = x_{lo} + a(x_{hi} - x_{lo}) \qquad (2\text{–}307)$$

where a is the golden–section ratio $\frac{\sqrt{5}-1}{2}$.

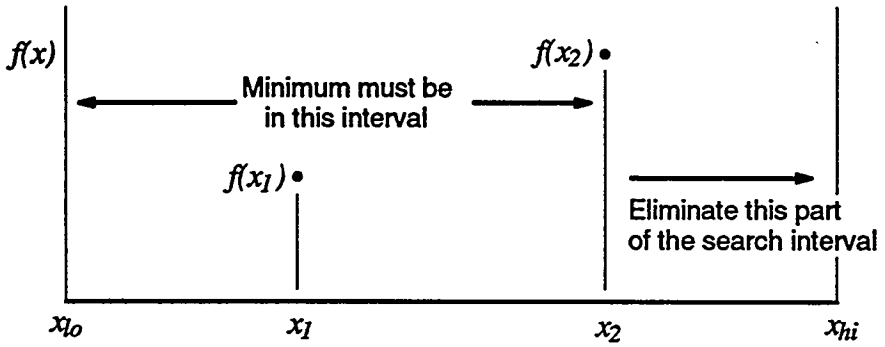

Figure 2–38. Golden Section Search.

Based on the function evaluations $f(x_1)$ and $f(x_2)$, part of the search interval is eliminated. If $f(x_2)$ is greater than $f(x_1)$, then the new upper search limit becomes $x_{hi} = x_2$ and $x_2 = x_1$ as shown in Figure 2–38. A new x_1 value is estimated from Equation (2–307) and the procedure is repeated. If $f(x_2)$ is less than $f(x_1)$ a similar procedure is used except that x_{lo} is set to x_1 and $x_1 = x_2$. This process continues until the value $x_{hi} - x_{lo}$ is less than a tolerance.

Parabolic Minimization

The parabolic minimization search is nearly identical to the parabolic root search. The only difference is that the minimum of the parabola is used instead of its roots. This method has the same startup phase as the parabolic root method resulting in three function evaluations. A parabola is fit through these points with Equations (2–302) through (2–304) and then the minimum is computed with

$$x = -\frac{b}{2a} \qquad (2\text{–}308)$$

Tests are made to determine if this is a minimum or maximum. If it is a minimum, then the search interval is redefined keeping the minimum point in the middle if possible. If this is not a minimum, then the search interval is redefined to keep the three lowest function evaluations.

After a new search interval has been determined, the function is evaluated at the new point, a parabola is fit through the points, and the procedure is repeated. It converges when the predicted minimum value $ax^2 + bx + c$ is within a given tolerance of the actual function evaluation $f(x)$.

2.6.3. Optimization

The general nonlinear programming problem varies a set of n variables or parameters x such that they minimize an objection function $f(x)$ subject to a set of m equality and inequality constraints $c_j(x)$, that is,

$$\begin{aligned} &\text{Minimize} \quad f(x_i) \qquad \text{for } i = 1,2,\ldots, n \\ &\text{subject to} \\ &\quad c_j(x_i) = 0 \qquad \text{for } j = 1,2,\ldots, m_e \\ &\quad c_j(x_i) \geq 0 \qquad \text{for } j = m_e + 1,\ldots, m \end{aligned} \tag{2–309}$$

Typically the equality constraints are grouped together and given first, followed by the inequality constraints.

Han–Powell Method (vf02)

TAOS uses the Han–Powell method, originally developed by Powell as the code *vf02ad*, to solve the nonlinear programming problem.[3,22] The Han–Powell method belongs to the class of methods called recursive quadratic programming (RQP). It solves the nonlinear programming problem iteratively as a sequence of quadratic programming problems using an active constraint approach. The active constraints are all of the equality constraints plus selected inequality constraints that are currently being treated as equality constraints.

Initial estimates for the optimization parameters x are required to start the procedure. The parameters are modified at each iteration to reduce the value of the objective function and to satisfy the constraints.

With the active constraint approach, the problem reduces to an equality problem

$$\begin{aligned} &\text{Minimize} \quad f(x) \\ &\text{subject to} \quad c(x) = 0 \end{aligned} \tag{2–310}$$

at each iteration. This problem is solved in two steps: the first step is to determine a search direction, and the second step is to search in this direction for a minimum.

The search direction δx is found by approximating the objective function $f(x)$ with a quadratic function $Q(\delta x)$, that is,

$$f(x + \delta x) - f(x) \cong Q(\delta x) = f_x(x)\delta x + \frac{1}{2!}\delta x^T f_{xx}(x)\delta x \tag{2–311}$$

and approximating the constraints $c(x)$ with linear functions, that is,

$$c(x + \delta x) \cong c(x) + c_x(x)\delta x \tag{2–312}$$

where $f_x(x)$ and $c_x(x)$ are the gradients of $f(x)$ and $c(x)$. The Hessian matrix $f_{xx}(x)$ can be replaced by a positive definite matrix B called the variable metric matrix resulting in

$$\text{Minimize} \quad Q(\delta x) = f_x(x)\delta x + \frac{1}{2}\delta x^T B \delta x \qquad (2\text{–}313)$$

$$\text{subject to} \quad c_x(x)\delta x + c(x) = 0$$

Details of the solution to this problem using active constraints are given in References 3 and 22. The result is a search direction δx that is used in a one–dimensional search to find a new estimate for x. The one–dimensional search uses a penalty function to ensure that it stays within the constraints.

TAOS uses a version of *vf02ad* obtained from D.G. Hull at The University of Texas at Austin. This code was converted to the C programming language by the author.

Optimization Parameters

The optimization parameters x are defined by the user in the problem file with special keywords (Section 4.4.6). Any input numerical value can be used as a parameter. The most common optimization parameters are trajectory initial conditions, guidance rules, and segment final conditions. The optimization process works best when there are only a few parameters, so the goal is to minimize the number of parameters while maintaining the fidelity of the simulation.

The shape of a trajectory is controlled by control variables (angle of attack, sideslip, bank angle, and power setting) whose values are determined by the guidance rules (Section 2.5). If the guidance rules are defined as optimization parameters, then optimization can be used to shape the trajectory to achieve a goal such as maximum range while satisfying a set of constraints.

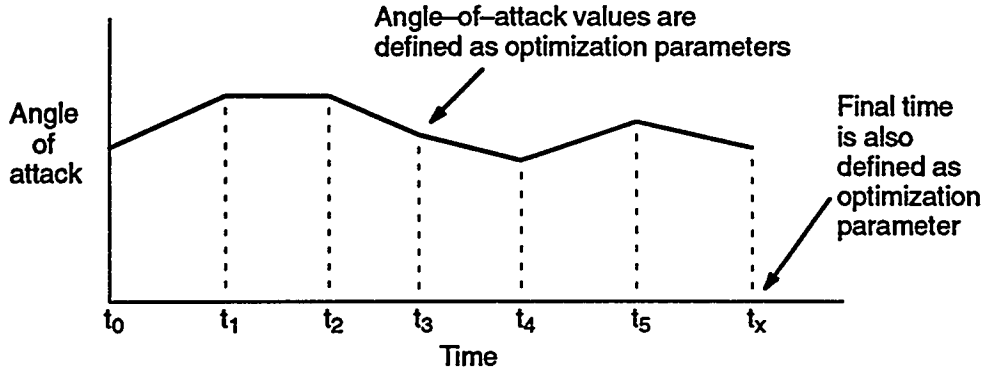

Figure 2–39. Optimization Used for Trajectory Shaping.

A common way to set up a trajectory shaping optimization problem is to enter a guidance rule table giving a control variable as a function of time. For example, a guidance rule can give angle of attack as a tabulated function of time as shown in Figure 2–39. This angle–of–attack time history is interpolated at each integration step to obtain the control variable value. The final time and the tabulated angles of attack are defined as the optimization parameters. The other time values are estimated constant values.

Preliminary trajectories must be calculated to get estimates for the final time and angles of attack that are reasonable. The optimization procedure is more sensitive to

the initial estimates for the parameters when an angle–of–attack time history is used. The starting trajectory must be reasonable.

Optimization varies these parameters, the final time and angles of attack, to achieve the objective function and constraints. This procedure works best when the angle–of–attack points in the time history are evenly distributed. Since the last time value is an optimization parameter that changes, it can get very large relative to the other values or it can get very close to the next to last value. To help solve this problem, TAOS can periodically restart the optimization procedure and automatically adjust the time history values such that they are evenly distributed.

Objective Functions and Constraints

In addition to the optimization parameters, the user also defines the objective function $f(x)$ and the constraints $c(x)$ (Section 4.4.6). The most common objective functions are time, range, and velocity. Although optimization problems are classically stated to minimize an objective function, the same procedure can be used to maximize a function because the maximum of $f(x)$ is the minimum of $-f(x)$. Thus, optimization can be used to determine maximum range and velocity.

Inequality constraints limiting the optimization parameters to reasonable values are recommended. Additional equality and inequality constraints can be applied to any trajectory output variable at the end of any segment. For problems with multiple trajectories, constraints can be defined using variables from all of the trajectories. Constraints are commonly used to solve for trajectory final conditions and to force events to occur at specific points in the trajectory.

Constraints affecting the entire trajectory, such as a maximum altitude or a minimum dynamic pressure, must be handled with special user–defined integral variables (Section 4.3.1). For example, the variable h_{max} can be defined as

$$h_{\max} = \int (h - 100000)^2 \quad \text{for } h > 100000 \qquad (2\text{–}314)$$

$$h_{\max} = 0 \quad \text{otherwise}$$

so that h_{max} is positive when a maximum altitude of 100,000 ft is exceeded and zero otherwise. This variable can be used in the following optimization constraint:

$$\text{Constrain} \quad h_{\max} < 0 \qquad (2\text{–}315)$$

The optimization procedure can have numerical problems if the objective function or constraints have values extending across several orders of magnitude. To avoid this problem, the objective function and constraints can be normalized with reference factors such that their values are in the range from 1 to 10. The reference factors are either automatically determined or input by the user.

Gradients

The optimization algorithm requires the gradients of the objective function $f(x)$ and the constraints $c(x)$, that is, $f_x(x)$ and $c_x(x)$. These are calculated numerically with for-

ward or central differences. For a given δx, central differences gives more accurate derivatives than forward differences, but at the expense of an additional function evaluation. The accuracy of forward differences can be improved by using a smaller δx value. Gradients computed with forward differences work well in practice for most problems and avoid extra function evaluations, but both gradient methods are provided in TAOS.

Intentionally Left Blank

3. Table Files

Before a flight trajectory can be computed, all the forces acting on the vehicle must be known. On an airplane or rocket, the primary forces are lift, drag, thrust, and weight as shown in Figure 3–1. Lift and drag forces are aligned with the wind axis system, thrust forces are aligned relative to the body axis system, and weight forces are aligned with the gravity vector.

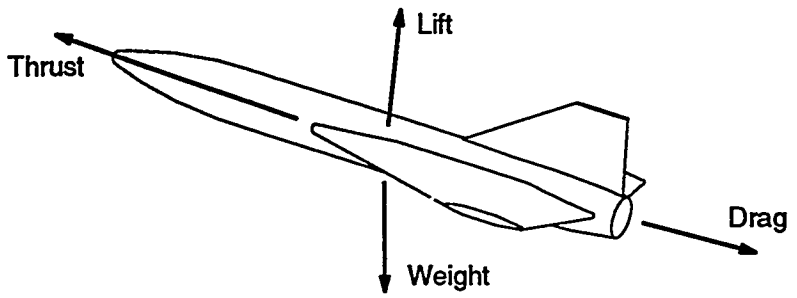

Figure 3–1. Forces Acting on Vehicle.

Lift and drag forces are normally given in terms of nondimensional lift and drag coefficients and a reference area. These aerodynamic coefficients are difficult to predict; they are often computed with aerodynamic prediction software, such as Missile Datcom[18] and Sandiac[19], or they are obtained from wind tunnel tests. The result is a tabulated set of aerodynamic coefficients that is often a function of Mach number, altitude, and angle of attack. For example, Figure 3–2 shows the axial force coefficient as a function of Mach number.

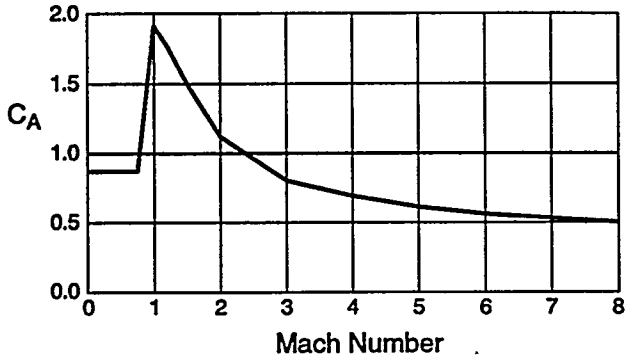

Figure 3–2. Axial Force Coefficient as Function of Mach Number.

Thrust and mass flow data for jet and rocket motors are usually obtained from analysis software or from tests. Again this data is normally tabulated because no analytical functions exist. For jet engines thrust and fuel flow are typically a function of Mach number, altitude, and power setting. For rocket motors thrust and mass flow are usually given as functions of time as shown in Figure 3–3.

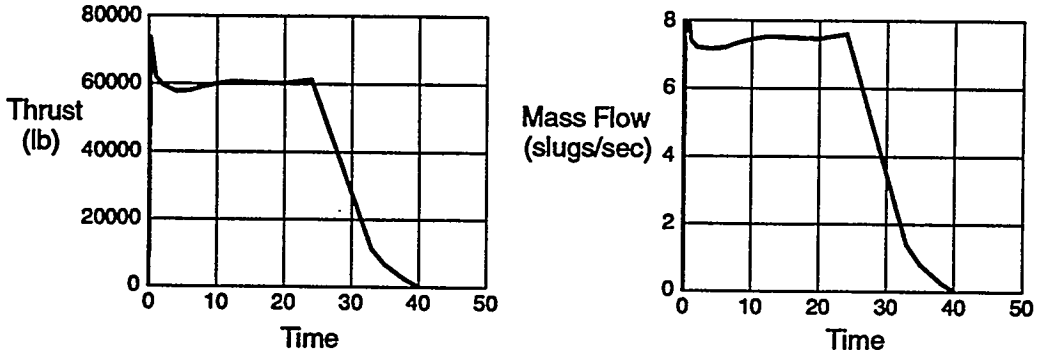

Figure 3–3. Thrust and Mass Flow as a Function of Time.

Tabulated data, defining forces on the vehicle, are input to TAOS with table files. Each table file consists of one or more tables with each table identified by a name as shown in Figure 3–4. A table defines a single value, such as thrust or mass flow. This value can be a constant, it can be interpolated from tabulated data, it can be computed from an equation, or it can be computed from a combination of interpolations and equations.

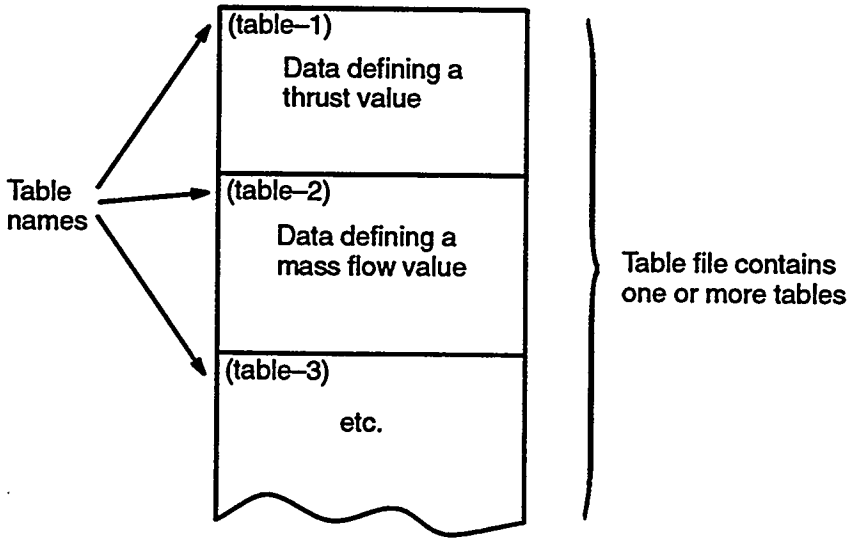

Figure 3–4. Table Files and Tables.

3.1. Table Types

Each table in a table file has a table type corresponding to the value that is defined in the table. For example, the *thrust* table type defines a thrust value. The table type is compared to its use in the problem file, and if it is not used correctly (for example, a thrust table used for axial force coefficient), an input error occurs. Table 3–1 contains a list of the table types used in TAOS.

Table 3–1. Table Types

Table Type	Description
ca	Axial force coefficient (in body coordinates)
cn	Normal force coefficient (in body coordinates)
cl	Lift force coefficient (in wind coordinates)
cd	Drag force coefficient (in wind coordinates)
cs	Side force coefficient (in wind coordinates)
cx	Body x–axis force coefficient
cy	Body y–axis force coefficient
cz	Body z–axis force coefficient
thrust	Thrust magnitude
tvec1	Thrust vector angle 1
tvec2	Thrust vector angle 2
mdot	Mass flow
cg	Center of gravity
windv	Wind magnitude
windh	Wind direction or heading
winde	Wind component in east direction
windn	Wind component in north direction
windd	Wind component in "down" direction
output	User–defined output variable

Aerodynamic force coefficients can be defined one of three ways: in the vehicle body axis with C_X, C_Y, and C_Z, in the wind axis with C_L, C_D, and C_S, or in an axisymmetric system with C_A and C_N. These sets of coefficients go together; they cannot be mixed. For example, C_L, C_D, and C_S form a valid set of aerodynamic coefficients; C_L, C_D, and C_A are not a valid set of aerodynamic data.

Thrust and mass flow tables are used for all types of jet and rocket motors. The thrust table defines the magnitude of the thrust vector; the direction of the thrust vector relative to the body coordinate system is given by the thrust vector angle tables.

Point–mass trajectory analysis methods assume a vehicle is always in a trimmed condition, that is, the moments always sum to zero. Therefore, trimmed aerodynamic coefficients should be input rather than untrimmed values. Trimmed aerodynamic coefficients are often a function of the vehicle's center of gravity, and center of gravity is in turn often a function of the vehicle's weight. The *cg* table type is provided to define a value for center of gravity that can be used later in the aerodynamic coefficient tables.

Wind tables define the components of the wind velocity vector in a local horizon coordinate system. The wind horizontal velocity is given by either the *windv* (velocity magnitude) and *windh* (wind heading) tables or by the *winde* (east component) and *windn* (north component) tables. The *windv* and *windh* tables go together, and the *winde* and *windn* tables go together. A vertical wind component can also be input with the *windd* table.

Output tables are used to calculate user–defined output variables such as heat transfer rates. Values from the output tables are not used during the trajectory calculations, but they are available when computing user–defined output variables (Sections 4.3.1 and 4.4.2).

3.2. Table/Problem File Relationship

Information in the problem file tells TAOS which tables to use for each part of the trajectory. Trajectories are divided into segments, and different tables can be used in each segment. For example, the line

```
*aero  ca=(stage1_ca_on)  cn=(stage1_cn)
```

in the problem file tells TAOS to use a table called *stage1_ca_on* for the axial force coefficient and a table called *stage1_cn* for the normal force coefficient (Section 4.2.1). These table names must be in the input table files. When TAOS needs a value for the axial force coefficient, it will retrieve it from the table *stage1_ca_on*.

Table names are given in **aero*, **prop*, **cg*, and **wind* data blocks in the problem file. They can also be used in **define* data blocks when defining user–defined output variables. All tables referenced in a problem file must be in the table files.

More than one table of each type can be given in a trajectory segment; the values from each table are added together. For example, if two thrust tables are given, they are evaluated separately and then added together to get the total thrust force. Multiple tables can also be used for a component buildup of the total vehicle lift and drag forces.

3.3. Table File Format

Table files are text files that can be created with a text editor or word processor. On UNIX the standard *vi* editor can be used, or on a Silicon Graphics workstation, the *jot* editor can be used. Table files can also be created with other software, such as Aero[20] and Rocket.[21]

Values

Table data is read line by line and is not column dependent. Information must be given in the correct order so TAOS knows how to interpret it. Lines can be any number of characters in length and blank lines are ignored. Names and values in the file can have up to 72 characters so they are essentially unlimited in length.

Names and values are separated with white–space characters (such as space, tab, or carriage–return), commas, equal signs, parentheses, colons, greater than signs, and less than signs. Because these special characters are used as delimiters to separate names and values, they cannot be imbedded within a name or value. For example, the value 30,000 is not correct because the comma is a delimiter that separates values; 30,000 is interpreted as two values: 30 and 0. The correct way to input this value is 30000.

Input values are not column dependent (they are free–field), so indentation can be used to make the table file easier to understand. Many examples are given in this report using indentation to make it easier to see how the values relate to each other. This style is encouraged because it reduces input errors.

Values can be input with or without a decimal point. All values are interpreted as real numbers; no distinction is made for integers. Values can also be input in scientific notation, that is, e–format. For example, the number 2000 can be input as "2000," "2000.0" or "2.0e3."

TAOS converts all characters to lower case before interpreting them as names or values so upper and lower case letters are equivalent. Use of lower–case characters is recommended for improved readability.

Comments

Comments start with the special character # and continue to the end of the line. They can be placed anywhere in the file except within a name or value. The following lines show examples of valid comments:

```
#  the entire line is used for comments
   0.2   0.3   0.4   0.5      # Mach numbers
```

In the first line, the special # symbol is in the first column. Everything after the # symbol is treated as a comment and ignored by TAOS, so the entire line becomes a comment. In the second example, the comment does not begin until after the four values. From the # symbol to the end of the line is treated as a comment.

State Variables

As TAOS computes the trajectory of a vehicle, it keeps track of many values. All of these values together define the vehicle's state as it moves along the trajectory, so they are called state variables. As mentioned previously, a table defines a single value, that is, it contains enough information so that TAOS can compute its value. Usually the value is a function of one or more of the state variables; for example, thrust is often a function of time. At each instant along the trajectory, the value of time is known; therefore, the value of thrust is known. Table 3–2 contains a list of the state variables that can be referenced in TAOS tables.

Table 3–2. TAOS State Variables.

Variable	Description	Variable	Description
alpha	Angle of attack	pitchgd	Geodetic pitch angle
alphat	Total angle of attack	pitchi	Inertial pitch angle
alt	Geodetic altitude†	power	Power setting
altdt	Geodetic altitude rate†	pres	Atmospheric pressure†
bankgc	Geocentric bank angle	psigc	Geocentric heading angle†
bankgd	Geodetic bank angle	psigd	Geodetic heading angle†
beta	Sideslip angle	range	Range
betae	Euler sideslip angle	rcm	Distance from earth center†
cg	Center of gravity	rcmdt	Rate from earth center†
dynprs	Dynamic pressure†	reypft	Reynold's number per foot
ep1	Thrust vector angle 1	rho	Atmospheric density†
ep2	Thrust vector angle 2	rollgc	Geocentric roll angle
gamgc	Geocentric flight path angle†	rollgd	Geodetic roll angle
gamgd	Geodetic flight path angle†	rolli	Inertial roll angle
grmark	Ground range since last reset	segment	Segment number
grseg	Segment ground range	sndspd	Atmospheric speed of sound†
latgc	Geocentric latitude†	sref	Aerodynamic reference area
latgcdt	Geocentric latitude rate	temp	Atmospheric temperature†
latgd	Geodetic latitude†	thrust	Total thrust
latgddt	Geodetic latitude rate	time	Time†
long	Longitude†	tmark	Time since last reset†
longdt	Longitude rate	tseg	Segment time†
mach	Mach number	vair	Velocity relative to air
mass	Vehicle mass†	vel	Inertial velocity†
nu	Atmospheric kinematic viscosity†	vgr	Velocity relative to ground
plength	Path length	visc	Atmospheric viscosity†
plmark	Path length since last reset	wt	Vehicle weight†

plseg	Segment path length	yawgc	Geocentric yaw angle
phi	Windward meridian	yawgd	Geodetic yaw angle
pitchgc	Geocentric pitch angle	yawi	Inertial yaw angle

Wind and center of gravity tables can only be functions of a limited number of the state variables which are designated with the † symbol in Table 3–2. These tables are evaluated before all of the state variables are known. Tables also cannot be functions of themselves; for example, a thrust table cannot be a function of thrust.

The default units for the state variables are given in the appendix. These can be changed in the problem file with the **units/fmt* data block (Section 4.4.13).

User–Defined Variables

There are times when the state variables alone do not provide enough flexibility; for example, there may be three thrust versus time curves for three different versions of a rocket motor. These curves can be placed in separate tables, or they can be placed in the same table. If they are put in the same table, thrust becomes a function of both time and another variable that can be called *version*. This new variable is called a user–defined variable because any name can be used except those reserved for the state variables in Table 3–2. When a user–defined variable is detected, it must be given a value in the problem file. For example, the line

```
*prop  thrust=(stage1)  version=2
```

says to use a table called *stage1* for thrust and to set the user–defined variable *version* equal to two. The value of *version* is constant within a segment of a trajectory, but it can be surveyed like any other numerical input value (Section 4.4.11).

3.4. Simple Tables

Two table formats are available in TAOS. The first is called the simple format because it is easy to create, but it has limited capability. The second is called the full format because it provides the full capabilities of the software, but it is somewhat more difficult to create.

Simple tables consist of a single set of tabulated numbers. The set of tabulated numbers defines a function of one or more state variables (or independent variables). The simplest table is a function of one variable; for example, a table of thrust values given as a function of time is shown below.

```
(motor_1)
   table  thrust(time)
      time   = 0.0,  0.5,  2.0,   4.0,   4.5,  5.0
      thrust = 0.0, 3500., 3100., 2950., 600., 0.0
```

The table begins with a name in parentheses followed by the keyword *table*. This is followed by the table type, *thrust*, and a list of independent variables, in this case the variable *time*. Tabulated values for the independent variable, *time*, and the dependent variable, *thrust*, are given next. For each value of *time*, a corresponding *thrust* value is given.

Table Identification Name

Simple tables always begin with a name enclosed in parentheses. This name is called the table identification name and is used in the problem file to reference the table.

The name must be unique among all input table files. The name cannot have spaces or other delimiters in it; however, the use of nondelimiter special characters, such as underscores, dots, or dashes, is encouraged to make the name readable.

Table Function Definition

The table identification name is followed by the keyword *table* and the table type (from Table 3–1). A list of up to five state variables (or independent variables) for the table is given next. These variables must be separated with commas and enclosed in parentheses, so it reads like a function definition. For example,

```
table  ca(mach,alt,alpha)
```

says that C_A is a function of Mach number, altitude, and angle of attack.

Extrapolation Option

The function definition is followed by an extrapolation keyword, either *extrap* or *no_extrap*. This keyword is optional and controls extrapolation of the table values when the independent values are not within the range of the tabulated values as shown in Figure 3–5.

All interpolation and extrapolation is linear, so this should be taken into account when entering values and setting the extrapolation option. The default is to allow extrapolation.

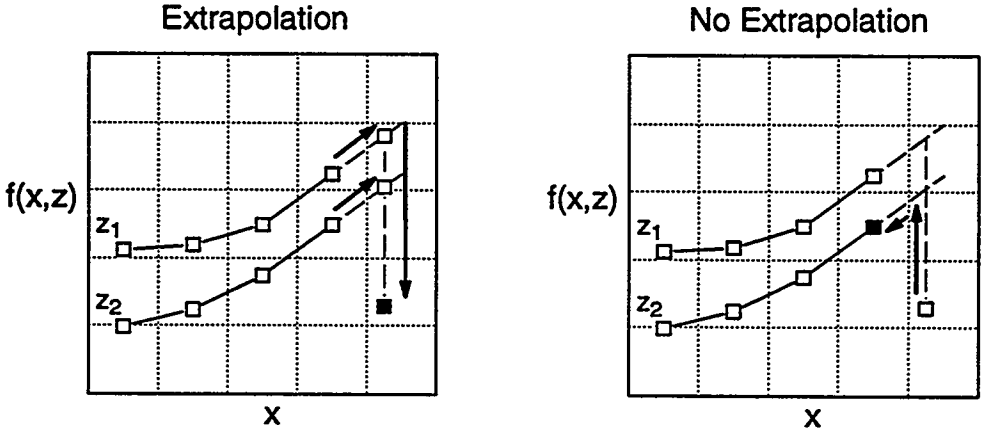

Figure 3–5. Table Extrapolation Option.

Table Parameters

The extrapolation option may be followed by a variable name and value called the table parameter. This parameter provides some additional information associated with the table.

For aerodynamic coefficient tables, the parameter is the reference area, *sref*. For thrust and mass flow tables, a parameter called *units* gives the units used in the table. Allowable units for the thrust and mass flow tables are given in Table 3–3. Other table types do not use a table parameter.

Table 3–3. Units for Thrust and Mass Flow Tables

Thrust Units	Mass Flow Units	Mass Flow Units	Mass Flow Units
lb	lb/sec	lb/min	lb/hr
n (Newtons)	slugs/sec	slugs/min	slugs/hr
kn (Kilonewtons)	g/sec	g/min	g/hr
	kg/sec	kg/min	kg/hr

Examples

An example of a complete table definition line for an axial force coefficient table is

```
table  ca(mach,alpha)  extrap  sref=2.162
```

which defines a table that is a function of Mach number and angle of attack, allows extrapolation, and uses a reference area of 2.162 ft^2. (The units of *sref* default to ft^2,

but they can be changed with the **units/fmt* data block in the problem file as shown in Section 4.4.13). Another example is

```
table  mdot(time,motor)  no_extrap  units=lb/hr
```

which defines a table for mass flow rate that is a function of *time* and a user–defined variable called *motor*, does not allow extrapolation, and has units of lb_m/hr.

Independent Variable Values

The table definition information is designed to be put on one line. This line is followed by values for the independent variables. The variable name is given first, followed by a list of values; for example,

```
mach = 0.2, 0.4, 0.6, 0.8, 0.9
alpha = 0, 2, 4
```

Any number of values can be input. The number of values is limited only by the amount of memory available in the computer. Values for independent variables must be strictly increasing or decreasing, and duplicate values are not allowed.

Values can be entered with or without decimal points; they are all treated as real numbers. Values can also be entered in scientific notation (e–format) as follows:

```
time = 0., 1.0e1, 2.0e1, 6.0e1, 7.0e1, 9.0e1,
       2e02, 4e02, 6e02
```

Spaces or commas are usually used to separate values because they provide the most distinctive separation for the reader. Values do not have to be all on one line; they can be continued as shown above.

When a table is a function of more than one independent variable, values for these variables need to be given in the same order as listed in the function definition. For example,

```
table  cx(alt,mach)  extrap  sref=5.30      # CORRECT:
   alt = 0, 10000                           # Altitudes first
   mach = 5, 10, 15, 20                     # then Mach numbers
```

is correct because altitude is given before Mach number in both the function definition and in the list of values. An error is generated if the Mach number values are given first:

```
table  cx(alt,mach)  extrap  sref=5.30  # INCORRECT: Altitude
   mach = 5, 10, 15, 20                 # is first in definition
   alt = 0, 10000                       # but Mach values given
                                        # first
```

Dependent Variable Values

After all independent values have been given, the dependent values are listed. For tables with a single independent variable, there is a one to one correspondence be-

tween the independent and dependent variable values. The following complete thrust table illustrates this:

```
(example_thrust)
   table  thrust(time)  units=lb  no_extrap
      time = 0.0, 0.1,  1.0,  2.0,  3.0,  3.4, 3.5
      thrust = 0, 5100, 4950, 4900, 4840, 4700, 0
```

At time = 2.0 seconds, the thrust is 4900 lb_f. The same number of values must be given for both time and thrust.

For tables with more than one independent variable, one must be careful that the dependent values are given in the correct order. The following example shows the correct order:

```
(example_cx)
   table  cx(alt,mach)  extrap  sref=5.30
      alt = 0, 10000
      mach = 5, 10, 15, 20
      cx = 0.030, 0.027, 0.025, 0.024   # Values for alt=0
           0.035, 0.031, 0.029, 0.028   # Values for alt=10000
```

Values for each Mach number for the first altitude are given first, followed by values for the second altitude. There must be a dependent value for each combination of independent values. In this example there are two altitudes and four Mach numbers; multiplying these together means there must be eight C_X values.

The following table is a function of three independent variables:

```
(example_ca)
 table  ca(alt,mach,alpha)  no_extrap  sref=10.0
  alt = 0, 5000
  mach = 4, 6, 8
  alpha = 0, 2, 4, 6
  ca = 0.020, 0.021, 0.022, 0.024  # Values for mach=4, alt=0
       0.018, 0.019, 0.020, 0.022  # Values for mach=6, alt=0
       0.017, 0.018, 0.019, 0.021  # Values for mach=8, alt=0

       0.021, 0.022, 0.023, 0.025  # Values for mach=4, alt=5000
       0.020, 0.021, 0.022, 0.024  # Values for mach=6, alt=5000
       0.019, 0.020, 0.021, 0.023  # Values for mach=8, alt=5000
```

There are two altitudes, three Mach numbers, and four angles of attack, so there must be $2 \cdot 3 \cdot 4 = 24$ values for C_A. Values for altitude = 0 are given first. Within this set of values, values for Mach = 4 are given first, followed by values for Mach = 6, and so on. This pattern continues for all values of Mach and altitude. It can be extended to tables that are functions of four or five independent variables.

3.5. Full Tables

Full tables provide more ways to define table values than simple tables. In a simple table, the table value is obtained by interpolating a table. Values of the independent variables (the state variables) are known from the flight path integration, and the tabulated data values are known from the input. Linear interpolation is used to compute the table value or dependent variable. The only flexibility is in the selection of the independent variables and their values.

In a full table, the table value is obtained by executing a sequence of math operations. When TAOS needs to compute a table value, it is first initialized to zero. Then a sequence of math operations, provided in the table input, is executed one at a time. Each math operation adds, subtracts, multiplies, etc. the current table value with another value. The other value can be a constant, a value interpolated from a table, a state variable, or a user–defined variable. It is analogous to using a calculator with an accumulation register to compute the table value.

The sequence of math operations is much like a simple programming language. The math operations let the user control how a table value is computed. Temporary values can be stored for later use just like a calculator, and simple *if–then* and *goto* statements give some flow control over the calculation.

The following example illustrates this concept:

```
(1st_stage)
   table   thrust  units=lb
      start
      sub pres              # subtract atmospheric pressure
      mult  3.219           # multiply by nozzle exit area
      add  tvac(time)       # add the vacuum thrust
         time = 0, 0.1,  0.2,  0.5, 1.0, 4.0, 4.1, 4.2, 4.3
         tvac = 0, 1200, 1050, 980, 950, 910, 100,  50,  0
      end
```

This table defines thrust magnitude in lb_f as the vacuum thrust minus the atmospheric pressure times the nozzle exit area, that is,

$$\| \vec{F}_{prop} \| = \| \vec{F}_{vac} \| - p \cdot S_{noz} \qquad (3\text{–}1)$$

The calculation begins at the *start* statement by initializing thrust to zero. Then the atmospheric pressure is subtracted from zero giving a negative value with the *sub pres* statement. After this statement is executed the table value is equal to negative pressure. The next statement, *mult 3.219*, multiplies the current table value by a constant representing the nozzle exit area. After this statement executes, the table value is equal to a negative pressure times the nozzle exit area. Finally the statement *add tvac(time)* interpolates the table values for the vacuum thrust as a function of time and adds this value to the table value. The resulting thrust value is used in the flight path calculations.

Like any programming language there are many ways that this same calculation can be done. Another method for the same calculation is to multiply the pressure and exit

area together, save this in a temporary variable, clear the table value, interpolate the vacuum thrust, and finally subtract the saved pressure times area value. This method takes more math operations and is less efficient.

Tables are evaluated many times while calculating a flight path so it is worth some effort to make them efficient. Reducing the number of math operations in a table can significantly reduce the processing time.

3.5.1. Math Operations

Math operations available in TAOS are given in Table 3–4. The sequence of math operations always begins with the *start* keyword and ends with the *end* math operation. The last math operation in a table must be the *end* operation.

Table 3–4. Math Operations.

Operation	Description	Operation	Description
add	Add value to table value	ln	Natural log of table value
sub	Subtract value from table value	log	Log base 10 of table value
mult	Multiply value by table value	e	e raised to the table value power
div	Divide table value by value	sin	sin of table value (in degrees)
idiv	Divide value by table value	cos	cos of table value (in degrees)
exp	Raise table value to power of value	tan	tan of table value (in degrees)
iexp	Raise value to power of table value	asin	$\sin^{-1}$ of table value (in degrees)
max	Limit table value to less than a value	acos	$\cos^{-1}$ of table value (in degrees)
min	Limit table value to greater than a value	atan	$\tan^{-1}$ of table value (in degrees)
set	Set table value to a value	zero	Set table value to zero
abs	Absolute value of table value	csto	Store table value and set it to zero
neg	Negate table value	if	If–then statement
sqr	Square table value	goto	Goto statement
sqrt	Square root of table value	end	End of table

Math operations that only operate on the current table value do not require additional information, such as a constant or table. For example,

```
start
  add alpha
  sin
  sqr
  add 1.0
end
```

computes $(1 + sin^2\alpha)$. The math operations *sin* and *sqr* do not require additional values; they operate only on the current table value. The math operation *add* does require another value. Angle of attack is added in the first *add* statement, and the constant 1.0 is added in the second *add* statement.

In Table 3–4, operations *add* through *set* require an additional value; operations *abs* through *zero* do not require an additional value. The *csto*, *if*, and *goto* operations are special operations discussed later in this document. The additional value required by operations *add* through *set* can be a constant, a state variable, a value interpolated from a table, a temporary storage variable, or a user–defined variable.

3.5.2. Constant Values and State Variable Values

Constant values are always treated as real numbers. They can be entered with or without decimal points and scientific notation using e–format is acceptable. Some examples of math operations with constants are

```
add  2.0
mult  1.5e-6
exp  3
```

which computes the value $((x + 2.0) \cdot 1.5 \times 10^{-6})^3$ where x is the current table value.

Values of the state variables change as the flight path is integrated. When table values are computed during path integration, current values of the state variables can be retrieved by using the variable names from Table 3–2. For example, dynamic pressure (defined as $0.7pM^2$) can be computed with

```
add  0.7
mult  pres
mult  mach
mult  mach
```

The default units for these variables are given in the appendix, but these can be changed in the problem file with the **units/fmt* data block (Section 4.4.13).

3.5.3. Tabulated Data Values

Values for math operations can be linearly interpolated from a table that is a function of up to five independent variables. The tables are given in the same format as shown previously for simple tables. The table dependent variable name is given first, followed by a list of independent variables enclosed in parentheses. A list of values is given for each independent variable defining a matrix. Then values for the dependent variable are given for each combination of independent values.

The simplest table is a function of one variable. For example,

```
add  cdh(alt)
  alt =  0,  50000, 100000, 150000, 200000
  cdh = 0.0, 0.002,  0.006,  0.011,  0.042
```

defines a value called *cdh* that is a function of altitude. The value interpolated from the table for *cdh* is added (because of the *add* operation) to the current table value. The name *cdh* must be unique, that is, it cannot be a state variable or a user–defined variable. It is only an identification name; it is not available as a storage variable so it cannot be referenced by subsequent math operations.

In the next example, the table value is multiplied by a factor obtained from a tabulated function of Mach number and a user–defined variable called *type*.

```
mult  factor(type,mach)
  type = 0, 1
  mach = 0.0, 0.999, 1.000, 25.0
  factor = 0.5, 0.5, 1.0, 1.0,       # Values for type = 0
           1.0, 1.0, 2.0, 2.0        # Values for type = 1
```

The value of *type* is set in the problem file, for example, in the **aero* or **prop* data block. If *type* is set to zero, then the table value is multiplied by 0.5 for subsonic Mach numbers and 1.0 for supersonic Mach numbers. If *type* is set to one, these multiplicative factors are doubled.

Tabulated functions of more than two variables are input using the same pattern as shown previously for simple tables (Section 3.4). Another method, called skewed tabulated data, is available as shown in Section 3.5.8

3.5.4. Storage Variables

The math operation *csto* (clear and store) is used to save temporary calculations for later use, just like a storage register on a calculator. After a value is saved and given a name, it can be retrieved and used in subsequent math operations. For example, the operations

```
add  alpha
cos
sqr
csto  ca2
```

calculate $cos^2\alpha$ and save it as the variable *ca2*. The *csto* operation also sets the current table value to zero in preparation for subsequent math operations.

The storage variable name given with the *csto* operation must be unique; in other words, it cannot be a state variable from Table 3–2 or a user–defined variable used elsewhere in the table or problem files.

The variable *ca2* with its value of $cos^2\alpha$ can be used in subsequent calculations, either as a value for a math operation or as an independent variable of a table. For example,

```
add  alpha
cos
sqr
csto  ca2
add  factor(ca2,mach)
```

uses the variable *ca2* as an independent variable in a table. This provides the capability to enter tables that are complex functions of the state variables. In this case *factor* is a function of $cos^2\alpha$ and Mach number.

3.5.5. User–defined Variables

If a variable name is used in a math operation and it is not a state variable or a storage variable, then TAOS assumes that it is a user–defined variable. TAOS expects its value to be given in the problem file in the **aero*, **constants*, **cg*, **prop*, or **wind* data blocks (Sections 4.2.1, 4.2.2, 4.2.3, 4.2.9, and 4.4.14). These are the same places that the table identification names are given telling TAOS which tables to use.

User–defined variables can be used in math operations just like state variables. They are just constants that have values set in the problem file. The advantage of using user–defined variables and setting the values in the problem file versus using constants in the table file is that the user–defined values can be automatically surveyed in the problem file (Section 4.4.11). Thus, if the table constant takes on different values, then it is wise to make it a user–defined variable. For example, the table

```
(constant_thr)
  table  thrust  units=lb
  start
     sub  pres
     mult anoz
     add  tvac
  end
```

uses two user–defined variables: *anoz* and *tvac*. Because this is a thrust table, values for *anoz* and *tvac* are given in the **prop* data block in the problem file. This table is generic in the sense that it can be used to represent any constant thrust rocket motor.

User–defined variables are a powerful tool, but they must be documented carefully. It is easy to create a table file with many user–defined variables and then later on forget that they are required when setting up a problem file. Comments should be used to document all user–defined variables.

3.5.6. If–Then Statements

The *if* math operation provides a simple way to control the flow of calculations in a table. *If* statements are of the form

```
if (relationship) then math_operation
```

where *relationship* is

```
value relation value
```

and where *value* is a constant or a variable name. The *relation* is the symbol <, >, or =. For example, the statement

```
if (mach > 5.0) then add f2(time) no_extrap
   time =  0,   1,    2,   3
   f2   = 2.5, 2.6, 2.95, 3.8
```

says to add the value *f2* that is a function of *time* to the table value if the Mach number is greater than 5.

In an *if* statement, the relationship, such as *mach > 5.0*, is evaluated first. If the relationship is true, then the math operation following the *then* keyword is executed. In this case if the Mach number is greater than 5, the value of *f2* is interpolated as a function of time and added to the table value. If the relationship is false, that is, the Mach number is less than or equal to 5, nothing is done and the calculations proceed to the next math operation.

Only simple relationships are allowed in TAOS. Combinations such as *mach < 5 & alt > 50000* are not allowed. Relations for not equal to and greater than or equal to are not provided because they are redundant.

3.5.7. Goto Statements

Goto math operations and *labels* are also used to control the flow of calculations, and they are often used in conjunction with *if* statements. Every math operation can have an optional label. A label is any name or value placed in front of the math operation and separated from it with a colon. For example, the statements

```
label_1:   add  time
label_2:   sub  tmark
lable_3:   csto  delta_time
```

all have labels. Labels are like other names in that they cannot have imbedded delimiters (blanks, commas, etc.), so underscores are used to make them readable.

Labels are referenced by *goto* math operations. For example,

```
start
super: if (mach < 1.0) then goto sub  # "if" test
       add mach
       mult mach                      # This set of math
       csto mach2                     # operations is for
       add cdsup(mach2)               # Mach > 1 (supersonic)
         mach2 = 1.0, 4.0, 9.0, 25.0
         cdsup = 0.120, 0.094, 0.075, 0.055
       goto done

sub:   add cdsub(alpha)               # This set of math
         alpha = 0,  5,  10           # operations is for
         cdsub = 0.020, 0.023, 0.027  # Mach < 1 (subsonic)

done:  end
```

performs one set of calculations if the Mach number is less than one or subsonic and performs a different set of calculations if the Mach number is greater than one.

3.5.8. Skewed Tabulated Data

So far all sets of tabulated values have had a dependent value for every combination of independent values. The independent values form a matrix and values of the dependent variable are given for each combination. This is shown on the left side of Figure 3–6 for a function of two independent variables. The matrix of independent values forms a square for functions of two independent variables or a cube for functions of three independent variables, so these tables are called square tables.

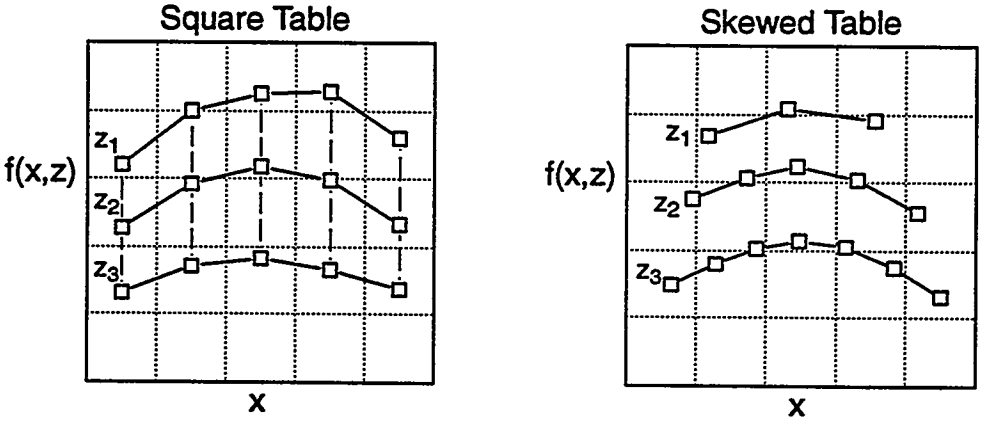

Figure 3–6. Square and Skewed Tabulated Data.

However, sets of tabulated data are often not complete. The right side of Figure 3–6 shows a function defined at different x values for z_1, z_2, and z_3. Not only are the values different, but the number of points in each curve is also different. In TAOS tabulated sets of data like this are called skewed. They can only be input using the full table format; skewed tables are not allowed in the simple table format.

The following example shows how skewed sets of tabulated data are input:

```
add cxo(alt,mach)
  alt  = 0, 50000
  mach =   3,    5,    7,    9
  cxo  = .020, .021, .019, .018,    # Values for alt = 0
         .021, .022, .021, .020     # Values for alt = 50000

  alt  = 100000, 150000
  mach =   7,   10,   15
  cxo  = .024, .023, .021,          # Values for alt = 100000
         .027, .025, .024           # Values for alt = 150000
```

The tabulated data is broken into smaller sets that are square in nature. In this example, values for the same Mach numbers are available at altitudes of 0 and 50,000 ft, so this part of the table is input first. Values for a different set of Mach numbers are available at altitudes of 100,000 and 150,000 ft, so these are input next. It is similar to entering two separate tables: one for the low altitudes and one for the high altitudes.

Another way to enter this table is to break it down even further as follows:

```
add cxo(alt,mach)
  alt  =  0
  mach =   3,    5,    7,    9
  cxo  = .020, .021, .019, .018,

  alt = 50000
  mach =   3,    5,    7,    9
  cxo  = .021, .022, .021, .020

  alt = 100000
  mach =   7,   10,   15
  cxo  = .024, .023, .021

  alt  = 150000
  mach =   7,   10,   15
  cxo  = .027, .025, .024
```

In this example, a simple one–dimensional table is given for each altitude. The one–dimensional tables can each have different Mach number values and a different number of Mach numbers.

In the general case, one–dimensional tables are given for the last independent variable (the right most variable) for each combination of remaining independent variables and values. The sets of data are grouped or nested such that values for the left most independent variable are in the outermost loop. The following three–dimensional example helps to clarify this:

```
add  ca(alt,mach,alpha)

  alt=10000,   mach=5,  alpha=0,2,4,6,8   # Values for 10,000
  ca = .20, .21, .23, .26, .28            # start here

  alt=10000,   mach=10,  alpha=0,2,4,6
  ca = .19, .19, .20, .22

  alt=10000,   mach=15,  alpha=0,2,4
  ca = .18, .19, .21

  alt=30000,   mach=10,  alpha=0,2,4,6    # Values for 30,000
  ca = .21, .23, .25, .29                 # start here

  alt=30000,   mach=15,  alpha=0,2,4
  ca = .20, .21, .23
```

A simple one–dimensional table that is a function of angle of attack is given for each combination of Mach number and altitude, but the number of angles of attack and the angle–of–attack values are different for each table. In this example, all of the independent values for a set of data are given on the same line; values are read free–field, so this format is allowed.

Altitude is the left most or first independent variable listed, so altitude values must be in the outermost loop. Thus, the Mach number values are nested within the altitude values. This means that the altitude values must be repeated until all Mach numbers

have been given. Note that there are more Mach number values at an altitude of 10,000 ft than at 30,000 ft. Altitude, Mach number, and angle of attack are all independent variables, so their values must be given in strictly increasing or decreasing order.

3.6 Examples

The following examples are provided to show what complete tables look like in practice. The first example is a thrust table for a rocket motor.

```
(recruits_thr)
  table  thrust
  start
     add thr(tmark)

        tmark = 0.00,   0.04,   0.06,   0.11,   0.21,   0.36
                0.65,   0.84,   0.96,   1.16,   1.36,   1.49
                1.52,   1.61,   1.70,   1.80,   2.00,   2.16
                99.0

        thr   =    0,    493,  37113,  37507,  36520,  36520,
                39482,  39235,  37803,  34842,  33263,  33361,
                32276,  18556,  11845,   7501,   2270,      0,
                    0
     mult  2.0            # No. of recruits
     mult  0.9936         # cosine of nozzle cant angle (6.5 deg)
   end
```

Thrust is a function of time, but the variable *tmark* is used instead of *time* because it can be reset to zero at any point in the trajectory. It is preferable to *tseg* because *tseg* is automatically reset at the beginning of each segment, whereas *tmark* is only reset when requested by the user. The value from the thrust table is multiplied by two factors: the first one is the number of rocket motors, and the second one accounts for the nozzles canted at 6.5° to the vehicle centerline.

The second example is an axial force coefficient table that is only a function of Mach number. This type of table is commonly used for ballistic sounding rockets that fly at zero angle of attack.

```
(strypi)
  table  ca  sref=5.241
  start
     add  ca(mach)
        mach = 0.00    0.6     0.72    0.8     0.85    0.9
               1.02    1.1     1.18    1.5     1.75    2.1
               2.9     3.2     4.0     5.2     6.2     8.0
        ca   = 0.97    0.96    0.99    1.03    1.1     1.27
               1.93    1.82    1.72    1.39    1.2     1.01
               0.74    0.68    0.62    0.54    0.47    0.36
  end
```

The table shown above is in the full table format, but it qualifies for the simple table format. In the simple table format, it appears as follows:

```
(strypi)
  table   ca(mach)   sref=5.241
        mach = 0.00    0.6     0.72    0.8     0.85    0.9
               1.02    1.1     1.18    1.5     1.75    2.1
               2.9     3.2     4.0     5.2     6.2     8.0
        ca   = 0.97    0.96    0.99    1.03    1.1     1.27
               1.93    1.82    1.72    1.39    1.2     1.01
               0.74    0.68    0.62    0.54    0.47    0.36
```

The tables shown so far are small, but they do not have to be. In practice, tables tend to be larger so they more accurately simulate the vehicle's characteristics. The trajectory calculations are often only as accurate as the aerodynamic and propulsion data.

The following mass flow table is for an Orbus rocket motor where the mass flow data is a function of time and is from an actual motor firing:

```
(orbus_mdt)
  table  mdot  units=lb/sec
  start
     add  mdt(tmark)

        tmark = -999.00     0.00     0.00     0.02     0.04     0.06
                  0.08     0.10     0.12     0.14     0.16     0.18
                  0.20     0.22     0.24     0.26     0.28     0.30
                  0.32     0.34     0.36     0.38     0.40     0.42
                  0.44     0.46     0.48     0.50     0.52     0.54
                  0.56     0.58     0.60     0.70     0.80     0.90
                  1.00     1.10     1.20     1.30     1.40     1.50
                  1.60     1.70     1.80     1.82     1.84     1.86
                  1.88     1.90     1.92     1.94     1.96     1.98
                  2.00     2.02     2.04     2.06     2.08     2.10
                  2.12     2.14     2.16     2.18     2.20     2.22
                  2.24     2.26     2.28     2.30     2.32     2.34
                  2.36     2.38     2.40     2.60     2.80     3.00
                  3.20     3.40     3.60     3.80     4.00     4.20
                  4.40     4.60     4.80     5.00     5.20     5.40
                  5.60     5.80     6.00     6.20     6.40     6.60
                  6.80     7.00     7.20     7.40     7.60     7.80
                  8.00     8.20     8.40     8.60     8.80     9.00
                  9.20     9.40     9.60     9.80    10.00    10.20
                 10.40    10.60    10.80    11.00    11.20    11.40
                 11.60    11.80    12.00    12.20    12.40    12.60
                 12.80    13.00    13.20    13.40    13.60    13.80
                 14.00    14.20    14.40    14.60    14.80    15.00
                 15.20    15.40    15.60    15.80    16.00    16.20
                 16.40    16.60    16.80    17.00    17.20    17.40
                 17.60    17.80    18.00    18.20    18.40    18.60
                 18.80    19.00    19.20    19.40    19.60    19.80
                 20.00    20.20    20.40    20.60    20.80    21.00
                 21.20    21.40    21.60    21.80    22.00    22.20
                 22.40    22.60    22.80    23.00    23.20    23.40
                 23.60    23.80    24.00    24.20    24.40    24.60
                 24.80    25.00    25.20    25.40    25.60    25.80
                 26.00    26.20    26.40    26.60    26.80    27.00
                 27.20    27.40    27.60    27.80    28.00    28.20
                 28.40    28.60    28.80    29.00    29.20    29.40
                 29.60    29.80    30.00    30.20    30.40    30.60
                 30.80    31.00    31.20    31.40    31.60    31.80
                 32.00    32.20    32.40    32.60    32.80    33.00
                 33.20    33.40    33.60    33.80    34.00    34.20
                 34.40    34.60    34.80    35.00    35.20    35.40
                 35.60    35.80    36.00    36.20    36.40    36.60
                 36.80    37.00    37.20    37.40    37.60    37.80
                 38.00    38.20    38.40    38.60    38.80    39.00
                 39.20    39.40    39.50    39.60    39.70    39.80
                 39.90    40.00    40.10    40.20    40.30    40.40
                 40.50    40.60    40.70    40.80    40.90    41.00
                 41.10    41.20    41.30    41.38    999.0

        mdt =     0.00     0.00     0.50    11.12    16.38    17.35
                 18.10    18.85    19.60    19.58    19.54    19.50
                 19.46    19.42    19.38    19.33    19.29    19.24
                 19.20    19.18    19.16    19.14    19.12    19.10
                 19.09    19.07    19.09    19.12    19.15    19.17
                 19.20    19.24    19.27    19.45    19.62    19.79
                 19.96    20.13    20.30    20.48    20.65    20.83
                 21.00    21.18    21.35    21.39    21.42    21.46
                 21.43    21.40    21.37    21.34    21.31    21.29
                 21.26    21.23    21.20    21.17    21.14    21.12
                 21.09    21.06    21.03    21.00    20.98    20.95
```

```
      20.92    20.89    20.86    20.83    20.81    20.78
      20.75    20.75    20.74    20.72    20.70    20.69
      20.68    20.67    20.65    20.63    20.61    20.60
      20.58    20.55    20.53    20.50    20.47    20.43
      20.40    20.36    20.32    20.28    20.24    20.20
      20.16    20.10    20.04    19.97    19.90    19.83
      19.77    19.70    19.64    19.56    19.48    19.37
      19.24    19.12    19.05    19.14    19.27    19.40
      19.54    19.68    19.83    19.97    20.12    20.27
      20.41    20.56    20.71    20.86    21.01    21.16
      21.30    21.44    21.58    21.73    21.88    22.01
      22.14    22.27    22.40    22.53    22.66    22.79
      22.92    23.05    23.18    23.31    23.43    23.55
      23.67    23.79    23.91    24.03    24.14    24.25
      24.35    24.44    24.54    24.63    24.74    24.84
      24.94    25.05    25.15    25.24    25.34    25.43
      25.51    25.59    25.66    25.73    25.79    25.85
      25.89    25.94    25.99    26.03    26.08    26.13
      26.18    26.23    26.27    26.32    26.36    26.40
      26.44    26.48    26.52    26.56    26.60    26.63
      26.66    26.68    26.70    26.71    26.72    26.74
      26.76    26.77    26.79    26.80    26.81    26.82
      26.83    26.85    26.86    26.88    26.90    26.92
      26.94    26.95    26.96    26.96    26.96    26.95
      26.90    26.74    26.58    26.49    26.42    26.35
      26.28    26.22    26.14    26.05    25.93    25.73
      25.53    25.32    25.10    24.87    24.65    24.42
      24.20    23.99    23.77    23.57    23.36    23.15
      22.94    22.73    22.54    22.34    22.13    21.92
      21.71    21.50    21.29    21.08    20.87    20.65
      20.43    20.21    19.99    19.77    19.55    19.31
      19.07    18.86    18.67    18.46    18.25    18.03
      17.82    17.62    17.52    17.41    17.29    17.18
      17.07    15.10    11.23     7.30     5.30     3.99
       3.11     2.57     2.08     1.76     1.44     1.15
       0.86     0.54     0.25     0.00     0.00

   csto  mdmotor

#  ACS roll control gas loss

   add   macsr(tmark)
      tmark =  -999.9   -0.000001  0.0    41.5   41.500001   999.9
      macsr =     0.      0.        1.0    1.0       0.         0.

   div  41.5
   div  32.174

   add  mdmotor
end
```

The first part of this table contains the mass flow in lb_m/sec as a function of time. The data points are given every 0.02 seconds. The second part of the table accounts for the mass loss from the attitude control system.

A final example is given of an axial force coefficient table for a reentry vehicle that is a function of Mach number, altitude, and angle of attack. This table is not shown in its entirety because it is too large, but enough is shown to illustrate the input pattern.

```
(rv_ca)

#       Aerodynamic axial force coefficient

table  ca     sref = 2.50

  start

    add  ca(alt,mach,alphat)

    alt =       0.0    mach = 14.00
    alphat =   0.000,   1.000,   2.000,   3.000,   4.000,   5.000,
               6.000,   8.000,  10.000,  12.000,  14.000,  15.000,
              16.000,  18.000,  20.000,  24.000,
        ca = 0.04556, 0.04673, 0.04990, 0.05401, 0.05877, 0.06408,
             0.06987, 0.08267, 0.09685, 0.11260, 0.13030, 0.13991,
             0.15006, 0.17202, 0.19634, 0.25216,

    alt =       0.0    mach = 16.00
    alphat =   0.000,   1.000,   2.000,   3.000,   4.000,   5.000,
               6.000,   8.000,  10.000,  12.000,  14.000,  15.000,
              16.000,  18.000,  20.000,  24.000,
        ca = 0.04263, 0.04381, 0.04720, 0.05167, 0.05684, 0.06255,
             0.06864, 0.08179, 0.09642, 0.11307, 0.13210, 0.14253,
             0.15358, 0.17765, 0.20473, 0.26911,

    alt =       0.0    mach = 18.00
    alphat =   0.000,   1.000,   2.000,   3.000,   4.000,   5.000,
               6.000,   8.000,  10.000,  12.000,  14.000,  15.000,
              16.000,  18.000,  20.000,  24.000,
        ca = 0.04051, 0.04169, 0.04508, 0.04955, 0.05472, 0.06043,
             0.06652, 0.07967, 0.09430, 0.11095, 0.12998, 0.14041,
             0.15146, 0.17553, 0.20261, 0.26699,

    alt =   20000.0    mach = 14.00
    alphat =   0.000,   1.000,   2.000,   3.000,   4.000,   5.000,
               6.000,   8.000,  10.000,  12.000,  14.000,  15.000,
              16.000,  18.000,  20.000,  24.000,
        ca = 0.04620, 0.04738, 0.05054, 0.05465, 0.05942, 0.06473,
             0.07052, 0.08332, 0.09750, 0.11325, 0.13094, 0.14056,
             0.15071, 0.17266, 0.19699, 0.25280,

    alt =   20000.0    mach = 16.00
    alphat =   0.000,   1.000,   2.000,   3.000,   4.000,   5.000,
               6.000,   8.000,  10.000,  12.000,  14.000,  15.000,
              16.000,  18.000,  20.000,  24.000,
        ca = 0.04330, 0.04448, 0.04787, 0.05234, 0.05751, 0.06322,
             0.06931, 0.08247, 0.09710, 0.11374, 0.13278, 0.14320,
             0.15425, 0.17832, 0.20540, 0.26978,

    alt =   20000.0    mach = 18.00
    alphat =   0.000,   1.000,   2.000,   3.000,   4.000,   5.000,
               6.000,   8.000,  10.000,  12.000,  14.000,  15.000,
              16.000,  18.000,  20.000,  24.000,
        ca = 0.04121, 0.04239, 0.04578, 0.05025, 0.05542, 0.06112,
             0.06722, 0.08037, 0.09500, 0.11165, 0.13068, 0.14111,
             0.15216, 0.17623, 0.20331, 0.26769,

.
.
.  etc.
.
.

    alt =  250000.0    mach = 18.00
    alphat =   0.000,   1.000,   2.000,   3.000,   4.000,   5.000,
               6.000,   8.000,  10.000,  12.000,  14.000,  15.000,
              16.000,  18.000,  20.000,  24.000,
```

```
    ca = 0.50320, 0.50438, 0.50777, 0.51224, 0.51741, 0.52311,
         0.52921, 0.54236, 0.55699, 0.57364, 0.59267, 0.60310,
         0.61415, 0.63822, 0.66530, 0.72968,

end
```

This table is a function of total angle of attack rather than angle of attack because C_A is symmetric, that is, C_A is the same for negative angles of attack as for positive angles of attack. Using total angle of attack eliminates duplicate tabulated values.

The skewed format for tabulated data is used even though the data meets the conditions for the square format. This is often done when the table is generated by a computer program, such as Aero,[20] because the skewed tabulated data format is more general and can be used for any type of tabulated data.

Intentionally Left Blank

4. Problem Files

Table files contain information defining the aerodynamic and propulsive forces acting on a vehicle. Problem files contain the remaining information required to compute a trajectory, for example, a vehicle's initial mass and its flight profile. This information is combined to integrate the equations of motion resulting in a trajectory or flight path.

TAOS has the capability to integrate several trajectories simultaneously. Each trajectory represents a separate vehicle. Information in the table and problem files is used to define the flight profiles for each trajectory or vehicle.

Each trajectory is divided into segments, where each segment can be described with simple guidance rules. Figure 4–1 shows an example of a maneuvering reentry trajectory. The first segment is flown ballistically, the second segment is a constant angle of attack pullout to level flight, the third segment is a level turn, and the last segment is flown ballistically to impact.

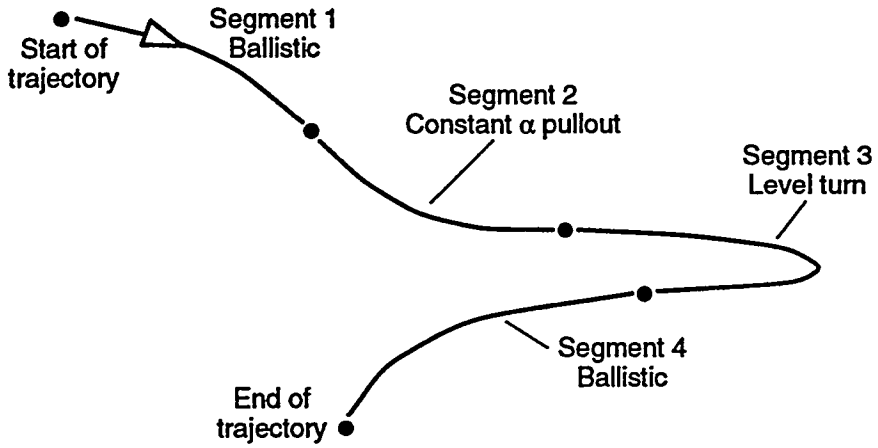

Figure 4–1. Segmented Flight Paths.

Input data in the problem file describes how each segment is computed, that is, the vehicle configuration, the guidance rules, the final conditions, and the next segment to execute. Segments are joined together continuously to form trajectories.

Problem files have a hierarchical structure as shown in Figure 4–2. A problem file contains one or more problems. Each problem contains information defining one or more trajectories. Each trajectory definition contains information defining one or more trajectory segments.

A problem can have any number of trajectories defined, and each trajectory can have any number of segments. The size of the problem is limited only by the amount of memory available on the computer and the computer's processing speed.

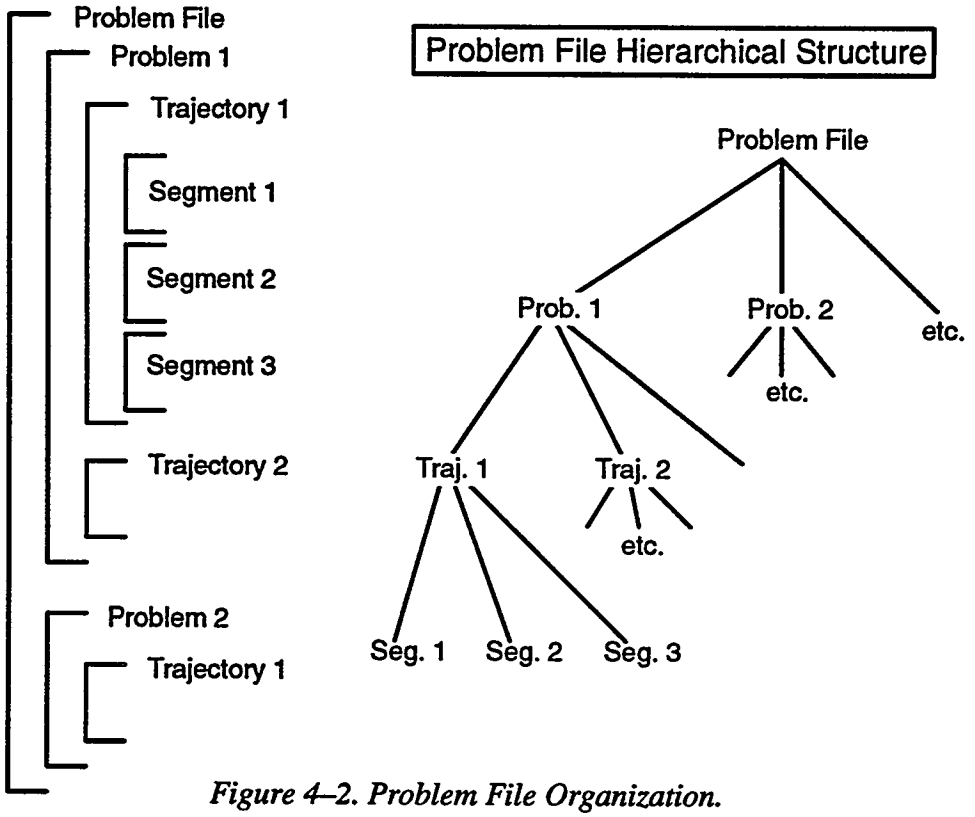

Figure 4–2. Problem File Organization.

Each trajectory in a problem is identified with a unique trajectory number and name. The trajectory number is used within TAOS to identify each trajectory. Both the trajectory number and name appear on output files to identify each trajectory.

Within a trajectory, each segment is identified with a unique segment number. Although a descriptive title can be given for a segment, it is not used to identify the segment. The segment numbers must be unique within a trajectory, but different trajectories can use the same set of segment numbers.

The following problem file defines a simple trajectory:

```
(pullout)

*title   Example of a maneuvering trajectory
*atmos  standard
*earth  wgs-84

*trajectory  1  rv  start on 1

  *initial  geodetic
     long=0.0        lat=0.0         psi=90.0        time=0.0
     alt=300000      vel=18000       gama=-30        wt=1000.0

  *print  time alt range mach vel gamgd dynprs alpha ca cn

  *segment 1  Ballistic flight
    *aero ca=(rv_ca)  cn=(rv_cn)
```

```
    *when  dynprs>1000  goto 2

  *segment 2  Constant alpha pullout to level flight
    *aero ca=(rv_ca)  cn=(rv_cn)
    *fly  alpha=10.0
    *when  gamgd=0  stop
*end
```

This problem file contains a single problem beginning with *(pullout)* and ending with **end*. Although problem files can contain more than one problem, it is good practice to only put one problem in each file and run each problem as a separate task or process. This way if the first problem fails, it does not prevent other problems from running.

Data Blocks

Each set of information in the problem file begins with an asterisk and a name, for example, **title*, **atmos*, and **trajectory*. These are special names TAOS keys on so it will know what type of information follows.

For example, information defining the initial conditions of the trajectory is given after the **initial* keyword as follows:

```
*initial  geodetic
   long=0.0       lat=0.0        psi=90.0       time=0.0
   alt=300000     vel=18000      gama=-30       wt=1000.0
```

This set of information (everything from the **initial* keyword up to the next **print* keyword) is called a data block. This particular one is called the **initial* data block. Information in this data block gives the vehicle's initial state.

Other data blocks in this problem file include the **title* data block which gives a problem title, the **atmos* data block which says to use the 1976 U.S. standard atmosphere, and the **earth* data block which says to use the WGS–84 earth model. Each data block has a variety of options to choose from.

This problem defines a single trajectory containing two segments. Indentation is used to show the problem file structure. The **initial*, **print*, and **segment* data blocks are all part of the **trajectory* block. The **aero*, **fly*, and **when* data blocks are part of the **segment* blocks; they are repeated for each segment. The indentation follows the structure shown in Figure 4–2.

Section 4 of this report is organized the same way as the problem file. Each data block is discussed in a separate subsection. The subsections are grouped according to the organization shown in Figure 4–2. Section 4.1 contains some general information about the problem file format, Section 4.2 discusses the segment data blocks, Section 4.3 discusses the trajectory data blocks, and Section 4.4 discusses the problem data blocks. The last section, Section 4.5, contains several example problem files.

4.1. File Format

Problem files are text files so they can be created with any standard text editor such as UNIX's *vi* or Silicon Graphics' *jot*. They can also be created with other software, such as an interactive trajectory design code.

Information in problem files is read into TAOS one line at a time. Each line of text contains some names and values separated by special characters such as spaces or commas. These special characters are called delimiters; valid delimiters are white-space characters (space, tab, etc.), commas, equal signs, parentheses, colons, greater–than signs and less–than signs. Because delimiters are used to separate names and values, they cannot be imbedded within a name or value.

Internally TAOS requires names that are lowercase. Names and keywords entered in uppercase are automatically converted to lowercase, so from a user standpoint case is not important.

Names and Values

Names and values can be placed anywhere on a line; they do not have to be in certain columns. This free–field style of input allows indentation to make it easier to visualize the problem file structure and to avoid input errors.

Because of the free–field style of input, the order of names and values in the problem file is often important. For example, in the trajectory data block

```
*trajectory 3 missile start on 23
```

the order of each item in the data block is important. The keyword **trajectory* must be followed by a trajectory number. This is followed by a trajectory or vehicle name. The keywords *start on* must follow these two values, and, finally, the last value is the starting segment number. In other data blocks, where lists of variables and values are given, the order may be less critical.

Numerical values can be input as integers without a decimal point, as real numbers with a decimal point, or as real numbers in scientific notation (e–format). For example,

```
*initial  geodetic
     time = 20,      vel = 2.0e4,     wt=345.5,      gama = -24.3
```

uses the value *2.0e4* which is read as 2.0×10^4 or *20,000.*

Note that delimiters cannot be imbedded within a value, so the value

```
wt = 10,000          # Incorrect value
```

is incorrect. The comma is interpreted as a delimiter, so TAOS thinks there are two values instead of one. In this case, it reads 10 and 0 instead of 10000.

Survey, Search, and Optimization Parameters

One of the powerful features of TAOS is its ability to systematically vary input parameters for tradeoff studies. This is accomplished with surveys, searches, and optimization loops. Surveys vary input parameters through a set of known values, and for each value, a trajectory is computed. Searches vary one or two input parameters to achieve certain conditions during the trajectory. For example, the initial flight path angle can be varied until the final range equals a desired value. Optimization loops vary a set of input parameters to maximize or minimize a trajectory value subject to some constraints. For example, the angle–of–attack history during a trajectory can be varied such that range is maximized while maintaining an impact velocity above a desired value and maintaining a minimum dynamic pressure.

All of these methods change or adjust values of input variables given in the problem file. They can only change numerical values; surveys, searches, and optimization loops cannot be used to change table names or other keywords in a problem file.

Numerical values are designated for a survey by setting a variable equal to a special keyword. For example,

```
*initial  geodetic
     time = 20,      vel = 2.0e4,     wt=surv-1,      gama = -24.3
```

has an initial weight that is no longer set to a numerical value. Instead it has been set to the keyword *surv–1*, which says to set the initial weight to values given in the **survey* data block for survey loop number 1 (Section 4.4.11).

Similarly, values can be designated for searches with the *srch–n* keyword where *n* is the search number (Section 4.4.9). Values are designated for optimization loops with the *optx–n* keyword, where *x* is the optimization loop letter and *n* is the optimization parameter number (Section 4.4.6).

At this point it is only necessary to know that this capability exists; for more details, see the separate sections on surveys, searches, and optimization.

Titles

Some data blocks have titles. Titles are special in that the delimiters are ignored. Titles extend from the first nonblank character to the end of the line. Titles are also special in that they are not converted to lowercase; case is preserved as input.

Comments

Comments, starting with the special # character, can be placed anywhere in a problem file. Comments extend from the # character to the end of the line. Blank lines are ignored, so they can be used to put space between sections of information to make it easier to read.

For example, the data block

```
*define  alt_km                  # Altitude in km
   alt_km = alt / 3280.84;       # Convert units of altitude
```

defines a new variable called *alt_km* that is the vehicle's altitude in kilometers rather than feet. Comments have been used to document what this data block does.

4.2. Segment Data Blocks

Segment data blocks provide information describing how to calculate a trajectory segment. This includes numerical integration parameters, the vehicle configuration, guidance rules, segment final conditions, and what to do next. This is a lot of information, so it is organized into data blocks. Thus, segment data blocks contain other types of data blocks.

Segment data blocks begin with a line containing the keyword **segment*, for example,

```
*segment  23  2nd Stage Boost
```

The value following **segment* is the segment number, in this case segment number 23. It is required and it must be unique within a trajectory. An optional title, such as *2nd Stage Boost*, follows the segment number.

The remainder of the segment data block is made up of other types of data blocks as shown in Table 4–1. A segment data block extends from the **segment* keyword to the next trajectory or problem data block keyword. Usually this is another **segment* keyword or a **trajectory* keyword (See Table 4–8 for a list of trajectory data block keywords and Table 4–13 for a list of problem data block keywords).

Table 4–1. Segment Data Blocks.

Data Block	Description
*aero	Defines aerodynamic coefficients
*constants	Provides values for user–defined variables
*cg	Defines the center of gravity
*fly	Defines a guidance rule
*increment	Allows discontinuities in the vehicle's state
*inertial	Aligns an inertial platform coordinate system
*integ	Provides parameters for numerical integration
*limits	Defines limits on the flight path
*prop	Defines propulsive thrust and mass flow
*rail	Provides parameters for rail launches and sleds
*reset	Allows discontinuities in the vehicle's state
*when	Provides final conditions and what to do next

The following segment data block is an example of a rail–launched missile:

```
*segment  1  Rail Launch
   *integ  dtprnt=0.10  dt=0.01
   *aero  ca=(stage1_ca)  cn=(stage1_cn)
   *prop  thrust=(thrust1)  mdot=(mdot1)
```

```
*prop  thrust=(t_recruits)  mdot=(m_recruits)
*rail  launch  cfstat=0.15  cfslid=0.01
*when  plength=25.0  goto 3
```

This segment data block contains an **integ* block giving integration and print step sizes, an **aero* block with aerodynamic table names, two **prop* blocks with thrust and mass flow table names, a **rail* block for the rail launch guidance, and a **when* block with the final conditions. The **when* block is also used for continuing the trajectory; in this case, segment 3 is computed next.

The following sections describe the input parameters and format for each segment data block. Additional examples of segment data blocks are given in Section 4.5.

4.2.1. *Aero Data Block

The **aero* data block is used to define the aerodynamic coefficients. It is optional; if it is not input, all aerodynamic coefficients are set to zero. If more than one **aero* data block is input in a segment, the aerodynamic forces are computed for each **aero* data block and then added together.

There are three different sets of aerodynamic coefficients that can be input: the axial and normal force coefficients (C_A and C_N), the lift, drag, and side force coefficients (C_L, C_D, and C_S), and the body x, y, and z force coefficients (C_X, C_Y, and C_Z). The aerodynamic coefficients are defined in Section 2.3.2.

Within each **aero* data block, a consistent set of aerodynamic coefficients must be input. Coefficients from different sets cannot be mixed. For example, C_L and C_D are in the same set, so they form a valid set of coefficients, whereas C_L and C_A are not in the same set, so they cannot be used together.

If a segment has more than one **aero* block, it is recommended that all of them use the same type of aerodynamic coefficients. If this is done, then all of the output aerodynamic coefficients are the sum of the coefficients from each **aero* block. Otherwise, the forces used to compute the trajectory are correct, but only the C_L, C_D, and C_S output coefficients are correctly totaled.

Constant Coefficients

Aerodynamic coefficients can be input either as constants or as tables. Constant values are input by giving the coefficient name and its value as follows:

```
*aero     ca=0.0215    cn=0.118    sref=5.437
```

This data block defines the axisymmetric coefficients C_A and C_N. These values are constant for the duration of the segment.

Besides the aerodynamic coefficient values, a value has been given for the reference area S_{ref}. The reference area is used to compute the aerodynamic forces from the coefficients and it has default units of ft^2. It is required when all coefficients are set to constants; when tables are used, the reference area can be given in one of the tables.

When a reference area is specified in the **aero* data block, it overrides values given in tables. When multiple **aero* blocks are given or when aerodynamic tables contain different values of S_{ref}, S_{ref} should not be given in the **aero* data block.

Aerodynamic Tables

Aerodynamic coefficients can also be given in tables by setting the coefficient names equal to a table identification name enclosed in parentheses. For example,

```
*aero     ca = (ca_clean)     cn = (cn_clean)
```

requests a table called *ca_clean* for the axial force coefficient and a table called *cn_clean* for the normal force coefficient.

Tables can be used for some coefficients while others can be set to constants. In the following example, the lift coefficient is set to a table name and the drag coefficient is set to a constant:

```
*aero    cl=(lift_x)    cd=0.2351
```

The side force coefficient is not given, so it is set to zero. The reference area is not given either, so it must be provided in the *lift_x* table.

User–Defined Variables

Tables can be functions of user–defined variables as shown in Section 3.5.5. If user–defined variables are referenced in a table, then values for them must be given in the problem file.

One place values can be given for user–defined variables is in the **aero* data block. Although values can be given for any user–defined variable within this data block, usually values are only given for the user–defined variables in the aerodynamic tables. Values for other user–defined variables are given in the data blocks where they are referenced. This way the table name and its user–defined variables are given together in the problem file. Another method is to give values for the user–defined variables in a **constants* data block (Section 4.2.2).

Once a value has been given for a user–defined variable, it remains set at that value unless changed in a subsequent data block. This means that values for user–defined variables do not have to be repeated within each segment if they do not change. However, it is good practice to provide values in each segment to avoid input errors.

The following **aero* data block defines values for two user–defined variables:

```
*aero  cd=(drg_model)  cl=(lft_model)
    flaps=25,   gear=0,  sref=225.0
```

This block provides table names for C_L and C_D, and it sets the reference area to 225 ft^2. It also provides values for two user–defined variables: *flaps* and *gear*. These variables must appear in the tables; arbitrary variable names that never appear in a table cannot be input. These variables could be flags in the table used for flap deflection and gear up or down.

4.2.2. *Constants Data Block

Tables can be functions of user–defined variables as shown in Section 3.5.5. If user–defined variables have been referenced in tables, then values for these variables must be provided. These values can be given in the **aero*, **prop*, **cg*, or **constants* data blocks. Values only need to be given for the user–defined variables in tables referenced in a segment; if a table is not used in a segment, then values do not have to be provided for its user–defined variables.

The **constants* data block is optional. It is provided as a convenience, so that values for all user–defined variables can be grouped together. The **constants* data block is only used to provide values for user–defined variables. If the tables are not a function of user–defined variables, a **constants* data block should not be input.

There are two styles for providing user–defined variable values. One style defines the user–defined variable values in the data block where the table is referenced. For example, an **aero* data block that references a table with a user–defined variable also provides a value for that variable. This style uses the **aero*, **prop*, and **cg* data blocks to define values for the user–defined variables.

The other style is to provide values for all user–defined variables in the **constants* data block. This groups all user–defined variables together in one place. Some people prefer the first style and others prefer this style, so both are provided.

The **constants* data block consists of the **constants* keyword followed by the user–defined variables set equal to their values. For example,

```
*constants  type=3,  tmult=1.2,
     mmult=1.15
```

sets the user–defined variable *type* equal to 3.0, *tmult* equal to 1.2, and *mmult* equal to 1.15. These variables must be used in the tables supplied with this problem file.

Once a value has been given for a user–defined variable, it remains set at that value unless changed in a subsequent data block. This means that values for user–defined variables do not have to be repeated within each segment if they do not change. However, it is good practice to provide values in each segment to avoid input errors.

4.2.3. *Cg Data Block

One of the assumptions in TAOS is that the control system is capable of maintaining the vehicle's body attitude. For example, when a constant angle of attack guidance scheme is requested, it is assumed that there is a control system capable of maintaining a constant angle–of–attack body attitude. For many aerospace vehicles, this control system uses aerodynamic controls, such as flaps, canards, and elevators. The control settings required to maintain steady flight at various body attitudes are a function of the vehicle's center of gravity, that is, maintaining a constant angle of attack requires different control settings depending on the vehicle's center of gravity.

The aerodynamic coefficients used in TAOS are for steady trimmed flight. Because of this, they are generally a function of the vehicle's center of gravity. The vehicle's center of gravity, in turn, is often a function of its weight. As fuel or propellant is used, the weight and center of gravity change, and this affects the aerodynamics.

This is handled in TAOS by making the aerodynamic tables a function of the center of gravity or *cg* state variable and by defining a value for *cg* in the **cg* data block. The *cg* can be set to a constant, or a table can be used to compute its value. The **cg* data block is optional; *cg* is set to zero if it is not input.

The **cg* data block consists of the **cg* keyword, followed by the *cg* variable name and its value. The following **cg* data block sets the center of gravity to a constant value:

```
*cg    cg=0.69
```

The center of gravity can be stated in inches, in percent of body length, or in any other convenient units as long as its use is consistent in both the **cg* data block and the aerodynamic tables. TAOS treats the center of gravity as a nondimensional value.

A *cg* table can also be used to compute the vehicle's center of gravity. For example,

```
*cg    cg=(wt_cg)    config=2
```

requests TAOS to use a table called *wt_cg* to compute the center of gravity. This table, like others, can be a function of weight or any of the other state and user–defined variables.

In the example above, a user–defined variable, *config*, has been assigned a value. The **cg* data block can be used to assign values to any of the user–defined variables, but it is usually only used to assign values to the user–defined variables referenced in the *cg* table.

4.2.4. *Fly Data Block

TAOS uses input guidance rules to determine values for four control variables so a trajectory can be calculated. The body attitude is specified by three control variables, such as, angle of attack, angle of sideslip, and bank angle. The thrust level is controlled by a fourth control variable called power setting. Guidance rules and control variable values are input with **fly* data blocks.

The **fly* data blocks are optional. If no **fly* data blocks are given for a segment, then all control variables are set to zero. Unless unusual aerodynamic tables are given, this represents a ballistic trajectory.

Each **fly* data block specifies a guidance rule which determines a value for one or more of the control variables. Since there are four control variables, there can be a maximum of four **fly* data blocks in each segment. The control variables default to zero, so if this is an acceptable value for a control variable, a **fly* data block is not required for that control variable.

Body Attitude Guidance

The three body–attitude control variables are treated separately from the power setting control variable. Values for these control variables can be specified directly relative to the surrounding air mass with aerodynamic angles, or they can be specified directly with conventional Euler yaw, pitch, and roll angles relative to inertially fixed, geocentric, or geodetic coordinate systems. They can also be specified indirectly by supplying a desired flight condition, such as level flight, and letting TAOS solve for the control values that give the desired flight condition.

Valid body–attitude angles that can be specified directly in guidance rules are given in Table 4–2. The angles in Table 4–2 can be combined to form nine sets of consistent angles as shown in Table 4–3. A set of guidance rules in a segment must contain a consistent set of angles.

Often when the angles are directly specified, all three angles in a set are input in three **fly* data blocks. This is not required, however. If only one angle is specified directly, it leaves the remaining two angles free. Other **fly* data blocks may specify flight conditions that must be satisfied and the free angles will be set such that the flight conditions are met. Or if no other **fly* data blocks are given, the angles default to zero.

Table 4–2. Body–Attitude Angles for Vehicle Guidance.

Variable	Symbol	Description
alpha	α	Angle of attack
alphat	α_T	Total angle of attack
bankgc	μ_{gc}	Geocentric bank angle
bankgd	μ_{gd}	Geodetic bank angle

beta	β	Angle of sideslip
betae	β_e	Euler angle of sideslip
phi	ϕ_w	Windward meridian
pitchgc	Θ_{gc}	Geocentric pitch angle
pitchgd	Θ_{gd}	Geodetic pitch angle
pitchi	Θ_i	Inertial platform pitch angle
rollgc	Φ_{gc}	Geocentric roll angle
rollgd	Φ_{gd}	Geodetic roll angle
rolli	Φ_i	Inertial platform roll angle
yawgc	Ψ_{gc}	Geocentric yaw angle
yawgd	Ψ_{gd}	Geodetic yaw angle
yawi	Ψ_i	Inertial platform yaw angle

Table 4–3. Consistent Sets of Body–Attitude Angles.

Set	Body–Attitude Angles		
1	alpha	betae	bankgc
2	alpha	beta	bankgc
3	alphat	phi	bankgc
4	alpha	betae	bankgd
5	alpha	beta	bankgd
6	alphat	phi	bankgd
7	yawgc	pitchgc	rollgc
8	yawgd	pitchgd	rollgd
9	yawi	pitchi	rolli

The angles given in the **fly* data blocks are checked to ensure that they form a consistent set according to Table 4–3. If some angles are not specified directly, those that are specified are checked for consistency with Table 4–3. The lowest numbered set is selected containing the specified angles. The remaining unspecified body–attitude angles in this set are allowed to vary to meet other **fly* flight conditions.

Some examples of **fly* data blocks will help clarify their use. The data blocks

```
*fly  alpha = 5.0
*fly  betae = 2.5
```

say to fly at an angle of attack of 5° and a sideslip of 2.5°. Only two angles are specified. Using Table 4–3, these angles are both in Set 1 and Set 4, but Set 1 is chosen because it has a lower set number, so the remaining body–attitude angle is *bankgc*. Since this angle is not specified and no other **fly* data blocks are given, it defaults to zero. Power setting also defaults to zero because it is not specified.

The most common format of each **fly* data block is the **fly* keyword followed by one of the guidance variables and its value. Another format is available, used primarily for optimization, which is discussed later in this section.

In the example above, use of *alpha* and *betae* is encouraged over *alpha* and *beta* because of the way that the two sideslip angles, *beta* and *betae*, are defined. When *alpha* and *beta* are used and they become $\pm 90°$, the standard angle definitions are undefined. The standard definitions are modified to avoid these problems, but this system of angles is not as robust as *alpha* and *betae* (Section 2.1.9).

The following example illustrates an incorrect set of **fly* data blocks:

```
*fly  alphat = 5.0      # Incorrect guidance rules
*fly  betae = 5.0       # Angles are inconsistent
```

This is an inconsistent set of body–attitude angles because total angle of attack is associated with windward meridian, not sideslip. *Alphat* and *betae* are not contained in one of the valid sets of angles in Table 4–3, so this set of **fly* data blocks generates an input error.

Referencing Current Values

The data blocks

```
*fly  yawgd = *
*fly  pitchgd = *
*fly  rollgd = *
```

show another feature of the **fly* data block. All of the values have been set to an asterisk rather than a number. An asterisk tells TAOS to use the value from the end of the previous segment; asterisks can be used in the **fly* data block in place of any number. Asterisk values should not be used on the first segment of a trajectory because there are no previous values to reference.

Power Setting

The power setting variable, *power*, is a special control variable that can be used in combination with the tables to simulate an engine throttle. If the thrust and mass flow tables are a function of *power*, then *power* can be used to vary the thrust and mass flow values like a throttle. Power setting is a nondimensional value so its units can be anything as long as they are consistent in both the table and problem files.

For example, if thrust and fuel flow tables for a jet engine are entered as a function of the engine rpm using *power* to represent rpm, then the rpm value can be controlled in the **fly* data block by setting the *power* variable. A value for *power* can be directly input, or it can be indirectly specified by entering a desired flight condition, such as constant Mach number.

Flight Condition Guidance

So far control variables have been specified directly, but they can also be specified indirectly with desired flight conditions, such as level flight. When they are specified

indirectly, TAOS determines the control variables that have not been directly specified (the free control variables) and varies them using an iterative Newton–Raphson search method to solve for the desired flight conditions (Section 2.5.1).

The guidance rules or flight conditions that can be specified in the **fly* data block are listed in Table 4–4. The iterative search method used to solve for these conditions usually converges, but convergence is not guaranteed. If the actual flight conditions are significantly different from the desired flight conditions, a convergence problem may occur.

Table 4–4. Flight Conditions for Vehicle Guidance.

Variable	Description
alt	Fly at a constant altitude
cl	Fly at a constant lift coefficient
cs	Fly at a constant side–force coefficient
downria	Fly down the range insensitive axis
dynprs	Fly at a constant dynamic pressure
gamgc	Fly at a constant geocentric flight path angle
gamgd	Fly at a constant geodetic flight path angle
intercept	Fly to intercept another trajectory
l/d	Fly at a constant lift–to–drag ratio
l/d_max	Fly at the maximum lift–to–drag ratio
mach	Fly at a constant Mach number
nx	Fly at a constant axial specific load factor
ny	Fly at a constant lateral specific load factor
nz	Fly at a constant normal specific load factor
propnav	Fly to intercept with proportional navigation
psigc	Fly at a constant geocentric heading angle
psigd	Fly at a constant geodetic heading angle
thrust	Fly at a constant thrust
upria	Fly up the range insensitive axis
vel	Fly at a constant velocity

If problems occur, first ensure that the free control variables are capable of controlling the vehicle to meet the desired flight conditions. For example, if bank angle is zero, then angle of attack has no effect on heading, so TAOS is not able to solve for an angle of attack to fly a given heading angle.

Then check the control variable limits, which are given in the **limits* data block (Section 4.2.8). Upper and lower bounds on the control variables can be specified to keep the values reasonable. Values should be limited to the range of values given in the tables to avoid extrapolation problems.

Finally, check the value of *dtguid* in the **integ* data block. It controls the rate at which the flight path is corrected from the actual flight conditions to the desired flight conditions. It acts as a time constant or gain, and if it is too large or too small, the flight path does not correct to the desired conditions.

The following set of **fly* data blocks demonstrates use of the flight conditions guidance rules:

```
*fly  gamgd = 0
*fly  mach = 0.80
*fly  bankgd = 0
*fly  betae = 0
```

All four data blocks have been input specifying two control variables directly, *bankgd* and *betae*, and two control variables indirectly, *alpha* and *power*. TAOS solves for values of *alpha* and *power* that result in a zero flight path angle (level flight) and a 0.8 Mach number. The last two **fly* data blocks are optional because these variables default to zero, but they have been included in this example for clarity.

Special Guidance Rules

The guidance rules *l/d_max*, *intercept*, *propnav*, *downria*, and *upria* are special cases. The *l/d_max* guidance rule solves for the angle of attack that maximizes the lift–to–drag ratio. It does not require a value, so it is used as follows:

```
*fly  l/d_max
```

The *intercept* guidance rule places the vehicle on an intercept trajectory with another vehicle. Its value is set to the trajectory number of the target vehicle, for example,

```
*fly  intercept = 3
```

says to intercept vehicle or trajectory number 3. The vehicle's trajectory will be immediately pointed towards the estimated intercept point using a predictive guidance method (Section 2.5.2).

Similarly the *propnav* guidance rule is used to intercept another trajectory with proportional navigation (Section 2.5.2). Again the guidance rule value is set to the trajectory number of the target vehicle. For example,

```
*fly  propnav = 2
```

uses proportional navigation to intercept trajectory number 2. The value of *dtguid*, given in the **integ* data block, is used for the proportional navigation constant.

The intercept and proportional navigation guidance rules require two free control variables: one control for yaw and one control for pitch. These two special guidance rules are equivalent to two normal guidance rules. When they are used, a maximum of three **fly* data blocks can be input.

The *downria* and *upria* guidance rules orient the body x axis along the range insensitive axis. If the velocity is incremented in the direction of the range insensitive axis,

it changes the time to impact but does not change the range. The *downria* guidance rule solves for the range insensitive axis that reduces the flight time, and the *upria* guidance rule solves for the axis that increases flight time (Section 2.5.3).

The range insensitive axis solution procedure uses the initial impact point (IIP) to determine the vehicle's range. The guidance rule value, generally zero, is used as the impact altitude. For example,

```
*fly downria = 100000
```

solves for the range insensitive axis using an "impact" altitude of 100,000 ft.

The *downria* and *upria* guidance rules are similar to the intercept guidance rules in that they require two free control variables: one control for yaw and one control for pitch. These special guidance rules are equivalent to two normal guidance rules.

Rate Guidance

So far all examples have shown guidance rules that specify body–attitude angles or flight conditions. Body–attitude rates and flight condition rates can also be specified by adding the suffix *dt* to the end of the variable name (*dt* stands for dot or the first derivative with respect to time). For example, angle of attack rate is given by *alphadt*. The data blocks

```
*fly  gamgd = 0
*fly  psigddt = 5.0
*fly  betae = 0
*fly  powerdt = 1000
```

request level flight (*gamgd = 0*), a heading rate of 5 deg/sec (*psigddt = 5*), no sideslip (*betae = 0*), and a power setting rate of 1000 units/sec (*powerdt = 1000*). If power setting is proportional to thrust in the propulsion tables, this produces a level accelerating turn. Angle of attack and geocentric bank angle are varied to turn at the requested heading rate while maintaining level flight.

Guidance Tables

If constant values or rates do not represent the desired trajectory, values can be interpolated from a table that is a function of a state variable as follows:

```
*fly  alpha vrs tseg  interp-2
       0.0      0.0
       5.0      1.0
       6.0      2.0
       5.0      3.0
       0.0      4.0
```

This data block says to interpolate angle of attack from the table as a function of the segment time. The *vrs* keyword separating the guidance variable and the state variable is required. Guidance tables can be functions of the same state variables as tables (Table 3–2).

The *interp–2* keyword is optional and requests interpolation method 2 (circular interpolation). Three interpolation methods are available for guidance tables:

interp–1	linear interpolation
interp–2	circular interpolation[23]
interp–3	polynomial interpolation

Linear interpolation, *interp–1*, is the default.

Guidance tables are primarily used for flight path optimization where the angle–of–attack (or another control variable) time history is varied to minimize or maximize a flight condition subject to constraints. Sections 4.4.6 and 4.5.3 contain examples of this type of guidance table.

During optimization, the table values are perturbed by a very small amount when calculating gradients. To ensure the gradient is computed accurately, it is important to integrate to each point in the table. Consequently, the integration step size is automatically adjusted such that the table points are at the end of integration steps. This only occurs when the guidance table is a function of a time variable (*time*, *tseg*, or *tmark*).

4.2.5. *Increment Data Block

Normally values for a vehicle's state and time are continuous across trajectory segments; however, the vehicle's state and time can be instantaneously changed forming a discontinuity by including an **increment* data block. This is most often used for weight changes, for example, dropping a weapon or missile staging. The data block

```
*increment   wt=-325
```

instantaneously reduces weight 325 lb (if default units are used).

The **increment* data block is identical to the **reset* data block (Section 4.2.11), except that **reset* sets a time or state variable to an input value, whereas **increment* adds the input value to the current value of the time or state variable. The **increment* data block increments or delta's the time or state variable plus or minus from its current value.

If both **reset* and **increment* data blocks are present in a segment, values are reset first and then incremented. Multiple **reset* and **increment* data blocks can be given in a segment; they are executed in the order given.

The variables that can be changed with the **increment* data block are listed in Table 4–5. Position and velocity variables are maintained in many different coordinate systems so they have been grouped in Table 4–5 in consistent sets. Sets of position and velocity variables in **increment* data blocks must be in the same coordinate system.

For example, if the velocity *xecicdt* is incremented, then values for *yecicdt* and *zecicdt* can also be changed. But values for velocity in other coordinate systems, for example, *vel* cannot be changed. This restriction applies to all of the position and velocity variables in Table 4–5.

*Table 4–5. *Increment Variables.*

Variable	Description
alt long latgd	Altitude Longitude Geodetic latitude
rcm long latgc	Distance from earth center Longitude Geocentric latitude
xecfc yecfc zecfc	Earth centered fixed x coordinate Earth centered fixed y coordinate Earth centered fixed z coordinate
xecic yecic zecic	Earth centered inertial x coordinate Earth centered inertial fixed y coordinate Earth centered inertial fixed z coordinate
dxb dyb dzb	Position increment along body x axis Position increment along body y axis Position increment along body z axis
vel gamgc psigc	Velocity Geocentric flight path angle Geocentric heading angle

vel gamgd psigd	Velocity Geodetic flight path angle Geodetic heading angle
xecfcdt yecfcdt zecfcdt	Velocity earth centered fixed x coordinate Velocity earth centered fixed y coordinate Velocity earth centered fixed z coordinate
xecicdt yecicdt zecicdt	Velocity earth centered inertial x coordinate Velocity earth centered inertial y coordinate Velocity earth centered inertial z coordinate
wt mass fuel	Weight Mass Fuel used
time tseg tmark	Time Segment time Mark time
range grseg grmark	Ground range Segment ground range Mark ground range
plength plseg plmark	Path length Segment path length Mark path length
iip_beta	Ballistic coefficient for IIP calculations

Weight and mass are similar. If one is given, then the other cannot be given because they both refer to the same state variable. Fuel, on the other hand, can be reset or incremented at any time.

The time, range, and path length variables are independent of each other, so any combination is valid. The segment variables (*tseg*, *grseg*, and *plseg*) are automatically reset to zero at the beginning of a segment. The **increment* values are applied after these variables have been reset to zero.

Problems with a single trajectory can have discontinuities in time so the *time* variable can be reset or incremented. But problems with more than one trajectory use time to relate the trajectories to each other so discontinuities in time can cause errors. The variable *time* should not be reset or incremented in problems with multiple trajectories. The variables *tseg* and *tmark* do not have this problem because they measure relative time.

The "mark" variables (*tmark*, *grmark* and *plmark*) are useful to keep track of time, range, and path length from intermediate points in the trajectory. At the beginning of the trajectory, they correspond to the variables *time*, *range*, and *plength*. But the mark variables can be restarted at any segment with the **reset* data block, or they can be incremented with the **increment* data block.

The ballistic coefficient for the initial impact point (IIP) calculation can be reset or incremented at any time. It is only used for the initial impact point calculation.

Object Deployment

The variables *dxb*, *dyb*, and *dzb* are used for deploying objects or dropping weapons where one trajectory is initialized from another. A deployed object's initial position is usually offset slightly from the vehicle's center of mass. This offset is the position of the object relative to the body coordinate system before the object is deployed or dropped.

A deployed object is generally initialized from the beginning of an existing trajectory segment, for example,

```
*trajectory 2 Obj2 start on 1

  *initial from trajectory 1, segment 6

  *segment  1  deployment
    *reset wt=50
    *increment velibx=5
    *increment dxb=1.5  dyb=3.2  dzb=0.35
    *fly pitchi = *
    *fly yawi = *
    *fly rolli = *
    *when tseg>0  goto 2
      .
      .
```

initializes a new trajectory 2 from the beginning of segment 6 in trajectory 1 representing an object deployment in space. The initial state of the new object is copied from trajectory 1 and it can be adjusted with **reset* and **increment* data blocks in the first segment.

In this example, the weight of the new object is set to 50 lb, and it is deployed with a velocity increment (delta–v) of 5 ft/sec along the body x axis. Its position is adjusted relative to the body coordinate system with the variables *dxb*, *dyb*, and *dzb* because its center of mass is slightly offset from the original vehicle. The variables *dxb*, *dyb*, and *dzb* always act as increments to a vehicle's position so their values can be set in a **increment* or **reset* data block with the same effect.

The guidance rules used in this example hold a constant body attitude in inertial space. This ensures that the body attitude of the object matches the body attitude of the original vehicle so the position (*dxb*, *dyb*, and *dzb*) and velocity (*velibx*) increments are applied in the correct direction. The same result can be obtained by making the guidance rules of the first segment the same as those of the original vehicle. The first segment can be immediately terminated with a final condition of *tseg>0* so that other guidance rules can be used to calculate the trajectory of the deployed object.

4.2.6. *Inertial Data Block

TAOS maintains an inertial platform coordinate system that is fixed in inertial space. The Euler angles *yawi*, *pitchi*, and *rolli* are referenced to this system and the position, velocity, and acceleration vectors can be output in this system (Section 2.1.10).

The inertial platform coordinate system is initially aligned with the local geodetic horizon coordinate system and located at the trajectory initial position. It is aligned such that the x axis points north, the y axis points east, and the z axis points down (towards the earth's center). This alignment and position can be changed at the beginning of any segment with the **inertial* data block.

The coordinate systems that the inertial system can be aligned with are given in Table 4–6. The default coordinate system is *geodetic*. All of the coordinate systems are defined in Section 2.1.

Table 4–6. Inertial Platform Alignment Coordinate Systems.

Name	Description
body	Body axis system
ecfc	Earth–centered–fixed coordinate system (specify east and down vectors)
geocentric	Local geocentric horizon system (specify longitude/latitude position)
geodetic	Local geodetic horizon system (specify longitude/latitude position)
velocity	x–axis parallel to velocity vector and z–axis coplanar with geodetic z–axis
wind	Wind axis system

Geocentric and Geodetic Alignment

The data block

```
*inertial platform alignment   geocentric
```

aligns the inertial coordinate system with the local geocentric horizon system and locates the system at the vehicle's center of mass. The keywords *platform* and *alignment* are both optional.

If aligning with the geodetic or geocentric coordinate systems, normally the current vehicle latitude, longitude, altitude, and time are used for alignment. However, alignment can occur at another location by specifying a latitude, longitude, and altitude or it can occur at a different time by specifying a time. For example,

```
*inertial platform  lat=28.8  long=-81.5  alt=550  time=100
```

aligns the coordinate system with the geodetic horizon system located at a geodetic latitude of 28.8° N, a longitude of 81.5°W, an altitude of 550 ft, and at a time of 100 seconds. The time is the same used for computing all trajectories; alignment time is required to account for the effects of earth rotation on the inertial system.

The variables *long*, *lat*, and *alt* are used to change the position of a platform aligned to local geodetic horizon coordinates, and the variables *long*, *lat*, and *rcm* are used to change the position of a platform aligned to local geocentric coordinates. For geodetic alignment, the *alt* variable is optional and has a default value of zero (sea level).

Body, Wind, and Velocity Alignment

The inertial coordinate system can be aligned with the body axis system with

```
*inertial platform  body
```

The body attitude information is not available at the beginning of the first segment of a trajectory because the guidance scheme that determines these axes has not been executed. Therefore, the inertial coordinate system should not be aligned with either the body or wind coordinate system on the first segment of a trajectory.

An initial dummy segment with a final condition of *tseg>0* can be used to calculate the body attitude from the trajectory initial conditions. Then, the inertial platform can be aligned with the body, wind, or velocity coordinate system at the start of the second segment.

The inertial platform is located at the vehicle's center of mass when aligning with the body, wind, or velocity coordinate systems. An alignment time can still be given, which aligns the platform as though the current position and velocity existed at the specified time.

ECFC Alignment

If the platform is aligned with the earth–centered, earth–fixed coordinate system (ECFC), the axes are parallel to the ECFC unit vectors and the platform is located at the vehicle's center of mass. The orientation of the unit vectors defining the inertial platform axes can be changed by entering east and down vectors. Only two unit vectors need to be defined; the remaining vector is calculated to give a right–handed coordinate system. The x, y, and z components of these vectors are given in the ECFC system with the variables *eastx*, *easty*, *eastz*, *downx*, *downy*, and *downz*. If only one of the vectors is given, the default value is used for the other one. For example, the data block

```
*inertial  ecfc  eastx=0.7071  easty=0.7071  eastz=0.0
                 downx=0.0     downy=0.0     downz=-1
```

twists the east and north inertial vectors 45°.

4.2.7. *Integ Data Block

Trajectories are computed by integrating the equations of motion with a 4th–order fixed step size Runge–Kutta numerical integration method. The integration step size, which controls the accuracy of the calculations, is fixed for each segment of the trajectory. It needs to be small enough to capture the effects of changing aerodynamic and propulsive forces, but large enough so that computation time is not excessive. The default step size of 0.10 seconds can be changed with the **integ* data block. For example,

```
*integ  dt=0.25
```

says to use a 0.25 second integration step size. Typical step sizes range from 0.01 to 0.5 seconds. The minimum integration step size is 1×10^{-6} seconds.

Output variables are printed at the beginning and end of each segment and at even increments of the print interval within the segment. Besides controlling the printout, the print interval also controls the amount of memory required for the trajectory because all trajectory output variables are saved in memory at the printout times. Thus, small print intervals increase memory requirements and vice versa. The print interval, *dtprnt*, defaults to 1.0 second. The data block

```
*integ  dtprnt=0.5   dt=0.05
```

changes the print interval to 0.5 seconds and the integration step size to 0.05 seconds.

Guidance Time Constant

Some guidance rules, such as those in Table 4–4, specify the control variables indirectly with a desired flight condition. If the actual flight conditions are different from the desired conditions, the control variables are adjusted such that the trajectory corrects to the desired conditions (Section 2.5.1). The rate of this correction is controlled by the guidance time constant, *dtguid*.

If *dtguid* is small, large corrections are made in an effort to quickly achieve the desired flight conditions. If *dtguid* is large, small corrections are made to gradually achieve the desired conditions. So *dtguid* acts somewhat like a time constant with small values giving oscillations that may or may not damp to the desired conditions, and large values giving exponential convergence to the desired flight conditions.

In the following data block, *dtguid* has been changed to 5 seconds:

```
*integ  dtprnt=1.0  dt=0.20  dtguid=5.0
```

A value of 5 seconds is considered large. It causes small corrections to be made to the flight path in order to meet the requested flight conditions. These corrections may be too small to achieve the desired conditions. Experimentation is required to get the desired results, and the default value of 1 second works well for most guidance rules.

4.2.8. *Limits Data Block

TAOS, being a 3–DOF simulation, assumes that a control system is available that can achieve the body attitudes requested in the guidance scheme. Usually the control system can only maintain body attitudes for certain flight conditions, and the body attitude angles are restricted as well. The **limits* data block provides a way to keep the trajectory within these bounds.

Limits can be set on any of the guidance variables given in Tables 4–2 and 4–4, except for *intercept, prop, downria, upria,* and *l/d_max.* For example,

```
*limits  alpha<20  alpha>-10
```

puts a 20° upper boundary and a –10° lower boundary on angle of attack. Greater–than and less–than signs are used to designate whether the limit is an upper or lower bound.

The limiting variables do not have to be the same as those used for the guidance scheme; limits can be imposed on any of the guidance variables. Any number of limits can be specified. For example,

```
*limits  bankgd>-60  bankgd<60  alpha>-15  alpha<15.0
           beta>-15   beta<15
```

puts limits on three of the control variables: *alpha, beta,* and *bankgd.*

Flight condition guidance (Table 4–4) requires iterative searches to solve for the control variables. Limits on the body attitude angles help control convergence during these searches.

4.2.9. *Prop Data Block

Each **prop* data block defines a thrust force and its corresponding fuel flow or mass flow rate. If more than one **prop* data block is given in a segment, the propulsive forces and mass flow rates from each data block are added together to get the total propulsive force and mass flow rate. If no **prop* data blocks are given in a segment, thrust and mass flow are set to zero.

A propulsive force vector is defined by a magnitude and a direction. The direction is given by two angles, *ep1* and *ep2*, defined in Figure 4–3 as ε_1 and ε_2. If both thrust vector angles are zero, the default, the thrust force is aligned with the body x axis.

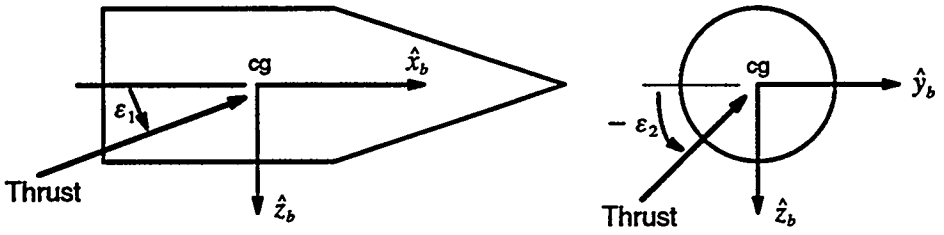

Figure 4–3. Thrust Vector Angles.

Values for the thrust force, the thrust vector angles, and the mass flow rate can be constants or they can be obtained from tables. The following data block sets the thrust and mass flow rate to constant values:

```
*prop    thrust=250      mdot=120.0
         thr_units=kn    mdt_units=kg/sec
```

The variables *thrust* and *mdot* give values for the thrust force and the mass flow rate. The variables *thr_units* and *mdt_units* give the units for the thrust and mass flow rate. If no value or table name is provided for a variable, it is set to zero. So in the example shown above, the thrust vector angles, *ep1* and *ep2*, are both zero.

The default unit for thrust is *lb* and for mass flow rate is *lb/sec*; other allowable units are given in Table 4–7.

Table 4–7. Thrust and Mass Flow Rate Units.

Thrust Units	Mass Flow Units	Mass Flow Units	Mass Flow Units
lb	lb/sec	lb/min	lb/hr
n (Newtons)	slugs/sec	slugs/min	slugs/hr
kn (Kilonewtons)	g/sec	g/min	g/hr
	kg/sec	kg/min	kg/hr

Another example,

```
*prop   thrust=(thr_recruits)   mdot(mdt_recruits)
        nrecruits=2
```

illustrates the use of tables to compute the thrust and mass flow rate. In this case the *thr_recruits* table is used for thrust and the *mdt_recruits* table is used for mass flow rate.

The tables in this example require a value for the user–defined variable, *nrecruits*, representing the number of motors. Like other data blocks that allow tables to be specified, values can be given for user–defined variables. Once a value has been given for a user–defined variable, it remains set at that value unless changed in a subsequent data block. This means that values for user–defined variables do not have to be repeated within each segment if they do not change.

Units for thrust and mass flow rate can be specified in the tables, so they are not given here. If units are given both places, the units in the **prop* data block override those given in the tables.

Some jet engines have thrust vector control, which can be used to improve takeoff, landing, and turning performance. The data block

```
*prop   thrust=(thr_f100x)   mdot=(mdt_f100x)
        ep1=10.0
```

sets the thrust vector angle to 10°. The thrust vector angles in this version of TAOS can be constant or computed with a table; they must be known before the flight path is computed. In future versions of TAOS, they will be control variables so they can be varied to meet the desired flight conditions.

4.2.10. *Rail Data Block

Rail launches and sled tests are special cases because the vehicle is constrained by rails to move only in one direction. For rail launches and sled tests, the **rail* data block is used instead of the **fly* data blocks to determine the body attitude and the trajectory. The direction of the rail or sled is obtained from the trajectory initial conditions or from the end of the previous segment. The motion of the vehicle is constrained in this direction for the duration of the segment (Section 2.2.2).

A rail launch is requested with the data block

```
*rail launch  cfstat=0.15   cfslid=0.01
```

where *cfstat* is the static coefficient of friction and *cfslid* is the sliding or kinematic coefficient of friction. The static coefficient of friction is used when the velocity is less than 0.001 ft/sec; otherwise, the sliding coefficient of friction is used. Because of this, two segments are often used to simulate rail launches: the first segment is from a velocity of 0 ft/sec to a velocity of 0.001 ft/sec, and the second segment is for velocities greater than 0.001 ft/sec. This simulates the amount of time required for the thrust forces to overcome the static friction force and move the vehicle.

For a rail launch, TAOS assumes the missile is attached to the rail such that it cannot move backwards, so negative accelerations in the direction of the rail are ignored. The path length, *plength*, can be used as a segment final condition to represent the rail length (Section 4.2.12).

A sled test can be simulated with

```
*rail sled  cfstat=0.17  cfslid=0.02
```

where again *cfstat* and *cfslid* are the coefficients of friction. Negative accelerations are allowed for the sled test because the vehicle accelerates up to speed and then slows to a stop.

4.2.11. *Reset Data Block

The ***reset*** data block works the same way as the ****increment*** data block (Section 4.2.5) except that variables are set to the input value instead of incremented by the input value. The rules for resetting variables are the same as those for incrementing variables. Variables that can be reset are given in Table 4–5.

Typical uses of the ***reset*** data block include resetting the variable *tmark* with

```
*reset  tmark=0
```

which is used to mark an event during the trajectory. For example, rocket motor thrust in the propulsion tables is often made a function of *tmark* rather than *time* or *tseg*. This allows the motor burn time to extend across several segments and allows motor ignition at any point in the trajectory. The value of *tmark* just needs to be reset to zero at motor ignition. Other similar variables are *grmark*, *plmark*, and *fuel*.

Another common use of the ***reset*** data block is to set the weight to a known value after missile staging, for example,

```
*reset  wt=1100
```

sets the weight to 1100 lb regardless of what has happened previously in the trajectory.

4.2.12. *When Data Block

Segment initial conditions are obtained from the trajectory initial conditions or from the final conditions of the previous segment. Segments are flown according to the guidance rules given in the **fly* or **rail* data blocks and they continue until a final condition is met. At that point, the segment ends, and if there is another segment in the trajectory, it is started.

The **when* data block is used to provide a segment final condition. It also tells TAOS what to do next, that is, whether to transfer to another segment or to end the trajectory. Each segment must have at least one **when* data block giving a final condition; otherwise, the segment never terminates.

An example of a segment final condition is

```
*when  alt<200000  goto 4
```

which reads "end the current segment when altitude becomes less than 200,000 ft and then start executing segment number 4." The variable name, in this case *alt*, can be any valid output variable name including user–defined output variables. A list of output variables is given in the appendix.

Final conditions can also compare two output variables, for example,

```
*when  alpha = alpha_l/d  goto 2
```

ends the segment when the angle of attack equals the angle of attack for maximum lift–to–drag ratio.

Values from other Trajectories

Output variables from other trajectories can be referenced by using subscripts, for example,

```
*when  mach[3] > 4  goto 21
```

says to end the segment when the Mach number on trajectory 3 becomes greater than 4.0. Subscripts are always enclosed in square brackets.

When final conditions are used to compare two variables, subscripts can be used on both variables to compare values from different trajectories. For example,

```
*when  east[2] = east[1]  goto 7
```

ends the segment when the two trajectories have the same east position.

Final Condition Relationships

The relationship in a **when* data block can be greater than, less than, or equal to. Since all values are continuous real numbers, using the greater than or less than often

has the same effect as the equal sign because the segment will end when the variable is equal to the value. The difference occurs at the beginning of a segment.

When a segment starts executing, it checks the final conditions for immediate termination. If any of the final conditions are satisfied initially, they cause an immediate transfer to another segment. As a result the final condition *mach>3* is much different from *mach=3*. A *mach>3* final condition terminates the segment immediately if the Mach number is greater than 3.0, whereas the *mach=3* final condition only terminates the segment immediately if Mach is exactly equal to 3.0. In practice use of greater than and less than is more common than the equal sign.

Ending Trajectories

So far, all of the examples have shown segment transfers using the *goto* keyword. When the end of a trajectory is reached, there is no segment to transfer to, so the *stop* keyword is used as follows:

```
*when  time=65  stop
```

This data block ends the trajectory when the time is equal to 65 seconds. Calculations proceed until a segment final condition with *stop* is encountered.

Multiple Final Conditions

Segments can have any number of final conditions. The first condition encountered, while computing the trajectory, is executed. This allows complex branching in the trajectory depending on what happens as shown in the following example:

```
*when  tmark>25  goto 6
*when  alphat>5  goto 11
```

If the "mark" time becomes equal to 25 seconds before the total angle of attack reaches 5°, then segment 6 is executed next; otherwise, segment 11 is executed next.

If two final conditions are immediately satisfied at the beginning of a segment, the first one listed is executed.

It is good practice to have at least one final condition on a time variable. For example,

```
*when  gamgd<0  goto 4
*when  tseg>1000  goto 10
```

normally ends the segment when the flight path angle becomes zero. But if there is a problem and the flight path angle never reaches zero, the segment still ends after 1000 seconds. This prevents infinite loops.

4.3. Trajectory Data Blocks

Trajectory data blocks contain information associated with the entire trajectory. Segment data blocks, discussed in Section 4.2, are one of the trajectory data blocks. Other trajectory data blocks are shown in Table 4–8.

Table 4–8. Trajectory Data Blocks.

Data Block	Description
*define	User–defined output variables
*dwn/crs	Downrange/crossrange reference
*file	Create output file
*iip	Initial impact point variables
*initial	Initial conditions
*print	Printout variables
*segment	Segment definition (Section 4.2)
*tangent	Tangent plane coordinate system

Trajectories begin with the **trajectory* keyword, a trajectory number, a trajectory name, the keywords *start on*, and a segment number. For example, the data block

```
*trajectory 2 target start on 31
```

begins trajectory number 2 for a vehicle called *target* on segment number 31. All of the items following the **trajectory* keyword are required.

Trajectory information continues until another **trajectory* keyword is found or until a unique problem data block keyword is found. Some keywords, such as **file* and **print*, are used both inside and outside of a trajectory definition, so they do not terminate a set of trajectory data. Only those keywords that are unique to a problem file, such as **trajectory*, **search*, and **atmos*, end a set of trajectory data (Section 4.4).

Trajectory definitions must contain an **initial* data block giving the initial conditions and at least one **segment* data block. Usually they contain at least one **print* data block so that trajectory results are written to a printout file.

The following gives an outline of a trajectory definition:

```
*trajectory 1 missile start on 1
   *initial ...
   *print ...
   *segment  1  First segment definition
      ...
   *segment  2  Second segment definition
      ...
*trajectory 2 target ...
```

Indentation has been used to show the organization of the trajectory information. Trajectory 1 extends from the first **trajectory* keyword to the second **trajectory* key-

word. Between these keywords, initial conditions are given in the **initial* data block, printout variables are listed in the **print* data block, and segments are defined in the **segment* data blocks. Each **segment* data block consists of additional data blocks, as shown in Section 4.2, that tell TAOS how to compute each segment.

The following sections give the details of each trajectory data block.

4.3.1. *Define Data Block

The standard trajectory variables computed in TAOS are listed in the appendix. These are always computed and available for printout and plotting; however, some problems require other variables that are not among the standard output variables. These variables are different depending on the type of problem.

In an effort to solve this problem, TAOS provides user–defined output variables. As long as the variables of interest can be calculated from the standard variables with one or more equations, that is, they are functions of the standard variables, then they can be computed as user–defined variables.

User–defined variables are given a name which must be different from the standard output variables. This is followed by one or more equations that show how to compute the new variable. The user–defined variable can be computed directly from the equations, or the equations can define the variable's derivative, which can be integrated along with the other equations of motion.

The equations defining how to calculate a user–defined variable are given in a simplified C language. This is similar to Fortran so users familiar with programming in either of these languages should not have trouble entering the equations. The examples below help clarify how equations are input.

The first example,

```
*define  alt_km
   alt_km = alt / 3280.84;
```

computes altitude in kilometers rather than feet. The name of the new variable is *alt_km*. It is given following the **define* keyword, and it must also appear on the left side of an equation somewhere in the definition. In this example, there is only one simple equation, so *alt_km* must be the variable on the left side of the equal sign. The values and variables on the right side of the equal sign must all be standard output variables, previously input user–defined variables, temporary variables, or constants. In this case, *alt* is a standard output variable and *3280.84* is a constant. The equation ends with a required semicolon.

Temporary Variables

In the next example, the stagnation point heat transfer rate for a blunt body of revolution in hypersonic flow is approximated with

$$q_s = \frac{17{,}600}{\sqrt{R_n}} \sqrt{\frac{\varrho}{\varrho_0}} \left[\frac{\| \vec{V}_{\oplus} \|}{\| \vec{V}_c \|} \right]^{3.15} \text{Btu/ft}^2\text{–sec} \qquad (4\text{–}1)$$

where R_n is the nose radius, ϱ and ϱ_0 are the ambient and sea–level density, $\| \vec{V}_{\oplus} \|$ is the flight velocity, and $\| \vec{V}_c \|$ is satellite velocity (usually 26,000 ft/sec).

The following data block defines a user–defined variable, *stag_heat*, with this equation:

```
*define  stag_heat
   rnose = 1.0 / 12.0;
   stag_heat  = (17600.0/sqrt(rnose))*sqrt(rho/0.0023769)
                *(vel/26000.0)^3.15;
```

A nose radius of 1 inch is converted to feet and saved in a temporary variable called *rnose*. Temporary variables are only used within the user–defined variable calculation. If a variable name is not a standard output variable or a previously input user–defined variable, it is assumed to be a temporary variable. Temporary variables must be defined before they can be used.

The only thing unusual about the equation for stagnation heating rate is the use of ^ for raising a value to a power. This symbol is used in TAOS rather than the *pow* function in C or the ** operator in Fortran.

The equation for stagnation heating rate requires two lines. Statements continue until a semicolon is reached; the semicolon is required at the end of each statement just like in the C language.

Math Operations and Functions

Table 4–9 contains a list of operators and functions that can be used in equations for user–defined variables. All of the standard mathematical operators are available, and they have the same precedence as in C and Fortran. Many standard functions are also provided; again these are defined in TAOS as they are in the C language.

Table 4–9. User–Defined Variable Math Operations and Functions.

Operator	Description	Function	Description
+	Add	sin(x)	Sine of x (deg)
–	Subtract	cos(x)	Cosine of x (deg)
*	Multiply	tan(x)	Tangent of x (deg)
/	Divide	asin(x)	$\sin^{-1}$ of x where x ∈ [–1,1]
^	Exponentiation	acos(x)	$\cos^{-1}$ of x where x ∈ [–1,1]
>	Greater than	atan(x)	$\tan^{-1}$ of x in range [–π/2,π/2]
<	Less than	atan2(x)	$\tan^{-1}$ of x in range [–π,π]
>=	Greater than or equal to	sinh(x)	Hyperbolic sine of x (deg)
<=	Less than or equal to	cosh(x)	Hyperbolic cosine of x (deg)
==	Equal to	tanh(x)	Hyperbolic tangent of x (deg)
!=	Not equal to	exp(x)	Exponential function e^x
&&	Logical and	log(x)	Natural logarithm of x
||	Logical or	log10(x)	Base 10 logarithm of x

		sqrt(x)	Square root of x
		ceil(x)	Smallest integer not less than x
		floor(x)	Largest integer not greater than x
		abs(x)	Absolute value of x
		table(id)	Lookup value from a table where *id* is the table name

The *table* function is special in that it is used to retrieve a value from a table in the table file. The function parameter, *id*, is the table identification name (Section 3.4), and the table must be of type *output*. For example,

```
*define  special
   special = (rho/0.0023769)*table(special_tbl);
```

references the output table *special_tbl*. A possible use of output tables is for aerodynamic heating analysis. Output tables containing heat transfer rates could be interpolated to get the heat transfer rates experienced in a trajectory, and then this value could be limited by a trajectory optimization constraint.

If–then statements can be used to control how a user–defined variable is calculated. For example,

```
*define  nx_max
   nx_max = max(nx);
   if (nx_max < 0)  nx_max = 0;
```

calculates the maximum axial g loading. The *if* statement, written in standard C syntax, limits *nx_max* to positive values. The relationship of the *if* statement is always enclosed in parentheses. If the relationship evaluates as true, then the statement following the relationship is executed.

More than one statement can be executed as a result of the *if* statement by enclosing them in curly brackets as follows:

```
*define  new_var
   new_var = 0;
   if (time > 20)  {
      ratio = pres / 2116.2;
      new_var = ratio * table(tbl_id);
   }
```

The C language uses curly brackets to group together parts of the *if–then* statement; the keywords *then* and *endif* are not used as in Fortran.

Curly brackets are also required for an *if–then–else* statement as shown in the following example:

```
*define  var
   if (mach > 1.0)  {
      x = 1 + mach*mach;
      var = sqrt(x) * table(outx);
```

```
} else {
   var = table(outy);
}
```

The current version of TAOS only supports assignment statements and *if–then–else* statements. *While* loops, *for* loops, and other C language control statements cannot be used.

Integral Variables

A common use of integral user–defined variables is in a flight path optimization constraint. Constraints at specific points along the flight path are given directly in the **optimize* data block (Section 4.4.6). However, an integral variable is necessary to keep the entire flight path within a constraint.

For example, to keep the flight path within a minimum dynamic pressure of 100 lb_f/ft^2, an integral variable can be defined as

$$\text{if q} > 100 \text{ or vel} < 1000 \qquad q_{min} = \int 0.0\ dt \qquad (4\text{–}2)$$

$$\text{otherwise} \qquad q_{min} = \int (100.0 - q)^2 dt \qquad (4\text{–}3)$$

Optimization constraints work best if the constraint variable is set up such that it is less than zero if the constraint is satisfied and greater than zero if the constraint is violated. In this case, if the dynamic pressure is greater than the minimum value while the vehicle is flying at a high speed, the derivative is set to zero. However, if the dynamic pressure drops below 100 lb_f/ft^2, then the derivative takes on a positive value.

This can be converted to a user–defined variable definition as follows:

```
*define  integral  qmin=-0.10
   if (dynprs > 100 || vel < 1000)  {
      qmin = 0.0;
   }  else  {
      qmin = (100.0 - dynprs)^2;
   }
```

The *integral* keyword says that the variable *qmin* is defined as a derivative that must be integrated with respect to time along the flight path. Its initial value is set to –0.10, a negative value, so initially the constraint is satisfied. The *if* statement is written using C syntax. It translates to "*if the dynamic pressure is greater than 100 or the velocity is less than 1000, set qmin equal to zero; otherwise, set qmin equal to 100 minus the dynamic pressure squared.*"

4.3.2. *Dwn/crs Data Block

East, north, downrange, and crossrange are defined from a reference point, given by a longitude and geodetic latitude, as shown in Figure 4–4. The trajectory is projected to the earth's surface, and the distances are measured along this curve (Section 2.4.1). East is the component of distance traveled along a constant latitude and north is the component of distance traveled along a constant longitude. Downrange is the component of distance traveled along a reference azimuth or heading, and crossrange is the component of distance traveled perpendicular to this heading.

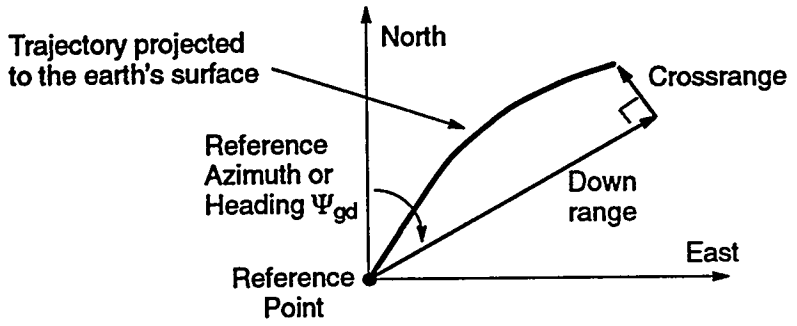

Figure 4–4. Downrange and Crossrange Definition.

All of these distances use the same reference point. The reference heading is only used by the downrange and crossrange calculations. The reference longitude, latitude, and heading default to the beginning of the trajectory given by the initial conditions.

If a different reference point is required or if multiple trajectories need to use the same reference point, then these values can be given in the **dwn/crs* data block. For example,

```
*dwn/crs  latgd=10  long=95  azm=45
```

places the reference point at a latitude of 10°N and a longitude of 95°E. The reference heading or azimuth is 45° from north or northeast.

4.3.3. *File Data Block

The **file* data block is used to create files containing columns of trajectory information. Trajectory output variables are written to the file as columns of numbers, and each column is separated by at least one space. Column headings are placed at the beginning of the file, but, thereafter, the columns are continuous. This format is ideal for importing to other software, such as a plotting program or a spreadsheet. There are no page headings or other information that must be removed; at most, the column headings must be removed.

For example,

```
*file  traj.dat
   time alt long latgd range vel
```

writes the trajectory variables to the file *traj.dat*. Figure 4–5 shows the first part of this file.

Time	Alt	Long	Latgd	Range	Vel
0.000	300000.0	20.00000	0.00000	0.000	18000.00
0.500	294835.2	20.01991	0.00000	1.196	18008.93
1.000	289665.1	20.03982	0.00000	2.394	18017.86
1.500	284489.8	20.05975	0.00000	3.591	18026.78
2.000	279309.3	20.07968	0.00000	4.789	18035.69
2.500	274123.5	20.09962	0.00000	5.988	18044.58
3.000	268932.4	20.11958	0.00000	7.188	18053.45
3.500	263736.2	20.13954	0.00000	8.388	18062.31
4.000	258534.7	20.15951	0.00000	9.588	18071.13
4.500	253328.0	20.17950	0.00000	10.789	18079.91
5.000	248116.2	20.19949	0.00000	11.991	18088.65
5.500	242899.1	20.21949	0.00000	13.193	18097.33
6.000	237676.9	20.23950	0.00000	14.396	18105.96
6.500	232449.5	20.25952	0.00000	15.599	18114.52

Figure 4–5. Trajectory Output File Example.

The filename must be valid. If the file already exists, it is overwritten with no warning and the old data is lost. This is normally satisfactory because the file will be updated to contain information from the most recent run.

Any standard output variable or user–defined output variable can be written to the file. User–defined variables referenced in a **file* data block must be defined prior to the reference. The units and printout format are controlled with the **units/fmt* data block (Section 4.4.13).

Each output line is limited to 400 characters, so this limits the number of output variables that can be listed to about 30. Usually it is best to limit the number of output variables to 12 or fewer because some screen editors and printers can only handle about 130 characters per line.

Radar and relative vehicle output variables, those starting with *rad* or *rel*, must contain a subscript enclosed in square brackets. For example, the variable *radrng[2]* is used to print the radar range of the current trajectory from radar station number 2,

and the variable *relvel[4]* is used to print the closing velocity of trajectory number 4 relative to the current trajectory.

Any number of **file* data blocks can be included in a trajectory definition. An output file is created for each **file* data block.

4.3.4. *Iip Data Block

The initial impact point (IIP) is obtained by integrating a ballistic trajectory from the current state of the vehicle to a final altitude with no propulsive forces (Section 2.4.4). A constant drag coefficient aerodynamic model, given by a ballistic coefficient, is used to simplify the analysis. This simulates the impact point if the propulsive system fails and is used for range safety analysis.

The initial impact point is calculated if one of the associated output variables is referenced in the problem (*iip_lat*, *iip_long*, *iip_time*, *iip_rng*, or *iip_azm*). These calculations require a ballistic coefficient and a final impact altitude which are given in the **iip* data block. For example,

```
*iip  iip_beta=550  iip_alt=1000
```

sets the ballistic coefficient to 550 lb_f/ft^2 and the impact altitude to 1000 ft for the IIP calculations. The default ballistic coefficient and impact altitude are both zero. For zero drag, the ballistic coefficient should be set to zero instead of a large number. The ballistic coefficient can be reset anytime during the trajectory with an **increment* (Section 4.2.5) or **reset* data block (Section 4.2.11).

The output variables *iip_rng* and *iip_azm* give the range and azimuth of the initial impact point relative to the origin of the tangent plane coordinate system. The range is the distance from the initial impact point to the origin of the tangent plane system. The azimuth is the forward azimuth from the tangent plane origin to the initial impact point measured from north. If these ouput variables are required, a **tangent* data block should be included to define the origin of this coordinate system (Section 4.3.7).

4.3.5. *Initial Data Block

The initial conditions or starting point of a trajectory are given in the **initial* data block. Initial conditions define the initial state of the vehicle, that is, its position and velocity vectors and its mass. They also define a starting time, range, and path length.

Initial conditions can be provided directly by giving specific values for the position, velocity, and mass, or they can be provided indirectly by using values from a different trajectory. The latter method is used when a vehicle releases or drops objects or when a missile stages. This way trajectories for all objects or stages can be computed at the same time.

Specific Initial Conditions

When entering specific initial conditions, the **initial* data block consists of a keyword designating a coordinate system followed by a set of variables and values defining the vehicle's initial state. The position and velocity variables must be consistent with the coordinate system. Table 4–10 contains sets of position variables for each coordinate system, and Table 4–11 contains sets of velocity variables for each system.

Table 4–10. Initial Position Variables.

Coordinate System	Variable	Description
Geodetic	alt	Altitude
	long	Longitude
	lat	Latitude
Geocentric	rcm	Distance to earth center
	long	Longitude
	lat	Latitude
ECFC or ECIC	x	Position x component
	y	Position y component
	z	Position z component

The geodetic coordinate system is the default for initial conditions. For example,

```
*initial
   alt=30000    long=0     lat=45      # Position vector variables
   vel=750      gamma=0    psi=90      # Velocity vector variables
   wt=2000      time=0                 # Other variables
```

defaults to the geodetic coordinate system. The three position variables are from Table 4–10 for the geodetic coordinate system, and the three velocity variables are from Table 4–11.

Table 4–11. Initial Velocity Variables.

Coordinate System	Variable	Description
Geodetic or Geocentric	vel	Velocity
	gamma	Flight path angle
	psi	Heading angle
ECFC or ECIC	xdt	Velocity x component
	ydt	Velocity y component
	zdt	Velocity z component

Table 4–12. Other Initial Condition Variables.

Variable	Description
mach	Mach number (use instead of vel)
wt	Weight
mass	Mass (use instead of wt)
time	Time
range	Ground range
path	Path length
t_0	Time at which omega_0 applies
omega_0	Angle between ECFC and ECIC at t_0

The units of these variables is the same as for the standard output variables given in the appendix. Units can be changed with the **units/fmt* data block (Section 4.4.13).

The other variables initializing weight and time are from Table 4–12. The only required variable from this table is *wt* (or *mass*). The variable *mass* can be substituted for *wt*; one or the other can be input, but not both. Similarly, the variable *mach* is special in that it can be substituted for *vel* when entering the initial velocity. Mach number or velocity can be entered, but not both.

The remaining variables in Table 4–12 are optional and default to zero. The variables *t_0* and *omega_0* control the angular difference between the ECFC and ECIC coordinate systems at a given time. At a time equal to *t_0*, the ECIC coordinate system is rotated an angle *omega_0* from the ECFC system. Thus, at time *t*, the ECIC coordinate system is rotated at an angle Ω from the ECFC system given by

$$\Omega = \Omega_0 + \| {}_I\vec{\omega}_{\oplus} \| \cdot (t - t_0) \qquad (4\text{–}4)$$

where the earth's rotation rate $\| {}_I\vec{\omega}_{\oplus} \|$ is given in the **earth* data block (Section 4.4.3).

The following data block illustrates initial conditions given in the ECFC coordinate system:

```
*initial  ecfc
   x = 19099345.2   y = 9974312.7   z = 6489105.4    # Position
   xdt = -173.973   ydt = 10327.52  zdt = 26425.9    # Velocity
   time = 769.35    wt = 1053.9                      # Time and Wt
   t_epoch = -1045.2     omega_0 = 25.3              # ECFC to ECIC
```

This trajectory starts at time 769.35 seconds. However, the angular difference between the ECFC and ECIC coordinate systems is not known at this time, so it is input at the known time of –1045.2 seconds or 1814.55 seconds before the trajectory starts. The times of all trajectories are relative to a $t = 0$ mission time.

Initial Conditions from a Trajectory

Initial conditions for a trajectory can be obtained from another trajectory, so that multiple trajectories can branch from a single trajectory. For example, in trajectory 1 an aircraft releases a weapon at the beginning of segment 6. The weapon's initial conditions are the same as the aircraft's initial conditions at the beginning of segment 6, that is,

```
*initial  from  segment 6, trajectory 1
```

The *from* keyword says to retrieve initial conditions from another trajectory. Initial conditions always are taken from the beginning of a segment; in this case, from the beginning of segment 6 in trajectory 1.

When initial conditions are obtained from another trajectory, the variables *trajectory* and *segment* are input as shown above. Variables from Tables 4–10 and 4–11 defining the position and velocity should not be given. Values for the variables *t_0* and *omega_0* from Table 4–12 can be given following the segment and trajectory numbers. These variables are often set to the same values in all trajectories to ensure the same ECIC coordinate system.

After a trajectory has been initialized from another, the **increment* (Section 4.2.5) and **reset* (Section 4.2.11) data blocks can be used in the first segment to adjust its initial state. In general, the deployed object's weight must be set because its weight is different from the original vehicle. In addition, its position is usually adjusted because its center of mass is not at exactly the same place as the original vehicle's center of mass.

4.3.6. *Print Data Block

The **print* data block is similar to the **file* data block, described in Section 4.3.3. However, it writes trajectory output variables to a standard output file formatted for printing rather than to a user–specified file. The standard output file has the same filename as the problem file, but it is of type *.out* instead of *.prb*. It has page breaks and headings formatted for a standard printer with 132 characters per line and 60 lines per page.

A table of trajectory data is written for each **print* data block. Any number of **print* data blocks can be included within a trajectory, and a maximum of 10 trajectory output variables can be given per **print* data block. For example,

```
*print  time  alt  range  mach  gamgd  dynprs
```

creates the output shown in Figure 4–6. The output begins with a page heading, a title, and column headings that are repeated on every page. This is followed by the trajectory variables. The output format of each variable is controlled with the **units/fmt* data block (Section 4.4.13).

```
Sandia National                          Trajectory Analysis & Optimization Software
Laboratories                                        (TAOS - Version 96.0)
-------------------------------------------------------------------------------

     Example of a ballistic RV trajectory
     Problem (marv)  /  1976 US Standard Atmosphere  /  Non-rotating WGS-84 Earth
     Trajectory for vehicle:  rv
     gama = -35
      Time          Alt         Range          Mach         Gamgd        Dynprs
     0.000     300000.0         0.000       19.9642       -35.000         0.829
     0.500     294835.2         1.196       19.9741       -35.021         1.100
     1.000     289665.1         2.394       19.9840       -35.042         1.457
     1.500     284489.8         3.591       19.9939       -35.063         1.932
     2.000     279309.3         4.789       19.9659       -35.083         2.537
     2.500     274123.5         5.988       19.8147       -35.104         3.295
     3.000     268932.4         7.188       19.6670       -35.125         4.263
     3.500     263736.2         8.388       19.5225       -35.146         5.495
     4.000     258534.7         9.588       19.3812       -35.167         7.059
     4.500     253328.0        10.789       19.2429       -35.187         9.036
     5.000     248116.2        11.991       19.1075       -35.208        11.528
     5.500     242899.1        13.193       18.9750       -35.229        14.660
     6.000     237676.9        14.396       18.8451       -35.249        18.584
```

Figure 4–6. Trajectory Printout Example.

A line of trajectory data is printed at the beginning and end of each segment as well as at each print interval within the segment. A line is skipped between segments so they can be easily identified.

Radar and relative vehicle output variables, those starting with *rad* or *rel*, must contain a subscript enclosed in square brackets. For example, the variable *radrng[2]* is used to print the radar range of the current trajectory from radar station number 2, and the variable *relvel[4]* is used to print the closing velocity of trajectory number 4 relative to the current trajectory.

4.3.7. *Tangent Data Block

The origin of the tangent plane coordinate system is fixed to the earth and its location is given in geodetic coordinates. The x and y axes are tangent to the earth's surface and the z axis points upward. The x axis is oriented along a heading or azimuth measured clockwise from north (Section 2.1.11).

Output variables, listed in the appendix, are calculated giving the position, velocity, and acceleration in tangent plane coordinates. In addition, the initial impact point range and azimuth are measured from the origin of the tangent plane coordinate system.

The **tangent* data block is used to provide the location and orientation of the tangent plane coordinate system. For example,

```
*tangent  latgd=21.982  long=-159.759  alt=87.2  azm=140
```

positions the origin at a latitude of 21.982°N, a longitude of 159.759°W, and an altitude of 87.2 ft above sea level. The coordinate system is oriented so the x axis points along a 140° azimuth or heading (approximately southeast).

The **tangent* data block is optional. The origin of the tangent plane coordinate system defaults to a point on the earth's surface directly below the vehicle's initial position. In its default orientation, the x axis points north.

4.4. Problem Data Blocks

The problem file contains one or more problems or cases, which in turn contain one or more trajectory definitions. Each trajectory definition begins with the **trajectory* data block (Section 4.3) and ends with the next nontrajectory and nonsegment keyword. Trajectory definitions contain one or more segments forming a hierarchical structure as shown previously in Figure 4–2.

It is important to group the data blocks according to their purpose. All data blocks defining a segment must be grouped together, and all data blocks defining a trajectory must be grouped together. The remaining data blocks do not have to be grouped together, but the problem file is much easier to understand if they are grouped than if they are randomly scattered through the file. Grouping the remaining data blocks also tends to reduce errors.

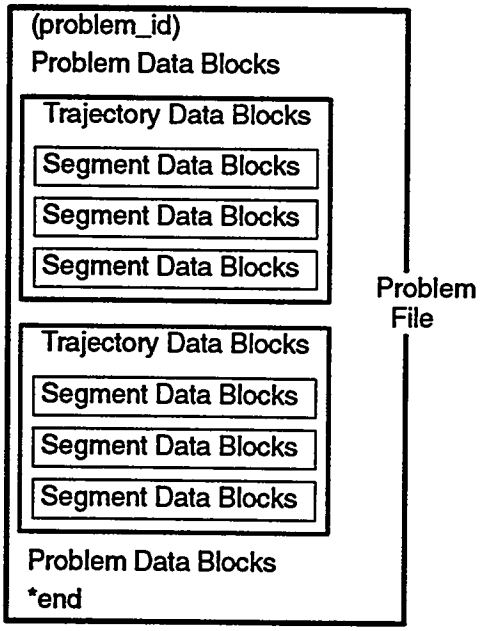

Figure 4–7. Problem and Data Block Organization.

The result is a problem file that is structured as shown in Figure 4–7. It begins with an identifier enclosed in parentheses. The identifier is often the same as the problem filename, but this is not required. This is followed by a group of data blocks containing information that applies to the entire problem.

These data blocks are not associated with a specific trajectory or segment; they apply to all of the trajectories. Examples include the **atmos*, **earth*, and **wind* data blocks. Table 4–13 contains a complete list of these data blocks. The data blocks marked with the † symbol are best placed at the beginning of the problem. The remaining data blocks should be put at the end of the problem.

After the initial group of problem data blocks, a set of data blocks is given defining the first trajectory. Within this set of data blocks are data blocks defining segments

for this trajectory. A set of data blocks for a second trajectory follows the first one. This continues for each trajectory in the problem. Finally, the last group of problem data blocks is given, and the problem ends with the **end* keyword.

Table 4–13. Problem Data Blocks.

Data Block	Description
*atmos†	Atmosphere model
*define†	User–defined variables
*earth†	Earth model
*egs	EGS database files
*file	Output files
*optimize	Trajectory optimization
*print	Printout file
*radar†	Radar observations
*search	Trajectory searches
*summarize	Summary variables
*survey	Trajectory surveys
*title†	Problem title
*units/fmt	Units and output format
*wind†	Winds

The data blocks at the beginning of the problem do not reference other trajectory or segment information, such as trajectory or segment numbers or user–defined variables. They contain general problem information that is independent of the trajectories. For example, the **atmos* and **earth* data blocks specify the atmosphere and earth models used for all trajectories.

The problem data blocks at the end usually reference trajectory and segment information that must be defined previously. For example, a **search* data block defines an objective function or goal of the search that references trajectory and segment numbers. Another example is an **egs* data block that references user–defined variables. The user–defined variables must be defined before they are referenced. This potential input error can be avoided if the **egs* data block is placed at the end of the problem.

4.4.1. *Atmos Data Block

An atmosphere model, defining temperature, pressure, density, speed of sound, and viscosity as a function of altitude, is required to integrate the equations of motion (Section 2.3.1). TAOS contains 20 standard model atmospheres, listed in Table 4–14, and has two user–defined model atmospheres. The default is the 1976 U.S. standard atmosphere.

The atmosphere models extend to an altitude of 1000 km; however, above an altitude of 146 km, all of the models are the same as the 1976 U.S. standard atmosphere. Below sea level, temperature is extrapolated based on the temperature gradient at sea level, and pressure is held constant at the sea level value. Other atmospheric properties are computed from the temperature and pressure.

Atmospheres are selected by entering the atmosphere number following the **atmos* keyword. For example

```
*atmos  20
```

selects the Kwajalein mean annual atmosphere model.

The selected atmosphere model applies to all trajectories in a problem; different trajectories cannot use different atmosphere models. The 1976 U.S. standard atmosphere is commonly used, so a special keyword *standard* can be used instead of entering the number one. Similarly, the special keyword *none* can be used instead of atmosphere type zero.

Table 4–14. Atmosphere Types.

Type	Atmosphere	Type	Atmosphere
0	None	11	60° North July
1	1976 U.S. Standard	12	75° North January
2	15° North Annual	13	75° North January (Cold)
3	30° North January	14	75° North January (Warm)
4	30° North July	15	75° North July
5	45° North January	16	Tonopah Winter
6	45° North July	17	Tonopah Spring
7	45° North Spring/Fall	18	Tonopah Summer
8	60° North January	19	Tonopah Fall
9	60° North January (Cold)	20	Kwajalein Mean Annual
10	60° North January (Warm)		

User–Defined Atmosphere

The first type of user–defined atmosphere requires tabulated values for all atmosphere properties as a function of altitude. The input format is best illustrated with the following example:

```
*atmos   user
    alt      temp      pres       rho      sndspd      visc
      0     548.2     2111.    .00224        1147     3.895e-7
   5000     530.0     1774.    .00195        1128     3.780e-7
  10000     511.0     1480.    .00168        1109     3.701e-7
  15000     494.9     1229.    .00124        1072     2.602e-7
  20000     477.5      833.    .00105        1051     3.400e-7
```

The keyword *user* indicates that this is a user–defined atmosphere with all properties specified. The next line gives column headers for the atmosphere properties and for altitude. All of these keywords are required, but their order is not important, that is, *temp* can be given before *pres* or vice versa. These keywords are the same as the atmosphere output variables given in the appendix, and the units for each variable can be changed with the **units/fmt* data block (Section 4.4.13).

The atmospheric properties are given as a function of altitude, so usually altitude is the first column of the table. The atmospheric properties on each line in the table correspond to an altitude. The altitude values must be strictly increasing, and no duplicate altitudes are allowed.

As the flight path is calculated the table values are linearly interpolated to obtain the atmospheric properties. Enough values must be provided so that linear interpolation models the atmospheric properties adequately. There are no limits on the number of values that can be input.

Site–Measured Atmosphere

The second type of user–defined atmosphere requires only pressure and density as a function of altitude. It is called a site–measured atmosphere and an exponential function is used for interpolation. This means that fewer points are required to represent a realistic atmosphere.

A site–measured atmosphere is input the same way as a user–defined atmosphere except that there are fewer columns of data. For example,

```
*atmos   site
    alt         pres         rho
      0         2115.     0.00223
  10000         1485.     0.00166
  20000          830.     0.00105
  30000          627.     0.00089
  40000          241.     0.00058
```

A temperature is estimated from the pressure and density with the ideal gas law, and speed of sound and viscosity are computed from this temperature.

4.4.2. *Define Data Block

TAOS always computes a standard set of variables for each trajectory. Additional variables can be defined within a trajectory definition that are functions of these standard variables (Section 4.3.1). These user–defined variables are defined only within a trajectory. If a quantity of interest is a function of variables from more than one trajectory, it must be defined outside of the trajectory definitions. Then, trajectory numbers can be used to reference values from different trajectories.

The difference between **define* data blocks within a trajectory definition and those outside of a trajectory definition is that **define* data blocks within a trajectory are restricted to be functions of that trajectory's output variables, and data blocks outside a trajectory are not restricted to be functions of a single trajectory. **Define* data blocks outside of trajectory definitions can be used to compute relative values comparing two or more trajectories.

User–defined variables are input the same way regardless of whether the **define* data block is within a trajectory definition or not. A unique name is given followed by equations that show how to compute the new variable. User–defined variables that are defined outside of trajectory definitions cannot be integral variables; this type of user–defined variable is only allowed within a trajectory definition.

The equations defining how to compute the user–defined variable are given in a simplified C language which is similar to Fortran. For example,

```
*define  del_alt
   del_alt = (alt[1] - alt[2]) / 3280.84;
```

computes the difference in altitude between trajectory numbers 1 and 2 in kilometers.

The name of the new variable is *del_alt*. It is given following the **define* keyword, and it must also appear on the left side of an equation somewhere in the definition. In this example, with one equation, *del_alt* must be the variable on the left side of the equal sign.

The values and variables on the right side of the equal sign must all be standard output variables, previously input user–defined variables, temporary variables, or constants. In this case, *alt* is a standard output variable and *3280.84* is a constant. Since this data block is located outside of the trajectory definitions in the problem file, trajectory numbers are required to fully specify the variables. Trajectory numbers are enclosed in square brackets so they look like subscripts.

Table 4–9 in Section 4.3.1 contains a list of operators and functions that can be used in equations for user–defined variables. All of the standard mathematical operators are available, and they have the same precedence as in C and Fortran. Many standard functions are also provided; again these are defined in TAOS as they are in the C language.

The table function shown in Table 4–9 cannot be used in **define* data blocks outside of a trajectory definition. Tables are functions of the state of a vehicle or trajectory

so they must be associated with a specific trajectory. They cannot be evaluated outside of a trajectory.

Equations must end with a semicolon like in C. The semicolon is required even when there is only one equation like in the example shown above. Since equations continue until a semicolon is encountered, they can extend across multiple lines as necessary.

When more than one equation is required to define a value, temporary variables can be used. These variables must be unique, that is, they cannot have the same name as a standard output variable or another user–defined variable. They are only defined within the **define* data block. For example,

```
*define  rel_range
   dx = (east[2] - east[1])^2;
   dy = (north[2] - north[1])^2;
   rel_range = sqrt(dx+dy);
```

computes a relative horizontal range between two trajectories using two temporary variables, *dx* and *dy*, in the calculation.

If–then–else statements can be used as shown in Section 4.3.1, but *while* loops, *for* loops, and other C–language control statements are not allowed.

User–defined variables given outside of the trajectory definitions can only be referenced by **print*, **file*, **egs*, and other **define* data blocks that are also outside of the trajectory definitions.

4.4.3. *Earth Data Block

TAOS requires an earth model to generate a trajectory. An earth model consists of an earth or planet shape, a gravity model, and some additional parameters such as the earth rotation rate. Four standard models are provided: a spherical earth model, the WGS–72 model,[5] the WGS–84 model,[6] and the GEM–T1 model.[11] If these models are not acceptable, parameters can be modified to produce a user–defined earth model.

The earth is assumed to be an ellipsoid with the equatorial radius greater than the polar radius. Its shape is given by an equatorial radius and either the polar radius, the eccentricity, or the flatness. If one of the latter parameters is provided, for example, eccentricity, the other two can be computed (Sections 2.1.5 and 2.1.6).

The earth's gravity is defined by a gravity constant (*GM*) and gravity harmonic coefficients (either J_n or $C_{n,m}$ and $S_{n,m}$). Two additional values, the earth rotation rate and the lb_m to slug conversion factor, are required to complete the earth model.

Spherical Earth Model

The simplest earth model is spherical. It requires an earth radius, an earth rotation rate, a gravity constant, and a lb_m to slug conversion factor. These values default to the WGS–84 values as shown in Table 4–15. The spherical earth model is selected with the **earth* data block as follows:

```
*earth  spherical  omega=0
```

where the keyword *spherical* sets the spherical earth default values. The remaining variables and values, in this case *omega=0*, override the defaults.

Table 4–15. Spherical Earth Model.

Symbol	Description	Default Value
reqtr	Earth radius (ft)	20925646.3255
omega	Rotation rate (rad/sec)	7.292115×10^{-5}
gm	Gravity constant (ft^3/sec^2)	$1.40764438125 \times 10^{16}$
g	lbm to slug conversion	32.1740485

WGS–72 and WGS–84 Earth Models

The WGS–72 and WGS–84 earth models assume the earth has an ellipsoidal shape with the equatorial radius greater than the polar radius. These models are selected with the *wgs–72* or *wgs–84* keywords. One of these keywords must immediately follow the **earth* data block name. Variables and values overriding the defaults can be given following this keyword. For example,

```
*earth  wgs-84  omega=0
```

selects the WGS–84 earth model but sets the earth rotation rate to zero. Nonrotating earth models are often used in conceptual design analysis.

Because of the ellipsoidal shape, the WGS–72 and WGS–84 earth models require additional parameters as shown in Table 4–16. The earth's shape is defined by the equatorial radius and one of the remaining shape parameters: the polar radius, the eccentricity, or the flatness. If more than one of these parameters is given, the geometry most likely will be inconsistent.

Table 4–16. WGS–72 and WGS–84 Earth Model Values.

Symbol	Description	WGS–84	WGS–72
reqtr	Equatorial radius (ft)	20925646.3255	20925639.7638
rpolr	Polar radius (ft)	20855486.5953	20855480.7087
ecc	Eccentricity	0.0818191908426	0.0818188106627
flat	Flatness parameter	1/298.257223563	1/298.26
omega	Rotation rate (rad/sec)	7.292115×10^{-5}	$7.292115147 \times 10^{-5}$
g	lbm to slug conversion	32.1740485	32.1740485
gm	Gravity constant (ft^3/sec^2)	$1.40764438125 \times 10^{16}$	$1.40764544069 \times 10^{16}$
j2	Degree 2 zonal harmonic	$1.08262998905 \times 10^{-3}$	$1.08261579002 \times 10^{-3}$

The gravity model is also modified with the second–degree zonal harmonic coefficient J_2 to account for the ellipsoidal shape of the earth. The remaining gravity coefficients are neglected.

TSAP Earth Models

TSAP and PMAST use a WGS–72 and WGS–84 earth model containing gravity zonal harmonic coefficients through degree 4. The second–degree gravitational zonal harmonic J_2 accounts for the oblateness of the earth. The remaining zonal harmonics account for additional gravitational variations with respect to latitude. Use of the J_3 and J_4 zonal harmonics are questionable for typical trajectories because they are of similar magnitude to the nonzonal harmonics which vary with longitude and to the gravitational effects of the sun and moon. If J_3 and J_4 are defined, then the nonzonal harmonic terms should also be defined for consistency.

However, the J_3 and J_4 zonal harmonics have been maintained in the *tsap–72* and *tsap–84* earth models so results from TAOS can be consistent with those from its predecessors TSAP and PMAST. The default values for J_3 and J_4 are shown in Table 4–17, and they are the same values used in TSAP and PMAST.

Table 4–17. TSAP Compatible Earth Model Values.

Symbol	Description	WGS–84	WGS–72
j3	Degree 3 zonal harmonic	$-2.532153068\times10^{-6}$	$-2.538810043\times10^{-6}$
j4	Degree 4 zonal harmonic	$-1.610987610\times10^{-6}$	$-1.655970000\times10^{-6}$

Full Degree–4 Earth Models

Two additional earth models, *wgs–84–full* and *gem–t1–full*, are available which include all terms in the gravity geopotential up through degree 4. Table 4–18 contains the variables and default values used for these two models. The coefficients input to TAOS are unnormalized, so if different values are input, they must also be unnormalized. Use of these earth models is not recommended because the gravitational effects of the sun and moon, which are neglected, are of the same order of magnitude as many of the coefficients.

Table 4–19. Full WGS–84 and GEM–T1 Earth Model Values.

Symbol	Description	WGS–84	GEM–T1
reqtr	Equatorial radius (ft)	20925646.3255	20925646.3255
rpolr	Polar radius (ft)	20855486.5953	20855486.5953
ecc	Eccentricity	0.0818191908426	0.0818191908426
flat	Flatness parameter	1/298.257223563	1/298.257223563
omega	Rotation rate (rad/sec)	7.292115×10^{-5}	7.292115×10^{-5}
g	lbm to slug conversion	32.1740485	32.1740485
gm	Gravity constant (ft^3/sec^2)	$1.40764438125\times10^{16}$	$1.4076441552\times10^{16}$
c20	Degree 2 zonal harmonic	-1.082629×10^{-3}	-1.082625×10^{-3}
c22	Degree 2 nonzonal harmonic	1.572805×10^{-6}	1.574322×10^{-6}
c30	Degree 3 zonal harmonic	2.532153×10^{-6}	2.532618×10^{-6}
c31	Degree 3 nonzonal harmonic	2.194673×10^{-6}	2.192402×10^{-6}
c32	Degree 3 nonzonal harmonic	3.096837×10^{-7}	3.086210×10^{-7}
c33	Degree 3 nonzonal harmonic	1.000789×10^{-7}	1.005372×10^{-7}
c40	Degree 4 zonal harmonic	1.610987×10^{-6}	1.616190×10^{-6}
c41	Degree 4 nonzonal harmonic	-5.080013×10^{-7}	-5.060561×10^{-7}
c42	Degree 4 nonzonal harmonic	7.780961×10^{-8}	7.759155×10^{-8}

c43	Degree 4 nonzonal harmonic	5.926679×10^{-8}	5.922238×10^{-8}
c44	Degree 4 nonzonal harmonic	-3.948164×10^{-9}	-4.015116×10^{-9}
s22	Degree 2 nonzonal harmonic	-9.023759×10^{-7}	-9.035928×10^{-7}
s31	Degree 3 nonzonal harmonic	2.709571×10^{-7}	2.695880×10^{-7}
s32	Degree 3 nonzonal harmonic	-2.121201×10^{-7}	-2.119137×10^{-7}
s33	Degree 3 nonzonal harmonic	1.973456×10^{-7}	1.970571×10^{-7}
s41	Degree 4 nonzonal harmonic	-4.498693×10^{-7}	-4.507384×10^{-7}
s42	Degree 4 nonzonal harmonic	1.466394×10^{-7}	1.484816×10^{-7}
s43	Degree 4 nonzonal harmonic	-1.189998×10^{-8}	-1.198933×10^{-8}
s44	Degree 4 nonzonal harmonic	6.540039×10^{-9}	6.517407×10^{-9}

4.4.4. *Egs Data Block

The Engineering Graphics System (EGS) is available on the Silicon Graphics workstations in the Aerospace Systems Development Center at Sandia.[4] This software is designed to interactively plot sets of analysis data, such as trajectory data. The data for EGS must be in a special format called the EGS database format, so the **egs* data block is available to produce a file containing trajectory variables in this format. It provides a simple interface to EGS, so that high–quality plots of the trajectory data can be produced quickly.

The **egs* data block contains a filename and a list of variable names. The file produced by this data block, an EGS database file, usually has the file type *.dbf*. A database file is created for each **egs* data block, and any number of **egs* data blocks can appear within a problem file.

An example of an **egs* data block is

```
*egs  traj.dbf
    time alt range vel mach long latgd gamgd
    psigd dynprs alpha nx ny nz ntotal
```

where the database filename is *traj.dbf* and 15 output variables have been listed. Any number of output variables can be listed in the data block. Figure 4–8 contains a listing of the first part of the database file created from this example.

```
*EGS DATA FILE
*TITLE
Example of a ballistic RV trajectory
*SYMBOLS
 gama = gama
 trajectory = Trajectory Number
 time = Time  (sec)
 alt = Altitude  (ft)
 range = Range  (nm)
 vel = Velocity  (ft/sec)
 mach = Mach Number
 long = Longitude  (deg)
 latgd = Geodetic Latitude  (deg)
 gamgd = Geodetic Flight Path Angle  (deg)
 psigd = Geodetic Heading Angle  (deg)
 dynprs = Dynamic Pressure  (slugs)
 alpha = Angle of Attack  (deg)
 nx = Body X-axis Load Factor  (g)
 ny = Body Y-axis G's  (g)
 nz = Body Z-axis G's  (g)
 ntotal = Total G's  (g)
*VARIABLES
  LEVEL  1  gama
  LEVEL  2  trajectory
  LEVEL  3  time  alt  range  vel  mach  long
            latgd  gamgd  psigd  dynprs  alpha  nx
            ny  nz  ntotal
*TABLE
      -35.000          1.00
        0.000     300000.0        0.000     18000.00      19.9642     20.00000
      0.00000      -35.000       90.000        0.829        0.000      -0.0026
       0.0000       0.0000       0.0026
        0.500     294835.2        1.196     18008.93      19.9741     20.01991
      0.00000      -35.021       90.000        1.100        0.000      -0.0034
       0.0000       0.0000       0.0034
```

Figure 4–8. Example of EGS Database File.

Data is automatically written to the database file for each trajectory defined in the problem, so no trajectory numbers are given. The units and output format of each variable can be set with the **units/fmt* data block (Section 4.4.13).

TAOS overwrites the **egs* database files without warning each time it runs. This automatically updates the files so they contain the latest trajectory information for a problem. If this is not desirable, then the filename must be changed between runs or the database files can be renamed before each new run.

Radar and relative vehicle output variables, those starting with *rad* or *rel*, must contain a subscript enclosed in square brackets. The subscript gives the radar station number or the trajectory number for the relative vehicle calculations. For example, the variable *radrng[2]* is used to output the radar range of each trajectory from radar station number 2, and the variable *relvel[4]* is used to output the closing velocity of trajectory number 4 relative to the other trajectories.

If surveys have been defined (Section 4.4.11), the survey loops form different data levels in the EGS database file so EGS can be used to plot the trajectory data as a function of the survey variables. In Figure 4–8 the variable *gama* is a survey loop variable, and a set of trajectories is computed for each value it takes on.

Summary variables (Section 4.4.10) are not written to the standard EGS database file. If summary variables have been defined, they can be written to an EGS database file with

```
*egs  summary  file.dbf
```

The *summary* keyword and the filename are required; no output or summary variable names are given. This form of the **egs* data block, with the *summary* keyword and a filename, requires at least one survey loop and at least one summary variable. All summary variables are written to the EGS database file as a function of the survey variables.

4.4.5. *File Data Block

The **file* data block is used to create files containing columns of trajectory information. A list of variable names is given in the data block, and a column of trajectory data is written for each variable. These files are not formatted for printing, that is, they do not contain page breaks and page headers. They are intended for use by other applications, such as plotting programs or spreadsheets, that require columns of data.

**File* data blocks can appear either within trajectory definitions or outside of the trajectory definitions. The difference is that **file* data blocks within a trajectory definition assume the information written to the file is for a specific trajectory; whereas, **file* data blocks outside of the trajectory definitions do not make this assumption. The result is that the variables listed for **file* data blocks outside of the trajectory definitions must contain trajectory numbers enclosed in brackets.

For example,

```
*file  compare.dat
   time[1]  alt[1]  alt[2]  alt[3]  vel[1]  vel[2]  vel[3]
```

writes the time from trajectory number 1, the altitudes from trajectories 1, 2, and 3, and the velocity from trajectories 1, 2, and 3 to the file *compare.dat.* Figure 4–9 shows the first part of this file.

Time[1]	Alt[1]	Alt[2]	Alt[3]	Vel[1]	Vel[2]	Vel[3]
0.000	100000.0	80000.0	90000.0	9000.00	7000.00	8000.00
0.500	100000.0	80000.0	90000.0	8992.80	6987.89	7991.04
1.000	100000.0	80000.0	90000.0	8985.61	6975.80	7982.09
1.500	100000.0	80000.0	90000.0	8978.43	6963.73	7973.14
2.000	100000.0	80000.0	90000.0	8971.24	6951.68	7964.21
2.500	100000.0	80000.0	90000.0	8964.07	6939.65	7955.29
3.000	100000.0	80000.0	90000.0	8956.89	6927.64	7946.39
3.500	100000.0	80000.0	90000.0	8949.72	6915.65	7937.49
4.000	100000.0	80000.0	90000.0	8942.56	6903.67	7928.60
4.500	100000.0	80000.0	90000.0	8935.40	6891.72	7919.73
5.000	100000.0	80000.0	90000.0	8928.24	6879.78	7910.86
5.500	100000.0	80000.0	90000.0	8921.09	6867.86	7902.00
6.000	100000.0	80000.0	90000.0	8913.94	6855.97	7893.16
6.500	100000.0	80000.0	90000.0	8906.80	6844.09	7884.33
7.000	100000.0	80000.0	90000.0	8899.66	6832.23	7875.51
7.500	100000.0	80000.0	90000.0	8892.53	6820.39	7866.69
8.000	100000.0	80000.0	90000.0	8885.40	6808.56	7857.88
8.500	100000.0	80000.0	90000.0	8878.27	6796.76	7849.08
9.000	100000.0	80000.0	90000.0	8871.15	6784.98	7840.28
9.500	100000.0	80000.0	90000.0	8864.04	6773.21	7831.49
10.000	100000.0	80000.0	90000.0	8856.93	6761.47	7822.70

Figure 4–9. Trajectory Output File Example.

The filename must be valid. If the file already exists, it is overwritten with no warning and old data is lost. This is normally satisfactory because the file always contains information from the most recent run. Any number of **file* data blocks can be included in a problem; one output file is created for each **file* data block.

Any standard output variable or user–defined output variable can be written to the file. Most variables are associated with a trajectory so a trajectory number is required, and it must be enclosed in brackets as shown above. The only exceptions are user–defined variables defined outside of the trajectory definitions because they are

associated with the entire problem, not a single trajectory. The units and the printout format are controlled with the **units/fmt* data block (Section 4.4.13).

Each output line is limited to 400 characters; this limits the number of output variables that can be listed to about 30. Usually it is best to limit the number of output variables to 12 or fewer because some screen editors and printers can only handle about 130 characters per line.

Radar and relative vehicle output variables, those starting with *rad* or *rel*, must contain an additional subscript enclosed in square brackets. The first subscript gives the radar station number or the trajectory number for the relative vehicle calculations, and the second subscript gives the vehicle trajectory number. For example, the variable *radrng[2][1]* is used to print the radar range of trajectory number 1 from radar station number 2, and the variable *relvel[4][2]* is used to print the closing velocity of trajectory number 4 relative to trajectory number 2.

4.4.6. *Optimize Data Block

Parametric optimization can be used to maximize or minimize a trajectory characteristic, such as final range or velocity, and it can be used to satisfy trajectory constraints, such as a final impact position or flight path angle. Although optimization is one of the most powerful features of TAOS, the numerical optimization methods are complex and not as reliable as the rest of the trajectory integration methods.

A parametric optimization problem varies a set of parameters x_i in order to maximize or minimize an objective function $f(x_i)$ subject to a set of constraints $c(x_i)$. The parameters x_i are input variables in the problem file. The objective function $f(x_i)$ is an output variable at a specified point in the trajectory, for example, an output variable at the end of a trajectory segment. The constraints $c(x_i)$ can be equality constraints, where an output variable must be equal to a value, or inequality constraints, where a limit or boundary is imposed.

TAOS uses a nonlinear constrained optimization method based on recursive quadratic programming called *vf02ad* (Section 2.6.3).[3] The partial derivatives required by the algorithm are estimated with forward or central differences.

Optimization Loops

Optimization is often only applied to part of a trajectory, called the optimization loop. The loop extends from the first segment containing an optimization parameter x_i to the segment defining the objective function $f(x_i)$. It can contain any number of segments.

A problem can have up to five optimization loops, but they must not overlap and they cannot be nested. Each loop in a problem must be completely independent from other search and optimization loops. The optimization loops are given a letter identifier (*a*, *b*, *c*, *d*, or *e*) which is used to associate the optimization parameters x_i with the optimization data block defining the objective function $f(x_i)$ and constraints $c(x_i)$.

An optimization loop can have any number of parameters x_i, but in practice the optimization method works best if the number of parameters is less than 25. The method converges faster and better with fewer parameters, so the goal in setting up an optimization loop is to include just enough parameters to adequately model a trajectory. The parameters are numbered starting from one. The numbers must be sequential, so if there are 10 parameters, they must be numbered from 1 to 10. However, they do not have to appear in any particular order in the problem file.

Optimization Parameters

Optimization parameters are identified by setting an input variable equal to the keyword *optx–n*, where *x* is the optimization loop letter and *n* is the parameter number. For example,

```
*fly  alpha = opta-2
```

designates the angle–of–attack value as optimization parameter number 2 in loop *a*. Any input variable with a numerical value (a number, not a table name) can be designated as an optimization parameter. The initial conditions and guidance rules are most commonly used as optimization parameters.

A common technique used to optimize part of a trajectory is to define a segment with a guidance rule and final condition similar to the following:

```
*fly  alpha  vrs  tseg
      opta-1      0.0
      opta-2      5.0
      opta-3      10.0
      opta-4      15.0
      opta-5      opta-6

*when  tseg>opta-6  stop
```

This uses six parameters to define an angle–of–attack time history for a segment. Although the segment times must be estimated, optimization is used to determine the angle of attack values required to satisfy the objective function and constraints. The final segment time is included as a parameter to give more flexibility to the flight path.

Objective function

So far, only the optimization parameters x_i have been specified. The remaining parts of the optimization loop, that is, the objective function and constraints, are given in an **optimize* data block.

The first line of the **optimize* data block defines the objective function $f(x_i)$. The most commonly used variables for the objective function are time, range, and velocity. For example,

```
*optimize a for vel=max on segment 6, trajectory 1
```

defines the objective function as the final velocity of segment number 6 in trajectory number 1. The goal of optimization is to maximize this value. The optimization loop letter, in this case *a*, is given first immediately following the **optimize* keyword. The keyword *for* is required between the loop letter and the objective function. Any output variable can be used for the objective function, and the output variable can be set to *min* or *max*. The segment and trajectory numbers are required.

Constraints

The next set of lines can be used to define constraints on the flight path. These constraints are optional, and they can be equality or inequality constraints. For example, the lines

```
constrain  vel=3000
constrain  gamgd=-80
constrain  qmin<0
```

define three constraints. These constraints are on the trajectory output variables, not on the input optimization parameters. Limits on the parameters are given later in the data block.

In this example, the constraints are evaluated at the same point in the trajectory as the objective function because no trajectory or segment numbers are provided. If these constraints are applied as a continuation to the **optimize* example above, then the velocity must be equal to 3000, the flight path angle must be equal to –80, and the user–defined variable *qmin* must be less than zero at the end of segment number 6 on trajectory number 1.

Segment and trajectory numbers can be specified on each constraint if they are different from those given for the objective function. For example,

```
constrain  vel on segment 3, trajectory 2 = 3000
```

constrains the velocity on segment 3 in trajectory 2 to 3000 ft/sec.

User–defined integral variables, such as *qmin* shown above, are often used for inequality constraints in optimization to keep a trajectory parameter within known boundaries. For example, vehicles with aerodynamic control surfaces must operate within the atmosphere, that is, they require a minimum dynamic pressure to maintain vehicle control. This limit must be maintained over the entire trajectory, not just at the end of a segment. If the limit is 500 lb_f/ft^2, then a user–defined integral variable can be defined as

$$q_{\min} = \int (q - 500)^2 \qquad \textit{for } q < 500 \qquad (4\text{–}5)$$

$$q_{\min} = 0 \qquad \textit{otherwise} \qquad (4\text{–}6)$$

If q_{min} is constrained to be less than zero, it forces the dynamic pressure to stay above the minimum value of 500 lb_f/ft^2. Altitude, g loads, and other flight path constraints can be defined similarly. Section 4.3.1 contains information explaining how to input an integral user–defined variable.

Multiple–Vehicle Constraints

The full constraint expression consists of the keyword *constrain* followed by a relationship and an optional reference factor. The relationship consists of an output variable name, followed by a relationship (=, <, or >), followed by a value. The value can either be a number, like shown in the examples above, or another output variable evaluated at the same or different point in the trajectory. This feature is useful when computing intercept trajectories, for example,

```
constrain  alt=alt[2]
```

says to make the altitude on the optimization trajectory equal to the altitude on trajectory number 2. Enclosing the trajectory number in brackets is one method for entering the trajectory number.

Another method is to use keywords as follows:

```
constrain  alt on segment 6, trajectory 1
             = alt on segment 4, trajectory 3
```

This makes the altitude on segment number 6 in trajectory number 1 equal to the altitude on segment number 4 in trajectory number 3. These altitudes are undefined unless both segment 6 in trajectory 1 and segment 4 in trajectory 3 have been computed at the time the objective function is evaluated. The keywords *trajectory* and *on segment* are used to provide the trajectory and segment numbers.

Trajectory numbers can be given with subscripts enclosed in square brackets, for example,

```
constrain  east[2] on segment 4 = east[1] on segment 7
```

Or they can be given following the keyword *trajectory* as follows:

```
constraint  east on segment 4, trajectory 2
                = east on segment 7, trajectory 1
```

Both methods result in the same constraint. Trajectory numbers cannot be given twice using both a subscript and the *trajectory* keyword.

If both trajectory and segment numbers are given, the segment number is given first followed by the trajectory number. If trajectory and segment numbers are not given in the constraint, they default to the objective function values.

Referencing Values

The optimization method is more reliable if all values involved in the calculations, that is, the parameters, the objective function, and the constraints, are of approximately the same order of magnitude. If some constraints or parameters have values that are orders of magnitude larger than others, then numerical difficulties often result. Methods are provided in TAOS to normalize or reference values involved in the optimization process.

Constraint relationships that contain a number, such as *alt=30000*, are automatically normalized or referenced by the number (unless it is zero). Other constraints can be referenced by an input value, for example,

```
constrain  vel on segment 11 = vel[3] on segment 4, ref=1000
```

references both velocity values in the constraint by a factor of 1000. The *ref* variable must be the last item in the constraint (it follows the relationship). It is optional, and its only purpose is to improve the numerical calculations.

Optimization Control Variables

After constraints have been given, some variables, summarized in Table 4–20, are input that control the optimization process. All of these variables have reasonable default values, so they are optional.

The objective function reference value *fref* is used to normalize the objective function the same way the *ref* variable is used on constraints. For best results the value $|f(x_i)/f_{ref}|$ should be between 1 and 10.

The convergence accuracy variable *tol* controls when the optimization procedure believes it has a valid solution. Smaller values may produce a better solution, but they require many more iterations to converge. If the value is too small, convergence may not be achieved.

Table 4–20. Optimization Data Block Input Variables.

Variable	Description	Variable	Description
fref	Objective function reference value (default = 1.0)	derivs	Derivative method (default=1)
tol	Convergence accuracy (default = 1.0×10^{-6})	dx	Derivative increment (default = 1.0×10^{-8})
maxitr	Maximum number of iterations (default = 50)	adjust	Guidance table adjust (default=0)
integ	Integration control (default=1)	surveys	Survey control flag (default=0)
restarts	Restart control (default = 0)	print	Printout control (default=0)

The maximum number of iterations *maxitr* controls the number of times that the optimization procedure *vf02ad* is called. It is related to the number of function evaluations (the number of trajectories computed). If *maxitr* is set to zero, the trajectory is calculated once with the initial parameter estimates. This is used to check the starting trajectory making sure it is reasonable.

During the optimization process, trajectories are recomputed many times, once for each function evaluation. The integration control variable *integ* controls whether all trajectories in a problem are recomputed (=1) or whether only the trajectory with the objective function is recomputed (=0). If the optimization problem is restricted to a single trajectory, then *integ* can be set to zero significantly reducing the amount of computation; the problem runs much faster. If the optimization problem has constraints that reference other trajectories, then it is probably necessary to recompute all trajectories with *integ* set to one.

Automatically Restarting Optimization

If the optimization procedure has not converged within the maximum number of iterations, it can be restarted. Sometimes restarting from the current solution provides additional progress towards the minimum. The variable *restarts* controls how many times the optimization procedure is restarted before it stops. A value of zero results in no restarts.

When restarts are used, the *maxitr* value needs to be reduced to a smaller value such as 20. On the last restart, optimization is allowed to proceed for twice the number of iterations given by *maxitr* in an effort to achieve convergence.

When angle–of–attack time histories are used in a **fly* data block for optimization, for example,

```
*fly  alpha  vrs  tseg
      opta-1      0
      opta-2      20
      opta-3      40
      opta-4      60
      opta-5      80
      opta-6      100
      opta-7      opta-8

*when  tseg>opta-8  stop
```

the segment times have to be estimated. The optimization procedure generally works best when these points are evenly distributed within the segment as shown above. However, during optimization the final segment time, *opta–8*, can change to either a very large number or to a number very close to the previous point, *100*. In either case, the distribution of time history points becomes uneven, which can affect the solution.

The guidance table adjustment variable, *adjust*, controls whether these time values are automatically adjusted to be evenly spaced when the optimization procedure restarts. If it is set to zero, no adjustment takes place, and if it is set to one, then the values are adjusted at each restart.

Numerical Derivatives

The optimization procedure requires partial derivatives during the solution process. These derivatives can be computed numerically with a central–difference method, *derivs* = 0, or a forward–difference method, *derivs* = 1. Forward–difference derivatives are computed from

$$\frac{\partial f(x)}{\partial x} = \frac{f(x + \delta x) - f(x)}{\delta x} \qquad (4\text{–}7)$$

and central–difference derivatives are computed from

$$\frac{\partial f(x)}{\partial x} = \frac{f(x + \delta x) - f(x - \delta x)}{2\delta x} \qquad (4\text{–}8)$$

The accuracy of both derivative methods is controlled by the derivative increment variable δx or dx. Smaller values of dx give more accurate derivatives than larger values until the limits of numerical precision are reached. The central–difference method requires more function evaluations, but gives more accurate derivatives than the forward–difference method for the same value of δx. The default values for *derivs* and dx generally work well and do not have to be modified.

Survey Control Flag

If surveys are used with optimization, the *surveys* variable controls the starting trajectory for each set of survey values. For the first set of survey values, the starting

trajectory is always the one given by the *par–n* variables in the **optimize* data block. The default (*surveys = 0*) uses this same starting trajectory for all other survey values. But if *surveys = 1*, the solution from the previous set of survey values is used as a starting trajectory for the next set of survey values. This often improves convergence when the optimum trajectories do not change much from one survey value to the next.

Printout Control

The optimization process always prints a summary giving the optimization parameters, the objective function, and the constraints at each iteration. By setting the variable *print* to one, the trajectory is printed after each iteration. This is sometimes useful for locating problems with the optimization process, but it produces a large amount of output. It is not recommended except when the variable *maxitr* is set to a small value such as 0 or 1.

Initial Estimates and Limits on Parameters

The optimization procedure requires an initial estimate or starting trajectory. The procedure works best when this trajectory has similar characteristics to the final solution. The initial estimate is given by providing a value for each optimization parameter. In addition to the initial estimate, upper and lower boundaries, and a reference or normalizing value can be specified for each parameter.

These values are given according to the optimization parameter numbers used in the problem file. For example, values for the parameter *opta–3* are given with

```
par-3=6.5,   lo-3=-10,   hi-3=30
```

This input line sets the initial estimate for parameter 3 to 6.5, and it limits parameter 3 to values between –10 and 30. No reference value is given with the *ref–3* variable because this parameter does not take on extremely large or small values.

The following example:

```
par-2=1000,  lo-2=-10000,  hi-2=100000,  ref-2=10000
```

uses the *ref–2* variable to normalize or reference the parameter to values in the range of –1 to 10. This is only necessary if the parameter values become very large (greater than 1000) or very small (less than 0.01).

Values for the initial estimate are required for each parameter. Values for the upper and lower limits are not required, but they are recommended. Reference values are optional. Usually values for each parameter are given on separate lines making it easy to read as shown in the example below.

Example Data Block

The full optimization data block consists of the objective function definition, the constraints, the control variables, and the parameter values. The following example contains a complete optimization data block showing how these parts fit together:

```
*optimize a for range=max on segment 12, trajectory 1

   constrain  vel=4000
   constrain  gamgd=-75

   fref=100       maxitr=40    tol=1.0e-7

   par-1=9.9      lo-1=-15     hi-1=15
   par-2=8.5      lo-2=-15     hi-2=15
   par-3=5.0      lo-3=-15     hi-3=15
   par-4=1.0      lo-4=-15     hi-4=15
   par-5=65.0     lo-5=51.0    hi-5=300.0
```

This data block sets up an optimization problem with five parameters that maximizes the range at the end of segment 12 on trajectory 1. While maximizing the range, the final velocity must be 4000 and the final flight path angle must be –75. The optimization procedure ends either when it has converged or after 40 iterations with no restarts.

Convergence

The optimization procedure ends when the maximum number of iterations is exceeded, when it detects convergence, or when it has problems converging. If the maximum number of iterations is exceeded, values for the objective function for the last few iterations can be inspected in the printout file. If the objective function is not changing much, then the resulting trajectory is probably near optimum. Otherwise, the value for the input variable *maxitr* needs to be increased.

The optimization procedure often ends with the message "optimization procedure does not detect convergence, but it cannot make further progress towards the minimum." This message means that the optimization procedure found a search direction, but while searching along this direction, it could not find a minimum. In other words, the one–dimensional search failed to locate a minimum. This usually means that the optimization procedure has converged as far as it can given the numerical precision of the computer, that is, the convergence tolerance is too small for the optimization method to achieve. This message can be treated as a warning message, but in general it means a good solution has been found.

Troubleshooting – Optimization Loop

Trajectory optimization is more an art than a science because the methods used, such as *vf02ad*, are sensitive to the starting trajectory and to the accuracy of the partial derivatives. Experience and judgement are required to successfully use optimization.

The optimization loop definition is the first place to check if the optimization procedure fails to converge or gives poor results. When an optimization parameter is varied, the objective function and constraint values should change. If they do not change, then the parameter has no effect and can be eliminated.

The end of the optimization loop is at the end of the segment given in the objective function definition. At this point in the trajectory, all other trajectories and segments referenced in constraints must have been computed. TAOS must be able to inspect the trajectory up to this point and retrieve values for all of the constraint variables.

If the optimization loop involves more than one trajectory and *integ = 0*, then problems can occur. The optimization loop must be set up carefully so that all values required by the objective function and constraints can be properly calculated. It is often more practical to use *integ = 1* even though it is less efficient.

Test runs can be made with *maxitr = 0* to check the optimization loop definition. The effect of changing values of the optimization parameters can quickly be determined with a few test runs. This method can also be used to verify that the starting trajectory is reasonable.

Troubleshooting – Segment Final Conditions

If you are using an optimization parameter to control the length of a segment, it is best to use a segment final condition based on a time variable. For example,

```
*when  tseg > opta-3  goto 5
```

lets the optimization procedure determine when to end the segment. Final conditions with time variables work better than other variables, such as altitude, because time is a strictly increasing variable.

When final conditions are optimized, only one final condition on time should be used. If more than one final condition is used, the optimization procedure becomes confused when sometimes one final condition is hit and other times a different condition is hit. Optimization constraints can be used to satisfy additional requirements at the end of a segment.

Troubleshooting – Starting Trajectory

The optimization procedure tends to be sensitive to the starting trajectory, that is, the initial estimates of the parameters. Optimization problems with a large number of parameters tend to be more sensitive to initial estimates than those with a small number of parameters. If an optimization loop fails to converge or gives a poor solution, then one should try different values for the initial estimates. Sometimes small changes to the initial estimate cause the optimization procedure to begin working properly.

A solution from optimization should not be trusted completely because the procedure sometimes converges to a local minimum instead of a global minimum. A good way to check the validity of an optimization solution is to rerun the problem with a different starting trajectory. If it results in the same solution, then a global minimum has probably been found.

Troubleshooting – Numerical Derivatives

Numerical derivatives are required by the optimization procedure at each iteration. As mentioned previously, these are calculated with forward or central differences by

making very small perturbations in the trajectory. The integration step size, given in the **integ* data block, must be small enough to accurately capture these small perturbations. If it is too large, numerical errors in the trajectory integration cause incorrect derivatives and these cause the optimization procedure to work towards an incorrect solution. Experimentation is required to judge how small to make the integration step size.

Accurate derivatives are critical to the performance of the optimization procedure. If the integration step size is not the problem, then the truncation error from the derivative calculation may be too large. The accuracy of the derivatives can be improved by decreasing the size of the derivative increment *dx* by an order of magnitude or more or by changing the derivative method to central differences with *derivs = 0.*

The value of *dx* is limited by the numerical precision of the computer. When it is too small, numerical errors become larger than the truncation error, and the optimization procedure continues to have difficulties. At this point, the value of *dx* should be increased and central differences should be used.

Troubleshooting – Surveys

Using surveys with optimization often causes problems because the same starting solution cannot be used for all survey values. This is particularly a problem when optimization parameters are in an angle–of–attack time history. Trajectories with optimized angle–of–attack time histories are best run individually and not surveyed unless the survey value does not change much. Sometimes the *restart* and *adjust* options can be used to solve this problem with surveys, but they should only be tried as a last resort.

Troubleshooting – Angle–of–Attack Time Histories

A common problem with optimization parameters in an angle–of–attack time history, such as

```
*fly  alpha vrs tseg
      opta-1      0
      opta-2     10
      opta-3     20
      opta-4   opta-5
```

is with the lower limit on the last time value *opta–5*. If this value becomes less than the previous time value *(20)*, then the time values in the table will not be continuously increasing and table interpolation will fail. So the lower limit of this parameter must be set to a value slightly larger than the previous time point, for example,

```
lo-5 = 20.5
```

4.4.7. *Print Data Block

The **print* data block can appear either within a trajectory or within a problem (outside of a trajectory definition). The purpose of the **print* data block is the same in either case: to write trajectory information to the standard output file. The standard output file has the same filename as the problem file, but it is of type *.out* instead of *.prb*. It has page breaks and headings formatted for a standard printer with 132 characters per line and 60 lines per page.

The difference between **print* data blocks located within a trajectory definition and those located outside of a trajectory is that those located within a trajectory assume the printed information is for that trajectory. The variables printed are assumed to be associated with the trajectory containing the **print* data block.

**Print* data blocks that are not within a trajectory definition do not make this assumption, so trajectory numbers enclosed in brackets are required in addition to the variable names. This provides the capability to print variables from several trajectories on the same page.

For example,

```
*print
   time[1]  alt[1]  alt[2]  range[1]  range[2]
```

prints the time from trajectory number 1, the altitude from trajectories 1 and 2, and the range from trajectories 1 and 2 as shown in Figure 4–10. This makes it easy to compare values from different trajectories. Data from different trajectories are related by mission time so *alt[1]* and *alt[2]* refer to the altitudes of trajectories 1 and 2 at the same instant in time.

```
Sandia National                          Trajectory Analysis & Optimization Software
Laboratories                                        (TAOS - Version 96.0)
-------------------------------------------------------------------------------

         Example of two vehicles relative to each other
         Problem (two)  /  1976 US Standard Atmosphere  /  Non-rotating WGS-84 Earth
         Trajectory for vehicle:  vehicle_1
    Time[1]       Alt[1]       Alt[2]     Range[1]     Range[2]
      0.000     100000.0      80000.0        0.000        0.000
      0.500     100000.0      80000.0        0.737        0.573
      1.000     100000.0      80000.0        1.473        1.146
      1.500     100000.0      80000.0        2.209        1.717
      2.000     100000.0      80000.0        2.944        2.287
      2.500     100000.0      80000.0        3.678        2.857
      3.000     100000.0      80000.0        4.412        3.425
      3.500     100000.0      80000.0        5.145        3.993
      4.000     100000.0      80000.0        5.878        4.559
      4.500     100000.0      80000.0        6.610        5.124
      5.000     100000.0      80000.0        7.341        5.689
      5.500     100000.0      80000.0        8.072        6.252
      6.000     100000.0      80000.0        8.803        6.815
      6.500     100000.0      80000.0        9.532        7.376
      7.000     100000.0      80000.0       10.262        7.937
      7.500     100000.0      80000.0       10.990        8.497
      8.000     100000.0      80000.0       11.718        9.055
      8.500     100000.0      80000.0       12.446        9.613
      9.000     100000.0      80000.0       13.172       10.170
      9.500     100000.0      80000.0       13.899       10.725
     10.000     100000.0      80000.0       14.624       11.280
```

Figure 4–10. Trajectory Printout Example.

The printout begins with a page heading, a title, and column headings that are repeated on every page. This is followed by the trajectory variables. Any of the standard output variables listed in the appendix can be printed along with any user–defined variables. The output format of each variable is controlled with the **units/fmt* data block (Section 4.4.13).

A table of trajectory data is written, with a maximum of 10 output variables, for each **print* data block. Any number of **print* data blocks can be included within a problem.

Radar and relative–vehicle output variables, those starting with *rad* or *rel*, must contain an additional subscript enclosed in square brackets. The first subscript gives the radar station number or the trajectory number for the relative–vehicle calculations, and the second subscript gives the vehicle trajectory number. For example, the variable *radrng[2][1]* is used to print the radar range of trajectory number 1 from radar station number 2, and the variable *relvel[4][2]* is used to print the closing velocity of trajectory number 4 relative to trajectory number 2.

4.4.8. *Radar Data Block

Radar observations of trajectories can be simulated from one or more fixed radar station locations. Radar observations include the range, azimuth, and elevation angles, their rates, and their accelerations. The vehicle aspect and meridional angles relative to the station are also computed (Section 2.4.2).

The presence of one or more radar output variables in the problem triggers the radar observation calculations. The radar stations are identified with a number, starting with 1. Subscripts enclosed in square brackets are used on the radar output variables to indicate the station number. For example, *radrng[2]* is the radar range from station number 2, and *radrng[3][2]* is the radar range from station number 3 for trajectory number 2. When double subscripts are used, the trajectory number is given last.

A **radar* data block is required for each radar station giving its location. For example,

```
*radar  1  station_x  alt=539.24,  long=-91.43562,
      latgd=30.02347
```

defines radar station number 1, called *station_x*, at a geodetic latitude of 30.02347°N, a longitude of 91.43562°E, and an altitude of 539.24 ft.

Any number of radar stations can be defined. Although they are identified with a station number, they also have a station name which is used as a comment to further identify the station. The station number and name must immediately follow the **radar* keyword. The remaining variables, giving the station location, follow. These variables can be given in any order.

The station location is input with the geodetic position variables *alt*, *long*, and *latgd*. Additional cartesian offset values from this location can be input with the variables *diste*, *distn*, and *distd*. These adjustments are made relative to the position given by *alt*, *long*, and *latgd*, in the local geodetic horizon coordinate system, and they do not follow the curvature of the earth's surface. Therefore, these adjustments should be small to maintain accuracy.

The station location variables are normally referenced to the earth model shape given in the **earth* data block; however, this can be changed by entering earth shape information in the **radar* data block. Earth geometry input in the **radar* data block is only used for the radar calculations; it is not used anywhere else in the trajectory calculations.

The keywords *wgs–72* and *wgs–84* can be used to select an earth model, or the earth shape values can be input directly with the variables *reqtr*, *rpolr*, *flat*, and *ecc*. As with the **earth* data block, the earth shape variables must produce consistent geometry. The variable *reqtr* is required, and one of the other shape variables (*rpolr*, *flat*, or *ecc*) must be given. Additional values are redundant and may form inconsistent geometry.

The input variables for the **radar* data block are summarized in Table 4–21. The following **radar* data block

```
*radar 2 station_y  alt=112.4  long=65.345  latgd=12.435
       wgs-72  distn=25.5  diste=15.2  distd=-10.3
```

shows another example which defines a station relative to a WGS–72 earth model that is located at an altitude of 112.4 ft, a longitude of 65.345°E, and a latitude of 12.435°N. From this position the radar is offset 25.5 ft north, 15.2 ft east, and 10.3 ft up.

Table 4–21. Radar Data Block Variables.

Variable	Description	Variable	Description
alt	Altitude of radar station (ft)	reqtr	Earth radius at equator (ft)
long	Longitude of radar station (deg)	rpolr	Earth radius at pole (ft)
latgd	Geodetic latitude of radar station (deg)	flat	Earth flatness parameter (0 < flat < 1)
distn	North offset (ft)	ecc	Earth eccentricity (0 < ecc < 1)
diste	East offset (ft)	wgs–72	Use WGS–72 earth shape
distd	Down offset (ft)	wgs–84	Use WGS–84 earth shape

4.4.9. *Search Data Block

Trajectories often have constraints that must be satisfied, such as a final velocity or range. Generally optimization is used in TAOS to obtain a trajectory that meets a set of constraints (Section 4.4.6). Optimization varies a set of parameters to maximize or minimize an objective function subject to a set of constraints. However, if the problem is simple and there are only one or two parameters and constraints, optimization may not be necessary, especially since optimization techniques are not completely reliable. A simpler, more reliable, search method is provided to handle the simple cases.

Search Parameters

Each search is one dimensional, that is, it repeatedly modifies a value in the problem file until a condition is met. For example, a constant angle of attack used in a pullout from descending flight to level flight can be varied until the pullout occurs at a given altitude. More than one variable in a problem file can be changed in a search, but all of the variables are set to the same value each time the variable is modified.

The search variables in a problem file are designated with the keyword *srch–n* (Section 4.1). For example,

```
*fly  alpha = srch-1
```

says that a search is going to be used to determine the value of the angle of attack. The search, that is, how to compute this value, is defined later in the problem file in a **search* data block.

The search number, the "1" in the example above, relates the search variables to the **search* data block. Any number of searches can be defined producing a set of nested one dimensional search loops, but if more than two nested searches are required, it is usually best to use optimization rather than searches.

Search Objective

If a search variable, like *srch–1*, is used in the problem file, then there must be a corresponding **search* data block in the file. A data block corresponding to the above example might look like

```
*search 1 vary alpha until alt=30000 on segment 3,trajectory 1
     xlo=0.5   xhi=10.0      xest=5.0    dx=1.0    tol=0.001
     xref=1.0  fref=10000    maxitr=20   print=1   integ=0
```

The first line defines the search objective; in this case, the angle of attack in the guidance policy is varied until the final altitude on segment 3 of trajectory 1 is equal to 30,000 ft. The next two lines give the search control parameters, for example, the angle of attack is varied between 0.5° and 10.0° with an initial estimate of 5.0°.

Parabolic search methods are used as shown in Section 2.6.2. These methods vary a search variable x until a search function $f(x)$ is equal to a value, a minimum, or a

maximum. The search function, given in the **search* data block, determines the type of search.

The first values in the **search* data block are the search number, the keyword *vary*, a name, and the keyword *until*. The name is used to identify the search parameter in the printout file; any name can be used.

The search objective is given next as an output variable set equal to a value. The value can be a constant, another output variable, the keyword *min*, or the keyword *max*. A segment and trajectory number must be provided for the output variable, and if a second output variable is given, segment and trajectory numbers can be provided for it as well. For example,

```
*search 1 vary pitch_rate until pitchi on segment 2
     = pitchi on segment 3, trajectory 2
```

says to modify the pitch rate (most likely a value in a guidance rule) until the inertial pitch angles on two segments are equal. The trajectory number is only given for one of the variables, so it is assumed to be the same for both.

In the following example, subscripts enclosed in square brackets are used to denote the trajectory numbers:

```
*search 1 vary weight until alt[1] on segment 3
     = alt[2] on segment 5
```

Trajectory numbers can be given with subscripts or with the *trajectory* keyword, but not with both. When segment and trajectory numbers are given and the *trajectory* keyword is used, the segment number must be given first, followed by the trajectory number. If a trajectory or segment number is not provided for a variable, it is assumed to be the same for both variables.

Search Control Variables

Table 4–22 contains a list of the search control variables. The search methods require an initial estimate of the search parameter *xest*, an initial search increment *dx*, a lower search boundary *xlo*, and an upper search boundary *xhi*. The other search control variables have default values so they are optional.

Table 4–22. Search Data Block Variables.

Variable	Description	Variable	Description
xlo	Lower search boundary	xref	Search variable reference value (default = 1.0)
xhi	Upper search boundary	fref	Search function reference value (default = 1.0)
xest	Initial search estimate	maxitr	Maximum no. of iterations (default = 20)

dx	Initial search interval	integ	Search trajectory calculation option (default = 1)
tol	Convergence accuracy (default = 1.0×10^{-6})	print	Print option (default = 0)

The search increment, *dx*, is an indication of how well the initial estimate is known. The values *(xest – dx)* and *(xest + dx)* should bracket the solution. If *dx* is small, the initial estimate is assumed to be close to the actual solution.

The search convergence accuracy, *tol*, controls when the search converges. As the convergence accuracy is reduced, more search iterations are required for convergence.

The reference factors, *xref* and *fref*, are chosen such that the ratios $|x/xref|$ and $|f(x)/fref|$ are between 1 and 10. In the first example above, the search parameter *x* is the angle of attack and has a value between 0.5 and 10.0, so a reference value of 1.0 is acceptable. The search function *f(x)* is the final altitude on a segment, and its value is going to be 30,000 when the search converges. A reference value of 10,000 is input to scale this large number to between 1 and 10.

The variable *maxitr* is the maximum number of search iterations or search attempts. If it is set to zero, only one trajectory is calculated using the initial estimate of the search value.

The variable *integ* controls whether all trajectories within a search loop are recomputed (a value of 1), or whether only the search trajectory is recomputed (a value of 0). A value of one is safe in that all trajectories are recomputed each search iteration; however, this may not be necessary and it can take significantly more computation time than the alternative.

The variable *print* is used to control trajectory printout during the search. A value of zero means that only a summary of the search is printed, and a value of 1 prints every trajectory during a search. When *print = 1*, a large amount of printout is produced so it should only be used when problems occur with a search loop.

Search Loops

A search is defined as starting at the beginning of the segment containing the search parameter. If there is more than one search parameter, the first segment encountered that has a search parameter is used to start the search. The search extends to the segment given by the search objective function. This part of the trajectory is recomputed each iteration of the search.

Multiple searches can be defined independently of each other as shown in the left diagram of Figure 4–11. They can also be nested so that one search is entirely within another search as shown in the middle diagram. But searches cannot be defined that intersect each other as shown in the right diagram.

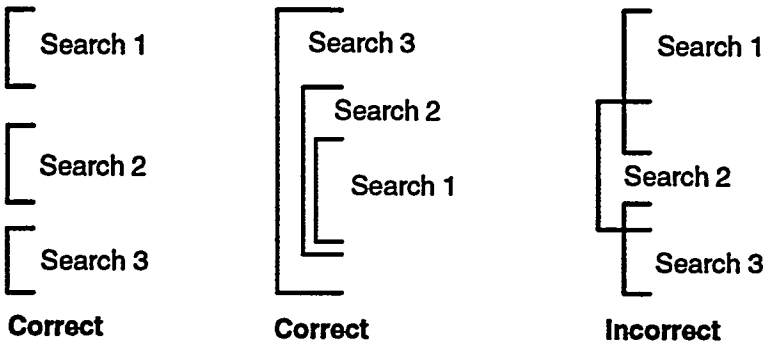

Figure 4–11. Valid Multiple Search Structures.

For nested search loops, the looping structure can be clarified by using dummy segments (segments with a *tseg* > *0* final condition). These segments do not affect the trajectory because they are of zero length; however, they can be used for evaluating search objective functions.

4.4.10. *Summarize Data Block

Summary variables are used in conjunction with surveys to perform tradeoff and sensitivity studies. A survey systematically changes the value of one or more input variables, and for each set of values, trajectories are calculated (Section 4.4.11). Summary variables are calculated after each set of trajectories is completed. When values are surveyed, summary variables can be tabulated as a function of the survey variables.

```
Summary Variables:

    qe      payload      vel_bo      max_alt       range

75.000       2000.0      5979.9       560694      178.38
75.000       3000.0      4975.7       390018      131.45
75.000       4000.0      4252.3       285922      101.79
75.000       5000.0      3702.0       216962       81.42
75.000       6000.0      3270.9       168834       66.64

80.000       2000.0      6042.7       652736      133.66
80.000       3000.0      5022.0       460668       99.95
80.000       4000.0      4282.3       342741       78.58
80.000       5000.0      3716.8       264181       63.91
80.000       6000.0      3270.7       209022       53.31
```

Figure 4–12. Summary Variable Printout.

Figure 4–12 contains an example printout of summary variables for a problem that calculates sounding rocket trajectories for different payload weights and launch angles. The survey variables are the launch elevation angle, *qe*, and the payload weight, *payload*. The summary variables are the burnout velocity, *vel_bo*, the maximum altitude or apogee, *max_alt*, and the final range, *range*. Each line of the summary printout represents a trajectory. In this case, ten trajectories have been calculated for five payload weights and two launch angles.

For each trajectory (or set of trajectories for a multi–vehicle problem), a summary variable has a single value. This value is calculated from a set of input commands, given in a **summarize* data block, representing math operations. The math operations work the same way as in table files (Section 3.5).

For example, the first summary variable shown above, *vel_bo*, is defined with the data block

```
*summarize  vel_bo
   add vel on segment 3, trajectory 1
```

The summary variable name, *vel_bo*, is given first, followed by a single math operation. Summary variable names do not have to be unique; they are only used for the column headings in the printout file.

Summary values are initialized to zero and then the math operations are executed. In this case, the velocity at the end of segment 3 on trajectory 1 is added to zero. Only one math operation is given, so the calculation is finished. This example is typical

of summary variables; they are usually used to retrieve a specific value from a trajectory rather than perform a complex sequence of math operations.

The two other summary variables in Figure 4–12 are defined with the data blocks

```
*summarize  max_alt
   add max(alt[1])

*summarize  range
   add last(range) trajectory 1
```

Max, Min, First, and Last Special Functions

These data blocks illustrate some special functions that are available for summary variables. The *max* function, used to retrieve the maximum altitude, searches the entire trajectory for the maximum value of a variable. The *last* function retrieves the last or final trajectory value. Other similar functions are *min* and *first*. These functions eliminate the need for segment numbers because they apply to the entire trajectory.

Trajectory data is saved in memory at the beginning and end of each segment and at every print interval. When the special functions *max* and *min* scan the trajectory data for maximum and minimum values, they only look at the saved trajectory data. They select the maximum or minimum saved value and then use the two adjacent values to fit a parabola and estimate the actual minimum. Thus, the accuracy of the result depends on the print interval. If the value changes rapidly, the print interval may need to be reduced to obtain more accurate results.

Normally segment and trajectory numbers are required when any variable is retrieved from a trajectory. Segment numbers are given first following the keywords *on segment*. Trajectory numbers are given next following the keyword *trajectory*, as shown in the first example for *vel_bo*. Trajectory numbers can also be given as subscripts enclosed in square brackets as shown in the definition for *max_alt*.

Math Operations

All of the examples shown above use the *add* math operation, but there are many others as shown in Table 4–23. The first set of operations (*add* through *iexp* in the first column of the table) require an additional value, in other words, they operate on the current summary value and an additional input value. This value can be an output variable from a trajectory as shown above or it can be a constant. The remaining math operations (*abs* through *atan*) do not require an additional value. They only operate on the current value of the summary variable.

Table 4–23. Summary Variable Math Operations.

Operation	Description	Operation	Description
add	Add a value to the summary value	sqrt	Square root of summary value

sub	Subtract a value from the summary value	ln	Natural log of summary value
mult	Multiply a value by the summary value	log	Log base 10 of summary value
div	Divide summary value by summary value	e	*e* raised to the summary value
idiv	Divide value by the summary value	sin	Sine of summary value (deg)
exp	Take summary variable to a power given by a value	cos	Cosine of summary value (deg)
iexp	Take value to the power given the summary value	tan	Tangent of summary value (deg)
abs	Absolute value of summary value	asin	$\sin^{-1}$ of summary value (deg)
neg	Negate summary value	acos	$\cos^{-1}$ of summary value (deg)
sqr	Square summary value	atan	$\tan^{-1}$ of summary value (deg)

A **summarize* data block can include any number of math operations, although most summary variables are defined with only one or two operations. The following example, which calculates the difference in final range between two trajectories, has 3 math operations:

```
*summarize  delta_rng
   add last(range[1])
   sub last(range[2])
   mult 1.852
```

The summary variable, *delta_rng*, is initialized to zero. The first operation adds the final range from trajectory 1 to the summary value which is zero. The next operation subtracts the final range from trajectory 2 resulting in the difference between the two ranges. The last operation converts the units of range from nautical miles to kilometers.

The previous example used the *last* function to select the final range so no segment numbers were given. The following example selects altitudes from within two trajectories and calculates the difference:

```
*summarize delta_alt
   add alt on segment 4, trajectory 1
   sub alt on segment 11, trajectory 2
   div 3280.84
```

This example calculates the difference between the altitude at the end of segment 4 on trajectory 1 and the altitude at the end of segment 11 on trajectory 2 in kilometers. Segment numbers are required because the special functions (*max*, *min*, *first* and *last*) are not used, and trajectory numbers are given with the *trajectory* keyword to illustrate this style.

4.4.11. *Survey Data Block

The purpose of a survey is to change the value of one or more input parameters and determine the effect on the trajectories. This is called a tradeoff study or sensitivity study. Trajectories are computed for all values of the survey variable.

As noted in Section 4.1, any numerical value in the problem file can be systematically varied by defining a survey. For example,

```
*initial   alt=surv-1  lat=0  long=0  wt=1200
```

has the variable *alt* set to the value of survey one. The keyword *surv–1* is a placeholder for the actual value that will be used when the problem is run. The actual value is determined from the corresponding **survey* data block. The survey numbers are used to relate the survey keywords to the survey data blocks, so *surv–1* corresponds to the **survey 1* data block, *surv–2* corresponds to the **survey 2* data block, and so on.

The *surv–1*, *surv–2*, etc., keywords control which variables in the problem file are modified by the survey. The **survey* data blocks control the values assigned to these variables. In the above example, the variable *alt* in the **initial* data block is set to *surv–1*, so there must be a corresponding **survey 1* data block in the problem that assigns values for the initial altitude.

For example,

```
*survey  1  alt0  lo=40000  hi=80000  inc=20000
```

says to run three cases with the initial altitude set to 40,000, 60,000, and 80,000 ft. TAOS substitutes the survey value everywhere the *surv–n* keyword is found in the problem file. So more than one variable in the problem file can be set to the survey value.

Survey Numbers and Names

The survey data block begins with the **survey* data block name and the survey number. Survey numbers are normally assigned in sequence beginning with one. Any number of survey loops can be defined. The surveys form nested loops with the lowest–numbered survey the inner most and the highest–numbered survey the outer most.

A survey variable name follows the survey number. This name must be unique from other output variable names. It is used in the EGS database files for plotting the results of a survey.[4] In the example above, the EGS database would contain one or more trajectories associated with *alt0=40000*, another set of trajectories associated with *alt0=60000*, and so on.

Lo, Hi, and Increment Survey Values

There are two ways that survey values can be assigned. The example above shows one method where the values take on incremental values. A starting value is given,

called the low or *lo* value. A set of trajectories is always run first with survey variables set to their starting values. After trajectories are computed for the low value, the low value is incremented by the value of *inc*. If the incremental value is zero, the trajectories are only run for the low value.

After the survey value is incremented, it is compared to the high or *hi* value. If it exceeds the high value, then it is set equal to the high value and a flag is set indicating that this is the last case. Then another set of trajectories is computed. This process continues until the last case, or high value, is reached.

Specific Survey Values

Another way of specifying the survey values is to enter them directly. For example,

```
*survey  1  alt0  vals=30000,60000,75000
```

runs three cases with the initial altitude set to 30,000, 60,000, and 75,000 ft. This type of input is used when the desired values do not fit the pattern of the first method (*lo*, *hi*, and *inc*).

The two methods can be combined as follows:

```
*survey  1  alt0  lo=40000,  hi=80000,  inc=20000
                  vals=35000,45000
```

This survey defines five values: 35000, 40000, 45000, 60000, and 80000. The values defined by the *lo*, *hi*, and *inc* variables are run along with the specific values given by the *vals* variable. The extra values given by the *vals* variable are inserted in the survey in order from lowest to highest value, so the first set of trajectories is run for *alt0=35000*, the next set is run for *alt0=40000*, etc.

4.4.12. *Title Data Block

A problem title, consisting of one or more lines of text, is optional. If a title is supplied, it is printed at the top of each page of printout and at the beginning of EGS database files. A title can be used to help identify or document a problem. There is no limit on the number of lines in a title; although the printout begins to look cluttered with more than 5 or 6 title lines.

Titles are input as follows:

```
*title   This is the first line of the title
         this is the second line of the title
         and so on, until a data block name is hit

*atmos   standard
```

The title begins with the first nonblank character following the **title* keyword, and extends to the next valid data block keyword. In the example above, the title extends from the word *This* to just before the **atmos* keyword; thus, a title can contain any text except for data block keywords. All data block keywords begin with an asterisk, so difficulties can be avoided by not putting asterisks in front of a word in the title.

4.4.13. *Units/Fmt Data Block

All output variables in TAOS have default units and printout formats given in the appendix. These can be changed with the **units/fmt* data block. When the units of a variable are changed, they are changed everywhere, that is, both on input and output. For example, if the units of altitude are changed from *ft* to *km*, then everywhere altitude is referenced in the table, problem, and printout files, it will be given in *km* instead of *ft*.

Units

Units are changed by giving the variable name and its new unit. Allowable units are given in Table 4–24. Units must be consistent; in other words, if the variable has a distance unit such as *ft*, then it can only be changed to other distance units such as *nm*, *m*, or *km*.

Table 4–24. Allowable Units.

Units	Description	Units	Description
ft	Feet	m/sec2	Meters/second^2
in	Inches	m/min2	Meters/minute^2
mi	Statute miles	m/hr2	Meters/hour^2
nm	Nautical miles	km/sec2	Kilometers/second^2
m	Meters	km/min2	Kilometers/minute^2
km	Kilometers	km/hr2	Kilometers/hour^2
sec	Seconds	deg/sec2	Degrees/second^2
min	Minutes	deg/min2	Degrees/minute^2
hr	Hours	deg/hr2	Degrees/hour^2
deg	Degrees	rad/sec2	Radians/second^2
rad	Radians	rad/min2	Radians/minute^2
ft/sec	Feet/second	rad/hr2	Radians/hour^2
ft/min	Feet/minute	rev/sec2	Revolutions/second^2
ft/hr	Feet/hour	rev/min2	Revolutions/minute^2
in/sec	Inches/sec	lb	Pounds mass
in/min	Inches/min	slugs	Slugs
in/hr	Inches/hour	gm	Grams
mi/sec	Statute miles/second	kg	Kilograms
mi/min	Statute miles/second	lb/sec	Pounds mass/second
mi/hr	Stature miles/hour	lb/min	Pounds mass/minute
knots	Nautical miles/hour	lb/hr	Pounds mass/hour
m/sec	Meters/second	slugs/sec	Slugs/second
m/min	Meters/minute	slugs/min	Slugs/minute

m/hr	Meters/hour	slugs/hr	Slugs/hour
km/sec	Kilometers/second	g/sec	Grams/second
km/min	Kilometers/minute	g/min	Grams/minute
km/hr	Kilometers/hour	g/hr	Grams/hour
deg/sec	Degrees/second	kg/sec	Kilograms/second
deg/min	Degrees/minute	kg/min	Kilograms/minute
deg/hr	Degrees/hour	kg/hr	Kilograms/hour
rad/sec	Radians/second	lbf	Pounds force
rad/min	Radians/minute	n	Newtons
rad/hr	Radians/hour	kn	Kilonewtons
rev/sec	Revolutions/sec	lbf/ft2	Pounds force/inch^2
rpm	Revolutions/minute	psi	Pounds force/inch^2
g	G's	pascal	Pascals
ft/sec2	Feet/second^2	kpascal	Kilopascals
ft/min2	Feet/minute^2	1/in	Per inch
ft/hr2	Feet/hour^2	1/ft	Per foot
in/sec2	Inches/sec^2	1/m	Per meter
in/min2	Inches/min^2	ft2/sec	Feet^2/second
in/hr2	Inches/hour^2	m2/sec	Meter^2/second
mi/sec2	Statute miles/second^2	lb/ft3	Pounds mass/feet^3
mi/min2	Statute miles/minute^2	lb/m3	Pounds mass/meter^3
mi/hr2	Statute miles/hour^2	g/cm3	Grams/centimeter^3
nm/hr2	Nautical miles/hour^2	kg/m3	Kilograms/meter^3

The aerodynamic reference area S_{ref} is not a standard output variable, but its units can be changed with the **units/fmt* data block. Its default units are *ft* (ft^2), but these units can be changed by giving a different distance unit. For example,

```
*units/fmt  sref  in
```

changes the units of S_{ref} to in^2.

Output Format

Printout format only affects output files, but it affects all output files including plot files and EGS database files. It is given by the letter *e* or *f*, followed by a period, followed by a number. The letter tells whether the variable is printed in e–format or f–format. E–format is commonly used in computers for scientific notation; for example, the number 250 is printed as 2.50e02 where e02 means $\times 10^2$. F–format prints numbers the standard way, that is, 250 is printed as 250. The number following the period tells how many digits are printed following the decimal point.

The data block

```
*units/fmt    alt     km    f.3
              vel     m/s   f.2
              range   km
              mach    e.5
```

changes the units or output format for four variables. The units of altitude are changed to *km* and it is printed with three decimal places. Velocity is changed to *m/s* and is printed with two decimal places. The units of range are changed to *km*, and the default format is used. The units of Mach number are not changed (it is nondimensional), but it is printed with five decimal places in scientific notation.

Any number of variables can be changed within the data block. The variable's units can be changed, its output format can be changed, or both can be changed. If both the units and the output format are changed, the new units must be given first, followed by the new output format as shown in the example above.

The output format of user–defined variables can be controlled with the **units/fmt* data block as shown above. But the units for a user–defined variable cannot be changed because TAOS does not know the original units of the variable. User–defined variables must be computed with the desired units in the **define* data block.

4.4.14. *Wind Data Block

Atmospheric winds with respect to the earth's surface can be defined with a **wind* data block. If a **wind* data block is not input, then all wind velocities are set to zero. Winds are divided into a vertical component and a horizontal component which can be defined as a wind velocity and heading or as east and north components. Winds can be defined in either the local geocentric or geodetic horizon coordinate system.

Wind heading is the direction the wind is from measured clockwise from north as shown in Figure 4–13. For example, a heading of 180° means the wind is from the south, that is, it moves from south to north. Similarly, the wind east component is positive when the wind is from the east or moving from east to west. The wind vertical component is positive when the air movement is towards the ground or downward.

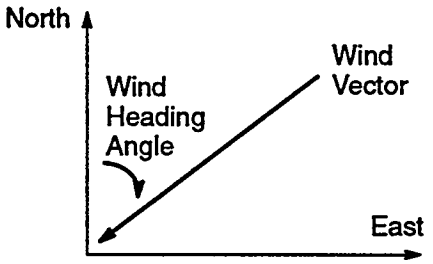

Figure 4–13. Wind Horizontal Component Direction.

The wind data block begins with the keyword **wind* followed by the keyword *geocentric* or *geodetic* giving the coordinate system. Three variables and their values from Table 4–25 are given next. The three variables in the first column can be given, or the ones in the second column can be given, but not both.

Table 4–25. Wind Data Block Variables.

Variable	Description	Variable	Description
windv	Total horizontal wind velocity	winde	East wind velocity component
windh	Horizontal wind heading	windn	North wind velocity component
windd	Wind velocity component towards earth center	windd	Wind velocity component towards earth center

Wind values can be constants or they can be table identification names enclosed in parentheses. TAOS keys on the parentheses to decide whether a table identification name or a constant has been input. Winds that are a function of a state variable, such as altitude or time, can be input with tables.

The following wind data block

```
*wind  geodetic  windv=(winds)  windh=90  windd=0
```

says the wind horizontal velocity is interpolated from a table named *winds*, the wind is from the east (it moves east to west) and there is no vertical wind component. The winds are defined in the local geodetic horizon coordinate system.

4.5. Examples

The previous sections have given detailed descriptions of each data block that can be used in a problem file, but it is not obvious how to put the data blocks together to calculate trajectories. So this section presents four example problems showing how to set up a complete trajectory problem. It also illustrates some of the capabilities in TAOS.

The first example calculates a set of simple ballistic reentry trajectories and it shows how surveys and summary variables can be used for tradeoff studies. Although this is a simple problem, it shows the overall structure of a problem file. Problems for more complex trajectories follow the same structure.

The second example calculates the trajectory of a rail–launched ballistic missile and uses optimization to solve for trajectory constraints. This problem illustrates multiple trajectories, multiple segments, and simple use of optimization.

The third example computes an intercept of a ballistic missile from long range with optimization. This is a more complex problem where optimization is used to shape the trajectory to hit the target.

The last example is also an intercept, but one of the intercept guidance algorithms is used instead of optimization to calculate the intercept trajectory.

The processing time or run time required for each problem is given for a Silicon Graphics Indigo–2 workstation with an 200Mhz R4400 processor.

4.5.1. Ballistic Reentry

One of the simplest problems for TAOS is a ballistic reentry. Ballistic flight occurs when all of the control variables (angle of attack, angle of sideslip, bank angle, and power setting) are zero so no guidance rules are required. If the vehicle's configuration is assumed to remain the same throughout reentry, its trajectory can be modeled with one segment that starts from the entry point and continues to impact.

Table File

The only forces acting on the vehicle are from aerodynamic drag and gravity so only a drag table is required. For conceptual design drag is often assumed to be only a function of Mach number because altitude effects are usually small. The following simple table file contains the axial force coefficient as a function of Mach number:

```
(ca_ex_1)

   table   ca(mach)   sref=2.1817

         mach    =   6.000,  8.000, 10.000, 12.000, 14.000,
                    16.000, 18.000, 20.000
         ca      =  0.0950, 0.0742, 0.0643, 0.0587, 0.0553,
                    0.0528, 0.0513, 0.0503
```

Problem File

The following problem file is used to calculate a set of ballistic reentry trajectories:

```
(ballistic)

*title   Ballistic reentry trajectories

*atmos  standard
*earth  wgs-84  omega=0

*trajectory  1  RV  start on 1

 *initial  geodetic
   long=0.0       lat=0.0        psi=90.0        time=0.0
   alt=300000     vel=surv-2     gama=surv-1     wt=550.0

 *print  time alt range vel dynprs nx

 *segment 1  Ballistic
   *integ dtprnt=0.50  dt=0.10
   *aero ca=(ca_ex_1)
   *when  alt<0  stop

*survey  1  gentry   vals=-20,-25,-30,-35
*survey  2  ventry   lo=15000  hi=18000  inc=1000
```

```
*summarize  max_q
   add max(dynprs[1])

*summarize  max_g
   add min(nx[1])
   neg

*egs  ex1.dbf  time alt range vel dynprs nx

*end
```

Like all problems, it begins with a name enclosed in parentheses. This is followed by data blocks that provide a title, an atmosphere model, and an earth model.

This problem involves only one vehicle so a single trajectory has been defined that starts with segment number 1. The trajectory definition extends from the **trajectory* keyword to the first **survey* keyword. Data blocks within the trajectory definition have been indented to show the problem structure.

Trajectory Definition and Surveys

Trajectories require initial conditions and at least one segment definition. Since this problem has only one trajectory, specific initial conditions must be given. It cannot be initialized from another trajectory because no other trajectories exist.

This set of trajectories is generic in that the location of the trajectory on the earth's surface and the direction of the trajectories is not important. Thus, the initial latitude and longitude can be set to zero and an initial heading angle of 90° (east) can be chosen. This is typical of trajectories calculated during conceptual design.

Two of the initial conditions, velocity and flight path angle, are set to survey variables. The **survey* data blocks near the end of the problem provide values for these variables. Survey number 1, which varies the initial flight path angle, is the inner loop and has a set of specific values. Survey number 2, which varies the initial velocity, is the outer loop. Its values have been given in the "*lo, hi, inc*" form so initial velocity takes on the values 15,000, 16,000, 17,000, and 18,000 ft/sec.

In general, trajectories must have a **print* data block giving a list of variables that will be written to the printout file. Often this is the same list of variables written to the EGS database file.

This trajectory only requires one segment. The **integ* data block is used to enter an integration step size and a print interval, and an **aero* data block is used to provide the name of the drag table. The segment final condition ends the trajectory when the altitude becomes less than zero.

Summary variables have been defined for the maximum dynamic pressure, *max_q*, and the maximum axial g loading, *max_g*. Even though there is only one trajectory, trajectory numbers are still required in the summary variable definitions. In this case, the trajectory numbers are given as subscripts enclosed in brackets. The vehicle is

decelerating giving negative axial g's so the *min* function is used to select the most negative value and then the *neg* operation is used to convert this to a positive number.

The **egs* data block is used to create an EGS database file so the trajectory information can be plotted. The names given in the **survey* data block, *gentry* and *ventry*, appear in the EGS database file and are used for plotting.

Results

This problem calculates 16 trajectories, one for each combination of survey values. On a Silicon Graphics workstation, the problem runs in about 9 seconds so it takes a little over 0.5 seconds to compute each trajectory.

The following listing is typical of the trajectory printout for this problem:

```
Sandia National                          Trajectory Analysis & Optimization Software
Laboratories                                        (TAOS - Version 96.0)
-------------------------------------------------------------------------------------------------

        Ballistic reentry trajectories

        Problem (ballistic)  /  1976 US Standard Atmosphere  /  Non-rotating WGS-84 Earth

        Trajectory for vehicle:  rv

        ventry = 15000          gentry = -35

      Time          Alt         Range          Vel        Dynprs          Nx

     0.000    300000.0         0.000     15000.00         0.575     -0.0001
     0.500    295695.2         1.011     15008.98         0.729     -0.0002
     1.000    291384.3         2.023     15017.97         0.922     -0.0002
     1.500    287067.3         3.034     15026.96         1.167     -0.0002
     2.000    282744.3         4.046     15035.97         1.478     -0.0003
     2.500    278415.3         5.058     15044.99         1.847     -0.0004
     3.000    274080.2         6.070     15054.02         2.298     -0.0005
     3.500    269739.0         7.083     15063.06         2.852     -0.0006
     4.000    265391.8         8.096     15072.10         3.532     -0.0007
     4.500    261038.6         9.108     15081.16         4.362     -0.0009
     5.000    256679.2        10.121     15090.22         5.375     -0.0011
     5.500    252313.8        11.135     15099.29         6.609     -0.0014
     6.000    247942.4        12.148     15108.36         8.107     -0.0017
     6.500    243564.9        13.162     15117.44         9.924     -0.0021
     7.000    239181.3        14.176     15126.53        12.122     -0.0026
     7.500    234791.7        15.190     15135.61        14.763     -0.0031
     8.000    230396.0        16.204     15144.70        17.860     -0.0038
     8.500    225994.2        17.219     15153.79        21.547     -0.0046
     9.000    221586.4        18.234     15162.87        25.925     -0.0055
     9.500    217172.5        19.248     15171.95        31.109     -0.0066
    10.000    212752.5        20.264     15181.02        37.234     -0.0080
    10.500    208326.6        21.279     15190.08        44.454     -0.0095
    11.000    203894.5        22.294     15199.12        52.946     -0.0114
    11.500    199456.4        23.310     15208.14        62.912     -0.0136
    12.000    195012.3        24.326     15217.14        74.584     -0.0161
    12.500    190562.2        25.342     15226.10        88.224     -0.0191
    13.000    186106.1        26.359     15235.01       104.134     -0.0226
    13.500    181644.0        27.375     15243.88       122.654     -0.0267
    14.000    177175.9        28.392     15252.69       144.171     -0.0314
    14.500    172701.8        29.409     15261.43       169.124     -0.0369
    15.000    168221.8        30.426     15270.08       198.292     -0.0434
    15.500    163735.9        31.443     15278.63       235.270     -0.0515
    16.000    159244.1        32.461     15287.04       279.221     -0.0611
    16.500    154746.5        33.478     15295.30       332.102     -0.0726
    17.000    150243.2        34.496     15303.36       400.575     -0.0874
```

This page corresponds to the first trajectory computed by TAOS which is for an initial velocity of 15,000 ft/sec and an initial flight path angle of –35°.

The summary variables are tabulated as a function of the survey variables on the last printout page as follows:

```
Sandia National                    Trajectory Analysis & Optimization Software
Laboratories                                (TAOS - Version 96.0)
-------------------------------------------------------------------------------

        Ballistic reentry trajectories

        Problem (ballistic)  /  1976 US Standard Atmosphere  /  Non-rotating WGS-84 Earth

        Summary Variables:

    ventry      gentry       max_g        max_q

     15000      -35.000      34.481       129020
     15000      -30.000      31.951       117570
     15000      -25.000      28.690       105508
     15000      -20.000      25.567        93019

     16000      -35.000      37.581       147262
     16000      -30.000      34.986       133858
     16000      -25.000      31.498       119649
     16000      -20.000      27.428       104853

     17000      -35.000      41.352       166764
     17000      -30.000      37.914       151231
     17000      -25.000      34.284       134723
     17000      -20.000      29.700       117416

     18000      -35.000      45.204       187562
     18000      -30.000      41.533       169742
     18000      -25.000      36.976       150720
     18000      -20.000      31.966       130690
```

The trajectory data is plotted with EGS in Figures 4–14 through 4–17. The first two figures show four trajectories for different initial flight path angles and a constant initial velocity. The last two figures show the summary variables plotted as a function of the survey variables.

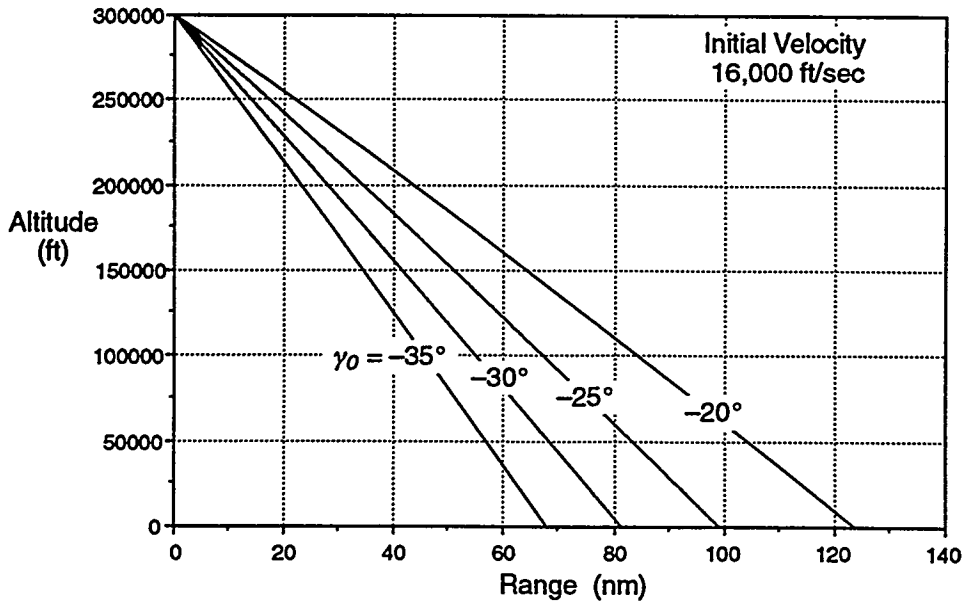

Figure 4–14. Altitude versus Range for Example Ballistic Trajectories.

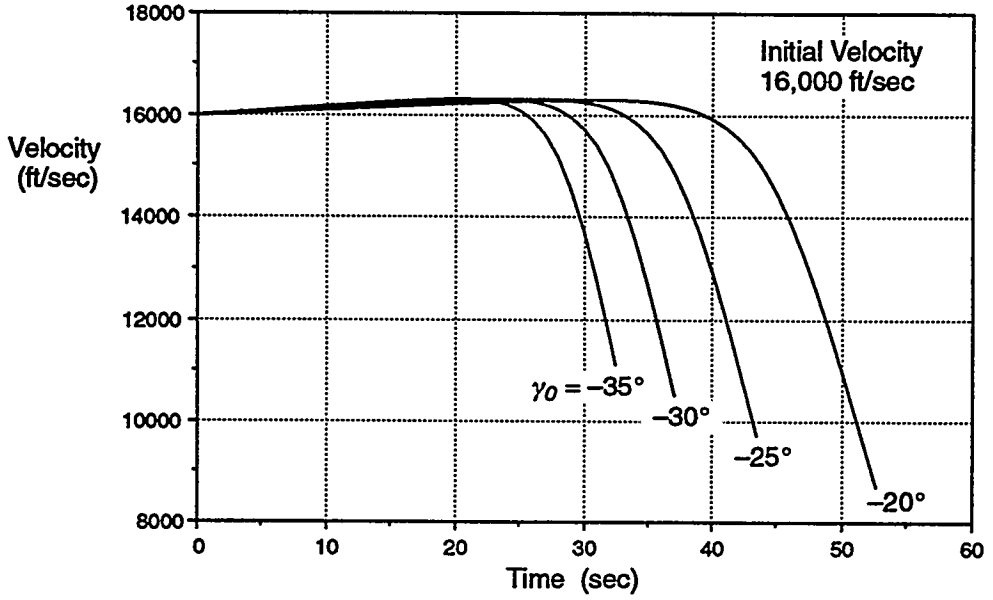

Figure 4–15. Velocity versus Time for Example Ballistic Trajectories.

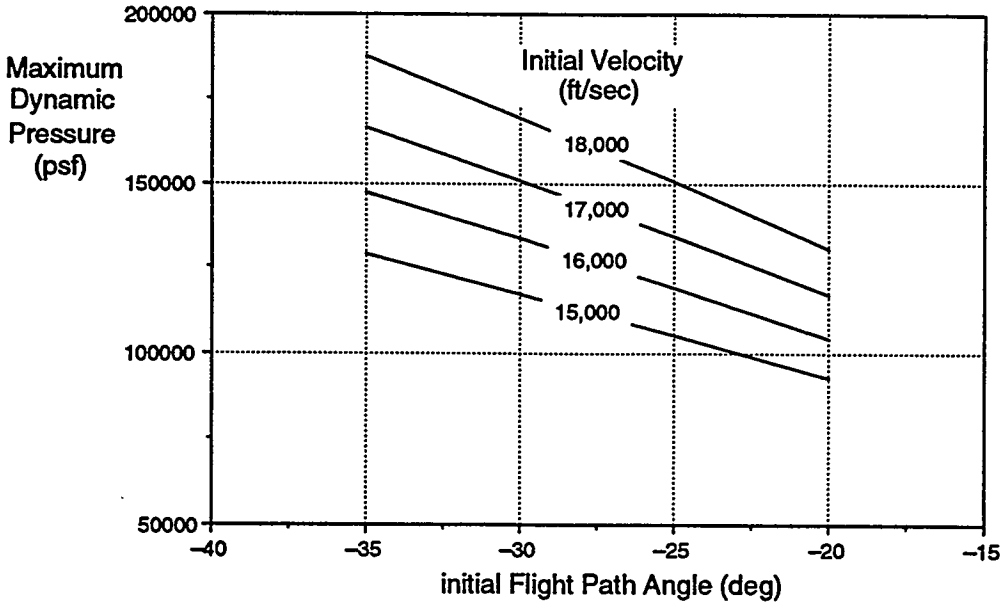

Figure 4–16. Maximum Dynamic Pressure for Example Ballistic Trajectories.

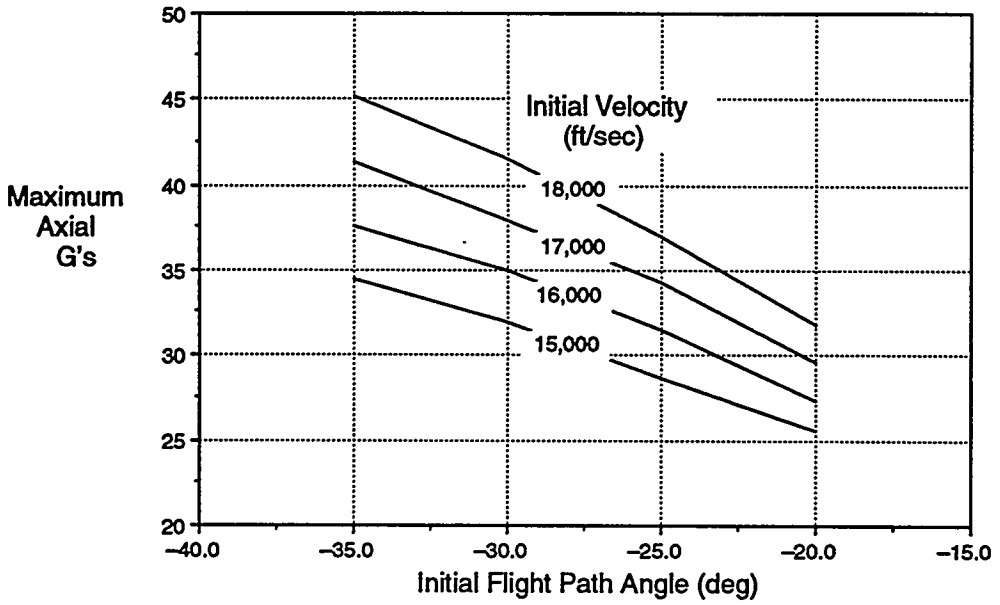

Figure 4–17. Maximum Axial G's for Example Ballistic Trajectories.

4.5.2. Ballistic Rocket

This example problem simulates a two–stage ballistic rocket trajectory which maximizes impact velocity. The rocket is rail launched, but the best launch–rail elevation angle is unknown. The second stage is fired downward after apogee to increase impact velocity, but the time of firing and the vehicle orientation at firing are unknown. Payload separation needs to occur above the atmosphere and second stage burnout needs to occur before this. A final constraint is that the range of the trajectory must be 60 nm. Optimization is used to solve for these unknown values to meet the constraints. The trajectory of the first–stage booster after separation is calculated in addition to the basic vehicle to illustrate multiple–vehicle calculations.

The first stage of the rocket has a Castor main motor with two smaller Recruit motors. The Recruits are jettisoned after they burnout at 3 seconds. The Castor motor continues to burn for about 40 seconds. The first stage has fins for aerodynamic spin stability so it is not separated until the vehicle is above the atmosphere which is assumed to be 300,000 ft. After first stage separation, the second stage is pointed with an attitude control system to its firing orientation. The vehicle coasts past apogee and then fires the second stage motor. Some time is allowed after second stage burnout to make sure the motor is no longer thrusting. Then, the payload separates and reenters the atmosphere.

Table files required for this problem are not given because they are too large. Tables are required for the thrust and mass flow of the first and second stage motors. Separate drag tables are required for all of the vehicle's configurations. Drag is different depending on whether a motor is firing or not, so separate tables are also required for boosting and coasting.

Problem File

The problem file appears as follows:

```
(rocket)

*title  Trajectory for a Ballistic Sounding Rocket
        2nd Stage Fires Down to Maximize Velocity

*atmos  standard
*earth  wgs-84  omega=0

*trajectory 1 Rocket start on 1

  *initial  geodetic
     long=0.0   lat=0.0    psi=90.0      time=0.0
     alt=0.0    vel=0.0    gama=opta-1   wt=12000.0

  *print  time alt vel range gamgd pitchgd wt

  *segment 1  1st stage ignition to first movement
    *integ dtprnt=0.1   dt=0.001
```

```
  *aero ca=(ca_1st_recruits)
  *prop thrust=(castor_thr)    mdot=(castor_mdt)
  *prop thrust=(recruits_thr)  mdot=(recruits_mdt)
  *rail launch cfstat=0.15  cfslid=0.01
  *when vel>0.001  goto 2

*segment 2  Rail launch
  *integ dtprnt=0.1   dt=0.01
  *aero ca=(ca_1st_recruits)
  *prop thrust=(castor_thr)    mdot=(castor_mdt)
  *prop thrust=(recruits_thr)  mdot=(recruits_mdt)
  *rail launch cfstat=0.15  cfslid=0.01
  *when plength>25.0  goto 3

*segment 3  Recruits & 1st stage burn
  *integ dtprnt=0.5  dt=.05
  *aero ca=(ca_1st_recruits)
  *prop thrust=(castor_thr)      mdot=(castor_mdt)
  *prop thrust=(recruits_thr)    mdot=(recruits_mdt)
  *when time>3.0  goto 4

*segment 4  Jetison recruits, continue 1st stage burn
  *integ dtprnt=0.5  dt=.10
  *increment wt=-350.0
  *aero ca=(ca_1st_stage)
  *prop thrust=(castor_thr)  mdot=(castor_mdt)
  *when  time>40.0  goto 5

*segment 5  Coast
  *integ dtprnt=1.0  dt=0.20
  *aero ca=(ca_1st_off)
  *when  alt>300000  goto 6
  *when  gamgd<0  goto 6

*segment 6  Stage and coast, allow time for orientation
  *integ dtprnt=1.0  dt=0.20
  *reset  wt=1650.0
  *aero ca=(ca_2nd_off)
  *when  tseg>30.0  goto 7

*segment 7  Orientation correct, coast until ready to fire
  *integ dtprnt=1.0  dt=0.20
  *aero ca=(ca_2nd_off)
  *fly  pitchgd = opta-2
  *fly  yawgd = *
  *fly  rollgd = *
  *when  tseg>opta-3  goto 8

*segment 8  Fire 2nd stage
  *integ dtprnt=1.0  dt=0.10
  *reset tmark=0
  *aero ca=(ca_2nd_stage)
  *prop thrust=(orbus_thr)  mdot=(orbus_mdt)
  *fly pitchi=*
  *fly yawi=*
```

```
    *fly rolli=*
    *when  tseg=41.5  goto 9

  *segment 9  Coast to payload separation
    *integ dtprnt=0.5  dt=0.20
    *aero ca=(ca_2nd_off)
    *when  alt<300000  goto 10

  *segment 10  Release payload & fly to impact
    *integ dtprnt=0.5  dt=0.10
    *reset  wt=350
    *aero ca=(rv)
    *when  alt<0  stop

*trajectory 2 Booster start on 1

  *initial  from trajectory 1, segment 6
  *print  time alt vel range gamgd pitchgd wt thrust nx dynprs

  *segment 1  Ballistic
    *reset wt=2100.0
    *integ dtprnt=0.5   dt=0.20
    *aero ca=(ca_1st_tumble)
    *when alt<0 stop

*optimize a for vel=max on segment 10, trajectory 1

   constrain  alt=450000 on segment 8
   constrain  range=60  on segment 10

   fref=10000        maxitr=30          tol=5.0e-7

   par-1 =  85.0     lo-1 = 80.0        hi-1  = 86.0
   par-2 = -90.0     lo-2 = -120.0      hi-2  = -70.0
   par-3 = 200.0     lo-3 = 150.0       hi-3  = 500.0

*egs  ex2.dbf
  time alt vel range gamgd pitchgd wt thrust nx dynprs

*end
```

The problem file begins, like all problems, with a name, title, atmosphere model, and earth model. The first trajectory simulates the entire flight and the second trajectory simulates the first stage booster as it falls to earth after separation.

Rail Launch

The initial conditions for the first trajectory position the vehicle at a zero longitude and latitude and start the vehicle heading east because a specific launch site has not been selected yet. The launch elevation angle has been set to an optimization parameter because the best value is unknown.

The first trajectory is divided into 10 segments. The first two segments simulate a rail launch. Two segments are used because the coefficient of friction changes instanta-

neously when the vehicle begins moving. The first segment simulates the time from ignition until the thrust overcomes the static friction force and gravity. The vehicle has just started to move at the end of the first segment. The second segment continues the rail launch to the end of the rail given by the *plength* final condition.

1st Stage Burn

The third segment computes the trajectory from the end of the launch rail to the time when the Recruit motors are jettisoned. During the first three segments, the Castor and Recruits are burning. The motor thrust and mass flow data for the Castor and the Recruits are in separate table files so two **prop* data blocks are used to interpolate data for both motors and add them together to get the total thrust.

The Recruits are jettisoned at the beginning of the fourth segment; the **increment* data block is used to account for the weight change. Since the vehicle's configuration changes, different drag tables must be used for this segment. The Recruit thrust and mass flow tables are no longer included because the motors have been jettisoned.

The vehicle coasts up to an altitude of 300,000 ft to get above the atmosphere in segment 5. No propulsion tables are given and the drag table has been changed for the power–off condition. An extra final condition, *gamgd<0*, has been included so the segment will terminate if the vehicle does not reach an altitude of 300,000 ft. This is a safeguard to prevent a possible infinite segment.

The first stage is separated from the vehicle at the beginning of segment 6. Segment 6 provides a margin of 30 seconds for the attitude control system to orient the vehicle in preparation for the second stage firing.

2nd Stage Burn

In segment 7 the vehicle is assumed to be oriented correctly for second stage firing. Although 30 seconds has been allowed for orientation in segment 6, this example assumes it occurs instantaneously between segments 6 and 7. In segment 6 the body is aligned with the velocity vector, whereas in segment 7 it is oriented at some angle relative to the inertial platform. This angle is unknown so it becomes an optimization parameter that will be varied. Furthermore, the vehicle must coast for an unknown amount of time before firing the second stage motor so the segment final time becomes another optimization parameter.

The second stage motor is fired in segment 8. The vehicle's inertial attitude is held constant in this segment with the *pitchi*, *yawi*, and *rolli* guidance rules. The asterisk values pick up the values of *pitchi*, *yawi*, and *rolli* from the end of the previous segment and use them in this segment.

Segment 9 accounts for the time between second stage burnout and payload separation. The minimum altitude for payload separation is 300,000 ft. Finally, the payload flies a ballistic trajectory to impact in segment 10.

Trajectory Number 2

The second trajectory gets its initial conditions from the vehicle's state at the beginning of segment 6 when first stage separation occurs. The second trajectory contains only one segment because it just tracks the empty first–stage motor from separation to impact. No significant events occur during this period.

Optimization

The **optimize* data block defines the objective function to be maximum velocity, and the trajectory is constrained to have a final range of 60 nm and a final altitude at the end of the second stage burnout (segment 8) of 450,000 ft. The objective function is normalized by a reference factor of 10,000 with the variable *fref* and a maximum of 30 iterations is specified. This problem only has three optimization parameters so it should not require many iterations to converge.

Initial estimates and upper and lower limits are given for each optimization parameter. Some preliminary runs were made with optimization turned off *(maxitr=0)* to determine initial estimates for the optimization parameters. These estimates do not have to be that accurate; they just need to result in a reasonable trajectory.

Results

The problem requires 12 iterations to converge and takes 5 minutes and 30 seconds to run on a Silicon Graphics workstation. The following listing shows the printout for the first part of the trajectory:

```
Sandia National                    Trajectory Analysis & Optimization Software
Laboratories                                (TAOS - Version 96.0)
-------------------------------------------------------------------------------------------

      Trajectory for a Strypi Sounding Rocket
      2nd Stage Fires Down to Maximize Velocity

      Problem (rocket)  /  1976 US Standard Atmosphere  /  Non-rotating WGS-84 Earth

      Trajectory for vehicle:  rocket

    Time          Alt          Vel          Range        Gamgd        Pitchgd          Wt

   0.000          0.0         0.00          0.000       86.000        86.000      12000.00
   0.003          0.0         0.00          0.000       86.000        86.000      11999.88

   0.003          0.0         0.00          0.000       86.000        86.000      11999.88
   0.100          0.9        24.03          0.000       86.000        86.000      11955.09
   0.200          5.0        58.78          0.000       86.000        86.000      11892.52
   0.300         12.6        93.20          0.000       86.000        86.000      11830.77
   0.400         23.6       127.64          0.000       86.000        86.000      11769.26
   0.410         24.9       131.12          0.000       86.000        86.000      11763.05

   0.410         24.9       131.12          0.000       86.000        86.000      11763.05
   0.500         38.1       162.51          0.000       85.920        85.920      11707.29
   1.000        163.6       340.29          0.002       85.638        85.638      11395.82
   1.500        374.7       504.53          0.005       85.467        85.467      11111.35
   2.000        654.4       603.99          0.008       85.335        85.335      10930.61
   2.500        971.3       666.74          0.013       85.216        85.216      10806.96
   3.000       1318.5       726.75          0.018       85.104        85.104      10687.55

   3.000       1318.5       726.75          0.018       85.104        85.104      10337.55
   3.500       1696.2       789.54          0.023       84.999        84.999      10219.84
```

4.000	2105.0	852.23	0.029	84.900	84.900	10102.67
4.500	2545.0	914.74	0.035	84.807	84.807	9986.03
5.000	3015.9	976.72	0.042	84.718	84.718	9869.69
5.500	3517.4	1037.38	0.050	84.633	84.633	9753.27
6.000	4048.3	1095.48	0.058	84.552	84.552	9636.77
6.500	4607.4	1151.07	0.067	84.473	84.473	9520.20
7.000	5194.1	1206.79	0.077	84.397	84.397	9403.54
7.500	5808.5	1263.11	0.087	84.324	84.324	9286.81
8.000	6451.1	1320.11	0.097	84.253	84.253	9170.00
8.500	7122.1	1377.80	0.108	84.185	84.185	9053.11
9.000	7822.0	1436.31	0.120	84.118	84.118	8936.14
9.500	8551.0	1495.79	0.133	84.054	84.054	8819.09
10.000	9309.9	1556.42	0.146	83.991	83.991	8701.97
10.500	10099.1	1618.37	0.159	83.930	83.930	8584.76
11.000	10919.4	1681.57	0.174	83.871	83.871	8467.48
11.500	11771.3	1745.98	0.189	83.813	83.813	8350.12

Figures 4–18 and 4–19 show plots of the two trajectories created from the EGS database file. The solid line represents the trajectory of the main vehicle and the dotted line represents the trajectory of the first–stage booster.

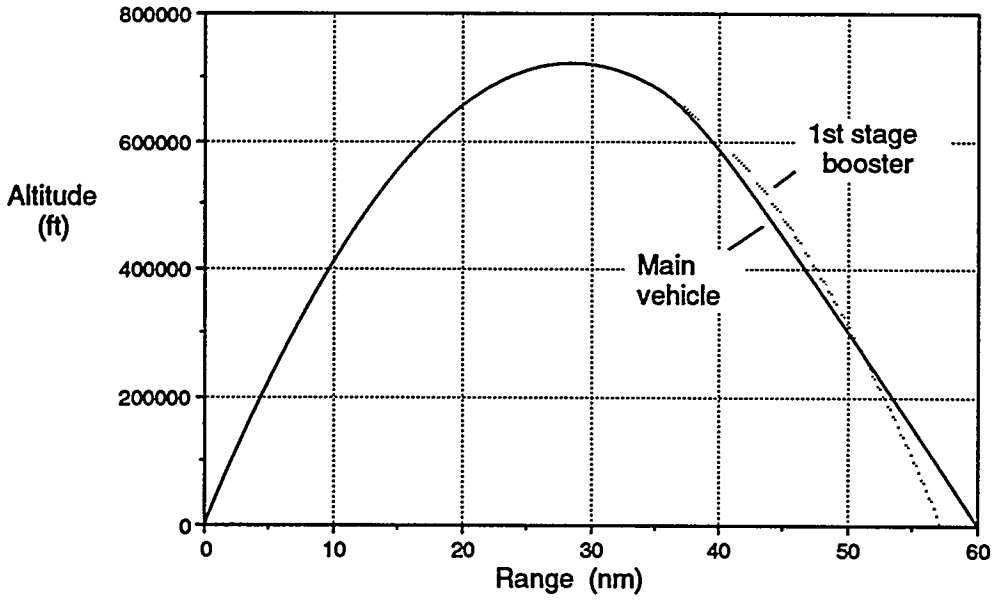

Figure 4–18. Altitude versus Range for Example Ballistic Rocket.

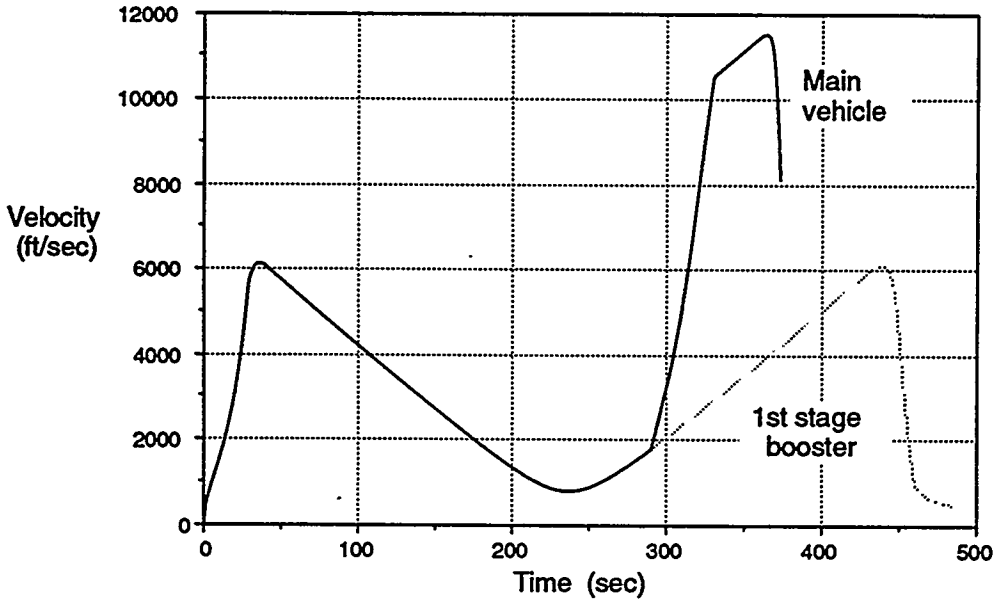

Figure 4–19. Velocity versus Time for Example Ballistic Rocket.

4.5.3. Air–Launched Intercept

This example simulates an air–launched interceptor flying against a single–stage ballistic missile target. The interceptor is a two–stage missile with a kill vehicle designed to hit the target just after its boost phase. In this simulation, all interceptor maneuvers are made while the two stages are firing; the kill vehicle coasts ballistically to intercept the target. The terminal guidance system on the kill vehicle is not simulated. The goals are to determine the interceptor standoff range or launch point and the velocity at intercept.

The target is launched at time zero. It is rail–launched for simplicity and flies ballistically. The interceptor is launched 25 seconds later and intercept is assumed to occur 90 seconds after target launch; thus, the interceptor flies 65 seconds. Angle of attack is used to steer the interceptor towards the target and it is assumed that a control system, such as thrust vector control, is capable of generating these angles of attack.

The table files are not listed because they are lengthy and do not help to understand the problem. Thrust and mass flow data for both stages of the interceptor and for the target are included in the tables. The interceptor requires drag and lift tables because it maneuvers with angle of attack. The target only requires drag tables because it flies ballistically.

Problem File

A listing of the problem file follows:

```
(intercept)

*title   Ballistic Missile Intercept

*atmos  standard
*earth  wgs-84  omega=0

#--------- Interceptor Trajectory ----------------------

*trajectory  1   Interceptor  start on 1

*define alt_km
    alt_km = alt/3280.84;
*define rng_km
    rng_km = range*1.852;

*dwn/crs long=0.0  latgd=0.0  azm=90.0

*initial  geodetic
    long=opta-13     lat=0.0         psi=-90.0
    time=25.0        range=0.0       gama=0.0
    alt=35000.0      vel=730.0       wt=1965.0

 *print time east alt vel gamgd alpha dynprs
```

```
*segment 1  Drop for 2 sec
  *integ dtprnt=0.50  dt=0.10
  *cg   cg=134.0
  *aero ca=(stage1_ca_off)  cn=(stage1_cn)
  *when  tseg=2  goto 2

*segment 2  1st stage burn
  *integ dtprnt=0.5  dt=0.10
  *cg   cg=119.8
  *aero ca=(stage1_ca_on)  cn=(stage1_cn)
  *prop thrust=(thrust1)  mdot(mdot1)
  *fly alpha vrs tseg
      opta-1   0.0
      opta-2   1.25
      opta-3   2.50
      opta-4   3.75
      opta-5   5.00
  *when  tseg=5.0  goto 3
  *when  alt<0  goto 8

*segment 3  Coast
  *integ dtprnt=0.5  dt=0.10
  *cg   cg=105.7
  *aero ca=(stage1_ca_off)  cn=(stage1_cn)
  *when  tseg=0.25  goto 4
  *when  alt<0  goto 8

*segment 4  Stage and coast
  *integ dtprnt=0.5  dt=0.10
  *reset wt=368.0
  *cg   cg=61.6
  *aero ca=(stage2_ca_off)  cn=(stage2_cn)
  *when  tseg=0.25  goto 5
  *when  alt<0  goto 8

*segment 5  2nd stage burn
  *integ dtprnt=0.5  dt=0.10
  *cg   cg=57.2
  *aero ca=(stage2_ca_on)  cn=(stage2_cn)
  *prop thrust=(thrust2)  mdot(mdot2)
  *fly alpha vrs tseg
      opta-6    0.0
      opta-7    2.0
      opta-8    4.0
      opta-9    6.0
      opta-10   8.0
      opta-11   10.0
      opta-12   13.1
  *when  tseg=13.1  goto 6
  *when  alt<0  goto 8

*segment 6  Coast and stage
  *integ dtprnt=0.5  dt=0.10
  *cg   cg=52.8
```

```
  *aero ca=(stage2_ca_off)  cn=(stage2_cn)
  *when  tseg=0.50  goto 7
  *when  alt<0  goto 8

 *segment 7  Separate kill vehicle and coast
  *integ dtprnt=0.5  dt=0.10
  *reset wt=80.0
  *cg   cg=32.3
  *aero ca=(payload_ca)  cn=(payload_cn)
  *when  time=80  goto 8
  *when  alt<0  goto 8

 *segment 8  Remove shroud and coast
  *integ dtprnt=0.5  dt=0.10
  *reset wt=70.0
  *cg   cg=18.0
  *aero ca=(kkv_ca)  cn=(kkv_cn)
  *when  time=90  stop
  *when  alt<0  stop

#------------ Target Trajectory -------------------------

*trajectory  2   Target  start on 21

*define alt_km
    alt_km = alt/3280.84;
*define rng_km
    rng_km = range*1.852;

*dwn/crs long=0.0  latgd=0.0  azm=90.0

*initial  geodetic
    long=0.0        lat=0.0        psi=90.0
    time=0.0        range=0.0      gama=79.0
    alt=0.0         vel=0.0        wt=13000.0

 *print time east alt vel gamgd alpha dynprs

 *segment 21  Launch
  *integ dtprnt=1.0  dt=0.01
  *aero ca=(target_ca_on)  cn=(target_cn)
  *prop thrust=(target_thrust)  mdot=(target_mdot)
  *rail launch  cfstat=0.15  cfslid=0.01
  *when  plength > 1000.0  goto 22

 *segment 22  1st stage burn
  *integ dtprnt=1.0  dt=0.10
  *aero ca=(target_ca_on)  cn=(target_cn)
  *prop thrust=(target_thrust)  mdot=(target_mdot)
  *when  time>75.0  goto 23

 *segment 23  Coast to intercept time
  *integ dtprnt=2.0  dt=0.10
  *aero ca=(target_ca)  cn=(target_cn)
  *when  time>90.0  goto 24
```

```
 *segment 24  Coast to impact
   *integ dtprnt=2.0  dt=0.10
   *aero ca=(target_ca)  cn=(target_cn)
   *when  alt<0  stop

#------------ Optimization ----------------------------

*optimize a for vel=max on segment 8, trajectory 1

   constrain alt[1] on segment 8 = alt[2] on segment 23
   constrain long[1] on segment 8 = long[2] on segment 23

   fref=10000        maxitr=60        integ=0

   par-1=9.9         lo-1=-10.0       hi-1=10.0
   par-2=9.9         lo-2=-10.0       hi-2=10.0
   par-3=9.9         lo-3=-10.0       hi-3=10.0
   par-4=8.4         lo-4=-10.0       hi-4=10.0
   par-5=3.7         lo-5=-10.0       hi-5=10.0
   par-6=3.7         lo-6=-10.0       hi-6=10.0
   par-7=-0.5        lo-7=-10.0       hi-7=10.0
   par-8=-2.0        lo-8=-10.0       hi-8=10.0
   par-9=-4.2        lo-9=-10.0       hi-9=10.0
   par-10=-5.0       lo-10=-10.0      hi-10=10.0
   par-11=-6.5       lo-11=-10.0      hi-11=10.0
   par-12=-8.3       lo-12=-10.0      hi-12=10.0
   par-13=3.50       lo-13=1.0        hi-13=6.0

*egs  ex3.dbf  time alt east alt_km rng_km vel

*end
```

Trajectory number 1 simulates the interceptor with 8 segments and trajectory number 2 simulates the target with 4 segments. This problem is set up similarly to previous examples so only the differences are discussed.

Optimization

The main feature of this problem is the use of optimization to calculate the shape of the intercept trajectory. Segments 2 and 5 in the first trajectory, representing the burn times of the first and second stages, have guidance rules with angle–of–attack time histories. All of the angle–of–attack values are optimization parameters. The optimization procedure varies these angle–of–attack values to achieve maximum velocity at impact. Maximum velocity is used instead of maximum range because it converges faster and gives similar results.

The other optimization parameter is the initial longitude of the interceptor. The target is launched from a zero longitude and latitude heading east. The interceptor is launched from zero latitude heading west, but its initial longitude is unknown. Its initial longitude needs to be as far east as possible and still intercept the target 90 seconds after launch.

The two constraints in the **optimize* data block force the intercept. These constraints force the longitude and altitude of the two vehicles to be the same at the intercept time of 90 seconds. The two segments referenced in the constraints have the same final condition of *time<90* seconds. Latitude is not constrained because both vehicles are traveling along the equator. The example has been set up as a two–dimensional problem so latitude is, by definition, the same for both vehicles.

The initial values for the optimization parameters are determined by using trial and error from some preliminary runs with optimization turned off. This is not too difficult because the starting trajectory does not have to be close to the final one; it just needs to have similar characteristics.

The optimization control variable *integ* can be set to zero in this problem to reduce the run time because the target trajectory is always the same. The intercept occurs at a fixed time of 90 seconds so the target trajectory only needs to be computed one time. It never changes so there is no need to recompute it each iteration during optimization.

Other Features

Two user–defined variables are defined, *alt_km* and *rng_km*, that calculate the altitude and range in kilometers instead of nautical miles. User–defined output variables are used to change the units of these two variables because use of the **units/fmt* data block changes the units for all references of altitude. The aerodynamic tables are a function of altitude and assume that altitude is given in feet. The initial altitude for trajectory 1 is also given in feet. If the **units/fmt data* block is used to change altitude to kilometers, these values would have to be changed to kilometers. The easiest solution is to calculate a user–defined variable and use it for the printout and plot files.

A common feature of multiple–trajectory problems is a **dwn/crs* data block in each trajectory that sets the reference point of the east/north coordinate system. For multiple–trajectory problems, the east/north coordinate system is ideal for showing the relationships between trajectories, but the same coordinate system must be used for all trajectories. The default location of this coordinate system is different for each trajectory (at the beginning of each trajectory) so this is changed with a **dwn/crs* data block to the same point.

Results

This problem converges after 50 iterations and requires 11 minutes and 30 seconds to run on a Silicon Graphics workstation. The following printout shows the first part of the interceptor trajectory:

```
Sandia National                    Trajectory Analysis & Optimization Software
Laboratories                                  (TAOS - Version 96.0)
-------------------------------------------------------------------------------------------------

        Ballistic Missile Intercept

        Problem (intercept)  /  1976 US Standard Atmosphere  /  Non-rotating WGS-84 Earth
```

Trajectory for vehicle: interceptor

Time	East	Alt	Vel	Gamgd	Alpha	Dynprs
25.000	159.002	35000.0	730.00	0.000	0.000	196.695
25.500	158.943	34996.0	729.41	-1.259	0.000	196.406
26.000	158.883	34984.0	729.17	-2.518	0.000	196.368
26.500	158.823	34964.0	729.28	-3.776	0.000	196.580
27.000	158.763	34936.0	729.74	-5.031	0.000	197.041
27.000	158.763	34936.0	729.74	-5.031	10.000	197.041
27.500	158.681	34917.3	1268.67	-0.035	10.000	595.968
28.000	158.554	34944.8	1842.92	3.752	10.000	1256.261
28.500	158.378	35049.1	2456.79	7.246	10.000	2223.641
29.000	158.153	35269.8	3114.68	10.914	10.000	3543.810
30.000	157.545	36269.4	4600.80	18.829	8.671	7431.352
30.500	157.159	37159.4	5470.27	22.496	7.343	10067.147
31.000	156.713	38380.8	6461.27	25.766	5.651	13247.085
31.500	156.199	39980.3	7619.95	28.215	3.596	17065.785
32.000	155.604	41995.9	9003.70	29.762	1.541	21634.940
32.000	155.604	41995.9	9003.70	29.762	0.000	21634.940
32.250	155.284	43109.6	8955.59	29.723	0.000	20293.170
32.250	155.284	43109.6	8955.59	29.723	0.000	20293.170
32.500	154.966	44215.9	8904.93	29.684	0.000	19029.550
32.500	154.966	44215.9	8904.93	29.684	0.000	19029.550
33.000	154.327	46429.8	9001.47	29.605	0.000	17489.631
33.500	153.680	48663.5	9107.63	29.528	0.000	16089.808
34.000	153.025	50919.2	9223.24	29.451	0.000	14813.096
35.000	151.688	55503.8	9481.16	29.231	-0.483	12571.639
35.500	151.003	57826.4	9620.92	28.956	-0.966	11584.664
36.000	150.306	60159.2	9767.44	28.569	-1.449	10680.508
37.000	148.869	64826.9	10080.34	27.495	-2.573	9101.865
37.500	148.127	67146.0	10245.75	26.800	-3.213	8402.093
38.000	147.369	69443.7	10417.14	26.013	-3.854	7761.076
39.000	145.795	73943.9	10780.37	24.246	-4.625	6674.944
39.500	144.979	76139.1	10976.73	23.368	-4.756	6221.524
40.000	144.143	78297.8	11183.90	22.505	-4.886	5818.652
41.000	142.405	82505.7	11632.88	20.827	-5.003	5141.478

Figure 4–20 shows a plot of the intercept trajectory. The interceptor can be launched approximately 300 km from the target launch point and still make the intercept. This distance is reduced if the target is not heading directly towards the interceptor or if the intercept must occur at an earlier time. The velocity at intercept is 13,450 ft/sec.

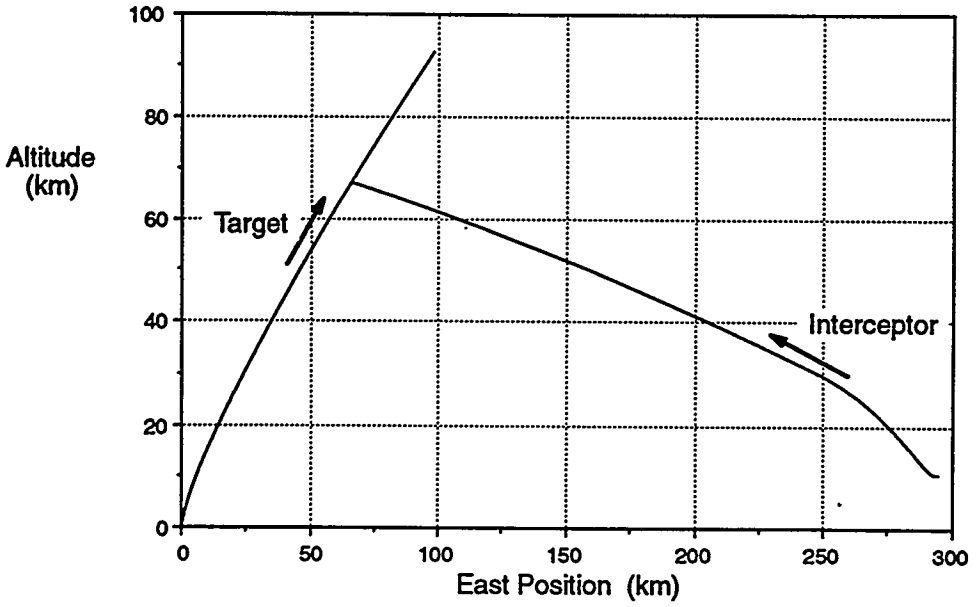

Figure 4–20. Altitude versus East Position for an Intercept Trajectory.

4.5.4. Ground–Launched Intercept

This problem simulates a ground–launched interceptor flying against a tactical ballistic missile target using the predictive guidance algorithm (Section 2.5). The target trajectory is computed from an altitude of 200,000 ft to impact. The interceptor is launched 30 seconds after the beginning of the target trajectory. This time is chosen so the target is within the interceptor's envelope. A listing of the problem file follows:

```
(intercept)

*title   Example intercept trajectory

*atmos  standard
*earth  wgs-84  omega=0

*trajectory  1  missile  start on 1

  *initial   geodetic
   long=0.0    lat=0.0      psi=90.0     time=30.0
   alt=0.0     vel=0.0      gama=75.0    wt=2000.0

  *dwn/crs long=0.0  latgd=0.0  azm=90.0

  *print  time alt east vel gamgd alpha relrng[2] relvel[2]

  *segment 1  Rail launch
    *integ dtprnt=0.5   dt=0.05
    *aero ca=(missile_on)  cn=(missile_cn)
    *prop thrust=25000  mdot=3.0
    *rail launch cfstat=0.12  cfslid=0.01
    *when plength>30  goto 2

  *segment 2  Ballistic to get up to speed
    *integ dtprnt=0.5   dt=0.10
    *aero ca=(missile_on)  cn=(missile_cn)
    *prop thrust=25000  mdot=3.0
    *when vel>2500  goto 3

  *segment 3  1st stage burn, intercept
    *integ dtprnt=0.5  dt=0.10  dtguid=1
    *limits alphat<15
    *aero ca=(missile_on)  cn=(missile_cn)
    *prop thrust=25000  mdot=3.0
    *fly intercept=2
    *when time>42.0  goto 4
    *when relvel[2]>0  stop

  *segment 4  coast & continue intercept
    *integ dtprnt=0.5   dt=0.10  dtguid=1
    *limits alphat<15
    *aero ca=(missile_off)  cn=(missile_cn)
    *fly intercept=2
    *when relvel[2]>0  stop
```

```
*trajectory  2  target  start on 10

  *initial   geodetic
   long=0.35     lat=0.0      psi=-90.0    time=0.0
   alt=200000   vel=4000     gama=-50.0   wt=1000.0

  *dwn/crs long=0.0  latgd=0.0  azm=90.0

  *print  time alt east vel

  *segment 10  Ballistic reentry
    *integ dtprnt=0.5   dt=0.10
    *aero ca=(rv)
    *when alt<0 stop

*egs ex4.dbf time alt east vel

*end
```

Trajectory number 1 is the interceptor missile and trajectory number 2 is the target missile. The interceptor is heading east and the target is heading west as in the previous problem. The **dwn/crs* data blocks are required in both trajectories to set the reference point for the east and north output variables.

The interceptor is rail launched and allowed to accelerate to a velocity of 2500 ft/sec before trying to intercept the target. The launch angle of 75° is an estimated value used to get the missile in the air.

The problem assumes constant thrust and mass flow for the interceptor missile so the values are entered directly in the **prop* data block rather than using table values. This illustrates that any table value can be replaced with a constant number for simple problems.

The *intercept* guidance rule is used in segments 3 and 4 to maneuver the missile to hit the target. The velocity of trajectory 2, the target, relative to trajectory 1, the interceptor, is the closing velocity given by the output variable *relvel[2]* within trajectory 1. The subscript indicates the relative velocity of trajectory number 2. When this velocity changes sign (when it is zero), the interceptor has reached its closest distance to the target so it is used as a final condition for segments 3 and 4.

Segment 3 also has a final condition of *time>42* which represents the burn time of the motor. The time value of 42 seconds does not look correct, but the missile is launched at a time of 30 seconds so the burn time is really 42 – 30 = 12 seconds.

Results

This problem requires 1.5 seconds to run on a Silicon Graphics workstation. The printout for the interceptor trajectory follows:

Sandia National Laboratories — Trajectory Analysis & Optimization Software (TAOS - Version 96.0)

Example intercept trajectory

Problem (intercept) / 1976 US Standard Atmosphere / Non-rotating WGS-84 Earth

Trajectory for vehicle: missile

Time	Alt	East	Vel	Gamgd	Alpha	Relrng[2]	Relvel[2]
30.000	0.0	0.000	0.00	75.000	0.000	107129.87	-4731.568
30.402	29.0	0.000	149.21	75.000	0.000	105196.07	-4885.690
30.402	29.0	0.000	149.21	75.000	0.000	105196.07	-4885.690
30.500	44.8	0.000	185.52	74.717	0.000	104716.17	-4923.407
31.000	178.6	0.008	370.96	73.782	0.000	102206.22	-5116.459
31.500	400.8	0.019	556.02	73.209	0.000	99599.71	-5309.504
32.000	710.8	0.035	740.45	72.791	0.000	96896.79	-5502.083
32.500	1107.9	0.055	924.08	72.459	0.000	94097.76	-5693.880
33.000	1591.5	0.081	1106.04	72.183	0.000	91203.20	-5883.948
33.500	2160.4	0.111	1285.29	71.945	0.000	88214.29	-6071.168
34.000	2813.0	0.146	1461.56	71.736	0.000	85132.59	-6255.191
34.500	3548.0	0.186	1635.96	71.549	0.000	81959.45	-6437.020
35.000	4364.5	0.231	1808.51	71.380	0.000	78695.95	-6616.610
35.500	5261.5	0.281	1979.14	71.224	0.000	75343.25	-6793.784
36.000	6238.0	0.336	2147.95	71.081	0.000	71902.56	-6968.562
36.500	7293.2	0.396	2315.09	70.947	0.000	68375.08	-7140.967
37.000	8426.0	0.460	2480.66	70.823	0.000	64761.99	-7311.005
37.059	8564.1	0.468	2500.00	70.808	0.000	64332.43	-7330.807
37.059	8564.1	0.468	2500.00	70.808	-7.684	64332.43	-7330.807
37.500	9628.8	0.532	2638.96	69.181	-4.479	61063.55	-7482.917
38.000	10894.5	0.614	2799.91	68.069	-2.893	57279.49	-7652.839
38.500	12226.8	0.704	2961.40	67.313	-2.132	53411.14	-7820.130
39.000	13627.0	0.802	3122.75	66.731	-1.753	49459.73	-7985.181
39.500	15095.5	0.907	3283.89	66.246	-1.542	45426.29	-8148.241
40.000	16632.6	1.019	3444.74	65.822	-1.415	41311.82	-8309.294
40.500	18238.0	1.139	3605.49	65.441	-1.339	37117.30	-8468.458
41.000	19911.8	1.266	3766.34	65.093	-1.301	32843.65	-8625.841
41.500	21654.0	1.400	3927.52	64.772	-1.297	28491.74	-8781.529
42.000	23464.7	1.541	4089.26	64.470	-1.331	24062.39	-8935.589
42.000	23464.7	1.541	4089.26	64.470	-1.323	24062.39	-8935.589
42.500	25297.3	1.686	4043.67	64.224	-1.268	19608.49	-8880.050
43.000	27106.7	1.830	4000.70	64.042	-1.221	15182.36	-8824.381
43.500	28894.8	1.974	3960.05	63.881	-1.531	10784.19	-8768.221
44.000	30662.6	2.117	3921.46	63.712	-2.479	6414.29	-8711.190
44.500	32410.5	2.259	3884.53	63.487	-6.877	2073.19	-8653.038
44.740	33242.0	2.328	3865.29	63.249	15.000	3.51	0.000

The interceptor guidance begins at segment 3 at a time of 37.059 seconds. The guidance immediately commands a pitchover with negative angle of attack and then stays on a near straight–line path. When the interceptor nears the target, it must make more rapid corrections. At the end of the trajectory, the interceptor is at a maximum angle of attack of 15°. Figure 4–21 shows the east position and altitude of the two trajectories as they intercept.

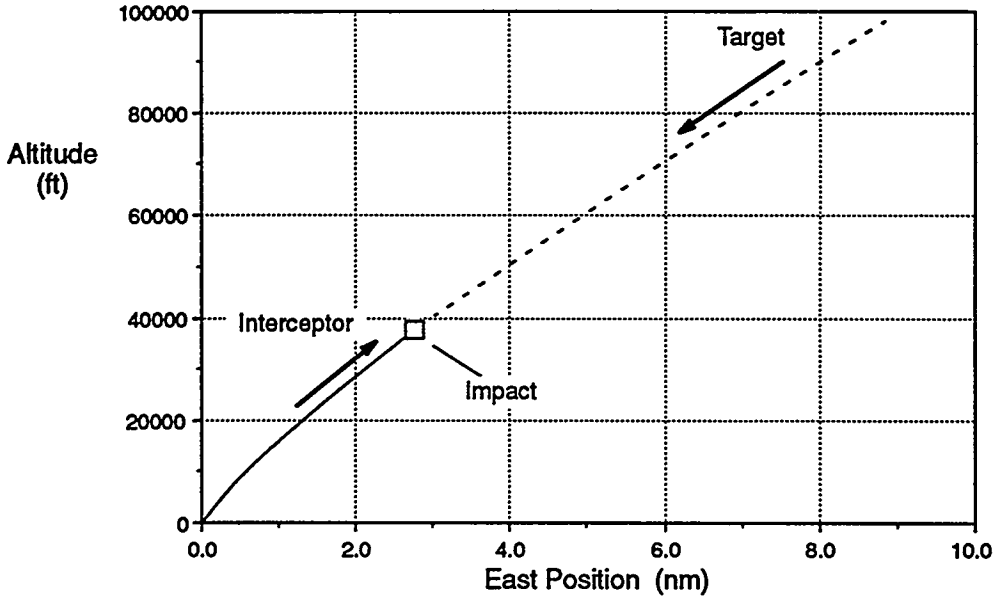

Figure 4–21. Ground–Launched Intercept Example.

Intentionally Left Blank

Appendix

Variable Name	Symbol	Description	Default Units	Decimal Places
accebx	$\vec{a}_{\oplus} \cdot \hat{x}_b$	component of acceleration with respect to the earth's surface in x direction of body coordinate system (Section 2.1.7).	ft/sec^2	4
acceby	$\vec{a}_{\oplus} \cdot \hat{y}_b$	component of acceleration with respect to the earth's surface in y direction of body coordinate system (Section 2.1.7).	ft/sec^2	4
accebz	$\vec{a}_{\oplus} \cdot \hat{z}_b$	component of acceleration with respect to the earth's surface in z direction of body coordinate system (Section 2.1.7).	ft/sec^2	4
accelx	$\vec{a}_I \cdot \hat{x}_{\oplus}$	x component of inertial acceleration in ECFC coordinates (Section 2.1.1).	ft/sec^2	4
accely	$\vec{a}_I \cdot \hat{y}_{\oplus}$	y component of inertial acceleration in ECFC coordinates (Section 2.1.1).	ft/sec^2	4
accelz	$\vec{a}_I \cdot \hat{z}_{\oplus}$	z component of inertial acceleration in ECFC coordinates (Section 2.1.1).	ft/sec^2	4
accibx	$\vec{a}_I \cdot \hat{x}_b$	component of inertial acceleration in x direction of body coordinate system (Section 2.1.7).	ft/sec^2	4
acciby	$\vec{a}_I \cdot \hat{y}_b$	component of inertial acceleration in y direction of body coordinate system (Section 2.1.7).	ft/sec^2	4
accibz	$\vec{a}_I \cdot \hat{z}_b$	component of inertial acceleration in z direction of body coordinate system (Section 2.1.7).	ft/sec^2	4
alpha	α	angle of attack (Section 2.1.9)	deg	3
alpha_l/d	$\alpha_{L/D}$	angle of attack at maximum lift–to–drag ratio (Section 2.4.5)	deg	3
alt	h	geodetic altitude (Section 2.1.6).	ft	1
altdt	$\dot{h}$	geodetic altitude rate (Section 2.2.3).	ft/sec	3

Variable Name	Symbol	Description	Default Units	Decimal Places
alphat	α_T	total angle of attack (Section 2.1.9).	deg	3
ballistic	β_c	ballistic coefficient (Section 2.4.5).	lb_f/ft^2	3
bankgc	μ_{gc}	geocentric bank angle (Section 2.1.9).	deg	3
bankgd	μ_{gd}	geodetic bank angle (Section 2.1.9).	deg	3
beta	β	sideslip angle (Section 2.1.9).	deg	3
betae	β_E	Euler sideslip angle (Section 2.1.9).	deg	3
ca	C_A	axial–force coefficient (Section 2.3.2).		6
cd	C_D	drag–force coefficient (Section 2.3.2).		6
cg		vehicle's center of mass.		3
cl	C_L	lift–force coefficient (Section 2.3.2).		6
cn	C_N	normal–force coefficient (Section 2.3.2).		6
crsrng	Δr_S	crossrange (Section 2.4.1).	nm	3
cs	C_S	side–force coefficient (Section 2.3.2).		6
cx	C_X	x component of body force coefficient (Section 2.3.2).		6
cy	C_Y	y component of body force coefficient (Section 2.3.2).		6
cz	C_Z	z component of body force coefficient (Section 2.3.2).		6
dwnrng	Δr_S	downrange (Section 2.4.1).	nm	3
dynprs	q	dynamic pressure (Section 2.3.2).	lb_f/ft^2	3
east	Δr_S	east position (Section 2.4.1).	nm	3
ep1	ε_1	thrust vector angle between thrust vector and body x axis (Section 2.3.3).		3
ep2	ε_2	thrust vector meridional angle in body y–z plane (Section 2.3.3).		3
fuel	Δm	fuel used (Section 2.2.1).	lb_m	2
gamgc	γ_{gc}	geocentric vertical flight path angle (Section 2.1.4).	deg	3
gamgcdt	$\dot{\gamma}_{gc}$	geocentric vertical flight path angle rate (Section 2.2.5).	deg/sec	3

Variable Name	Symbol	Description	Default Units	Decimal Places
gamgd	γ_{gd}	geodetic vertical flight path angle (Section 2.1.6).	deg	3
gamgddt	$\dot{\gamma}_{gd}$	geodetic vertical flight path angle rate (Section 2.2.5).	deg/sec	3
grmark	r_S	ground range since last "mark" (Section 2.2.4).	nm	3
grseg	r_S	segment ground range (Section 2.2.4).	nm	3
iip_azm	α_{iip}	azimuth from origin of tangent plane coordinate system to the initial impact point (Section 2.4.4).	deg	3
iip_latgd	δ_{iip}	geodetic latitude of initial impact point (Section 2.4.4).	deg	5
iip_long	λ_{iip}	geodetic latitude of initial impact point (Section 2.4.4).	deg	5
iip_rng	r_{iip}	range from origin of tangent plane coordinate system to the initial impact point (Section 2.4.4).	nm	3
iip_time	t_{iip}	time to initial impact point (Section 2.4.4).	sec	3
latgc	δ_{gc}	geocentric latitude (Section 2.1.4).	deg	5
latgcdt	$\dot{\delta}_{gc}$	geocentric latitude rate (Section 2.2.3).	deg/sec	7
latgd	δ_{gd}	geodetic latitude (Section 2.1.6).	deg	5
latgddt	$\dot{\delta}_{gd}$	geodetic latitude rate (Section 2.2.3).	deg/sec	7
l/d	L/D	lift–to–drag ratio (Section 2.4.5).		4
long	λ	longitude (Section 2.1.4).	deg	5
longdt	$\dot{\lambda}$	longitude rate (Section 2.2.3).	deg/sec	7
mach	M	Mach number (Section 2.3.2).		4
mass	m	vehicle mass (Section 2.2.1).	lb_m	4
mdt	$\dot{m}$	mass rate (Section 2.2.1).	lb_m/sec	4
north	Δr_S	north position (Section 2.4.1).	nm	3

Variable Name	Symbol	Description	Default Units	Decimal Places
ntotal	$\|\| \vec{a}_{sp} \|\|$	magnitude of specific load factor (Section 2.2.6).		4
nu	ν	kinematic viscosity (Section 2.3.1).	ft^2/sec	8
nx	$\vec{a}_{sp} \cdot \hat{x}_b$	specific load factor in body x direction (Section 2.2.6).		4
ny	$\vec{a}_{sp} \cdot \hat{y}_b$	specific load factor in body y direction (Section 2.2.6).		4
nz	$\vec{a}_{sp} \cdot \hat{z}_b$	specific load factor in body z direction (Section 2.2.6).		4
phi	ϕ_w	windward meridian angle (Section 2.1.9).	deg	3
pitchgc	Θ_{gc}	geocentric pitch angle (Section 2.1.7).	deg	3
pitchgd	Θ_{gd}	geodetic pitch angle (Section 2.1.7).	deg	3
pitchi	Θ_i	inertial platform pitch angle (Section 2.1.10).	deg	3
plength	r	path length (Section 2.2.1).	ft	3
plmark	r	path length since last "mark" (Section 2.2.1).	ft	3
plseg	r	segment path length (Section 2.2.1).	ft	3
power	P	power setting (Section 2.5).		3
pres	p	pressure (Section 2.3.1).	lbf/ft^2	4
psigc	ψ_{gc}	geocentric horizontal flight path angle (Section 2.1.4).	deg	3
psigcdt	$\dot{\psi}_{gc}$	geocentric horizontal flight path angle rate (Section 2.2.5).	deg/sec	3
psigd	ψ_{gd}	geodetic horizontal flight path angle (Section 2.1.6).	deg	3
psigddt	$\dot{\psi}_{gd}$	geodetic horizontal flight path angle rate (Section 2.2.5).	deg/sec	3
radasp[i]	η_r	aspect angle of vehicle with respect to i^{th} radar station (Section 2.4.2).	deg	4

Variable Name	Symbol	Description	Default Units	Decimal Places
radaz[i]	α_r	azimuth angle relative to ith radar station (Section 2.4.2).	deg	4
radazdt[i]	$\dot{\alpha}_r$	azimuth angle rate relative to ith radar station (Section 2.4.2).	deg/sec	4
radazdt2[i]	$\ddot{\alpha}_r$	azimuth angle acceleration relative to ith radar station (Section 2.4.2).	deg/sec^2	5
radelv[i]	ε_r	elevation angle relative to ith radar station (Section 2.4.2).	deg	4
radelvdt[i]	$\dot{\varepsilon}_r$	elevation angle rate relative to ith radar station (Section 2.4.2).	deg/sec	4
radelvdt2[i]	$\ddot{\varepsilon}_r$	elevation angle acceleration relative to ith radar station (Section 2.4.2).	deg/sec^2	5
radmer[i]	ϕ_r	meridional angle of vehicle with respect to ith radar station (Section 2.4.2).	deg	4
radrng[i]	$\|\Delta\vec{r}_r\|$	range relative to ith radar station (Section 2.4.2).	ft	2
radrngdt[i]	$\|\Delta\dot{\vec{r}}_r\|$	range rate relative to ith radar station (Section 2.4.2).	ft/sec	3
radrngdt2[i]	$\|\Delta\ddot{\vec{r}}_r\|$	range acceleration relative to ith radar station (Section 2.4.2).	ft/sec^2	4
range	r_S	ground range (Section 2.2.4).	nm	3
rcm	$\|\vec{r}\|$	distance from center of earth (Section 2.1.4).	ft	2
rcmdt	$\|\dot{\vec{r}}\|$	rate of change of the distance from the center of the earth (Section 4.5.5).	ft	3
relaz[i]	α_i	ith vehicle's azimuth angle relative to the body coordinate system (Section 2.4.3).	deg	4
relelv[i]	ε_i	ith vehicle's elevation angle relative to the body coordinate system (Section 2.4.3).	deg	4
relrng[i]	$\|\Delta\vec{r}_i\|$	ith vehicle's relative range (Section 2.4.3).	ft	2
relvel[i]	$\|\Delta\dot{\vec{r}}_i\|$	closure velocity of ith vehicle (Section 2.4.3).	ft/sec	3

Variable Name	Symbol	Description	Default Units	Decimal Places
relxb[i]	$\Delta\vec{r}_i \cdot \hat{x}_b$	component of the i^{th} vehicle's position relative to the body in the x direction of the body coordinate system (Section 2.4.3).	ft	2
relyb[i]	$\Delta\vec{r}_i \cdot \hat{y}_b$	component of the i^{th} vehicle's position relative to the body in the y direction of the body coordinate system (Section 2.4.3).	ft	2
relzb[i]	$\Delta\vec{r}_i \cdot \hat{z}_b$	component of the i^{th} vehicle's position relative to the body in the z direction of the body coordinate system (Section 2.4.3).	ft	2
reypft	R_N	Reynold's number per foot (Section 2.3.2).	ft^{-1}	0
rho	ϱ	density (Section 2.3.1).	lb_m/ft^3	7
rollgc	Φ_{gc}	geocentric roll angle (Section 2.1.7).	deg	3
rollgd	Φ_{gd}	geodetic roll angle (Section 2.1.7).	deg	3
rolli	Φ_i	inertial platform roll angle (Section 2.1.10).	deg	3
segment		segment number		0
sndspd	c	speed of sound (Section 2.3.1).	ft/sec	3
temp	T	temperature (Section 2.3.1).	°R	3
thrust	$\lVert \vec{F}_{prop} \rVert$	magnitude of thrust vector (Section 2.3.3).	lb_f	2
time	t	mission time.	sec	3
tmark	t	time since last "mark".	sec	3
tseg	t	segment time.	sec	3
uxbx	$\hat{x}_b \cdot \hat{x}_\oplus$	component of body x axis unit vector in x direction of ECFC coordinate system (Section 2.1.7).		6
uxby	$\hat{x}_b \cdot \hat{y}_\oplus$	component of body x axis unit vector in y direction of ECFC coordinate system (Section 2.1.7).		6
uxbz	$\hat{x}_b \cdot \hat{z}_\oplus$	component of body x axis unit vector in z direction of ECFC coordinate system (Section 2.1.7).		6

Variable Name	Symbol	Description	Default Units	Decimal Places
uybx	$\hat{y}_b \cdot \hat{x}_\oplus$	component of body y axis unit vector in x direction of ECFC coordinate system (Section 2.1.7).		6
uyby	$\hat{y}_b \cdot \hat{y}_\oplus$	component of body y axis unit vector in y direction of ECFC coordinate system (Section 2.1.7).		6
uybz	$\hat{y}_b \cdot \hat{z}_\oplus$	component of body y axis unit vector in z direction of ECFC coordinate system (Section 2.1.7).		6
uzbx	$\hat{z}_b \cdot \hat{x}_\oplus$	component of body z axis unit vector in x direction of ECFC coordinate system (Section 2.1.7).		6
uzby	$\hat{z}_b \cdot \hat{y}_\oplus$	component of body z axis unit vector in y direction of ECFC coordinate system (Section 2.1.7).		6
uzbz	$\hat{z}_b \cdot \hat{z}_\oplus$	component of body z axis unit vector in z direction of ECFC coordinate system (Section 2.1.7).		6
vair	$\| \vec{V}_{w_\oplus} \|$	airspeed (Section 2.1.8).	ft/sec	2
vel	$\| \vec{V}_\oplus \|$	velocity with respect to the earth's surface (Section 2.1.6).	ft/sec	2
veldt	$\| \vec{a}_\oplus \|$	acceleration with respect to the earth's surface (Sections 2.1.1 and 2.2).	ft/sec^2	2
velebx	$\vec{V}_\oplus \cdot \hat{x}_b$	component of velocity with respect to the earth's surface in x direction of body coordinate system (Section 2.1.7).	ft/sec	3
veleby	$\vec{V}_\oplus \cdot \hat{y}_b$	component of velocity with respect to the earth's surface in y direction of body coordinate system (Section 2.1.7).	ft/sec	3
velebz	$\vec{V}_\oplus \cdot \hat{z}_b$	component of velocity with respect to the earth's surface in z direction of body coordinate system (Section 2.1.7).	ft/sec	3

Variable Name	Symbol	Description	Default Units	Decimal Places
velibx	$\vec{V}_I \cdot \hat{x}_b$	component of inertial velocity in x direction of body coordinate system (Section 2.1.7).	ft/sec	3
veliby	$\vec{V}_I \cdot \hat{y}_b$	component of inertial velocity in y direction of body coordinate system (Section 2.1.7).	ft/sec	3
velibz	$\vec{V}_I \cdot \hat{z}_b$	component of inertial velocity in z direction of body coordinate system (Section 2.1.7).	ft/sec	3
vgr	$\|\vec{V}_S\|$	ground speed (Section 2.2.4).	ft/sec	2
windd	$\Delta\vec{V}_{w_\oplus} \cdot \hat{z}_{gd}$	wind component in the geodetic z direction (Section 2.1.8).	ft/sec	2
winde	$\Delta\vec{V}_{w_\oplus} \cdot \hat{y}_{gd}$	wind component in the geodetic y direction (Section 2.1.8).	ft/sec	2
windn	$\Delta\vec{V}_{w_\oplus} \cdot \hat{x}_{gd}$	wind component in the geodetic x direction (Section 2.1.8).	ft/sec	2
wt	$m \cdot g$	weight (Section 2.2.1).	lb_f	2
wtdt	$\dot{m} \cdot g$	weight rate or fuel flow (Section 2.2.1).	lb_f/sec	3
xecfc	$\vec{r} \cdot \hat{x}_\oplus$	component of position vector in x direction of ECFC coordinate system (Section 2.1.1).	ft	2
xecfcdt	$\vec{V}_\oplus \cdot \hat{x}_\oplus$	component of velocity vector with respect to the earth's surface in x direction of ECFC coordinate system (Section 2.1.1).	ft/sec	3
xecfcdt2	$\vec{a}_\oplus \cdot \hat{x}_\oplus$	component of acceleration vector with respect to the earth's surface in x direction of ECFC coordinate system (Section 2.1.1).	ft/sec^2	4
xecic	$\vec{r} \cdot \hat{x}_I$	component of position vector in x direction of ECIC coordinate system (Section 2.1.2).	ft	2
xecicdt	$\vec{V}_I \cdot \hat{x}_I$	component of inertial velocity vector in x direction of ECIC coordinate system (Section 2.1.2).	ft/sec	3
xecicdt2	$\vec{a}_I \cdot \hat{x}_I$	component of inertial acceleration vector in x direction of ECIC coordinate system (Section 2.1.2).	ft/sec^2	4

Variable Name	Symbol	Description	Default Units	Decimal Places
xip	$(\vec{r} - \vec{r}_p) \cdot \hat{x}_p$	component of position relative to the inertial platform x direction (Section 2.1.10).	ft	2
xipdt	$\vec{V}_I \cdot \hat{x}_p$	component of inertial velocity in the inertial platform x direction (Section 2.1.10).	ft/sec	3
xipdt2	$\vec{a}_I \cdot \hat{x}_p$	component of inertial acceleration in the inertial platform x direction (Section 2.1.10).	ft/sec^2	4
xtp	$(\vec{r} - \vec{r}_{tp}) \cdot \hat{x}_{tp}$	component of position relative to the tangent plane coordinate system x direction (Section 2.1.11).	ft	2
xtpdt	$\vec{V}_I \cdot \hat{x}_{tp}$	component of inertial velocity in the tangent plane coordinate system x direction (Section 2.1.11).	ft/sec	3
xtpdt2	$\vec{a}_I \cdot \hat{x}_{tp}$	component of inertial acceleration in the tangent plane coordinate system x direction (Section 2.1.11).	ft/sec^2	4
yawgc	Ψ_{gc}	geocentric yaw angle (Section 2.1.7).	deg	3
yawgd	Ψ_{gd}	geodetic yaw angle (Section 2.1.7).	deg	3
yawi	Ψ_i	inertial platform yaw angle (Section 2.1.10).	deg	3
yecfc	$\vec{r} \cdot \hat{y}_\oplus$	component of position vector in y direction of ECFC coordinate system (Section 2.1.1).	ft	2
yecfcdt	$\vec{V}_\oplus \cdot \hat{y}_\oplus$	component of velocity vector with respect to the earth's surface in x direction of ECFC coordinate system (Section 2.1.1).	ft/sec	3
yecfcdt2	$\vec{a}_\oplus \cdot \hat{y}_\oplus$	component of acceleration vector with respect to the earth's surface in y direction of ECFC coordinate system (Section 2.1.1).	ft/sec^2	4
yecic	$\vec{r} \cdot \hat{y}_I$	component of position vector in y direction of ECIC coordinate system (Section 2.1.2).	ft	2
yecicdt	$\vec{V}_I \cdot \hat{y}_I$	component of inertial velocity vector in y direction of ECIC coordinate system (Section 2.1.2).	ft/sec	3
yecicdt2	$\vec{a}_I \cdot \hat{y}_I$	component of inertial acceleration vector in y direction of ECIC coordinate system (Section 2.1.2).	ft/sec^2	4

Variable Name	Symbol	Description	Default Units	Decimal Places
yip	$(\vec{r} - \vec{r}_p) \cdot \hat{y}_p$	component of position relative to the inertial platform y direction (Section 2.1.10).	ft	2
yipdt	$\vec{V}_I \cdot \hat{y}_p$	component of inertial velocity in the inertial platform y direction (Section 2.1.10).	ft/sec	3
yipdt2	$\vec{a}_I \cdot \hat{y}_p$	component of inertial acceleration in the inertial platform y direction (Section 2.1.10).	ft/sec^2	4
ytp	$(\vec{r} - \vec{r}_{tp}) \cdot \hat{y}_{tp}$	component of position relative to the tangent plane coordinate system y direction (Section 2.1.11).	ft	2
ytpdt	$\vec{V}_I \cdot \hat{y}_{tp}$	component of inertial velocity in the tangent plane coordinate system y direction (Section 2.1.11).	ft/sec	3
ytpdt2	$\vec{a}_I \cdot \hat{y}_{tp}$	component of inertial acceleration in the tangent plane coordinate system y direction (Section 2.1.11).	ft/sec^2	4
zecfc	$\vec{r} \cdot \hat{z}_\oplus$	component of position vector in z direction of ECFC coordinate system (Section 2.1.1).	ft	2
zecfcdt	$\vec{V}_\oplus \cdot \hat{z}_\oplus$	component of velocity vector with respect to the earth's surface in x direction of ECFC coordinate system (Section 2.1.1).	ft/sec	3
zecfcdt2	$\vec{a}_\oplus \cdot \hat{z}_\oplus$	component of acceleration vector with respect to the earth's surface in z direction of ECFC coordinate system (Section 2.1.1).	ft/sec^2	4
zecic	$\vec{r} \cdot \hat{z}_I$	component of position vector in z direction of ECIC coordinate system (Section 2.1.2).	ft	2
zecicdt	$\vec{V}_I \cdot \hat{z}_I$	component of inertial velocity vector in z direction of ECIC coordinate system (Section 2.1.2).	ft/sec	3
zecicdt2	$\vec{a}_I \cdot \hat{z}_I$	component of inertial acceleration vector in z direction of ECIC coordinate system (Section 2.1.2).	ft/sec^2	4

Variable Name	Symbol	Description	Default Units	Decimal Places
zip	$(\vec{r} - \vec{r}_p) \cdot \hat{z}_p$	component of position relative to the inertial platform z direction (Section 2.1.10).	ft	2
zipdt	$\vec{V}_I \cdot \hat{z}_p$	component of inertial velocity in the inertial platform z direction (Section 2.1.10).	ft/sec	3
zipdt2	$\vec{a}_I \cdot \hat{z}_p$	component of inertial acceleration in the inertial platform z direction (Section 2.1.10).	ft/sec^2	4
ztp	$(\vec{r} - \vec{r}_{tp}) \cdot \hat{z}_{tp}$	component of position relative to the tangent plane coordinate system z direction (Section 2.1.11).	ft	2
ztpdt	$\vec{V}_I \cdot \hat{z}_{tp}$	component of inertial velocity in the tangent plane coordinate system z direction (Section 2.1.11).	ft/sec	3
ztpdt2	$\vec{a}_I \cdot \hat{z}_{tp}$	component of inertial acceleration in the tangent plane coordinate system z direction (Section 2.1.11).	ft/sec^2	4

Intentionally Left Blank

References

1. D. E. Salguero, *Point–Mass Simulation Tool (PMAST) User's Manual*, SAND85–2039, Sandia National Laboratories, Albuquerque, NM, March 1986.

2. D. E. Outka, *User's Manual for the Trajectory Simulation and Analysis Program (TSAP)*, SAND88–3158, Sandia National Laboratories, Albuquerque, NM, July 1990.

3. M. J. D. Powell, A Fast Algorithm for Nonlinearly Constrained Optimization Calculations, *Proceedings of the Biennial Conference on Numerical Analysis, 28 June – 1 July 1977*, G. A. Watson, ed., Springer–Verlag, Berlin, Germany, pp. 144–57, 1978.

4. D. E. Salguero, *Engineering Graphics System (EGS) User's Manual*, SAND89–0156, Sandia National Laboratories, Albuquerque, NM, January 1989.

5. T. O. Seppelin, *The Department of Defense World Geodetic System 1972*, World Geodetic System Committee, International Symposium on Problems Related to the Redefinition of North American Geodetic Networks, Frederiction, New Brunswick, Canada, May 1974.

6. *Department of Defense World Geodetic System 1984*, DMA TR 8350.2, Defense Mapping Agency WGS 84 Development Committee, Washington, DC, September 30, 1987.

7. J. L. McDowell, B. E. Schutz, R. E. McKenzie, and B. D. Tapley, *Trajectory Analysis Program Mathematical Specifications (Version 770701)*, TR 77–1, Center for Advanced Study in Orbital Mechanics, Department of Aerospace Engineering and Engineering Mechanics, The University of Texas at Austin, 1977.

8. L. W. Johnson and R. D. Riess, *Numerical Analysis, 2nd ed.*, Addison–Wesley, Reading, MA, 1982.

9. W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*, Brooks/Cole, Monterey, CA, 1980.

10. G. W. Rosborough, Gravitational Effects on Low–Earth Orbiters, *Advances in the Astronautical Sciences*, Vol. 74, pp. 587–610, 1991.

11. J. G. Marsh, F. J. Lerch, B. H. Putney, D. C. Christodoulidis, D. E. Smith, T. L. Felsentreger, B. V. Sanchez, S. M. Klosko, E. C. Pavlis, T. V. Martin, J. W. Robbins, R. G. Williamson, O. L. Colombo, D. D. Rowlands, W. F. Eddy, N. L. Chandler, K. E. Rachlin, G. B. Patel, S. Bhati, and D. S. Chinn, A New Gravitational Model for the Earth from Satellite Tracking Data: GEM–T1, *Journal of Geophysical Research*, Vol. 93, pp. 6169–6215, June 1988.

12. D. B. Landrum, *Standard Atmosphere Subroutine*, Internal Memorandum RS1555/87/010, Sandia National Laboratories, Albuquerque, NM, January 1987.

13. *U.S. Standard Atmosphere, 1976*, National Oceanic and Atmospheric Administration, National Aeronautics and Space Administration, and the United States Air Force, U.S. Government Printing Office, Washington, DC, 1976.

14. *U.S. Standard Atmosphere Supplements, 1966*, Environmental Science Services Administration, National Aeronautics and Space Administration, and the United States Air Force, U.S. Government Printing Office, Washington, DC, 1966.

15. *Kwajelein Reference Atmosphere 1979, 1966*, AFGL–TR–79–0241, Air Force Geophysical Laboratory, September 1979.

16. E. M. Sodano and T. A. Robinson, *Direct and Inverse Solutions of Geodesics*, Technical Report 7 (Revised), U.S. Army Map Service, Washington, DC, pp. 15–27, 1963.

17. P. Zarchan, *Tactical and Strategic Missile Guidance, 2nd ed.*, vol. 157, Progress in Astronautics and Aeronautics, American Institute of Aeronautics and Astronautics, Washington, DC, 1994.

18. K. D. Bruns, M. E. Moore, S. L. Stoy, and S. R. Vukelich, *Missile Datcom User's Manual – Rev 4/91*, WL–TR–91–3039, Wright Laboratory, Wright Patterson Air Force Base, Ohio, April 1991.

19. R. W. Noack and A. R. Lopez, *Inviscid Flow Field Analysis of Complex Reentry Vehicles*, vol. I and II, SAND87–0776, Sandia National Laboratories, Albuquerque, NM, October 1988.

20. K. V. Chavez and D. E. Salguero, *A User's Manual for the Aerodynamic Prediction Software (AERO)*, SAND93–0479, Sandia National Laboratories, Albuquerque, NM, April 1993.

21. D. E. Salguero, *Solid Rocket Motor Performance for Conceptual Design Studies*, SAND89–1181, Sandia National Laboratories, Albuquerque, NM, May 1989.

22. Jia–Yih Chang, Recursive Quadratic Programming with Best Feasible Point, PhD Dissertation, The University of Texas at Austin, May 1989.

23. R. W. Stineman, A Consistently Well–Behaved Method of Interpolation, *Creative Computing*, pp. 54–57, July 1980.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z

Intentionally Left Blank

Distribution

1	MS 0826	J. K. Cole, 9114
1	MS 0826	D. W. Kuntz, 9114
1	MS 0826	D. L. Potter, 9114
1	MS 0826	T. M. Sterk, 9114
1	MS 0826	L. W. Young, 9114
1	MS 0825	K. V. Chavez, 9115
1	MS 0825	T. M. Jordan–Culler, 9115
1	MS 0825	M. W. Kniskern, 9115
1	MS 0825	A. R. Lopez, 9115
1	MS 0825	W. A. Millard, 9115
1	MS 0825	J. L. Payne, 9115
1	MS 0825	L. R. Rollstin, 9115
1	MS 0825	W. H. Rutledge, 9115
1	MS 0303	D. J. Rigali, 2400
1	MS 0303	D. L. Davidson, 2411
1	MS 0303	S. A. Kerr, 2411
1	MS 0303	J. L. McDowell, 2411
21	MS 0303	D. E. Salguero, 2411
1	MS 0303	B. R. Sturgis, 2411
1	MS 0313	D. L. Keese, 2412
1	MS 0309	J. J. Hochrein, 2413
1	MS 0312	R. W. Greene, 2414
1	MS 0312	W. E. Williamson, 2414
1	MS 0303	M. W. Sterk, 2415
1	MS 0313	A. C. Bustamante, 2416
1	MS 0307	E. W. Reese, 2417
1	MS 0309	A. K. Miller, 2418
1	MS 0658	R. G. Hay, 2419
1	MS 0659	E. J. Schindwolf, 2425
1	MS 0105	B. B. Asher, 2435
1	MS 1174	D. E. Outka, 2526
1	MS 9018	Central Technical Files, 8523–2
5	MS 0899	Technical Library, 13414
1	MS 0619	Print Media, 12615
2	MS 0100	Document Processing, 7613–2 For DOE/OSTI

Intentionally Left Blank