JSC-61777-models/interactions/aerodynamics

JSC Engineering Orbital Dynamics Aerodynamics Model

Simulation and Graphics Branch (ER7)
Software, Robotics, and Simulation Division
Engineering Directorate

Package Release JEOD v5.0

Document Revision 1.2
July 2022

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

JSC-61777-models/interactions/aerodynamics

JSC Engineering Orbital Dynamics Aerodynamics Model

Document Revision 1.2
July 2022

Andrew Spencer

Simulation and Graphics Branch (ER7)
Software, Robotics, and Simulation Division
Engineering Directorate

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

Abstract

The Aerodynamics Model computes the forces and torques that model the effects of aerodynamics on a vehicle. These forces and torques are summed into the forces and torques that model the overall vehicle dynamics. These models are used to characterize the affects of aerodynamics on a vehicle's trajectory. The Aerodynamics Model consists of two drag modeling options. There is a simple drag model of using the coefficient of drag or the ballistic coefficient (also known as the ballistic number) method. A composite plate model with surfaces that include accommodation factors for specular, diffuse or mixed reflections (mixed here means a combination of specular and diffuse).

Additionally, an extensible framework for the addition of new methods of calculating drag has been implemented, giving flexibility to the end user of the JEOD v5.0 package.

Contents

Chapter 1

Introduction

1.1 Model Description

Atmospheric force is the largest non-gravitational perturbation acting on spacecraft in low Earth orbit. Accurate modeling of aerodynamic forces presents three difficulties: the physical properties of the atmosphere are not known with high accuracy, the modeling requires knowledge of the interactions of neutral gas with spacecraft surfaces, and the requirement of higher fidelity, non-spherical representations of a vehicle. The dominant atmosphere force acting is called drag and is directed opposite to the velocity of the vehicle motion. When the line of incidence of the aerodynamic drag is not through the center of mass there will be a torque. Aerodynamic torques result from the force at the incidence point of atmospheric drag on the moment arm defined by the distance between the center of pressure and the center of mass. On-orbit aerodynamics is modeled in JEOD as simple drag and as free molecular drag. In addition a further refinement may be made by modeling the way a surface in the thermosphere accommodates the impinging atmospheric molecules.

Additionally, this model provides an extensible framework for the addition of new methods of calculating aerodynamic drag on a vehicle, utilizing both the Aerodynamics Model as well as the JEOD Surface Model utility [9].

1.2 Document History

Author	Date	Revision	Description
Andrew Spencer	November, 2009	1.0	Initial Version
Andrew Spencer	April, 2010	1.1	Added FlatPlateThermalAeroFactory documentation
Andrew Spencer	October, 2009	1.2	Reflected changes in enumerations, added metrics

This document derives heavily from it's prececessor, JSC Engineering Orbital Dynamics On-Orbit Aerodynamics Models released with JEOD v1.5.2.

The following document is parent to this document:

- *JSC Engineering Orbital Dynamics* [5]

1.3 Document Organization

This document is formatted in accordance with the NASA Software Engineering Requirements Standard [6] and is organized into the following chapters:

Chapter 1: Introduction - This introduction contains three sections: description of model, document history, and organization. The first section provides the introduction to the Aerodynamics Model and its reason for existence. It also contains a brief description of the interconnections with other models, and references to any supporting documents. The second section displays the history of this document which includes author, date, and reason for each revision; it also lists the document that is parent to this one. The final section contains a description of how the document is organized.

Chapter 2: Product Requirements - Describes requirements for the Aerodynamics Model.

Chapter 3: Product Specification - Describes the underlying theory, architecture, and design of the Aerodynamics Model in detail. It is organized in three sections: Conceptual Design, Mathematical Formulations, and Detailed Design.

Chapter 4: User Guide - Describes how to use the Aerodynamics Model in a Trick simulation. It is broken into three sections to represent the JEOD defined user types: Analysts or users of simulations (Analysis), Integrators or developers of simulations (Integration), and Model Extenders (Extension).

Chapter 5: Verification and Validation - Contains Aerodynamics Model verification and validation procedures and results.

Chapter 2

Product Requirements

This chapter will describe the requirements for the Aerodynamics Model.

Requirement aerodynamics_1: Top-level requirement

Requirement:
This model shall meet the JEOD project requirements specified in the JEOD v5.0 top-level document.

Rationale:
This model shall, at a minimum, meet all external and internal requirements applied to the JEOD v5.0 release.

Verification:
Inspection

2.1 Data Requirements

This section identifies requirements on the data represented by the Aerodynamics Model. These as-built requirements are based on the Aerodynamics Model data definition header files.

Requirement aerodynamics_2: Aerodynamic Data

Requirement:

2.1 Drag Modeling Data - Aerodynamics Model shall consist of

- a model of simple drag ,
- free molecular flow drag,
- a method of representing the aerodynamic characteristics of a complex body with a composite of simple surfaces.

The modeling shall produce the forces and torques on a vehicle as a function of its translational and rotational state. The Aerodynamics Model will provide the input selection of ballistic coefficient (BC), coefficient of drag (CD), under Newtonian flow and Free Molecular Flow conditions. Free Molecular Flow will provide the modeling of specular and diffuse reflection or a combination thereof. If not using BC or CD a complex vehicle may be modeled using a set of flat plates such that the sum approximates the overall drag characteristics of the vehicle. The output of the Aerodynamics Model will be the force and moments due to drag in the structural frame of the vehicle.

2.2 Functional Requirements

This section identifies requirements on the functional capabilities provided by the Aerodynamics Model. These as-built requirements are based on the Aerodynamics Model source files.

Requirement aerodynamics_3: On Orbit Drag Modeling

Requirement:

The Aerodynamics Model shall model aerodynamic drag on a surface, such that forces and torques are produced. Vehicle area and atmospheric density will be inputs to the modeling. The drag modeling shall include:

3.1 Simple Drag - The Aerodynamics Model shall provide the user with the capability to choose either a coefficient of drag or a ballistic number to model vehicle drag. In addition a simple option for a constant atmospheric density will be available.

3.2 Approximate Free Molecular Flow - The Aerodynamics Model shall provide the user with the capability to choose an approximate computation of free molecular drag flow allowing three conditional modes of expression. One will be computation of free molecular flow using specular reflection. One will be computation of free molecular flow using diffuse reflection. One will be computation of free molecular flow using a combination of specular and diffuse reflection.

3.3 Free Molecular Flow - The Aerodynamics Model shall provide the user with the capability to choose an approximate computation of free molecular drag flow allowing two conditional modes of expression. One will be the computation of free molecular flow using specular reflection and diffuse reflection by specifying the fraction of each. The second will be the computation of flow using normal and tangential reflection by specifying the fraction of each.

3.4 Aerodynamic Surfaces - The Aerodynamics Model shall provide the ability to represent the forces and torques acting on the vehicle by using an ensemble of flat plates to represent the projected area for drag on the vehicle.

Rationale:

The purpose of the Aerodynamics Model is to simulate as close as possible the on-orbit state vector propagation of a vehicle accounting for aerodynamics drag.

Verification:
Inspection and Testing.

Requirement aerodynamics_4: Aerodynamic Extensibility

Requirement:
This model shall provide a mechanism for user extension of the Aerodynamics Model.

Rationale:

Verification:
The verification for this item will be done by inspection.

Chapter 3

Product Specification

3.1 Conceptual Design

This section will present the conceptual design for the Aerodynamics Model including the extensible framework available to users for implementations of different methods of aerodynamic drag calculation.

The JEOD v5.0 Aerodynamics Model is made up of the following classes:

- AerodynamicDrag,
- DefaultAero,
- AeroDragEnum,
- AeroDragParameters,
- AerodynamicsMessages,
- AeroSurface,
- AeroFacet,
- AeroParams,
- AeroSurfaceFactory,
- FlatPlateAeroFacet,
- FlatPlateAeroParams,
- FlatPlateAeroFactory,
- FlatPlateThermalAeroFactory.

These classes can be split into two groups; those that are associated with the main AerodynamicDrag class, responsible for calculating forces and torques associated with aerodynamic drag, and those responsible for extending the JEOD Surface Model [9] for use with the AerodynamicDrag class. The following sections will discuss these two groups.

Note that the JEOD Aerodynamics Model contains a specific implementation of the JEOD Surface Model [9], allowing for the use of a collection of flat plates to be used as a surface model for the aerodynamic drag calculation. These classes may also be extended to use different Facet types in the surface model for aerodynamics. The use of these classes may require knowledge of the JEOD Surface Model, thus the reader should consult that documentation [9] for any required further details.

3.1.1 AerodynamicDrag Classes

This section will present the main AerodynamicDrag class as well as the classes associated with its basic use.

AerodynamicDrag

The AerodynamicDrag class is the main interface to the Aerodynamics Model. It takes in current information about the vehicle affected by the drag, and supplies the user with the total force and torque resulting from aerodynamic drag.

DefaultAero

The DefaultAero class supplies the simple ballistic coefficient and coefficient of drag methods for calculating aerodynamic forces, as well as an option for constant drag. These simple options are included in the AerodynamicDrag class by default.

It is also meant to be a virtual class, meaning that it is possible for a user to specify their own class, inheriting from DefaultAero, and overriding the functions used to calculate drag. This class can then be given to the AerodynamicDrag class, and a new method for calculating drag can be implemented with little to no change to the user interface.

AeroDragEnum

The AeroDragEnum class contains enumerations associated with calculating drag in the AerodynamicDrag class.

AeroDragParameters

The AeroDragParameters class contains parameters associated with calculating drag in the AerodynamicDrag class.

AerodynamicsMessages

The AerodynamicsMessages class contains messages used through the JEOD Message Handler [8] to indicate warnings and failures associated with the aerodynamics class to the end user.

3.1.2 Surface Model Extension Classes

This section will describe the interface of JEOD's Surface Model [9] with the Aerodynamics Model. The Surface Model will be extended for use in the AerodynamicDrag class, using flat plate constructs as described in the Mathematical Formulation section.

AeroSurface

The AeroSurface class is an InteractionSurface [9] derived class, intended to be used for calculating aerodynamic drag on a vehicle using the AerodynamicDrag class. It contains a collection of polymorphic pointers to AeroFacets, which are created from Facets contained in an originating SurfaceModel object [9]. This AeroSurface class can then be used to calculate the aerodynamic drag affects.

AeroFacet

The AeroFacet class is a pure virtual, InteractionFacet [9] derived class, used as a polymorphic interface for aerodynamic drag based interaction facets for use in the AeroSurface class. It is intended to be inherited from, if a user wishes to extend the AeroSurface to incorporate new types of facets into an aerodynamics model.

AeroParams

The AeroParams class is a FacetParams [9] derived class, used as a polymorphic interface for associating aerodynamic parameters with surface model facets, which can then be used to create AeroFacet inherited objects.

AeroSurfaceFactory

The AeroSurfaceFactory is an InteractionSurfaceFactory [9] derived class, used to create an AeroSurface object from a SurfaceModel object. The AeroSurfaceFactory class, by default, contains the InteractionFacetFactory classes to create a FlatPlateAeroFacet from either a FlatPlate or a FlatPlateThermal. These classes are further explained below or in the Surface Model documentation [9].

FlatPlateAeroFacet

The FlatPlateAeroFacet is an AeroFacet derived class. It is an aerodynamic-specific interaction facet created from a flat plate facet [9], and is used in an AeroSurface object to calculate aerodynamic drag effects on a flat plate.

FlatPlateAeroParams

The FlatPlateAeroParams class is an AeroParams derived class. It is used to specify values used to create a FlatPlateAeroFacet object from a FlatPlate [9] object.

FlatPlateAeroFactory

The FlatPlateAeroFactory class is an InteractionFacetFactory derived [9] object. It is used to create a FlatPlateAeroFacet object from a FlatePlate [9] object, using a FlatPlaetAeroParams object.

FlatPlateThermalAeroFactory

The FlatPlateThermalAeroFactory class is an InteractionFacetFactory derived [9] object. It is used to create a FlatPlateAeroFacet object from a FlatePlateThermal [9] object, using a FlatPlateAeroParams object.

Note that this class creates an interaction facet of the same type as the FlatPlateAeroFactory class. This is because the aerodynamic interaction is not currently integrated with the thermal aspect of the FlatPlateThermal facet (note that FlatPlateThermal derives from FlatPlate). As such, a specific aerodynamic version of the FlatPlateThermal class is not necessary, and instead a FlatPlateAeroFacet will suffice.

3.2 Mathematical Formulations

3.2.1 Aerodynamic Force

In the rarefied atmosphere at orbital altitudes, the gas molecules that hit the spacecraft are re-emitted and travel far before colliding with other molecules. In this regime of free molecular flow gas dynamics, the effect of the re-emitted particles on the incident stream can be neglected, at least so far as subsequent effects on the spacecraft are concerned. Therefore, for the purposes of this model description, the incident flow is considered to be undisturbed by the presence of the spacecraft. This non-interaction of the incident and re-emitted particles allow the net aerodynamic torque to be calculated by summing up the torque contributions of each of the spacecraft elements. Thus, the vehicle may be dissected into simple sub-shapes to facilitate estimation of the aerodynamic forces. Summing these forces gives the result for the entire spacecraft. However, before this process is undertaken, an approximation of the total aerodynamic force acting on the vehicle is often used to obtain a conservative estimate of the aerodynamic forces and torque. The approximate value of

the aerodynamic force for each applicable vehicle orientation may be obtained using the following expression from Vallado [13]:

$$\mathbf{a}_{drag} = -\frac{1}{2}\frac{c_d A_d}{m}\rho v_{rel}^2 \frac{\mathbf{v}_{rel}}{|\mathbf{v}_{rel}|} \tag{3.1}$$

where:

c_d =coefficient of drag
A_d = projected drag area
m =mass
ρ =atmospheric density
$\mathbf{v_{rel}}$ =velocity is the vehicle velocity relative to the atmosphere.

If $\omega_\oplus$ is the Earth's rotation rate vector then the relative velocity is given by:

$$\mathbf{v}_{rel} = \mathbf{v}_{inertial} - \omega_\oplus \times \mathbf{r_{inertial}} + \mathbf{v_{wind}} \tag{3.2}$$

where:

$\mathbf{v}_{inertial}$ =velocity with respect to the J2000 inertial frame,
$\mathbf{r}_{inertial}$ =position with respect to the J2000 inertial frame,
$\mathbf{v}_{wind}$ = upper atmospheric super-rotational winds (optional scaling in the simulation).

Equation (3.1) can be re-written using the concept of ballistic coefficient (BC). For the purposes of the JEOD Aerodynamics Model, ballistic coefficient is defined such that Equation (3.1) can be rewritten as:

$$\mathbf{a}_{drag} = -\frac{1}{2}\frac{1}{BC}\rho v_{rel}^2 \frac{\mathbf{v}_{rel}}{|\mathbf{v}_{rel}|} \tag{3.3}$$

where BC is the ballistic coefficient of drag, as defined for JEOD v5.0.

3.2.2 Aerodynamic Free Molecular Flow

When the use of approximations indicate that the aerodynamic force is significant, as compared to other environmental forces, a more detailed calculation may be necessary. Such a calculation goes beyond empirical estimates of the surface interaction and requires an understanding of the basic physics of particle interactions that depend on surface roughness, temperature, and composition of both the gas and the surface. At orbital altitudes, the atmosphere is sufficiently tenuous that it is considered collisionless, and the spacecraft's interaction with it is considered in terms of free molecular flow, i.e., the incident and emitted fluxes of molecules are considered as independent. In this regime,one treats aerodynamic drag and torques in terms of energy and momentum exchange between the impinging atoms or molecules and the spacecraft surfaces. The exchange can range from *specular* reflection in which the molecule "bounces off" the surface with unchanged energy and tangential momentum and reversed normal momentum, to completely *diffuse* reflection in which

the incoming molecules are "absorbed" on the surface and subsequently re-emitted randomly (in direction) and in thermodynamic equilibrium with the surface.

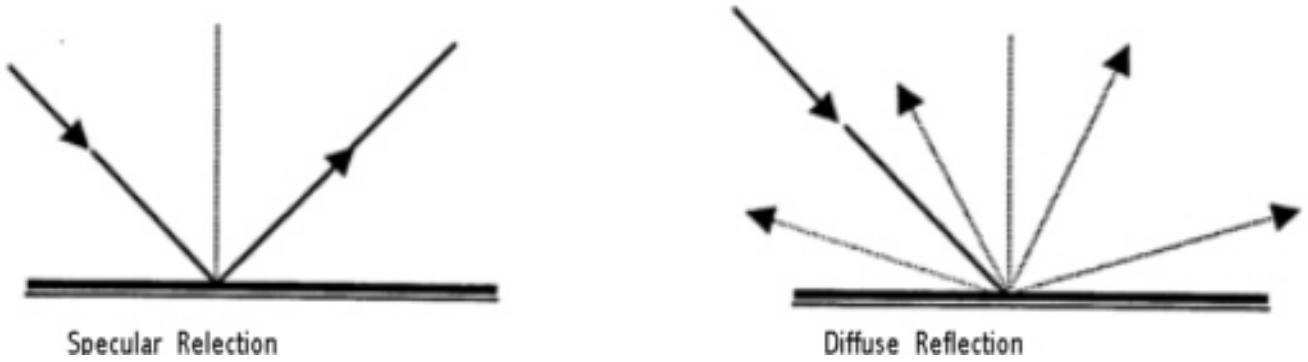

Figure 3.1: Specular and Diffuse Reflection

A complete analysis requires consideration of the details of spacecraft geometry and surface reflection characteristics as well as consideration of the velocity distributions of the atmospheric gases, usually treated as Maxwellian on the assumption that they are in thermodynamic equilibrium. For diffuse molecular re-emission and specular reflection one can compute the drag and lift coefficients for a flat plate in free molecular flow, Schaaf and Cambre [7] and the NACA [11]:

$$C_{Lexact} = \frac{2\cos\alpha}{s^2}\{\frac{(2-f_n-f_t)}{\sqrt{\pi}} s\sin\alpha e^{-s^2\sin^2\alpha} + \frac{f_n}{2}\sqrt{\pi}\sqrt{\frac{T_{ref}}{T_{fs}}}\sin\alpha + (2-f_n-f_t)s^2\sin^2\alpha erf(s\sin\alpha) + \frac{1}{2}(2-f_n)erf(s\sin\alpha)\} \tag{3.4}$$

$$C_{Dexact} = C_{Lexact}\tan\alpha + \frac{2f_t}{\sqrt{\pi}s}e^{-s^2\sin^2\alpha} + f_t\sin\alpha erf(s\sin\alpha) \tag{3.5}$$

:

s = molecular speed ratio (ratio of stream mass velocity to most probable molecular speed),
α = angle of attack of body element with respect to the free stream
and

$$erf(s) = \frac{2}{\sqrt{\pi}}\int_0^s e^{-x^2}dx \tag{3.6}$$

f_t = the tangential accommodation coefficient
and
f_n = the normal accommodation coefficient
$f_t = f_n = 0$ for Specular reflection
and
$f_t = f_n = 1$ for Diffuse reflection.

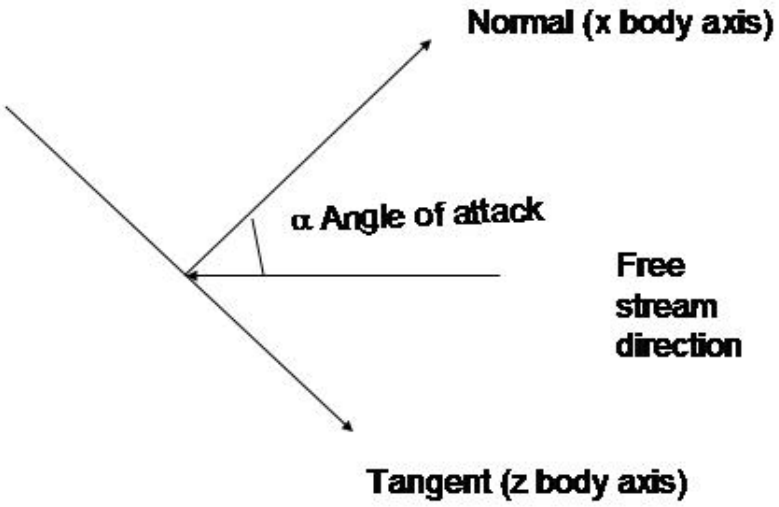

Figure 3.2: Angle of Attack

From the definition of angle of attack α, shown in Figure 3.2, the coefficients of lift and drag one can write for the normal coefficient C_n and the tangential C_t coefficient (also known as Axial C_a):

$$C_n = C_L \cos\alpha + C_D \sin\alpha \tag{3.7}$$

$$C_t = -C_L \sin\alpha + C_D \cos\alpha \tag{3.8}$$

Within the JSC Engineering Orbital Dynamics Aerodynamics Model free molecular flow is represented by two two different formulations. There is an approximation for free molecular flow expressed as maximum and minimum drag :

$$C_{d\,\max} = \frac{1}{s^2}\left(\frac{2(1-\varepsilon)}{\sqrt{\pi}} s\sin\alpha e^{-s^2\sin^2\alpha} + (1+\varepsilon)(1+2s^2\sin^2\alpha)\right) \tag{3.9}$$

and

$$C_{d\,\min} = \frac{1}{s^2}\left(\frac{2}{\sqrt{\pi}} s\sin\alpha e^{-s^2\sin^2\alpha} + \sqrt{\pi}\sqrt{\frac{T_r}{T_{fs}}} s\sin\alpha + + \frac{(1+2s^2\sin^2\alpha)}{s^2}\right) \tag{3.10}$$

All the variables have the same definitions as given above with the addition of the parameter ϵ which is the fraction of molecules reflected specularly and $1-\epsilon$ is the fraction reflected diffusely. There is an option to compute the normal and tangential forces exactly for a flat plate after the formulation of Bird [1]:

$$C_{normal}(Bird) = \frac{1}{s^2}\left(\frac{2(1+\epsilon)}{\sqrt{\pi}} s\sin\alpha e^{-s^2\sin^2\alpha} + (1-\epsilon)\sqrt{\pi}\sqrt{\frac{T_r}{T_{fs}}} + s\sin\alpha + (1+\epsilon)(1+2s^2\sin^2\alpha) erf(s\sin\alpha)\right) \tag{3.11}$$

and

$$C_{\tan}(Bird) = \frac{1}{s^2}\frac{2(1-\epsilon)}{\sqrt{\pi}} s\cos\alpha\left(e^{-s^2\sin^2\alpha} + \sqrt{\pi} s\sin\alpha erf(s\sin\alpha)\right) \tag{3.12}$$

However, in JEOD the absolute value of $C_{\tan}(Bird)$ is used and the angle of attack occurs as a direction factor later in the computation to account for direction, again all the variable definitions given above apply.

3.2.3 Aerodynamic Torque

When the drag acceleration (or force per unit mass) given by equation 3.1 is used to evaluate the total aerodynamic force, the aerodynamic torque is, in turn, estimated by the following expression:

$$\mathbf{N}_{aero} = \sum_{i=1}^{k} \mathbf{r}_i \times \mathbf{F}_i \tag{3.13}$$

where:

$\mathbf{F_i}$ = aerodynamic force on surface element i
and
$\mathbf{r_i}$ = is the vector from the spacecraft center to the center of pressure of the ith element.

If the spacecraft is modeled as a number of plane surfaces the aerodynamic torque can also be written as:

$$\mathbf{N}_{aero} = \frac{1}{2} C_D \rho v^2 \sum_{i=1}^{k} A_i (\mathbf{n}_i \bullet \mathbf{v}) \times \mathbf{r}_i \tag{3.14}$$

where:

A_i are the surface areas.

3.3 Detailed Design

The complete API for the Aerodynamics Model can be found in the *Reference Manual* [3].

3.4 Inventory

All Aerodynamics Model files are located in the directory `${JEOD_HOME}/models/interactions/aerodynamics`. Relative to this directory,

- Header and source files are located in the model `include` and `src` subdirectories. Table 3.1 lists the configuration-managed files in these directories.
- Data files are located in the model `data` subdirectory. See table 3.2 for a listing of the configuration-managed files in this directory.
- Documentation files are located in the model `docs` subdirectory. See table 3.3 for a listing of the configuration-managed files in this directory.
- Verification files are located in the model `verif` subdirectory. See table 3.4 for a listing of the configuration-managed files in this directory.

Table 3.1: Source Files

File Name
include/aero_drag.hh
include/aero_facet.hh
include/aero_params.hh
include/aero_surface.hh
include/aero_surface_factory.hh
include/aerodynamics_messages.hh
include/class_declarations.hh
include/default_aero.hh

Continued on next page

Table 3.1: Source Files (continued from previous page)

File Name
include/flat_plate_aero_facet.hh
include/flat_plate_aero_factory.hh
include/flat_plate_aero_params.hh
include/flat_plate_thermal_aero_factory.hh
src/aero_drag.cc
src/aero_facet.cc
src/aero_params.cc
src/aero_surface.cc
src/aero_surface_factory.cc
src/aerodynamics_messages.cc
src/default_aero.cc
src/flat_plate_aero_facet.cc
src/flat_plate_aero_factory.cc
src/flat_plate_aero_params.cc
src/flat_plate_thermal_aero_factory.cc

Table 3.2: Data Files

File Name
data/include/aero_model.hh
data/src/aero_model.cc

Table 3.3: Documentation Files

File Name
docs/aerodynamics.pdf
docs/refman.pdf
docs/tex/aerodynamics.bib
docs/tex/aerodynamics.sty
docs/tex/aerodynamics.tex
docs/tex/aerodynamicsAbstract.tex
docs/tex/aerodynamicsChapters.tex
docs/tex/makefile
docs/tex/model_name.mk
docs/tex/figs/diffuse_cc.jpg

Continued on next page

Table 3.3: Documentation Files (continued from previous page)

File Name
docs/tex/figs/diffuse_cc_err.jpg
docs/tex/figs/diffuse_max.jpg
docs/tex/figs/fmf.jpg
docs/tex/figs/mixed_cc.jpg
docs/tex/figs/mixed_err.jpg
docs/tex/figs/mixed_max.jpg
docs/tex/figs/ratio.pdf
docs/tex/figs/ratiodiff.jpg
docs/tex/figs/sd.jpg
docs/tex/figs/specular_cc.jpg
docs/tex/figs/specular_max.jpg
docs/tex/figs/torque_diff.jpg

Table 3.4: Verification Files

File Name
verif/SIM_VER_DRAG/S_define
verif/SIM_VER_DRAG/S_overrides.mk
verif/SIM_VER_DRAG/DP_Product/DP_validation.xml
verif/SIM_VER_DRAG/Log_data/log_drag_rec.py
verif/SIM_VER_DRAG/Modified_data/common_input.py
verif/SIM_VER_DRAG/Modified_data/input_common.py
verif/SIM_VER_DRAG/Modified_data/one_plate_accel_calc_coef_eps00.py
verif/SIM_VER_DRAG/Modified_data/one_plate_accel_calc_coef_eps05.py
verif/SIM_VER_DRAG/Modified_data/one_plate_accel_calc_coef_eps1.py
verif/SIM_VER_DRAG/Modified_data/one_plate_accel_diff_max_coef.py
verif/SIM_VER_DRAG/Modified_data/one_plate_accel_mixed_eps05_max_coef.py
verif/SIM_VER_DRAG/Modified_data/one_plate_accel_spec_max_coef.py
verif/SIM_VER_DRAG/Modified_data/one_plate_torque.py
verif/SIM_VER_DRAG/Modified_data/shuttle_aero_model.py
verif/SIM_VER_DRAG/Modified_data/shuttle_plate_aero_drag.py
verif/SIM_VER_DRAG/Modified_data/shuttle_plate_orbiter.py
verif/SIM_VER_DRAG/Modified_data/temp_header
verif/SIM_VER_DRAG/Modified_data/two_sided_plate.py

Continued on next page

Table 3.4: Verification Files (continued from previous page)

File Name
verif/SIM_VER_DRAG/SET_test/RUN_aero_drag_BC/input.py
verif/SIM_VER_DRAG/SET_test/RUN_aero_drag_CD/input.py
verif/SIM_VER_DRAG/SET_test/RUN_aero_drag_const/input.py
verif/SIM_VER_DRAG/SET_test/RUN_one_plate_accel_calc_coef_eps00/input.py
verif/SIM_VER_DRAG/SET_test/RUN_one_plate_accel_calc_coef_eps05/input.py
verif/SIM_VER_DRAG/SET_test/RUN_one_plate_accel_calc_coef_eps1/input.py
verif/SIM_VER_DRAG/SET_test/RUN_one_plate_accel_diff_max_coef/input.py
verif/SIM_VER_DRAG/SET_test/RUN_one_plate_accel_mixed_eps05_max_coef/input.py
verif/SIM_VER_DRAG/SET_test/RUN_one_plate_accel_spec_max_coef/input.py
verif/SIM_VER_DRAG/SET_test/RUN_one_plate_torque/input.py
verif/SIM_VER_DRAG/SET_test/RUN_orbiter/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_aero_drag_BC/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_aero_drag_BC/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_aero_drag_BC/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_aero_drag_CD/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_aero_drag_CD/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_aero_drag_CD/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_aero_drag_const/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_aero_drag_const/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_aero_drag_const/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_calc_coef_eps00/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_calc_coef_eps00/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_calc_coef_eps00/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_calc_coef_eps05/input.py

Continued on next page

Table 3.4: Verification Files (continued from previous page)

File Name
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_calc_coef_eps05/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_calc_coef_eps05/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_calc_coef_eps1/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_calc_coef_eps1/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_calc_coef_eps1/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_diff_max_coef/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_diff_max_coef/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_diff_max_coef/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_mixed_eps05_max_coef/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_mixed_eps05_max_coef/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_mixed_eps05_max_coef/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_spec_max_coef/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_spec_max_coef/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_accel_spec_max_coef/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_torque/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_torque/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_one_plate_torque/log_drag.header
verif/SIM_VER_DRAG/SET_test_val/RUN_orbiter/input.py
verif/SIM_VER_DRAG/SET_test_val/RUN_orbiter/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val/RUN_orbiter/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_aero_drag_BC/input.py

Continued on next page

Table 3.4: Verification Files (continued from previous page)

File Name
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_aero_drag_BC/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_aero_drag_BC/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_aero_drag_CD/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_aero_drag_CD/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_aero_drag_CD/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_aero_drag_const/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_aero_drag_const/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_aero_drag_const/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_calc_coef_eps00/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_calc_coef_eps00/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_calc_coef_eps00/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_calc_coef_eps05/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_calc_coef_eps05/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_calc_coef_eps05/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_calc_coef_eps1/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_calc_coef_eps1/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_calc_coef_eps1/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_diff_max_coef/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_diff_max_coef/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_diff_max_coef/log_drag.header

Continued on next page

Table 3.4: Verification Files (continued from previous page)

File Name
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_mixed_eps05_max_coef/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_mixed_eps05_max_coef/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_mixed_eps05_max_coef/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_spec_max_coef/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_spec_max_coef/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_accel_spec_max_coef/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_torque/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_torque/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_one_plate_torque/log_drag.header
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_orbiter/input.py
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_orbiter/log_drag.csv
verif/SIM_VER_DRAG/SET_test_val_rh8/RUN_orbiter/log_drag.header
verif/include/aero_logging.hh
verif/src/aero_logging.cc

Chapter 4

User Guide

The Analysis section of the user guide is intended primarily for users of pre-existing simulations. It contains:

- A description of how to modify Aerodynamics Model variables after the simulation has compiled, including an in-depth discussion of the input file,
- An overview of how to interpret (but not edit) the S_define file,
- A sample of some of the typical variables that may be logged.

The Integration section of the user guide is intended for simulation developers. It describes the necessary configuration of the Aerodynamics Model within an S_define file, and the creation of standard run directories. The latter component assumes a thorough understanding of the preceding Analysis section of the user guide. Where applicable, the user may be directed to selected portions of Product Specification (Chapter 3).

The Extension section of the user guide is intended primarily for developers needing to extend the capability of the Aerodynamics Model. Such users should have a thorough understanding of how the model is used in the preceding Integration section, and of the model specification (described in Chapter 3).

4.1 Analysis

This section will assume an S_define object of the following form:

```
sim_object {

   interactions/aerodynamics: AerodynamicDrag aero_drag
      (interactions/aerodynamics/data/aero_model.d);

   utils/surface_model: SurfaceModel surface;
   interactions/aerodynamics: AeroSurfaceFactory surface_factory;
```

```
   interactions/aerodynamics: AeroSurface aero_surface;

   utils/surface_model: Facet ** facet_ptr;
   unsigned int integer;
   (0.0, environment) utils/surface_model:
   aerodynamics.surface.add_facets(
      In Facet** new_facets = aerodynamics.facet_ptr,
      In unsigned int num_new_facets = aerodynamics.integer
      );

   utils/surface_model: FacetParams * facet_params;
   (0.0, environment) interactions/aerodynamics:
   aerodynamics.surface_factory.add_facet_params(
      In FacetParams* to_add = aerodynamics.facet_params
      );

} aerodynamics;
```

The default data file associated with the AerodynamicDrag object sets the gas constant and the temperature of the free stream the aerodynamic object will use to calculate the drag. Additionally, it sets the active flag to 'true', so the aerodynamic drag will be calculated. The form of these calls in the default data file is:

```
AerodynamicDrag.param.gas_const = 287 ;
AerodynamicDrag.param.temp_free_stream = 1487 ;
AerodynamicDrag.active = true;
```

These values can be overriden, of course, in the input file. The units for the gas constant are N*M/kg/K, and the temperature of the free stream is in Kelvin.

There are two cases for setup, one for the simple case of ballistic coefficient or drag coefficient, and one for the case of a higher fidelity flat plate model.

4.1.1 Simple Aerodynamic Drag

The Aerodynamics Model gives three simple cases for calculating aerodynamic drag:

- Coefficient of Drag (CD), as described in Equation (3.1),
- Ballistic Coefficient (BC), as described in Equation (3.3),
- Constant Drag, a constant force in the direction of the relative velocity of the vehicle.

If one of these simple formulations for aerodynamic drag is desired, the AerodynamicDrag object must be told to use the default behavior. In the above code example, this can be done with the following command:

```
aerodynamics.aero_drag.use_default_behavior = true;
```

This is set to 'true' by default. Additionally, the user must ensure that the AerodynamicDrag object is active. In the above example, this can be done with the following command:

```
aerodynamics.aero_drag.active = true;
```

This is the default behavior. The AerodynamicDrag object can also be turned off by setting this variable to 'false', as it is a simple 'bool' type.

By default, the constant drag option is chosen with a drag of 0.0. The following sections will explain how to use each option from a Trick input file.

4.1.2 Coefficient of Drag (CD)

Three items must be set to use the coefficient of drag (CD) option with the simple aerodynamic drag module. First, the option of using CD must be set. This can be done in the above code using the following command:

```
aerodynamics.aero_drag.ballistic_drag.option = DefaultAero::DRAG_OPT_CD;
```

The second and third items to be set are the CD and the effective aerodynamic area must then be set, or they will stay defaulted to zero. In the above example, these commands would be:

```
aerodynamics.aero_drag.ballistic_drag.Cd = 2.0; // your coefficient of drag
aerodynamics.aero_drag.ballistic_drag.area = 1400.0; // your effective
aerodynamic area
```

It should be noted that, of course, the values being set are representative only. The area will automatically convert to meters squared if the user leverages the Trick unit conversions tool; otherwise the value for area must be inputted in meters squared.

4.1.3 Ballistic Coefficient (BC)

Two items must be set to use the ballistic coefficient (BC) option with the simple aerodynamic drag module. First, the option of using BC must be set. This can be done in the above code using the following command:

```
aerodynamics.aero_drag.ballistic_drag.option = DragOption::DRAG_OPT_BC;
```

The BC must then be set, or it will stay defaulted to zero. In the above example, these commands would be:

```
aerodynamics.aero_drag.ballistic_drag.BC = 166.0; // your ballistic coefficient
```

It should be noted that, of course, the value being set is representative only, and that the user should use an appropriate value for their simulation.

Please note this warning about using the BC option. Because the formulation for using BC involves a divide by the coefficient, it is imperative that the BC itself be set to something other than zero. If this is not the case then a failure will be issued through the JEOD Message Handler [8].

4.1.4 Constant Drag

Two items must be set to use the constant drag option with the simple aerodynamic drag module. First, the option of using constant drag must be set. This can be done in the above code using the following command:

```
aerodynamics.aero_drag.ballistic_drag.option = DefaultAero::DRAG_OPT_CONST;
```

The constant drag value must then be set, or it will stay defaulted to zero. In the above example, these commands would be:

```
aerodynamics.aero_drag.ballistic_drag.drag = 200.0; // your constant drag
```

It should be noted that, of course, the value being set is representative only, and that the user should use an appropriate value for their simulation.

The drag will automatically convert to Newtons if the user leverages the Trick unit conversions tool; otherwise the value for drag must be entered in Newtons.

4.1.5 Flat Plate Model Aerodynamic Drag

The flat plate model for aerodynamic drag implements the JEOD Surface Model [9] for the flat plate formulation of aerodynamic drag presented in the Mathematical Formulation section.

There are a few steps needed to create a flat plate surface model appropriate for use with the aerodynamics package:

- Define the flat plates contained in the surface model,
- Define the aerodynamic parameters necessary to convert a flat plate to an aerodynamic drag specific flat plate,
- Setup the AerodynamicDrag object to use the aerodynamic specific flat plate model created from the surface model.

The following is an example of creating a flat plate based aerodynamic surface model. This example utilizes the S_define example contained at the start of this section.

The following code illustrates creating two flat plates in a non-interaction specific surface model.

```
#define NUM_PLATE 2

FlatPlate** temp_facets;
temp_facets = alloc(NUM_PLATE);
temp_facets[0] = new FlatPlate[1];
temp_facets[1] = new FlatPlate[1];

temp_facets[0]->position[0] {M} = 0.0,  0.0,   0.0;
temp_facets[0]->area {M2} = 1.0;
temp_facets[0]->normal[0] = 1.0 , 0.0 , 0.0;
temp_facets[0]->param_name = "flat_plate_material";
temp_facets[0]->temperature = 70;

temp_facets[1]->position[0] {M} = 0.0,  0.0,   0.0;
temp_facets[1]->area {M2} = 1.0;
temp_facets[1]->normal[0] = -1.0 , 0.0 , 0.0;
temp_facets[1]->param_name = "flat_plate_material";
temp_facets[1]->temperature = 70;

aerodynamics.facet_ptr = temp_facets;
aerodynamics.integer = NUM_PLATE;
call aerodynamics.aerodynamics.surface.add_facets(
   aerodynamics.facet_ptr);
```

This specifies two flat plates, both with an area of one meter squared, a temperature of 70 Kelvin, and both utilizing the parameter name "flat_plate_material". This parameter name must, of course, match the name of the parameters that will be sent to the interaction surface factory in the next step.

The next step is to specify the parameters necessary for creating an aerodynamic specific flat plate from a generic flat plate. These parameters will dictate how the flat plate behaves with the environment, using the concepts presented for aerodynamic drag of flat plates in the Mathematical Formulations section. These paramters are set in the FlatPlateAeroParams object, and are as follows:

- coef_method, the method used to determine the method that will dictate how the aerodynamic drag is calculated,
- calculate_drag_coef, a bool that determines if the algorithm calculates its own coefficients of drag, otherwise it will leave the user specified values,
- epsilon, a fraction out of one that will be used if "coef_method" is set to either "mixed" or "Calc_coef",
- temp_reflect, the temperature of molecules reflected specularly off the plate,

- drag_coef_norm, a coefficient of drag used if "coef_method" is set to Calc_coef,
- drag_coef_tang, a coefficient of drag used if "coef_method" is set to Calc_coef,
- drag_coef_spec, a coefficient of drag used if "coef_method" is set to Specular or Mixed,
- drag_coef_diff, a coefficient of drag used if "coef_method" is set to Diffuse or Mixed.

where the variable "coef_method" has the following options:

- Specular, where only specular type drag, as described in the Mathematical Formulations section, will be used. If "calculate_drag_coef" is set to true, the coefficient will be calculated, otherwise it will be left based on whatever value is currently set in "drag_coef_spec".
- Diffuse, where only diffuse type drag, as described in the Mathematical Formulations section, will be used. If "calculate_drag_coef" is set to true, the coefficient will be calculated, otherwise it will be based on whatever value is currently set in "drag_coef_diffuse".
- Mixed, where a mix of specular and diffuse are used, based on the epsilon value set, and as described in the Mathematical Formulations section. If "calculate_drag_coef" is set to true, these two coefficients will be calculated, otherwise they will be based on whatever values are currently set in "drag_coef_diffuse" and "drag_coef_spec".
- Calc_coef, where the normal and tangential forces are calculated as described in Equation (3.11). This option will always calculate the coefficients found in "drag_coef_tang" and "drag_coef_norm", and calculate the full force from a combination of these two based on epsilon, as seen in Equations (3.11) and (3.12).

These paramters are set by creating a FlatPlateAeroParams object and giving it to the InteractionSurfaceFactory responsible for creating the AeroSurface that will be used. An example is shown in the following set of input code:

```
#define SPEC_DRAG_COEF 5.0
#define DIFF_DRAG_COEF 5.0

FlatPlateAeroParams* params;
params = new FlatPlateAeroParams;

params->drag_coef_norm = SPEC_DRAG_COEF;
params->drag_coef_spec = SPEC_DRAG_COEF;
params->drag_coef_tang = DIFF_DRAG_COEF;
params->drag_coef_diff = DIFF_DRAG_COEF;
params->epsilon = 0.0 ;
params->coef_method = AeroDragEnum::Calc_coef ;
params->calculate_drag_coef = No;
params->name = "flat_plate_material";
```

```
aerodynamics.facet_params = params;
call aerodynamics.aerodynamics.surface_factory.add_facet_params(
   aerodynamics.facet_params);

#undef DIFF_DRAG_COEF
#undef SPEC_DRAG_COEF
```

This example creates a single, aerodynamic flat plate specific parameter object, populates it and adds it to the surface factory that is responsible for creating an aerodynamic specific surface model from the generic surface model.

Note that the name "flat_plate_material" matches the name given for the "param_name" supplied earlier to the flat plate objects. If there is not a parameters object added to the interaction surface factory object with the correct name for a flat plate, then there will be a failure upon initialization of the sim.

Note also that, if more than one set of parameters is desired, this process can be repeated, with a different name given to the set of parameters.

Finally, the AerodynamicDrag object must be told to use the created aerodynamic interaction surface model. First, the aerodynamics object must be told to not use the default aerodynamic drag behavior. This can be done with, assuming the earlier S_define object, the following call:

```
aerodynamics.aero_drag.use_default_behavior = false;
```

The AerodynamicDrag object must then be told what aerodynamic interaction surface to use, done using the following call:

```
aerodynamics.aero_drag.aero_surface_ptr = &aerodynamics.aero_surface;
```

The AerodynamicDrag object will now use the newly created aerodynamic interaction surface.

The applicable output from the AerodynamicDrag object, for an analyst, is minimal. The total aerodynamic force and torque can be read from the AerodynamicDrag object, in Newtons and Newton-meters, as shown in the following statements:

```
aerodynamics.aero_drag.aero_force;
aerodynamics.aero_drag.aero_torque;
```

The form of this output is arrays of doubles, of length three.

If the default aerodynamic behavior is used, the magnitude of the calculated drag can also be accessed, using the following call:

```
aerodynamics.aero_drag.ballistic_drag.drag;
```

This value will be a scalar, with units of Newtons.

Because of the dynamically allocated and polymorphic nature of the the Surface Model [9], Trick has limited access to the aerodynamic interaction facets that make up an aerodynamic interaction surface. However, it is possible to access features of the individual FlatPlateAeroFacets in the aerodynamic interaction surface using a logging object, included in the S_define, and set up to cache dynamically casted pointers through which the parameters of the FlatPlateAeroFacets can then be copied for logging. The process for this will be briefly described below, in the Integration section.

If this has been done, the following parameters can be accessed through a FlatPlateAeroFacet:

- force_n, the last calculated force normal to the plate.
- force_t, the last calculated force either tangential to the plate, or parallel to the velocity vector (parallel to the velocity in the case of diffuse or mixed drag, tangential in all other cases).
- drag_coef_norm, which has been calculated by the aerodynamics module if coef_method was set to Calc_coef,
- drag_coef_tang, which has been calculated by the aerodynamics module if coef_method was set to Calc_coef,
- drag_coef_spec, which was calculated by the aerodynamics module if coef_method was set to Specular or Mixed, and calculate_drag_coef was set to true, and
- drag_coef_diff, which was calculated by the aerodynamics module if coef_method was set to diffuse or mixed, and calculate_drag was set to true.

Note that these are only the parameters that are calculated during the aerodynamic drag functions. Other parameters are present in the FlatPlateAeroFacet object that were set indirectly by the user during the aerodynamic interaction surface creation. Details of these values can be found in the Deatiled Design section.

In addition to the aerodynamic parameters presented here, atmospheric parameters that contribute to aerodynamic drag can be accessed through the atmosphere object being used to calculate drag. Further details on what is available in the atmosphere state objects are available in the Atmosphere Model documentation [10].

4.2 Integration

This section will assume an example S_define object of the following form:

```
sim_object {

   /* Vehicle orbital dynamics parameters. */
   double inertial_vel[3];
   double T_inertial_struct[3][3];
```

```
   double center_grav[3]; /* M center of grav */
   double mass;

   interactions/aerodynamics:          AerodynamicDrag           aero_drag
               (interactions/aerodynamics/data/aero_model.d);

   environment/atmosphere: AtmosphereState atmos_state;

   utils/surface_model: SurfaceModel surface;
   interactions/aerodynamics: AeroSurface aero_surface;
   interactions/aerodynamics: AeroSurfaceFactory surface_factory;


   utils/surface_model: Facet ** facet_ptr;
   utils/surface_model: FlatPlate * flat_plate;
   unsigned int integer;
   (0.0, environment) utils/surface_model:
   aerodynamics.surface.add_facets(
      In Facet** new_facets = aerodynamics.facet_ptr,
      In unsigned int num_new_facets = aerodynamics.integer
      );

   interactions/aerodynamics: FlatPlateAeroParams * aero_params;
   utils/surface_model: FacetParams * facet_params;
   (0.0, environment) interactions/aerodynamics:
   aerodynamics.surface_factory.add_facet_params(
      In FacetParams* to_add = aerodynamics.facet_params
      );

   (initialization) interactions/aerodynamics:
      aerodynamics.surface_factory.create_surface(
         In SurfaceModel * surface = &aerodynamics.surface,
         Out InteractionSurface * inter_surface = &aerodynamics.aero_surface);

   (DYN, scheduled) interactions/aerodynamics: aerodynamics.aero_drag.aero_drag(
       In double inertial_velocity[3]  = aerodynamics.inertial_vel,
       In AtmosphereState* atmos_state = &aerodynamics.atmos_state,
       In double T_inertial_struct[3][3] = aerodynamics.T_inertial_struct,
       In double mass = aerodynamics.mass,
       In double center_grav[3] = aerodynamics.center_grav);

} aerodynamics;
```

Note that there is an AtmosphereState object included in this S_define example. The details for using this object will not be discussed here, and can be found in the Atmosphere Model documentation [10].

If all that is required is a simple ballistic coefficient version of aerodynamic drag, then all that must be done is an aerodynamic drag instantiated, and the aerodynamic drag function called with the appropriate parameters. This simple instantiation and call is demonstrated in the following S_define components:

```
interactions/aerodynamics:          AerodynamicDrag          aero_drag
             (interactions/aerodynamics/data/aero_model.d)
```

The default data file included in this instantiation is explained in the Analysis section.

The call to calculate aerodynamic drag, which must be included for any implementation, will then be the following:

```
(DYN, scheduled) interactions/aerodynamics: aerodynamics.aero_drag.aero_drag(
    In double inertial_velocity[3]  = aerodynamics.inertial_vel,
    In AtmosphereState* atmos_state = &aerodynamics.atmos_state,
    In double T_inertial_struct[3][3] = aerodynamics.T_inertial_struct,
    In double mass = aerodynamics.mass,
    In double center_grav[3] = aerodynamics.center_grav);
```

The first parameter is the velocity of the vehicle in question, in meters per second, in its current inertial frame. The third parameter is the transformation matrix from the inertial frame to the vehicle structural frame. The fourth parameter is the mass, in kg, and the last parameter is the location of the vehicle center of gravity, in M, in the structural frame. These parameters are being filled with dummy values in this example, but they will most often be obtained from the DynBody object associated with the vehicle in question. More information on this can be found in the Dynamics Body documentation [4].

If the full plate model for aerodynamics is required, then additional integration will be required. First the parts for the surface models (both general and aerodynamic specific) are required:

```
utils/surface_model: SurfaceModel surface;
interactions/aerodynamics: AeroSurface aero_surface;
interactions/aerodynamics: AeroSurfaceFactory surface_factory;
```

Note that these instantiations also include the AeroSurfaceFactory that will be used to produce the AeroSurface object from the SurfaceModel object.

Next, the hooks for dynamically allocating flat plates and adding them to the SurfaceModel object through an input file are added:

```
utils/surface_model: Facet ** facet_ptr;
utils/surface_model: FlatPlate * flat_plate;
unsigned int integer;
(0.0, environment) utils/surface_model:
aerodynamics.surface.add_facets(
```

```
   In Facet** new_facets = aerodynamics.facet_ptr,
   In unsigned int num_new_facets = aerodynamics.integer
   );
```

Note that the FlatPlate pointer 'flat_plate' is never legitimately used. It is only included so that Trick is aware of its existence and so that it can be dynamically allocated in the input file. This is described further, along with additional information about the parameters given to the surface model during the 'add_facets' routine, in the Surface Model documentation [9].

Next, the hooks to dynamically add aerodynamic parameters used to create the aerodynamic interaction facets must be added to the S_define, using the following code:

```
interactions/aerodynamics: FlatPlateAeroParams * aero_params;
utils/surface_model: FacetParams * facet_params;
(0.0, environment) interactions/aerodynamics:
aerodynamics.surface_factory.add_facet_params(
   In FacetParams* to_add = aerodynamics.facet_params
   );
```

Similar to the FlatPlate pointer described above, the FlatPlateAeroParams pointer seen here is never used and is only necessary for Trick awareness.

Finally, the call to create the aerodynamic interaction surface from the user defined surface model object must be added:

```
(initialization) interactions/aerodynamics:
   aerodynamics.surface_factory.create_surface(
      In SurfaceModel * surface = &aerodynamics.surface,
      Out InteractionSurface * inter_surface = &aerodynamics.aero_surface);
```

The aerodynamic interaction surface model is now available to be used for drag calculations.

Note that it is also possible to create an aerodynamic specific surface model from a general surface model made up of FlatPlateThermal facets [9], be replacing all instances of FlatPlate in the above example with FlatPlateThermal. This is necessary when using the same generic SurfaceModel to create both an interaction surface for aerodynamics as well as one for radiation pressure (further details of setting up radiation pressure can be found in the appropriate documentation document [12]).

One additional thing that can be done in the integration is to create an object that allows for logging of the parameters found in the dynamically allocated FlatPlateAeroFacets contained in the aerodynamic interaction surface. While basic C++ object oriented and polymorphism knowledge will be necessary for this, the applicable information from the aerodynamic interaction surface necessary for this will be briefly described.

After the 'create_surface' call has been made, the AeroSurface object will hold two key parameters, which are:

```
AeroFacet** aero_facets;
unsigned int facets_size;
```

AeroFacet is a pure virtual class, so 'aero_facets' is an array of pointers to AeroFacet derived classes. The most common deriving class used, and the ones supplied with the model, is the FlatPlateAeroFacet, an aerodynamic specific version of a FlatPlate [9] object. The unsigned int 'facets_size' is the size of this array of pointers.

A logging object must then be created to cache the pointers that are stored in the 'aero_facets' array in a Trick loggable form. This can be done by having knowledge of what will be created in the 'aero_facets' array, dynamically casting these pointers to their actual, inherited type at initialization, and storing these pointers in a logging object that is Trick accessible.

4.3 Extension

There are two avenues to extension for the Aerodynamics Model model. One is to extend the AerodynamicDrag object itself to accept a different default behavior. The other is to extend the Surface Model. Both of these paths will be discussed here.

Note that this section will use the same example S_define found in the Integration section.

4.3.1 Extending the Default Behavior

The AerodynamicDrag object, will use what it knows as the default aerodynamic behavior if the parameter 'use_default_behavior' is set to true. This behavior is dictated by the following parameter:

```
DefaultAero* default_behavior;
```

The basic aerodynamic model contains an instantiation of DefaultAero, which implements the coefficient of drag and ballistic coefficient forms of aerodynamic drag described in this document. This is, by default, where the 'default_behavior' parameter points.

DefaultAero is, however, a virtual class, thus allowing the 'default_behavior' pointer to also point to any class that inherits from DefaultAero.

Thus, to extend the model all one must do is create a class that inherits from DefaultAero, and overrides the following virtual function:

```
virtual void aerodrag_force (
   const double velocity_mag,
   const double rel_vel_hat[3],
   AeroDragParameters* aero_drag_param_ptr,
   double mass,
   double force[3],
   double torque[3]);
```

This overriden function must calculate the aerodynamic drag, both in terms of force and torque, and store it in the function parameters 'double force[3]' and 'double torque[3]'. Then all that needs to happen is for the AerodynamicDrag pointer 'default_behavior' to get set to point to a newly created object of the inheriting class, and the Aerodynamics Model will automatically use the newly defined behavior.

It is possible that a user will require other parameters for the calculation of aerodynamic drag. If this is the case, then additional functions where these paramters are set can be added to the class that inhertis from DefaultBehavior, and these functions can be called as scheduled jobs in the S_define before the AerodynamicDrag::aero_drag function is called.

4.3.2 Extending the Surface Model for Aerodynamics

The extension of the surface model for specific interactions is well covered in the Surface Model documentation [9]. This section will give a brief overview of the structure set that is already in place for the aerodynamic interaction, and briefly describe how to extend it.

Three classes must be created to extend the aerodynamic surface model to a new calculation of aerodynamic drag. These classes will inherit from the following base classes:

- AeroParams, an inheriting example being FlatPlateAeroParams,
- AeroFacet, an example inheriting class being FlatPlateAeroFacet, and
- InteractionFacetFactory, an example inheriting class being FlatPlateAeroFactory.

The following sections will briefly describe what must be done in these inheriting classes.

Inheriting from AeroParams

There are no functional requirements on inheriting from AeroParams. All that must be done is to add the data fields required for instantiating and using the AeroFacet inheriting class described below. It is intended that a user would then fill out these parameters, and load the object onto an aerodynamic surface factory as described in the Analysis and Integration sections.

Inheriting from AeroFacet

AeroFacet, as discussed in the Integration Section, is a pure virtual class. Any inheriting class must implement the following function:

```
virtual void aerodrag_force (
   const double velocity_mag,
   const double rel_vel_hat[3],
   AeroDragParameters* aero_drag_param_ptr,
   double center_grav[3]);
```

This function must take the parameters given and calculate the aerodynamic force and torque felt by the facet, and store it in the inherited data fields:

```
double force[3];
double torque[3];
```

Additionally, any data fields needed to calculate the aerodynamic drag for this facet can be added, and set by the user at runtime through AeroParams inherited class described in the previous section, using the method described in the Analysis section, the Integration section and the Surface Model documentation [9].

Inheriting from InteractionFacetFactory

InteractionFacetFactory is a pure virtual class. Any classes that inherit from it must implement the following pure virtual functions:

```
virtual bool is_correct_factory(Facet* facet);
virtual InteractionFacet* create_facet(Facet* facet, FacetParams* params)
```

'is_correct_factory' is a simple implementation. As described in the Surface Model documentation, every InteractionFacetFactory is designed to turn one type of Facet (a class inheriting from Facet) into an interaction specific (in this case, aerodynamics) equivalent of that facet. This function takes in a pointer to a facet, and the function checks if this facet is of the type that this Interaction-FacetFactory is designed to use, then the function returns true. Otherwise, the function returns false.

'create_facet' takes a pointer to a basic Facet type, a pointer to a basic FacetParams type, and creates an interaction facet from these two items. The normal flow of these operations is to:

- Check if the FacetParams object is of the appropriate type,
- Check if the Facet object is of the appropriate type,
- Create the interaction facet,
- Populate any information needed in the interaction facet, and return it to the invoker of the function.

An example of this implementation can be found in the source code for the FlatPlateAeroFactory class.

Once these classes are implemented they can be used as described in the Analysis and Integration sections as well as the Surface Model documentation [9].

Chapter 5

Verification and Validation

5.1 Verification

Inspection aerodynamics_1: Top-level inspection

This document structure, the code, and associated files have been inspected, and together satisfy requirement aerodynamics_1.

Inspection aerodynamics_2: Mathematical Formulation

Collected data:
The Aerodynamics Model functions calculate collected aerodynamic forces and torques in agreement with section 10.0 of the Trick User's Guide [14].

Forces and Torques:
The Aerodynamics Model routines calculate total external forces and torques as described by the mathematical formulations in the Aerodynamics Model Requirements Section.

Inspection aerodynamics_3: Aero Drag

The Aerodynamics Model header file *aero_drag.h* provides means for using simple aerodynamic drag using the options:

```
DRAG_OPT_CD    = 0,  Use Coefficient of drag for drag computations.
DRAG_OPT_BC    = 1,  Use Ballistic Coefficient for drag computations.
DRAG_OPT_CONST       Use specified constant drag.
```

to select either the coefficient of drag or ballistic coefficient.

Inspection aerodynamics_4: Free Molecular Flow

The Aerodynamics Model provides means for using simple aerodynamic drag using the option:

```
For exact free molecular flow
Coef_method coef_method          Flag indicating which method of coef
                                 calculation to use, specular, diffuse,
                                 calculated, and mixed.
```

Inspection aerodynamics_5: Aerodynamic Extension

The Aerodynamics Model through its extension of the JEOD Surface Model [9], provides an extensible architecture for specifying implementations of aerodynamic drag. Additionally, the class "DefaultAero" also provides extensible functionality. These frameworks fulfil requirement aerodynamics_4.

5.2 Validation

Various tests are conducted to verify and validate that the Aerodynamics Model satisfies the requirements. All verification and validation test source code, simulations and procedures are archived in the Dynamics package directory.

5.3 Single Plate Tests

The purpose of the following tests is to assess the case where a coefficient of drag or ballistic coefficient is input along with a constant atmospheric density and see if the calculations are performed correctly. In the following test cases a simple plate is used, the atmospheric density is constant and a relative velocity is specified.

Test aerodynamics_1: Simple Drag

Test directory:

`models/interactions/aerodynamics/verif/SIM_VER_DRAG/SET_test`

Test description:
By hand the drag acceleration is computed from:

$$\mathbf{a}_{drag} = -\frac{1}{2}\frac{c_d A_d}{m}\rho v_{rel}^2 \frac{\mathbf{v}_{rel}}{|\mathbf{v}_{rel}|} \tag{5.1}$$

For this case a hand computation was done using coefficient of drag of 2.0 and a ballistic coefficient of 0.005.

Test directory:

`RUN_aero_drag`

Success criteria:
The value the drag from 5.1 by hand calculation is 0.00562 Newtons, and the model calculation should match this.

Test results:
The value from the run is 0.00562 Newtons for both a coefficient of drag of 2.0 and a ballistic coefficient of 0.005. This unit test is satisfied.

Test aerodynamics_2: Approximate Free Molecular Flow Specular Case

Test directory:

`models/interactions/aerodynamics/verif/SIM_VER_DRAG/SET_test`

Test description:
Using the following input file:

`/RUN_one_plate_accel_spec_max_coef`

This tests the option for the approximate free molecular flow modeling in the full specular case using the model for the aerodynamic surfaces requirements. A plate is rotated 180 degrees in angle of attack in the streamflow and the body accelerations are recorded. An independent model of the free molecular flow for a plate found in Bird [1] (Equations 3.11 and 3.12) were coded in a FORTRAN program and the benchmark normal and axial acceleration data generated.

Success criteria:
There should be very close agreement between the exact theoretical computation for specular reflection and the approximate simulator value.

Test results:
The test co-plots are shown in figure 5.1. There is no discernible difference between the simulation and the exact theoretical calculation. By inspection and hand calculation the differences are only one part in 10^{-6}.

Test aerodynamics_3: Approximate Free Molecular Flow Diffuse Case

Test directory:

`models/interactions/aerodynamics/verif/SIM_VER_DRAG/SET_test`

Test description:
Using the following input file:

`RUN_one_plate_accel_diff_max_coef`

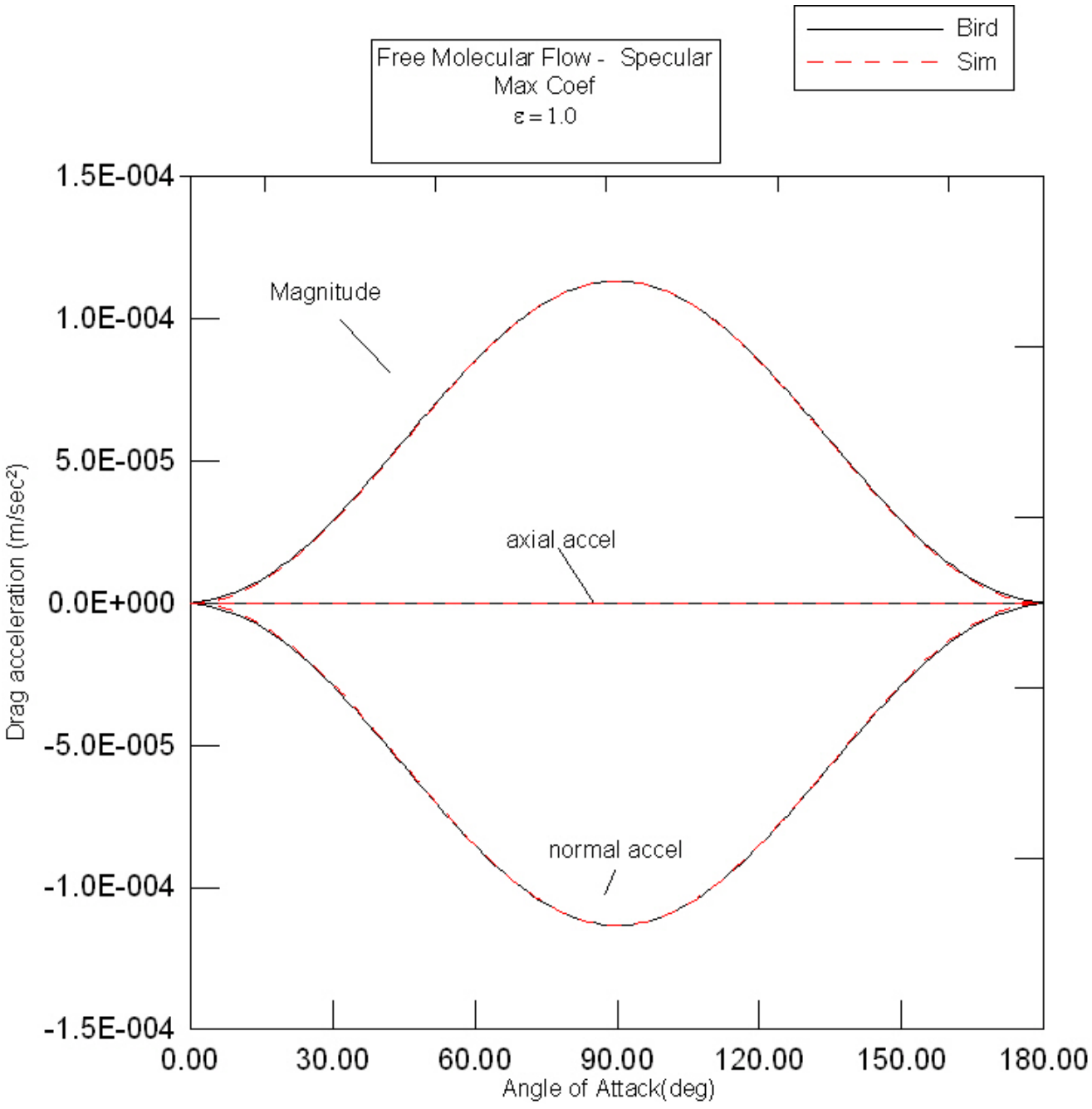

Figure 5.1: Approximate Free Molecular Flow

This tests the option for the approximate free molecular flow modeling in the full diffuse case using the model in the aerodynamic requirements. A plate is rotated 180 degrees in angle of attack in the streamflow and the accelerations are recorded. An independent model of the free molecular flow for a plate found in Bird [1] (Equations 3.11 and 3.12) was coded in a FORTRAN program and the benchmark normal and axial acceleration data generated.

Success criteria:

There should be very close agreement between the exact theoretical computation for specular reflection and the approximate simulator value.

Test results:

The test co-plots are shown in the following figure 5.2: There is only a small discernible difference between the simulation and the exact theoretical calculation. By inspection and hand calculation the differences are only one part in 10^{-6}.

Test aerodynamics_4: Approximate Free Molecular Flow Mixed Case

Test directory:

`models/interactions/aerodynamics/verif/SIM_VER_DRAG/SET_test`

Test description:

Using the following input file:

`RUN_one_plate_accel_mixed_eps05_max_coef`

This tests the option for the approximate free molecular flow modeling in the mixed case, with the parameter ϵ set to 0.5 giving a 50 percent specular and 50 percent diffuse mixed case. A plate is rotated 180 degrees in angle of attack in the streamflow and the accelerations are recorded. An independent model of the free molecular flow for a plate in Bird [1]. Equations 3.11 and 3.12 was coded in FORTRAN and the benchmark data generated.

Success criteria:

There should be very close agreement between the exact theoretical computation for specular reflection and the approximate simulator value.

Test results:

The test co-plots are shown in figure 5.3: There is a discernible difference between the simulation and the exact theoretical calculation at the end points in figure 5.3. However the simulation matches the exact model very well as is seen in the difference in the magnitudes shown in figure 5.4. In can be seen that the differences occur mostly at the end points, but the differences are less that $2 \cdot 10^{-6}$. *Causes of the end point differences are unknown at this time.*

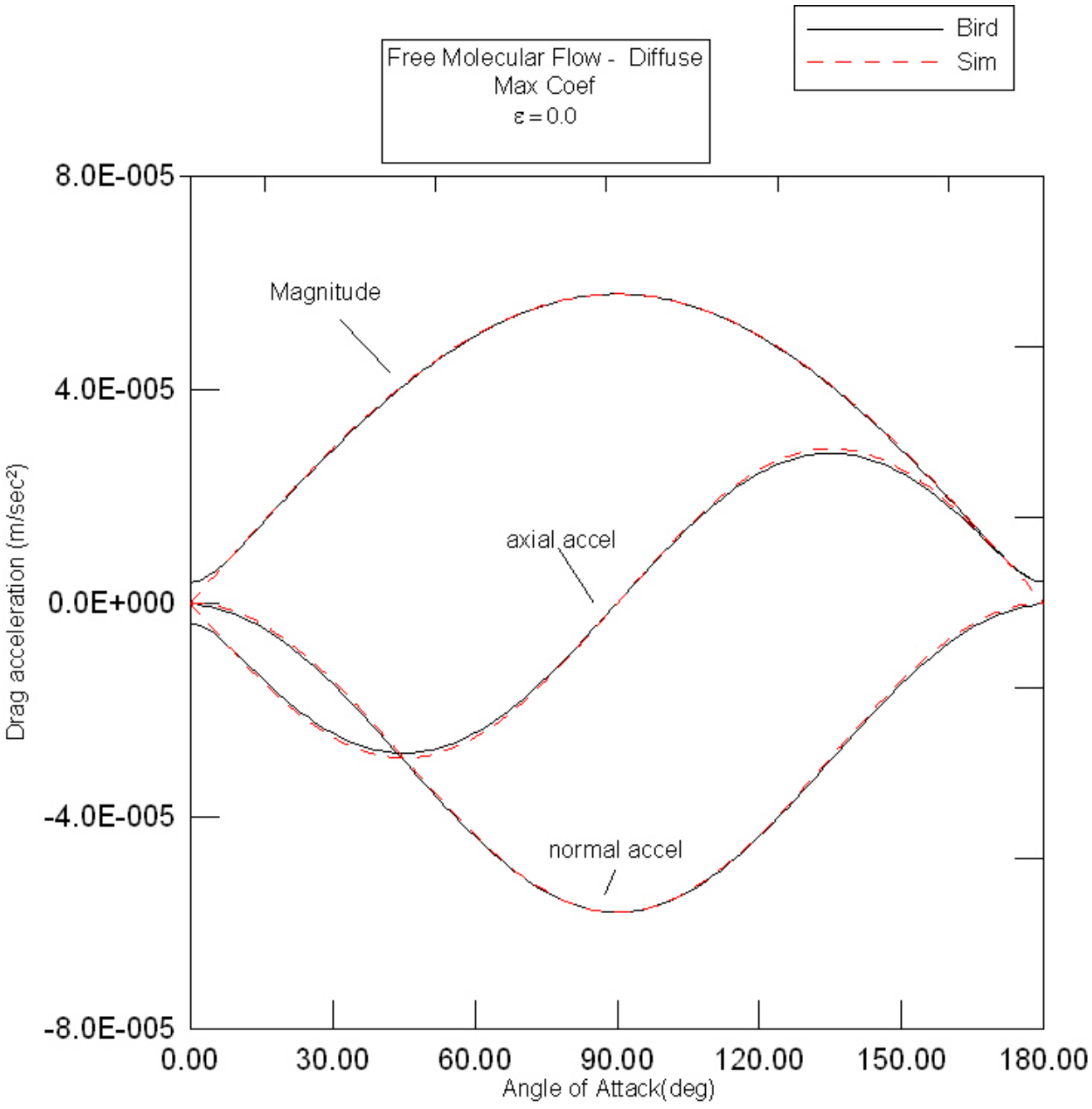

Figure 5.2: Approximate Diffuse Free Molecular Flow

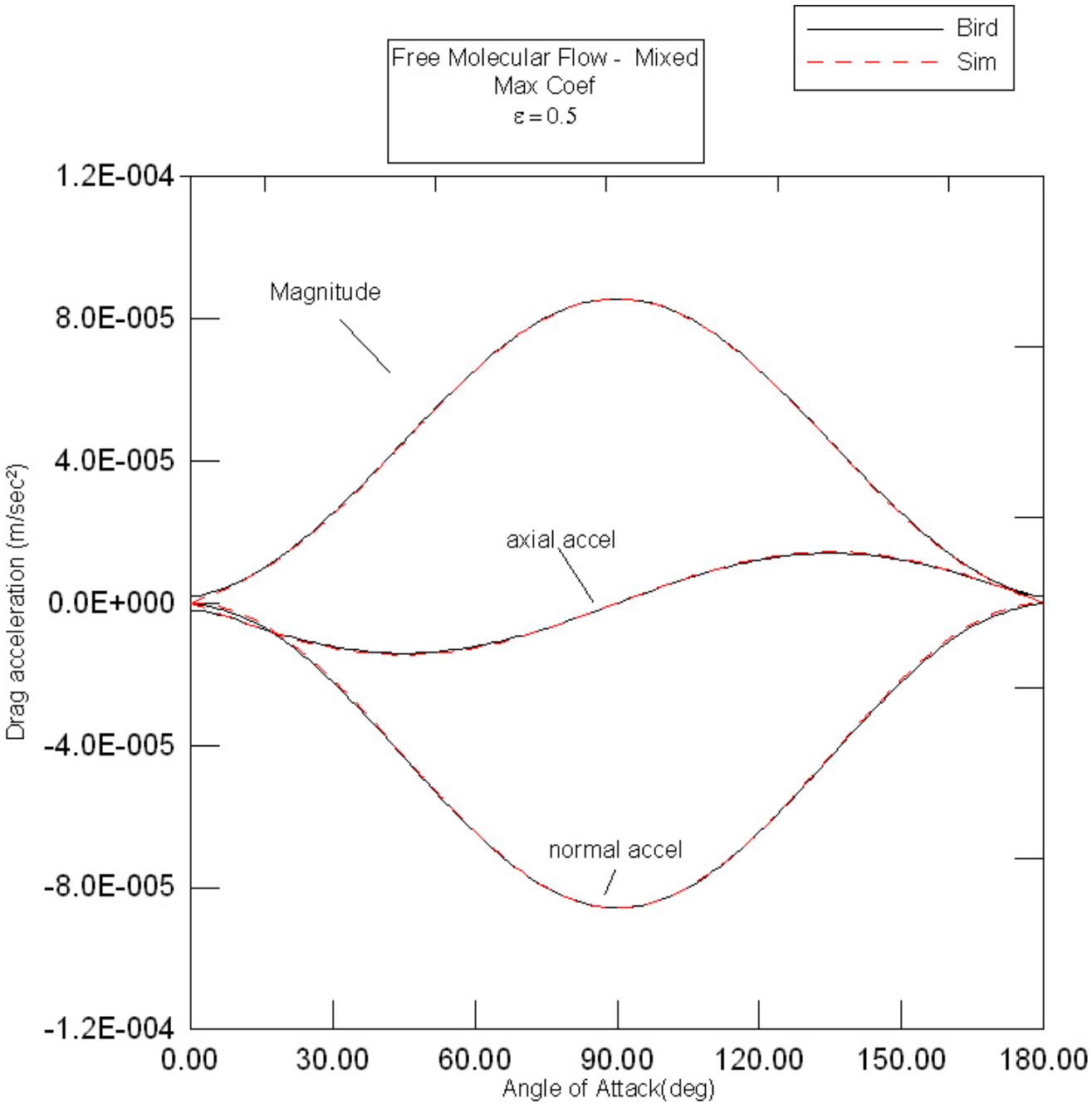

Figure 5.3: Approximate Mixed Free Molecular Flow

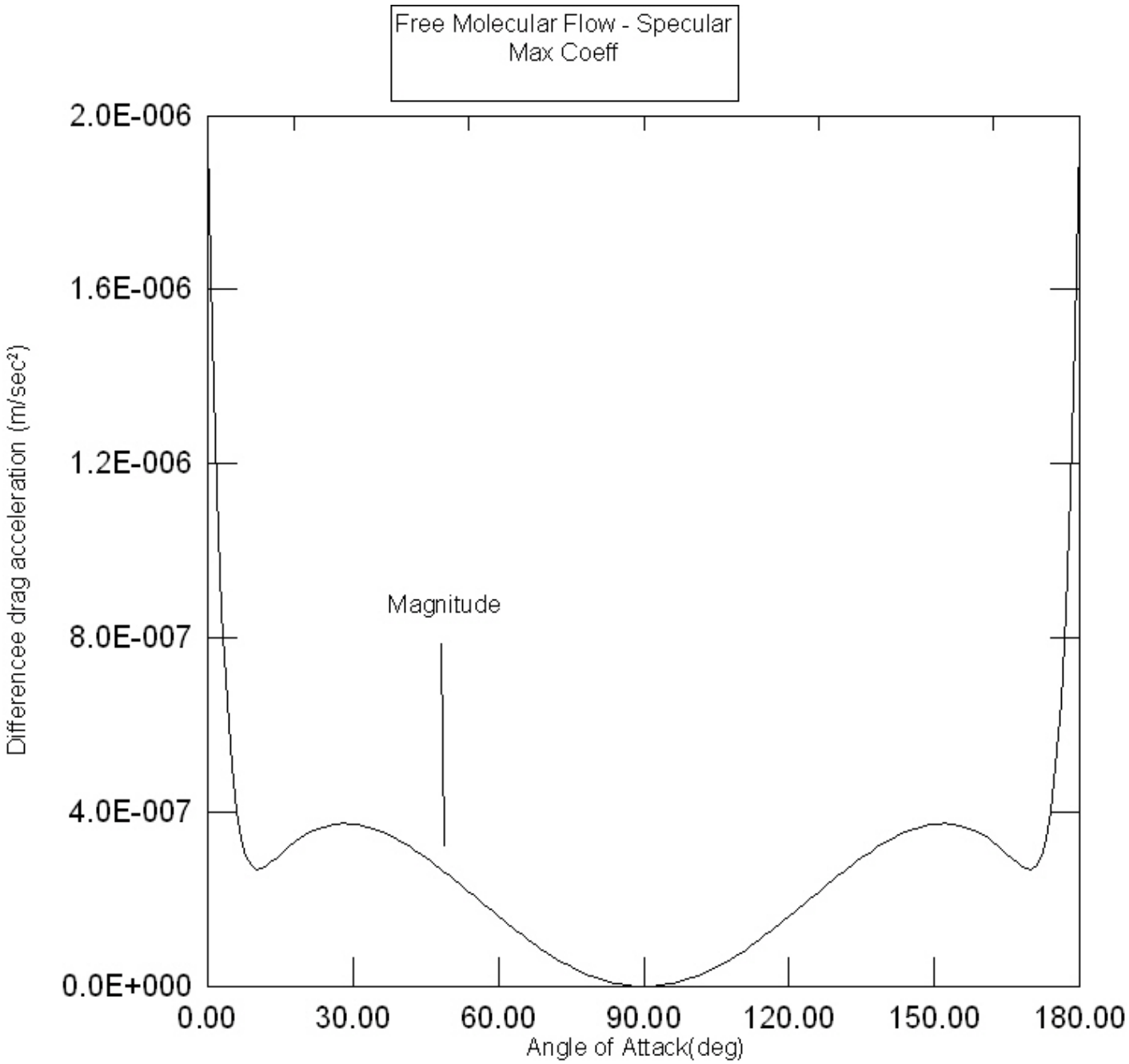

Figure 5.4: Approximate Mixed Free Molecular Flow Difference Plot

Test aerodynamics_5: Free Molecular Flow Calculate-Coefficient Specular Case

Test directory:

`models/interactions/aerodynamics/verif/SIM_VER_DRAG/SET_test`

Test description:
Using the following input file:

`RUN_one_plate_accel_calc_coef_eps1/`

This tests the option for the free molecular flow modeling in the full specular case, using the calculate coefficient option. with the parameter ϵ set to 1.0 giving the exact specular flow case. A plate is rotated 180 degrees in angle of attack in the streamflow and the accelerations are recorded. An independent model of the free molecular flow for a plate found in Bird [1] (Equations 3.11 and 3.12 was coded in FORTRAN and the benchmark data generated.

Success criteria:
There should be very close agreement between the exact theoretical computation for specular reflection and the simulator value.

Test results:
The test co-plots are shown in the following figure: There is no discernible difference between the simulation and the exact theoretical calculation are shown in figure 5.5 the axial component, for pure specular reflection is zero in both cases. No difference comparison needs to be made.

Test aerodynamics_6: Free Molecular Flow Calculate-Coefficient Diffuse Case

Test directory:

`models/interactions/aerodynamics/verif/SIM_VER_DRAG/SET_test`

Test description:
Using the following input file:

`RUN_one_plate_accel_calc_coef_eps00`

This tests the option for the free molecular flow modeling in the full specular case, using the calculate coefficient option, with the parameter ϵ set to 0.0 giving the exact diffuse flow case. A plate is rotated 180 degrees in angle of attack in the streamflow and the accelerations are recorded. An independent model of the free molecular flow for a plate found in Bird [1] (Equations 3.11 and 3.12) was coded in FORTRAN and the benchmark data generated.

Success criteria:
There should be very close agreement between the exact theoretical computation for specular reflection and the approximate simulator value.

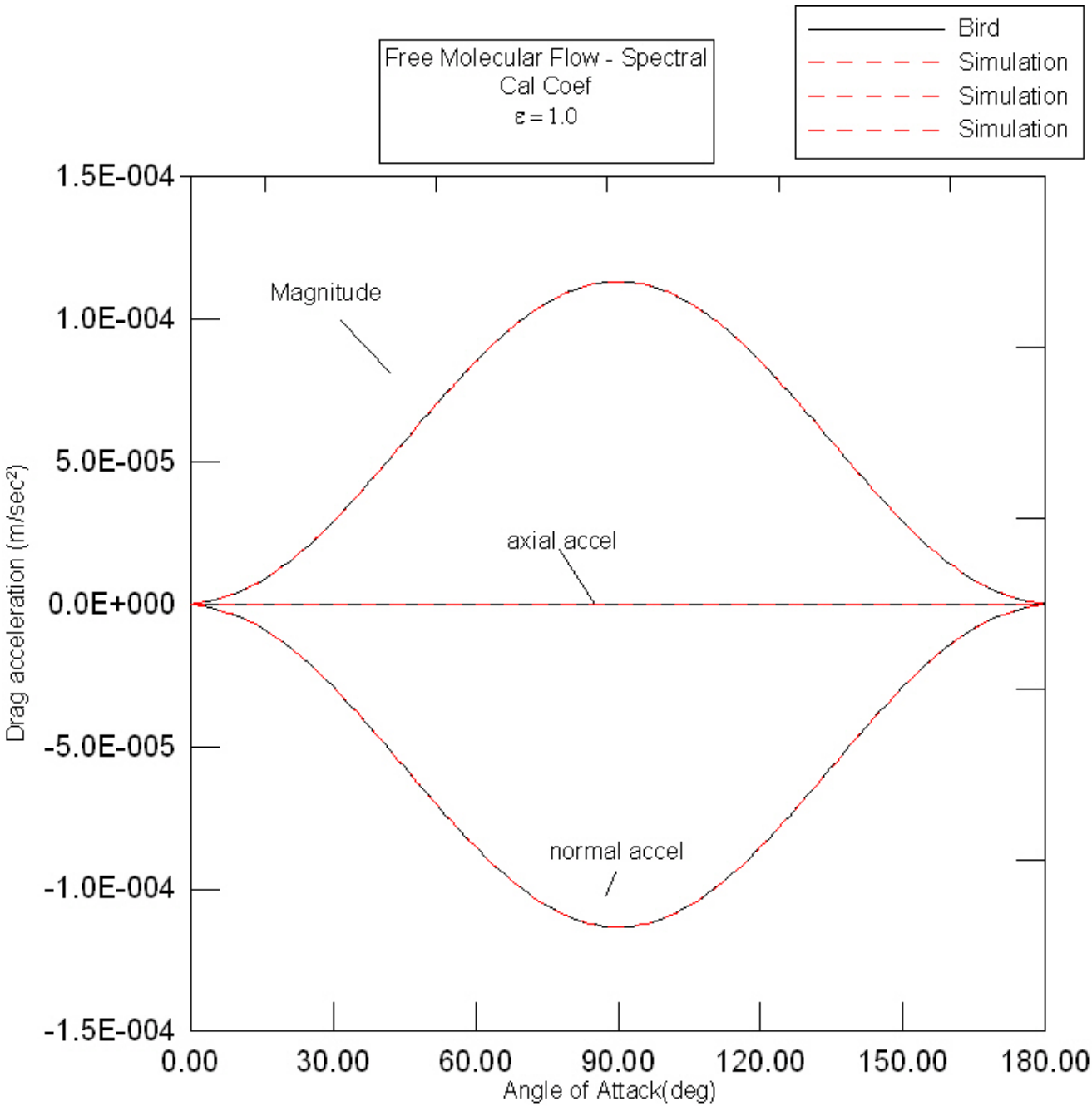

Figure 5.5: Specular Free Molecular Flow

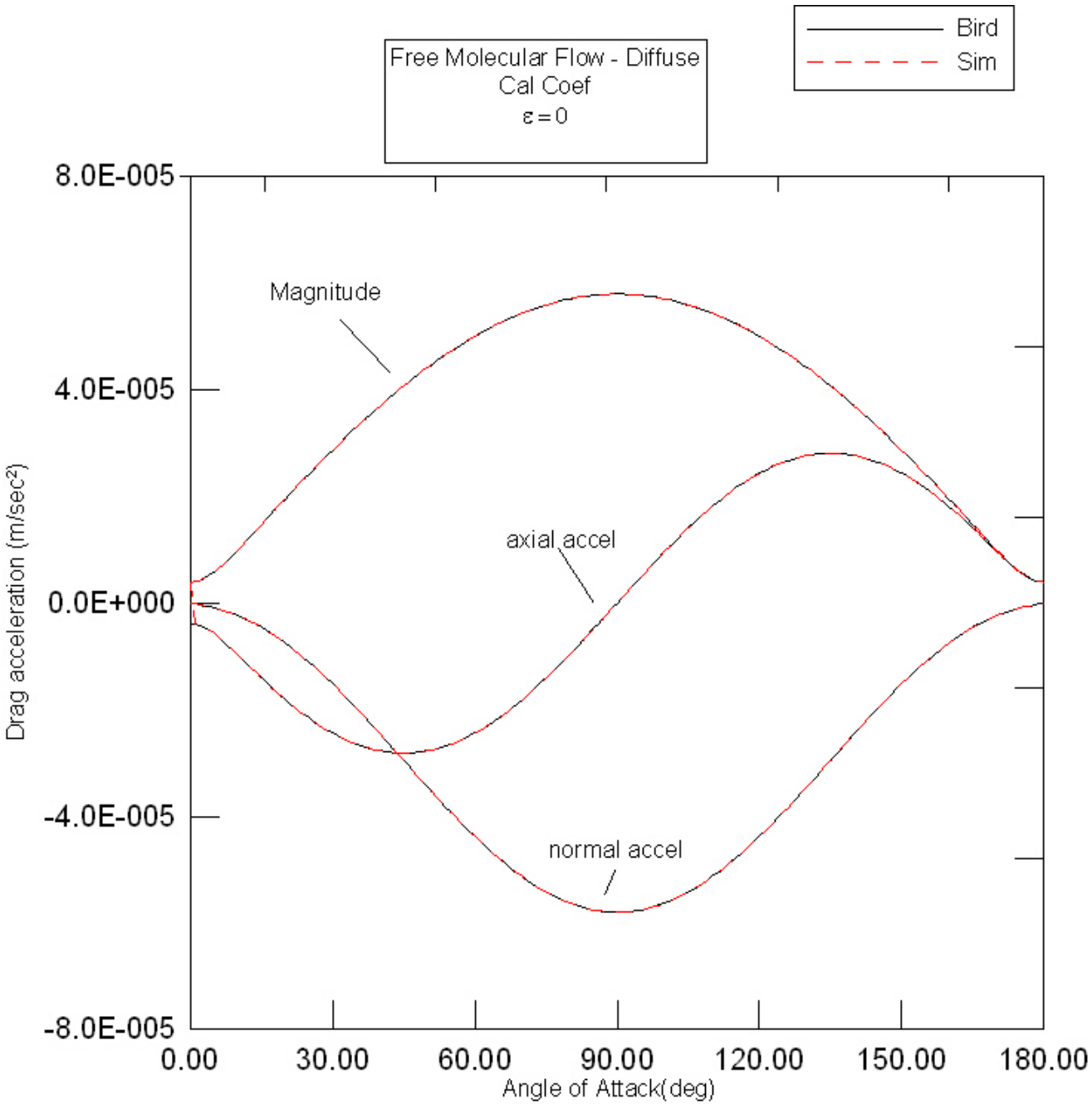

Figure 5.6: Diffuse Free Molecular Flow

Test results:
The test co-plots are shown in the following figure: There is a discernible difference between the simulation and the exact theoretical calculation are shown in figure 5.6 and the differences in figure 5.7. The differences are 10^{-7} or smaller the origin of these differences are unknown but are probably attributable to numerical roundoff. The calculated coefficient diffuse case matches better than specular case.

Test aerodynamics_7: Free Molecular Flow Calculate-Coefficient Mixed Case

Test directory:

`models/interactions/aerodynamics/verif/SIM_VER_DRAG/SET_test`

Test description:
Using the following input file:

`RUN_one_plate_accel_calc_coef_eps05`

This tests the option for the free molecular flow modeling in the full specular case, using the calculate coefficient option, with the parameter ϵ set to 0.5 giving the exact specular flow case. A plate is rotated 180 degrees in angle of attack in the stream flow and the accelerations are recorded. An independent model of the free molecular flow for a plate found in Bird [1] (Equations 3.11 and 3.12) was coded in FORTRAN and the benchmark data generated.

Success criteria:
There should be very close agreement between the exact theoretical computation for specular reflection and the simulator value.

Test results:
The test co-plots are shown in the following figure: There no noticeable difference between the simulation and the exact theoretical calculation as shown in figure 5.8.

Test aerodynamics_8: Free Molecular Flow Calculate Torque on a Plate

Test directory:

`models/interactions/aerodynamics/verif/SIM_VER_DRAG/SET_test`

Test description:
Using the following input file:

`RUN_one_plate_torque`

This tests the option for the free molecular flow modeling in the full specular case, using the calculated coefficient option, with the parameter ϵ set to 0.0 giving the fully diffuse flow case. A plate is rotated 180 degrees in angle of attack in the streamflow and the drag torque recorded. An independent model of the free molecular flow for a plate found in Bird [1]

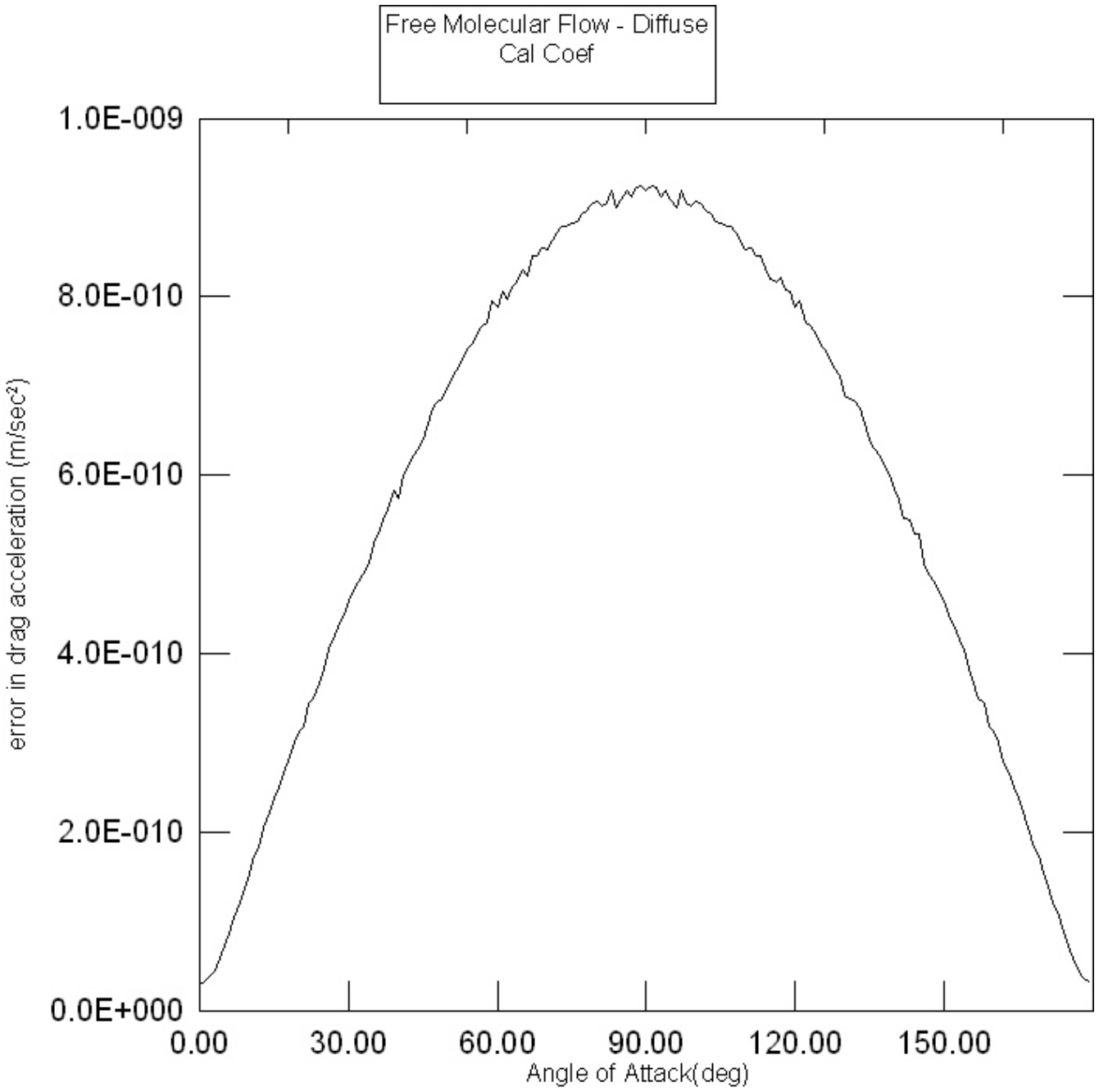

Figure 5.7: Difference Magnitude Diffuse Free Molecular Flow

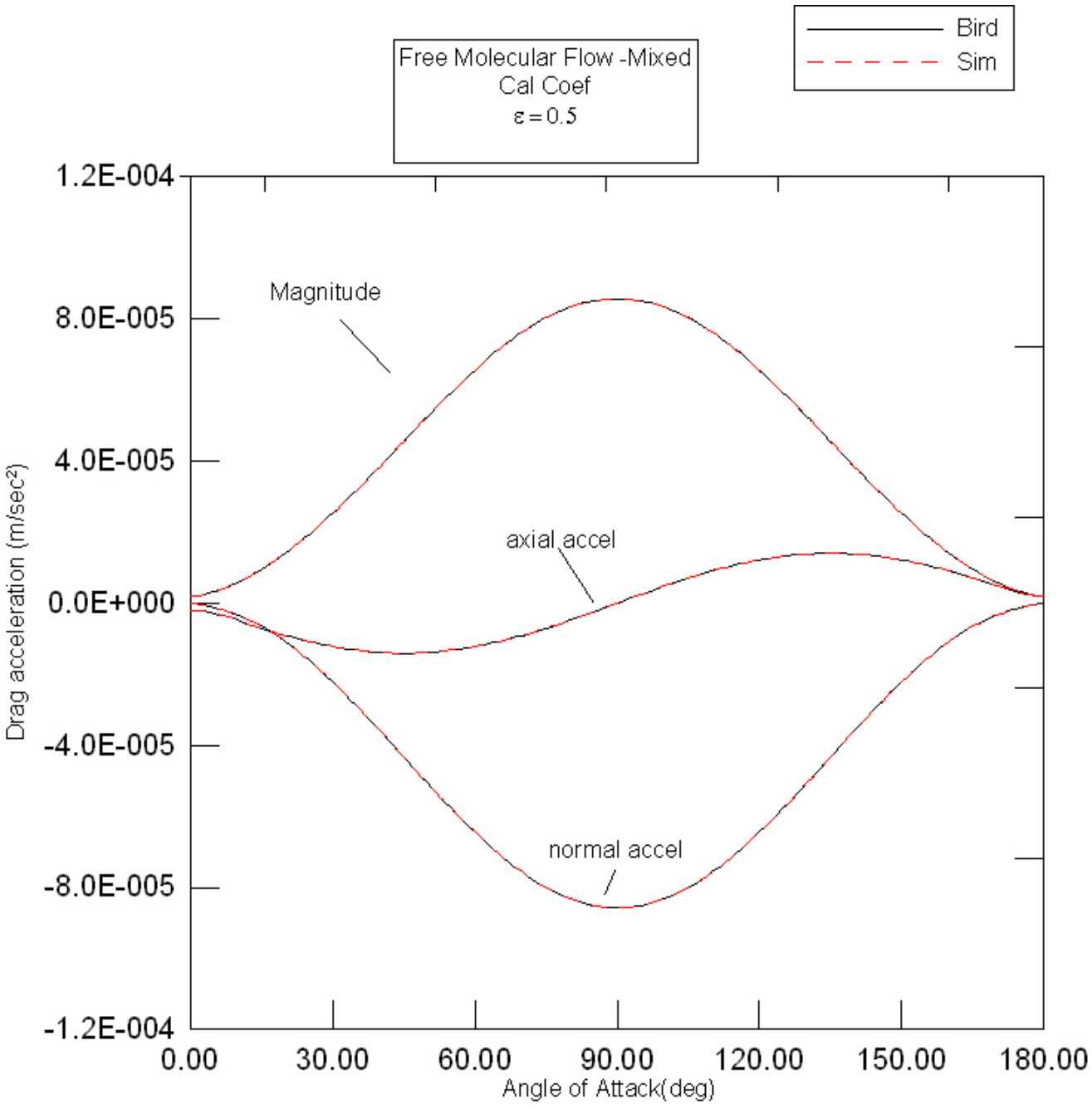

Figure 5.8: Mixed Free Molecular Flow

(Equations 3.11 and 3.12) was coded in FORTRAN and the benchmark data generated by modeling the following equation:

$$T_y = F_n(z_{cp} - z_{cm}) \tag{5.2}$$

where T_y = the torque about y body axis, F_n = normal aerodynamic force to the plate and z_{cp} and z_{cm} are the center of pressure and center of mass respectively.

Success criteria:

There should be very close agreement between the exact theoretical computation for specular reflection and the approximate simulator value.

Test results:

A plot of the difference in the magnitude of the torque in the sim and the exact theoretical value is given in the following figure 5.9 The differences are 10^{-9} or smaller, the nominal torques for the given physical conditions are of the order of 10^{-4} newton meters.

Test aerodynamics_9: Free Molecular Flow Orbiter Case

Test directory:

`models/interactions/aerodynamics/verif/SIM_VER_DRAG/SET_test`

Test description:

`RUN_orbiter`

This tests the option for the free molecular flow modeling in the full diffuse case using the plate model for the aerodynamic surfaces of the Orbiter, with the parameter ϵ set to 0.0 for the full exact diffuse case. In this case both full exact diffusion and the composite plate model of the Orbiter with forces summed is tested. The orbiter plate model dyn_aero_shuttle.d is rotated 180 degrees in angle of attack in the streamflow and the accelerations are recorded.

Success criteria:

This test is a comparison of empirically generated data derived from highly accurate accelerometers flown on the Shuttle Orbiter. Using these data Blanchard [2] derived a least square fit for the normal coefficient of drag

$$C_n = 7.16528 \times 10^{-6}\alpha^3 + 9.66197 \times 10^{-4}\alpha^2 + 9.18422 \times 10^{-3}\alpha + 1.58739 \tag{5.3}$$

and the axial coefficient of drag

$$C_a = -1.17117 \times 10^{-6}\alpha^3 + 5.92205 \times 10^{-4}\alpha^2 + 0.0164864\alpha + 0.751105 \tag{5.4}$$

where α is the angle of attack in degrees. Writing the ratio of the coefficients implies the ratio of the accelerations, namely the ratio of the z body acceleration to the x body acceleration, that is:

$$\frac{C_n}{C_a} = \frac{a_z}{a_x} \tag{5.5}$$

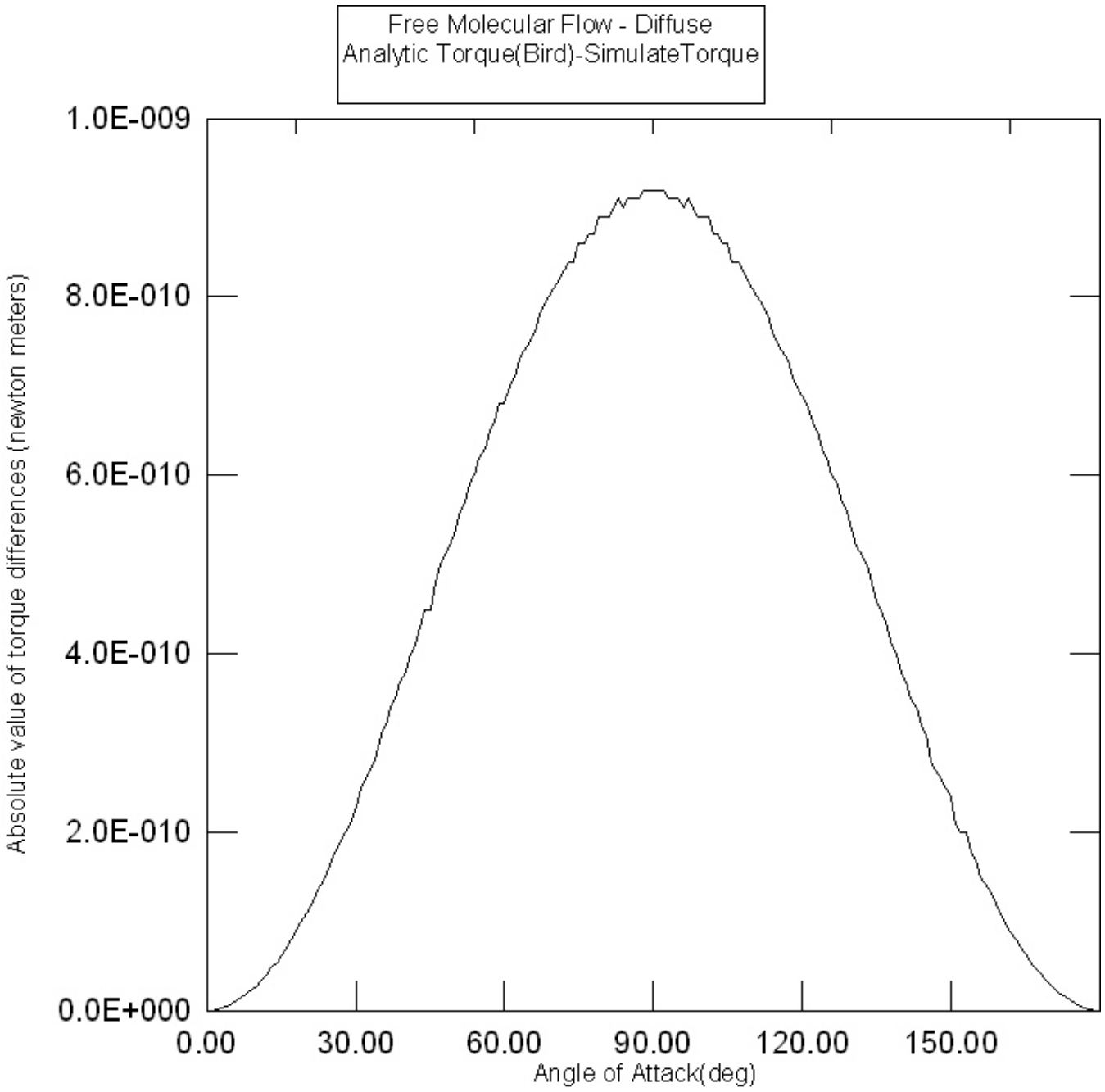

Figure 5.9: Mixed Free Molecular Flow Torque on Plate

The convenience of this expression is that there is no dependence on the local atmospheric density and speed. The empirical fit is only good for an angle of attack between 0 and 60 degrees. Therefore if the simulation is in agreement with the empirical data the modeling with the plate surface model will be considered a valid representation. It is to be noted that for theoretical reasons the space shuttle orbiter can be considered as an object subjected to fully diffuse flow, Blandford [2].

Test results:

The test co-plots are shown in the following figure:

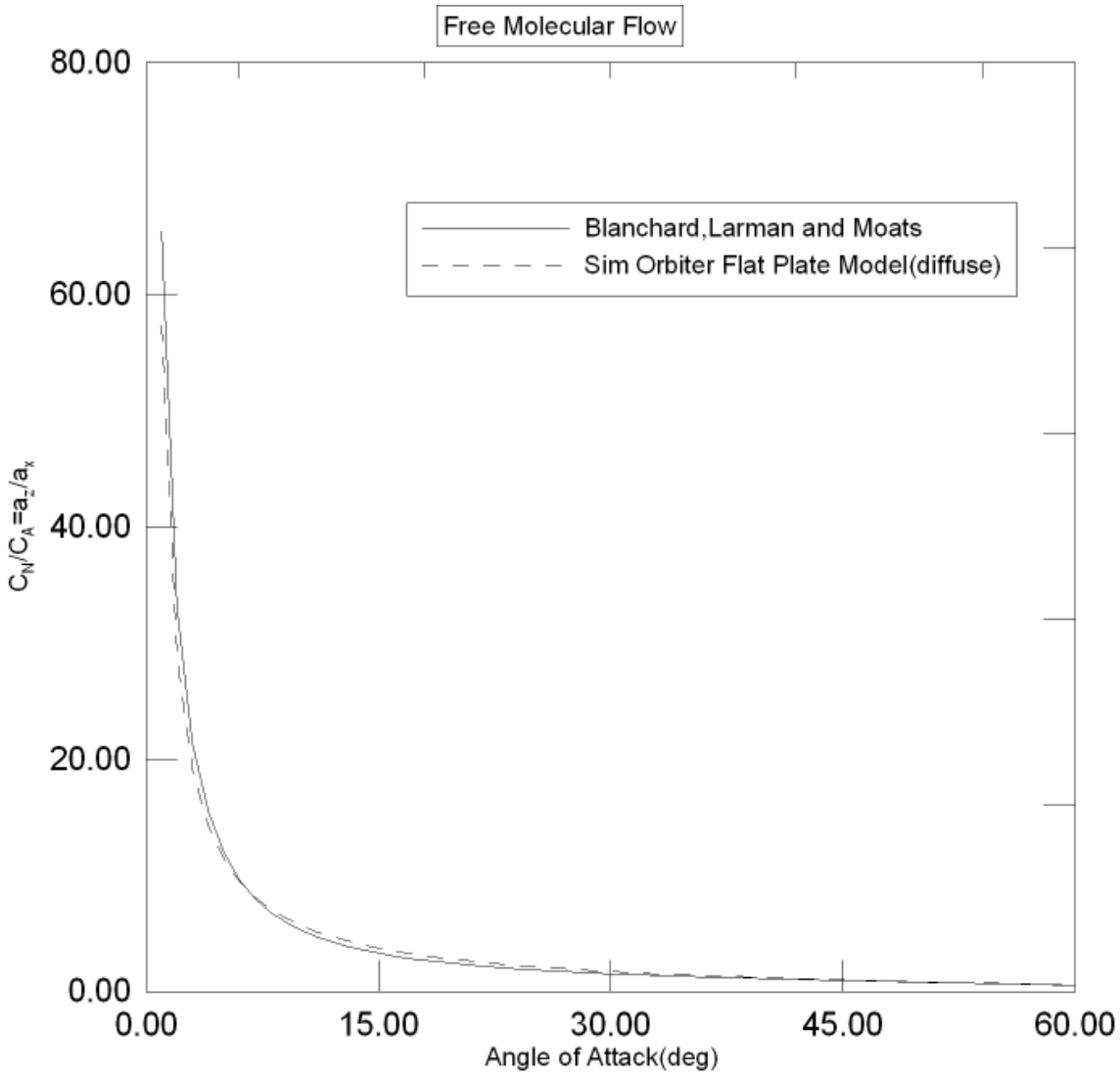

Figure 5.10: Ratio of C_n and C_a

There is a small difference between the simulation and the empirical data shown in figure 5.11. The comparison between the empirical and the simulation ratio differ by an average factor of 0.3. Considering all the approximations and modeling errors that are involved this is a good figure of merit for the flat plate representation of the orbiter. In fact a very good validation comparison using empirical data.

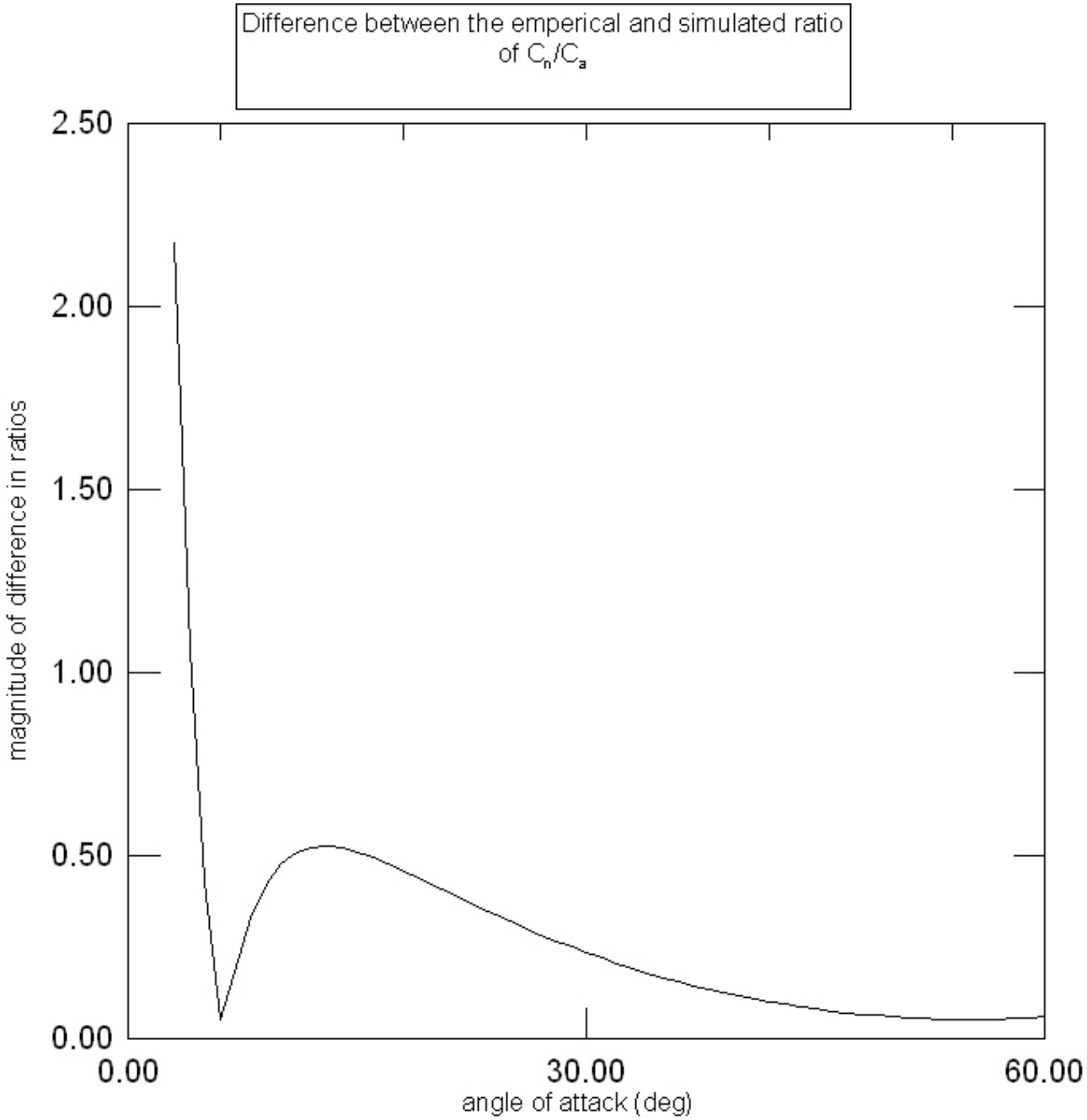

Figure 5.11: Difference simulated and empirical ratios

5.4 Metrics

5.4.1 Code Metrics

Table 5.1 presents coarse metrics on the source files that comprise the model.

Table 5.1: Coarse Metrics

File Name	Number of Lines			
	Blank	Comment	Code	Total
include/aero_drag.hh	44	138	53	235
include/aero_facet.hh	28	79	24	131
include/aero_params.hh	24	71	17	112
include/aero_surface.hh	31	85	32	148
include/aero_surface_factory.hh	28	86	23	137
include/aerodynamics_messages.hh	27	87	19	133
include/class_declarations.hh	16	61	17	94
include/default_aero.hh	30	96	34	160
include/flat_plate_aero_facet.hh	27	123	37	187
include/flat_plate_aero_factory.hh	27	73	26	126
include/flat_plate_aero_params.hh	24	97	26	147
include/flat_plate_thermal_aero_factory.hh	25	72	23	120
src/aero_drag.cc	36	87	97	220
src/aero_facet.cc	19	38	13	70
src/aero_params.cc	19	37	11	67
src/aero_surface.cc	58	71	96	225
src/aero_surface_factory.cc	30	51	43	124
src/aerodynamics_messages.cc	18	29	12	59
src/default_aero.cc	36	55	60	151
src/flat_plate_aero_facet.cc	66	120	193	379
src/flat_plate_aero_factory.cc	41	59	77	177
src/flat_plate_aero_params.cc	16	37	20	73
src/flat_plate_thermal_aero_factory.cc	24	43	31	98

Continued on next page

Table 5.1: Coarse Metrics (continued from previous page)

File Name	Number of Lines			
	Blank	Comment	Code	Total
data/include/aero_model.hh	7	44	10	61
data/src/aero_model.cc	13	12	14	39
Total	714	1751	1008	3473

Table 5.2 presents the extended cyclomatic complexity (ECC) of the methods defined in the model.

Table 5.2: Cyclomatic Complexity

Method	File	Line	ECC
jeod::AerodynamicDrag:: AerodynamicDrag (void)	src/aero_drag.cc	68	1
jeod::AerodynamicDrag::~ AerodynamicDrag (void)	src/aero_drag.cc	86	1
jeod::AerodynamicDrag::aero_ drag (double inertial_ velocity[3], Atmosphere State* atmos_ptr, double T_ inertial_struct[3][3], double mass, double center_grav[3])	src/aero_drag.cc	101	8
jeod::AerodynamicDrag::set_ aero_surface (AeroSurface& to_set)	src/aero_drag.cc	191	1
jeod::AerodynamicDrag:: clear_aero_surface ()	src/aero_drag.cc	202	1
jeod::AeroFacet::AeroFacet (void)	src/aero_facet.cc	44	1
jeod::AeroFacet::~AeroFacet (void)	src/aero_facet.cc	56	1
jeod::AeroParams::Aero Params (void)	src/aero_params.cc	40	1
jeod::AeroParams::~Aero Params (void)	src/aero_params.cc	52	1
jeod::AeroSurface::Aero Surface (void)	src/aero_surface.cc	58	1
jeod::AeroSurface::~Aero Surface (void)	src/aero_surface.cc	72	4
jeod::AeroSurface::allocate_ array (unsigned int size)	src/aero_surface.cc	96	4

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::AeroSurface::allocate_interaction_facet (Facet* facet, InteractionFacet Factory* factory, Facet Params* params, unsigned int index)	src/aero_surface.cc	147	4
jeod::AeroSurfaceFactory::AeroSurfaceFactory (void)	src/aero_surface_factory.cc	48	1
jeod::AeroSurfaceFactory::~AeroSurfaceFactory (void)	src/aero_surface_factory.cc	61	1
jeod::AeroSurfaceFactory::add_facet_params (Facet Params* to_add)	src/aero_surface_factory.cc	78	3
jeod::DefaultAero::DefaultAero (void)	src/default_aero.cc	52	1
jeod::DefaultAero::~DefaultAero (void)	src/default_aero.cc	68	1
jeod::DefaultAero::aerodrag_force (const double, const double rel_vel_hat[3], Aero DragParameters* aero_drag_param_ptr, double mass, double force[3], double torque[3])	src/default_aero.cc	88	5
jeod::FlatPlateAeroFacet::FlatPlateAeroFacet (void)	src/flat_plate_aero_facet.cc	45	1
jeod::FlatPlateAeroFacet::~FlatPlateAeroFacet (void)	src/flat_plate_aero_facet.cc	70	1
jeod::FlatPlateAeroFacet::aerodrag_force (const double rel_vel_mag, const double rel_vel_struct_hat[3], AeroDragParameters* aero_drag_param_ptr, double center_grav[3])	src/flat_plate_aero_facet.cc	86	17
jeod::FlatPlateAeroFactory::FlatPlateAeroFactory (void)	src/flat_plate_aero_factory.cc	58	1

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::FlatPlateAeroFactory::~FlatPlateAeroFactory (void)	src/flat_plate_aero_factory.cc	69	1
jeod::FlatPlateAeroFactory::create_facet (Facet* facet, FacetParams* params)	src/flat_plate_aero_factory.cc	85	3
jeod::FlatPlateAeroFactory::is_correct_factory (Facet* facet)	src/flat_plate_aero_factory.cc	157	2
jeod::FlatPlateAeroParams::FlatPlateAeroParams (void)	src/flat_plate_aero_params.cc	40	1
jeod::FlatPlateAeroParams::~FlatPlateAeroParams (void)	src/flat_plate_aero_params.cc	59	1
jeod::FlatPlateThermalAeroFactory::JEOD_DECLARE_ATTRIBUTES (FlatPlateAeroFacet)	src/flat_plate_thermal_aero_factory.cc	43	1
jeod::FlatPlateThermalAeroFactory::~FlatPlateThermalAeroFactory (void)	src/flat_plate_thermal_aero_factory.cc	63	1
jeod::FlatPlateThermalAeroFactory::is_correct_factory (Facet* facet)	src/flat_plate_thermal_aero_factory.cc	78	2

5.5 Requirements Traceability

Table 5.3: Requirements Traceability

Requirement	Inspection and Testing
aerodynamics_1 - Top-level Requirements	Inspection aerodynamics_1
aerodynamics_4 - Aerodynamic Extensibility	Inspection aerodynamics_5
aerodynamics_2 - Drag Data	Inspection aerodynamics_2 Inspection aerodynamics_3 Inspection aerodynamics_4
aerodynamics_3 - Simple Drag	Inspection aerodynamics_3 Test aerodynamics_1
3.2 - Approximate Free Molecular	Inspection aerodynamics_2

Table 5.3: Requirements Traceability (continued)

Requirement	**Inspection and Testing**
Flow	Test aerodynamics_2
3.2 - Approximate Free Molecular Flow	Inspection aerodynamics_2 Test aerodynamics_3
3.2 - Approximate Free Molecular Flow	Inspection aerodynamics_2 Test aerodynamics_4
3.3 - Free Molecular Flow	Inspection aerodynamics_2 Test aerodynamics_5
3.2 - Approximate Free Molecular Flow	Inspection aerodynamics_2 Test aerodynamics_6
3.2 - Free Molecular Flow	Inspection aerodynamics_2 Test aerodynamics_7
3.4 - Aerodynamic Forces and Torques	Inspection aerodynamics_2 Test aerodynamics_2 Test aerodynamics_3 Test aerodynamics_4 Test aerodynamics_5 Test aerodynamics_6 Test aerodynamics_7 Test aerodynamics_8 Test aerodynamics_9

Bibliography

[1] G.A. Bird. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Clarendon Press, Oxford,England, 1994.

[2] Blanchard. Rarefied-flow shuttle aerodynamics flight model. Technical Report NACA-TM-107698, Langley Research Center, Moffett Field, Calif., 1993.

[3] Generated by doxygen. *Aerodynamics Reference Manual*. National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, July 2022.

[4] Hammen, D. Dynamic Body Model. Technical Report JSC-61777-dynamics/dyn_body, NASA, Johnson Space Center, Houston, Texas, July 2022.

[5] Jackson, A., Thebeau, C. JSC Engineering Orbital Dynamics. Technical Report JSC-61777-docs, NASA, Johnson Space Center, Houston, Texas, July 2022.

[6] NASA. NASA Software Engineering Requirements. Technical Report NPR-7150.2, NASA, NASA Headquarters, Washington, D.C., September 2004.

[7] S.A. Schaaf and P.L. Chambre. *Flow of rarefied gases High Speed Aerodynamics and Jet Propulsion*. Princeton University Press, Princeton N.J., 1961.

[8] Shelton, R. Message Handler Model. Technical Report JSC-61777-utils/message, NASA, Johnson Space Center, Houston, Texas, July 2022.

[9] Spencer, A. Surface Model. Technical Report JSC-61777-utils/surface_model, NASA, Johnson Space Center, Houston, Texas, July 2022.

[10] Spencer, A. Atmosphere Model. Technical Report JSC-61777-environment/atmosphere, NASA, Johnson Space Center, Houston, Texas, July 2022.

[11] J.R. Stalder and V.J. Zurick. Theoretical aerodynamic characteristics of bodies in a free-molecule-flow field. Technical Report NACA-TN-2423, Ames Aeronautical Laboratory, Moffett Field, Calif., 1951.

[12] Turner, G. Radiation Pressure Model. Technical Report JSC-61777-interactions/radiation_pressure, NASA, Johnson Space Center, Houston, Texas, July 2022.

[13] Vallado, D.A., with tech contributions by McClain, W.D. *Fundamentals of Astrodynamics and Applications, Second Edition*. Microcosm Press, El Segundo,Calif., 2001.

[14] Vetter, K. *The Trick User's Guide*. NASA, Johnson Space Center, Software, Robotics, and Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058.