JSC-61777-models/utils/orbital_elements

JSC Engineering Orbital Dynamics Orbital Elements Model

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

Package Release JEOD v5.1

Document Revision 1.0
July 2023

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

JSC-61777-models/utils/orbital_elements

JSC Engineering Orbital Dynamics
Orbital Elements Model

Document Revision 1.0
July 2023

Dr. Edwin Z. Crues

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 orbital elements class provides a representation of the translational orbital state of an orbital body in standard Keplerian orbital elements. In addition, this class provides methods for converting between the Keplerian orbital elements and planet centered inertial Cartesian position and velocity vectors. The methods in this class are primarily based on the methods presented in Vallado [3] although some modifications have been made to increase the numerical accuracy of the routines in limiting cases.

Contents

Chapter 1

Introduction

1.1 Model Description

The orbital elements class provides a representation of the translational orbital state of an orbital body in standard Keplerian orbital elements. In addition, this class provides methods for converting between the Keplerian orbital elements and planet centered inertial Cartesian position and velocity vectors. The methods in this class are primarily based on the methods presented in Vallado [3] although some modifications have been made to increase the numerical accuracy of the routines in limiting cases.

In the object design sense, the OrbitalElement class is almost completely independent from the remainder of JEOD. The OrbitalElement class is used to initialize DynBodyInitOrbit, to construct OrbElemDerivedState and to perform initializations in the verification sims of the DerivedState class.

This documentation describes the design of the OrbitalElement class in detail. The orbital element class borrows heavily from published work on the design of algorithms for converting between Keplerian orbital elements and Cartesian position and velocity vectors.

This documentation covers a complete guide for the use of the orbital element class. It presents the mathematical equations and algorithms used to create the orbital element class source code, defines requirements on the code, reports the results of test cases run the verify and validate the model, and contains a guide for the use of the orbital element class in a Trick simulation environment.

1.2 Document History

Author	Date	Revision	Description
Scott Piggott	1/6/2006	1.0	Initial Version
Edwin Z. Crues	2/3/2009	2.0 Alpha	JEOD 2.0 Conversion
Robert O. Shelton	12/8/2009	2.1	JEOD 2.0 Release
Robert O. Shelton	July 2023	2.2	Added metrics

1.3 Document Organization

This document is formatted in accordance with the NASA Software Engineering Requirements Standard [2] 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 Orbital Elements 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. The final section contains a description of how the document is organized.

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

Chapter 3: Product Specification - Describes the underlying theory, architecture, and design of the Orbital Elements 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 Orbital Elements Model in a Trick simulation.

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

Chapter 2

Product Requirements

This model shall meet the JEOD project requirements specified in the JEOD top-level document.

Requirement OE_1: Cartesian to Element Conversion

Requirement:
The OrbitalElement class shall be capable of taking in an arbitrary set of Cartesian position and velocity vectors and outputting the correct set of Keplerian orbital elements.

Rationale:
This is one of the main two functions of the OrbitalElement class.

Verification:
Test

Requirement OE_2: Element to Cartesian Conversion

Requirement:
The OrbitalElement class shall be capable of taking in an arbitrary set of Keplerian orbital elements and outputting the correct set of Cartesian position and velocity vectors.

Rationale:
This is one of the main two functions of the OrbitalElement class.

Verification:
Test

Chapter 3

Product Specification

3.1 Conceptual Design

3.1.1 Purpose and Scope

The OrbitalElement class is a class whose methods convert between Keplerian orbital elements and Cartesian position/velocity representations of the translational state. This module is not a stand-alone program; rather, it is designed to be incorporated into the Trick Simulation Environment, and applies to space vehicles or objects whose states need to be converted between Cartesian vectors and Keplerian orbital elements.

3.1.2 Goals and Objectives

The goal of the OrbitalElement class is to be able to convert between any arbitrary set of orbital elements and Cartesian position and velocity vectors with a high degree of accuracy and reliability. That is the primary function of this class. A secondary function is for the OrbitalElement class to output error messages if for some reason it is unable to fulfill its primary goal.

3.2 Mathematical Formulations

Conversion between Keplerian orbital elements and Cartesian position and velocity is a well understood problem that has been solved many times. There are several methods that can be used with no clear distinction between them in terms of computational loading and algorithm reliability. A classic technique used in many engineering applications is described in [3]. The algorithms presented in [3] were initially implemented in Trick and now constitute the OrbitalElement class. A quick description of these algorithms is presented here but for a more complete treatment, please see [3].

A satellite's translational state can be completely defined by using either position and velocity information or classical Keplerian orbital elements. It is advantageous to be able to use Keplerian elements for some applications and position and velocity vectors for other applications. Therefore, it

is important that the mission designer be able to convert between the two types of state description quickly. The simpler of the two conversions involves determining the spacecraft position and velocity from a set of classical orbital elements.

3.2.1 Nomenclature

The mathematical formulations in this chapter follow a common orbital element nomenclature:

p	The object's semi-parameter
ν	The object's true anomaly
μ	The gravitational body's gravitational constant
Ω	Right Ascension of the Ascending Node
ω	The argument of periapsis
i	The orbital inclination of the body
λ_{true}	The true longitude of the body
$\tilde{\omega}_{true}$	The true argument of periapsis
$\vec{e}$	The eccentricity vector
$\vec{r}$	The Cartesian position vector
$\vec{v}$	The Cartesian velocity vector
$\vec{h}$	The angular momentum vector
$\vec{n}$	The vector for the line of nodes
I, J, K	The x, y, and z unit vector directions

3.2.2 Position and Velocity to Orbital Elements

If a user has access to a full set of Keplerian orbital elements, the conversion from orbital element description to position and velocity description is a simple process. First, the position and velocity in the perifocal coordinate system are determined and then they are rotated into the inertial reference frame of interest (frequently J2000).

The position and velocity of the system in perifocal coordinates are functions of the orbit's semi-parameter p, the orbital eccentricity e, the true anomaly of the orbit ν, and the gravitational parameter of the gravitational body μ. These parameters are used because a formulation for each is defined regardless of the type of orbit (circular, eccentric, inclined, parabolic, etc.). The position values of the spacecraft in the perifocal coordinate system can be calculated as shown in Equation 3.1.

$$\vec{r}_{PQW} = \begin{bmatrix} \frac{p\cos(\nu)}{1+e\cos(\nu)} \\ \frac{p\sin(\nu)}{1+e\cos(\nu)} \\ 0 \end{bmatrix} \tag{3.1}$$

The velocity of the spacecraft in the perifocal coordinate system is then given by Equation 3.2.

$$\vec{v}_{PQW} = \begin{bmatrix} \sqrt{\frac{\mu}{p}}\sin(\nu) \\ \sqrt{\frac{\mu}{p}}(e+\cos(\nu) \\ 0 \end{bmatrix} \tag{3.2}$$

In the above equations, the parameter e is the orbital eccentricity and is defined as the magnitude of the eccentricity vector which is given in section 2.4.1 of [3]. The parameter ν is the true anomaly of the spacecraft which is defined as the angle between the line from the primary focus of the orbit to periapsis and the line from primary focus to spacecraft. This parameter is not always defined but a formulation can be used for these special cases. The parameter p is the semi-parameter of the orbit and is defined as the width of the orbit at the primary focus. The parameter μ is the gravitational parameter of the primary body and is the gravitational constant multiplied by the body's mass.

With the perifocal position and velocity vectors defined, the spacecraft need only be rotated into the inertial system of interest. This is accomplished through use of a rotation matrix that is a function of the orbital inclination i, the right ascension of the ascending node (RAAN) Ω, and the argument of periapsis ω. The values for the elements of this rotation matrix are given in Equation 3.3

$$\left[\frac{IJK}{PQW}\right] = \begin{bmatrix} \cos(\Omega)\cos(\omega) - \sin(\Omega)\sin(\omega)\cos(i) & -\cos(\Omega)\sin(\omega) - \sin(\Omega)\cos(\omega)\cos(i) & \sin(\Omega)\sin(i) \\ \sin(\Omega)\cos(\omega) + \cos(\Omega)\sin(\omega)\cos(i) & -\sin(\Omega)\sin(\omega) + \cos(\Omega)\cos(\omega)\cos(i) & \cos(\Omega)\sin(i) \\ \sin(\omega)\sin(i) & \cos(\omega)\sin(i) & \cos(i) \end{bmatrix} \tag{3.3}$$

Now for certain cases, these parameters are not always defined. For example, with circular orbits, the argument of periapsis and true anomaly are not defined because there is no periapsis point to provide reference. There are a set of rules that deal with these special cases and they are listed here.

- If orbit is circular and equatorial
  - $\omega = 0.0$
  - $\Omega = 0.0$
  - ν=λ_{true}
- If orbit is circular and inclined
  - $\omega = 0.0$
  - ν=u
- If orbit is elliptical and equatorial
  - $\Omega = 0.0$
  - $\omega = \tilde{\omega}_{true}$

In the equations given above, some new parameters are introduced. The parameter λ_{true} is the true longitude and is defined as the angle between the coordinate frame x-axis and the spacecraft measured eastward from the axis (towards y). The parameter u is the argument of latitude and is angle between the ascending node and the spacecraft's position. And lastly, the parameter $\tilde{\omega}_{true}$ is the true longitude of periapsis and is defined as the angle between the eccentricity vector and the x-axis.

With these equations and definitions, it is now possible to compute the spacecraft state for any orbit. These equations describe the algorithms used in the OrbitalElement class and can be found in section 2.6 of [3].

3.2.3 Orbital Elements from Position and Velocity

The orbital elements of the spacecraft can be calculated using the gravitational parameter of the primary body and the Cartesian position and velocity vectors. The algorithms for this conversion are relatively simple and also very robust. Similarly to how the equations from [3] were used for the position and velocity computation, there are algorithms in the text for conversion from position and velocity to orbital elements. These algorithms are used in the code and are described briefly here. For a more complete treatment, please see [3].

First, several classic parameters must be calculated. The spacecraft's angular momentum is $\vec{h}$ and is defined as the cross product of the spacecraft's position and velocity. The parameter h is the magnitude of this vector. The spacecraft's line of nodes is denoted as $\vec{n}$ and it is the cross product of the z-axis and the spacecraft's angular momentum. The specific energy of the system is ξ and is defined as the potential energy of the spacecraft subtracted from the kinetic energy. The eccentricity vector is a complex relation between position and velocity described by Equation 3.4.

$$\vec{e} = \frac{(v^2 - \frac{\mu}{r})\vec{r} - (\vec{r} \cdot \vec{v})\vec{v}}{\mu} \tag{3.4}$$

The eccentricity e is then just the magnitude of the vector value given in Equation 3.4. Using these relations, four of the orbital elements discussed above can be computed from the relations shown in Equation 3.5.

$$\begin{aligned}
\cos(i) &= \frac{h_K}{|\vec{h}|} \\
\cos(\Omega) &= \frac{n_I}{|\vec{n}|} \quad IF(n_j < 0) THEN \Omega = 360^o - \Omega \\
\cos(\omega) &= \frac{\vec{n} \cdot \vec{e}}{|\vec{n}||\vec{e}|} \quad IF(e_K < 0) THEN \omega = 360^o - \omega \\
\cos(\nu) &= \frac{\vec{e} \cdot \vec{r}}{|\vec{e}||\vec{r}|} \quad IF(\vec{r} \cdot \vec{v} < 0) THEN \nu = 360^o - \nu
\end{aligned} \tag{3.5}$$

In the above equations, the subscripts indicate the direction of the vector. For example, the value n_K is the z-component of the $\vec{n}$ vector.

The semi-major axis and semi-parameter have two different calculation methods for use when the orbit is parabolic and when it is not. These calculations are show in Equation 3.6.

$$\begin{aligned}
&IF\ e \neq 1.0\ THEN \\
&\quad a = \frac{-\mu}{2\xi} \\
&\quad p = a(1 - e^2) \\
&ELSE \\
&\quad p = \frac{h^2}{2} \\
&\quad a = \infty
\end{aligned} \tag{3.6}$$

Once again, there are special cases for when the orbit is circular or equatorial. These special rules

are given in Equation 3.7.

$$
\begin{aligned}
&Elliptical Equatorial \\
&\quad \cos(\tilde{\omega}_{true}) = \frac{e_I}{|\vec{e}|} \quad IF(e_J < 0) THEN\ \tilde{\omega}_{true} = 360^o - \tilde{\omega}_{true} \\
&Circular Inclined \\
&\quad \cos(u) = \frac{\vec{n} \cdot \vec{r}}{|\vec{n}||\vec{r}|} \quad IF(r_K < 0) THEN\ u = 360^o - u \\
&Circular Equatorial \\
&\quad \cos(\lambda_{true}) = \frac{r_I}{|\vec{r}|} \quad IF(r_J < 0) THEN \lambda_{true} = 360^o - \lambda_{true}
\end{aligned}
\tag{3.7}
$$

These parameters function as a substitute for the argument of periapsis or the true anomaly depending on the case.

With these equations, it is now possible to convert over from the Cartesian position and velocity to the Keplerian orbital elements. Once again, this is a very brief description of the complete algorithms which can be found in section 2.5 of [3].

3.2.4 Solving Kepler's Problem

Frequently, a mission designer needs to calculate the complete set of orbital elements from a few of the available orbital elements. This often entails finding the solution to Kepler's equation. The equations used in the code convert between mean anomaly and the anomaly for the orbit. The mean anomaly is used such that if the orbit was circular, that is the anomaly it would have. The orbital anomaly is then the angle formed between the line from the focus to the periapsis and the position vector in the orbit. So, for a circular orbit, the orbital anomaly is always equal to the mean anomaly. There is a version of Kepler's equation for each orbit type (elliptical, parabolic, and hyperbolic).

So with these three equations, it is possible to determine all of the orbital elements from a smaller set. These equations represent a minor part of the overall software model so they have not been included in this document. However, if an algorithm check is desired, the algorithms from which the OrbitalElement class have been copied can be located in section 2.2 of [3]

3.3 Detailed Design

The Orbital Elements Model uses a single class to accomplish its purpose. This section describes the OrbitalElement class.

3.3.1 Class Description

The *orbital_elements.h* file contains the class used by the Orbital Elements Model source code. A detailed description of the OrbitalElement class follows.

Fields

The public data fields of the OrbitalElement class are as follows:

- Orbit Definition Parameters
  - semi_major_axis: M Semi-major-axis (a)
  - semiparam: M Semiparameter (p)
  - e_mag: – Magnitude of eccentricity (e)
  - inclination: r Orbit inclination (i)
  - arg_periapsis: r Argument of periapsis (ω)
  - long_asc_node: r Longitude of ascending node (Ω)
- Orbital Position Parameters
  - r_mag: M Magnitude of orbital radius
  - vel_mag: M/s Magnitude of orbital velocity
  - true_anom: r True Anomaly (ν)
  - mean_anom: r Mean Anomaly (M)
  - mean_motion: r/s Mean motion of orbit (n)
  - orbital_anom: r Eccentric (E), Hyperbolic (H), or Parabolic (B) anomaly
- Working Parameters
  - sin_v: – Sine of the true anomaly
  - cos_v: – Cosine of the true anomaly
  - orb_energy: M^2/s^2 Specific orbital energy
  - orb_ang_momentum: M^2/s Specific orbital angular momentum

Methods

In addition to the default constructor and the destructor, the OrbitalElements class also contains the following member functions.

set_object_name This member function sets the name of the orbiting object.

- return: void - no return
- IN: const char * name - Orbital object name

get_object_name This member function returns the name of the orbital object.

- return: const char* The name of the orbital object
- IN: void – no inputs

set_planet_name This member function sets the name of the planet.

- return: void - no return
- IN: const char * name - Planet name

get_planet_name This member function returns the name of the planet.

- return: const char* The name of the planet
- IN: void – no inputs

from_cartesian This function initializes fields of the OrbitalElement class which correspond to Keplerian orbital elements based on a position and velocity in planet-centered inertial coordinates and the gravitational parameter of the primary

- return: int – always returns 0
- IN: double μ the gravitational parameter of the planet (product of planetary mass and universal gravitational constant)
- IN: const double pos[3] The position of the orbital object in planet-centered inertial coordinates
- IN: const double vel[3] The velocity of the orbital object in planet-centered inersial coordinates

to_cartesian

- RETURN: int – always returns 0
- IN: μ The gravitational parameter of the planet
- OUT: double pos[3] The position of the orbital object corresponding to the Keplerian orbital elements of this OrbitalElements object
- OUT: double vel[3] The velocity of the orbital object corresponding to the Keplerian orbital elements of this OrbitalElementsObject

nu_to_anomalies This method calculates and sets the Mean Anomaly and the Eccentric, Hyperbolic, or Parabolic Anomalies, given the True Anomaly and eccentricity of this OrbitalElements object

- RETURN: int – always returns 0
- PARAMETERS: none

mean_anom_to_nu This method calculates and sets the true anomaly from the mean anomaly and eccentricity of this OrbitalElements object.

- RETURN: int – always returns 0
- PARAMETERS: none

3.4 Inventory

All Orbital Elements Model files are located in the directory $JEOD_HOME/models/utils/orbital_elements. Tables **??** to **??** list the configuration-managed files in this directory.

Chapter 4

User Guide

Although it is possible to run the OrbitalElement class from a Trick S_define file, this is not a common usage in most simulations. Rather, the OrbitalElement class is typically employed as a conversion utility as in the definition of the OrbElemDerivedState class. The verification cases for the OrbitalElement class are run from an S_define file using the test class OrbElemVer to store and log data as well as to exercise the to_cartesian, from_cartesian and mean_anom_to_nu conversion methods of the OrbitalElement class.

The following are lines from an S_define file which declare and initialize an OrbitalElement class.

```
/* The only necessary structure for this test is ORBITAL ELEMENTS */
utils/orbital: OrbitalElements orb_elem
   (orb_elem_ver/data/orb_elem_init.d);
```

The test class OrbElemVer contains a single method which exercises the methods of the OrbitalElement class. It's data consist of public fields for inertial position, velocity, gravitational parameter and a flag to determine which conversion is to be performed.

```
Flag   to_cartesian; // --    Flag to determine which direction.
double mu;           // M3/s2 Gravity constant for routine.
double position[3];  // M     Pos vector for elem conversion.
double velocity[3];  // M/s   Vel vector for elem conversion.
```

The to_cartesian variable determines if the conversion is going to Cartesian from Keplerian elements or Keplerian elements to Cartesian. If the to_cartesian variable is set to True, the OrbitalElements::to_cartesian method is used:

```
int OrbitalElements::to_cartesian( // RETURN: --  Return always zero.
   double mu,       // IN:  M3/s2  Gravitational parameter of planet.
   double pos[3],   // OUT: M      Orbital position vector of vehicle.
   double vel[3] ); // OUT: M/s    Orbital velocity vector of vehicle.
```

If the to_cartesian variable is set to False, the OrbitalElements::from_cartesian method is used:

```
int OrbitalElements::from_cartesian( // RETURN: --  Return always zero.
         double mu,       // IN: M3/s2   Gravitational parameter of planet.
   const double pos[3],   // IN: M       Position vector of vehicle.
   const double vel[3] ); // IN: M/s     Velocity vector of vehicle.
```

Chapter 5

Verification and Validation

5.1 Inspection and Verification

Inspection OE_1: Data conversion

The OrbitalElement class contains the fields and methods needed to completely satisfy OE_2 and OE_1.

5.2 Validation

In this set of validation tests, 56 test cases were run in order to determine that the system was performing nominally. For each test bullet below, several of those test cases are included as part of the testing description. The code and data files associated with these testing bullets can be found in the verif directory inside of the orbital/dynamics directory. Inside of the verif directory, there is the SIM_orb_elem/SET_test sub-directory which contains all of the various testing directories. They are all labeled with the convention:

RUN_T#_OE_VER

In this convention, the # ranges from 01-57 (There is one unused testing directory). The run numbers associated with a particular test are called out in the test bullet.

Test OE_1: Circular Equatorial Orbits

Purpose:

The purpose of this test is to examine the performance of the orbital element class when dealing with circular equatorial orbits.

Requirements:

By passing this test, the OrbitalElement class partially satisfies requirement OE_2 and partially satisfies requirement OE_1.

Procedure:

This test is designed to test the conversion from Cartesian position and velocity to orbital elements. It tests the model's capabilities with respect to circular, equatorial orbits. These orbits are used to verify that the code can correctly convert circular equatorial values and because the determination of the correct results by hand is a trivial matter.

Results:

Figure 5.1 shows the results computed for four different circular orbits. They all had the same radius of 7,378,145 m and velocity magnitude of 7,350.1 m/s. The only difference was the direction of the position and velocity vectors.

	Test inputs (x, y, z)		Selected Orbital Elements					
			a (m)	e ()	i (deg)	Omega (deg)	true anomaly (deg)	orbital anomaly (deg)
Test 1	position (m)	7378145	7378145.0	3.569E-17	0.0	0.0	0.0	0.0
		0.0						
		0.0						
	velocity (m/s)	0.0						
		7350.142						
		0.0						
Test 2	position (m)	0.0	7378145.0	3.569E-17	0.0	0.0	90.0	90.0
		7378145						
		0.0						
	velocity (m/s)	-7350.14						
		0.0						
		0.0						
Test 3	position (m)	-7378145	7378145.0	3.569E-17	0.0	0.0	180.0	180.0
		0.0						
		0.0						
	velocity (m/s)	0.0						
		-7350.14						
		0.0						
Test 4	position (m)	0	7378145.0	3.569E-17	0.0	0.0	270.0	270.0
		-7378145						
		0.0						
	velocity (m/s)	7350.142						
		0.0						

Figure 5.1: Table of Testing Results for Circular Equatorial Orbits

This table shows nominal results for all sub-tests. The presence of non-zero values for the true anomaly would normally indicate problems because for circular orbits, the true anomaly is undefined. However, in the model, the true anomaly variable is set to the value for the true longitude so that the element to Cartesian computation can be used interchangeably.

One parameter not recorded in the above table but useful for the orbital element class is the spacecraft mean motion. This parameter is the average angular velocity magnitude of the spacecraft over one orbit. Because the angular velocity magnitude of the spacecraft remains constant for the duration of the orbit, the instantaneous angular velocity is the same as the mean motion. The formula for the instantaneous angular velocity of the spacecraft is given by Equation 5.1.

$$\omega_s c = \frac{\vec{r}\, x\, \vec{v}}{|\vec{r}|^2} \tag{5.1}$$

According to this equation as well as the position and velocity magnitudes, the expected mean motion should be 9.9620E-4 rad/s. The value computed by the equations of motion matches

this value to 13 digits of precision.

Run directories:

01-04

Test OE_2: Circular Polar Orbits

Purpose:

The purpose of this test is to examine the performance of the orbital element class when dealing with circular polar orbits.

Requirements:

By passing this test, the OrbitalElement class partially satisfies requirement OE_2 and partially satisfies requirement OE_1.

Procedure:

This test is designed to test the conversion from Cartesian position and velocity to orbital elements. It tests the model's capabilities with respect to circular, polar orbits. These orbits are used to verify that the code can correctly convert circular polar values and because the determination of the correct results by hand is a trivial matter.

Results:

Figure 5.2 shows the results computed for four different circular polar orbits. They all had the same radius of 7,378,145 m and velocity magnitude of approximately 7,350.1 m/s. The only difference was the direction of the position and velocity vectors.

	Test inputs (x, y, z)		Selected Orbital Elements					
			a (m)	e ()	i (deg)	Omega (deg)	true anomaly (deg)	orbital anomaly (deg)
Test 1	position (m)	7378145	7378145.0	3.569E-17	90.0	0.0	0.0	0.0
		0.0						
		0.0						
	velocity (m/s)	0.0						
		0.0						
		7350.1416						
Test 2	position (m)	0.0	7378145.0	3.569E-17	90.0	0.0	90.0	90.0
		0.0						
		7378145						
	velocity (m/s)	-7350.1416						
		0.0						
		0.0						
Test 3	position (m)	-7378145	7378145.0	3.569E-17	90.0	0.0	180.0	180.0
		0.0						
		0.0						
	velocity (m/s)	0.0						
		0.0						
		-7350.1416						
Test 4	position (m)	0	7378145.0	3.569E-17	90.0	0.0	270.0	270.0
		0.0						
		-7378145						
	velocity (m/s)	7350.1416						
		0						
		0.0						

Figure 5.2: Table of Testing Results for Circular Polar Orbits

Figure 5.2 shows nominal results for all four of the sub-tests run. Once again, there appears to be incorrect results when it outputs non-zero values for the true anomaly but this is once again the case of another variable being stored in the true anomaly portion of the structure. In this case, the stored parameter is argument of latitude u. All of the other parameters are the expected values. Because the same radius and velocity magnitude was used, the same mean motion of 9.962E-4 rad/s was observed with the same accuracy.

Run directories:

05-08

Test OE_3: Circular Inclined Orbits

Purpose:

The purpose of this test is to examine the performance of the orbital element class when dealing with circular inclined orbits.

Requirements:

By passing this test, the OrbitalElement class partially satisfies requirement OE_2 and partially satisfies requirement OE_1.

Procedure:

This test is designed to test the conversion from Cartesian position and velocity to orbital elements. It tests the model's capabilities with respect to circular, inclined orbits. These orbits are used to verify that the code can correctly convert circular inclined values and because the determination of the correct results by hand is a trivial matter.

Results:

Figure 5.3 shows the results computed for four different circular inclined orbits. They all had the same radius of 7,378,145 m and velocity magnitude of approximately 7,350.1 m/s. The only difference was the direction of the position and velocity vectors. This test case is slightly more complex than the previous two test cases. The basic orbit was a 30 degree inclined orbit with the RAAN equal to zero degrees. However, in order to test the RAAN computation, all of the position and velocity vectors were rotated by 45 degrees to place the RAAN at 45 degrees. Figure 5.3 shows nominal results for all four of the sub-tests run. While this may not be clear for tests two and four, they do represent nominal behavior despite the fact that the argument of periapsis value was non-zero. With these tests, due to numerical precision of the computer, the eccentricity was set higher than the eccentricity tolerance (1E-15) set in the source code. Therefore, these orbits were treated as eccentric orbits and the argument of periapsis was calculated. This is not a problem. The mean motion for this test was also 9.962E-4 rad/s.

Run directories:

09-12

Test OE_4: Elliptical Orbits

Purpose:

The purpose of this test is to examine the performance of the orbital element class when

	Test inputs (x, y, z)		Selected Orbital Elements					
			a (m)	e ()	i (deg)	Omega (deg)	true anomaly (deg)	arg. of periapsis (deg)
Test 1	position (m)	5217136.4	7378145.0	8.748E-16	30.0	45.0	0.0	0.0
		5217136.4						
		0.0						
	velocity (m/s)	-4501.024						
		4501.024						
		3675.071						
Test 2	position (m)	-4518172.6	7378145.0	1.734E-15	30.0	45.0	0.0	90.0
		4518172.6						
		3689072.5						
	velocity (m/s)	-5197.335						
		-5197.335						
		0.000						
Test 3	position (m)	-5217136.4	7378145.0	8.748E-16	30.0	45.0	180.0	0.0
		-5217136.4						
		0.0						
	velocity (m/s)	4501.024						
		-4501.024						
		-3675.071						
Test 4	position (m)	4518172.6	7378145.0	1.734E-15	30.0	45.0	0.0	270.0
		-4518172.6						
		-3689072.5						
	velocity (m/s)	5197.335						
		5197.335						
		0.000						

Figure 5.3: Table of Testing Results for Circular Inclined Orbits

dealing with elliptical orbits of all types.

Requirements:

By passing this test, the OrbitalElement class partially satisfies requirement OE_2 and partially satisfies requirement OE_1.

Procedure:

This test is designed to test the conversion from Cartesian position and velocity to orbital elements. It tests the model's capabilities with respect to elliptical orbits of all inclinations. The calculation of analytical results for these orbits is more complicated than for other cases but it is still possible.

In order to preserve the ability to validate the algorithms being tested, equations from section 1.6 of [1] were used to compute the initial conditions that would result in the desired orbital elements. For this test, there were constant eccentricity (.2) and constant radius at periapsis (7,378,145 m), and variable RAAN, inclination, and true anomaly. The inclination and RAAN were increased from zero by 45 degrees for each test and the orbit was tested with both 0 and 180 degree true anomaly because the formulations for position and velocity are easy to compute.

Results:

Figure 5.4 shows a table with the results from all of the studies performed with elliptical orbits. In all cases the results were nominal with occasional switches between angles of 0 and 360 degrees which occurred due to numerical precision issues.

Run directories:

13-20

Test OE_5: Conversion to Cartesian Position and Velocity

Purpose:

The purpose of this test is to examine the performance of the orbital element to Cartesian position and velocity conversion methods.

Requirements:

By passing this test, the OrbitalElement class partially satisfies requirement OE_2 and partially satisfies requirement OE_1.

Procedure:

This test is designed to test the conversion from Keplerian orbital elements to Cartesian position and velocity vectors. The test itself simply uses the output from all of the tests described previously and reconverts those orbital elements to the Cartesian velocity vectors. If the computation is performed correctly, the output position and velocity vectors should be exactly the same as the position and velocity vectors input for all of the previous tests.

Results:

Figure 5.5 shows a table of the first 8 tests from above (circular equatorial, and circular polar orbits). Figure 5.6 shows a table of the second 8 tests (circular, 45 degree inclination and RAAN, and partial elliptical orbits). And lastly, Figure 5.7 shows a table of the final four tests from the elliptical test cases.

During the course of this testing, several errors in the source code were discovered. The errors resulted from numerical instability of the acos C function. In order to remove this problem, all of the code was modified to use the atan2 function instead of acos to improve the numerical stability of the solution. Because of this change, the overall performance of the system for these tests is excellent. The answers were all consistent to 15 digits of precision which is the best performance that can be obtained with conventional double precision arithmetic.

Run directories:

21-40

Test OE_6: Circular Continuous Orbit

Purpose:

The purpose of this test is to examine the performance of the orbital element class with a circular orbit that is not just a single point test.

Requirements:

By passing this test, the OrbitalElement class partially satisfies requirement OE_2 and partially satisfies requirement OE_1.

Procedure:

This test is designed to examine the OrbitalElement class' performance when dealing with several continuous data points. It is run to ensure that there are no disconnects in the system when dealing with circular orbits and that there are no clear numerical instability issues.

In order to run this test, a nominal orbit was generated using a MATLAB propagator (ode45) and simplified system dynamics (two-body problem). The trajectory was initialized in a

circular orbit and then propagated for a period of time. The output position and velocity were then compared with the input position and velocity.

Results:

Figure 5.8 shows a plot of the position errors from the conversion process.

As this plot shows, the results all agree to the precision of the computer. Figure 5.9 shows a similar plot for the velocity errors away from nominal.

The velocity performance of the simulation is also quite good. Figure 5.10 shows a plot of the inclination error away from nominal for this trajectory.

As these tests show, the OrbitalElement class performed nominally for the tests on the circular orbit problem.

Run directories:

41

Test OE_7: Non-Circular Continuous Orbit

Purpose:
The purpose of this test is to examine the performance of the orbital element class with non-circular orbits of varying eccentricity.

Requirements:
By passing this test, the OrbitalElement class partially satisfies requirement OE_2 and partially satisfies requirement OE_1.

Procedure:
This test is designed to examine the OrbitalElement class' performance when dealing with several continuous data points. It is run to ensure that there are no disconnects in the system when dealing with circular orbits and that there are no clear numerical instability issues.

In order to run this test, a nominal orbit was generated using a MATLAB propagator (ode45) and simplified system dynamics (two-body problem). There are 8 different eccentricities of the orbit to test all of the possibilities and borderline cases.

Results:

Figure 5.11 shows a plot of the position errors for a continuous elliptical orbit.

Figure 5.12 shows a plot of the position errors in the conversion for a nearly parabolic trajectory that crosses the border between parabolic and eccentric trajectories due to propagation errors.

Figure 5.13 shows a plot of the position errors for a parabolic trajectory that remains in the parabolic region for the entire trajectory.

Figure 5.14 shows a plot of the position errors for a parabolic trajectory that crosses the line between parabolic and hyperbolic trajectories due to numerical propagation errors.

Figure 5.15 shows the position errors from a purely hyperbolic trajectory of eccentricity 1.25.

Figure 5.16 shows the position errors from a purely hyperbolic trajectory of eccentricity 2.5.

Figure 5.17 shows the position errors from a purely hyperbolic trajectory of eccentricity 3.9.

As these plots all show, the OrbitalElement class demonstrates good performance across the board. In the parabolic region, the digits of precision are slightly degraded because of numerical instability of the eccentric orbit calculations in this region.

As part of this testing, the tolerance values from the OrbitalElement class had to be slightly modified. The tolerance variable that determines when to switch between eccentric, parabolic, and hyperbolic trajectories was initially set at 1E-15. However, when on the border between eccentric and parabolic orbits, the eccentric orbit calculations experienced numerical instability because of a subtraction of two numbers that were nearly equal. When this was noticed, new tolerance variables were added and the values were tweaked until the numerical errors in the border region reached a minimum. The final value for the type-switching tolerance value was set at 1E-2.

Run directories:

42, 44-50

Test OE_8: Published Validation Cases

Purpose:

The purpose of this set of tests is to compare the performance of the orbital element class against published data from other sources (NASA, Vallado, etc).

Requirements:

By passing this test, the OrbitalElement class partially satisfies requirement OE_2 and partially satisfies requirement OE_1.

Procedure:

This test is designed to compare the OrbitalElement class' performance to published data from various sources. There are six tests run. The first three are courtesy of NASA's human spaceflight website, the next two are from Vallado, and the last one is from [1].

Results:

Figure 5.18 shows a table of the results from this test.

The results from this test are all nominal although it is not apparent at first. Particularly, with the Vallado test, where there are only 4 digits of precision, it does not appear that the methods are particularly accurate with regards to truth. However, the Vallado data is only given to four significant digits so the results shown here are actually nominal.

Test OE_9: Code Coverage Test

Purpose:

The purpose of this test is to ensure that the source code is adequately covered by the tests run.

Requirements:

By passing this test, the OrbitalElement class partially satisfies requirement OE_2 and partially satisfies requirement OE_1.

Procedure:

In this test, several different conditions were used to increase the code coverage of the entire test. This included things like negative anomalies and various other tests that should have no effect on the overall simulation. since the tabulated results do not really add value because they were only run to increase source coverage, the table has been omitted although it was examined as part of the testing.

The code coverage obtained through the testing was quite excellent. checking the results with the Gnu gcov code coverage tool, the statement coverage level was 87%. The only reason that coverage was not 100% is that some of the code failure cases where the system quits the simulation were not tested.

After this test, all requirements are entirely satisfied.

5.3 Requirements Traceability

Requirement	Inspection or test
OE_1 - Cartesian to Element Conversion	Insp. OE_1 - file inspection, All Tests
OE_2 - Element to Cartesian Conversion	Insp. OE_1 - file inspection, All Tests

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
Total	0	0	0	0

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::OrbitalElements:: OrbitalElements (void)	src/orbital_elements.cc	58	1
jeod::OrbitalElements::~ OrbitalElements (void)	src/orbital_elements.cc	90	1
jeod::OrbitalElements::set_ object_name (const char * name)	src/orbital_elements.cc	100	2
jeod::OrbitalElements::set_ planet_name (const char * name)	src/orbital_elements.cc	118	2
jeod::OrbitalElements::get_ object_name (void)	src/orbital_elements.cc	136	1
jeod::OrbitalElements::get_ planet_name (void)	src/orbital_elements.cc	149	1
jeod::OrbitalElements::from_ cartesian (double mu, const double pos[3], const double vel[3])	src/orbital_elements.cc	163	21
jeod::OrbitalElements::to_ cartesian (double mu, double pos[3], double vel[3])	src/orbital_elements.cc	406	9
jeod::OrbitalElements::nu_to_ anomalies ()	src/orbital_elements.cc	561	5
jeod::OrbitalElements::mean_ anom_to_nu ()	src/orbital_elements.cc	639	9
jeod::OrbitalElements::Kep EqtnE (double M, double e, double * E)	src/orbital_elements.cc	763	6
jeod::OrbitalElements::Kep EqtnH (double M, double e, double * H)	src/orbital_elements.cc	837	10
jeod::OrbitalElements::Kep EqtnB (double M, double * B)	src/orbital_elements.cc	888	2

	Test inputs (x, y, z)			Selected Orbital Elements					
				a (m)	e ()	i (deg)	Omega (deg)	true anomaly (deg)	arg. of periapsis (deg)
Test 1	position (m)	738145.00	-11067217.5	92222681.3	2.000E-01	0.0	0.0	0.0	0.0
	periapsis,	0.00	0.0						
	apoapsis	0.00	0.0						
	velocity (m/s)	0.00	0.00					180.0	
	periapsis,	8051.68	-5367.78						
	apoapsis	0.00	0.00						
Test 2	position (m)	5217136.36	-7825704.54	92222681.3	2.000E-01	45.0	45.0	0.0	0.0
	periapsis,	5217136.36	-7825704.54						
	apoapsis	0.00	0.00						
	velocity (m/s)	-4025.84	2683.89					180.0	
	periapsis,	4025.84	-2683.89						
	apoapsis	5693.40	-3795.60						
Test 3	position (m)	0.00	0.00	92222681.3	2.000E-01	90.0	90.0	360.0	0.0
	periapsis,	7378145.00	-11067217.50						
	apoapsis	0.00	0.00						
	velocity (m/s)	0.00	0.00					180.0	360.0
	periapsis,	0.00	0.00						
	apoapsis	8051.68	-5367.78						
Test 4	position (m)	-5217136.36	7825704.54	92222681.3	2.000E-01	135.0	135.0	360.0	0.0
	periapsis,	5217136.36	-7825704.54						
	apoapsis	0.00	0.00						
	velocity (m/s)	4025.84	-2683.89					180.0	
	periapsis,	4025.84	-2683.89						
	apoapsis	5693.40	-3795.60						
Test 5	position (m)	-738145.00	11067217.5	92222681.3	2.000E-01	180.0	0.0	360.0	180.0
	periapsis,	0.00	0.0						
	apoapsis	0.00	0.0						
	velocity (m/s)	0.00	0.00					180.0	
	periapsis,	8051.68	-5367.78						
	apoapsis	0.00	0.00						
Test 6	position (m)	-5217136.36	7825704.54	92222681.3	2.000E-01	135.0	45.0	360.0	180.0
	periapsis,	-5217136.36	7825704.54						
	apoapsis	0.00	0.00						
	velocity (m/s)	-4025.84	2683.89					180.0	
	periapsis,	4025.84	-2683.89						
	apoapsis	-5693.40	3795.60						
Test 7	position (m)	0.00	0.00	92222681.3	2.000E-01	90.0	90.0	0.0	180.0
	periapsis,	-7378145.00	11067217.50						
	apoapsis	0.00	0.00						
	velocity (m/s)	0.00	0.00					180.0	
	periapsis,	0.00	0.00						
	apoapsis	-8051.68	5367.78						
Test 8	position (m)	5217136.36	-7825704.54	92222681.3	2.000E-01	45.0	135.0	0.0	180.0
	periapsis,	-5217136.36	7825704.54						
	apoapsis	0.00	0.00						
	velocity (m/s)	4025.84	-2683.89					180.0	
	periapsis,	4025.84	-2683.89						
	apoapsis	-5693.40	3795.60						

Figure 5.4: Table of Testing Results for Elliptical Orbits

	Test outputs (x, y, z)			Selected Orbital Elements Input					
				p (m)	e ()	i (deg)	Omega (deg)	mean anomaly (deg)	arg. of periapsis (deg)
Test 1	position (m)	738145.00	unused	7378145.0	3.569E-17	0.0	0.0	0.0	0.0
	periapsis,	0.00							
	apoapsis	0.00							
	velocity (m/s)	0.00							
	periapsis,	7350.14							
	apoapsis	0.00							
Test 2	position (m)	0.00	unused	7378145.0	3.569E-17	0.0	0.0	90.0	0.0
	periapsis,	7378145.00							
	apoapsis	0.00							
	velocity (m/s)	-7350.14							
	periapsis,	0.00							
	apoapsis	0.00							
Test 3	position (m)	-7378145.00	unused	7378145.0	3.569E-17	0.0	0.0	180.0	0.0
	periapsis,	0.00							
	apoapsis	0.00							
	velocity (m/s)	0.00							
	periapsis,	-7350.14							
	apoapsis	0.00							
Test 4	position (m)	0.00	unused	7378145.0	3.569E-17	0.0	0.0	270.0	0.0
	periapsis,	-7378145.00							
	apoapsis	0.00							
	velocity (m/s)	7350.14							
	periapsis,	0.00							
	apoapsis	0.00							
Test 5	position (m)	738145.00	unused	7378145.0	3.569E-17	90.0	0.0	0.0	0.0
	periapsis,	0.00							
	apoapsis	0.00							
	velocity (m/s)	0.00							
	periapsis,	0.00							
	apoapsis	7350.14							
Test 6	position (m)	0.00	unused	7378145.0	3.569E-17	90.0	0.0	90.0	0.0
	periapsis,	0.00							
	apoapsis	738145.00							
	velocity (m/s)	-7350.14							
	periapsis,	0.00							
	apoapsis	0.00							
Test 7	position (m)	-7378145.00	unused	7378145.0	3.569E-17	90.0	0.0	180.0	0.0
	periapsis,	0.00							
	apoapsis	0.00							
	velocity (m/s)	0.00							
	periapsis,	0.00							
	apoapsis	-7350.14							
Test 8	position (m)	0.00	unused	7378145.0	3.569E-17	90.0	0.0	270.0	0.0
	periapsis,	0.00							
	apoapsis	-738145.00							
	velocity (m/s)	7350.14							
	periapsis,	0.00							
	apoapsis	0.00							

Figure 5.5: First Table of Element to Cartesian Testing Results

Test 9	position (m)	5217136.36	unused	7378145.0	8.748E-16	30.0	45.0	0.0	0.0
	periapsis,	5217136.36							
	apoapsis	0.00							
	velocity (m/s)	-4501.02							
	periapsis,	4501.02							
	apoapsis	3675.07							
Test 10	position (m)	-4518172.62	unused	7378145.0	1.734E-15	30.0	45.0	0.0	90.0
	periapsis,	4518172.62							
	apoapsis	3689072.50							
	velocity (m/s)	-5197.33							
	periapsis,	-5197.33							
	apoapsis	0.00							
Test 11	position (m)	-5217136.36	unused	7378145.0	8.748E-16	30.0	45.0	180.0	0.0
	periapsis,	-5217136.36							
	apoapsis	0.00							
	velocity (m/s)	4501.02							
	periapsis,	-4501.02							
	apoapsis	-3675.07							
Test 12	position (m)	4518172.62	unused	7378145.0	1.734E-15	30.0	45.0	0.0	270.0
	periapsis,	-4518172.62							
	apoapsis	-3689072.50							
	velocity (m/s)	5197.33							
	periapsis,	5197.33							
	apoapsis	0.00							
Test 13	position (m)	7378145.00	-11067217.50	8853774.0	2.000E-01	0.0	0.0	0.0	0.0
	periapsis,	0.00	0.00					180.0	
	apoapsis	0.00	0.00						
	velocity (m/s)	0.00	0.00						
	periapsis,	8051.68	-5367.78						
	apoapsis	0.00	0.00						
Test 14	position (m)	5217136.36	-7825704.54	8853774.0	2.000E-01	45.0	45.0	0.0	0.0
	periapsis,	5217136.36	-7825704.54					180.0	
	apoapsis	0.00	0.00						
	velocity (m/s)	-4025.84	2683.89						
	periapsis,	4025.84	-2683.89						
	apoapsis	5693.40	-3795.60						
Test 15	position (m)	0.00	0.00	8853774.0	2.000E-01	90.0	90.0	0.0	0.0
	periapsis,	7378145.00	-11067217.50					180.0	360.0
	apoapsis	0.00	0.00						
	velocity (m/s)	0.00	0.00						
	periapsis,	0.00	0.00						
	apoapsis	8051.68	-5367.78						
Test 16	position (m)	-5217136.36	7825704.54	8853774.0	2.000E-01	135.0	135.0	360.0	0.0
	periapsis,	5217136.36	-7825704.54					180.0	0.0
	apoapsis	0.00	0.00						
	velocity (m/s)	4025.84	-2683.89						
	periapsis,	4025.84	-2683.89						
	apoapsis	5693.40	-3795.60						

Figure 5.6: Second Table of Element to Cartesian Testing Results

Test 17	position (m)	-7378145.00	11067217.50	8853774.0	2.000E-01	180.0	0.0	360.0	180.0
	periapsis,	0.00	0.00						
	apoapsis	0.00	0.00					180.0	
	velocity (m/s)	0.00	0.00						
	periapsis,	8051.68	-5367.78						
	apoapsis	0.00	0.00						
Test 18	position (m)	-5217136.36	7825704.54	8853774.0	2.000E-01	135.0	45.0	360.0	180.0
	periapsis,	-5217136.36	7825704.54						
	apoapsis	0.00	0.00					180.0	
	velocity (m/s)	-4025.84	2683.89						
	periapsis,	4025.84	-2683.89						
	apoapsis	-5693.40	3795.60						
Test 19	position (m)	0.00	0.00	8853774.0	2.000E-01	90.0	90.0	0.0	180.0
	periapsis,	-7378145.00	11067217.50						
	apoapsis	0.00	0.00					180.0	
	velocity (m/s)	0.00	0.00						
	periapsis,	0.00	0.00						
	apoapsis	-8051.68	5367.78						
Test 20	position (m)	5217136.36	-7825704.54	8853774.0	2.000E-01	45.0	135.0	0.0	180.0
	periapsis,	-5217136.36	7825704.54						
	apoapsis	0.00	0.00					180.0	
	velocity (m/s)	4025.84	-2683.89						
	periapsis,	4025.84	-2683.89						
	apoapsis	-5693.40	3795.60						

Figure 5.7: Third Table of Element to Cartesian Testing Results

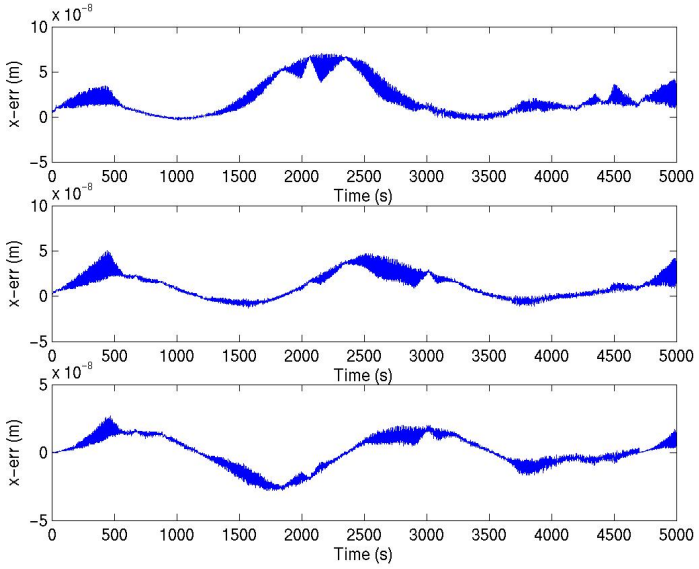

Figure 5.8: Plot of position error for circular orbit

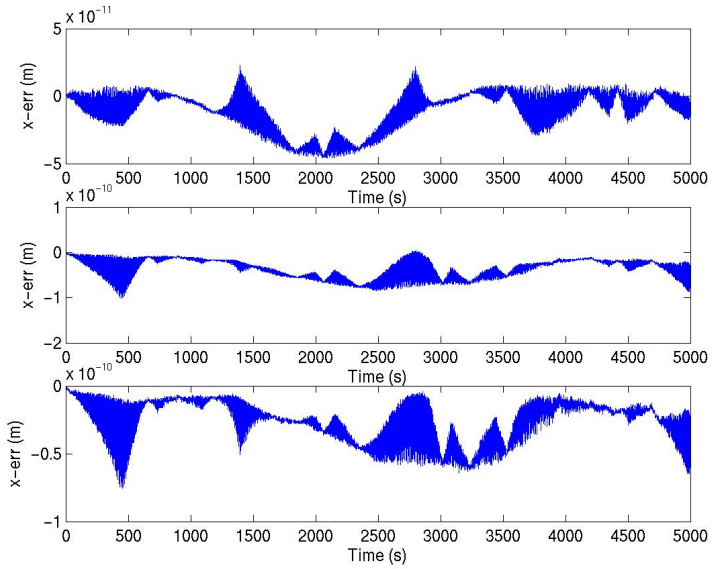

Figure 5.9: Plot of velocity error for circular orbit

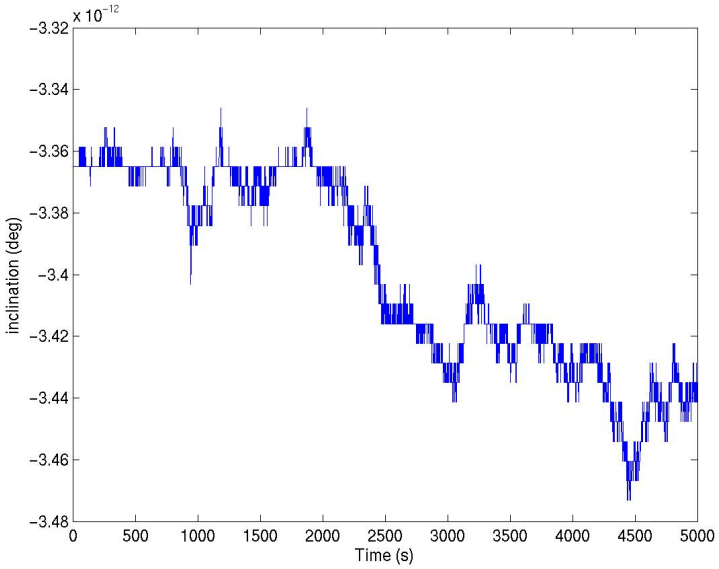

Figure 5.10: Plot of inclination error for circular orbit

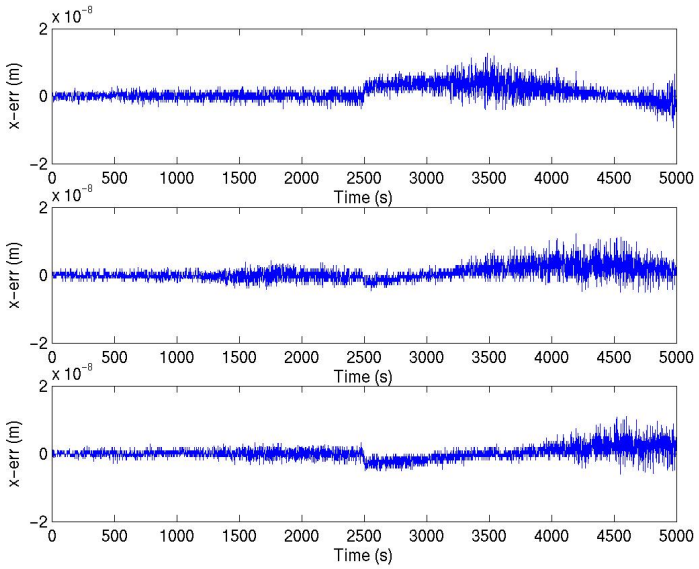

Figure 5.11: Plot of position error for elliptical orbit

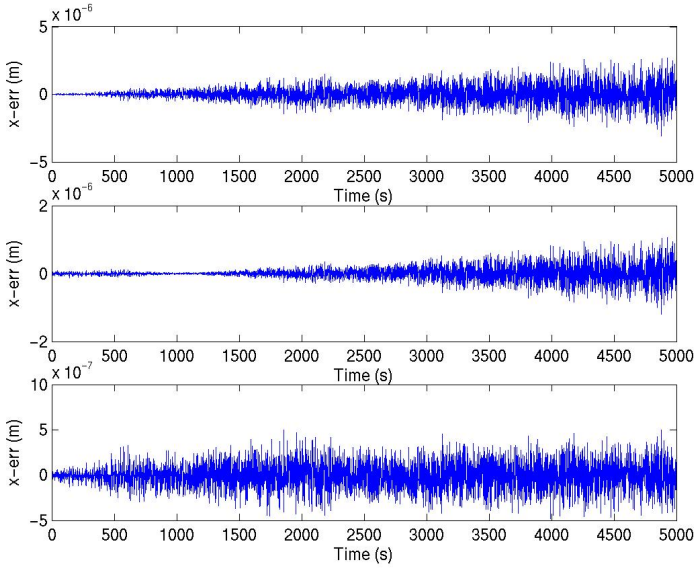

Figure 5.12: Plot of position error for low parabolic eccentricity

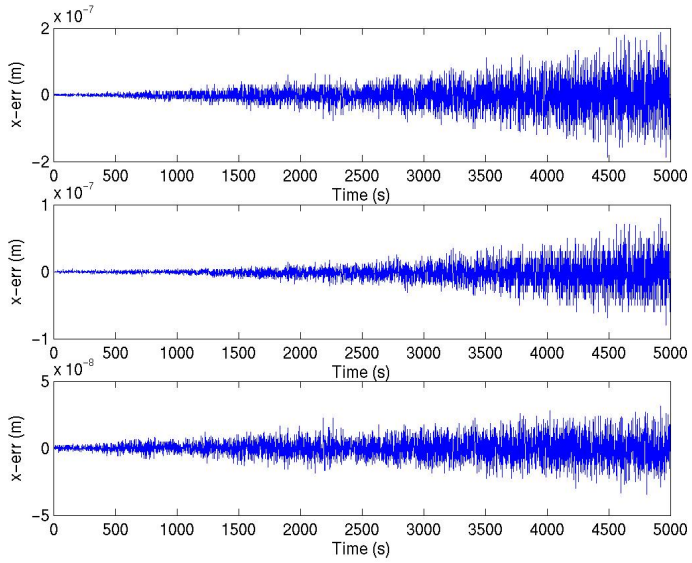

Figure 5.13: Plot of position error for mid parabolic eccentricity

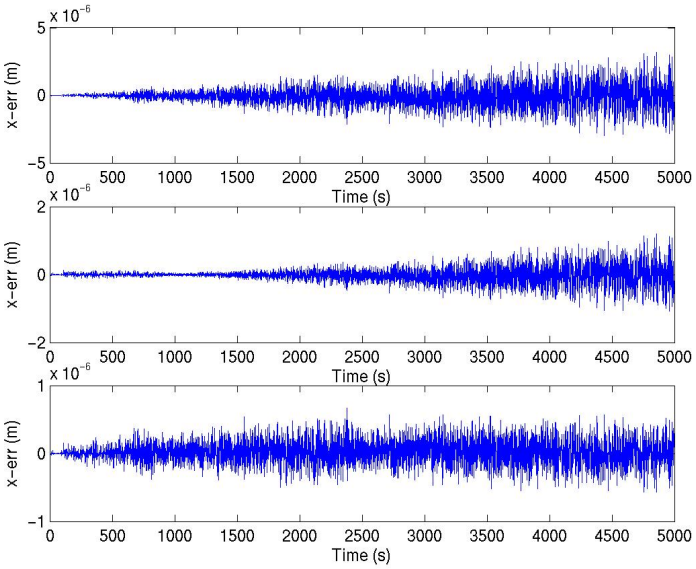

Figure 5.14: Plot of position error for high parabolic eccentricity

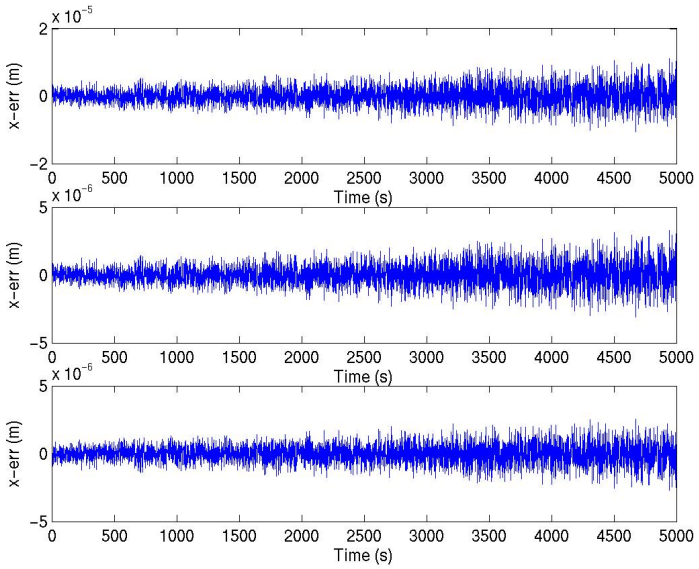

Figure 5.15: Plot of position error for hyperbolic eccentricity 1.25

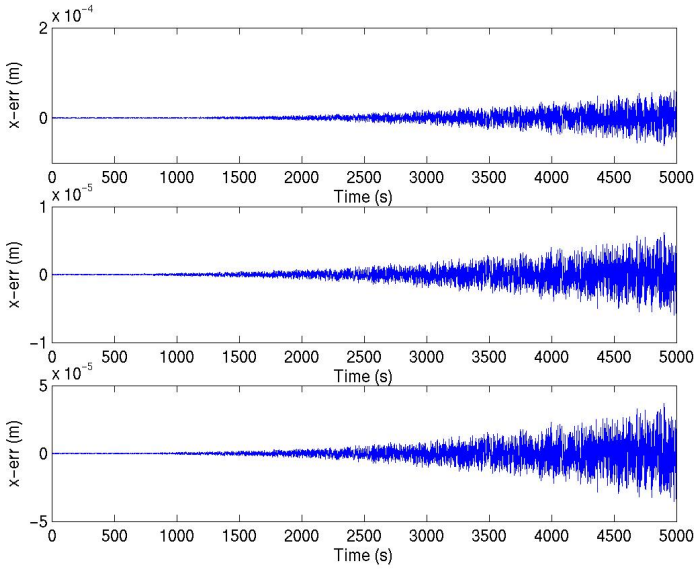

Figure 5.16: Plot of position error for hyperbolic eccentricity 2.5

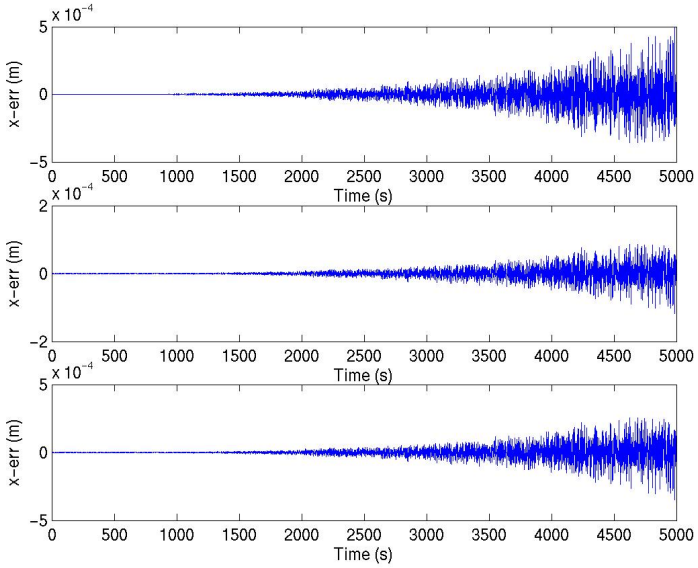

Figure 5.17: Plot of position error for hyperbolic eccentricity 3.9

		Selected Orbital Elements					position			source
		p (m)	e ()	i (deg)	Omega (deg)	ωp (deg)	x (m)	y (m)	z (m)	
Test 1	reference	6725408.2	1.87E-03	51.79	30.30	82.78	4671696.48	-1482158.85	-4620191.58	NASA
	calculated	6725408.2	1.87E-03	51.79	30.30	82.78	4671696.02	-1482158.92	-4620191.82	
	difference	-6.44E-04	2.56E-08	4.82E-07	-3.42E-06	4.12E-05	-4.61E-01	-6.95E-02	-2.40E-01	
Test 2	reference	6731934.0	1.33E-03	51.79	25.23	83.78	-4640789.30	-4261640.89	-2384608.32	NASA
	calculated	6731934.0	1.33E-03	51.79	25.23	83.78	-4640789.06	-4261641.09	-2384608.91	
	difference	1.25E-02	-4.88E-08	-3.19E-06	-3.27E-07	7.04E-06	2.43E-01	-2.00E-01	-5.91E-01	
Test 3	reference	6724305.8	1.78E-03	51.74	20.07	61.96	-3884577.77	2621655.56	4811943.86	NASA
	calculated	6724305.8	1.78E-03	51.74	20.07	61.96	-3884576.95	2621656.24	4811944.02	
	difference	4.74E-03	-2.28E-08	2.13E-07	4.15E-06	1.62E-05	8.23E-01	6.78E-01	1.60E-01	
Test 4	reference	6732527.3	2.46E-03	51.74	14.98	67.47	5657077.34	3080301.15	1918782.98	NASA
	calculated	6732527.3	2.46E-03	51.74	14.98	67.47	5657076.81	3080301.62	1918783.62	
	difference	-6.96E-03	-1.17E-08	-3.82E-06	-2.73E-06	1.27E-05	-5.35E-01	4.72E-01	6.38E-01	
Test 5	reference	11067790.0	8.33E-01	87.87	227.89	53.38	6525483.40	6862875.00	6448296.00	Vallado
	calculated	11067798.4	8.33E-01	87.87	227.90	53.38	6525183.48	6861323.43	6449193.03	
	difference	8.35E+00	3.99E-07	-8.74E-04	8.26E-03	4.93E-03	-3.00E+02	-1.55E+03	8.97E+02	
Test 6	reference	2.3	5.00E-01	45.00	30.00	0.00	1.30	0.75	0.00	BMW
	calculated	2.2	5.00E-01	45.00	30.00	360.00	1.30	0.75	0.00	
	difference	-1.02E-14	-6.00E-15	3.27E-13	-4.62E-14	3.60E+02	-9.99E-15	9.99E-16	0.00E+00	

Figure 5.18: Validation cases for OrbitalElement class

Bibliography

[1] Bate, R., Mueller, D., White J.E. *Fundamentals of Astrodynamics*. Dover Publications Inc., New York, 1971.

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

[3] Vallado, D.A. *Fundamentals of Astrodynamics and Applications*. McGraw-Hill, New York, 1997.