Orbit Determination Using Modern Filters/Smoothers and Continuous Thrust Modeling

by

Zachary James Folcik

B.S. Computer Science
Michigan Technological University, 2000

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2008

Signature of Author:_____
Department of Aeronautics and Astronautics
May 23, 2008

Certified by:_____
Dr. Paul J. Cefola
Lecturer, Department of Aeronautics and Astronautics
Thesis Supervisor

Certified by:_____
Professor Jonathan P. How
Professor, Department of Aeronautics and Astronautics
Thesis Advisor

Accepted by:_____
Professor David L. Darmofal
Associate Department Head
Chair, Committee on Graduate Students

[This page intentionally left blank.]

Orbit Determination Using Modern Filters/Smoothers and Continuous Thrust Modeling

by

Zachary James Folcik

Submitted to the Department of Aeronautics and Astronautics on May 23, 2008
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Aeronautics and Astronautics

ABSTRACT

The development of electric propulsion technology for spacecraft has led to reduced costs and longer lifespans for certain types of satellites. Because these satellites frequently undergo continuous thrust, predicting their motion and performing orbit determination on them has introduced complications for space surveillance networks. One way to improve orbit determination for these satellites is to make use of new estimation techniques. This has been accomplished by applying the Backward Smoothing Extended Kalman Filter (BSEKF) to the problem of orbit determination. The BSEKF outperforms other nonlinear filters because it treats nonlinearities in both the measurement and dynamic functions. The performance of this filter is evaluated in comparison to an existing Extended Semianalytic Kalman Filter (ESKF). The BSEKF was implemented in the R&D Goddard Trajectory Determination System (GTDS) for this thesis while the ESKF was implemented in 1981 and has been tested extensively since then. Radar and optical satellite tracking observations were simulated using an initial truth orbit and were processed by the ESKF and BSEKF to estimate satellite trajectories. The trajectory estimates from each filter were compared with the initial truth orbit and were evaluated for accuracy and convergence speed. The BSEKF provided substantial improvements in accuracy and convergence over the ESKF for the simulated test cases. Additionally, this study used the solutions offered by optimal thrust trajectory analysis to model the perturbations caused by continuous thrust. Optimal thrust trajectory analysis makes use of Optimal Control Theory and numerical optimization techniques to calculate minimum time and minimum fuel trajectories from one orbit to another. Because satellite operators are motivated to save fuel, it was assumed that optimal thrust trajectories would be useful to predict thrust perturbed satellite motion. Software was developed to calculate the optimal trajectories and associated thrust plans. A new force model was implemented in GTDS to accept externally generated thrust plans and apply them to a given satellite trajectory. Test cases are presented to verify the correctness of the mathematics and software. Also, test cases involving a real satellite using electric propulsion were executed. These tests demonstrated that optimal thrust modeling could provide order of magnitude reductions in orbit determination errors for a satellite with low-thrust electric propulsion.

Thesis Supervisor: Dr. Paul J. Cefola
Title: Lecturer, Department of Aeronautics and Astronautics

DISCLAIMERS

The views expressed in this article are those of the author and do not reflect the official policy or position of MIT, MIT Lincoln Laboratory, the United States Air Force, Department of Defense, or the U.S. Government.

This work is sponsored by the Air Force under Air Force Contract FA8721-05-C-002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Acknowledgements

I would first like to thank Paul Cefola for his outstanding advising, teaching and mentoring over the past six years. I have been very lucky to have such a thoughtful and attentive thesis advisor. Through your example, you have inspired me to always strive for excellence in my work. I look forward to many more years of collaboration with you.

I thank David Chan, Rick Abbot, Susan Andrews, Forrest Hunsberger, Val Heinz, and Curt von Braun for their support and guidance which allowed me to enter the Lincoln Scholars Program. I am very grateful for the opportunity to go back to school. I also thank Bonnie Tuohy and Deborah Simmons for their gracious help with release review.

I also thank Sid Sridharan and Kent Pollock for hiring me at MIT Lincoln Laboratory. Kent, you have taught me so much about software programming and how to do so in a team. Sid, I thank you for guiding my career, giving me interesting projects and showing me how to be a creative engineer.

I thank my family, Tom and Peggy Folcik and Jeffrey Dawley, for their constant support, encouragement and love. Your words and understanding helped me stay grounded and secure during my foray into graduate school.

I am very grateful to the system administrators in Group 93: Susan Champigny, Douglas Piercey, Ernest Cacciapuoti, John Duncan, and Edward Christiansen. Thank you very much for your diligence and time in resurrecting the data I needed for this thesis.

I thank the members of the Lincoln Scholars Committee for giving me the opportunity to join the program and go to graduate school. I thank my academic advisor, Prof. Jonathan How, for his guidance. I also thank Jean Kechichian at the Aerospace Corporation for his advice in trajectory optimization. I also thank Paul Schumacher and Jim Wright for their helpful comments at the American Astronautical Society meeting in Galveston, TX.

I appreciate the assistance of Erick Lansard, the Director of Research at Thales Alenia Space, Leonardo Mazzini at Thales Alenia Space, Heiner Klinkrad at the European Space Operations Center, and Andrea Cotellessa at the European Space Agency in being very responsive to my requests for information about the ARTEMIS satellite and its recovery. I thank you for all of the information you provided.

I appreciate the efforts of Paul Cefola, Wayne McClain, Ron Proulx, Mark Slutsky, Leo Early, and the many MIT students at Draper Laboratory who have made GTDS and DSST what it is today. Your work has culminated in an orbit determination tool that is accurate, useful, and extensible. My thesis work would have been much more difficult, if not impossible, without standing on your shoulders.

[This page intentionally left blank]

Table of Contents

List of Figures

[This page intentionally left blank]

List of Tables

[This page intentionally left blank]

Chapter 1 Introduction

1.1 Motivation

Orbit determination has had a long, remarkable history. Its roots lie in astronomy and in particular, predicting the motion of planets and comets. Copernicus, Kepler, Newton, Lagrange, Gauss and others have contributed much to this science and it is upon their shoulders that work continues today. Newton's Laws of Gravity still serve as the starting point for modeling the motion of orbiting satellites. Carl Friedrich Gauss invented and first used the method of least-squares and his method still serves as the basis for orbit determination. Many impressive methods and techniques for orbit prediction and determination have been invented since the start of the Space Age, but these inventions all rely on the fundamental work done well before man-made Earth satellites were launched.

The work presented in this thesis is based on the enormous body of work that has come before it. Improvement in orbit determination in specific cases is still a research area that sees several advances each year. In the experience of the author, orbit determination for satellites upon which unmodeled thrusting forces act remains a topic with unsolved problems. These problems are relevant to the field of space surveillance in which populations of satellites are non-cooperatively tracked in order to maintain knowledge about their orbits. Spacecraft that are undergoing orbit transfers or station-keeping are more challenging for space surveillance because the motion cannot simply be modeled with known natural forces. Additional modeling of thrusting forces must be

undertaken to accurately capture spacecraft motion. Modeling such thrusting forces is most challenging when the spacecraft operator and the space surveillance network do not cooperate to share information. For these non-cooperative space surveillance cases, satellite orbit determination and prediction systems with maneuver detection and prediction capabilities are sought to improve *space situational awareness*. Such systems would help provide more accurate predictions of satellite motion for spacecraft with either chemical or electric propulsion. Figure 1.1 depicts such a system.

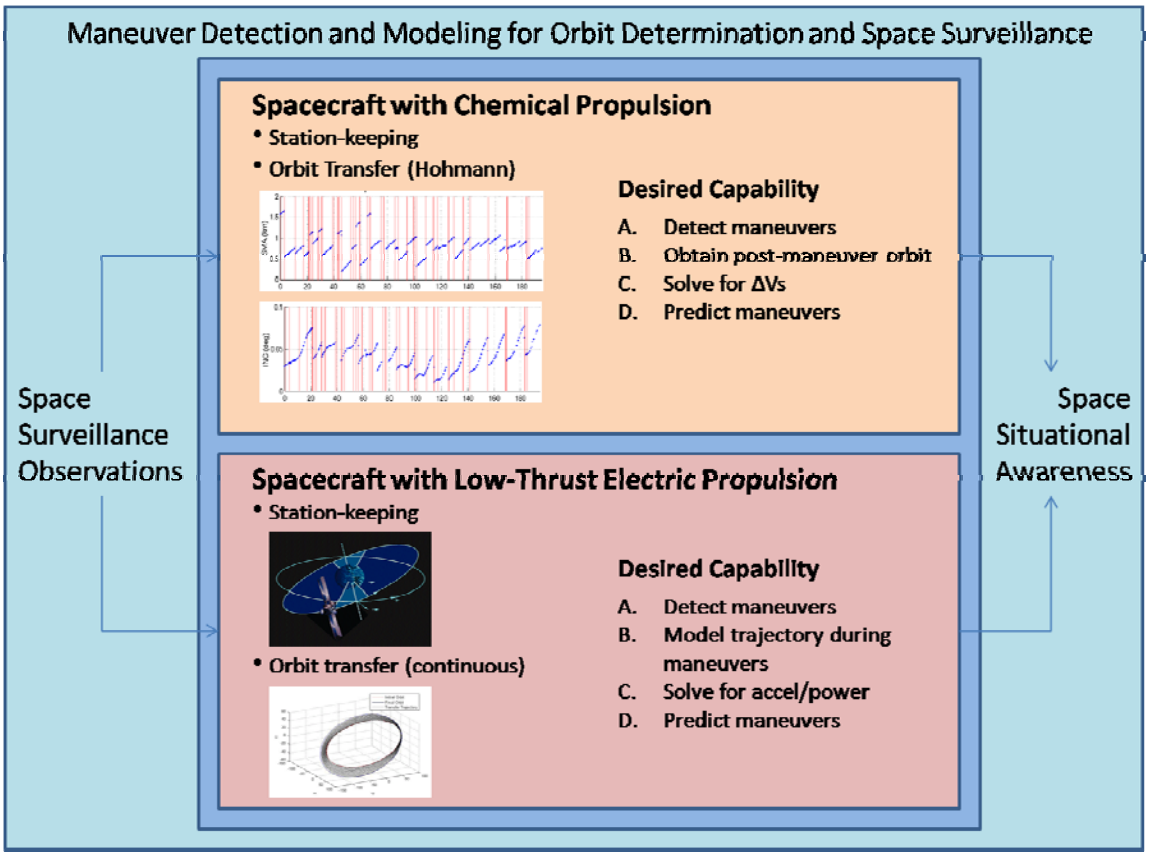

Figure 1.1 Notional System for Enhancing Space Surveillance for Thrusting Spacecraft

In previous work (1), the authors presented ways in which unmodeled chemical thrusting satellite maneuvers can be detected. However, improved methods for quickly

obtaining accurate orbital estimates after a satellite maneuvers are still sought. In reference (1), a batch, Bayesian Least Squares estimator is used to obtain post-maneuver orbital estimates because of limitations in using an Extended Kalman Filter (EKF). However, improvements in sequential estimators have been made since the original EKF formulation. Among these estimators is the Backwards Smoothing Extended Kalman Filter (BSEKF) developed by Mark Psiaki (2). This estimator treats nonlinearities more accurately than does the EKF and so converges more robustly and produces more accurate estimates. For this thesis, the BSEKF was implemented in the Goddard Trajectory Determination System (GTDS) and was evaluated in simulated observation test cases.

In addition, an attempt has been made to improve orbit prediction for a new and growing collection of man-made satellites. These satellites use low-thrust, electric propulsion technology. This technology is based on ion and plasma physics. Because of the constraints on electric propulsion in space, i.e. solar panels and batteries provide limited electrical power; such thrusters provide small accelerations compared with more common chemical thrusters. Because of these small accelerations, ion-electric thrusters must operate continuously rather than impulsively, and this affects the modeling that must be done to accurately predict the motion of these satellites. Experience has shown that high accuracy orbits cannot be obtained when neglecting thrust accelerations. This thesis develops initial predictive models that are based on *Optimal Control Theory*. A central assumption made regarding the operation of satellites using ion thrusters is that the fuel-optimal solution governs their control. With this assumption, optimal control

theory yields useful control solutions that can be used to model the perturbing accelerations due to low, continuous thrust. The substantial work of Jean Kechichian (3), (4), (5), (6), (7), (8), (9) has been leveraged for the calculation of optimal thrust trajectories for satellites. Software has been developed to generate optimal thrust plans and to use those thrust plans to model satellite motion within the GTDS framework.

1.2 Overview of Thesis

The overall purpose of this thesis is to evaluate an application of the BSEKF algorithm for orbit estimation and to develop an orbital motion model for low-thrust satellites. Specifically, the BSEKF algorithm has been implemented in the R&D GTDS software at MIT Lincoln Laboratory. The BSEKF has been coupled to the Draper Semianalytic Satellite Theory (DSST) forming the Backward Smoothing Extended Semianalytic Kalman Filter (BSESKF) algorithm. The DSST algorithm was chosen as the first orbit propagator to be used with the BSEKF algorithm because of its high computational efficiency and the linearity of the mean equinoctial orbital elements propagated by DSST.

Chapter 2 provides background on orbit propagation. Sections 2.1.1 and 2.1.2 provide some background on Cowell's propagation method and analytical propagation methods, respectively. Section 2.1.3 develops some of the mathematical background for DSST. Section 2.2 provides background for electric propulsion (EP) for satellites. Several types of EP engines are described. In Section 2.2.2, background in optimal

control theory is presented and Jean Kechichian's (5) formulation for solving constant thrust, fuel-optimal satellite control problems is detailed. Background for recursive orbit estimation techniques including the BSEKF is provided in section 2.3.

Chapter 3 outlines previous work done by Stephen Taylor (10) to couple the Extended Kalman Filter (EKF) to DSST and thus form the Extended Semianalytic Kalman Filter (ESKF). Coupling the BSEKF to DSST was done similarly.

The complete BSESKF algorithm, its implementation in GTDS, and simulation results are described in detail in Chapter 4. Section 4.1 details the BSESKF algorithm. Section 4.2 details the software implementation of the BSESKF. Sections 4.3 and 4.4 describe the testing methodology and the estimation results for the BSESKF using simulated observations. These results show that the BSESKF is more accurate and converges in less time than the ESKF.

Chapter 5 documents the optimal thrust plan software and presents verification of the correctness of the solutions generated by the software. Section 5.1 documents the standalone optimal thrust planning software. Section 5.2 describes the software modifications done in GTDS to model the thrust plans created by the optimal thrust planning software. These modifications allow GTDS to make orbit predictions and perform orbit determination using externally generated thrust plans. Section 5.3 describes verification testing done for the optimal thrust planning software. Section 5.4 records the results of real data test cases developed for the ARTEMIS satellite. The real

data test cases demonstrate the usefulness of optimal thrust plans for modeling continuous thrust accelerations of a satellite. In some cases, orbit determination which did not converge when thrust was unmodeled, converged when using optimal thrust plans to model thrust acceleration. In all test cases attempted, optimal thrust plans substantially improved orbit determination metrics and agreement with AFSSN data. In some cases, the observation residuals improved in aggregate by an order of magnitude or more.

Chapter 6 summarizes the conclusions of the thesis and outlines future work to improve orbit determination and EP modeling capability. Appendix A documents new GTDS keywords implemented for this thesis. Appendix B documents the B^L matrix and its partial derivatives which are used in Kechichian's optimal thrust trajectory formulation and in the optimal thrust planning software implemented for this thesis. Appendix C describes the Euler-Hill rotating polar reference frame and how coordinates in that frame can be transformed to the Earth centered inertial frame. This transformation was useful in applying thrust plans to acceleration modeling in GTDS. Appendix D describes how to execute and operate the optimal thrust planning software. Appendices E and F include the source code for the exact equation and averaged equation optimal thrust planning software.

Chapter 2 Background in Orbit Propagation, Optimal Thrust Trajectories, and Estimation

2.1 Satellite Orbit Propagation

Satellite orbit propagation is the problem of starting with initial conditions for a satellite orbit and calculating the satellite's position at later times. Typically the initial conditions are in the form of orbital elements of which at least six are required. These six orbital parameters, plus the epoch time comprise an orbital *element set* which describes the orbital size, shape, orientation and phase of a satellite in its orbit. The *element set* can also be thought of as a state vector describing the motion of a satellite. The six parameters are most commonly the Keplerian elements, i.e. semimajor axis, eccentricity, inclination, right ascension of the ascending node, argument of perigee and mean anomaly. Another common orbital element set is composed of Cartesian position and velocity vectors. A third orbital element set is the set of equinoctial elements. These elements have useful properties and are described in detail in section 2.1.3.5.

A common way to calculate the satellite state at some time given the satellite state at some initial time involves using *variational of parameter (VOP) equations* which compute the derivatives of the orbital elements with respect to time. The VOP equations are used along with a numerical integration technique such as the Runge-Kutta method or a finite-difference method (11) to integrate the VOP equations. This numerical integration procedure produces a time history of the orbital elements from the time of the initial condition to the desired output time. Because integration methods require the use

of an integration grid, the desired final time may be overshot by the numerical integration. Interpolation can be used to produce the orbital elements at exactly the desired final time. The accuracy of the satellite orbit propagation depends on how well the right hand sides of the VOP equations describe the actual physical motion of the satellite. Many forces need to be modeled in order to accurately predict satellite motion. The basic Newtonian two-body force describes the circular, elliptical, or hyperbolic conic trajectory of a satellite around a central body such as the Earth. Other forces arise from the non-spherical shape of the central body, third-body gravity, atmospheric drag, radiation pressure, central body tidal deformations, propulsion devices, and others. These forces are generally much smaller in magnitude than the basic two-body force and so are modeled as *perturbations* to the basic equations of motion. *Perturbations* are deviations from the undisturbed two-body motion. There are three main ways to include the effects of perturbations in modeling satellite motion. Special perturbation techniques such as Cowell's method numerically integrate the equations of motion and include all acceleration terms that perturb the two-body motion. The solutions obtained using special perturbations are specific to the initial conditions used. Cowell's method is described in section 2.1.1. General perturbations, i.e. analytic methods, such as the methods introduced by Kozai and Brouwer in the late 1950s and 1960s (12), (13) employ analytic expressions for the satellite perturbations. These general perturbations are very efficient computationally. Some of these theories can yield high accuracy, but the most commonly used today, i.e. SGP4, suffers from reduced accuracy because of the term truncation (14). Section 2.1.2 briefly discusses analytic theories. *Semianalytic* methods separate the effects of perturbations causing long-period and secular deviations from the

two-body motion from the effects of perturbations causing short-period deviations. This separation allows for speedy integration of the equations of motion that include the long-period and secular motion. The short-period motion is added only when high accuracy is required. In this way, semianalytic methods provide accuracy and computational efficiency (15), (14). One semianalytic method, the Draper Semianalytic Satellite Theory (DSST) developed by Paul Cefola, Wayne McClain, Leo Early, Ron Proulx, Mark Slutsky and others, is described in section 2.1.3.

2.1.1 Satellite Orbit Propagation Using Cowell's Method

One of the most common methods for orbit propagation is Cowell's method. This method predicts satellite motion by numerically integrating the equations of motion in terms of Cartesian elements, i.e. rectangular, position and velocity (11). The following equation describes the disturbed relative motion of two bodies (11):

$$\frac{d^2\mathbf{r}}{dt^2} + \frac{\mu}{r^3}\mathbf{r} = \mathbf{a}_d \tag{2.1}$$

Here, $\mathbf{r}$ is the position vector of a satellite with respect to the central mass, $\mathbf{a}_d$ is the vector of acceleration arising from the presence of general disturbing forces, and μ is the constant of gravitation associated with the central body (11). The Cowell method has the virtue of being the most straightforward way to determine the position, $\mathbf{r}(t)$, and velocity, $\mathbf{v}(t)$. The equations of motion used in this method are described in several references including (11) and (16). While this method is widely used for its simplicity, it suffers from some drawbacks. Because the disturbing forces modeled by $\mathbf{a}_d$ are often small in comparison to the two-body forces, many of the significant figures used in the

calculations will only be used to reproduce the dominating two-body motion (16). In order to maintain reasonable accuracy in the integrations of the equations of motion and accurately include the perturbing acceleration terms, small step-sizes are needed. When using finite precision arithmetic and when integrating for long periods, truncation error and round off error build up as the square root of the number of calculations performed (16). Because of these drawbacks, analytic and semianalytic methods have been developed.

2.1.2 Analytic Satellite Theory

Analytical satellite theories have been developed and used operationally for many years (13), (17), (18). The SGP theory developed by C. G. Hilton and J. R. Kuhlman used a simplified version of Kozai's gravitational theory (18). A modified form of Brouwer's gravitational theory was used in SGP4, the successor to SGP (19). SGP4 is described in references (17) and (20). The analytic satellite motion theories introduced by Brouwer use canonical transformations to separate the short period, long period and secular components of the motion (15), (18). Brouwer's method operates with what are known as *double averaged* equations of motion and is purely analytic in its formulation. The SGP and SGP4 analytic theories truncate many of the terms in Kozai's and Brouwer's gravitational theories, respectively, but allow the satellite motion to be propagated very quickly in terms of computing time. However, the term truncation reduces the accuracy of SGP and SGP4 over the original formulations from which they were derived. Nevertheless, the double averaged methods have proven very useful in that

they enable computation of all earth satellites on a daily basis to perform satellite orbit catalog maintenance.

It should be noted that analytic theory based systems that provide higher accuracy than SGP and SGP4 have been developed. The Aeronutronic Complete First-Order General Perturbations (AGP) theory includes first and second order secular terms. AGP also includes first order long and short periodic expressions with coefficients of Earth gravity terms J_2, J_3 and J_4 (18). AGP was the antecedent of SGP theory (18). The SGP4 Theory was written by Cranford in 1970, but was derived as a truncated form of the AFGP4 Theory. The AFGP4 Theory was developed by Lane and Cranford and included gravitational zonal terms through J_5 (21). A power density function was used to model atmospheric density for the drag calculations (19). The ANODE analytic orbit determination system developed and used at MIT Lincoln Laboratory includes analytic lunar and solar gravitational perturbations, analytic drag perturbations and perturbations due to geopotential terms J_2, J_2^2, J_3 and J_4 (22) (23). Applications requiring high accuracy for which SGP4 is not suitable are growing in number and include satellite maneuver detection, atmospheric density correction, high precision orbit catalog maintenance, long-term orbit evolution, etc.

2.1.3 Draper Semianalytic Satellite Theory

The Draper Semianalytic Satellite Theory (DSST) developed at the Computer Sciences Corporation (CSC) and later at the Charles Stark Draper Laboratory (CSDL) is

an efficient alternative to the brute-force numerical integration techniques. Although computation per dollar has drastically improved since the conception of DSST, computationally efficient algorithms such as DSST remain relevant. Applications such as monitoring the orbital elements of all Earth satellites, performing atmospheric density correction, satellite maneuver detection, long-term mission studies and other applications requiring the prediction and orbit determination of thousands of satellite orbits in reasonable time frames benefit from efficient analytic or semianalytic orbit prediction methods. The implementation of DSST included in the R&D version of GTDS is used extensively in this study. This section serves to describe DSST in some detail.

DSST was developed by Paul Cefola, Wayne McClain, Leo Early, Ron Proulx, Mark Slutsky and their colleagues at the Computer Sciences Corporation and the Charles Stark Draper Laboratory (CSDL) in the 1970s and 1980s. In its development at the CSDL, DSST also benefited from numerous enhancements made by MIT graduate students under the direction of the CSDL staff. DSST was developed with an emphasis on accuracy and computational efficiency. To accomplish this, DSST models conservative perturbing forces with Lagrangian Variation of Parameters (VOP) and non-conservative perturbing forces with Gaussian VOP. The Generalized Method of Averaging is used to isolate the short periodic motion from long period and secular motion. Through this method, the averaged VOP equations of motion and the short periodic models are obtained (15).

DSST differs from purely analytical methods in that it is a hybrid approach taking advantage of the efficiency of analytic methods and the accuracy of special perturbations methods. DSST is a *single averaged* approach. It uses the Generalized Method of Averaging to isolate the short period satellite motion, i.e. the motion on the order of one satellite orbital period. The long period and the secular motion are retained in the equations of motion. Integrators with relatively large time steps are used to integrate the secular and long-period motion (15). DSST propagates long period and secular satellite motion using a set of mean equinoctial elements. The near-linear mean elements have the advantage of allowing large integration time steps. The equinoctial coordinates avoid singularities for small inclination and eccentricity orbits. Because of the large time steps allowed, DSST is computationally efficient. DSST also provides highly accurate orbits by computing short period motion when explicitly needed, i.e. when ephemeris points are requested or when observations are computed. The short period motion is computed using Fourier series in the fast orbital element which accurately and efficiently reproduces the motion (10). Specifically, the short period variations are obtained by evaluating the slowly-varying short periodic Fourier coefficients on the mean element integration grid and interpolating to the desired output time. The short periodic variations are then added to the mean elements to obtain the osculating elements (24).

The accuracy of DSST comes from its extensive treatment of perturbing forces. The theory includes third-body models and nonspherical Earth gravitational force models. Included in the nonspherical gravitational model are zonal harmonics, tesseral harmonics, combined zonal and tesseral harmonics, resonant tesseral harmonics, and

second order terms such as J_2^2 and J_2/m-dailies (15). Figure 2.1 taken from reference (25) depicts the spherical harmonic coefficients of degree 8 and orders 6-8 in terms of the zonal, tesseral, and sectoral deviations from a sphere. The deviations are exaggerated to illustrate the shaping. DSST is currently capable of modeling central-body spherical harmonics up to degree and order 50. This capability was introduced by Dan Fonte (26).

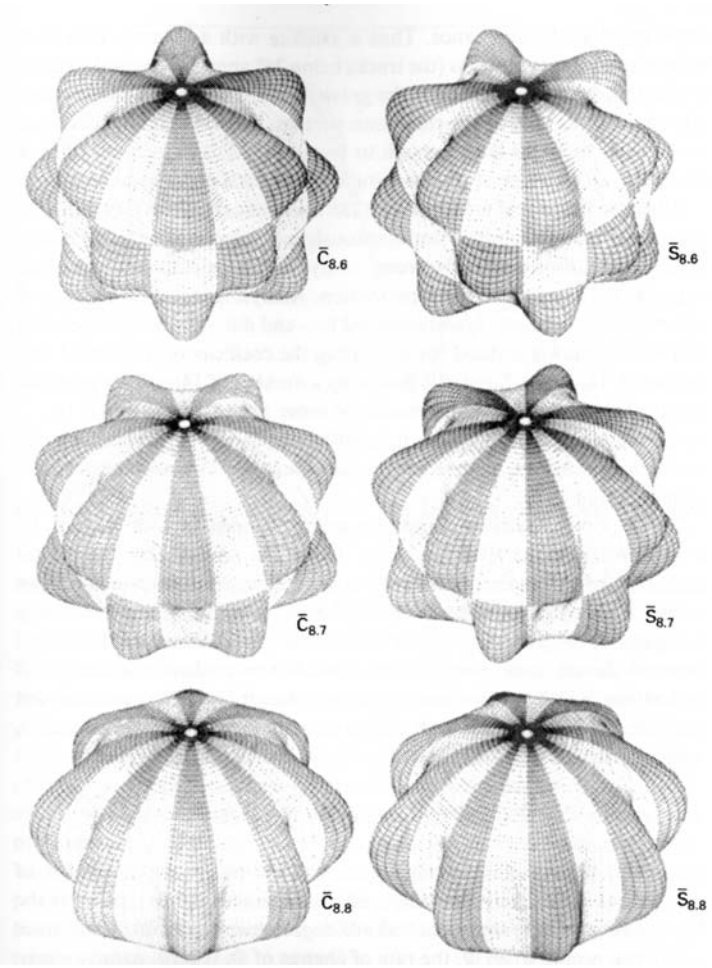

Figure 2.1 Illustration of Zonal, Sectoral and Tesseral Harmonics

The attractions of DSST include its computational efficiency and its high accuracy. In addition, the fact that single revolution, short-period oscillations are not present in the

mean equinoctial solve-for elements enhances the visibility of long-period and secular satellite motion.

2.1.3.1 Perturbing Forces and Variation of Parameters

In order to gain an understanding of the efficiency of the Draper Semianalytic Satellite Theory, it is helpful to summarize its derivation from basic principles. This summary is borrowed from references (11), (15), (10), and (27). The two-body equations of motion for the satellite orbiting the Earth are:

$$\ddot{\mathbf{r}} + \mu \frac{\mathbf{r}}{r^3} = 0 \tag{2.2}$$

where $\mathbf{r}$ is the position vector of the satellite relative to the center of mass and μ is the gravitational parameter. The position vector solution is $\mathbf{r} = \mathbf{f}(c_1, c_2, ..., c_6, t)$. The $c_1, ..., c_6$ constants are the constants of integration for the solution. When perturbing forces are introduced, the equation becomes:

$$\ddot{\mathbf{r}} + \mu \frac{\mathbf{r}}{r^3} = Q(\mathbf{r}, \dot{\mathbf{r}}, t) \tag{2.3}$$

where $Q(\mathbf{r}, \dot{\mathbf{r}}, t)$ is an acceleration vector depending on the position, its derivative with respect to time, and time. The constants, $c_1, c_2, ..., c_6$, are replaced by time varying parameters, $a_k = a_k(t), k = 1,2,...,6$. Velocity is given by the time derivative of position. The chain rule is applied to yield:

$$\begin{aligned} \dot{\mathbf{r}} &= \frac{d}{dt}\mathbf{r}(a_1, a_2, ..., a_6, t) \\ &= \frac{\partial \mathbf{r}}{\partial t} + \sum_{j=1}^{6} \frac{\partial \mathbf{r}}{\partial a_j} \dot{a}_j \end{aligned} \tag{2.4}$$

The second term is equated to zero for convenience to apply the constraint known as the *condition of osculation*:

$$\sum_{j=1}^{6} \frac{\partial \mathbf{r}}{\partial a_j} \dot{a}_j \equiv \Phi(a_1,...,a_6,t) = 0 \tag{2.5}$$

This constraint is also known as the *Lagrange constraint.* This is not the only choice of constraint as the arbitrary function, $\Phi(a_1,...,a_6,t)$, might not be equal to zero. Equating the second term with zero is often done for convenience (28). If one explores the *gauge freedom* of this Lagrangian constraint as is done by Efroimsky in references (28) and (29), the function, $\Phi(a_1,...,a_6,t)$, is arbitrary and this arbitrariness parallels gauge invariance in electrodynamics. A careful choice of $\Phi(a_1,...,a_6,t)$ may considerably simplify the process of finding the solution (29). With the constraint imposed by equation (2.5), the osculating orbit is the unperturbed orbit that is tangent to the perturbed orbit at a given time instant. If at a particular time instant, the effects embodied in the time varying parameters, a_k, cease to exercise any influence on the motion, the resulting orbit would be a conic and the position and velocity vectors would be exactly computable from the two-body formulas (11). This requirement defines the condition for the osculating orbit (14) and maintains the general form of the velocity obtained in the unperturbed problem (29):

$$\dot{\mathbf{r}} = \frac{\partial \mathbf{r}}{\partial t} = \frac{\partial \mathbf{f}}{\partial t} = \mathbf{g}(a_1, a_2,..., a_6, t) + \Phi(a_1, a_2,..., a_6, t) \tag{2.6}$$

Differentiating the above expression and applying the chain rule yields (29):

$$\ddot{\mathbf{r}} = \frac{\partial^2 \mathbf{f}}{\partial t^2} + \sum_{k=1}^{6} \frac{\partial^2 \mathbf{f}}{\partial a_k \partial t} \dot{a}_k + \frac{d\Phi}{dt} = \frac{\partial^2 \mathbf{r}}{\partial t^2} + \sum_{k=1}^{6} \frac{\partial \dot{\mathbf{r}}}{\partial a_k} \dot{a}_k + \frac{d\Phi}{dt} \tag{2.7}$$

When this equation for $\ddot{\mathbf{r}}$ is substituted into the original two-body equations of motion, one obtains (29):

$$\frac{\partial^2 \mathbf{r}}{\partial t^2} + \sum_{k=1}^{6} \frac{\partial \dot{\mathbf{r}}}{\partial a_k} \dot{a}_k + \frac{d\Phi}{dt} + \mu \frac{\mathbf{r}}{r^3} = Q(\mathbf{r}, \dot{\mathbf{r}}, t) \tag{2.8}$$

We recall the equation of motion for the two-body problem:

$$\ddot{\mathbf{r}} + \mu \frac{\mathbf{r}}{r^3} = 0 \tag{2.9}$$

When substituting equation (2.9) into equation (2.8), the perturbing acceleration vector is shown to be (29):

$$\sum_{k=1}^{6} \frac{\partial \dot{\mathbf{r}}}{\partial a_k} \dot{a}_k + \frac{d\Phi}{dt} = Q(\mathbf{r}, \dot{\mathbf{r}}, t) \tag{2.10}$$

If using the constraint from equation (2.5), the $d\Phi/dt$ term is zero. Equation (2.10) shows that the equations of motion for the perturbed system and those for the unperturbed system only differ by the terms containing the time dependent parameters. This last result intuitively indicates that as the perturbing acceleration diminishes, the perturbed equations approach the equations for the unperturbed system.

The variation of parameters (VOP) concept is based on the assumption that perturbing forces are several orders of magnitude smaller than the two-body point mass force. Examples of perturbing forces are the Earth's oblateness, third body gravitational attraction from the moon and sun, solar radiation pressure, and Earth atmospheric drag. The constants of integration produced when solving the two-body differential equations of motion must be made time varying. This changes the solution vector, $\mathbf{r}$, to

$\mathbf{r} = f(a_1, a_2, ..., a_6, t)$ where the a_k are time varying counterpart parameters related to the c_k constants of integration.

In the last result, the partial derivative summation includes partial derivatives with respect to time in $\dot{\mathbf{r}}$ and $\dot{a}_k$. There are also partial derivatives of the velocity, $\dot{\mathbf{r}}$, with respect to the time varying parameters, a_k. The time varying parameters, a_k, represent a specific orbital ellipse at each instant in time. The actual perturbed orbit is represented by a more complex curve, but the orbital ellipse represented by the a_k parameters is tangent to the more complex curve at the time instant for which it is valid. The a_k parameters or elements are thus referred to as osculating in that they are tangent or "kiss" the perturbed orbit at an instant of time.

It is more convenient to use equations of motion of the form:

$$\frac{da_i}{dt} = G_i \tag{2.11}$$

This is an alternative to the form used in the previously shown equation for Q which is a linear combination of orbital element rates. Therefore, the time derivatives of the time varying parameters, a_i, will be sought.

2.1.3.2 Gaussian VOP Formulation

There are several ways in which to formulate the VOP equations. Here, the Gaussian and Lagrange formulations will be outlined. These formulations are taken from

(15). The Gaussian form expresses orbital element rates, G_i, in terms of perturbing forces. The Lagrange form expresses G_i in terms of a potential function. The Gaussian form is obtained by forming dot products of equations (2.10) and (2.5) (15):

$$\begin{aligned}&\frac{\partial a_j}{\partial \dot{\mathbf{r}}} \cdot \left\{ \sum_{k=1}^{6} \frac{\partial \dot{\mathbf{r}}}{\partial a_k} \dot{a}_k \right\} + \frac{\partial a_j}{\partial \mathbf{r}} \cdot \left\{ \sum_{j=1}^{6} \frac{\partial \mathbf{r}}{\partial a_j} \dot{a}_j \right\} \\ &= \sum_{k=1}^{6} \left(\frac{\partial a_j}{\partial \dot{\mathbf{r}}} \cdot \frac{\partial \dot{\mathbf{r}}}{\partial a_k} + \frac{\partial a_j}{\partial \mathbf{r}} \cdot \frac{\partial \mathbf{r}}{\partial a_k} \right) \dot{a}_k = \frac{\partial a_j}{\partial \dot{\mathbf{r}}} \cdot Q(\mathbf{r}, \dot{\mathbf{r}}, t) \qquad j = 1,2,...,6\end{aligned} \tag{2.12}$$

The elements, a_j, are mutually independent and are only functions of position and velocity. This means that (2.12) reduces to our final form of the Gaussian VOP equations (15):

$$\sum_{k=1}^{6} \delta_{j,k} \dot{a}_k = \dot{a}_j = G_j = \frac{\partial a_j}{\partial \dot{\mathbf{r}}} \cdot Q(\mathbf{r}, \dot{\mathbf{r}}, t) \qquad j = 1,2,...,6 \tag{2.13}$$

The Gaussian VOP equations allow both conservative and nonconservative perturbations. The G_i function from equation (2.11) can be derived using partial derivatives of the perturbations, Q. This produces closed-form expressions for the equations of motion. However, because Q is a function of position and velocity, these quantities must be calculated whenever the rates are evaluated. This evaluation is done at each integration time step if the G_i functions are to be used in an ODE solver. The isolation of periodic frequencies is done in the development of the Draper Semianalytic Satellite Theory (DSST) through the Generalized Method of Averaging (GMA). Also, because many acceleration models involve functions of position and velocity instead of Fourier series, the isolation of particular periodic frequencies in the perturbed motion must be done numerically rather than by inspection.

2.1.3.3 Lagrangian VOP Formulation

The VOP formulation developed by Lagrange was designed specifically to deal with planetary orbital perturbations caused by gravitational force from other planets. This formulation only allows for modeling perturbations caused by conservative forces, i.e. the disturbing acceleration can be modeled as the gradient of a potential function (15):

$$Q(\mathbf{r}) = \frac{\partial R(\mathbf{r})}{\partial \mathbf{r}} \tag{2.14}$$

The *R* function is called the disturbing function. The Lagrangian VOP form can be derived through the following sequence involving partials of equations (2.10) and (2.5):

$$\begin{aligned} &\frac{\partial \mathbf{r}}{\partial a_j} \cdot \left\{ \sum_{k=1}^{6} \frac{\partial \dot{\mathbf{r}}}{\partial a_k} \dot{a}_k \right\} - \frac{\partial \dot{\mathbf{r}}}{\partial a_j} \cdot \left\{ \sum_{j=1}^{6} \frac{\partial \mathbf{r}}{\partial a_j} \dot{a}_j \right\} \\ &= \sum_{k=1}^{6} \left(\frac{\partial \mathbf{r}}{\partial a_j} \cdot \frac{\partial \dot{\mathbf{r}}}{\partial a_k} - \frac{\partial \dot{\mathbf{r}}}{\partial a_j} \cdot \frac{\partial \mathbf{r}}{\partial a_k} \right) \dot{a}_k = \frac{\partial R}{\partial \mathbf{r}} \cdot \frac{\partial \mathbf{r}}{\partial a_j} \qquad j = 1,2,...,6 \end{aligned} \tag{2.15}$$

Introduce the *Lagrange bracket* which equates the expression in the parenthesis in (2.15) to $[a_j, a_k]$ and simplify (2.15):

$$\left(\frac{\partial \mathbf{r}}{\partial a_j} \cdot \frac{\partial \dot{\mathbf{r}}}{\partial a_k} - \frac{\partial \dot{\mathbf{r}}}{\partial a_j} \cdot \frac{\partial \mathbf{r}}{\partial a_k} \right) = [a_j, a_k] \tag{2.16}$$

$$\sum_{k=1}^{6} [a_j, a_k] \dot{a}_k = \frac{\partial R}{\partial a_j} \qquad j = 1,2,...,6 \tag{2.17}$$

The indices *j* and *k* both vary from 1 to 6 and so this produces 36 different Lagrange brackets. However, examination of the definition of the Lagrangian bracket shows the following relations (15):

$$[a_j, a_j] = 0 \tag{2.18a}$$

$$[a_j, a_k] = -[a_k, a_j] \tag{2.18b}$$

One can define a matrix **L** to be a matrix containing all the Lagrange brackets as follows:

$$\mathbf{L} = \begin{bmatrix} [a_1, a_1] & \cdots & [a_1, a_6] \\ \vdots & \ddots & \vdots \\ [a_6, a_1] & \cdots & [a_6, a_6] \end{bmatrix} \tag{2.19}$$

Because of (2.18a) and (2.18b) the diagonal terms of **L** are zero and the off diagonal terms have only 15 distinct values. The partials of the potential functions with respect to the elements can now be written as (15):

$$\mathbf{L}\,\dot{\mathbf{a}} = \frac{\partial R}{\partial \mathbf{a}} \tag{2.20}$$

From the definition of the Lagrange brackets in (2.16), it is apparent that the Lagrange brackets only depend on the functional relationship between the orbital elements and the position and velocity for the two-body problem. Therefore, the Lagrange brackets only depend on the elliptical formulae describing the two-body problem. A consequence of this is that the Lagrange brackets are independent of time:

$$\frac{\partial [a_j, a_k]}{\partial t} = 0 \tag{2.21}$$

Proof of this statement is shown in reference (15). This time independence means that the Lagrange brackets can be evaluated at any time in the two-body orbit. This is useful when specific, advantageous points in the orbit are calculated (15).

Another useful construct used in the DSST formulation is the inverse of the Lagrange bracket, i.e. the *Poisson bracket*. It is defined in references (15) and (28) as:

$$\left(a_k, a_j\right) = \frac{\partial a_k}{\partial \mathbf{r}} \cdot \frac{\partial a_j}{\partial \dot{\mathbf{r}}} - \frac{\partial a_k}{\partial \dot{\mathbf{r}}} \cdot \frac{\partial a_j}{\partial \mathbf{r}} \tag{2.22}$$

The Poisson brackets also have the properties of the Lagrange brackets shown in (2.18). The relationship between the matrix of Lagrange brackets and the matrix of Poisson brackets is (15):

$$\mathbf{L}\mathbf{P}^{\mathbf{T}} = \mathbf{I} \tag{2.23a}$$

$$\mathbf{L}^{-1} = \mathbf{P}^{T} = -\mathbf{P} \tag{2.23b}$$

2.1.3.4 DSST Formulation for VOP Equations

There is an alternate derivation of the Lagrange Planetary Equations that turns out to be more useful, this derivation involves Poisson brackets as opposed to Lagrange brackets. The following sequence from reference (15) outlines the derivation. The derivation relies on the following relation from reference (30) which uses the Poisson bracket defined in equation (2.22).

$$\frac{\partial a_k}{\partial \dot{\mathbf{r}}} = -\sum_{j=1}^{6} \left(a_k, a_j\right) \frac{\partial \mathbf{r}}{\partial a_j} \tag{2.24}$$

Substituting Q from equation (2.13) into the Gaussian VOP equation (2.13) yields the Gaussian form of the VOP equations using the potential or disturbing function (15):

$$\dot{a}_k = \frac{\partial a_k}{\partial \dot{\mathbf{r}}} \cdot \frac{\partial R}{\partial \mathbf{r}} \qquad k = 1,2,...,6 \tag{2.25}$$

Substituting equation (2.22) into equation (2.25) and simplifying yields:

$$\dot{a}_k = -\sum_{j=1}^{6}(a_k, a_j)\frac{\partial \mathbf{r}}{\partial a_j}\cdot\frac{\partial R}{\partial \mathbf{r}} = -\sum_{j=1}^{6}(a_k, a_j)\frac{\partial R}{\partial a_j} \qquad k = 1,2,...,6 \qquad (2.26)$$

This can also be expressed in matrix notation yielding the Poisson bracket representation of the Lagrange Planetary Equations (15):

$$\dot{\mathbf{a}} = -\mathbf{P}\frac{\partial R}{\partial \mathbf{a}} \qquad (2.27)$$

The DSST development uses a modified form of the Lagrange Planetary Equations from (2.27). In equation 2.28, the Gaussian VOP terms including the Q function have been added to allow for non-conservative forces. (15):

$$\frac{da_i}{dt} = -\sum_{j=1}^{6}\left[(a_i, a_j)\frac{\partial R}{\partial a_j}\right] + \frac{\partial a_j}{\partial \dot{\mathbf{r}}}\cdot Q(\mathbf{r}, \dot{\mathbf{r}}, t) \qquad i = 1,2,...,5 \qquad (2.28a)$$

$$\frac{dl}{dt} = n - \sum_{j=1}^{6}\left[(l, a_j)\frac{\partial R}{\partial a_j}\right] + \frac{\partial a_j}{\partial \dot{\mathbf{r}}}\cdot Q(\mathbf{r}, \dot{\mathbf{r}}, t) \qquad (2.28b)$$

Here, n is the mean motion and is generally defined for two-body dynamics by: $n^2 a^3 = \mu$. The semimajor axis is a and the gravitational constant is μ. The new variable, l, is defined by:

$$l = nt + a_6 \qquad (2.29)$$

Here, a_6 is the fast element in the vector of orbital elements, **a**. The modification from equation (2.27) to equation (2.28) is used in order to prevent the subtraction of two large secular, i.e. non-periodic, values in the computations of the VOP equations. The avoidance of the addition and subtraction of large numbers makes DSST more computationally stable (15).

2.1.3.5 Equinoctial Orbital Elements and Variational Equations

The elements, a_k , can be represented in many ways including position and velocity, Keplerian elements, equinoctial elements, etc. The set of elements chosen for DSST is a nonsingular equinoctial set. There are computational advantages for choosing the following set of equinoctial elements in that the conversion to position and velocity is computationally inexpensive. Also, the partial derivatives of the equations of motion for these equinoctial elements are also nonsingular (31). The elements $\mathbf{a} = (a, h, k, p, q, \lambda)$ are shown here in terms of the Keplerian elements:

$$
\begin{aligned}
a &= a \\
h &= e\sin(\omega + I\Omega) \\
k &= e\cos(\omega + I\Omega) \\
p &= \tan^I\left(\frac{i}{2}\right)\sin(\Omega) \\
q &= \tan^I\left(\frac{i}{2}\right)\cos(\Omega) \\
\lambda &= M + \omega + I\Omega
\end{aligned}
\qquad
I = \begin{cases} +1 & 0 \le i < \pi \\ -1 & 0 < i \le \pi \end{cases}
\tag{2.30a}
$$

$$
\begin{aligned}
e &= \sqrt{(h^2 + k^2)} \\
i &= 2\tan^{-1}\left(\sqrt{p^2 + q^2}\right) \\
\Omega &= \tan^{-1}\left(p/q\right) \\
\omega &= \tan^{-1}\left(h/k\right) - \tan^{-1}\left(p/q\right) \\
M &= \lambda - \tan^{-1}\left(h/k\right)
\end{aligned}
\tag{2.30b}
$$

where a is the semimajor axis, e is the eccentricity, i is the inclination, ω is the argument of perigee, Ω is the right ascension of the ascending node, M is the mean anomaly and I is the retrograde factor. When the retrograde factor is -1, the $\tan^I$ function becomes the cotangent function.

The direct equinoctial reference frame is shown in Figure 2.2. The unit vectors, $\hat{\mathbf{f}}, \hat{\mathbf{g}}, \hat{\mathbf{w}}$, are aligned so that $\hat{\mathbf{f}}$ and $\hat{\mathbf{g}}$ are contained in the instantaneous orbit plane with the direction of $\hat{\mathbf{f}}$ obtained through a clockwise rotation of the angle, Ω, from the direction of the ascending node (3).

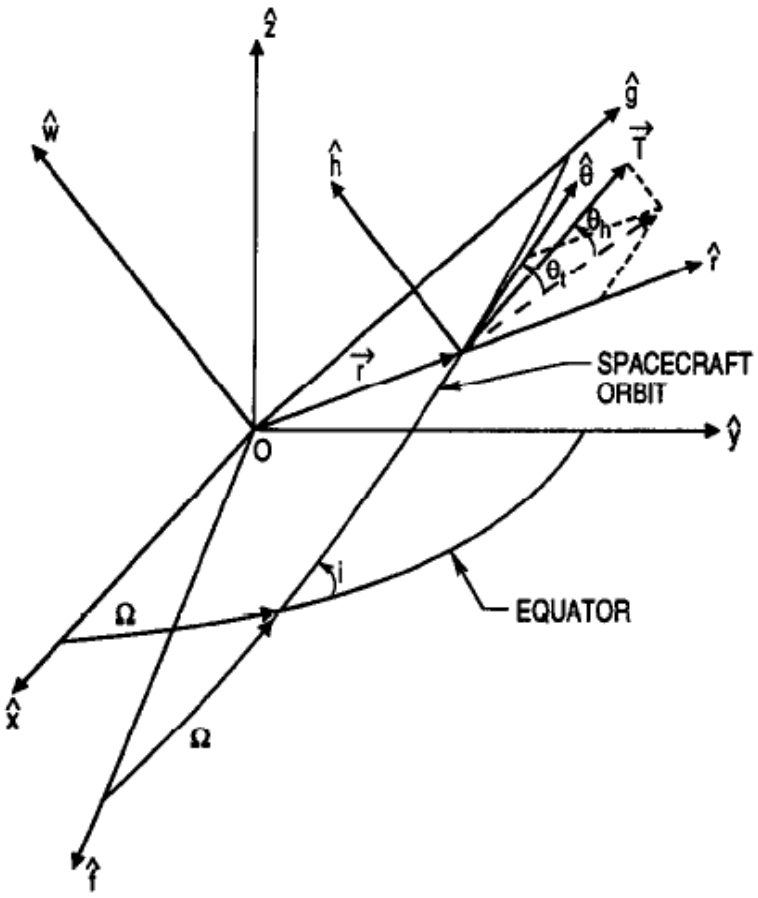

Figure 2.2 Equinoctial Orbital Element Frame (3)

At this point it is sensible to show the variational equations for the osculating equinoctial elements. These are taken from the *GTDS Mathematical Specification* (32):

$$\frac{da}{dt} = \frac{2\mathbf{v}}{n^2 a} \cdot \mathbf{a}_d \tag{2.31a}$$

$$\frac{dh}{dt} = \left\{ \frac{1}{\mu} \left[\left[2\dot{X}_1 Y_1 - X_1 \dot{Y}_1 \right] \hat{f} - X_1 \dot{X}_1 \hat{g} \right] + \frac{k}{G} (qIY_1 - pX_1) \hat{w} \right\} \cdot \mathbf{a}_d \tag{2.31b}$$

$$\frac{dk}{dt} = \left\{ -\frac{1}{\mu} \left[Y_1 \dot{Y}_1 \hat{f} - (2X_1 \dot{Y}_1 - \dot{X}_1 Y_1) \hat{g} \right] - \frac{h}{G} (qIY_1 - pX_1) \hat{w} \right\} \cdot \mathbf{a}_d \tag{2.31c}$$

$$\frac{dp}{dt} = \left\{ \frac{Y_1}{2G} (1 + p^2 + q^2) \hat{w} \right\} \cdot \mathbf{a}_d \tag{2.31d}$$

$$\frac{dq}{dt} = \left\{ \frac{IX_1}{2G} (1 + p^2 + q^2) \hat{w} \right\} \cdot \mathbf{a}_d \tag{2.31e}$$

$$\frac{d\lambda}{dt} = \left\{ n - \frac{2}{na^3} \mathbf{r} + \beta \left(k \frac{\partial h}{\partial \mathbf{v}} - h \frac{\partial k}{\partial \mathbf{v}} \right) + \frac{1}{na^2} \left(qIY_1 - pX_1 \right) \hat{w} \right\} \cdot \mathbf{a}_d \tag{2.31f}$$

The $\hat{f}, \hat{g}, \hat{w}$ unit vectors are in the equinoctial reference frame, **r** and **v** are Cartesian coordinates in the inertial reference frame, *I* is the retrograde factor, and *n* is the mean motion. The perturbing acceleration vector, $\mathbf{a}_d$, is a Cartesian vector. The position and velocity in the equinoctial reference frame within the orbit plane are:

$$X_1 = a\left[\left(1 - h^2 \beta \right) \cos F + hk\beta \sin F - k \right] \tag{2.32a}$$

$$Y_1 = a\left[\left(1 - k^2 \beta \right) \sin F + hk\beta \cos F - h \right] \tag{2.32b}$$

$$\dot{X}_1 = \frac{na^2}{r}\left[hk\beta\cos F - (1-h^2\beta)\sin F\right] \tag{2.32c}$$

$$\dot{Y}_1 = \frac{na^2}{r}\left[(1-k^2\beta)\cos F - hk\beta\sin F\right] \tag{2.32d}$$

The new variable F is the eccentric longitude and can be solved using Kepler's transcendental equation:

$$\lambda = F - k\sin F + h\cos F \tag{2.33}$$

The variables β and G are defined by:

$$G = na^2\sqrt{1-h^2-k^2} \tag{2.34}$$

$$\beta = \frac{1}{1+\sqrt{1-h^2-k^2}} \tag{2.35}$$

Equations in (2.31) can be integrated forward or backward in time using an integrator in Matlab, for example. Equations in (2.31) include two-body motion and accelerating perturbations through the $\mathbf{a}_d$ vector. Equations in (2.31) are not the final form used by DSST because the $\mathbf{a}_d$ vector is in a general form and no specific perturbing forces have been introduced yet. DSST includes several perturbing forces. The short period and long period motion produced by these perturbations are separated using the Generalized Method of Averaging.

2.1.3.6 DSST Application of Generalized Method of Averaging

Lagrange developed and used the Lagrange Planetary Equations to investigate the long period and secular motion of the planets. He expanded the disturbing function in a literal Fourier series (15). Because a Fourier series was used, the terms in the equation associated with first-order long period and secular motion were isolated by inspection. These terms were then used to predict planetary motion with excellent results because the perturbations were small. However, higher order solutions are needed when the perturbations are relatively large, or when the solution must predict motion more accurately for longer time spans (15). In these cases, the Generalized Method of Averaging (GMA) can be applied to the VOP equations of motion.

When applying the GMA to either the Gaussian or Lagrangian VOP equations of motion, the short-period terms are isolated[1] from the long-period and secular terms and one is left with averaged equations of motion in terms of time varying parameters. The time varying parameters are called mean elements because the short period motion is no longer represented in the solution. There are several ways to define mean elements. The exact definition depends on the time interval over which the equations of motion are averaged (15).

[1] It should be noted that the GMA operations can be done easily for the first order terms. Higher order terms cannot be easily isolated in this way.

In general the VOP element rates can be represented by:

$$\frac{da_i}{dt} = \varepsilon F_i(\mathbf{a}, l) \qquad i = 1,2,...,5 \tag{2.36a}$$

$$\frac{dl}{dt} = n(a_1) + \varepsilon F_6(\mathbf{a}, l) \tag{2.36b}$$

where **a** represents the vector of five mean elements and l is the fast variable. The function, F, is the perturbing function which is 2π periodic in the variable l (15). When applying the Generalized Method of Averaging to these VOP equations, the short period terms must be defined. This definition will constrain the integration step size that can ultimately be used with the averaged VOP equations of motion. DSST was developed to maximize the integration step size to provide an efficient, yet accurate representation of the satellite motion. In DSST, the short period terms are defined as those with a period on the order of one orbital revolution or less of the satellite (15). These have been referred to as *single averaged* theories. This means that all terms dependent on multiples of the fast variable, l, are considered short period. Terms introduced by the fast variables in third bodies such as the sun and moon as well as Greenwich Hour Angle dependent terms in the Earth's spherical gravity harmonic expansion can also be considered short-period. The terms exclusively dependent on the other five slowly-varying orbital elements are considered long-period or secular terms (15).

Among the short period terms, the following summarizes some of the larger forces contributing perturbations to satellite motion. *Zonal harmonics* are the latitude dependent terms in the Earth's harmonic expansion. This includes the largest zonal term,

J_2, which is also much larger than all other zonal short-periodic perturbation contributions (15).

Tesseral harmonics model the non-spherical gravitational effects of the Earth by dividing the Earth's field into intersections of the zonal and sectoral terms. Because this array of roughly rectangular regions is fixed to the Earth's surface, i.e. the tesseral terms are dependent on the Greenwich Hour Angle, the Earth's rotation contributes short period effects to a satellite's orbit.

In addition, *tesseral m-daily* terms have periods on the order of one day, i.e. one rotation of the Earth. Reference (33) describes motion contributions from tesseral resonance and from so called *earth rotation terms*. These are terms that are linear combinations of the satellite fast variable, l, and the Greenwich Hour Angle. In analytic and semianalytic formulations, the terms dependent on the earth rotation are often on the order of the satellite period and need to be separated from the longer period terms (32). Also, terms that combine the J_2 and tesseral m-daily terms add short period motion to the equations of motion and require separation from the long period motion (15).

Third body gravitational effects also add short period motion to the equations of motion. For Earth orbiting satellites, the sun and moon contribute terms with periods on the order of one year and 28 days, respectively. These effects can be considered long-period if using GMA to average over periods of approximately on satellite orbit, i.e. single averaging. Although not used in DSST, some satellite motion theories use *double*

averaging. These double averaged theories average over the satellite's fast variable and also the fast variables of third bodies (15).

To develop the formulation of the Draper Semianalytic Satellite Theory (DSST), the *near-identity* transformation is introduced to express the osculating elements in terms of single averaged mean elements (15). The following form for the osculating orbital elements is assumed:

$$a_i = \bar{a}_i + \sum_{j=1}^{N} \varepsilon^j \eta_{i,j}(\bar{\mathbf{a}}, \bar{l}) + O(\varepsilon^{N+1}) \qquad i = 1,2,...,5 \tag{2.37a}$$

$$l = \bar{l} + \sum_{j=1}^{N} \varepsilon^j \eta_{6,j}(\bar{\mathbf{a}}, \bar{l}) + O(\varepsilon^{N+1}) \tag{2.37b}$$

where $\eta_{i,j}$ represents the short-periodic variation of order j in element $\bar{a}_i$, the $\bar{a}_i$ are the slowly-varying mean elements, $\bar{l}$ is the *mean* mean longitude, and the quantity ε is the small parameter in the perturbation model.

The assumed form of the transform of the equations of motion for the mean elements in equation (2.36) is:

$$\frac{d\bar{a}_i}{dt} = \sum_{j=1}^{N} \varepsilon^j A_{i,j}(\bar{\mathbf{a}}) + O(\varepsilon^{N+1}) \qquad i = 1,2,...,5 \tag{2.38a}$$

$$\frac{d\bar{l}}{dt} = n(\bar{a}_i) + \sum_{j=1}^{N} \varepsilon^j A_{6,j}(\bar{\mathbf{a}}) + O(\varepsilon^{N+1}) \tag{2.38b}$$

where $n(\bar{a}_i)$ is the *mean* mean motion (14). For this assumed transform, the rate of change in the mean elements only depends on the slowly-varying mean elements (15).

The expressions for the short-periodic variations, $\eta_{i,j}$, and the functions of the slowly varying mean elements, $A_{i,j}$, are now sought. Differentiate the osculating elements (2.37) to obtain expressions for the osculating element rates:

$$\frac{da_i}{dt} = \frac{d\overline{a}_i}{dt} + \sum_{j=1}^{N} \varepsilon^j \sum_{k=1}^{6} \frac{\partial \eta_{i,j}}{\partial \overline{a}_k} \frac{d\overline{a}_k}{dt} + O(\varepsilon^{N+1}) \qquad i = 1,2,...,5 \tag{2.39a}$$

$$\frac{dl}{dt} = \frac{d\bar{l}}{dt} + \sum_{j=1}^{N} \varepsilon^j \sum_{k=1}^{6} \frac{\partial \eta_{6,j}}{\partial \overline{a}_k} \frac{d\overline{a}_k}{dt} + O(\varepsilon^{N+1}) \tag{2.39b}$$

Substitute the mean element rates (2.38) into the osculating element rates (2.39). This introduces $A_{i,j}$ into the osculating element equations of motion (15).

$$\frac{da_i}{dt} = \sum_{j=1}^{N} \varepsilon^j \left[A_{i,j}(\overline{\mathbf{a}}) + n(\overline{a}_1) \frac{\partial \eta_{i,j}}{\partial \bar{l}} + \sum_{m=1}^{N-j} \varepsilon^m \sum_{k=1}^{6} A_{k,m} \frac{\partial \eta_{i,j}}{\partial \overline{a}_k} \right] \tag{2.40a}$$
$$+ O(\varepsilon^{N+1}) \quad i = 1,2,...,5$$

$$\frac{dl}{dt} = n(\overline{a}_i) + \sum_{j=1}^{N} \varepsilon^j \left[A_{6,j} + n(\overline{a}_i) \frac{\partial \eta_{6,j}}{\partial \bar{l}} + \sum_{m=1}^{N-j} \varepsilon^m \sum_{k=1}^{6} A_{k,m} \frac{\partial \eta_{6,j}}{\partial \overline{a}_k} \right] \tag{2.40b}$$
$$+ O(\varepsilon^{N+1})$$

Rearrange the equations to obtain a single summation over ε (15):

$$\frac{da_i}{dt} = \sum_{j=1}^{N} \varepsilon^j \left[A_{i,j}(\overline{\mathbf{a}}) + n(\overline{a}_i) \frac{\partial \eta_{i,j}}{\partial \bar{l}} + \sum_{k=1}^{6} \sum_{p=1}^{j-1} A_{k,p} \frac{\partial \eta_{i,j-p}}{\partial \overline{a}_k} \right] \tag{2.41a}$$
$$+ O(\varepsilon^{N+1}) \qquad i = 1,2,...,5$$

$$\frac{dl}{dt} = n(\overline{a}_1) + \sum_{j=1}^{N} \varepsilon^j \left[A_{6,j}(\overline{\mathbf{a}}) + n(\overline{a}_i) \frac{\partial \eta_{6,j}}{\partial \bar{l}} + \sum_{k=1}^{6} \sum_{p=1}^{j-1} A_{k,p} \frac{\partial \eta_{6,j-p}}{\partial \overline{a}_k} \right] + O(\varepsilon^{N+1}) \tag{2.41b}$$

Now, expand the perturbing functions in the VOP equations (2.36) about the mean elements using a Taylor series:

$$F_i(\mathbf{a},l) = \sum_{n=0}^{\infty} \frac{1}{n!} \left(\sum_{k=1}^{6} \Delta a_k \frac{\partial}{\partial a_k} \right)^n F_i(\mathbf{a},l) \Bigg|_{\substack{\mathbf{a}=\bar{\mathbf{a}} \\ l=\bar{l}}} \tag{2.42}$$

where $\Delta a_k = a_k - \bar{a}_k$ and are defined by (2.37).

Define $\frac{\partial}{\partial \bar{a}_k} = \frac{\partial}{\partial a_k}\Big|_{a=\bar{a}_k}$ and rearrange the equation for F_i as a power series in ε to obtain (15):

$$F_i(\mathbf{a},l) = \sum_{j=0}^{N} \varepsilon^j f_{i,j}(\bar{\mathbf{a}},\bar{l}) + O(\varepsilon^{N+1}) \tag{2.43}$$

where

$$f_{i,0}(\bar{\mathbf{a}},\bar{l}) = F_i(\mathbf{a},l) \tag{2.44a}$$

$$f_{i,1}(\bar{\mathbf{a}},\bar{l}) = \sum_{k-1}^{6} \eta_{k,1} \frac{\partial F_i}{\partial \bar{a}_k} \tag{2.44b}$$

$$f_{i,2}(\bar{\mathbf{a}},\bar{l}) = \sum_{k=1}^{6} \left(\eta_{k,2} \frac{\partial F_i}{\partial \bar{a}_k} + \frac{1}{2} \sum_{l=1}^{6} \eta_{k,1}\eta_{l,1} \frac{\partial^2 F_i}{\partial \bar{a}_k \partial \bar{a}_l} \right) \tag{2.44c}$$

⋮

Also, expand the mean motion about the *mean* mean motion in a Taylor series about the mean semimajor axis, $\bar{a}_1$ (15):

$$n(a_1) = \sum_{k=0}^{\infty} \frac{(\Delta a_k)^k}{k!} \frac{\partial^k n}{\partial \bar{a}_1^k} \tag{2.45}$$

Rearrange this equation for *mean* mean motion as a power series in ε to obtain (15):

$$n(a_1) = \sum_{k=0}^{\infty} \varepsilon^k N_k(\bar{\mathbf{a}}, \bar{l}) \tag{2.46}$$

where

$$N_0(\bar{\mathbf{a}}, \bar{l}) = n(a_1) = \bar{n} \tag{2.47a}$$

$$N_1(\bar{\mathbf{a}}, \bar{l}) = -\frac{3}{2} \frac{\bar{n}}{\bar{a}_1} \eta_{1,1} \tag{2.47b}$$

$$N_2(\bar{\mathbf{a}}, \bar{l}) = \frac{15}{4} \frac{\bar{n}}{\bar{a}_1^2} \eta_{1,1}^2 - \frac{3}{2} \frac{\bar{n}}{\bar{a}_1} \eta_{1,2} \tag{2.47c}$$

$\vdots$

Now, substitute the rearranged equations for the osculating element rates (2.41) into the left-hand side of the original equations for the osculating element rates (2.36) in terms of the mean elements. Substitute equations (2.43) and (2.46) into the right hand side of equation (2.36). Both sides are power series expansions in ε and therefore, terms with like powers of ε must be equal. Equate such terms to obtain the j[th]-order contribution to the osculating element rates (15):

$$A_{i,j}(\bar{\mathbf{a}}) + \bar{n}\frac{\partial \eta_{i,j}}{\partial \bar{l}} + \sum_{k=1}^{6}\sum_{p=1}^{j-1} A_{k,p}\frac{\partial \eta_{i,j-p}}{\partial \bar{a}_k} = f_{i,j-1}(\bar{\mathbf{a}},\bar{l}) \qquad i = 1,2,...,5 \qquad (2.48a)$$

$$A_{6,j}(\bar{\mathbf{a}}) + \bar{n}\frac{\partial \eta_{6,j}}{\partial \bar{l}} + \sum_{k=1}^{6}\sum_{p=1}^{j-1} A_{k,p}\frac{\partial \eta_{6,j-p}}{\partial \bar{a}_k} = f_{6,j-1}(\bar{\mathbf{a}},\bar{l}) + N_j(\bar{\mathbf{a}},\bar{l}) \qquad (2.48b)$$

These equations show the relationship between the unknown functions of the slowly-varying mean elements, $A_{i,j}$, and the partial derivatives of the unknown short-periodic variations, $\eta_{i,j}$ to the known terms of the power series expansion of the osculating perturbing function, $f_{i,j}$ (14). Averaging over the mean fast variable, $\bar{l}$, eliminates the dependency on that variable. This takes advantage of the fact that the short periodic variations are 2π periodic in the fast variable, $\bar{l}$ (15).

Integrate both sides of equation (2.48) from 0 to 2π over the mean fast variable, $\bar{l}$. This eliminates the partial derivatives, $\frac{\partial \eta_{i,j}}{\partial \bar{l}}$. This integration is known as the *averaging operation* and is defined as (15):

$$\left\langle H(\bar{\mathbf{a}},\bar{l})\right\rangle_{\bar{l}} = \frac{1}{2\pi}\int_0^{2\pi} H(\bar{\mathbf{a}},\bar{l})\,d\bar{l} \tag{2.49}$$

Because the $\eta_{i,j}$ functions are 2π periodic in $\bar{l}$, the partial derivatives of the $\eta_{i,j}$ functions are also 2π periodic in $\bar{l}$ (15).

$$\left\langle \bar{n}\frac{\partial \eta_{i,j}}{\partial \bar{l}}\right\rangle_{\bar{l}} = 0 \tag{2.50}$$

Applying the averaging operation to equations (2.48) thus yields (15):

$$A_{i,j}(\bar{\mathbf{a}}) = \left\langle f_{i,j-1}(\bar{\mathbf{a}},\bar{l})\right\rangle_{\bar{l}} + \sum_{p=1}^{j-1}\sum_{k=1}^{5} A_{k,p}\left\langle \frac{\partial \eta_{i,j-p}}{\partial \bar{a}_k}\right\rangle_{\bar{l}} \qquad i = 1,2,...,5 \tag{2.51a}$$

$$A_{6,j}(\bar{\mathbf{a}}) = \left\langle f_{6,j-1}(\bar{\mathbf{a}},\bar{l}) + N_j(\bar{\mathbf{a}},\bar{l})\right\rangle_{\bar{l}} + \sum_{p=1}^{j-1}\sum_{k=1}^{5} A_{6,p}\left\langle \frac{\partial \eta_{6,j-p}}{\partial \bar{a}_k}\right\rangle_{\bar{l}} \tag{2.51b}$$

These equations are simplified by imposing a constraint such that the $\eta_{i,j}$ functions do not contain any long-period or secular terms (15):

$$\left\langle \frac{\partial \eta_{i,j-p}}{\partial \bar{a}_k}\right\rangle_{\bar{l}} \equiv 0 \qquad i = 1,2,...,6, \quad k = 1,2,...,6 \tag{2.52}$$

Applying the averaging operation to equation (2.39) for the osculating element rates, and then applying the constraint from equation (2.52), the following equivalences are obtained (15):

$$\left\langle \frac{da_i}{dt} \right\rangle_{\bar{l}} = \frac{d\bar{a}_i}{dt} \qquad i = 1,2,...,5 \tag{2.53a}$$

$$\left\langle \frac{dl}{dt} \right\rangle_{\bar{l}} = \frac{d\bar{l}}{dt} \tag{2.53b}$$

Using equations (2.50) and (2.52) has eliminated the averaged partials of $\eta_{i,j}$. The mean elements are now seen to represent the long-period and secular contributions to the osculating elements plus a constant (15):

$$\left\langle \eta_{i,j}(\bar{\mathbf{a}}, \bar{l}) \right\rangle_{\bar{l}} = C_{i,j} \tag{2.54}$$

Eliminating these constants means a constraint such as the following should be imposed (15):

$$C_{i,j} = 0 \tag{2.55}$$

Now, applying the averaging operation to the equations for the osculating elements yields (15):

$$\left\langle a_i \right\rangle_{\bar{l}} = \bar{a}_i \qquad i = 1,2,...,5 \tag{2.56a}$$

$$\left\langle l \right\rangle_{\bar{l}} = \bar{l} \tag{2.56b}$$

The result of the constraints (2.52) and (2.55) is that the $\eta_{i,j}$ functions contain only short-periodic terms and terms that mix the short-periodic and long-period effects. The equations for $A_{i,j}$ now reduce to (15):

$$A_{i,j}(\bar{\mathbf{a}}) = \left\langle f_{i,j-1}(\bar{\mathbf{a}},\bar{l}) \right\rangle_{\bar{l}} \qquad i = 1,2,...,5 \tag{2.57a}$$

$$A_{6,j}(\bar{\mathbf{a}}) = \left\langle f_{6,j-1}(\bar{\mathbf{a}},\bar{l}) + N_j(\bar{\mathbf{a}},\bar{l}) \right\rangle_{\bar{l}} \tag{2.57b}$$

The averaged equations of motion can now be expressed in terms of the power series expansion of the disturbing function. The *mean* mean motion can be represented by the power series expansion of the osculating mean motion (15). Substituting the equations for $A_{i,j}$ into the original equations of motion for the mean element rates yields (15):

$$\frac{d\bar{a}_i}{dt} = \sum_{j=1}^{N} \varepsilon^j \left\langle f_{i,j-1}(\bar{\mathbf{a}},\bar{l}) \right\rangle_{\bar{l}} + O(\varepsilon^{N+1}) \qquad i = 1,2,...,5 \tag{2.58a}$$

$$\frac{d\bar{l}}{dt} = n(\bar{a}_i) + \sum_{j=1}^{N} \varepsilon^j \left\langle f_{6,j-1}(\bar{\mathbf{a}},\bar{l}) + N_j(\bar{\mathbf{a}},\bar{l}) \right\rangle_{\bar{l}} + O(\varepsilon^{N+1}) \tag{2.58b}$$

Equations for $f_{i,j}$ and N_j, i.e. equations (2.44) and (2.47), show that for j≥1, terms containing the short periodic functions, $\eta_{i,j}$, are still present. Therefore, the averaging operation did not remove all dependence on the short period terms.

To determine the short period functions, $\eta_{i,j}$, subtract equations (2.51) from (2.48) (15):

$$\overline{n}\frac{\partial \eta_{i,j}}{\partial \bar{l}} + \sum_{p=1}^{j-1}\sum_{k=1}^{6} A_{k,p}\left(\frac{\partial \eta_{i,j-p}}{\partial \overline{a}_k} - \left\langle \frac{\partial \eta_{i,j-p}}{\partial \overline{a}_k} \right\rangle_{\bar{l}}\right) = f_{i,j-1} - \left\langle f_{i,j-1}\right\rangle_{\bar{l}} \qquad i = 1,2,...,5 \qquad (2.59a)$$

$$\overline{n}\frac{\partial \eta_{6,j}}{\partial \bar{l}} + \sum_{p=1}^{j-1}\sum_{k=1}^{6} A_{k,p}\left(\frac{\partial \eta_{6,j-p}}{\partial \overline{a}_k} - \left\langle \frac{\partial \eta_{6,j-p}}{\partial \overline{a}_k} \right\rangle_{\bar{l}}\right) = f_{6,j-1} + N_j - \left\langle f_{6,j-1} + N_j\right\rangle_{\bar{l}} \qquad (2.59b)$$

Define the superscript, s, to be the short-periodic part of a function, which gives (15):

$$f^S_{i,j-1} = f_{i,j-1} - \left\langle f_{i,j-1}\right\rangle_{\bar{l}}$$

Now, rewrite the difference equations (2.59) as (15):

$$\overline{n}\frac{\partial \eta_{i,j}}{\partial \bar{l}} = f^S_{i,j-1} - \sum_{p=1}^{j-1}\sum_{k=1}^{6} A_{k,p}\left(\frac{\partial \eta^S_{i,j-p}}{\partial \overline{a}_k}\right) \qquad i = 1,2,...,5 \qquad (2.60a)$$

$$\bar{n}\frac{\partial \eta_{6,j}}{\partial \bar{l}} = f_{6,j-1}^{S} + N_j^{S} - \sum_{p=1}^{j-1}\sum_{k=1}^{6} A_{k,p}\left(\frac{\partial \eta_{6,j-p}^{S}}{\partial \bar{a}_k}\right) \tag{2.60b}$$

The short period variation of order j, represented here by $\eta_{i,j}$ can be seen to be dependent on quantities of lower order than j for mean elements other than, $\bar{l}$. In the case of the fast variable, it can be seen that $\eta_{6,j}$ is dependent on $\eta_{1,j}$ through the term N_j. This means that the function, $\eta_{1,j}$, must be calculated before the function, $\eta_{6,j}$ (15).

Multiplying equations (2.60) by $1/\bar{n}$, and integrating with respect to $\bar{l}$ yields the short-periodic functions to within an arbitrary function of the slowly varying mean elements (15):

$$\eta_{i,j}(\bar{\mathbf{a}},\bar{l}) = \alpha_{i,j}(\bar{\mathbf{a}},\bar{l}) + C_{i,j}(\bar{\mathbf{a}}) \tag{2.61}$$

where $C_{i,j}$ is an arbitrary function of integration, and $\alpha_{i,j}$ is defined as (15):

$$\alpha_{i,j} = \frac{1}{\bar{n}}\int\left[f_{i,j-1}^{S} - \sum_{k=1}^{6}\sum_{p=1}^{j-1} A_{k,p}\frac{\partial \eta_{i,j-p}^{S}}{\partial \bar{a}_k}\right]d\bar{l} \qquad i = 1,2,...,5 \tag{2.62a}$$

$$\alpha_{6,j} = \frac{1}{\bar{n}}\int\left[f_{6,j-1}^{S} - \sum_{k=1}^{6}\sum_{p=1}^{j-1} A_{k,p}\frac{\partial \eta_{6,j-p}^{S}}{\partial \bar{a}_k}\right]d\bar{l} \tag{2.62b}$$

Averaging equation (2.61) yields:

$$\left\langle \eta_{i,j} \right\rangle_{\bar{l}} = C_{i,j}(\bar{\mathbf{a}})$$

which shows that the constraints imposed earlier on equations (2.52) and (2.54) are valid (15). Assuming that $C_{i,j}$ is zero to again apply the constraint of equation (2.55), this requires that the $\eta_{i,j}$ functions contain only short-periodic terms and terms that include both short-periodic and long-period terms (15).

A set of functions, $\eta_{i,j}$, has been obtained that contain only short-periodic terms. The near-identity transformation of equations (2.37) can then be expressed as (15):

$$a_i = \bar{a}_i + \frac{1}{\bar{n}} \sum_{j=1}^{N} \varepsilon^j \int \left[f_{i,j-1}^S - \sum_{k=1}^{6} \sum_{p=1}^{j-1} A_{k,p} \frac{\partial \eta_{i,j-p}^S}{\partial \bar{a}_k} \right] d\bar{l} + O(\varepsilon^{N+1}) \qquad i = 1,2,...,5 \qquad (2.63a)$$

$$l = \bar{l} + \frac{1}{\bar{n}} \sum_{j=1}^{N} \varepsilon^j \int \left[f_{6,j-1}^S + N_j^S - \sum_{k=1}^{6} \sum_{p=1}^{j-1} A_{k,p} \frac{\partial \eta_{6,j-p}^S}{\partial \bar{a}_k} \right] d\bar{l} + O(\varepsilon^{N+1}) \qquad (2.63b)$$

Determination of the jth order contribution to the mean element rates in equation (2.58) and the $\eta_{i,j}$ functions is interdependent must be done on an order by order basis. Reference (15) illustrates this for second order terms in equation (2.58) and for the $\eta_{i,j}$ functions. The first order contributions of the mean element rates, $A_{i,1}$, are independent

of the $\eta_{i,j}$ functions. However, the second order computation of the mean element rates, $A_{i,2}$, must proceed as follows (15):

$$A_{i,1} = \langle F_i(\bar{a}, \bar{l}) \rangle_{\bar{l}} \qquad i = (1,2, \ldots ,6) \tag{2.64a}$$

$$\eta_{i,1} = \frac{1}{\bar{n}} \int F_i^6(\bar{a}, \bar{l}) d\bar{l} \qquad i = (1,2, \ldots ,5) \tag{2.64b}$$

$$\eta_{6,1} = \frac{1}{\bar{n}} \int \left[F_6^5(\bar{a}, \bar{l}) - \frac{3}{2} \frac{\bar{n}}{\bar{a_1}} \eta_{1,1} \right] d\bar{l} \tag{2.64c}$$

$$A_{i,2} = \langle \sum_{k=1}^{6} \eta_{k,1} \frac{\partial F_i}{\partial \bar{a_k}} \rangle_{\bar{l}} \qquad i = (1,2, \ldots ,5) \tag{2.64d}$$

$$A_{6,2} = \langle \sum_{k=1}^{6} \eta_{k,1} \frac{\partial F_6}{\partial \bar{a_k}} + \frac{15}{8} \frac{\bar{n}}{\bar{a_1^2}} \eta_{1,1}^2 \rangle_{\bar{l}} \tag{2.64e}$$

This procedure can be used to extend the averaged equations of motion to higher order (15).

If the appropriate fast variable is used, the first order short-periodic effects can be formulated as closed-form expressions. The zonal short-periodics can be formulated in terms of the true longitude, the tesseral m-dailies can be formulated in terms of the Greenwich Hour Angle, and the third body short-periodics can be formulated in terms of the eccentric longitude (34). The coefficients of the periodic terms in these expressions are slowly-varying because they are only functions of the mean elements. The closed-

form short-periodic motion formulae can be used with an efficient interpolator to solve equations (2.63) for the second order mean element rates when needed.

The McClain reference (15) includes further details which describe how to formulate the averaged equations of motion while including multiple perturbations for each element rate, modeling resonance phenomena, including nonspherical gravitational perturbations, and including third body effects. There are also variants of this basic derivation of the VOP equations. Green (35) extended the VOP equation derivation to include a second order drag theory in the averaged equations of motion. Green also introduced a weak time dependence formulation to deal with disturbing functions having two fast variables. One of the fast variables was the satellite's mean motion while the other was small in comparison to the satellite's mean motion. J_2/m-daily coupling has also been developed and incorporated into the DSST formulation. The development of third body disturbing potentials for the DSST can be found in Collins' Ph.D. Thesis (36).

2.2 Application of Electric Propulsion (EP) to satellites

The central problem addressed in this thesis is to evaluate ways of improving orbit determination for satellites that use continuous thrust, electric propulsion. Continuous implies that the thruster typically operates for extended periods which are significant fractions of the orbital period. This could include extended orbit raising applications or stationkeeping operations. In this section, electric propulsion (EP), optimal ways of using EP to control satellite motion, and ways of modeling optimally controlled continuous thrust will be described. Understanding how to optimally control EP satellites is important so that one can develop ways in which to improve the models of the motion of such satellites. Through these models, it is hoped that the trajectory plans for spacecraft can be anticipated. If these models successfully predict spacecraft thrusting, the improved motion models can be used to improve orbit determination and predictions of satellite motion for space surveillance applications.

2.2.1 Electric propulsion motors

There are several types of electric propulsion engines that have been developed for spacecraft. This thesis will focus on operational electric thrusters that use relatively low levels of thrust, i.e. less than 1 Newton. Electric thrusters that are modified versions of monopropellant or bipropellant chemical thrusters such as resistojets and arcjets are not discussed in this thesis. The spacecraft of focus in this thesis use low-thrust electric propulsion, typically thrust continuously for long periods, and are challenging to model in orbit determination (OD) for space surveillance. Because it is probable that satellite

operators and space surveillance networks do not cooperate or share information, OD for space surveillance is challenging because of the somewhat unpredictable thrusting undergone by active satellites. The types of electric propulsion engines already launched and operating include the Hall thruster, the gridded ion engine, and the Pulsed Plasma Thruster (PPT) (37). The Hall thruster electrostatically accelerates ions, but transmits the thrust to magnets through their magnetic interaction with electrons in a Hall current (38). Gridded ion engines use a charged grid to electrostatically accelerate ions (38). The PPT thruster operates by creating a pulsed, high-current discharge across the exposed surface of a solid insulator. This provides propellant that is ionized, heated and accelerated to high speed (39).

Other types of electric propulsion such as arcjets, resistojets, magneto-plasma devices, field effect electrostatic propulsion (FEEP), and colloid thrusters either have not yet been launched on spacecraft or thrust in ways not unlike traditional chemical thrusters, i.e. short bursts of thrust that can be modeled as impulses. These types of thrusters will not be described in this thesis because the scope here is limited to propulsion technologies that operate for long durations with relatively low levels of thrust, i.e. typically much less than one Newton.

Hall thrusters usually consist of a cylindrical, annular chamber which is open at one end. Propellant, usually Xenon, is introduced at the closed end near the anode. This is shown in Figure 2.3 as the Anode/Gas Distributor. The Xenon atoms are ionized by electrons flowing into the chamber from the open end of the cylinder. The electrons

originate from an external cathode neutralizer also shown in Figure 2.3. The external cathode neutralizer produces electrons to ionize the Xenon propellant atoms and to neutralize the thruster exhaust. The external, negative cathode neutralizer also produces an axial electrostatic field which accelerates the ions toward the open end of the chamber. The Xenon ions and some of the electrons from the external cathode neutralizer leave the engine as the thruster exhaust. It is important that the exhaust be electrically neutral to prevent charge buildup on the spacecraft. Such a charge buildup could attract the thrust exhaust and cancel the thrust effects. Magnets installed on the inner and outer edges of the engine cylinder opening create a radial magnetic field which forces the electrons into an azimuthal drift (Hall current). The Hall current is the *ExB* current produced by the axial electric field (E) and the radial magnetic field (B). The radial magnetic field strongly affects the electrons due to their magnetic charge. The ions, however, are not magnetically charged and so are not strongly affected by the radial magnetic field (38). The Hall current thus produces an azimuthal drift for the electrons, but not the propellant ions (38).

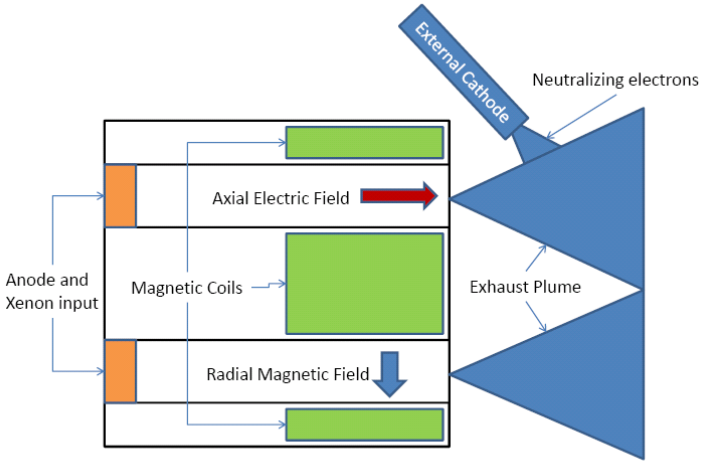

Figure 2.3 Hall Thruster Schematic

The azimuth directed electron Hall current produces a volume within which propellant atoms are readily ionized and are immediately accelerated by the strong cathode generated axial electrostatic field. In Figure 2.3 the volume in which the Xenon atoms are ionized by the Hall current is in the vicinity of the arrows which show the axial electric field and the radial magnetic field directions. This volume is shaped like a torus and correlates with the glowing ring in Figure 2.4. The engine thrust is produced because the ions accelerate against the electrons circulating in the magnetically confined Hall current. Because the electrons cannot freely accelerate toward the anode, they exert a magnetic force on the magnetic coil. Eventually, the electrons diffuse toward the anode/gas distributor and are pumped using the power supply to the external cathode neutralizer (38). Relatively efficient Hall thrusters have been developed and have flown on several Russian spacecraft. These SPT thrusters have been flown operationally since the 1980s on spacecraft such as the EXPRESS and GALS spacecraft (37). The SMART-1 European spacecraft also used a Hall thruster for several months to reach lunar orbit

(40). Hall thrusters also have applicability for geostationary satellite station keeping and U.S. vendors are manufacturing and testing Hall thrusters for such purposes. Hall thrusters can have efficiency factors of around 50% with specific impulse around 1500 seconds (38). The efficiency factor is the ratio of the kinetic energy produced by the engine for actual thrust to the energy supplied by the fuel and electric power systems. Specific impulse is a ratio of the speed of the beam exhaust to the Earth's acceleration due to gravity at sea level, i.e. 9.8 m/s^2.

Figure 2.4 shows the Aerojet BPT-2000 Hall thruster operating in a vacuum chamber. The BPT-2000 graphic is from Dr. Brad King (41) at Michigan Technological University.

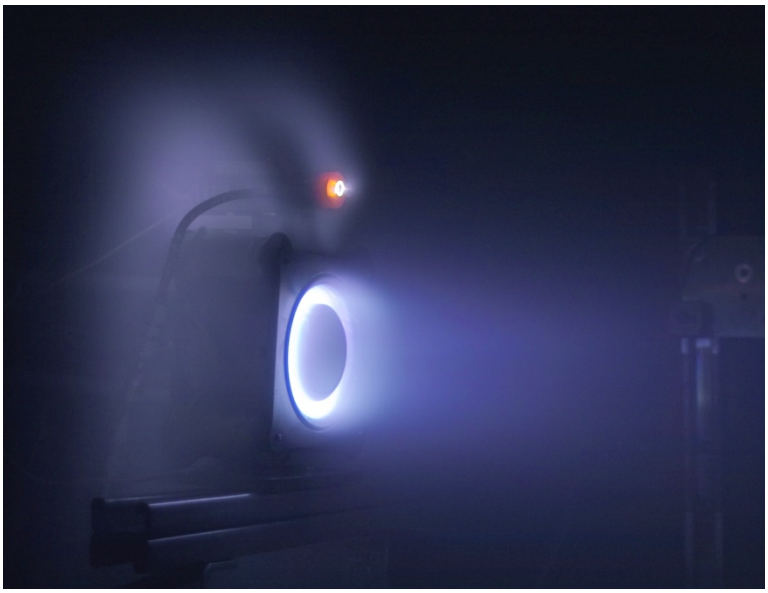

Figure 2.4 Aerojet BPT-2000 Hall Thruster
(courtesy, Dr. Brad King at Michigan Technological University)

Gridded ion engines produce ions, typically Xenon, by pumping electrons via a cathode into a magnetically confined chamber. An ion engine schematic is shown in Figure 2.5.

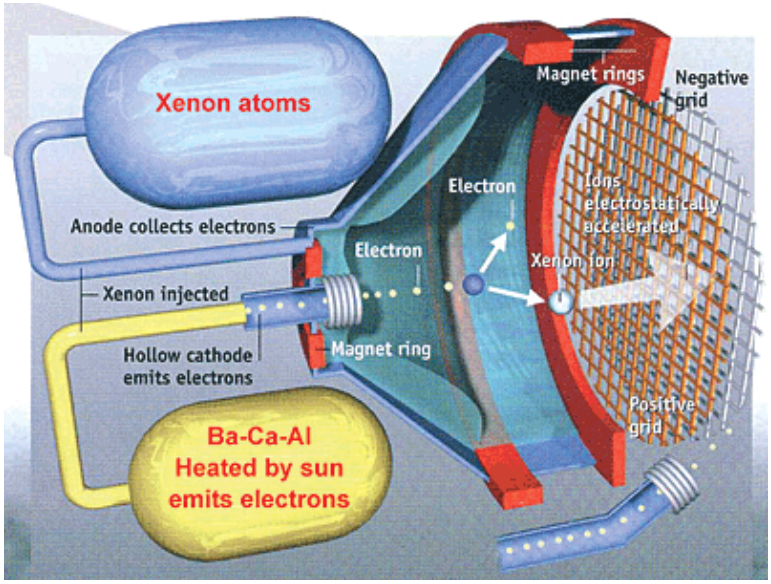

Figure 2.5 NASA Deep Space 1 Gridded Ion Engine Illustration[2]

The injected electrons bombard propellant atoms also introduced into the chamber and produce ions as a result. After a time, electrons are collected by the anode surrounding the magnetically confining chamber and are ejected using an external cathode. In Figure 2.5, the Anode is annotated as "Anode collects electrons," and the cathode that injects electrons into the chamber is annotated as "Hollow cathode emits electrons." The external cathode is not annotated, but is shown at the bottom right hand corner of Figure 2.5. The electron ejection by the external cathode prevents excess negative charge from building up in the engine. One side of the chamber, directed opposite the desired engine

[2] NASA Graphic obtained from http://www.nasa.gov/centers/glenn/images/content/83902main_ipsdiag.jpg visited Feb, 2008

thrust direction, is covered with two or more closely spaced grids. These grids are charged with voltage that accelerates Xenon ions that wander into the sheath covering the inner, positively charged grid and fall through (38). The grid spacing is on the order of 1 mm and it is within the grid gap that the Xenon ions accelerate due to the strong attraction of the Xenon ions to the negatively charged outer grid. The acceleration grids are designed with an ion optic geometry that reduces ion collision with the grids. The ion optic geometry essentially steers the ions toward the holes in the grid rather than toward the grid structure. This extends the life span of the grids. Because of the necessary small gap between the grids, a space charge limitation inhibits the number of ions that can be accelerated at any one time. This means that gridded ion engines must be larger in diameter than Hall thrusters in order to provide comparable thrust (38). Ion engines are at optimum efficiency at a high specific impulse and therefore provide less thrust per unit power than Hall thrusters (38). However, gridded ion engines offer more control of the plasma location within the structure and can offer longer life and better efficiency than Hall thrusters. Gridded ion engines can have a specific impulse greater than 2500 seconds and efficiency factors around 65% (38). Gridded ion engines have more complex power supply and processing requirements and therefore the power processing units (PPU) must be more complex and take up more mass than the PPU for a comparable Hall thruster (38).

Gridded ion engines are used on the Boeing 702 spacecraft (42). Several of these spacecraft have been launched and are operating currently (43). The NASA Deep Space 1 spacecraft flew with a 30-cm ion engine to intercept comet Wilson-Harrington (44).

The Pulsed Plasma Thruster (PPT), shown in Figure 2.6, is a markedly different type of thruster than the gridded ion engine or the Hall thruster. One difference is the propellant typically used. Instead of gaseous Xenon, PPTs use a solid bar of Teflon® (38). Another difference is that the PPT operates using short pulses on the order of 10 microseconds. A power supply charges the discharge capacitor and applies 1-2 kilovolts across the exposed Teflon face. A spark plug initiates the discharge. The combination of thermal flux, particle bombardment and surface reactions depolymerizes, evaporates, and mostly ionizes a small amount of material (1.5 micrograms per Joule). The instantaneous current is in the tens of kilo Amps and the self-induced magnetic field creates a magnetic pressure that is comparable to the gas kinetic pressure in the thin ionized layer (38). The combination of pressure gradients accelerates the gas to speeds in the vicinity of the "critical Alfven velocity." At this velocity, the kinetic energy is equal to the ionization energy (38). The PPT can achieve a specific impulse of 1500 seconds, but an engine efficiency factor of only around 7% (38). Figure 2.6 shows a diagram of a PPT.

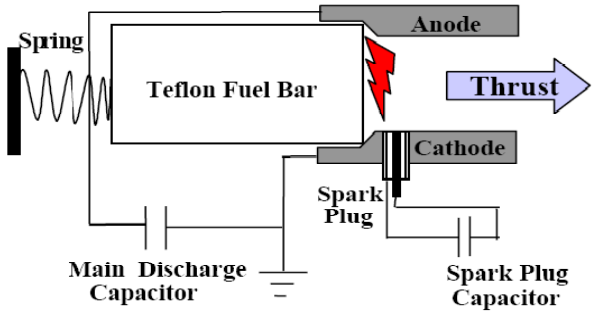

Figure 2.6 Pulsed Plasma Thruster from NASA Earth Observing 1 (EO-1)[3]

[3] NASA Graphic obtained from http://eo1.gsfc.nasa.gov/miscPages/TechForumPres/25-PPT.pdf visited Feb, 2008

An advantage of the PPT is its ability to operate over a wide range of thrust by varying the repetition rate. This ability allows the thruster to precisely apply thrust for spacecraft orbit or attitude adjustments. This thruster has been used since the 1960s on the series of LES 6,7,8, and 9 (Lincoln Experimental Satellites). These thrusters have also been used on the U.S. Navy's NOVA constellation (38).

For the purposes of orbit determination, the necessary engine parameters are the thrust characteristics. These characteristics include whether the thruster operates continuously for long periods and whether it uses variable thrust. Also, the thrust magnitude and the thruster orientation with respect to the spacecraft body and attitude must be considered when modeling the spacecraft thrust acceleration. For most thrusters that operate at low-thrust levels for continuous periods in orbit, the thrust magnitude, specific impulse and power have been published. Table 2.1 includes several electric thrusters that have been used on operational spacecraft. The sources of this information came from references (37), (45), (46), (47) and (48).

Table 2.1 EP Thruster Parameters for Several Launched Satellites

Satellite/ Launch Date	Thruster	Type	Manu-facturer/ Model	Isp (sec)	Thrust Mag. (N)	SC Mass (BOL[4]) (kg)	Power (Watts)	Apert. (cm)
Anik F1 11/21/2000	4 XIPS	Grid. Ion	Boeing 702	3800	0.08	3015	4500	25
Anik F2 6/18/2004	4 XIPS	Grid. Ion	Boeing 702	3800	0.165	3805	4500	25
DirecTV10 7/7/2007	4 XIPS	Grid. Ion	Boeing 702	3800	0.165	3700	4500	25
Galaxy IIIC 6/15/2002	4 XIPS/ 4 XIPS	Grid. Ion	Boeing 702	3800/ 2568	0.165/ 0.08	2873	4500/ 500	25/ 13
Galaxy XI 12/21/1999	8 XIPS	Grid. Ion	Boeing 702	2568	0.08	2775	500	13
NSS-8 1/30/2007	4 XIPS	Grid. Ion	Boeing 702	3800	0.165	3800	4500	25
PAS 1-R 11/15/2000	4 XIPS/ 4 XIPS	Grid. Ion	Boeing 702	3800/ 2568	0.165/ 0.08	3059	4500/ 500	25/ 13
Spaceway 1 4/26/2005	4 XIPS	Grid. Ion	Boeing 702	3800/ 2568	0.165/ 0.08	3832	4500/ 500	25
Spaceway 2 11/16/2005	4 XIPS	Grid. Ion	Boeing 702	3800/ 2568	0.165/ 0.08	3832	4500/ 500	25
Spaceway 3 8/14/2007	4 XIPS	Grid. Ion	Boeing 702	3800/ 2568	0.165/ 0.08	3832	4500/ 500	25
XM-1 3/18/2001	4 XIPS	Grid. Ion	Boeing 702	3800	0.165	2950	4500	25
XM-2 5/8/2001	4 XIPS	Grid. Ion	Boeing 702	3800	0.165	2950	4500	25
XM-3 1/3/2005	4 XIPS	Grid. Ion	Boeing 702	3800	0.165	~3000	4500	25
XM-4 10/30/2006	4 XIPS	Grid. Ion	Boeing 702	3800	0.165	~3000	4500	25
WGS F1 10/11/2007	4 XIPS	Grid. Ion	Boeing 702	3800	0.165	3680	4500	25
MBSAT 3/13/2004	SPT-100	Hall	Loral/ISTI	1600	0.083	3800	1350	10
DeepSpace1 10/24/1998	XIPS/ NSTAR	Grid. Ion	Boeing/ NASA	3100	0.020-0.092	486	500-2300	30
SMART-1 9/27/2003	PPS-1350	Hall	ESA/ SNECMA	1640	0.068	305	1190	10
EO-1 11/21/2000	EO-1 PPT	PPT	Swales/ Northrup/ Aerojet	650-1400	90-860 μN-sec	529	70	
ARTEMIS 7/12/2001	IPP/ Kaufman	Grid. Ion	UK (DRA) ESA	3285-3370	0.016-0.018		570	10

[4] BOL stands for beginning of life

2.2.2 Electric propulsion optimal orbit transfer

Controlling satellites with electric propulsion often means that fuel usage and transfer time should be minimized. In this thesis, the satellite control problem involves thrusting a satellite continuously to take the satellite from an initial orbit to a desired final orbit in minimum time. Constant continuous thrust magnitude is assumed and therefore, the minimum time and minimum fuel solutions are the same. The direction of thrust, i.e. pitch and yaw angles, can be optimized by formulating the problem in terms of *Optimal Control Theory*. Several references were used for understanding this formulation. They include Bryson and Ho's *Applied Optimal Control* (49) and Kirk's *Optimal Control Theory* (50). The ultimate formulation used for this research came from Jean Kechichian's series of papers (3), (4), (5), (6), (7) and (8). These were published starting in the 1990's and were devoted to deriving equinoctial element formulations to calculate optimal thrust plans for orbital transfers. Kechichian's development uses equinoctial elements to represent the orbital state because singularities due to zero eccentricity and inclination inherent to the Keplerian elements are avoided. These singularities would cause problems with the numeric integration required in this optimal control problem (3).

2.2.2.1 Optimal Control Theory Background

Before explaining Kechichian's development of optimal orbit transfers, it is useful to discuss the elements of *Optimal Control Theory* that are used. This discussion is taken from Kirk's *Optimal Control Theory* (50), from Bryson and Ho's *Applied Optimal Control* (49) and from Professor Jonathan How's MIT Course 16.323 notes (51).

The orbital system dynamics are as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{2.65a}$$

The initial conditions are defined by:

$$\mathbf{x}(t_0) = \mathbf{x_0} \tag{2.65b}$$

A performance measure, i.e. cost functional, is introduced:

$$J(\mathbf{u}) = h(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t)dt \tag{2.66}$$

Here, the initial time, t_0, is specified and the final time, t_f, is not bounded, i.e. it is free. The state, **x**, is an *n* x 1 state vector and **u** is an *m* x 1 control input vector. The function, *h*, is used to assign cost to the terminal state. The state cost function, *g*, is used to assign cost to the path obtained using a given state history, **x**(*t*), and control history, **u(***t*). Our goal is to find an optimal control history, $\mathbf{u}^*(t)$**,** that produces a time and fuel optimal state history, i.e. trajectory, $\mathbf{x}^*(t)$.

We can adjoin Lagrange multipliers to augment the performance measure (50), (49) and (51):

$$J_a = h(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} \left[g(\mathbf{x}(t), \mathbf{u}(t), t) + \boldsymbol{\lambda}^T \{\mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t) - \dot{\mathbf{x}}\} \right] dt \tag{2.67}$$

where the vector of *n* Lagrange multipliers, **λ**, is multiplied to a quantity that is equal to zero given the system dynamics equation shown earlier. The *variation* of this equation can be taken in order to minimize the functional, J_a. We can use the *Fundamental Theorem of the Calculus of Variations* to find the minima of the functional, J_a, given the *variation* of the functional, δJ_a. Here, we assume the state time history, **x**(*t*) is

continuous. The *variation* of the functional, J_a, vanishes on the minimum or maximum state time histories (50):

$$\delta J_a(\mathbf{x}^*, \delta\mathbf{x}) = 0 \tag{2.68}$$

The *variation*, δ, is defined in terms of the *increment*. The *increment* of functional J is defined as (50):

$$\Delta J \equiv J(\mathbf{x} + \delta\mathbf{x}) - J(\mathbf{x}) \tag{2.69}$$

The *increment* can also be written as (50):

$$\Delta J(\mathbf{x}, \delta\mathbf{x}) = \delta J(\mathbf{x}, \delta\mathbf{x}) + q(\mathbf{x}, \delta\mathbf{x}) \cdot \|\delta\mathbf{x}\| \tag{2.70}$$

where $\|*\|$ denotes a *norm* operation. The linear part of the *increment*, $\delta J(\mathbf{x}, \delta\mathbf{x})$, is defined as the *variation* of the functional, J. The functional, q, collects all higher order terms of the *increment* of J.

Now, find the variation of the functional, δJ_a (51):

$$\begin{aligned}\delta J_a = &\int_{t_0}^{t_f} \left[g_{\mathbf{x}}\delta\mathbf{x} + g_{\mathbf{u}}\delta\mathbf{u} + (\mathbf{a} - \dot{\mathbf{x}})^T \delta\boldsymbol{\lambda}(t) + \boldsymbol{\lambda}^T(t)\{\mathbf{a}_{\mathbf{x}}\delta\mathbf{x} + \mathbf{a}_{\mathbf{u}}\delta\mathbf{u} - \delta\dot{\mathbf{x}}\} \right] dt \\ &+ h_{\mathbf{x}}\delta\mathbf{x}_f + h_{t_f}\delta t_f + \left[g + \boldsymbol{\lambda}^T(\mathbf{a} - \dot{\mathbf{x}}) \right](t_f)\delta t_f\end{aligned} \tag{2.71}$$

Here, the subscripts of *g, h,* and **a** imply partial derivatives, i.e. $\mathbf{a}_{\mathbf{u}} \equiv \frac{\partial \mathbf{a}}{\partial \mathbf{u}}$. The *Hamiltonian* is defined as follows:

$$H(\mathbf{x}, \mathbf{u}, \mathbf{p}, t) = g(\mathbf{x}(t), \mathbf{u}(t), t) + \boldsymbol{\lambda}^T(t)\,\mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{2.72}$$

Using the *Hamiltonian*, the variation of the functional, δJ_a, becomes (51):

$$\begin{aligned}\delta J_a = &h_{\mathbf{x}}\delta\mathbf{x}_f + \left[h_{t_f} + g + \boldsymbol{\lambda}^T(\mathbf{a} - \dot{\mathbf{x}}) \right](t_f)\delta t_f \\ &+ \int_{t_0}^{t_f} \left[H_{\mathbf{x}}\delta\mathbf{x} + H_{\mathbf{u}}\delta\mathbf{u} + (\mathbf{a} - \dot{\mathbf{x}})^T \delta\boldsymbol{\lambda}(t) - \boldsymbol{\lambda}^T(t)\delta\dot{\mathbf{x}} \right] dt\end{aligned} \tag{2.73}$$

In equation (2.73), the (t_f) at the end of the second term is used to signify that t_f is an argument for all quantities within the brackets. Integrating the last term in the integrand by parts yields (51):

$$\begin{aligned} -\int_{t_0}^{t_f} \boldsymbol{\lambda}^T(t)\delta\dot{\mathbf{x}}dt &= -\int_{t_0}^{t_f} \boldsymbol{\lambda}^T(t)d\delta\mathbf{x} \\ &= -\boldsymbol{\lambda}^T\delta\mathbf{x}\Big|_{t_0}^{t_f} + \int_{t_0}^{t_f}\left(\frac{d\boldsymbol{\lambda}(t)}{dt}\right)^T\delta\mathbf{x}dt \\ &= -\boldsymbol{\lambda}^T(t_f)\delta\mathbf{x}(t_f) + \int_{t_0}^{t_f}\dot{\boldsymbol{\lambda}}^T(t)\delta\mathbf{x}dt \\ &= -\boldsymbol{\lambda}^T(t_f)(\delta\mathbf{x}_f - \dot{\mathbf{x}}(t_f)\delta t_f) + \int_{t_0}^{t_f}\dot{\boldsymbol{\lambda}}^T(t)\delta\mathbf{x}dt \end{aligned} \tag{2.74}$$

Rewrite the variation of the functional, δJ_a (51):

$$\begin{aligned} \delta J_a = (h_{\mathbf{x}} - \boldsymbol{\lambda}^T(t_f))\delta\mathbf{x}_f + \Big[h_{t_f} + g + \boldsymbol{\lambda}^T(\mathbf{a}-\dot{\mathbf{x}}) + \boldsymbol{\lambda}^T\dot{\mathbf{x}}\Big](t_f)\delta t_f \\ + \int_{t_0}^{t_f}\Big[(H_{\mathbf{x}} + \dot{\boldsymbol{\lambda}}^T)\delta\mathbf{x} + H_{\mathbf{u}}\delta\mathbf{u} + (\mathbf{a}-\dot{\mathbf{x}})^T\delta\boldsymbol{\lambda}(t)\Big]dt \end{aligned} \tag{2.75}$$

The (t_f) at the end of the third term in the right hand side of equation (2.75) indicates that all quatities inside the bracket are functions of the final time. By using the *Fundamental Theorem of the Calculus of Variations*, the necessary conditions for the minimum or maximum of the functional, J_a, are met when $\delta J_a = 0$. The necessary conditions are also known as the Euler-Lagrange equations and are (49), (50), (51):

$$\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u}, t) \tag{2.76a}$$

$$\dot{\boldsymbol{\lambda}} = -H_{\mathbf{x}}^T = -\left(\frac{\partial H}{\partial \mathbf{x}}\right)^T \tag{2.76b}$$

$$H_{\mathbf{u}} = \frac{\partial H}{\partial \mathbf{u}} = 0 \tag{2.76c}$$

Because the dimension of **x** and **p** are n x 1, the dimensions of (2.76a) and (2.76b) are also n x 1. The dimension of **u** is m x 1 and therefore, the dimension of $H_{\mathbf{u}}$ is also m x 1. If our final time, t_f is free, we also have the boundary condition (51):

$$h_{t_f} + g + \boldsymbol{\lambda}^T \mathbf{a} = h_{t_f} + H(t_f) = 0 \tag{2.77}$$

We still have the fixed initial, and fixed or free final conditions (51):

$$\mathbf{x}(t_0) = \mathbf{x}_0 \tag{2.78}$$

$$\mathbf{x}_i(t_f) = \mathbf{x}_{i_f} \quad \text{if } \mathbf{x}_f \text{ is fixed} \tag{2.79}$$

$$\boldsymbol{\lambda}_i(t_f) = \frac{\partial h}{\partial x_i}(t_f) \quad \text{if } \mathbf{x}_f \text{ is free} \tag{2.80}$$

According to references (52) and (49), the Lagrange multipliers are sensitivities of the performance index, *J*, to small changes in the initial conditions, i.e. the Lagrange multiplier functions, $\boldsymbol{\lambda}^T$(t), are the partial derivatives of the performance measure, *J*, with respect to the initial conditions, $\mathbf{x}(t_0)$. The Lagrange multipliers are sometimes called the *influence functions* (49).

This derivation allows us to proceed with the optimal control problem for controlling the constant, continuous thrust of a satellite from an initial orbit to a final orbit while optimizing fuel usage.

2.2.2.2 Minimum-Time Trajectory Optimization Problem

The optimal low-thrust control problem for initial and final orbits was initially solved in the 1960s by Gobetz and Edelbaum using an application of the equinoctial orbital elements (53), (54). These solutions mostly dealt with circular initial and final orbits. However, the more general problem of thrusting from initial and final orbits with

significant eccentricity and inclination was dealt with in reference (55)[5]. Starting in the 1990s, Jean Albert Kechichian further developed these methods (9) and his work is the basis for the optimal control problem formulation in this thesis. Because of the robust convergence characteristics and the relatively simple formulation, the development from Kechichian's paper (5) is followed.

The nonsingular equinoctial elements, $\mathbf{x} = \{a, h, k, p, q, L\}$, are used as the state elements in the dynamic equations. The elements *a,h,k,p,q* are identical to the equinoctial orbit elements described in section 2.1.3.5. These elements were developed by Broucke and Cefola (31), (56). The *L* element is the true longitude and is defined as $L = \theta^* + \omega + \Omega$. Here, θ^* is the true anomaly, ω is the argument of perigee, and Ω is the right ascension of the ascending node. The following variational equations define the time derivatives of the equinoctial elements. These equations match the form of the constraint equations from the optimal control formulation, $\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u}, t)$ (5):

$$\dot{a} = \frac{2}{n(1-h^2-k^2)^{\frac{1}{2}}}\left[(ks_L - hc_L) f_r + (1 + hs_L + kc_L) f_\theta\right] \tag{2.81a}$$

$$\begin{aligned}\dot{h} &= \frac{(1-h^2-k^2)^{\frac{1}{2}}}{na(1+hs_L+kc_L)} \\ &\times \left\{-(1+hs_L+kc_L)c_L f_r + \left[h + (2+hs_L+kc_L)s_L\right] f_\theta - k(pc_L - qs_L) f_h\right\}\end{aligned} \tag{2.81b}$$

$$\begin{aligned}\dot{k} &= \frac{(1-h^2-k^2)^{\frac{1}{2}}}{na(1+hs_L+kc_L)} \\ &\times \left\{(1+hs_L+kc_L)s_L f_r + \left[k + (2+hs_L+kc_L)c_L\right] f_\theta + h(pc_L - qs_L) f_h\right\}\end{aligned} \tag{2.81c}$$

[5] This application of the equinoctial orbital elements was significant because it was the first use of the equinoctial orbital elements apart from the authors who introduced the concept, Broucke and Cefola.

$$\dot{p} = \frac{(1-h^2-k^2)^{\frac{1}{2}}}{2na(1+hs_L+kc_L)}\left(1+p^2+q^2\right)s_L f_h \tag{2.81d}$$

$$\dot{q} = \frac{(1-h^2-k^2)^{\frac{1}{2}}}{2na(1+hs_L+kc_L)}\left(1+p^2+q^2\right)c_L f_h \tag{2.81e}$$

$$\dot{L} = \frac{n(1+hs_L+kc_L)^2}{(1-h^2-k^2)^{\frac{1}{2}}} + \frac{(1-h^2-k^2)^{\frac{1}{2}}}{na(1+hs_L+kc_L)}(qs_L - pc_L)f_h \tag{2.81f}$$

Equations in (2.81) are transformed versions of equations in (2.31). The true longitude, *L*, has replaced the mean longitude, λ, as the fast variable. Equations in (2.81) are formulated to use thrust acceleration in a polar frame while the equations in (2.31) are formulated to use the perturbing acceleration in equinoctial (*f,g,w*) inertial coordinates. The similar formulations are notable because Kechichian's formulation follows from the work of Edelbaum in the 1970s (55) which used the equinoctial elements introduced by Broucke and Cefola (31). In equation (2.81), the symbol, *n,* denotes the mean motion, the s_L and c_L variables are *sin(L)* and *cos(L),* respectively, and the control vector, **u**, has rotating polar components, $[u_r, u_\theta, u_h]^T$. The complete expression for the disturbing acceleration due to thrust is $\Gamma = \mathbf{f}/m = f_t\mathbf{u} = f_t[u_r, u_\theta, u_h]^T$. The symbol, *m*, is the spacecraft mass, and f_t is the magnitude of the thrust vector, **f**. The variational equations can also be represented by (5):

$$\begin{pmatrix} \dot{a} \\ \dot{h} \\ \dot{k} \\ \dot{p} \\ \dot{q} \\ \dot{L} \end{pmatrix} = \begin{pmatrix} B_{11}^L & B_{12}^L & B_{13}^L \\ B_{21}^L & B_{22}^L & B_{23}^L \\ B_{31}^L & B_{32}^L & B_{33}^L \\ B_{41}^L & B_{42}^L & B_{43}^L \\ B_{51}^L & B_{52}^L & B_{53}^L \\ B_{61}^L & B_{62}^L & B_{63}^L \end{pmatrix} \begin{pmatrix} u_r \\ u_\theta \\ u_h \end{pmatrix} f_t + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dfrac{na^2(1-h^2-k^2)^{\frac{1}{2}}}{r^2} \end{pmatrix} \tag{2.82a}$$

where the 6 x 3 B^L matrix is fully defined in Appendix B. Other perturbations in the rotating Euler-Hill polar frame, $f_p = \begin{bmatrix} f_r & f_\theta & f_h \end{bmatrix}^T$, can be added to the perturbation acceleration due to thrust:

$$\dot{\mathbf{x}} = B^L\left(\begin{bmatrix} u_r & u_\theta & u_h \end{bmatrix}^T f_t + \begin{bmatrix} f_r & f_\theta & f_h \end{bmatrix}^T\right) + \frac{na^2(1-h^2-k^2)^{\frac{1}{2}}}{r^2} \tag{2.82b}$$

Adding perturbations such as J_2 and lunar and solar gravity is future work with respect to this thesis. However, adding the J_2 perturbation is shown in reference (57).

We can put this formulation into the optimal control framework by adjoining Lagrange multipliers to the equinoctial element time derivatives. We can then define the *Hamiltonian* as (5):

$$H = \boldsymbol{\lambda}_{\mathbf{x}}^T B^L(\mathbf{x}) f_t \mathbf{u} + \lambda_L \frac{na^2(1-h^2-k^2)^{\frac{1}{2}}}{r^2} \tag{2.83}$$

Here, the vector of Lagrange multipliers is defined as $\boldsymbol{\lambda}_{\mathbf{x}}^T = \{\lambda_a, \lambda_h, \lambda_k, \lambda_p, \lambda_q, \lambda_\lambda\}$. In the case of $\boldsymbol{\lambda}_{\mathbf{x}}^T$, no partial derivative is implied. The necessary conditions for the optimal control are then (5):

$$\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u}, t) \tag{2.84}$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \mathbf{x}} \tag{2.85}$$

$$\frac{\partial H}{\partial \mathbf{u}} = 0 \tag{2.86}$$

The first of the conditions is already defined by the variational equations. The second condition, $\dot{\boldsymbol{\lambda}} = -\dfrac{\partial H}{\partial \mathbf{x}}$, can be written as (5):

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \mathbf{x}} = -\boldsymbol{\lambda}_{\mathbf{x}}^{T} \frac{\partial B^{L}}{\partial \mathbf{x}} f_t \mathbf{u} - \lambda_L \frac{\partial}{\partial \mathbf{x}} \left[\frac{na^2 (1-h^2-k^2)^{\frac{1}{2}}}{r^2} \right] \tag{2.87}$$

This adjoint equation requires the partials of the B^L matrix with respect to each of the equinoctial elements. These partial derivatives were derived by Kechichian (5) and are shown in Appendix B along with the elements of the B^L matrix. The B^L matrix includes all terms that are required to reproduce the variational equations when multiplied by the perturbing thrust acceleration vector as shown in equation (2.82a). The partials of the B^L matrix with respect to the equinoctial orbital elements are required in order to evaluate the variational equations for the Lagrange multipliers as shown in equation (2.87). These partial derivatives are also included in Appendix B.

The third equation in the necessary conditions, $\partial H / \partial \mathbf{u} = 0$, can be met by choosing the control or thrust vector, **u**, such that it is always parallel to $\boldsymbol{\lambda}_{\mathbf{x}}^{T} B^{L}(\mathbf{x}) f_t$. This maximizes the *Hamiltonian* because of how it was defined in equation (2.83) (5). Therefore, the optimized thrust vector, $\mathbf{u}^{*}$, is obtained from (5):

$$\mathbf{u} = \frac{(\boldsymbol{\lambda}_{\mathbf{x}}^{T} B^{L})^{T}}{\left| \boldsymbol{\lambda}_{\mathbf{x}}^{T} B^{L} \right|} \tag{2.88}$$

This thrust control vector produces a trajectory that is optimized for the choice of the cost functional in equation (2.66). The exact choices for the *h* and *g* functions are shown later. The thrust pitch and yaw angles can be calculated from the thrust vector. The

thrust vector is defined in terms of rotating Euler-Hill polar coordinates defined in Appendix C, $\{u_r, u_\theta, u_h\}$, and so the thrust pitch and yaw angles are (6):

$$\theta_{pitch} = \tan^{-1}\left(\frac{u_r}{u_\theta}\right) \tag{2.89}$$

$$\theta_{yaw} = \tan^{-1}\left(\frac{u_h}{u_\theta}\right) \tag{2.90}$$

Because we are assuming constant thrust, we can minimize fuel usage by minimizing the total transfer time. We can choose the cost functional with $h(\mathbf{x}(t_f), t_f) = 0$ and $g(\mathbf{x}(t), \mathbf{u}(t), t) = 1$ and therefore, the cost functional to minimize is (3):

$$J = \int_{t_0}^{t_f} dt = t_f - t_0 \tag{2.91}$$

This is equivalent to maximizing the cost functional (3):

$$J = -\int_{t_0}^{t_f} dt = -(t_f - t_0) \tag{2.92}$$

We then redefine our *g* function as $g(\mathbf{x}(t), \mathbf{u}(t), t) = -1$ so that we are again minimizing the cost. With $g(\mathbf{x}(t), \mathbf{u}(t), t) = -1$, we can specify the augmented *Hamiltonian* as (3):

$$H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = -1 + \boldsymbol{\lambda}^T(t)\,\mathbf{a}(\mathbf{x}(t), \mathbf{u}(t)) = -1 + \boldsymbol{\lambda}^T(t)\dot{\mathbf{x}} \tag{2.93}$$

At the final time, t_f, the augmented *Hamiltonian* will be $H_f = 0$. This comes out of the boundary condition at the final time (3):

$$g + \boldsymbol{\lambda}^T \mathbf{a} = H(t_f) = 0 \tag{2.94}$$

Therefore, our augmented *Hamiltonian* will be zero at the final time and will thus indicate a time-optimal trajectory if the un-augmented *Hamiltonian* is equal to unity, i.e. $\boldsymbol{\lambda}^T(t_f)\dot{\mathbf{x}}(t_f) = 1$. This fact can be used later in the numerical quasi-Newton search algorithm. We also know that because the *Hamiltonian* is not an explicit function of

time, it will be constant throughout the time interval from t_0 to t_f when the trajectory is optimal (49). The constancy of the *Hamiltonian* is an important indicator of the optimality of the trajectory.

2.2.2.3 Numerical Solution Method for Trajectory Optimization Problem

In order to solve the two-point boundary value problem posed by this formulation of the satellite optimal control problem, one needs to be able to integrate the variational equations for the equinoctial elements (2.81) and the variational equations for the associated Lagrange multipliers (2.87). The 6 variational equations for the equinoctial elements are integrated from the initial conditions (2.78) at t_0 to the guessed final time, t_f. The 6 variational equations for the Lagrange multipliers are integrated from initial guesses at the initial time, $t_0, \boldsymbol{\lambda}_0^T = \{\lambda_a, \lambda_h, \lambda_k, \lambda_p, \lambda_q, \lambda_\lambda\}_0$, to final values, $\boldsymbol{\lambda}_f^T = \{\lambda_a, \lambda_h, \lambda_k, \lambda_p, \lambda_q, \lambda_\lambda\}_f$ at the final guessed time, t_f. The integrator chosen for this task must have sufficient accuracy because the 7-parameter unconstrained minimization depends on very accurate correspondence between the initial and final conditions. Kechichian (4) uses a 7^{th} order Runge-Kutta-Fehlberg integrator (RK78) with 10^{-9} error tolerance for this purpose, and so that integrator is also used for the trajectory optimization in this thesis. The source code for the RK78 integrator was developed at NASA and is available on the web at (http://www.astro.su.se/~pawel/rk78.html).

The guesses for the initial Lagrange multipliers and final time can be refined through the use of a 7-parameter search. The search tries to find the best initial guesses for Lagrange multipliers and final time to match the desired final equinoctial orbital

elements. For this search, Kechichian chose the minimization algorithm UNCMIN (58). This algorithm performs an unconstrained minimization on a given real-valued function $F(\mathbf{x})$. The number of variables in the vector, $\mathbf{x}$, is n. This dimension, n, must match the number of variables to be guessed by the minimization algorithm. The UNCMIN algorithm uses a quasi-Newton search which is based on the general descent method (58). In the Newton method, the step p is computed from the solution of a set of n linear equations known as the Newton equations (3):

$$\nabla^2 F(\mathbf{x}) p = -\nabla F(\mathbf{x}) \tag{2.95}$$

The solution is updated by using (3):

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + p \\ &= \mathbf{x}_k - [\nabla^2 F(\mathbf{x}_k)]^{-1} \nabla F(\mathbf{x}_k) \end{aligned} \tag{2.96}$$

The gradient of $F(\mathbf{x})$ is denoted by $\nabla F(\mathbf{x})$, and the constant matrix of second partial derivatives of $F(\mathbf{x}), \nabla^2 F(\mathbf{x})$, denotes the Hessian matrix (3). The direction given by the step, p, is guaranteed to be a descent direction only if $[\nabla^2 F(\mathbf{x}_k)]^{-1}$ is positive definite, i.e. $\mathbf{z}^T [\nabla^2 F(\mathbf{x}_k)]^{-1} \mathbf{z} > 0$ for all $\mathbf{z} \neq 0$ (3). This can be shown using the Taylor expansion for F at $\mathbf{x} + p$ (58):

$$\begin{aligned} F(\mathbf{x} + \varepsilon p) &= F(\mathbf{x}) + \nabla F^T (\varepsilon p) + O(\varepsilon^2) \\ &= F(\mathbf{x}) + \varepsilon \nabla F^T (-\nabla^2 F^{-1} \nabla F) + O(\varepsilon^2) \\ &= F(\mathbf{x}) - \varepsilon \nabla F^T \nabla^2 F^{-1} \nabla F + O(\varepsilon^2) \end{aligned} \tag{2.97}$$

If we assume $[\nabla^2 F(\mathbf{x}_k)]^{-1}$ is positive definite, then $\nabla F^T \nabla^2 F^{-1} \nabla F > 0$ as long as $\nabla F \neq 0$. Therefore, If ε is small and $\nabla F \neq 0$, then $F(\mathbf{x} + \varepsilon p) < F(\mathbf{x})$ and p is in a downhill direction. If $\nabla F = 0$, then $\mathbf{x}$ is a critical point, and further conditions involving second derivatives must be checked to determine if $\mathbf{x}$ minimizes the function (58).

The Newton method described above is modified in the implementation of UNCMIN to build an approximation to the Hessian matrix using a *secant approximation* (58). This modified method avoids the cost of computing the 2nd derivative Hessian matrix explicitly. The *secant approximation* works by starting with $B_k \approx \nabla^2 F_k = \nabla^2 F(\mathbf{x}_k)$ and then using a step, p, defined by (58):

$$B_k p = -\nabla F(\mathbf{x}_k) \tag{2.98}$$

This step, p, is used with the general descent method shown in equations (2.95) and (2.96). After the line search obtains $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha p$, the approximate Hessian, B_k, is updated using the values of $\mathbf{x}_{k+1}$ and $\nabla F(\mathbf{x}_{k+1})$ to produce the new approximation, B_{k+1} (58). This can be illustrated by starting with a quadratic function, $F(\mathbf{x})$, which satisfies:

$$(\nabla^2 F)(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla F(\mathbf{x}_{k+1}) - \nabla F(\mathbf{x}_k) \tag{2.99}$$

In this case, the approximation of the Hessian matrix will be chosen so that (58):

$$B_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla F(\mathbf{x}_{k+1}) - \nabla F(\mathbf{x}_k) \tag{2.100}$$

The advantages of this method are that the solution converges rapidly near the solution, only gradient values are needed rather than second derivatives, a positive definite B_k can always be used so that a descent direction is always chosen, and the work per iteration can be reduced to $O(n^2)$ owing to the modification of B_k by a low-rank matrix (58).

The function, $F(\mathbf{x})$, chosen for the trajectory optimization problem is (3):

$$\begin{aligned} F(\mathbf{z}, \mathbf{z}_f, \mathbf{w}, H) = {} & w_1(a - a_f)^2 + w_2(h - h_f)^2 + w_3(k - k_f)^2 + w_4(p - p_f)^2 \\ & + w_5(q - q_f)^2 + w_6(\lambda_L - 0)^2 + w_7(H - 1)^2 \end{aligned} \tag{2.101}$$

Here, the relevant equinoctial elements and Lagrange multipliers at the final time are produced by the latest guess in the UNCMIN algorithm. The equinoctial elements are contained in vector $\mathbf{z} = \{a, h, k, p, q, \lambda_L\}$. The desired final orbital elements are contained in vector $\mathbf{z}_f = \{a_f, h_f, k_f, p_f, q_f\}$. This function includes weighting parameters in the vector, **w**, which can be used to emphasize a combination of elements during the execution of the UNCMIN algorithm (4). The Hamiltonian, *H*, is penalized for any difference from 1. This follows from equations (2.93) and (2.94) which define the optimality condition involving the Hamiltonian. The true longitude element is not included in the function to be minimized. It is left out and is therefore a free parameter. Instead of the true longitude, the Lagrange multiplier associated with the true longitude is included so that the optimal arrival point for the minimum-time transfer on the final orbit is reached (5).

2.2.2.4 Averaged Numerical Solution Method for Trajectory Optimization Problem

In order to obtain initial guesses for the Lagrange multipliers needed for the numerical method in 2.2.2.3, it is practical to apply the more robust, averaged variational equations to the problem (57). The averaged variational equation formulation uses a mean longitude formulation developed by Jean Kechichian in reference (4). The mean longitude formulation starts with the following equations of motion (4) in terms of the equinoctial elements, $\mathbf{x} = \{a, h, k, p, q, \lambda\}$. This formulation is equivalent to the equinoctial variational equation formulation outlined in section 2.1.2.5 in equation (2.31), but is expressed in a form that can be conveniently partitioned into a matrix.

$$\dot{a} = \left(\frac{\partial a}{\partial \dot{\mathbf{r}}}\right)^T \cdot \hat{\mathbf{u}} f_t \tag{2.102a}$$

$$\dot{h} = \left(\frac{\partial h}{\partial \dot{\mathbf{r}}}\right)^T \cdot \hat{\mathbf{u}} f_t \tag{2.102b}$$

$$\dot{k} = \left(\frac{\partial k}{\partial \dot{\mathbf{r}}}\right)^T \cdot \hat{\mathbf{u}} f_t \tag{2.102c}$$

$$\dot{p} = \left(\frac{\partial p}{\partial \dot{\mathbf{r}}}\right)^T \cdot \hat{\mathbf{u}} f_t \tag{2.102d}$$

$$\dot{q} = \left(\frac{\partial q}{\partial \dot{\mathbf{r}}}\right)^T \cdot \hat{\mathbf{u}} f_t \tag{2.102e}$$

$$\dot{\lambda} = n + \left(\frac{\partial \lambda}{\partial \dot{\mathbf{r}}}\right)^T \cdot \hat{\mathbf{u}} f_t \tag{2.102f}$$

The equinoctial elements are the same as those described in the section 2.2.2.2 except for the λ element which is the mean longitude. The mean longitude is defined in terms of the mean anomaly, M, and the argument of perigee, ω, and the right ascension of the ascending node, Ω.

$$\lambda = M + \omega + \Omega \tag{2.103}$$

The unit thrust vector, $\hat{\mathbf{u}}$, is in the direction of thrust and n is the orbital mean motion. The following equations show the partial derivatives of the equinoctial elements with respect to $\dot{\mathbf{r}}$ (4):

$$\frac{\partial a}{\partial \dot{\mathbf{r}}} = 2a^{-1}n^{-2}(\dot{X}_1\hat{f} + \dot{Y}_1\hat{g}) = M_{11}\hat{f} + M_{12}\hat{g} + M_{13}\hat{w} \tag{2.104a}$$

$$\begin{aligned}\frac{\partial h}{\partial \dot{\mathbf{r}}} &= \frac{G}{na^2}\left[\left(\frac{\partial X_1}{\partial k} - h\beta\frac{\dot{X}_1}{n}\right)\hat{f} + \left(\frac{\partial Y_1}{\partial k} - h\beta\frac{\dot{Y}_1}{n}\right)\hat{g}\right] + \frac{k(qY_1 - pX_1)}{na^2G}\hat{w} \\ &= M_{21}\hat{f} + M_{22}\hat{g} + M_{23}\hat{w}\end{aligned} \tag{2.104b}$$

$$\frac{\partial k}{\partial \dot{\mathbf{r}}} = -\frac{G}{na^2}\left[\left(\frac{\partial X_1}{\partial h} + k\beta\frac{\dot{X}_1}{n}\right)\hat{f} + \left(\frac{\partial Y_1}{\partial h} + k\beta\frac{\dot{Y}_1}{n}\right)\hat{g}\right] - \frac{h(qY_1 - pX_1)}{na^2 G}\hat{w} \qquad (2.104c)$$
$$= M_{31}\hat{f} + M_{32}\hat{g} + M_{33}\hat{w}$$

$$\frac{\partial p}{\partial \dot{\mathbf{r}}} = KY_1\frac{1}{2na^2 G}\hat{w} = M_{41}\hat{f} + M_{42}\hat{g} + M_{43}\hat{w} \qquad (2.104d)$$

$$\frac{\partial q}{\partial \dot{\mathbf{r}}} = KX_1\frac{1}{2na^2 G}\hat{w} = M_{51}\hat{f} + M_{52}\hat{g} + M_{53}\hat{w} \qquad (2.104e)$$

$$\frac{\partial \lambda}{\partial \dot{\mathbf{r}}} = \frac{1}{na^2}\left[-2X_1 + G\left(h\beta\frac{\partial X_1}{\partial h} + k\beta\frac{\partial X_1}{\partial k}\right)\right]\hat{f} +$$
$$\frac{1}{na^2}\left[-2Y_1 + G\left(h\beta\frac{\partial Y_1}{\partial h} + k\beta\frac{\partial Y_1}{\partial k}\right)\right]\hat{g} + \qquad (2.104f)$$
$$\frac{(qY_1 - pX_1)}{na^2 G}\hat{w}$$
$$= M_{61}\hat{f} + M_{62}\hat{g} + M_{63}\hat{w}$$

It should be noted that equation (2.104) is also compatible with the VOP equation derivation found in McClain (15). The reference frame for these equations is the direct equinoctial frame which is also used for the true longitude formulation described in section 2.2.2.2. The 6x3 M matrix is defined with these equations and is dependent on the elements and the eccentric longitude, *F*. The position and velocity vectors are also given in terms of the eccentric longitude (4). The eccentric longitude is related to the eccentric anomaly, *E*, by (4):

$$F = E + \tan^{-1}(h/k) \qquad (2.105)$$

The position and velocity are given by (4):

$$\mathbf{r} = X_1\hat{f} + Y_1\hat{g} \qquad (2.106)$$

$$\dot{\mathbf{r}} = \dot{X}_1\hat{f} + \dot{Y}_1\hat{g} \qquad (2.107)$$

The parameters are the same as those given earlier in equation (2.23) and are repeated here for convenience (32), (4):

$$X_1 = a\left[(1-h^2\beta)\cos F + hk\beta \sin F - k\right] \tag{2.108}$$

$$Y_1 = a\left[hk\beta \cos F + (1-k^2\beta)\sin F - h\right] \tag{2.109}$$

$$\dot{X}_1 = \frac{na^2}{r}\left[hk\beta \cos F - (1-h^2\beta)\sin F\right] \tag{2.110}$$

$$\dot{Y}_1 = \frac{na^2}{r}\left[(1-k^2\beta)\cos F - hk\beta \sin F\right] \tag{2.111}$$

Here, $\beta = 1/(1+G)$, $G = \sqrt{(1-h^2-k^2)}$, $K = (1+p^2+q^2)$, and $r = a(1-k\cos F - h\sin F)$. As the mean longitude λ is being integrated, it becomes necessary to solve Kepler's transcendental equation by iteration, i.e. $\lambda = F - k\sin F + h\cos F$ (4). The following partial derivatives are needed in the equations of motion shown above (4):

$$\frac{\partial X_1}{\partial h} = a\left[-(h\cos F - k\sin F)\left(\beta + \frac{h^2\beta^3}{1-\beta}\right) - \frac{a}{r}(h\beta - \sin F)\cos F\right] \tag{2.112}$$

$$\frac{\partial X_1}{\partial k} = -a\left[(h\cos F - k\sin F)\frac{hk\beta^3}{1-\beta} + 1 + \frac{a}{r}(\sin F - h\beta)\sin F\right] \tag{2.113}$$

$$\frac{\partial Y_1}{\partial h} = a\left[(h\cos F - k\sin F)\frac{hk\beta^3}{1-\beta} - 1 + \frac{a}{r}(k\beta - \cos F)\cos F\right] \tag{2.114}$$

$$\frac{\partial Y_1}{\partial k} = a\left[(h\cos F - k\sin F)\left(\beta + \frac{k^2\beta^3}{1-\beta}\right) + \frac{a}{r}(\cos F - k\beta)\sin F\right] \tag{2.115}$$

The averaged variational equations for the averaged equinoctial elements and the averaged Lagrange multipliers are now sought. This averaging procedure is like the one used in the DSST development in section 2.1.2.6 in that the short period motion, i.e.

periodic motion on the order of one orbital revolution, is averaged out of the equations of motion. The following averaging procedure is taken from references (4) and (55). First, the averaged Hamiltonian is formed. From the averaged Hamiltonian, a first-order approximation to the state and costate is derived by holding these quantities constant over the averaging interval of one orbital revolution. Only the eccentric longitude, *F*, is varied on the orbit (4):

$$\widetilde{H} = \frac{1}{T_0}\int_0^{T_0} H\,dt = \frac{1}{T_0}\int_{-\pi}^{\pi} \frac{H\,dF}{\dot{F}(\widetilde{\mathbf{z}}, F)} \tag{2.116}$$

The integrand in equation (2.116) is the *Hamiltonian* from equation (2.83). Here, T_0 is the orbital period at time *t* which is given by $T_0 = 2\pi/\widetilde{n}$ with $\widetilde{n} = \mu^{1/2}\widetilde{a}^{-3/2}$. The symbol, $\widetilde{a}$, denotes the averaged value of the semimajor axis at time *t*. From Kepler's equation, $\widetilde{\lambda} = F - \widetilde{k}\sin F + \widetilde{h}\cos F = \widetilde{n}\,t$, we have (4):

$$\dot{F} = \frac{2\pi}{T_0(1 - \widetilde{k}\cos F - \widetilde{h}\sin F)} \tag{2.117}$$

Defining the variable *s* as follows and making a substitution into the Euler-Lagrange equations yields (4):

$$s = \frac{\left(\frac{1}{T_0}\right)}{\dot{F}} = \frac{1}{2\pi}\left[1 - \widetilde{k}\cos F - \widetilde{h}\sin F\right] \tag{2.118}$$

$$\dot{\widetilde{\mathbf{z}}} = \left(\frac{\partial \widetilde{H}}{\partial \widetilde{\boldsymbol{\lambda}}_{\mathbf{z}}}\right)^T = \int_{-\pi}^{\pi}\left(\frac{\partial H}{\partial \widetilde{\boldsymbol{\lambda}}_{\mathbf{z}}}\right)^T s(\widetilde{\mathbf{z}}, F)\,dF \tag{2.119}$$

$$\dot{\widetilde{\boldsymbol{\lambda}}}_{\mathbf{z}} = -\left(\frac{\partial \widetilde{H}}{\partial \widetilde{\mathbf{z}}}\right)^T = -\int_{-\pi}^{\pi}\left[\left(\frac{\partial H}{\partial \widetilde{\mathbf{z}}}\right)^T s(\widetilde{\mathbf{z}}, F) + H\left(\frac{\partial s}{\partial \widetilde{\mathbf{z}}}\right)^T\right]dF \tag{2.120}$$

Here, the averaged equinoctial element vector is $\widetilde{\mathbf{z}} = \left\{\widetilde{a}, \widetilde{h}, \widetilde{k}, \widetilde{p}, \widetilde{q}, \widetilde{\lambda}\right\}$, and the averaged vector containing the Lagrange multipliers is $\widetilde{\boldsymbol{\lambda}}_{\mathbf{z}} = \left\{\widetilde{\lambda}_a, \widetilde{\lambda}_h, \widetilde{\lambda}_k, \widetilde{\lambda}_p, \widetilde{\lambda}_q, \widetilde{\lambda}_\lambda\right\}$.

The partials derivatives of *s* with respect to the averaged equinoctial elements are (4):

$$\frac{\partial s}{\partial \widetilde{a}} = 0 \tag{2.121}$$

$$\frac{\partial s}{\partial \widetilde{h}} = \frac{-\sin F}{2\pi} \tag{2.122}$$

$$\frac{\partial s}{\partial \widetilde{k}} = \frac{-\cos F}{2\pi} \tag{2.123}$$

$$\frac{\partial s}{\partial \widetilde{p}} = \frac{\partial s}{\partial \widetilde{q}} = 0 \tag{2.124}$$

$$\frac{\partial s}{\partial \widetilde{\lambda}} = \frac{\widetilde{k}\sin F - \widetilde{h}\cos F}{2\pi(1 - \widetilde{k}\cos F - \widetilde{h}\sin F)} \tag{2.125}$$

Using the previous equations, the averaged variational equations for the equinoctial elements and the Lagrange multipliers with constant acceleration, f_t, are (4):

$$\dot{\widetilde{a}} = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} \left(\frac{\partial a}{\partial \dot{\mathbf{r}}}\right)^T \cdot \hat{\mathbf{u}}(1 - \widetilde{k}\cos F - \widetilde{h}\sin F)\, dF \tag{2.126}$$

$$\dot{\widetilde{h}} = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} \left(\frac{\partial h}{\partial \dot{\mathbf{r}}}\right)^T \cdot \hat{\mathbf{u}}(1 - \widetilde{k}\cos F - \widetilde{h}\sin F)\, dF \tag{2.127}$$

$$\dot{\widetilde{k}} = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} \left(\frac{\partial k}{\partial \dot{\mathbf{r}}}\right)^T \cdot \hat{\mathbf{u}}(1 - \widetilde{k}\cos F - \widetilde{h}\sin F)\, dF \tag{2.128}$$

$$\dot{\tilde{p}} = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} \left(\frac{\partial p}{\partial \dot{\mathbf{r}}} \right)^T \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF \tag{2.129}$$

$$\dot{\tilde{q}} = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} \left(\frac{\partial q}{\partial \dot{\mathbf{r}}} \right)^T \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF \tag{2.130}$$

$$\begin{aligned} \dot{\tilde{\lambda}} = & \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} \left(\frac{\partial \lambda}{\partial \dot{\mathbf{r}}} \right)^T \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF + \\ & \frac{1}{2\pi} \int_{-\pi}^{\pi} \tilde{n} (1 - \tilde{k} \cos F - \tilde{h} \sin F) dF \end{aligned} \tag{2.131}$$

$$\begin{aligned} \dot{\tilde{\lambda}}_a = & \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} - \tilde{\boldsymbol{\lambda}}_{\mathbf{z}}^T \frac{\partial M}{\partial \tilde{a}} \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF + \\ & \frac{1}{2\pi} \int_{-\pi}^{\pi} - \tilde{\lambda}_\lambda \frac{\partial \tilde{n}}{\partial \tilde{a}} \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF \end{aligned} \tag{2.132}$$

$$\dot{\tilde{\lambda}}_h = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} - \tilde{\boldsymbol{\lambda}}_{\mathbf{z}}^T \frac{\partial M}{\partial \tilde{h}} \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF \tag{2.133}$$

$$\dot{\tilde{\lambda}}_k = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} - \tilde{\boldsymbol{\lambda}}_{\mathbf{z}}^T \frac{\partial M}{\partial \tilde{k}} \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF \tag{2.134}$$

$$\dot{\tilde{\lambda}}_p = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} - \tilde{\boldsymbol{\lambda}}_{\mathbf{z}}^T \frac{\partial M}{\partial \tilde{p}} \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF \tag{2.135}$$

$$\dot{\tilde{\lambda}}_q = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} - \tilde{\boldsymbol{\lambda}}_{\mathbf{z}}^T \frac{\partial M}{\partial \tilde{q}} \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF \tag{2.136}$$

$$\dot{\tilde{\lambda}}_\lambda = \frac{1}{2\pi} f_t \int_{-\pi}^{\pi} - \tilde{\boldsymbol{\lambda}}_{\mathbf{z}}^T \frac{\partial M}{\partial \tilde{\lambda}} \cdot \hat{\mathbf{u}} (1 - \tilde{k} \cos F - \tilde{h} \sin F) \, dF \tag{2.137}$$

The thrust direction, $\hat{\mathbf{u}}$, is a function of the averaged equinoctial elements, $\tilde{\mathbf{z}}$, the eccentric longitude, *F*, and the averaged Lagrange multipliers, $\tilde{\boldsymbol{\lambda}}_{\mathbf{z}}$. The thrust direction is chosen so it is always parallel to $\tilde{\boldsymbol{\lambda}}_{\mathbf{z}} M(\tilde{\mathbf{z}}, F)$. This optimizes the acceleration direction

according to the necessary conditions derived in sections 2.2.2.1 and 2.2.2.2. The partial derivatives of the *M* matrix with respect to the equinoctial elements are provided in reference (4). The *M* matrix is formulated with respect to mean longitude while the B^L matrix defined earlier in equation (2.82a) is defined with respect to true longitude. The exact and averaged formulations of the mean longitude equations are presented in reference (4).

In the averaged formulation, the averaged Hamiltonian can be obtained from the integrated variables (4):

$$\widetilde{H} = \widetilde{\lambda}_a \dot{\widetilde{a}} + \widetilde{\lambda}_h \dot{\widetilde{h}} + \widetilde{\lambda}_k \dot{\widetilde{k}} + \widetilde{\lambda}_p \dot{\widetilde{p}} + \widetilde{\lambda}_q \dot{\widetilde{q}} + \widetilde{\lambda}_\lambda \dot{\widetilde{\lambda}} \tag{2.138}$$

When given the initial and final orbits, one can solve for the averaged Lagrange multipliers, $\widetilde{\boldsymbol{\lambda}}_z$, using the same two-point boundary value problem defined in section 2.2.2.2 and the numerical shooting method described in section 2.2.2.3. The major difference in the averaged and exact problems is that the averaged problem requires quadrature to produce the averaged time derivatives of the equinoctial elements and Lagrange multipliers. It is perhaps possible to analytically evaluate the quadrature expressions in equations (2.126-2.137), but numerical quadrature was used by Kechichian and was used in this thesis. The usefulness of the averaged Lagrange multipliers comes from the fact that, in practice, the averaged two-point boundary value problem is much more robust when given errors in the initial guesses for the Lagrange multipliers. The solution for the Lagrange multipliers obtained using the shooting

method with the averaged two-point boundary value problem can be used as initial guesses for the exact solution which uses the non-averaged variational equations as defined in section 2.2.2.2. This strategy was suggested by Jean A. Kechichian in his papers referenced throughout chapter 2. In section 5, this strategy will be shown to work well for the test cases done for this thesis.

2.2.3 Electric Propulsion for Satellite Station Keeping

Hundreds of telecommunications satellites in Geosynchronous Earth Orbit (GEO) provide services that enable television, radio, telephony, and other communication across most of the globe. Arthur C. Clark popularized the concept of GEO satellites which maintain a nearly constant position in the sky from the perspective of an Earth-based observer. GEO satellites follow circular, Earth equatorial orbits with a mean semimajor axis of approximately 42,164 km. This orbit gives the satellite an orbital period equal to one day. However, maintaining a true geostationary orbit requires that a satellite precisely counteract Earth's nonspherical gravity, lunar and solar gravity and solar radiation pressure. All of these forces, though small in comparison to the primary two-body gravitational attraction of the Earth on the satellite, act to move the satellite from its ideal geostationary orbit. These forces change the orbital semimajor axis, eccentricity and inclination (42). Counteracting the small forces requires the use of thrusters. In particular, inclination control for geostationary satellites requires 95% of the needed ΔV, i.e. change in orbital velocity (59), (43).

Since the 1960s, when GEO satellites were first launched, such satellites have used chemical thrusters. However, in 1997, the first commercial spacecraft to use Xenon ion engines for station keeping was launched (43). This spacecraft was the Hughes/Boeing 601 spacecraft which used Xenon ion thrusters for orbital inclination control. Later, the Hughes/Boeing 702 spacecraft introduced Xenon ion thrusters (XIPS) for semimajor axis, eccentricity and inclination control (43).

The application of Xenon ion electric propulsion (EP) to GEO satellites has mainly been driven by the reduced fuel mass required by Xenon over the mass required by chemical thruster fuel such as hydrazine for comparable mission lifetimes. The savings in mass can be as much as a factor of ten. The efficiency improvement is achieved in part because electric, Xenon fuel-based ion engines can provide a specific impulse that is ten times greater than that of chemical thrusters and the efficiency of such thrusters is optimal for high specific impulse (43), (38). Specific impulse is proportional to the exit velocity of the engine exhaust. Low-thrust characteristics of Xenon ion propulsion require longer engine burn durations. A typical bipropellant, chemical thruster with a force of 22 Newtons need only be used on the order of once every several days to maintain acceptable semimajor axis, eccentricity and inclination control. A Xenon ion engine with a force of 0.1 Newtons must be operated every day for several hours at specific locations in the orbit in order to efficiently maintain acceptable control (59), (43). Acceptable control is defined as maintaining the semimajor axis and eccentricity so the satellite's longitude stays within 0.05 degrees of its assigned slot and maintaining the inclination under 0.1 degrees (60).

Aside from the fuel mass benefit of Xenon ion propulsion systems is the capability to maintain tighter control of the satellite. Because the thrusters are operated during every orbit, thrusts made with ion engines to counteract perturbing forces can be executed more frequently and efficiently than thrusts made with chemical thrusters. Figure 2.7 taken from reference (43) shows the tighter inclination control that can be maintained with ion thrusters over chemical thrusters.

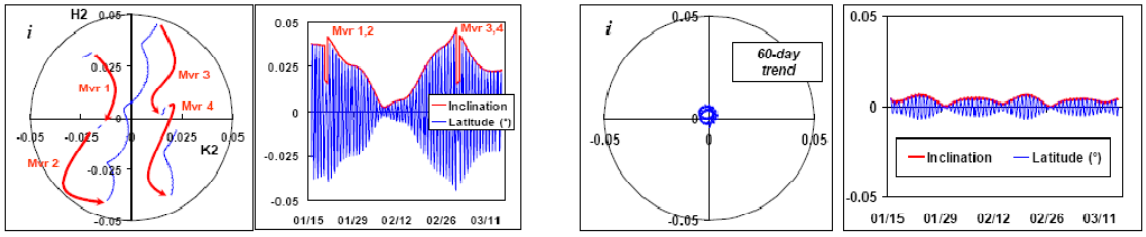

Figure 2.7 Inclination Control with Chemical vs. XIPS Thrusters

Figure 2.8 was also taken from reference (43) and shows the tighter eccentricity and longitude drift control that can be maintained using ion thrusters.

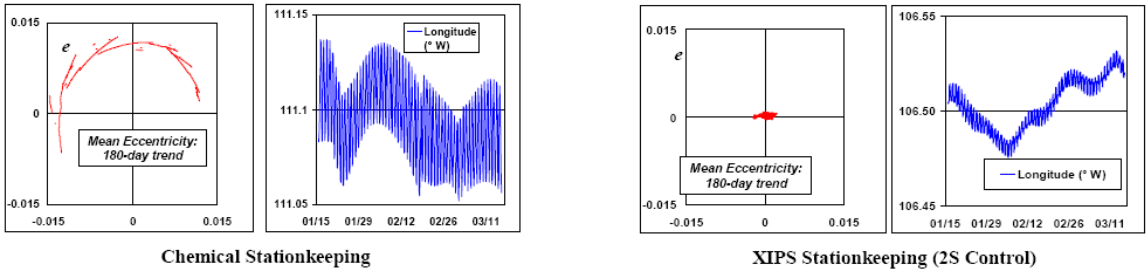

Figure 2.8 Eccentricity and Longitude Control with Chemical vs. XIPS Thrusters

The advantages of tighter inclination, eccentricity and longitude control mainly benefit Earth-based users of the spacecraft. Requirements and therefore costs for antenna pointing are reduced with tighter satellite control because the satellite will wander less from its fixed position relative to the Earth's surface.

2.3 Recursive Orbit Estimation Techniques

Satellite orbital estimation is the problem of solving for the constants associated with the orbital equations of motion using observations of a satellite while in its orbit. In satellite orbit estimation, one typically tries to obtain an optimal estimate. However, the optimal estimate is often difficult to achieve exactly due to the nonlinear characteristics of the orbital dynamics involved. In order to perform optimal estimation, one requires knowledge of an initial satellite orbit, the orbital dynamics, the measurement dynamics and the associated errors for each set of dynamics. Orbital and continuous thrust dynamics were discussed in sections 2.1 and 2.2, respectively. The measurement dynamics are not discussed as fully in this document, but ample treatment of several types of observations can be found in *Methods of Orbit Determination* by Pedro Escobal (61) and in *An Introduction to the Mathematics and Methods of Astrodynamics* by Richard Battin (11). The *GTDS Mathematics Specification* (32) also has a great deal of useful material on orbit determination. The orbital dynamics and measurement errors are important for the study and scope of this thesis and so are discussed in this section as they relate to orbital estimate errors. Documents that also present these issues are references (62) and (63).

A number of estimation algorithms have been applied to the satellite orbit estimation problem. Among them are various variations on the Kalman Filter. The discussion in this document will focus on a few of these including the Extended Kalman Filter (EKF) derived from Kalman Filter introduced by R.E. Kalman (64), the Unscented Kalman Filter (65), (66), and the Backward Smoothing Extended Kalman Filter (BSEKF)

(2). Because the equations describing satellite motion and the geometry of the measurements taken on satellites are nonlinear, steps are taken in the derivations of the aforementioned estimators to approximate the dynamics and measurement equations with linear equations. Linear equations are necessary because dynamics and measurement equations are assumed linear in the derivations of the most popular and efficient optimal estimation algorithms, i.e. Kalman filters (67). Allowing for nonlinear dynamics and measurement equations would mean many of the basic assumptions and techniques used in deriving the Kalman filter would not be valid. Some of these assumptions and techniques include linear algebra and Gaussian normal distributions of errors. A fundamental assumption allowing the use of linear techniques to solve estimation problems is that the Linear Least Squared Error (LLSE) estimator and the Bayesian Least Squared Error (BLSE) estimator, i.e. the Minimum Mean Square Error (MMSE) estimator, are equivalent when process noise and measurement noise statistics are independent, identically distributed (i.i.d) Gaussian distributions. (68), (69), (70). As long as the dynamic and measurement equations transform the associated errors linearly, the LLSE and BLSE equivalence assumption holds (69).

The orbit estimation problem is a specific problem of state estimation. There are two ways in which a state can be estimated. The non-Bayesian or Fisher approach is a nonrandom approach which tries to estimate an unknown constant (69). The random or Bayesian approach treats the state parameters as a vector of random variables, **x**, with a *prior* probability density function, $p(\mathbf{x})$. The Bayesian approach starts with a *prior*

probability density function (PDF) of the state vector and one can obtain the *posterior* PDF using Bayes' formula (69):

$$p(\mathbf{x}|Z) = \frac{p(Z|\mathbf{x})\,p(\mathbf{x})}{P(Z)} = \frac{1}{c}\,p(Z|\mathbf{x})\,p(\mathbf{x}) \tag{2.139}$$

Here, c is simply a normalizing constant not dependant on $\mathbf{x}$. The result is a PDF describing the probability of values of $\mathbf{x}$ given the observations, Z. The non-Bayesian approach doesn't use the *prior* PDF of $\mathbf{x}$. Rather, it simply uses the likelihood function which is a PDF of the measurements, Z, conditioned on the parameter vector, $\mathbf{x}$ (69):

$$\Lambda_Z(\mathbf{x}) = p(Z|\mathbf{x}) \tag{2.140}$$

This function can also be used as a measure of how *likely* is a realization of values in $\mathbf{x}$ given the obtained observations and serves as a measure of evidence from the observed data (69). However, because the likelihood function doesn't use the full Bayesian formula, it is non-Bayesian.

The estimators based on these approaches are the non-Bayesian Maximum Likelihood Estimator (ML) and the Bayesian Maximum A Posteriori Estimator (MAP). The ML estimator maximizes the likelihood function as follows (69):

$$\hat{\mathbf{x}}^{ML}(Z) = \arg\max_{\mathbf{x}} \Lambda_Z(\mathbf{x}) = \arg\max_{\mathbf{x}} p(Z \mid \mathbf{x}) \tag{2.141}$$

The ML estimator finds the *mean* of the PDF. The MAP estimator also uses a *prior* PDF of $\mathbf{x}$ and follows from the maximization of the posterior PDF from Bayes' formula (69):

$$\hat{\mathbf{x}}^{MAP}(Z) = \arg\max_{\mathbf{x}} p(\mathbf{x} \mid Z) = \arg\max_{\mathbf{x}} [\,p(Z \mid \mathbf{x})\,p(\mathbf{x})] \tag{2.142}$$

The MAP estimator finds the *mode* of the posterior PDF. The ML and MAP estimators both depend on observations, Z, but the MAP estimator also depends on a realization of $\mathbf{x}$

which is a random variable (69). The non-Bayesian approach is really a degenerate case of the Bayesian approach. If one looks at a *prior* which is a Gaussian PDF with a variance that approaches infinity, the PDF will approach that of a uniform distribution (69):

$$\lim_{\sigma \longrightarrow \infty} p(\mathbf{x}) = \mathbf{x} \tag{2.143}$$

In this case, the MAP estimate and the ML estimate coincide because the MAP estimate becomes proportional to the ML estimate (69).

Another non-random estimator is the least squares estimator (LSE). If one is given scalar and nonlinear measurements (69):

$$z(j) = h(j, \mathbf{x}) + w(j) \qquad j = 1,..., k \tag{2.144}$$

the LSE of the vector, **x**, is obtained by (69):

$$\hat{\mathbf{x}}^{LS}(k) = \arg\min_{\mathbf{x}} \left\{ \sum_{j=1}^{k} \left[z(j) - h(j, \mathbf{x}) \right]^2 \right\}$$

Here, if the measurement errors are Gaussian, i.e. $w(j) \approx N(0, \sigma^2)$, then the least squares estimator coincides with the maximum likelihood estimator (MLE) described previously (69).

For random parameters, the counterpart of the LS estimator is the minimum mean square error (MMSE) estimator (69):

$$\hat{\mathbf{x}}^{MMSE}(Z) = \arg\min_{\hat{\mathbf{x}}} E[(\hat{\mathbf{x}} - \mathbf{x})^2 \mid Z] \tag{2.145}$$

This equation finds the value of the state estimate, $\hat{\mathbf{x}}$, that minimizes the expectation or the square of the error in the estimate. The solution to this estimator is the *conditional mean* of **x**:

$$\hat{\mathbf{x}}^{MMSE}(Z) = E[\mathbf{x} \mid Z] \equiv \int_{-\infty}^{\infty} \mathbf{x}\, p(\mathbf{x} \mid Z) d\mathbf{x} \tag{2.146}$$

Because the *mean* and *mode* of a Gaussian *posterior* distribution are equal, the MMSE and MAP estimates are equal when given a Gaussian *posterior* PDF.

The variances of the MAP and ML estimators are not equal because the MAP estimate variance also includes the *prior* PDF. The definition of the estimator's variance is the expected value of the square of the estimation error (69):

$$E[(\hat{\mathbf{x}} - \mathbf{x})^2] = \text{var}(\hat{\mathbf{x}}) \tag{2.147}$$

This quantity provides a measure of the accuracy of the estimator. Often, the estimation error is assumed to be Gaussian (69). In the Kalman Filter derivation in section 2.3.1, the minimum variance estimator for the orbit estimation problem is derived. The ML and MAP estimators described above can also be used to derive an optimal estimator for the orbit estimation problem. In fact, reference (70) shows that due to the Gaussian statistics assumed for measurement and process noise, the ML, MAP and minimum variance estimators are equivalent.

The estimation techniques described in section 2.3 make use of the i.i.d Gaussian error assumption, but also operate with the crucial flaw that the approximate linear dynamic and measurement models differ from the actual nonlinear physics occurring.

The Extended Kalman Filter (EKF), for example, works well for systems with a small degree of nonlinearity. For systems with higher degrees of nonlinearity, smoothing and other methods can improve the accuracy of the estimators, but this is not guaranteed (68). The problem of how to ascertain the radius of convergence and to guarantee convergence for *extended* or *linearized* estimators has an elusive solution.

2.3.1 Extended Kalman Filter

A filter is an estimation algorithm that uses *physically realizable* data. These data are observations that have already been taken so the interval of these observations is $[0,t_k]$. The algorithm is called a filter because it is meant to *filter out* the noise in the available signals (71).

The following derivations were taken verbatim from the *R&D GTDS Filter Program Software Specification and User's Guide* by J. Dunham (72), i.e. sections 2.3.1.1-2.3.1.5. Comparison for accuracy was done with reference (70). Some details regarding covariance properties were published online by Richard Duda (73) and the section on the expectation operator, 2.3.1.1.7, is from *Optimal Control and Estimation* by Robert Stengel (71).

The orbit problem is one in which several conditions apply. The equations of motion are nonlinear. The equations describing the observations are nonlinear functions of the variables describing the satellite state. There is a wealth of data; considerably more than is needed for a deterministic solution and much more than is the case in the

typical binary star orbit determination from celestial mechanics. Neither the dynamics nor the observations can be perfectly modeled. These characteristics determine the filtering requirements and algorithms needed to solve the problem.

Given a set of observations and an a priori estimate of the spacecraft solve-for parameters; an improved knowledge of them is to be determined. The solve-for parameters, $X(t)$, are an nx1 vector which may include the position and velocity of the spacecraft, constants from the equations of motion, attitude parameters, and clock parameters. Orbital elements or spherical coordinates may be estimated instead of the spacecraft position and velocity. The solve-for parameter (state vector) differential equation

$$\dot{X} = F(X, t) \tag{2.148}$$

is a set of n simultaneous equations, which are nonlinear.

These parameters are not observed directly, but they can be inferred from the observations. The observations, $Y(t_i)$, can be expressed as a function of the solve-for parameters and time, i.e. $G[X(t), t]$. This observation equation is, in general, nonlinear in the solve-for parameters.

For a solution to be possible, at least as many observations are needed as there are solve-for parameters. That is, for a set of l observations, l must be greater than or equal to n, where n is the number of parameters to be estimated.

2.3.1.1 Linear Unbiased Minimum Variance Batch Estimate

2.3.1.1.1 The Linearized State Equation

The predicted solve-for vector at time t is denoted $X^*(t)$. If the true parameters at that time are $X(t)$, then

$$\begin{aligned} X(t) &= X^*(t) + x(t) \\ \dot{X}(t) &= \dot{X}^*(t) + \dot{x}(t) \end{aligned} \tag{2.149}$$

where $x(t)$ is the nx1 correction vector which needs to be estimated. If the initial guess for the parameters are "close" to the true value, the correction vector $x(t)$, is small relative to $X(t)$. Under that condition, a linear differential equation can be obtained for the propagation of the correction vector. This is done by taking the differential equation for the solve-for vector (Equation 2.148) and expanding it in a Taylor series about $X^*(t)$ as follows:

$$\begin{aligned} \dot{X}(t) &= F(X^*(t) + x(t), t) \\ &= F(X^*(t), t) + \left.\frac{\partial F}{\partial X}\right|_{X=X^*(t)} x(t) + \ldots \end{aligned} \tag{2.150}$$

Truncating equation (2.150) yields

$$\dot{X}(t) = \dot{X}^*(t) + \left[\frac{\partial F}{\partial X}\right]_{X^*} x(t) \tag{2.151}$$

where $\dot{X}^*(t) = F(X^*, t)$. Rearranging equation (2.151) yields.

$$\dot{X}(t) - \dot{X}^*(t) = \left[\frac{\partial F}{\partial X}\right]_{X^*} x(t) \tag{2.151}$$

or, using equation (2.149) in differential form,

$$\dot{x}(t) = \left[\frac{\partial F}{\partial X}\right]_{X^*} x(t) \tag{2.152}$$

Equation (2.152) is a set of *n* linearized differential equations for the propagation of the correction vector. This equation can be written as

$$\dot{x}(t) = A(t)x(t) \tag{2.153}$$

where $A(t)$ is the nxn matrix of partial derivatives evaluated along the trajectory $X^*(t)$, given by:

$$A(t) = \left[\frac{\partial F}{\partial X}\right]_{X^*} \tag{2.154}$$

2.3.1.1.2 State Transition Matrix

The differential equations for the propagation of the correction vector, i.e. equation (2.153), form a system of n homogeneous linear differential equations. It is assumed that the solution has the form:

$$x(t_1) = \Phi(t_1, t_0)x(t_0) \tag{2.155}$$

where $\Phi(t_1, t_0)$ is an *n*x*n* matrix called the state transition matrix. The state transition matrix relates perturbations ($x(t_1)$) in the state vector at time t_1 to perturbations ($x(t_0)$) at time t_0. At an arbitrary time *t*, the following results:

$$x(t) = \Phi(t, t_0)x(t_0) \tag{2.156}$$

The $x(t_0)$ are constants of the integration of equation (2.153). They are the perturbations in the state vector, *X*, at the epoch time t_0.

To find the differential equation for the state transition matrix, equation (2.156) is differentiated:

$$\dot{x}(t) = \dot{\Phi}(t, t_0)x(t_0) + \Phi(t, t_0)\dot{x}(t_0) \tag{2.157}$$

Using equation (2.153) and the fact that $\dot{x}(t_0) = 0$ (since the $x(t_0)$ are constants) produces:

$$A(t)x(t) = \dot{\Phi}(t, t_0)x(t_0) \tag{2.158}$$

Substituting equation (2.156) for $x(t)$ and rearranging the result produces the following expression for the state transition matrix differential equation:

$$\dot{\Phi}(t, t_0) = A(t)\Phi(t, t_0) \tag{2.159}$$

The initial conditions for integration of equation (2.159) can be found from considering the state transition matrix over a zero-length time interval. In this case,

$$x(t_0) = \Phi(t_0, t_0)x(t_0) \tag{2.160}$$

and $\Phi(t_0, t_0)$ can be assumed to be the identity matrix.

2.3.1.1.3 Properties of the State Transition Matrix

The state transition matrix has a number of properties which can be employed to advantage in estimation work. This subsection summarizes a few of them for use in later discussion. The state transition matrix, $\Phi(t\ , t_0)$, relates perturbations in the state vector at time t to perturbations at time t_0. The matrix obeys the following differential equation:

$$\dot{\Phi}(t, t_0) = A(t)\Phi(t, t_0) \tag{2.161}$$

where $A(t)$ is the nxn matrix of partial derivatives defined by equation (2.154) and $\Phi(t_0,t_0) = \mathrm{I}$, the identity matrix. The following properties of the state transition matrix can also be obtained:

$$\Phi(t_1, t_0) = \Phi^{-1}(t_0, t_1) \tag{2.162}$$

$$\Phi(t_1, t_0) = \frac{\partial x(t_1)}{\partial x(t_0)} \tag{2.163}$$

$$\Phi(t_2, t_0) = \Phi(t_2, t_1)\Phi(t_1, t_0) \tag{2.164}$$

If the quantity $x(t_2)$ is known, the property of equation (2.162) can be used to obtain $x(t_1)$. Since

$$x(t_2) = \Phi(t_2, t_1)x(t_1) \tag{2.165}$$

then

$$x(t_1) = \Phi^{-1}(t_2, t_1)x(t_2) \tag{2.166a}$$

or, from equation (2.162),

$$x(t_1) = \Phi(t_1, t_2)x(t_2) \tag{2.166b}$$

As a practical matter, it is usually easier to integrate backwards in time to obtain the state transition matrix from t_2 to t_1 instead of integrating forward in time from $t_1$to t_2 and then inverting the state transition matrix. Computing $\Phi^{-1}(t_2, t_1)$ involves first the integration from t_1 to t_2 and then the inversion of an nxn matrix. If $X(t_1)$ and/or $x(t_1)$ is not known or if n is a large number, finding $\Phi^{-1}(t_2, t_1)$ can be an expensive computation. The differential equation for Φ in (2.161) can be integrated from t_2 to t_1 to obtain $\Phi(t_1, t_2)$ directly, if the initial conditions $x(t_2)$ and $X^*(t_2)$ are available.

2.3.1.1.4 The Linearized Observation Equation

The observation $Y(t_i)$ can be represented as a function of the state and time as follows:

$$Y(t_i) = G(X(t_i), t_i) + \varepsilon_i \tag{2.167}$$

where ε_i is the error in the observation at time t_i. Equation (2.167) can be linearized in a similar manner as the state equation. Substituting equation (2.149) for $X(t_i)$ and expanding in a Taylor series about $X^*(t_i)$ gives

$$Y(t_i) = G(X^*(t_i), t_i) + \left[\frac{\partial G}{\partial X}\right]_{X^*_i} x(t) + \ldots + \varepsilon_i \tag{2.168}$$

The observation residual, $y(t_i)$, is defined as

$$y(t_i) = Y(t_i) - G(X^*(t_i), t_i) \tag{2.169}$$

This is the observed-minus-computed (O-C) observation residual based on the state estimate, $X^*(t_i)$.

Linearizing equation (2.168) and substituting the result into equation (2.169) yields the following equation for the O-C residuals:

$$y(t_i) = \left[\frac{\partial G}{\partial X}\right]_{X^*(t_i)} x(t_i) + \varepsilon(t_i) \tag{2.170}$$

The matrix $H(t_i)$ is defined as

$$H(t_i) = \left[\frac{\partial G}{\partial X}\right]_{X^*(t_i)} \tag{2.171}$$

For a single observation, $H(t_i)$ is a 1xn matrix of partial derivatives of the observation equation with respect to the state parameters. Then equation (2.169) can be written as:

$$y(t_i) = H(t_i)x(t_i) + \varepsilon(t_i) \tag{2.172}$$

2.3.1.1.5 Summary of Notation

The following summarizes the previously developed equations and notation. A set of scalar observations $Y(t_i)$ exists at times t_i (i = 1,2,…,l). (In the next subsection, the more general case of vector observations is considered.) The state vector is the nx1 vector of independent parameters to be estimated. Thus,

$X^*(t_i)$ = the nx1 predicted state vector at time t_i

$Y(t_i)$ = the observation at time t_i

And

$$\dot{X}^*(t_i) = F(X^*, t_i) \tag{2.173}$$

is the state differential equation of motion. The function

$$G[X^*(t_i), t_i] \tag{2.174}$$

is the nonlinear expression which predicts the observation at time t_i as a function of the predicted state vector at t_i. The matrix

$$A(t_i) = \left[\frac{\partial F}{\partial X}\right]_{X^*(t_i)} \tag{2.175}$$

contains the partial derivatives of the equations of motion. The vector

$$H(t_i) = \left[\frac{\partial G(X^*, t_i)}{\partial X}\right]_{X^*(t_i)} \tag{2.176}$$

contains the partial derivatives of the observation at time t_i with respect to the components of the state vector at time t_i (a 1xn vector).

The state deviation equation, or the equation of motion for the state correction vector, is given by:

$$\dot{x}(t) = A(t)x(t) \tag{2.177}$$

with the solution

$$x(t_i) = \Phi(t_i, t_k)x(t_k) \tag{2.178}$$

when integrated from t_k to t_i.

Finally, the equation describing the O-C observation residual is

$$y(t_i) = H(t_i)x(t_i) + \varepsilon(t_i) \tag{2.179}$$

2.3.1.1.6 Reduction to a Common Epoch

A correction is needed to the state at some epoch time t_k. The correction is to be determined from a set of l observations, Y, made at times t_i (i = 1,2,…,l). The time t_k may be within the span t_1 to t_l, earlier than the span, or later.

The state correction vector appearing in equation (2.179), $x(t_i)$, is related to the state correction vector at epoch $x(t_k)$, according to equation (2.178). Replacing $x(t_i)$ in equation (2.179) yields:

$$y(t_i) = H(t_i)\Phi(t_i, t_k)x(t_k) + \varepsilon(t_i) \tag{2.180}$$

This set of l observational equations can be written as a vector equation

$$y = Hx(t_k) + \varepsilon \tag{2.181}$$

where

$$y = \begin{bmatrix} y_1 \\ .. \\ y_l \end{bmatrix}$$

is an lx1 vector of residuals

$$H = \begin{bmatrix} H_1\Phi(t_1, t_k) \\ ... \\ H_l\Phi(t_l, t_k) \end{bmatrix}$$

is an $l$xn matrix, and

$$\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \dots \\ \varepsilon_l \end{bmatrix}$$

is an lx1 vector. The H matrix contains the partial derivatives of the observational equations at their observed times, t_i, with respect to the n components of the state vector at epoch, t_k.

2.3.1.1.7 The Expectation Operator

The average or expected value of a random variable v is defined as:

$$E(v) = \sum_{i=1}^{\infty} v_i \, probability(v_i) = v' \tag{2.182}$$

$E(v)$ is also called the first moment about the origin or the mean value of v, and it is denoted by v'. This is a measure of a value toward which a large number of observations of v tends.

Higher moments provide measures of the variability of x, and the nth moments of discrete variables are defined by

$$E(v^n) = \sum_{i=1}^{\infty} v_i^n \, probability(v_i) \tag{2.183}$$

Higher moments about the origin reflect not only variation about the mean but variation in the mean value itself. The variation about the mean is useful if isolated which leads to the nth central moments for discrete variables:

$$E[(v - v')^n] = \sum_{i=1}^{\infty} [(v - v')^n \, probability(v_i)] \tag{2.184}$$

The second central moment or variance is defined when *n* is 2.

$$\sigma^2 = E[(v - v')^2] \tag{2.185}$$

The square root of the variance is the standard deviation, σ. The expected value of the product of the deviations of two random variables v_1 and v_2 is called the covariance, *P*, and is expressed as

$$P = E[(v_1 - v_1')(v_2 - v_2')] \tag{2.186}$$

The covariance has several important properties:

- If variable v_1 and variable v_2 tend to increase together, then $P(1,2) > 0$
- If variable v_1 tends to decrease when variable v_2 increases then $P(1,2) < 0$
- If variable v_1 and variable v_2 are independent, then $P(1,2) = 0$
- $| P(1,2) | <= \sigma_1 \sigma_2$, where σ_i is the standard deviation of variable v_i
- $P(1,1) = \sigma_1^2$

Thus, the covariance measures the dependence between variable v_1 and v_2. If the covariance value for the two variables is 0, the variables are independent.

2.3.1.1.8 The Linear Unbiased Minimum Variance Estimate

The linear unbiased minimum variance estimate of the state correction vector, x, at the epoch time, t_k, is $\hat{x}(t_k)$. The best estimate of the state vector at epoch is then $X^*(t_k) + \hat{x}(t_k)$. The estimate, $\hat{x}(t_k)$, is linearly related to the vector of observation residuals, y, as follows:

$$\hat{x}(t_k) = My \tag{2.187}$$

The matrix M will be shown to be a combination of the observation partial derivatives and the observational error covariance which is selected to choose the best estimate. The best estimate is one for which the expectation function that defines $\hat{x}(t_k)$ contains x explicitly and not as an argument of another function.

The requirement that the estimate be unbiased may be stated as:

$$E[\hat{x}] = x \tag{2.188}$$

Substituting equations (2.181) and (2.187) into equation (2.188) gives the following requirement that the estimate be unbiased:

$$E[M(Hx_k + \varepsilon)] = x_k \tag{2.189}$$

The observation errors are treated as zero-mean variables. Thus, the following assumptions are made:

$$E[\varepsilon] = 0 \tag{2.190a}$$

and

$$E[\varepsilon\varepsilon^{\mathrm{T}}] = \begin{bmatrix} R_1 & 0 & 0 & 0 & \dots & 0 \\ 0 & R_2 & 0 & 0 & \dots & 0 \\ 0 & 0 & R_3 & 0 & \dots & 0 \\ . & . & . & . & & . \\ . & . & . & . & & . \\ 0 & 0 & 0 & 0 & \dots & R_l \end{bmatrix} = R \tag{2.190b}$$

where ε is the vector of the observation error given by $\varepsilon^{\mathrm{T}} = [\varepsilon(t_1), \varepsilon(t_2), \dots, \varepsilon(t_l)]$, and

$$R_i = E[\varepsilon(t_i)\varepsilon^{\mathrm{T}}(t_i)] = \sigma^2(t_i) \tag{2.191}$$

The fact that the off-diagonal elements of the matrix in equation (2.190b) are zero is a result of the assumption that the observational error at time t_i is completely independent of the error at any other time. This assumption is termed stochastic independence. Since the expectation of observation errors, $E[\varepsilon]$, is assumed to be zero, equation (2.189) reduces to

$$MHx_k = x_k \text{ or } MH = I \tag{2.192}$$

The covariance matrix, P_k, is defined as

$$P_k = E\left[\{\hat{x}_k - E[\hat{x}_k]\}\{\hat{x}_k - E[\hat{x}_k]\}^{\mathrm{T}}\right] \tag{2.193}$$

The minimum variance requirement is equivalent to choosing $\hat{x}_k$ to minimize P_k.

To determine the value of $\hat{x}_k$ which minimizes P_k, P_k must first be arranged into a more convenient form. First, equations (2.187) and (2.188) are substituted into equation (2.193) to obtain

$$P_k = E\left[\{My - x_k\}\{My - x_k\}^{\mathrm{T}}\right] \tag{2.194}$$

Then, substituting equation (2.181) for y yields

$$P_k = E\left[\{M(Hx_k + \varepsilon) - x_k\}\{M(Hx_k + \varepsilon) - x_k\}^{\mathrm{T}}\right] \tag{2.195}$$

which, when rearranged, is

$$P_k = E\left[\{(MH - I)x_k + M\varepsilon\}\{(MH - I)x_k + M\varepsilon\}^{\mathrm{T}}\right] \tag{2.196}$$

Thus, since $MH = I$ from equation (2.192),

$$E\left[(MH - I)x_k\right] = (MH - I)E\left[x_k\right] = 0 \tag{2.197}$$

Then from equation (2.190b),

$$P_k = E\left[M\varepsilon\varepsilon^{\mathrm{T}}M^{\mathrm{T}}\right] = ME\left[\varepsilon\varepsilon^{\mathrm{T}}\right]M^{\mathrm{T}} = MRM^{\mathrm{T}} \tag{2.198}$$

The steps given below are followed to minimize P_{k}, subject to the constraint imposed by the requirement that $\hat{x}_k$ be unbiased.

The quantity

$$\lambda^{\mathrm{T}}(I - MH)^{\mathrm{T}} + (I - MH)\lambda \tag{2.199}$$

where λ is an nxn matrix of Lagrangian multipliers, is added to the expression for P_{k}, resulting in:

$$P_k = MRM^{\mathrm{T}} + \lambda^{\mathrm{T}}(I - MH)^{\mathrm{T}} + (I - MH)\lambda \tag{2.200}$$

The technique of Lagrangian multipliers was developed for the purpose of finding the extrema of functions which are subject to constraints. In the case above, equation (2.199), which is added to the function to be minimized, equation (2.198), is made a symmetric function by treating equation (2.199) as a matrix plus its transpose. The covariance matrix is a symmetric one, and this preserves the symmetric property.

The first variation of P_k, δP_k, is given by

$$\delta P_k = \delta M(RM^{\mathrm{T}} - H\lambda) + (MR - \lambda^{\mathrm{T}} H^{\mathrm{T}})\delta M^{\mathrm{T}} + \delta\lambda^{\mathrm{T}}(I - MH)^{\mathrm{T}} + (I - MH)\delta\lambda \tag{2.201}$$

For minimum variance, δP_k equals zero. This requires that the following conditions hold:

$$RM^{\mathrm{T}} - H\lambda = 0 \tag{2.202}$$

$$I - MH = 0 \tag{2.203}$$

Solving for M and λ,

$$M = \lambda^{\mathrm{T}} H^{\mathrm{T}} R^{-1} \tag{2.204a}$$

$$\lambda^{\mathrm{T}} H^{\mathrm{T}} R^{-1} H = I \tag{2.204b}$$

$$\lambda^{\mathrm{T}} = (H^{\mathrm{T}} R^{-1} H)^{-1} \tag{2.204c}$$

Substitute λ^{T} in (2.204a) to get M in terms of H and R only.

$$M = (H^{\mathrm{T}} R^{-1} H)^{-1} H^{\mathrm{T}} R^{-1} \tag{2.205}$$

Substitute M in equation (2.198)

$$\begin{aligned} P_k &= MRM^{\mathrm{T}} \\ P_k &= (H^{\mathrm{T}} R^{-1} H)^{-1} H^{\mathrm{T}} R^{-1} R[(H^{\mathrm{T}} R^{-1} H)^{-1} H^{\mathrm{T}} R^{-1}]^{\mathrm{T}} \\ P_k &= (H^{\mathrm{T}} R^{-1} H)^{-1} H^{\mathrm{T}} (H^{\mathrm{T}} R^{-1})^{\mathrm{T}} (H^{\mathrm{T}} R^{-1} H)^{-\mathrm{T}} \\ P_k &= (H^{\mathrm{T}} R^{-1} H)^{-1} \end{aligned} \tag{2.206}$$

Substitute M into equation (2.187) to yield:

$$\hat{x}_k = My = (H^{\mathrm{T}} R^{-1} H)^{-1} H^{\mathrm{T}} R^{-1} y \tag{2.207}$$

Equations (2.206) and (2.207) are the covariance and the correction vector equations respectively, purely in terms of H, R and y. H contains the partial derivatives of the observations with respect to the components of the state vector (Equation 2.171).

R is the diagonal matrix ($E[\varepsilon\varepsilon^{\mathrm{T}}]$) containing the expected value of the observation errors (Equation 2.190b) and y is the vector of observation residuals (Equation 2.180).

A batch least squares estimate of $\hat{x}_k$, given l observations, would proceed as follows. Given the initial state, $X^*(t_0)$, and a vector of observations, Y, from time t_0 to time t_f, the estimate of the state, $\hat{X}_k$, can be made by following the steps. These steps can be iterated using some convergence criteria by returning to step 1 after step 4 is completed.

1. Integrate the state differential equation (Equation 2.148) and the state transition matrix differential equation (Equation 2.161) to the time of each observation.

2. At each observation at time t_i, compute the observation partial derivative from

$$\widetilde{H}(t_i) = \left[\frac{\partial G(X, t_i)}{\partial X}\right]_{X=X^*(t_i)} \tag{2.208}$$

and propagate the partial derivative to time t_k using

$$H(t_k) = \widetilde{H}_i \Phi(t_i, t_k) \tag{2.209}$$

Computation of the state transition matrix, $\Phi(t_i, t_k)$, may require several steps, depending on the value of t_k relative to the interval (t_0,t_f) and depending on the number of parameters in the state, n. In general

$$\Phi(t_i, t_k) = \Phi(t_i, t_0)\Phi(t_0, t_k) \tag{2.210}$$

where $\Phi(t_i, t_0)$ and $\Phi(t_0, t_k)$ can be computed by integrating Equation (2.161) from t_0 to t_i and t_k to obtain $\Phi(t_i, t_0)$ and $\Phi(t_0, t_k)$.

Then,

$$\Phi(t_0, t_k) = \Phi^{-1}(t_k, t_0)$$

This could be precomputed and stored. If t_k has been chosen to be t_0, then

$$\Phi(t_0, t_k) = I$$

and the observational partial derivative matrix is

$$H(t_0) = \tilde{H}(t_i)\Phi(t_i, t_0) \tag{2.211}$$

3. Compute the observation $G[X^*(t_i), t_i]$ and the O-C residual, $y(t_i)$, from

$$y(t_i) = Y(t_i) - G(t_i)$$

4. When all observations have been processed, compute the state update and covariance matrix as

$$\hat{x}_k = (H^T R^{-1} H)^{-1} H^T R^{-1} y \tag{2.212a}$$

$$P = (H^T R^{-1} H)^{-1} \tag{2.212b}$$

and compute the state estimate at t_k from

$$\hat{X}(t_k) = X^*(t_k) + \hat{x}(t_k) \tag{2.213}$$

2.3.1.2 Derivation of the Kalman Filter

In this derivation, an estimate, $\hat{x}_{k-1}$, and a covariance matrix, $P_{k\text{-}1}$, based on measurements from t_0 to $t_{k\text{-}1}$, are considered. Further observational information, either a single observation or a vector of observations, at time t_k is to be added to this set, and the values of the estimate, $\hat{x}_k$, and the covariance matrix P_k, at time t_k are to be found. First, it is necessary to predict the estimate, $\hat{x}_{k-1}$, forward to time t_k. The prediction of perturbations in the state has been previously developed (Equation 2.178) and may be written as

$$\bar{x}_k = \Phi(t_k, t_{k-1})\hat{x}_{k-1} \tag{2.214}$$

This is an estimate of the error in the state, X^*, at time t_k, based on observations from t_0 to $t_{k\text{-}1}$.

The predicted covariance is defined as

$$\bar{P}_k = E\left[(x_k - \bar{x}_k)(x_k - \bar{x}_k)^{\mathrm{T}}\right] \tag{2.215}$$

This is the predicted covariance at time t_k, based on observations from t_0 to $t_{k\text{-}1}$. Using Equation 2.214, Equation 2.215 can be rewritten as

$$\begin{aligned} \bar{P}_k &= E[\Phi(t_k, t_{k-1})(x_{k-1} - \hat{x}_{k-1})(x_{k-1} - \hat{x}_{k-1})^{\mathrm{T}}\Phi(t_k, t_{k-1})] \\ \bar{P}_k &= \Phi(t_k, t_{k-1})P_{k-1}\Phi^{\mathrm{T}}(t_k, t_{k-1}) \end{aligned} \tag{2.216}$$

The linearized observation equation at t_k is

$$y_k = H_k \Phi(t_k, t_k) x_k + \varepsilon_k \tag{2.217a}$$

$$y_k = H_k x_k + \varepsilon_k \tag{2.217b}$$

where

$$E[\varepsilon_k] = 0 \tag{2.218a}$$

and

$$E[\varepsilon_k \varepsilon_k^{\mathrm{T}}] = R_k \tag{2.218b}$$

If the predicted correction is considered to be a variable which contains a random error, η_k, then x_k is the true value of the correction at time t_k, i.e.,

$$\bar{x}_k = x_k + \eta_k \tag{2.219}$$

where

$$E[\eta_j] = 0 \tag{2.220a}$$

and

$$E[\eta_k \eta_k^{\mathrm{T}}] = \bar{P}_k \tag{2.220b}$$

This quantity contains the information from observations from t_0 to t_{k-1}.

Then, the same formalism can be used for both the observation residual at time t_k and the correction vector. First, the following definitions are made:

$$y = \begin{bmatrix} \bar{x}_k \\ y_k \end{bmatrix} \tag{2.221}$$

$$H = \begin{bmatrix} I \\ H_k \end{bmatrix} \tag{2.222}$$

and

$$R = \begin{bmatrix} \overline{P}_k & 0 \\ 0 & R_k \end{bmatrix} \tag{2.223}$$

The information from all observations previous to t_k is contained in $\bar{x}_k$ and $\overline{P}_k$. The problem can now be treated as though there are two "observations," $\bar{x}_k$ and y_k.

From Equation 2.207, the state correction estimated from all observations to t_k is

$$\hat{x}_k = (H^T R^{-1} H)^{-1} H^T R^{-1} y \tag{2.224}$$

Written explicitly, with the aid of Equations (2.213) through (2.216), the state correction becomes:

$$\begin{aligned} \hat{x}_k &= \left[\begin{bmatrix} I & H_k^T \end{bmatrix} \begin{bmatrix} \overline{P}_k^{-1} & 0 \\ 0 & R_k^{-1} \end{bmatrix} \begin{bmatrix} I \\ H_k \end{bmatrix} \right]^{-1} \begin{bmatrix} I & H_k^T \end{bmatrix} \begin{bmatrix} \overline{P}_k^{-1} & 0 \\ 0 & R_k^{-1} \end{bmatrix} \begin{bmatrix} \bar{x}_k \\ y_k \end{bmatrix} \\ &= (H_k^T R_k^{-1} H_k + \overline{P}_k^{-1})^{-1} (H_k^T R_k^{-1} y_k + \overline{P}_k^{-1} \bar{x}_k) \end{aligned} \tag{2.225}$$

and the updated covariance matrix can be similarly expressed as

$$\begin{aligned} P_k &= (H^T R^{-1} H)^{-1} \\ &= \left[\begin{bmatrix} I & H_k^T \end{bmatrix} \begin{bmatrix} \overline{P}_k^{-1} & 0 \\ 0 & R_k^{-1} \end{bmatrix} \begin{bmatrix} I \\ H_k \end{bmatrix} \right]^{-1} \\ &= (H_k^T R_k^{-1} H_k + \overline{P}_k^{-1})^{-1} \end{aligned} \tag{2.226}$$

This equation can be rewritten to eliminate the inversion of an nxn matrix by using the Schur identity (also known as the inside-out rule). First, the following inversion is made:

$$P_k^{-1} = H_k^T R_k^{-1} H_k + \overline{P}_k^{-1} \tag{2.227}$$

This expression is then premultiplied by P_k and postmultiplied by $\overline{P}_k$ to obtain

$$\overline{P}_k = P_k H_k^{\mathrm{T}} R_k^{-1} H_k \overline{P}_k + P_k \tag{2.228}$$

or equivalently,

$$P_k = \overline{P}_k - P_k H_k^{\mathrm{T}} R_k^{-1} H_k \overline{P}_k \tag{2.229}$$

Postmultiplying Equation (2.228) by $H_k^{\mathrm{T}} R_k^{-1}$ and reordering the result yields

$$P_k H_k^{\mathrm{T}} R_k^{-1} [I + H_k \overline{P}_k H_k^{\mathrm{T}} R_k^{-1}] = \overline{P}_k H_k^{\mathrm{T}} R_k^{-1} \tag{2.230}$$

This can be written as

$$P_k H_k^{\mathrm{T}} R_k^{-1} [H_k \overline{P}_k H_k^{\mathrm{T}} + R_k] R_k^{-1} = \overline{P}_k H_k^{\mathrm{T}} R_k^{-1} \tag{2.231}$$

Postmultiplying this by R_k and solving for $P_k H_k^{\mathrm{T}} R_k^{-1}$ yields

$$P_k H_k^{\mathrm{T}} R_k^{-1} = \overline{P}_k H_k^{\mathrm{T}} \left[H_k \overline{P}_k H_k^{\mathrm{T}} + R_k \right]^{-1} \tag{2.232}$$

Replacing this expression for $P_k H_k^{\mathrm{T}} R_k^{-1}$ in Equation (2.229) with (2.232) results in the following

$$P_k = \overline{P}_k - \overline{P}_k H_k^{\mathrm{T}} \left[H_k \overline{P}_k H_k^{\mathrm{T}} + R_k \right]^{-1} H_k \overline{P}_k \tag{2.233}$$

Equation (2.226) for the covariance has now been rewritten. For a single observation at t_k, the quantity $\left[H_k \overline{P}_k H_k^{\mathrm{T}} + R_k \right]$ is a scalar, and P_k can be evaluated without inverting a matrix.

The Kalman gain, K_k, is defined to be

$$K_k \equiv \overline{P}_k H_k^{\mathrm{T}} \left[H_k \overline{P}_k H_k^{\mathrm{T}} + R_k \right]^{-1} \tag{2.234}$$

Therefore,

$$P_k = [I - K_k H_k]\overline{P}_k \tag{2.235}$$

The original inversion of an *nxn* matrix is now reduced to the inversion of a scalar quantity for a single observation.

Substituting Equation (2.235) into the equation for the estimated state correction, Equation (2.225) yields:

$$\begin{aligned}\hat{x}_k &= [I - K_k H_k]\overline{P}_k \left[H_k^{\mathrm{T}} R_k^{-1} y_k + \overline{P}_k^{-1}\overline{x}_k\right] \\ &= [I - K_k H_k]\overline{x}_k + [I - K_k H_k]\overline{P}_k H_k^{\mathrm{T}} R_k^{-1} y_k\end{aligned} \tag{2.236}$$

The coefficient of y_k (the second part of the above equation) can be reduced to a less complex form by expanding K_k according to Equation (2.234) to obtain

$$[I - K_k H_k]\overline{P}_k H_k{}^{\mathrm{T}} R_k = \overline{P}_k H_k{}^{\mathrm{T}} R_k - \overline{P}_k H_k{}^{\mathrm{T}} [H_k \overline{P}_k H_k{}^{\mathrm{T}} + R_k]^{-1} H_k \overline{P}_k H_k{}^{\mathrm{T}} R_k{}^{-1} \tag{2.237}$$

This term can also be expressed as:

$$\overline{P}_k H_k{}^{\mathrm{T}} [I - (H_k \overline{P}_k H_k{}^{\mathrm{T}} + R_k)^{-1} H_k \overline{P}_k H_k{}^{\mathrm{T}}] R_k{}^{-1} \tag{2.238}$$

The identity matrix can be written as:

$$I = (H_k \overline{P}_k H_k{}^{\mathrm{T}} + R_k)^{-1} (H_k \overline{P}_k H_k{}^{\mathrm{T}} + R_k) \tag{2.239}$$

Substituting this identity into Equation (2.238) yields the equality:

$$\begin{aligned}&\overline{P}_k H_k{}^{\mathrm{T}} [H_k \overline{P}_k H_k{}^{\mathrm{T}} + R_k]^{-1} [H_k \overline{P}_k H_k{}^{\mathrm{T}} + R_k - H_k \overline{P}_k H_k{}^{\mathrm{T}}] R_k{}^{-1} \\ &= \overline{P}_k H_k{}^{\mathrm{T}} [H_k \overline{P}_k H_k{}^{\mathrm{T}} + R_k]^{-1} = K_k\end{aligned} \tag{2.240}$$

The estimate from Equation (2.236) can thus be written in the form:

$$\hat{x}_k = [I - K_k H_k]\overline{x}_k + K_k y_k = \overline{x}_k + K_k (y_k - H_k \overline{x}_k) \tag{2.241}$$

The covariance for the estimate equation (2.241) is given by equation (2.235).

2.3.1.3 Algorithm for the Sequential Kalman Filter

For a Kalman sequential filter, the steps in the computation of t_k, given information at t_{k-1}, are the following:

Given $\hat{x}_{k-1}$, P_{k-1}, $X^*(t_{k-1})$, and an observation $Y(t_k)$:

1. Propagate the state and the state transition matrix from t_{k-1} to t_k to obtain $X^*(t_k)$ and $\Phi(t_k, t_{k-1})$

$$\dot{X}^*(t) = F(X^*, t) \qquad (X^*(t_{k-1}); \text{ initial conditions}) \tag{2.242a}$$

$$\dot{\Phi}(t, t_{k-1}) = A(t)\Phi(t, t_{k-1}) \quad (\Phi(t_{k-1}, t_{k-1}) = I \text{ ; initial conditions}) \tag{2.242b}$$

2. Predict the covariance matrix and the state correction

$$\overline{P}_k = \Phi(t_k, t_{k-1}) P_{k-1} \Phi^{\mathrm{T}}(t_k, t_{k-1}) \tag{2.243a}$$

$$\overline{x}_k = \Phi(t_k, t_{k-1}) \hat{x}_{k-1} \tag{2.243b}$$

3. Compute the observation, O-C residuals, and observation partial derivatives

$$G(X^*(t_k), t_k) \tag{2.244a}$$

$$y_k = Y(t_k) - G(X^*(t_k), t_k) \tag{2.244b}$$

$$H_k = \left[\frac{\partial G}{\partial X} \right]_{X = X^*} \tag{2.244c}$$

4. Compute the gain and update the state correction and covariance matrix

$$K_k = \overline{P}_k H_k^{\mathrm{T}} [H_k \overline{P}_k H_k^{\mathrm{T}} + R_k]^{-1} \tag{2.245a}$$

$$P_k = [I - K_k H_k]\overline{P}_k \tag{2.245b}$$

$$\hat{x}_k = \overline{x}_k + K_k(y_k - H_k\overline{x}_k) \tag{2.245c}$$

5. Select the next observation at t_{k+1} and go back to step 1.

When all observations have been processed, the computed state correction at the last time, $\hat{x}_l$, is then added to the state, $X^*(t_l)$, to obtain the estimated state at t_l as follows:

$$\hat{X}(t_l) = X^*(t_l) + \hat{x}_l \tag{2.246}$$

2.3.1.4 The Algorithm for the Extended Kalman Filter

A variation of the Kalman filter is to add the correction vector, x_k, to the solve-for vector at each observation, instead of waiting until the last observation. In this case, X^* is computed instead of x, since $\hat{X}$ and $\overline{X}$ are treated directly. Because the correction is added to the solve-for vector at each observation, the predicted correction, $\overline{x}(t)$, at the observation time, t_k, is equal to zero.

There are several reasons for using the extended Kalman filter (EKF) instead of a standard Kalman filter. The EKF will yield a new state at each observation, which is of value when a real-time solution is desired as the filter processes data. By adding the corrections into the state at each observation, the effects of the nonlinearities in the equations of motion are not as severe, since the trajectory is being corrected at each

observation. Also, the partials of the system dynamic function are recomputed at each time step given the updated state. This allows for a more accurate state transition matrix.

The extended Kalman filter algorithm is as follows:

Given $X^*_{k-1} = \hat{X}_{k-1}, P_{k-1}$, and an observation $Y(t_k)$:

1. Propagate the state and state transition matrix from t_{k-1} to t_k to obtain $X^*_k = \overline{X}(t_k)$ and $\Phi(t_k, t_{k-1})$ according to the following equations:

$$\dot{X}(t) = F(X,t) \qquad (\hat{X}(t_{k-1})\text{; initial conditions}) \tag{2.247a}$$

$$\dot{\Phi}(t,t_{k-1}) = A(t)\Phi(t,t_{k-1}) \quad (\Phi(t_{k-1},t_{k-1}) = I\text{; initial conditions}) \tag{2.247b}$$

where

$$A(t) = \left[\frac{\partial F}{\partial X}\right]_{X=\overline{X}}$$

2. Predict the covariance matrix. The predicted correction, $\bar{x}(t_k)$ is not computed because it is zero in the EKF.

$$\overline{P}_k = \Phi(t_k,t_{k-1})P_{k-1}\Phi^{\mathrm{T}}(t_k,t_{k-1}) \tag{2.248}$$

3. Compute the observation $G(\overline{X}(t_k),t_k)$, the O-C residuals, y_k, and the observation partial derivatives, H_k, from

$$y_k = Y(t_k) - G(\overline{X}(t_k),t_k) \tag{2.249a}$$

$$H_k = \left[\frac{\partial G}{\partial X}\right]_{X_k=\overline{X}(t_k)} \tag{2.249b}$$

4. Compute the gain, K_k, update the covariance matrix, P_k, and the solve-for vector, $\hat{X}_k$, as follows:

$$K_k = \overline{P}_k H_k^{\mathrm{T}} [H_k \overline{P}_k H_k^{\mathrm{T}} + R_k]^{-1} \quad (2.250a)$$

$$P_k = [I - K_k H_k] \overline{P}_k \quad (2.250b)$$

$$\hat{X}_k = \overline{X}_k + K_k y_k \quad (2.250c)$$

5. Select the next observation at t_{k+1} and go back to step 1.

If Equations (2.250a-c) are compared with the update equations in (2.245), it can be seen that step 4 of the EKF algorithm includes the computation of $\hat{x}_k$ and the addition of $\hat{x}_k$ to $X^*(t_k)$ (which is, in this case, equal to $\overline{X}(t_k)$), to obtain $\hat{X}(t_k)$. That is, the Kalman filter steps

$$\hat{x}_k = \overline{x}_k + K_k (y_k - H_k \overline{x}_k)$$

and

$$\hat{X}(t_l) = X^*(t_l) + \hat{x}_l$$

from Equations (2.245) and (2.246) are identical to

$$\hat{X}_k = \overline{X}_k + K_k y_k$$

from Equation (2.250) (remembering that $\overline{x}_k = 0$ for the EKF process).

2.3.1.5 Glossary of Mathematical Symbols

A(t)	*n* x *n* matrix of partial derivatives of the equations of motion, *F*
F(X,t)	Vector of state differential equations
G(X(t), t)	Observation equation
H(t)	1 x *n* matrix of partial derivatives of *G(X(t), t)* with respect to *X(t)*
I	Identity matrix

K	Kalman gain
l	Number of observations
M	Combination of the observation partial derivatives and the covariance
$\overline{P}$	Predicted Covariance matrix
P	Covariance matrix
R	Matrix of observation variances
t	Independent variable time
$X(t)$	Solve-for parameter vector, i.e. the state
$\overline{X}(t)$	Predicted state vector
$X*$	Predicted state vector, i.e. $X*_k = \overline{X}(t_k)$
$\hat{X}(t)$	Estimated (a posteriori) state vector
$\hat{x}$	Estimated correction vector for the state
$\overline{x}$	Predicted correction vector for the state
Y	Vector of observations
y(t)	observed minus computed observation (residual)
δ	Variational operator
ε	Vector of errors in the computed observations
η	Vector of errors in the predicted solve-for parameter correction vector, *x(t)*
λ	*n* x *n* matrix of Lagrange multipliers
σ	Standard deviation
$\Phi(t_i, t_j)$	State transition matrix from t_j to t_i

2.3.2 Filters/Smoothers

When dealing with systems with highly nonlinear system dynamics and when observations can be processed in an offline sense, i.e. real-time state estimates are not needed, *smoothing* is a way to compute more accurate state estimates than the Kalman Filter can alone. There are several types of smoothing to be found in the literature (74), (75), (76), (77).

References (74) and (77) classify smoothing problems into three categories. These categories are *Fixed-interval* smoothing, *fixed-point* smoothing, and *fixed-lag* smoothing. Fixed-interval smoothing keeps the time interval of measurements fixed and optimal state estimates are sought for interior times within the interval. Information from both past and future measurements is applied to compute optimal state estimates for these interior points (77). Fixed-point smoothing is used to seek state estimates for a single point in time. The measurements occurring after this single point in time are subsequently used to improve the estimate at that point. An example of this would be the estimation of initial conditions based on later observations of a trajectory (77). Fixed-lag smoothing is used to seek estimates of a state which is a fixed number of time points behind the current measurement time point (77). Because the Backward Smoothing

Extended Kalman Filter described in section 2.3.4 incorporates *fixed-interval* smoothing in its algorithm, this type of smoothing will be the focus of this section.

Fixed-interval smoothing was introduced in the papers in references (75) and (76). Reference (77) refers to the algorithm as the Rauch-Tung-Striebel (RTS) algorithm and so that will be the usage in this section also. Consider a fixed-length interval containing $N + 1$ measurements. These will be indexed from $\mathbf{z}_0$ to $\mathbf{z}_N$. We assume the estimated random process can be modeled in the form:

$$\mathbf{x}_{k+1} = \Phi_k \mathbf{x}_k + \mathbf{w}_k \tag{2.251}$$

This is also known as the dynamic equation and is a discrete, linearized form of the continuous dynamic differential equation (2.148). This is analogous to equation (2.155) in the EKF derivation. Here, the dynamic equation is purely linear because Φ_k is simply a matrix with dimensions compatible with $\mathbf{x}_k$. This problem could take a form similar to the nonlinear form as the Extended Kalman Filter described in section 2.3.1 where the state transition matrix, Φ_k, is a linearized approximation to the nonlinear dynamics. The dynamic equation describes how a state at a later time is related to one at the current time. The $\mathbf{w}_k$ process noise vector describes how noise is introduced into the dynamics. It is assumed to be a white sequence with a known covariance.

The measurement equation for the process is given by:

$$\mathbf{z}_k = H_k \mathbf{x}_k + \mathbf{v}_k \tag{2.252}$$

Here, the relationship between the state vector at the current time and any measurements taken at the current time is described. The H_k matrix means the relationship is linear. Like the state transition matrix in the dynamic equation, the H_k matrix either represents a linear measurement equation or could be the result of linearization of a nonlinear measurement equation. This linearization procedure is described in the Extended Kalman Filter section 2.3.1. The $\mathbf{v}_k$ vector is assumed to be a white sequence with known covariance and having zero correlation with the $\mathbf{w}_k$ sequence.

Fixed-interval smoothing, i.e. the RTS algorithm, consists of a forward recursive filter sweep followed by a backward sweep. The forward filter sweep is identical to the Extended Kalman Filter (EKF) algorithm described in section 2.3.1. The backward sweep requires that the a priori and a posteriori estimates, $\hat{\mathbf{x}}_k$, and associated covariance matrices, $\mathbf{P}_k$, be saved. The backward sweep starts with initial conditions which are the last state estimate and covariance computed using the forward filter sweep, $\hat{\mathbf{x}}(N \mid N)$ and $\mathbf{P}(N \mid N)$ (77). With each step of the backward sweep, the old estimate from the forward filter sweep is updated to yield an improved smoothed estimate. This improved estimate is based on all the measurement data. The recursive equations for the backward sweep are (77):

$$\hat{\mathbf{x}}(k \mid N) = \hat{\mathbf{x}}(k \mid k) + \mathbf{A}(k)[\hat{\mathbf{x}}(k+1 \mid N) - \hat{\mathbf{x}}(k+1 \mid k)] \tag{2.253}$$

The notation $\hat{\mathbf{x}}(k \mid N)$ means the estimate of **x** at time *k*, given measurements, $\mathbf{z}_0$ to $\mathbf{z}_N$. The smoothing gain, $\mathbf{A}(k)$, is given by (77):

$$\mathbf{A}(k) = \mathbf{P}(k \mid k)\Phi^T(k+1,k)\mathbf{P}^{-1}(k+1 \mid k) \tag{2.254}$$

The error covariance for the smoothed estimates is given by the recursive equation (77):

$$\mathbf{P}(k \mid N) = \mathbf{P}(k \mid k) + \mathbf{A}(k)[\mathbf{P}(k+1 \mid N) - \mathbf{P}(k+1 \mid k)]\mathbf{A}^T(k) \tag{2.255}$$

It should be noted that the smoothed error covariance matrix is not required in order to compute the state estimates in the backward sweep. This is, of course, different than for the forward filter sweep in which the filtered error covariance is needed to compute the gain used in computing updated state estimates (77).

Smoothing is typically used when one desires state estimates with more accuracy than what is achievable with a forward Kalman filter pass alone. This is true with linear and nonlinear systems [(67), pp. 200]. The improved accuracy is obtained because the smoother incorporates information from future and past measurements to estimate each state. The forward filter only uses past measurements to estimate each state. Figures 2.7 and 2.8 illustrate the improvement in state estimate accuracy and covariance obtained with a smoother over a forward filter sweep Kalman filter. These figures were generated by writing software to implement the Kalman Filter and Smoother as described in reference (77). This case uses linear system dynamic and measurement equations, but

improvements are also expected for nonlinear systems. Figure 2.7 shows that the smoothed estimates from the fixed-interval (RTS) and fixed-lag (LAG) smoothers remain closer to the actual state (Truth) when observations cause the Kalman Filter (KF) to diverge from truth periodically.

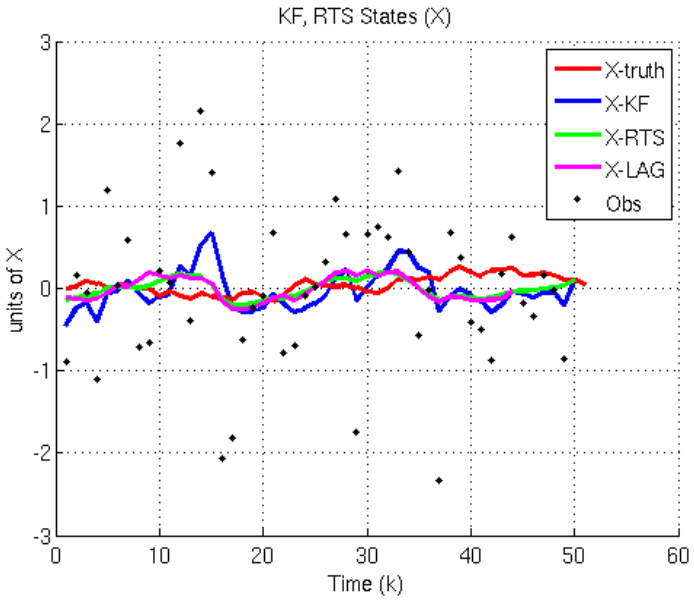

Figure 2.7 Kalman Filter vs. Rauch-Tung-Striebel Smoothed Estimates

Figure 2.7 illustrates that the covariance for the RTS smoother is slightly better than that of the fixed-lag (LAG) smoother and is significantly better than that of the Kalman Filter (KF).

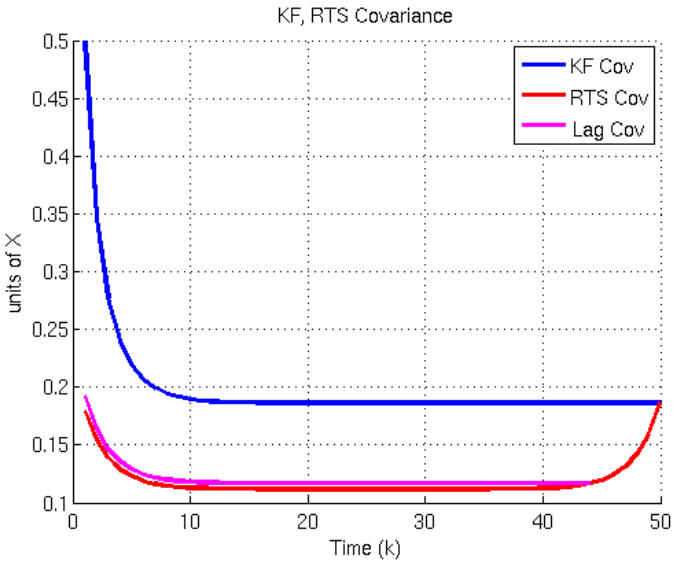

Figure 2.8 Kalman Filter vs. Rauch-Tung-Striebel Smoothed Covariance

It is worth noting that only those states which are controllable by the noise driving the system state vector are smoothable. Constant states are not smoothable, while randomly time-varying states are smoothable (78).

Nonlinear smoothing differs from linear smoothing in that the nonlinear smoothing problem is more difficult [(67), pp. 180]. The linear Gaussian case of the optimal estimate of the state for most reasonable Bayesian optimization criteria is the conditional mean of the state given the observations [(67), pp. 180]. The Gaussian property implies that the condition mean can be computed from a unique linear operation on the measurement data, i.e. the Kalman filter algorithm [(67), pp. 181]. In contrast, the nonlinear filtering/smoothing problem is not generally Gaussian. Therefore, many Bayesian criteria lead to estimates that are different from the condition mean of the state given the observations [(67), pp. 181]. Optimal estimation algorithms for nonlinear systems often cannot be expressed in closed form which requires methods for

approximating optimal nonlinear filters [(67), pp. 181]. The Extended Kalman Filter (EKF) described in section 2.3.1 performs this approximation by linearizing the dynamic and measurement equations and then forming the minimum variance estimate from the linearized equations. Reference [(67), pp. 193-194] uses an example to show that the nonlinearity of the dynamic and measurement functions can have an important effect on the estimation accuracy. The degree of importance depends on the degree of nonlinearity, the shape of the joint density function of the state and observations, and the strength of the measurement noise.

The fixed-interval smoother just described has a different formulation from the square-root information smoother (SRIS) used in section 5 with the Backward Smoothing Extended Kalman Filter (2). Mark Psiaki used the SRIS from reference (79) in order to incorporate estimation of process noise vectors and to take advantage of the improved numerical stability of the SRIS form over the original Kalman filter/smoother formulation. The SRIS form for section 5 is taken from (79).

2.3.3 Unscented Kalman Filter

Simon Julier and Jeffrey Uhlmann introduced the Unscented Kalman Filter (UKF) in the 1990s and it has proved to be a useful extension to the Kalman Filter for nonlinear systems. The UKF yields performance equivalent to the KF for linear systems and generalizes to nonlinear systems without the linearization steps required by the EKF (65), (66). Analytically, and in practice, the UKF has been shown to be more accurate and more robust than the EKF (65), (66). In order to obtain the optimal solution to a

nonlinear filtering problem, the condition probability density function (PDF) of the state given the observations must be maintained accurately in the filter (65), (66). The EKF only maintains the mean and covariance of the conditional density which is passed through linear approximations of the dynamic and measurement functions. The UKF addresses these deficiencies by applying the *unscented transformation*. This transformation uses a set of "appropriately chosen weighted points to parameterize the mean and covariance of a give probability distribution" (65), (66). Another advantage of the UKF is that the Jacobian matrices, i.e. the partials of the observation equations with respect to the state and the partials of the dynamic equations with respect to the state, required by the EKF are not required in the UKF. Rather, the dynamic and measurement functions can be treated as "black boxes" (65), (66).

The UKF is applied to a nonlinear discrete time system of the form (66):

$$\mathbf{x}(k+1) = \mathbf{f}[\mathbf{x}(k), \mathbf{u}(k), \mathbf{w}(k), k] \tag{2.256}$$

$$\mathbf{z}(k) = \mathbf{h}[\mathbf{x}(k), \mathbf{u}(k), k] + \mathbf{v}(k) \tag{2.257}$$

Here, $\mathbf{x}(k)$ is the n-dimensional state of the system at time k, $\mathbf{u}(k)$ is the input or control vector, $\mathbf{w}(k)$ is the q-dimensional state process noise vector due to disturbances and modeling errors, $\mathbf{z}(k)$ is the observation vector, and $\mathbf{v}(k)$ is the measurement noise. It is assumed that the noise vectors, $\mathbf{w}(k)$ and $\mathbf{v}(k)$, are zero-mean and (66):

$$E[\mathbf{v}(i)\mathbf{v}^T(j)] = \delta_{ij}\mathbf{Q}(i),\ E[\mathbf{w}(i)\mathbf{w}^T(j)] = \delta_{ij}\mathbf{R}(i),\ E[\mathbf{v}(i)\mathbf{w}^T(j)] = 0, \quad \forall i, j \tag{2.258}$$

For the EKF, the derivation is shown previously, in section 2.3.1. In the UKF, the *unscented transform* is used to transform the statistics of random variables, i.e. the state variables, when undergoing a nonlinear transformation (66). "The *unscented transformation* is based on the intuition that it is easier to approximate a probability distribution than it is to approximate an arbitrary nonlinear function." (66) This approach is illustrated in Figure 2.9.

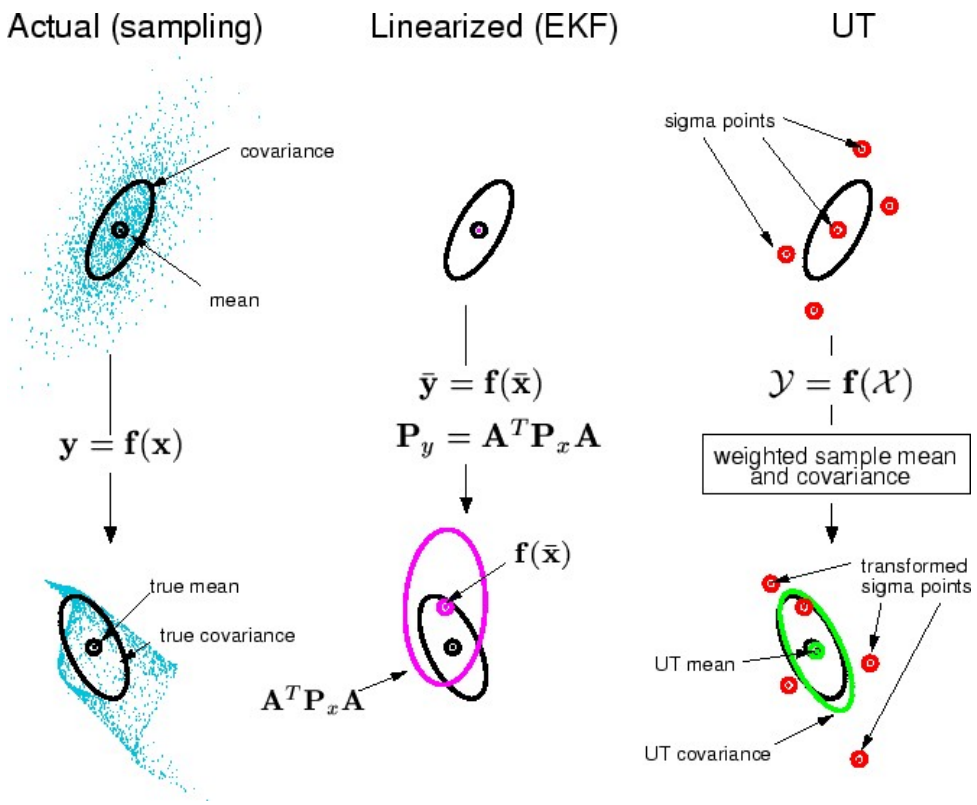

Figure 2.9 The Unscented Transform for Mean and Covariance Propagation (80)

A set of points, *sigma points*, are chosen so the sample mean and covariance are $\bar{\mathbf{x}}$ and $\mathbf{P}_{xx}$. The nonlinear function is applied to each point in turn to yield a new collection of points transformed by the nonlinear function. The vector and matrix, $\bar{\mathbf{y}}$ and $\mathbf{P}_{yy}$, are the

mean and covariance statistics of the transformed points (66). The method thus resembles a Monte-Carlo scheme. However, in the UKF, the samples are drawn deterministically rather than at random (66). Because "the problem of statistical convergence is not an issue here, high order information about the transformed distribution can be captured using only a very small number of points" (66).

The *n*-dimensional random variable **x** with mean, $\bar{\mathbf{x}}$, and covariance, $\mathbf{P}_{xx}$, is approximated by $2n+1$ weighted points given by (66):

$$\begin{aligned}
\chi_0 &= \bar{\mathbf{x}} \\
\chi_i &= \bar{\mathbf{x}} + (\sqrt{(n+\kappa)\mathbf{P}_{xx}})_i \\
\chi_{i+n} &= \bar{\mathbf{x}} - (\sqrt{(n+\kappa)\mathbf{P}_{xx}})_i \\
W_0 &= \kappa/(n+\kappa) \\
W_i &= (1/2)(n+\kappa) \\
W_{i+n} &= (1/2)(n+\kappa)
\end{aligned} \tag{2.259}$$

Here, $\kappa \in \Re$, $(\sqrt{(n+\kappa)\mathbf{P}_{xx}})_i$ is the *i*th row or column of the matrix square root of $(n+\kappa)\mathbf{P}_{xx}$, and W_i is the weight which is associated with the *i*th point. The transformation procedure is as follows (66):

1. Instantiate each point through the function to yield the set of transformed sigma points:

$$\gamma_i = \mathbf{f}[\chi_i] \tag{2.260}$$

2. The mean is given by the weighted average of the transformed points:

$$\bar{\mathbf{y}} = \sum_{i=0}^{2n} W_i \gamma_i \tag{2.261}$$

3. The covariance is the weighted outer product of the transformed points:

$$\mathbf{P}_{yy} = \sum_{i=0}^{2n} W_i \{\gamma_i - \bar{\mathbf{y}}\}\{\gamma_i - \bar{\mathbf{y}}\}^T \tag{2.262}$$

The mean and covariance of **x** are determined by the algorithm and are precise to second order. The mean and covariance of **y** are likewise precise to the second order (66). This is notable because the mean is will be more precisely known than the mean in the EKF, but the covariance will be known the same as in the EKF, i.e. to second order (66). "Since the distribution of **x** is approximated rather than **f[ ]**, its series expansion is not truncated at a particular order." "It can be shown that the unscented algorithm is able to partially incorporate information from higher orders of **f[ ]** which allows for more accurate treatment of the system dynamics" (66).

The sigma points capture will capture identical mean and covariances for the choice of matrix square-root, therefore numerically efficient and stable methods such as the Cholesky decomposition can be used (66). Because the mean and covariance are calculated using standard vector and matrix operations, the algorithm is suitable for any choice of process model. Implementation is potentially more rapid than with the EKF because Jacobian matrices are not needed (66).

The UKF algorithm makes use of the unscented transformation for the prediction steps in the Kalman Filter. These Kalman Filter steps include prediction of the new state of the system, $\hat{\mathbf{x}}(k+1|k)$, and the associated covariance, $\mathbf{P}(k+1|k)$, while accounting for system process noise. Also, the KF involves prediction of the expected observation, $\hat{\mathbf{z}}(k+1|k)$, and the residual covariance, $\mathbf{P}_{\nu\nu}(k+1|k)$, which should include the effects of observation noise. Lastly, the cross-correlation matrix, $\mathbf{P}_{xz}(k+1|k)$, is predicted (66). These steps can be accommodated by the unscented transform by restructuring the state vector and process and observation models. First, the state vector is augmented with the process noise terms to give an $n^a = n+q$ dimensional vector (66):

$$\mathbf{x}^a(k) = \begin{bmatrix} \mathbf{x}(k) \\ \mathbf{w}(k) \end{bmatrix} \tag{2.263}$$

The process model is rewritten as a function of $\mathbf{x}^a(k)$:

$$\mathbf{x}(k+1) = \mathbf{f}[\mathbf{x}^a(k), \mathbf{u}(k), k] \tag{2.264}$$

The unscented transform uses $2n^a + 1$ sigma points which are drawn from:

$$\hat{\mathbf{x}}^a(k|k) = \begin{pmatrix} \hat{\mathbf{x}}(k|k) \\ 0_{qx1} \end{pmatrix} \tag{2.265}$$

$$\mathbf{P}^a(k|k) = \begin{bmatrix} \mathbf{P}(k|k) & \mathbf{P}_{xw}(k|k) \\ \mathbf{P}_{xw}(k|k) & \mathbf{Q}(k) \end{bmatrix} \tag{2.266}$$

The matrices on the leading diagonal are the covariances and off-diagonal sub-blocks are the correlations between the state errors and the process noise. Although this method requires the use of addition sigma points due to the augmented state vector, it means that the effects of process noise are introduced with the same order of accuracy as the uncertainty in the state (66).

The following UKF algorithm is taken from (80). Initialize the UKF with:

$$\begin{aligned}\hat{\mathbf{x}}_0 &= E[\mathbf{x}_0] \\ \mathbf{P}_0 &= E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T] \\ \hat{\mathbf{x}}_0^a &= E[\mathbf{x}^a] = [\mathbf{x}_0^T \; 0 \; 0]^T\end{aligned} \tag{2.267}$$

$$\mathbf{P}_0^a = E[(\mathbf{x}_0^a - \hat{\mathbf{x}}_0^a)(\mathbf{x}_0^a - \hat{\mathbf{x}}_0^a)^T] = \begin{bmatrix} \mathbf{P}_0 & 0 \\ 0 & \mathbf{P}_w \end{bmatrix} \tag{2.268}$$

For $k \in \{1,...,\infty\}$, Calculate sigma points as in equation (2.259):

$$\chi^a(k-1) = \left[\hat{\mathbf{x}}^a(k-1) \qquad \hat{\mathbf{x}}^a(k-1) \pm \sqrt{(n+\kappa)[\mathbf{P}^a(k-1)]}\right] \tag{2.269}$$

Time update:

$$\chi^{\mathbf{x}}(k \mid k-1) = \mathbf{f}(\chi^{\mathbf{x}}(k-1), \chi^{\mathbf{w}}(k-1)) \tag{2.270}$$

$$\hat{\mathbf{x}}^-(k) = \sum_{i=0}^{2n^a} W_i \chi_i^{\mathbf{x}}(k \mid k-1) \tag{2.271}$$

$$\mathbf{P}^-(k) = \sum_{i=0}^{2n^a} W_i [\chi_i^{\mathbf{x}}(k \mid k-1) - \hat{\mathbf{x}}^-(k)][\chi_i^{\mathbf{x}}(k \mid k-1) - \hat{\mathbf{x}}^-(k)]^T \tag{2.272}$$

$$\mathbf{Y}(k \mid k-1) = \mathbf{h}(\chi^{\mathbf{x}}(k \mid k-1), \chi^{\mathbf{v}}(k-1)) \tag{2.273}$$

$$\hat{\mathbf{y}}^-(k) = \sum_{i=0}^{2n^a} W_i \, \mathbf{Y}_i(k \mid k-1) \tag{2.274}$$

Measurement update equations:

$$\mathbf{P}_{\mathbf{yy}} = \sum_{i=0}^{2n^a} W_i [\mathbf{Y}_i(k \mid k-1) - \hat{\mathbf{y}}_i^-(k)][\mathbf{Y}_i(k \mid k-1) - \hat{\mathbf{y}}_i^-(k)]^T \tag{2.275}$$

$$\mathbf{P}_{\mathbf{xy}} = \sum_{i=0}^{2n^a} W_i [\chi_i(k \mid k-1) - \hat{\mathbf{x}}^-(k)][\mathbf{Y}_i(k \mid k-1) - \hat{\mathbf{y}}_i^-(k)]^T \tag{2.276}$$

$$\mathbf{K} = \mathbf{P}_{xy}\mathbf{P}_{yy}^{-1} \tag{2.277}$$

$$\hat{\mathbf{x}}(k) = \hat{\mathbf{x}}^{-}(k) + \mathbf{K}(\mathbf{y}(k) - \hat{\mathbf{y}}^{-}(k)) \tag{2.278}$$

$$\mathbf{P}(k) = \mathbf{P}^{-}(k) - \mathbf{K}\mathbf{P}_{yy}\mathbf{K}^{T} \tag{2.279}$$

Here, the subscripted dash, "-", indicates a predicted quantity, $\mathbf{x}^{a} = [\mathbf{x}^{T}\ \mathbf{w}^{T}]^{T}$, and $\chi^{a} = [(\chi^{\mathbf{x}})^{T}\ (\chi^{\mathbf{w}})^{T}]^{T}$ (80).

2.3.4 Backward Smoothing Extended Kalman Filter

The Backward Smoothing Extended Kalman Filter (BSEKF) is a type of Iterated Extended Kalman Filter (IEKF) developed by Dr. Mark Psiaki at the Cornell University. Dr. Psiaki developed the filter to be used for state estimation problems in which the dynamic and measurement equations are highly nonlinear. The BSEKF was developed to provide more reliable convergence and robustness than other types of filters such as the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF). In his paper (2), Mark Psiaki applied the BSEKF to a "difficult spacecraft attitude estimation problem with sensing of fewer than three axes and dynamic model uncertainty" (2). He was able to demonstrate better convergence reliability and accuracy using the BSEKF than either the EKF or UKF.

Figure 2.10 illustrates the main features of the BSEKF estimation algorithm. The BSEKF algorithm attempts to improve the approximation of both the measurement and dynamic equations by introducing a Gauss-Newton iteration to minimize a cost function

that penalizes measurement error and state estimate error. The cost function includes terms for the *m* latest measurements and states. In addition, the *m* latest estimates are filtered and smoothed to treat the system dynamics over that span.

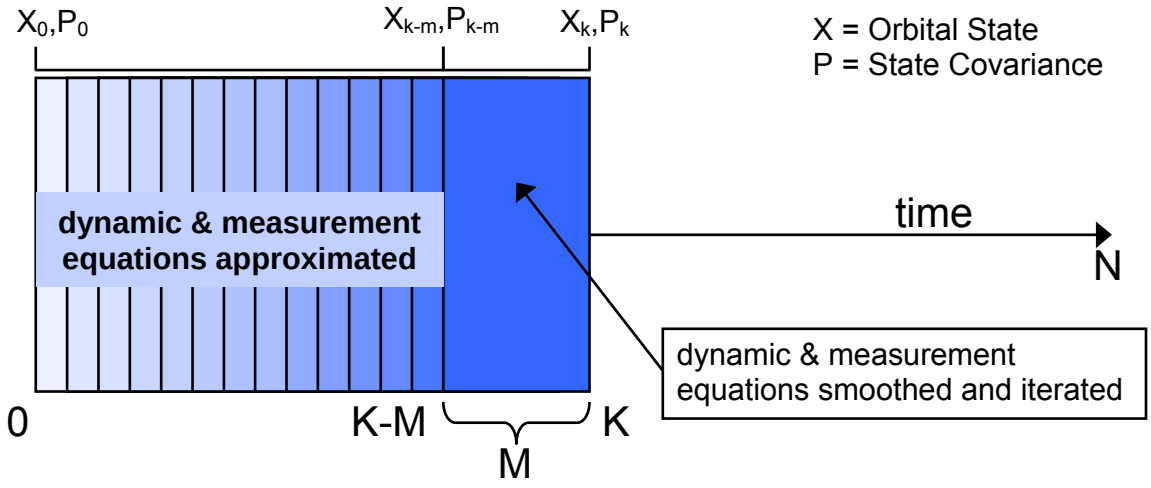

Figure 2.10 Illustration of BSEKF Estimation Algorithm

The BSEKF incorporates a Gauss-Newton iteration to solve for the state vectors, $\mathbf{x}_k$, $\mathbf{x}_i$, and process noise vectors, $\mathbf{w}_i$, for $i = k\text{-}m,\ldots,k\text{-}1$ which minimize the following cost function:

$$J = \frac{1}{2}\sum_{i=k-m}^{k-1}\left\{\mathbf{w}_i Q_i^{-1}\mathbf{w}_i + \left[\mathbf{y}_{i+1} - \mathbf{h}_{i+1}(\mathbf{x}_{i+1})\right]^T R_{i+1}^{-1}\left[\mathbf{y}_{i+1} - \mathbf{h}_{i+1}(\mathbf{x}_{i+1})\right]\right\}$$
$$+ \frac{1}{2}(\mathbf{x}_{k-m} - \hat{\mathbf{x}}_{k-m}^*)^T (P_{k-m}^*)^{-1}(\mathbf{x}_{k-m} - \hat{\mathbf{x}}_{k-m}^*) \tag{2.280}$$

The following constraint must also be adhered to for $i = k\text{-}m,\ldots,k\text{-}1$.

$$\mathbf{x}_{i+1} = \mathbf{f}_i(\mathbf{x}_i, \mathbf{w}_i) \tag{2.281}$$

The cost function components are the process noise vector, $\mathbf{w}_i$, the inverse process noise matrix, Q_i^{-1}, the observation vector, $\mathbf{y}_{i+1}$, the computed observation from the observation equation, $\mathbf{h}_{i+1}(\mathbf{x}_{i+1})$, and the inverse matrix of observation variance values, R_{i+1}^{-1}.

The quantity, $0.5(\mathbf{x}_{k-m} - \hat{\mathbf{x}}_{k-m}^*)^T (P_{k-m}^*)^{-1}(\mathbf{x}_{k-m} - \hat{\mathbf{x}}_{k-m}^*)$, is an approximation of the optimal cost function, $J_{opt[k-m]}(\mathbf{x}_{k-m})$. This optimal cost function retains the nonlinearities in the latest *m* stages, but approximates the nonlinearities for any previous stages. The quantities $\hat{\mathbf{x}}_{k-m}^*$ and P_{k-m}^* are not the filtered a posteriori state estimate and corresponding error covariance matrix. They include information from times after t_{k-m} and are therefore not true filtered values, rather their purpose is to reasonably approximate $J_{opt[k-m]}(\mathbf{x}_{k-m})$. The actual state and covariance are computed using linear filtering and smoothing techniques. This linear filter/smoother is described in section 5. One characteristic of the BSEKF is that it filters and smoothes over the last *m* stages, and at the latest stage *k*, new smoothed state estimates are produced for each of the last *m* stages. In contrast, an EKF only produces an estimate for the latest stage.

The preceding cost function is minimized over *m* stages, i.e. measurements, in order to improve the approximations for both the measurement and dynamic equations. Because the measurement equation, $\mathbf{y}_{i+1} = \mathbf{h}_{i+1}(\mathbf{x}_{i+1}) + \mathbf{v}_{i+1}$, and the dynamic equation, $\mathbf{x}_{i+1} = \mathbf{f}_i(\mathbf{x}_i, \mathbf{w}_i)$, are both included in the Gauss-Newton cost minimization for not one, but *m* stages, the nonlinearities in both the measurement and dynamic equations are treated explicitly for those *m* stages. This yields a more accurate representation of the

cost minimization problem than either the EKF or the IEKF. The EKF implicitly uses a single Gauss-Newton iteration for each observation while the IEKF can use multiple iterations. Neither the EKF nor IEKF capture the system dynamics over *m* stages as the BSEKF does. In his paper, Mark Psiaki also compares the BSEKF to the Unscented Kalman Filter (UKF). The UKF (81) includes second order effects for the dynamic and measurement equations due to its propagation of chosen sigma points through those dynamic and measurement equations. Choosing appropriate sigma points can allow the UKF to converge more quickly and provide higher accuracy than the EKF. Psiaki suggests that the BSEKF treats more than second order effects in the dynamic and measurement functions and can therefore outperform the UKF in terms of convergence reliability and estimation accuracy. Figure 2.11 shows the performance of the BSEFK in comparison to the UKF and EKF for the problem of estimating moment of inertia parameters for attitude parameter estimation. This comparison comes from reference (2).

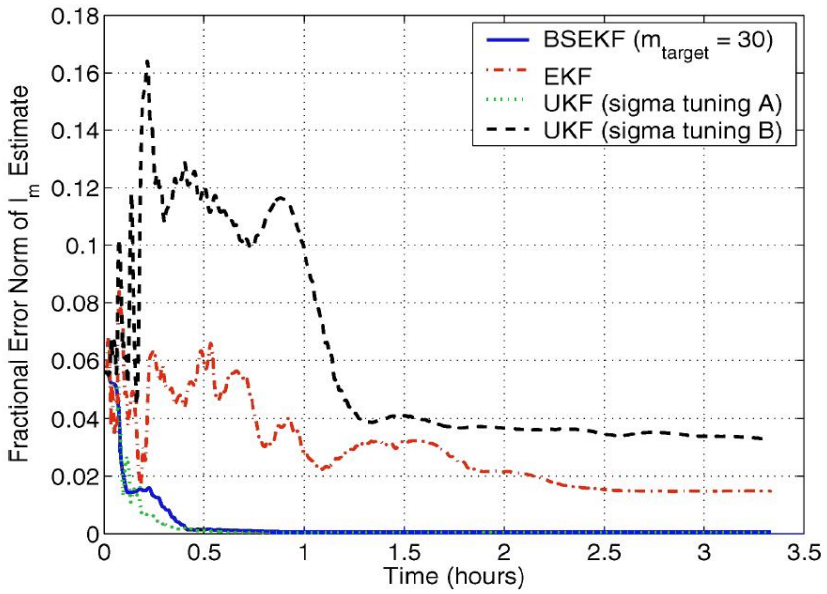

Figure 2.11 Error History of Several Filters in Estimating Moments of Inertia

In Figure 2.11, it is clear that the BSEKF is able to estimate moment of inertia parameters with higher accuracy than the EKF and the UKF with certain sigma-point tuning. In another paper (82), Mark Psiaki develops and tests a filter/smoother using the unscented transform. This sigma-points smoother is at best able to produce estimates that are of comparable accuracy to the BSEKF. In reference (82), the BSEKF is referred to as the Gauss-Newton Smoother (GNS). Because of the work done by Mark Psiaki and the favorable results he achieved, the BSEKF was chosen as the filter/smoother to be implemented for this thesis.

Because of the complexities involved in computing short periodic motion and mean element interpolation in the DSST propagator, careful attention was paid to the interaction of DSST and the BSEKF algorithm. The details of these interactions are discussed in chapter 5. Also in chapter 5, the test methodology for the BSEKF will be described and orbit estimation test cases and results for a simulated GEO satellite and LEO satellite will be shown.

[This page intentionally left blank]

Chapter 3 Extended Semianalytic Kalman Filter (ESKF) Implementation in GTDS

Stephen Taylor (10) designed the Extended Semianalytic Kalman Filter (ESKF) to couple the Extended Kalman Filter which operates on the observation time grid to the Draper Semianalytic Satellite Theory (DSST) which operates on an integration time grid using mean equinoctial element dynamics. The idea for coupling the DSST propagator to the EKF was proposed in Andy Green's thesis (35). This coupling would take advantage of the efficiency of DSST from its large allowable step sizes and the near-linear time varying behavior of the mean equinoctial solve-for state. For GEO satellites, the integration time for DSST can be large. It is usually configured with half-day grid points. This large step size is attractive, but introduces the question of when the Extended Semianalytic Kalman Filter algorithm should update the state. This question doesn't affect Cowell or numerical orbit propagators because the step size is on the order of minutes. Little time passes between receipt of an observation and the next opportunity to update the state at the next integration time step. Robert Herklotz implemented a Square Root Information Filter (SRIF) coupled to the DSST propagator (83), but this software made use of the *DSST Standalone* software rather than the R&D GTDS software.

Steve Taylor used the concept of the mean element integration grid, i.e. the time frame used by the integrator and the short periodic interpolators in DSST, and the

observation grid, i.e. the time frame defined by the observation times and thus the output times for the satellite state generated by the integrator.

The efficient implementation of DSST would degrade if relinearization of the equations of motion occurred between integration time steps. Therefore, the nominal orbit state is only updated at the integration grid points. The integrator used in DSST is Runge-Kutta so all integration steps are performed in the same way. The procedures described below (based on reference (10)) show the previous time as t_0 and the current time as t. For subsequent iterations of this procedure, the previous time is t_{k-1} and the current time is t_k.

3.1 Operations on the Integration Grid

1. At time $t = t_0$ update the nominal state for the new integration step using the predicted mean equinoctial element state, $\overline{Z}(t_0)$, and estimated filter correction, $\Delta\hat{Z}_0^0$, from the previous step and set the initial covariance, $P_0 = P_0^0$.

$$Z_N(t_0) = \overline{Z}(t_0) + \Delta\hat{Z}_0^0 \tag{3.1}$$

where $Z = \begin{bmatrix} \widetilde{a} \\ c \end{bmatrix}$, $\widetilde{a}$ is the vector of mean orbital elements and c is the vector of dynamic solve for parameters. The notation, $\hat{Z}_k^l$, indicates the estimate at time t_k given observations Y_l. If l<k, one can say that $\hat{Z}_k^l$ is a prediction yet to be corrected with the latest observation. If l=k, one can say that $\hat{Z}_k^l$ is a prediction that has been corrected with the latest observation.

Initialize the mean element filter correction and transition matrices for time $t = t_0$.

$$\Delta\hat{Z}_0^0 = 0 \tag{3.2a}$$

$$\Phi(t_0, t_0) = I \tag{3.2b}$$

$$\Psi_S = \Psi(t_0, t_0) = 0 \tag{3.2c}$$

$$\Phi_S = \Phi^{-1}(t_0, t_0) = I \tag{3.2d}$$

where $\Phi(t, t_0) = \dfrac{\partial \widetilde{a}(t)}{\partial \widetilde{a}(t_0)}$ and $\Psi(t, t_0) = \dfrac{\partial \widetilde{a}(t)}{\partial c}$

For subsequent times, $\Delta \hat{Z}_k^l$, Φ_S, and Ψ_S will be set using the previous observation. This is shown in phases 9 and 10 of the observation processing shown below.

Compute force evaluations for the equations of motion and variational equations $\dot{\widetilde{a}}_N(t_0), \dot{\Phi}(t_0, t_0), \dot{\Psi}(t_0, t_0), \dot{\Phi}^{-1}(t_0, t_0)$.

2. Integrate the averaged mean elements until time $t = t_0 + \Delta t$.

   Obtain $\widetilde{a}_N(t), \Phi(t, t_0), \Psi(t, t_0)$ and invert $\Phi(t, t_0)$ to get $\Phi^{-1}(t, t_0)$. Evaluate the corresponding rates to allow set up of the mean interpolators for $\widetilde{a}_N, \Phi, \Psi, \Phi^{-1}$.

3. Compute the short periodics $\varepsilon C_\sigma(\widetilde{a}_N), \varepsilon D_\sigma(\widetilde{a}_N)$ at time t_0 and t to initialize the short periodic coefficient interpolators. C_σ and D_σ are the Fourier coefficients in the Fourier expansion of the short periodic functions. In DSST, the short periodic functions are necessary for accurate recovery of the precise osculating orbit at each observation time. Fourier series expansions are used instead of direct integration of the osculating force model in order to avoid small step sizes.

The operations on the observation grid are triggered by receipt of a new observation. The observation grid procedure is followed in a loop-wise manner until no more observations are available or the next observation is later than the next integration time step. In that case, the integration step procedure described above is followed to advance the integration by one grid point. The observation grid procedure is as follows.

3.2 Operations on the Observation Grid

1. Obtain the new observations, $Y(t_k)$, at time t_k.

2. Interpolate to obtain the nominal mean elements, $\widetilde{a}_N(t_k)$, the state transition matrix, $\Phi(t_k,t_0)$ and the partials of the mean elements with respect to the dynamic parameters, $\Psi(t_k,t_0)$. Use existing $\Phi^{-1}(t_k,t_0)$ from Φ_S so there is no need to do a matrix inversion.

3. Interpolate for the short periodic coefficients and compute the short periodic functions. The ε symbol formally denotes the small magnitude of these functions.

$$\varepsilon C_\sigma(\widetilde{a}_N(t_k)),\ \varepsilon D_\sigma(\widetilde{a}_N(t_k)) \tag{3.3a}$$

$$\varepsilon\eta(\widetilde{a}_N) = \sum_{\sigma=1}^{N} \varepsilon C_\sigma \sin(\sigma\lambda) - \varepsilon D_\sigma \cos(\sigma\lambda) \tag{3.3b}$$

4. Compute the transitional matrices.

$$\Phi(t_k,t_{k-1}) = \Phi(t_k,t_0)\Phi_S \tag{3.4a}$$

$$\Psi(t_k,t_{k-1}) = \Psi(t_k,t_0) - \Phi(t_k,t_{k-1})\Psi_S \tag{3.4b}$$

5. Obtain the estimate of the predicted mean element solve-for state vector corrections and the estimate of the predicted dynamic solve-for parameter corrections.

$$\Delta\hat{\widetilde{a}}_k^{k-1} = \Phi(t_k,t_{k-1})\Delta\hat{\widetilde{a}}_{k-1}^{k-1} + \Psi(t_k,t_{k-1})\Delta\hat{c}_{k-1}^{k-1} \tag{3.5a}$$

$$\Delta\hat{c}_k^{k-1} = \Delta\hat{c}_{k-1}^{k-1} \tag{3.5b}$$

The filter correction, $\Delta\hat{Z}_{k-1}^{k-1} = \begin{bmatrix} \Delta\hat{\widetilde{a}}_{k-1}^{k-1} \\ \Delta\hat{c}_{k-1}^{k-1} \end{bmatrix}$, is known for k=1, t_0, and is known for subsequent k values using the filter update phase shown below.

6. Compute the estimate of the predicted osculating elements by summing the nominal mean elements known from interpolation, the estimated mean element corrections predicted by the state transition matrix, the small magnitude short periodic functions evaluated with the nominal mean elements, and the partials of the short periodic functions with respect to the nominal mean elements multiplied by the estimate of the predicted mean element corrections.

$$\hat{a}(t_k) = \widetilde{a}_N(t_k) + \Delta\hat{\widetilde{a}}_k^{k-1} + \varepsilon\eta(\widetilde{a}_N) + B_1\Delta\hat{\widetilde{a}}_k^{k-1} \tag{3.6}$$

where $B_1 = \dfrac{\partial\varepsilon\eta_1(\widetilde{a}_N)}{\partial\widetilde{a}_N}$

Transform the estimated osculating elements to cartesian elements

$$\hat{X}_k = X(\hat{a}(t_k)) \tag{3.7}$$

7. Compute the estimate of the predicted observation.

$$\hat{Y}_k = h(\hat{X}_k, t_k) \tag{3.8}$$

where *h(X,t)* is the deterministic model for transforming the state, *X*, into an observation.

Compute the observation residual.

$$y_k = Y_k - \hat{Y}_k \tag{3.9}$$

Compute the observation partial derivatives. H_k is computed through a linearization of the observation model about the nominal trajectory.

$$H_k = \frac{\partial h(Z_N, t_k)}{\partial \hat{Z}_N} = \frac{\partial h}{\partial a_N}\left[I + B_1 \middle| B_4\right] \tag{3.10}$$

$$B_1 = \frac{\partial \varepsilon \eta_1(\tilde{a}_N)}{\partial \tilde{a}_N} \tag{3.11}$$

$$B_4 = \frac{\partial \varepsilon \eta_1(\tilde{a}_N)}{\partial c} \tag{3.12}$$

8. Compute the predicted covariance.

$$P_k^{k-1} = \begin{bmatrix} \Phi(t_k, t_{k-1}) & \Psi(t_k, t_{k-1}) \\ 0 & I \end{bmatrix} P_{k-1}^{k-1} \begin{bmatrix} \Phi(t_k, t_{k-1}) & \Psi(t_k, t_{k-1}) \\ 0 & I \end{bmatrix}^T \tag{3.13}$$

9. Complete the update phase of the filter.

Calculate the gain: $K_k = P_k^{k-1} H_k \left[H_k P_k^{k-1} H_k^T + R\right]^{-1}$ (3.14)
where *R* is the diagonal matrix of a-priori known observation variances.

Update the state estimate correction: $\Delta\hat{Z}_k^k = \Delta\hat{Z}_k^{k-1} + K_k y_k$ (3.15)

Update the covariance: $P_k^k = (I - K_k H_k) P_k^{k-1}$ (3.16)

10. Interpolate for the transition matrix and its inverse and save for the next observation.

$$\Phi_S = \Phi^{-1}(t_k, t_0) \tag{3.17}$$

$$\Psi_S = \Psi(t_k, t_0) \tag{3.18}$$

The ESKF continues with step 1 until all observations have been processed or the next integration time is encountered. At the time of the next integration step, the operations on the integration time grid as outlined above are followed. Often, observations recorded at the same time can be processed without execution of all of the above phases. In those cases, only steps 1, 6, 7, and 9 must be executed for the subsequent observations at that time.

The Extended Kalman Filter (EKF) algorithm and the Extended Semianalytic Kalman Filter (ESKF) algorithms are very similar. However, the ESKF algorithm contains additional logic to handle the complexity in propagating the mean elements efficiently. The EKF assumes propagation of the state is simply done through integration of the equations of motion and through the variations of the orbital parameters. For non-linear systems, this requires short step sizes. The ESKF instead relies on the pre-computed mean elements and pre-computed short periodic functions computed for times both before and after the current observation time. The ESKF can then interpolate both the near-linear mean elements and the short periodic Fourier coefficients to accurately evaluate the orbit prediction at the observation time. This interpolation depends on the Fourier coefficients having smooth variations over time. Andy Green demonstrated this property in his thesis (35). The interpolation is much more efficient than the evaluation of the orbital elements at frequent integration steps. It should be noted that Leo Early developed many of the interpolation schemes used in GTDS using three point Hermite interpolators for the mean elements and state transition matrices and four point Lagrange

interpolators for the short periodic Fourier coefficients. The Extended Semianalytic Kalman Filter (ESKF) is similar to the Extended Kalman Filter algorithm described above except for accommodations for DSST which are computation of short periodic functions and osculating equinoctial elements, computation of the osculating position and velocity vector and finally, computation of the resulting observations.

[This page intentionally left blank]

Chapter 4 Backward Smoothing Extended Semianalytical Kalman Filter (BSESKF) Design

The BSESKF as it was applied for this effort closely follows the BSEKF algorithm described in detail in (2). The algorithm solves for a state vector, $\mathbf{x}_k$, along with intermediate state vectors and process noise vectors, $\mathbf{x}_i$ and $\mathbf{w}_i$, for *i=k-m,...,k-1*. The state vector and process noise solutions are chosen so they minimize the cost function given by equation (2.280), and the state vectors and process noise vectors are related to each other by the constraint given by equation (2.281). As indicated by equations (2.280) and (2.281), the cost calculation and the constraint equation are valid for the latest *m* stages previous to the latest stage, *k*. For stages before *k-m*, the equation (2.280) cost is supplemented with the term, $0.5(\mathbf{x}_{k-m} - \hat{\mathbf{x}}^*_{k-m})^T (P^*_{k-m})^{-1} (\mathbf{x}_{k-m} - \hat{\mathbf{x}}^*_{k-m})$. This term approximately accounts for the cost of all stages previous to *k-m*. The nonlinearities in the dynamic equations and measurement equations over the latest *m* stages are treated through the use of the summed part of equation (2.280) and the constraint defined by equation (2.281). Because an important feature of the BSEKF is the filter/smoother that operates over the latest *m* stages, choosing a value of *m* that could improve the convergence reliability and accuracy over other nonlinear filters is desirable. However, because of the significant computations required for the BSEKF, it is also desirable to choose a value of *m* that provides the benefits of the BSEKF, but does not include more stages than necessary. To minimize the cost function defined in equation (2.280), a

guarded Gauss-Newton iteration in an outer loop is combined with an execution of a square-root information filter (SRIF) and smoother.

In the context of orbit determination, the state vector, $\mathbf{x}$, is typically a vector containing the position and velocity of a satellite in an earth-centered, inertial coordinate system, $\{x, y, z, \dot{x}, \dot{y}, \dot{z}\}$, or the set of equinoctial elements $\{a, h, k, p, q, \lambda\}$. These two choices of orbital elements are among the most common. This thesis focuses on equinoctial elements. These elements are defined in terms of the classical Keplerian elements, $\{a, e, i, \Omega, \omega, M\}$, as follows [(11), pp. 490-492]:

$$
\begin{aligned}
a &= a \\
P_1 &= h = e\sin(\Omega + \omega) \\
P_2 &= k = e\cos(\Omega + \omega) \\
Q_1 &= p = \tan\left(\frac{i}{2}\right)\sin\Omega \\
Q_2 &= q = \tan\left(\frac{i}{2}\right)\cos\Omega \\
l &= \lambda = \Omega + \omega + M
\end{aligned}
\tag{4.1}
$$

These elements avoid singularities at zero eccentricity and zero inclination. Geostationary satellites typically have small eccentricity and inclination and so these elements are well suited.

The observations typically used for orbit estimation of satellites for space surveillance consist of radar observations and angular optical observations. The radar observations are typically provided as True Equator, True Equinox of Date (TETE)

topocentric azimuth, elevation, range, and range-rate measurements of a satellite from a given radar location. The optical observations are often True Equator Mean Equinox of Date (TEME) right ascension, declination observations measured against the star background.

The orbit estimation software system used in this study is the R&D Goddard Trajectory Determination System (GTDS). This system includes several orbit integration methods, e.g. Cowell, Draper Semianalytic Satellite Theory (DSST), PPT2, SGP4, and several others. The orbit integration method of focus in this thesis is the DSST method. This method integrates in mean equinoctial elements and only computes short-period deviations from the mean elements as necessary (15), i.e. at observation times when using DSST with an orbit estimation program. Previously, the GTDS system was modified to include the Extended Semianalytic Kalman Filter (ESKF) by Stephen Taylor (10). Elaine Wagner (84) later used the ESKF with GEO satellites. In order to preserve the efficiency of DSST when used with an Extended Kalman Filter, Stephen Taylor introduced several grids to differentiate the observation times from the integration time steps. Because of the long integration time steps, i.e. on the order of half a day, allowed in DSST, and the possibility for observations to arrive at any time, interpolators were implemented to provide accurate and efficient orbital state and state transition matrices between integrator time steps. Also, in DSST, short periodic motion, i.e. oscillations on the order of one orbital revolution, is reproduced using Fourier series. The Fourier coefficients for the short period Fourier series are interpolated between integration time steps also. To efficiently accomplish the state estimate update that is done by the Kalman Filter at

observation times, the averaged orbit was designed to be updated only at integration time steps. It is computationally expensive to update the averaged orbital state and its dependent short period Fourier coefficients, state, and transition matrix interpolators at each observation time. In fact, requiring updates to the averaged orbital state, recalculation of Fourier coefficients, and re-initialization of state and transition matrix interpolators at each observation time would defeat the purpose of allowing long integration step sizes in DSST. Accuracy between integrator time steps is maintained by storing the averaged orbit as a "nominal" trajectory and also storing a running sum of updates to that nominal trajectory. Both the running sum of updates and the nominal trajectory are used to calculate the state prediction. The state prediction is in turn used to calculate the predicted measurement, $\mathbf{h}_{i+1}(\mathbf{x}_{i+1})$. The running sum of updates can be modified by the Kalman Filter, and the orbit state prediction in the Kalman Filter accounts for the previous state updates by using the existing state and state transition matrix interpolators.

The implementation of the BSEKF within the GTDS software framework, i.e. the Backward Smoothing Extended Semianalytic Kalman Filter (BSESKF), uses the DSST propagator to provide state dynamics and also retains much of the efficiency achieved by the ESKF. The separated grids, short period Fourier series, and interpolation schemes were reused for the BSESKF. Nevertheless, the most challenging aspect of the implementation was carefully coupling the BSESKF estimator to the system dynamics computed by DSST. The following section explicitly describes the BSESKF algorithm implemented in GTDS.

4.1 Detailed BSESKF Algorithm Description

4.1.1 Operations on the Observation Grid

For the following algorithm is adapted from Mark Psiaki's BSEKF algorithm and uses a Square Root Information Filter (SRIF) from Bierman [(79), pp. 69-76, 115-122, and 214-217]. The following notation is used: $\mathbf{a}_c^b$ refers to a vector **a** at time point c for iteration b. All time points are referenced from the current time, *k*. Often, the time points are incremented from *k*-*m* to *k*-1 meaning that *m* time points are incremented. The collection of *m* previous states, observations, covariance square roots, and process noise vectors is referred to in the algorithm as the m-buffer.

1) Set m=0, k=1, j=0, and assign the initial guesses for the state and process noise, i.e. $\mathbf{x}_{k-m}^j$, and $\mathbf{w}_{k-m}^j, \mathbf{w}_{k-m+1}^j, ..., \mathbf{w}_{k-1}^j$. The initial state guess is the set of orbital elements such that $\mathbf{x} = \{a, h, k, p, q, \lambda\}$. The initial guesses for $\mathbf{w}$ are typically zero. Set the initial covariance, P_0, and factor it using Choleski decomposition: $P_0 = [R_{xx}^{-1}][R_{xx}^{-T}]$. R_{xx} is the square root information matrix associated with the state, **x**, and will be used later in the SRIF. Choose a value for m_{target}.
2) Begin the observation loop. The counter, *k*, is used to identify each observation.
3) If m_{target} observations have been processed, i.e. $k \geq m_{target}$, the m-buffer has been filled and values will be replaced rather than appended. Perform the following assignments:
$$\Delta\mathbf{z}_{k-m} = \Delta\mathbf{z}_{k-m+1}$$
$$\mathbf{x}_{k-m} = \mathbf{x}_{k-m+1}$$
$$R_{xx(k-m)} = R_{xx(k-m+1)}$$
$$\mathbf{w}_{k-m} = \mathbf{w}_{k-m+1}, \cdots, \mathbf{w}_{k-2} = \mathbf{w}_{k-1}, \mathbf{w}_{k-1} = 0$$
$$\hat{\mathbf{x}}_{k-m}^* = \mathbf{x}_{k-m} + R_{xx(k-m)}^{-1}\Delta\mathbf{z}_{x(k-m)}$$
$$P_{k-m}^* = [R_{xx(k-m)}^{-1}][R_{xx(k-m)}^{-T}]$$
4) If m_{target} observations have not yet been processed, then the m-buffer is still being filled, there is no need to perform the assignments in step 3). Instead:

$\hat{\mathbf{x}}^*_{k-m} = \hat{\mathbf{x}}_0 = \mathbf{x}_0$ and $P^*_{k-m} = P_0$.

5) Compute the covariance inverse, P^{*-1}_{k-m}, This is the covariance used in the approximation for the cost for all stages before *k-m*.
6) Retrieve observations, $\mathbf{y}_{i+1}$, compute and store the residuals, $\mathbf{v}_{i+1} = \mathbf{y}_{i+1} - \mathbf{h}_{i+1}(\mathbf{x}_{i+1})$ for *i*=*k-m*+1...*k*. Also, determine and store the measurement square root information matrix, R_i.
7) Compute and store the observation partial derivative matrix, $H_i = \partial \mathbf{h}_i / \partial \mathbf{x}_i$. H_i is an nx1 vector where n is the dimension of the state vector.
8) Starting from our state guess, $\mathbf{x}^j_{k-m}$, compute all subsequent state guesses from *i*=*k-m*+1… *k*-1 using the system dynamics, $\mathbf{x}_{i+1} = \mathbf{f}_i(\mathbf{x}_i, \mathbf{w}_i)$. Also compute and store the variational partial derivative matrix, $\Phi_i = \partial \mathbf{f}_i / \partial \mathbf{x}_i$ for *i*=*k-m*...*k*-1. Φ_i is a nxn matrix where n is the dimension of the state vector. Determine the process noise transition matrix, Γ_i. In this application, it is assumed $\Gamma_i = I$ (identity). DSST computes the state transition matrix at each integration step and then uses interpolation to compute the matrix at intermediate observation times. Because the BSEKF examines observations in the past, this interpolation scheme becomes inaccurate when applied to observations more than about three integration steps previous to the current integrator step time. This interpolation limits the value of *m* that can be reasonably used. See section 4.1.2 for details about the computation of the state transition matrix during the integration procedure.
9) Begin the Square Root Information Filter (SRIF) and Smoother. This method is taken from Bierman [(79), pp. 69-76, 115-122, and 214-217]. Start with *i*=*k-m* and assign $\Delta \mathbf{z}_{x(i)} = R_{xx(k-m)}[\hat{\mathbf{x}}^*_{k-m} - \mathbf{x}^j_{k-m}]$.
10) Obtain the process noise matrix, Q_i, the measurement noise matrix, R_i, and factor them to obtain the $R_{ww(i)}$ and $R_{vv(i)}$ matrices such that: $Q_i = [R_{ww(i)}]^{-1}[R_{ww(i)}]^{-T}$ and $R_i = [R_{vv(i)}]^{-1}[R_{vv(i)}]^{-T}$
11) Perform the following QR factorization:

The resulting T_i is an orthonormal matrix of dimension 2*n*+*l* by 2*n*+*l* where *n* is the dimensionality of the state and *l* is the dimensionality of the observation vector. In the current application we have scalar observations and our state is of dimension 6, therefore *l*=1 and *n*=6.

$$T_i \begin{bmatrix} \overline{R}_{ww(i)} & \overline{R}_{wx(i)} \\ 0 & R_{xx(i+1)} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} R_{ww(i)} & 0 \\ -R_{xx(i)}\Phi_i^{-1}\Gamma_i & R_{xx(i)}\Phi_i^{-1} \\ 0 & R_{vv(i+1)}H_{i+1} \end{bmatrix}$$

$\overline{R}_{ww(i)}$ and $R_{xx(i+1)}$ are square, nonsingular, upper-triangular matrices. Store all left hand terms.

12) Compute the vectors $\Delta\bar{\mathbf{z}}_{w(i)}$, $\Delta\mathbf{z}_{x(i+1)}$, and $\Delta\mathbf{z}_{r(i)}$ by performing the following matrix multiplication:

$$\begin{bmatrix} \Delta\bar{\mathbf{z}}_{w(i)} \\ \Delta\mathbf{z}_{x(i+1)} \\ \Delta\mathbf{z}_{r(i)} \end{bmatrix} = T_i^T \begin{bmatrix} -R_{ww(i)}\mathbf{w}_i^j \\ \Delta\mathbf{z}_{x(i)} \\ R_{vv(i+1)}\mathbf{v}_{i+1} \end{bmatrix}$$

13) If i=k-1, go to step 14, otherwise, set i=i+1 and go to step 10.

14) Compute $\Delta\mathbf{x}_k = R_{xx(k)}^{-1}\Delta\mathbf{z}_{x(k)}$ and set i=k-1.

15) Compute $\Delta\mathbf{w}_i = R_{ww(i)}^{-1}[\Delta\bar{\mathbf{z}}_{w(i)} - \bar{R}_{wx(i)}\Delta\mathbf{x}_{i+1}]$.

16) Compute $\Delta\mathbf{x}_i = \Phi_i^{-1}[\Delta\mathbf{x}_{i+1} - \Gamma_i\Delta\mathbf{w}_i]$.

17) If i-k-m, go to step 18, otherwise set i=i-1 and go to step 15.

18) We are now finished with the SRIF portion, now begin the Gauss-Newton iteration to search for the minimum arguments of the cost function in equation (2.280). Set the initial trial search step size: $\gamma = 1$.

19) Compute the candidate next guess of the smoothed solution by computing the state and process noise vectors with the addition of the corrections obtained in the SRIF in steps 14-16.

$$\mathbf{x}_{k-m}^{j+1} = \mathbf{x}_{k-m}^{j} + \gamma\Delta\mathbf{x}_{k-m}$$
$$\mathbf{w}_i^{j+1} = \mathbf{w}_i^{j} + \gamma\Delta\mathbf{w}_i \qquad \text{for } i=k\text{-}m,\ldots,k\text{-}1$$
$$\mathbf{x}_{i+1}^{j+1} = \mathbf{f}_i(\mathbf{x}_i^{j+1}, \mathbf{w}_i^{j+1}) \qquad \text{for } i=k\text{-}m,\ldots,k\text{-}1$$

20) Compute the cost, J^{j+1}, by evaluating equation (2.280). This implies that the functions, $\mathbf{h}_{i+1}(\mathbf{x}_{i+1})$, are recomputed also.

21) If $J^{j+1} \geq J^{j}$, then activate the guarding procedure by setting $\gamma = 0.5\gamma$ and go to step 19. Otherwise, go to step 22.

22) Compute the linearized prediction of the cost and determine whether convergence has been reached:

$$J_{newapprox}^{j+1} = \frac{1}{2}\sum_{i=k-m}^{k-1}\Delta\mathbf{z}_{r(i)}^T\Delta\mathbf{z}_{r(i)}$$

If $J_{newapprox}^{j+1} - J_{newapprox}^{j} \leq \varepsilon$ where ε is sufficiently small, then we have converged to the local cost minimum. If this is true or if j has gotten too large, then assign our state estimate for time k: $\hat{\mathbf{x}}_k = \mathbf{x}_k^{j+1}$ obtained in step 19. Then go to step 3 to process the next observation and set k=k+1. If we have not converged and j is not yet too large, set j=j+1 and go to step 6.

4.1.2 Operations on the Integration Grid

There are several special modifications that were made to the GTDS and DSST software in order to efficiently and accurately estimate the orbital state. The ESKF code written by Stephen Taylor was never designed to compute states and transition matrices for past time points. Recalculating past states in order to recalculate predicted observations and residuals is needed in step 6 of the BSEKF algorithm. Calculating states at past observation times involved using short periodic interpolation coefficients calculated at the most recent integration step time. Modifying the GTDS software to recalculate short periodic coefficients and re-initializing interpolators for past observation times would introduce significant additional complexity in the DSST-BSESKF interface. It was decided that for this investigation, using the current integration step short periodic coefficients and interpolators for past observations would suffice. It may be the case that this shortcut inhibits the accuracy of the BSESKF. A similar issue complicates the recalculation of the mean element state transition matrices. One performance enhancement made by Stephen Taylor in writing the ESKF was to avoid recalculating the state transition matrix for times at which it had already been computed. This enhancement eliminated recalculation when several observations were tagged with the same observation time. This performance enhancement did not reduce the accuracy of the ESKF because it did not consider recalculation of state transition matrices for past observation times as the BSESKF does. As shown in the BSEKF algorithm sequence described above, the state transition matrix must be recalculated and stored anew for each iteration even if the state transition matrix was already calculated for a given past

observation time. Therefore, the interface between the BSEKF and DSST was modified to recalculate the mean element interpolator coefficients as necessary to calculate the state transition matrices at the necessary past and present observation times. The state transition matrix calculation relies on mean element interpolation schemes similar to the short periodic coefficient interpolation used for calculating the osculating orbital elements. These mean element interpolators are calculated and are therefore most accurate for the time span between the two latest integration steps. Because of this and the complexity needed to recalculate past mean element interpolators, it was decided that the current version of the software should shorten the length of the m-buffer, i.e. reduce m_{target}, when observations are too far in the past to compute accurate state and transition matrices for them. The other approach involving recalculation of the mean element interpolators would allow a constant length m-buffer for relatively large values of $m_{target,}$ but would necessitate much higher software complexity. By running several test cases, it became apparent that the mean element interpolators are accurate enough to allow the interpolators to be used for observations as long as they are not too far in the past. For the cases examined for this thesis, three integration steps, each on the order of 0.5 days, were found to be a reasonable number of time steps during which past observations could be allowed in the m-buffer.

The following section (again based on (10)) outlines the steps taken on the integration grid time scale in the BSESKF software.

1. At time $t = t_0$ update the nominal state for the new integration step using the predicted mean equinoctial element state, $\overline{Z}(t_0)$, and estimated filter

correction, $\Delta\hat{Z}_0^0$, from the previous step and set the initial covariance, $P_0 = P_0^0$.

$$Z_N(t_0) = \overline{Z}(t_0) + \Delta\hat{Z}_0^0 \tag{4.2}$$

where $Z = \begin{bmatrix} \widetilde{a} \\ c \end{bmatrix}$, $\widetilde{a}$ is the vector of mean orbital elements and c is the vector of dynamic solve for parameters. The notation, $\hat{Z}_k^l$, indicates the estimate at time t_k given observations Y_l. If l<k, one can say that $\hat{Z}_k^l$ is a prediction yet to be corrected with the latest observation. If l=k, one can say that $\hat{Z}_k^l$ is a prediction that has been corrected with the latest observation.

Initialize the mean element filter correction and transition matrices for time $t = t_0$.

$$\Delta\hat{Z}_0^0 = 0 \tag{4.3a}$$
$$\Phi(t_0, t_0) = I \tag{4.3b}$$
$$\Psi_S = \Psi(t_0, t_0) = 0 \tag{4.3c}$$
$$\Phi_S = \Phi^{-1}(t_0, t_0) = I \tag{4.3d}$$

where $\Phi(t, t_0) = \dfrac{\partial\widetilde{a}(t)}{\partial\widetilde{a}(t_0)}$ and $\Psi(t, t_0) = \dfrac{\partial\widetilde{a}(t)}{\partial c}$

For subsequent times, $\Delta\hat{Z}_k^l$, Φ_S, and Ψ_S will be set using the previous observation.

Compute force evaluations for the equations of motion and variational equations $\dot{\widetilde{a}}_N(t_0), \dot{\Phi}(t_0, t_0), \dot{\Psi}(t_0, t_0), \dot{\Phi}^{-1}(t_0, t_0)$.

2. Integrate the averaged mean elements until time $t = t_0 + \Delta t$.

   Obtain $\widetilde{a}_N(t), \Phi(t, t_0), \Psi(t, t_0)$ and invert $\Phi(t, t_0)$ to get $\Phi^{-1}(t, t_0)$. Evaluate the corresponding rates to allow set up of the mean interpolators for $\widetilde{a}_N, \Phi, \Psi, \Phi^{-1}$.

3. Compute the short periodics $\varepsilon C_\sigma(\widetilde{a}_N), \varepsilon D_\sigma(\widetilde{a}_N)$ at time t_0 and t to initialize the short periodic coefficient interpolators. C_σ and D_σ are the Fourier coefficients in the Fourier expansion of the short periodic functions. In DSST, the short

periodic functions necessary for accurate recovery of the precise osculating orbit at each observation time are represented as a Fourier series. Fourier series expansions are used instead of direct integration of the osculating force model in order to avoid small step sizes.

4.2 Incorporation of the BSESKF Software into GTDS

The method of application of the BSEKF to the orbit estimation problem was to modify the R&D Goddard Trajectory Determination System (GTDS) to include the BSESKF as a subprogram. This approach saved time and effort by making use of GTDS's high precision orbital dynamic propagators, measurement processing, and overall software system infrastructure. GTDS includes a special perturbations (Cowell) propagator and also the Draper Semianalytic Satellite Theory (DSST). Both can be used to replicate the satellite orbital system dynamics; however, the DSST propagator was used in this estimation improvement investigation. Once the BSESKF was implemented and initially tested within the GTDS framework, comparisons of the GTDS BSESKF performance could be made with the existing GTDS ESKF subprogram. Within the common GTDS framework, the ESKF and BSESKF could be subject to the same initial conditions, measurements, process noise, and system dynamics.

Figure 4.1 shows the hierarchy of GTDS subprograms with the BSEKF included as one of these. The BSEKF was intended to be used with both Cowell and DSST propagators. When implemented in the software, it is referred to as BSEKF when referring to the subprogram and estimation software itself, as BSESKF when used with

the DSST propagator, and Cowell BSEKF when used with the Cowell propagator. This is similar to the existing usage of EKF when referring to the general GTDS software and to ESKF when referring to the EKF when used with DSST.

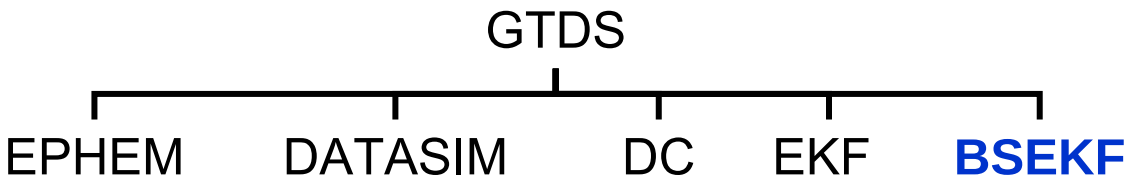

Figure 4.1 GTDS Subprogram Hierarchy

The other subprograms in Figure 4.1 include EPHEM which generates an ephemeris for a satellite with a given initial state. EPHEM can execute several propagators including the Cowell and DSST methods. DATASIM is a subprogram that reads input from a previous Cowell EPHEM execution and produces simulated observations for a list of user-defined radar and optical sensors. The DC subprogram reads a prior state and covariance. It then reads real or simulated observations and finds the posterior Linear Least Squares (LLS) estimate of the state given the observations. Assuming Gaussian observation noise and linear system dynamics and measurement functions, this also corresponds to a posterior Bayes' Least Squares (BLS) estimate. The EKF subprogram also reads a prior state, covariance, and observations and produces sequential (LLS) estimates of the posterior state at each observation time given the observations up to the current time. Both the DC and EKF linearize the system dynamics and the measurement equations. The BSEKF subprogram was designed to accept the same inputs as the EKF and present similar output to the user. Once executed, the BSEKF follows an independent program flow from the EKF and DC subprograms.

Certain aspects are reused, such as system dynamic function and measurement function evaluations.

4.2.1 GTDS Modification Summary

The BSEKF subprogram within GTDS includes many new subroutines and also reuses several subroutines from the Kalman Filter (KF) GTDS subprogram. The program flow for the BSEKF subprogram is shown in Figure 4.2.

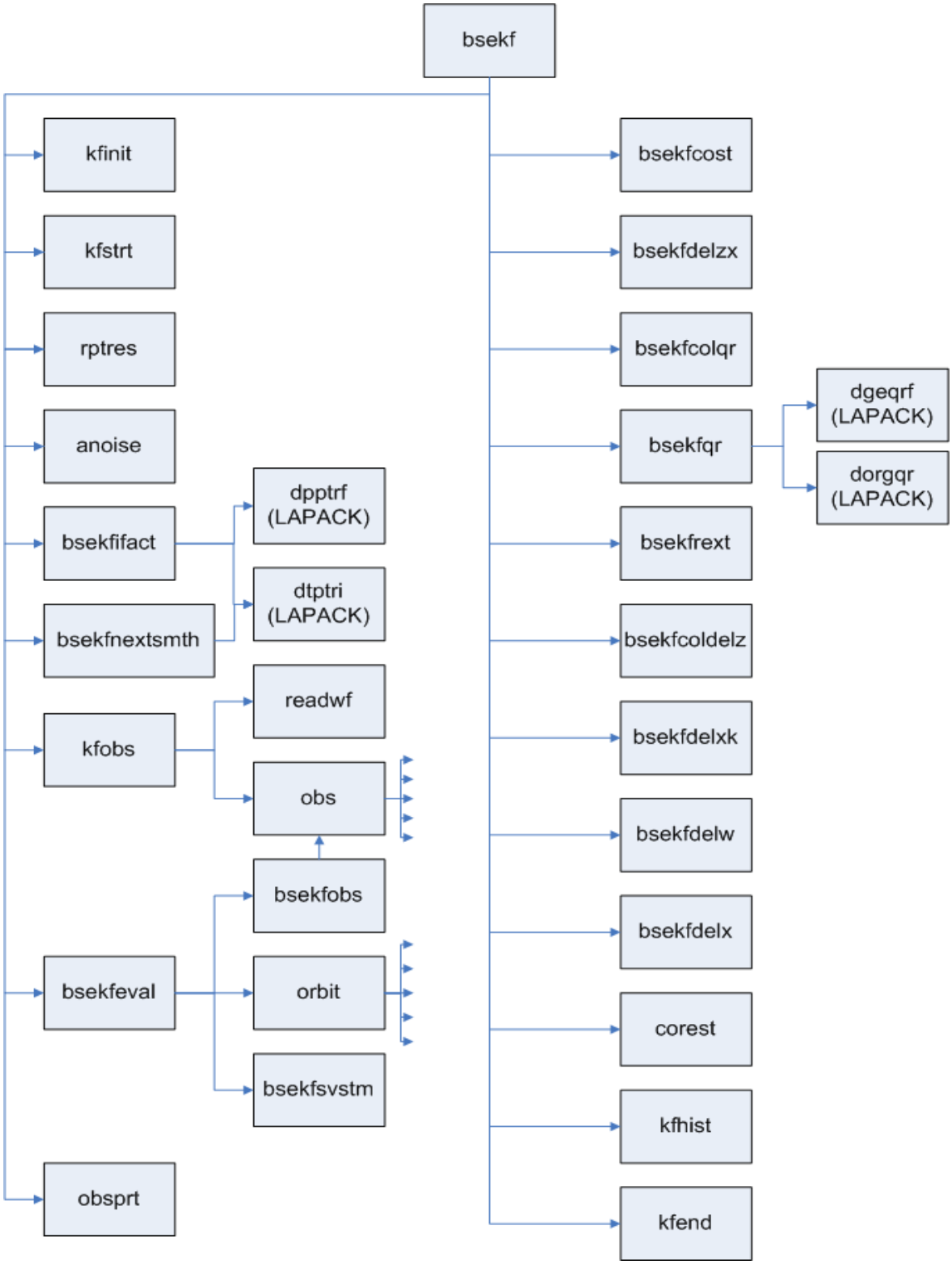

Figure 4.2 BSEKF Subprogram Subroutine Flow

4.2.2 New Subroutines

There were several subroutines added to the GTDS source code tree to implement the BSEKF subprogram. These subroutines are new to GTDS.

BSEKF is the driver for the BSEKF subprogram. It is called by the GTDS driver subroutine, ODSEXEC. This subroutine implements the algorithm from Mark Psiaki's paper, "Backward Smoothing Extended Kalman Filter." This subroutine calls many new subroutines and several subroutines that already existed as part of the KF subprogram.

BSEKFIFACT computes and returns an inverse Cholesky factorization of an input matrix. This factorization is important for the square root information filter/smoother used within the BSEKF. This subroutine uses the LAPACK subroutines DTPTRI and DPPTRF to perform the Cholesky factorization.

BSEKFNEXTSMTH computes the state vector and covariance matrix used to initialize the BSEKF algorithm when a new observation is about to be processed. The state vector and covariance matrix are computed recursively using the results from the last observation.

BSEKFEVAL predicts the state and state transition matrix at the requested observation time. Also, the observation and residual are stored in the necessary buffers.

BSEKFCOST computes the scalar cost given the process noise vectors, the residuals and the predicted state and covariance.

BSEKFDELZX computes a necessary information vector used in the square root information filter/smoother.

BSEKFCOLQR collects necessary matrices and assembles them into the block matrix used in a later QR factorization.

BSEKFQR calls the DGEQRF and DORGQR LAPACK subroutines to perform the QR factorization and to assemble the results required in this implementation of the square root information filter/smoother.

BSEKFREXT extracts matrices from the block matrix result of a QR factorization.

BSEKFCOLDELZ computes needed vectors using the result of a QR factorization.

BSEKFDELXK computes the kth state vector change in the smoothing part of the square root information filter/smoother.

BSEKFDELW computes the process noise vector change in the smoothing part of the square root information filter/smoother.

BSEKFDELX computes the ith state vector change in the smoothing part of the square root information filter/smoother.

BSEKFSVSTM computes and saves the inverse state transition matrix for a given time into a buffer so that it can be used in later computations involving the state transition matrix.

A few of the subroutines listed above made use of subroutines from the LAPACK linear algebra library package. This package can be accessed from the netlib website at, http://www.netlib.org/lapack/. This package was chosen because it is well known as a reliable FORTRAN linear algebra package. The subroutines from this package that were used include the following.

DGEQRF computes a QR factorization of a real M-by-N matrix A: A = Q * R.

DORGQR generates an M-by-N real matrix Q with orthonormal columns, which is defined as the first N columns of a product of K elementary reflectors of order M, Q = H(1) H(2) . . . H(k), as returned by DGEQRF.

DTPTRI computes the inverse of a real upper or lower triangular matrix A stored in packed format.

DPPTRF computes the Cholesky factorization of a real symmetric positive definite matrix A stored in packed format. The factorization has the form: $A = U^T * U$, if UPLO = 'U', or $A = L * L^T$, if UPLO = 'L', where U is an upper triangular matrix and L is lower triangular.

4.2.3 Modified Subroutines

Several source code files were changed to add the BSEKF subprogram to GTDS. The following files were modified.

ODSEXEC is the executable subroutine for GTDS. It is the first subroutine called upon execution of GTDS. Comments and code were changed to add the BSEKF subprogram to GTDS. The variable INDRUN is set to 9 for the BSEKF subprogram after the SETRUN subroutine finishes and ODSEXEC then calls the BSEKF subroutine.

LNDMRK computes landmark observables for a spinning satellite. BSEKF was included as an estimator along with the existing KF and DC subprograms. The variable IND48 is equal to 9 for the BSEKF.

GVCVL generates the title array for solve-for and consider parameters. The case when INDRUN equals 9 for the BSEKF subprogram is included as an estimator. The

same steps that are taken for the KF and DC subprograms are now also taken for the BSEKF subprogram.

SORREG sorts the regression arrays for the differential correction and filter subprograms. The regression arrays are the arrays storing the partial derivatives of the observation equation with respect to the state variables at a given observation time. Here also, the case when INDRUN equals 9 for the BSEKF subprogram is handled in the same way as for the KF and DC estimators. Here, the number of solve for parameters is incremented only when running the DC. The KF and BSEKF are handled separately.

SNGSTP is a subroutine to initialize necessary arrays for the single step integration of the VOP equations. When INDRUN is equal to 9 for the BSEKF, the same step to update the state update flag that is taken for the KF is now also taken for the BSEKF subprogram.

SETRUN is an initialization subroutine that reads the GTDS keyword cards for each run. Here, INDRUN is set to 9 for the BSEKF subprogram. Modifications to include the BSEKF subprogram as a valid subprogram were made. This includes allowing the ELEMENT1 - ELEMENT7, OBSINPUT, ORBTYPE, BSEKFOPT, and EPOCH cards, making settings for epoch advancement, creating observation working files, and running the SETBSEKF subroutine. Also, comments where changed to show the BSEKF modifications and certain variables were set to be initialized. The BSEKF

text string was added to the PROGRM array, and the BSEKFOPT string was added to the TRMCRD array to allow it as a valid keyword.

RKINTG integrates the equations of motion and the variational equations using a Runge-Kutta method. This subroutine was modified so that the same processing done for the KF subprogram is also done for the BSEKF subprogram. Specifically, this includes interpolating for the inverse of the state transition matrix.

RESINV initializes parameters needed to start another integration span beginning at the epoch time. This is in conjunction with the VOP orbit integrators, namely, the subroutines RKINTG and ORBITV. Here, modifications were made to treat the BSEKF subprogram in the same way the KF is treated.

POSRES computes position residuals given actual and computed observations and accumulates position residual statistics. Modifications were made to accumulate statistics for the BSEKF in the same way they are accumulated for the KF.

ORBITV is one of the subroutines called by the main ORBIT subroutine which drives the GTDS orbit generators. ORBITV drives the orbit generation for the Draper Semianalytic Satellite Theory (DSST) averaged VOP equations of motion. Specifically, ORBITV integrates the equations of motion to obtain position and velocity of the satellite at a requested time. This subroutine was modified so that the BSEKF is treated like the KF subprogram in calling the SKFUDT and ORBSKF subroutines.

ORBSKF performs computations for the Semianalytic Kalman Filter. This includes predicting the state correction and calling SKFPRT to compute the state transition matrix. ORBSKF was modified to call SKFPRT regardless of whether it was previously called for the given observation time. One performance enhancement added to the Extended Semianalytic Kalman Filter (ESKF) by Stephen Taylor was to only call the SKFPRT subroutine when a new time point is encountered. Because of the necessity of recalculating the state transition matrix in the BSEKF for past observation times, this performance enhancement is bypassed for the BSESKF.

SKFPRT computes the partial derivative (state transition) matrices via short arc interpolation and the averaged interpolator. This interpolation improves the performance over methods that involve recomputing the partical derivatives explicitly. A modification to this subroutine was made so that the state transition matrix is computed by interpolation regardless of whether the time requested is earlier than the last time requested. This ensures that the state transition matrix supplied to the BSESKF is as accurate as possible with current software. This subroutine should be changed in the future to avoid interpolation for request times that are outside the valid interpolation range. The interpolation range includes times between the last integration time step and the current integration time step.

OBSTRK computes estimated observations for the differential correction (DC), data simulation (DATASIM), Kalman Filter (KF) and now Backward Smoothing

Extended Kalman Filter (BSEKF) subprograms. This subroutine was modified to treat the BSEKF subprogram in the same way the KF subprogram is treated.

OBSPRT retrieves the partial derivatives of the observation equations with respect the the state at the given time by calling the OBSP subroutine. This subroutine was modified so the BSEKF subprogram is treated like the KF subprogram.

OBSPCE computes position and velocity observations. This subroutine was modified to treat the BSEKF like the KF.

OBSPCE_ELSET computes single-averaged equinoctial element observables. This subroutine was modified to treat the BSEKF like the KF.

INTPPT initializes parameters required for the NAVSPASUR General Perturbation Theory. This subroutine was modified to treat the BSEKF like the KF and DC subprograms.

INTOGS initializes parameters for the NORAD General Perturbation Theories. This subroutine was modified to treat the BSEKF like the KF subprogram.

INTOGN initializes parameters for the orbit generator program (EPHEM) which are directly derivable from input, permanent files, or block data and which are not

changed after a DC iteration. Here, the BSEKF is treated like the DC and KF subprograms.

WFCONT creates the working files using input from permanent files and user-supplied input files. This subroutine was modified to treat the BSEKF in the same way the KF and DC subprograms are treated.

SETDC processes keywords that are input as part of a DC run. However, some parameters are also input for the KF and new BSEKF subprogram. Here, the BSEKF subprogram is treated in the same way as the KF subprogram.

PSET resets dynamic solve-for parameters and tracking station positions adjusted by the DC and KF subprograms. This subroutine was modified to treat the BSEKF in the same way as the KF subprogram.

OUTPUT is a driver for several output subroutines. Here the BSEKF subprogram is now treated in the same way as the KF.

OBSLMK computes landmark observations and was modified to treat the BSEKF subprogram in the same way as the KF.

OBGPS1 computes GPS pseudo-range and delta-range observations. It was modified to treat the BSEKF the same as the KF.

LNDPRT computes observation partial derivatives for landmark observations. It was modified to treat the BSEKF in the same way as the KF.

GPSSEE checks the visibility of a satellite from a GPS satellite. It was modified to treat the BSEKF in the same way as the KF.

GPSPR2 computes observation partial derivatives for GPS observations. It was modified to treat the BSEKF in the same way as the KF.

GPSPR1 also computes observation partial derivatives for GPS observations. It was modified to treat the BSEKF in the same way as the KF.

ESKFOUT prints out run-time status of the KF and its options. It was modified to treat the BSEKF in the same way as the KF.

SWITCHBD is a block-data initialization subroutine. Its comments were changed to reflect the addition of the BSEKF subprogram.

OUTTIC computes trajectory initial conditions and prints them. It was modified to also make these computations and print them for the BSEKF.

4.3 Test Methodology

The test methodology involved writing software to run both the GTDS BSESKF and the GTDS ESKF programs on the same input. The input included initial states which could be perturbed somewhat from truth in order to test filter convergence. The input also included an initial covariance matrix reflecting the variance of the initial state values. Other input information included a white process noise matrix and simulated observations with added Gaussian noise. The initial state was passed to a GTDS ephemeris generator run. This ephemeris generator created an ephemeris which was passed to a GTDS data simulation run. The data simulation created simulated observations. The observations consisted of range, azimuth and elevation observations from the following geodetic sensor locations shown in Table 4.1.

Table 4.1 Simulated Observation Sensor Locations

Name	Latitude	East Longitude
MIL	42° 37’ 2’’	288° 30’ 32’’
HAY	42° 37’ 23’’	288° 30’ 42’’
ATV	9° 23’ 43’’	167° 28’ 45’’
KPT	21° 34’ 19’’	201° 44’ 00’’

The actual observation timing varies by test case and is described later in the test case descriptions.

Once the observations were generated, they were passed to the BSESKF and ESKF filter programs. The filter programs were set to accept all observations, i.e. no outlier rejection occurred. The initial state given to the filter programs could be perturbed from the initial, “truth,” state used to generate the ephemeris and the subsequent simulated observations. This perturbation was included to test the relative

response of the filters to an initial estimate with some amount of error. The initial covariance matrix given to the BSESKF and ESKF filters was also an input. This initial covariance could be set to anything, but for the test cases in this thesis, the initial covariance matrix was a diagonal matrix in which the diagonal terms were set approximately to the square of the difference of the perturbed elements from the initial "truth" elements. This matrix was *n*x*n* where n is the number of solve-for parameters. In the cases done for this thesis, *n*=6 for the 6 orbital parameters or *n*=7 if a drag or solar radiation pressure parameter was also included. The initial process noise matrix is also an input to both the BSESKF and ESKF filters. This process noise matrix was a diagonal noise matrix for the test cases in this thesis and was the same dimension as the initial covariance matrix. The input process noise matrix was constant for all test cases with diagonal elements, [$1.0\text{x}10^{-18}$, $1.0\text{x}10^{-25}$, $1.0\text{x}10^{-26}$, $1.0\text{x}10^{-24}$, $1.0\text{x}10^{-19}$, $1.0\text{x}10^{-17}$]. These 6 diagonal elements correspond to the 6 mean equinoctial orbital element state parameters, i.e. $\{a, h, k, p, q, \lambda\}$. If the coefficient of drag was estimated as a solve-for parameter, the process noise diagonal element associated with it was $1.0\text{x}10^{-18}$. The BSESKF and ESKF sometimes reacted differently to the process noise values. Many process noise matrices were attempted with some causing either the BSESKF or the ESKF to diverge. These process noise values allowed both filters to converge for a large number of test cases. The BSESKF program also required an input to specify the maximum size of the m-buffer. In the evaluations for this thesis, *m* was typically set to 12, 24 or 48 measurements or 4, 8 or 16 observation triplets, respectively.

Once the input values were passed to the BSESKF and ESKF filters, they were executed and the output from each filter was automatically collected. The output consisted of measurement residuals, updated orbital elements and other solve-for parameters, and updated covariance matrices. For this thesis, the initial primary interest was in the accuracy of the filters. To quantify the accuracy of each filter, the output orbital elements were compared with the orbital elements from the "truth" ephemeris. Because the truth ephemeris was generated using a Cowell ephemeris generation, a direct comparison of the truth and filter output required that the truth ephemeris be transformed to the coordinate system of the orbital elements used in the filter programs. To do this, the truth ephemeris was fit using iterative, nonlinear Bayes' Least Squares (BLS), i.e. differential correction (DC). The DSST propagator was used with the DC to estimate the best mean equinoctial element ephemeris representative of the truth ephemeris yet compatible with the filter solve-for state. Because the BLS fit had full observability of the position and velocity with many data points, it was able to reproduce the original ephemeris to a very high degree of accuracy. This procedure is sometimes called Precise Conversion of Elements (PCE) (26). In order to quantify the accuracy of the BSESKF and ESKF relative to the truth orbital elements, plots of the output vs. time were generated. In this way, comparison of the truth ephemeris and the filter output state was accomplished by plotting the filter results against the truth ephemeris. Difference plots were also generated because the mean elements change over time and in some cases, the orbital element differences between the ESKF, truth, and the BSESKF were small relative to the magnitude of the orbital element in question.

4.4 Simulation Test Case Results

The test cases included LEO satellite test cases and GEO test cases. The initial Keplerian orbital elements for the LEO and GEO orbits are shown in Table 4.2.

Table 4.2 LEO and GEO Mean Orbital Elements for Test Cases

	LEO Elements	GEO Elements
Epoch	Jan 18, 2003 00:00:00	Mar 20, 2004 00:00:00
Semimajor axis (km)	6643	42165.56
Eccentricity	8.9×10^{-2}	3.062×10^{-2}
Inclination (deg)	38	6.024
RAAN (deg)	214	71.373
Arg of Perigee (deg)	344	307.091
Mean Anomaly (deg)	74	118.653
Drag Coefficient	2.0	N/A

As described in the test case methodology, these initial elements were used to generate simulated observations from the sensors in Table 4.1. The modeling used to generate the truth ephemeris for the LEO test case included 30x30 geopotential terms from the EGM96 model, Jacchia-Roberts atmospheric drag, lunar and solar point mass gravity and Earth polar motion. In the LEO test cases, observations were simulated for a span of six days and were generated for each sensor when the satellite was geometrically visible, when the elevation angle with respect to the sensor was at least 15 degrees, and when the satellite pass was at least 600 seconds in duration. In the simulation, all four sensors observed the satellite during the six day span and the total number of observations (range-azimuth-elevation triplets) was 777. The modeling used to generate the GEO truth ephemeris included 8x8 geopotential, lunar and solar point mass gravity and solar radiation pressure modeling. In the GEO test cases, observations were only simulated for the MIL and HAY sensors. These sensors have very close geographic locations and so

present a challenging test case for both the ESKF and BSESKF. Geometrically, observability improves when observing sensors are geographically distant. The observations were generated once every six hours, and were generated for a total span of ten days. The exact time between observations was varied somewhat so that observations from both sensors were not exactly the same. The total number of observations (range-azimuth-elevation triplets) was 255.

White, zero-mean, Gaussian measurement noise was also included in the data simulation. Table 4.3 shows the standard deviation for the noise for each sensor and observation type. These measurement errors were chosen to be realistic for radar sensors that have the capability to track GEO and LEO satellites. For the GEO test case, only MIL and HAY were simulated.

Table 4.3 Sensor Measurement Noise Standard Deviations

Sensor Name	Measurement Type	Standard Deviation
MIL	Azimuth, Elevation	18 arc-seconds
	Range	10 meters (LEO), 5 meters (GEO)
HAY	Azimuth, Elevation	18 arc-seconds
	Range	10 meters (LEO), 3 meters (GEO)
ATV	Azimuth, Elevation	18 arc-seconds
	Range	10 meters (LEO)
KPT	Azimuth, Elevation	67 arc-seconds
	Range	23 meters (LEO)

To test the accuracy and convergence characteristics of the ESKF and BSESKF for these test cases, the LEO and GEO initial elements were perturbed from the elements used to generate the truth ephemeris and simulated observations. In the LEO and GEO test cases, the differences in the perturbed elements from the initial elements are shown in Table 4.4. The actual elements passed to the filters are the set of mean equinoctial elements rather

than the Keplerian elements shown. Because most readers may be more familiar with Keplerian elements, the transformation using equation (2.30) was used to calculate the elements shown in Tables 4.2 and 4.3. It should be noted that for most applications, the equation (2.30) transformation is only valid for osculating elements and is generally not valid for mean elements. The equation (2.30) transformation was only used here for presentation purposes.

Table 4.4 LEO and GEO Perturbed minus Initial Truth Mean Orbital Elements

	LEO Perturbations (case 1)	LEO Perturbations (case 2)	GEO Perturbations (case 3)	GEO Perturbations (case 4)
Epoch	Jan 18, 2003 00:00:00	Jan 18, 2003 00:00:00	Mar 20, 2004 00:00:00	Mar 20, 2004 00:00:00
Semimajor axis (km)	10	10	12	65
Eccentricity	$5x10^{-5}$	$5x10^{-5}$	$1.5x10^{-7}$	$7x10^{-5}$
Inclination (deg)	1.6	2.8	0.007	0.7
RAAN (deg)	0.28	2.3	0.02	4.8
Arg of Perigee (deg)	11	2.5	0.04	19
Mean Anomaly (deg)	19	0.3	0.4	95
Drag Coefficient	0.0	0.0	N/A	N/A

The initial diagonal covariance entries used for both the BSESKF and ESKF filters are shown in Table 4.5. Because the solve-for elements were equinoctial elements rather than Keplerian, the variances shown are the equinoctial variances.

Table 4.5 LEO and GEO Diagonal Covariance Entries

	LEO Variances (case 1)	LEO Variances (case 2)	GEO Variances (case 3)	GEO Variances (case 4)
Epoch	Jan 18, 2003 00:00:00	Jan 18, 2003 00:00:00	Mar 20, 2004 00:00:00	Mar 20, 2004 00:00:00
Semimajor axis (km)	$1.0\text{x}10^{4}$	$1.0\text{x}10^{4}$	$1.0\text{x}10^{2}$	$1.0\text{x}10^{4}$
h	$1.0\text{x}10^{-7}$	$1.0\text{x}10^{-4}$	$1.0\text{x}10^{-12}$	$1.0\text{x}10^{-5}$
k	$1.0\text{x}10^{-7}$	$1.0\text{x}10^{-4}$	$1.0\text{x}10^{-12}$	$1.0\text{x}10^{-5}$
p	$1.0\text{x}10^{-2}$	$1.0\text{x}10^{-2}$	$1.0\text{x}10^{-8}$	$1.0\text{x}10^{-3}$
q	$1.0\text{x}10^{-2}$	$1.0\text{x}10^{-2}$	$1.0\text{x}10^{-8}$	$1.0\text{x}10^{-3}$
λ (deg)	1.0	1.0	$1.0\text{x}10^{-2}$	1.0
Drag Coefficient	$1.0\text{x}10^{-3}$	$1.0\text{x}10^{-3}$	N/A	N/A

The covariance entries shown in Table 4.5 roughly represent the accuracy of the perturbed initial state passed to the filters. However, it was found that covariance matrices that are too optimistic about the initial state accuracy seemed to cause divergence first in the BSESKF and eventually in the ESKF. Covariance matrices that were too pessimistic caused divergence first in the ESKF and then the BSESKF. As with the process noise matrix, some experimentation was required to find covariance matrices like the ones in Table 4.5 that worked well with both the BSESKF and ESKF filters.

The modeling used in both the ESKF and BSESKF was identical to the modeling used to generate the truth orbits. It is left for future work to test BSESKF behavior when used with system models that are either more or less accurate than models used to generate the truth orbit.

Figures 4.3-4.8 show the test case results for the LEO cases. The values of *m* tried for each case were 12, 24 and 48 with each increase in *m* resulting in slightly improved accuracy over the previous value of *m*. The filter accuracy differences for semimajor axis shown in Figure 4.3 indicate that the BSESKF converges to within less than 50 meters of truth within 1.5 days while the ESKF takes about 3.5 to 4 days.

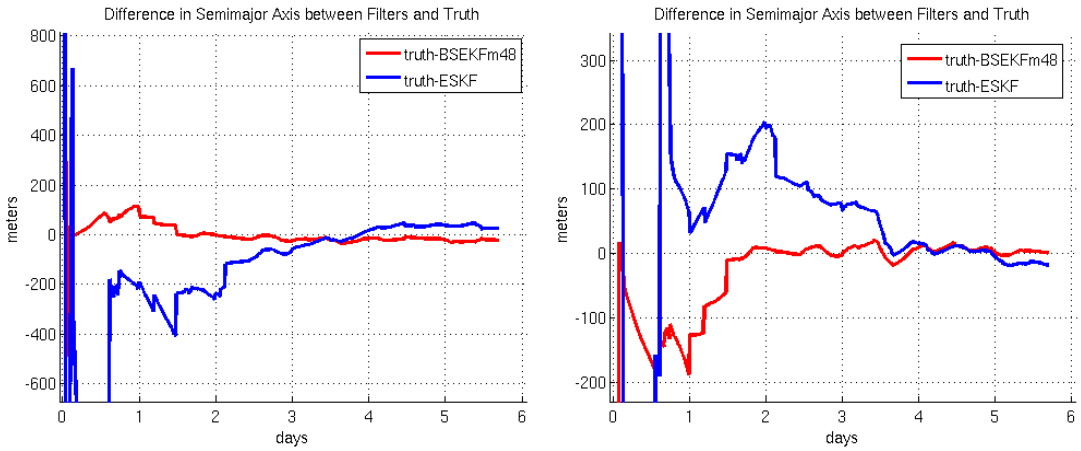

Figure 4.3 LEO mean semimajor axis state variable for cases 1 and 2

Figures 4.4 and 4.5 show the equinoctial elements h and k related to eccentricity. For both of these elements, the BSESKF converged in about 2 days. The ESKF didn't converge with comparable accuracy within the 6 day span.

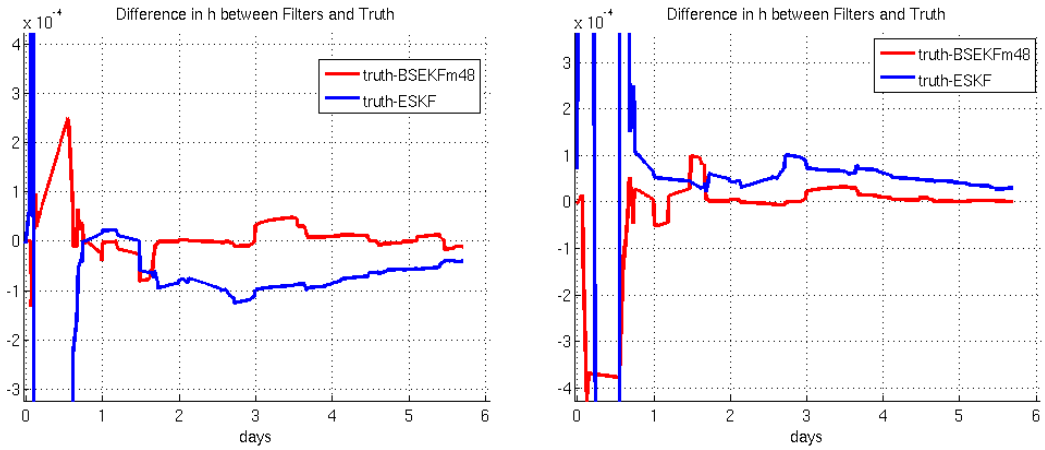

Figure 4.4 LEO mean h state variable for cases 1 and 2

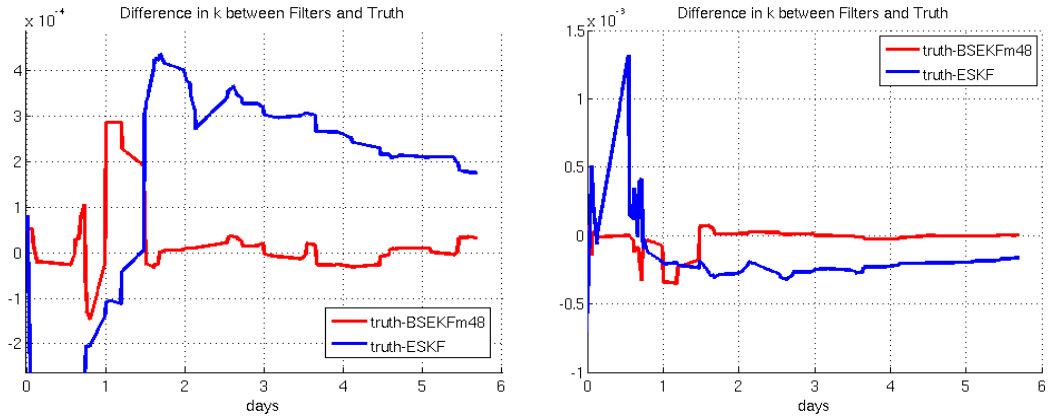

Figure 4.5 LEO mean k state variable for cases 1 and 2

Figures 4.6 and 4.7 display the p and q equinoctial elements related to inclination. The BSESKF seems to be more accurate in these cases and has a shorter and less dramatic initial transient period.

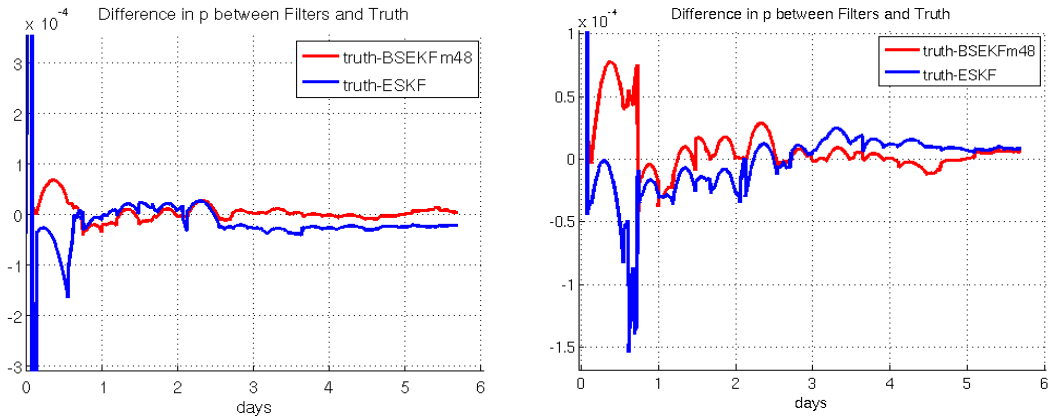

Figure 4.6 LEO mean p state variable for cases 1 and 2

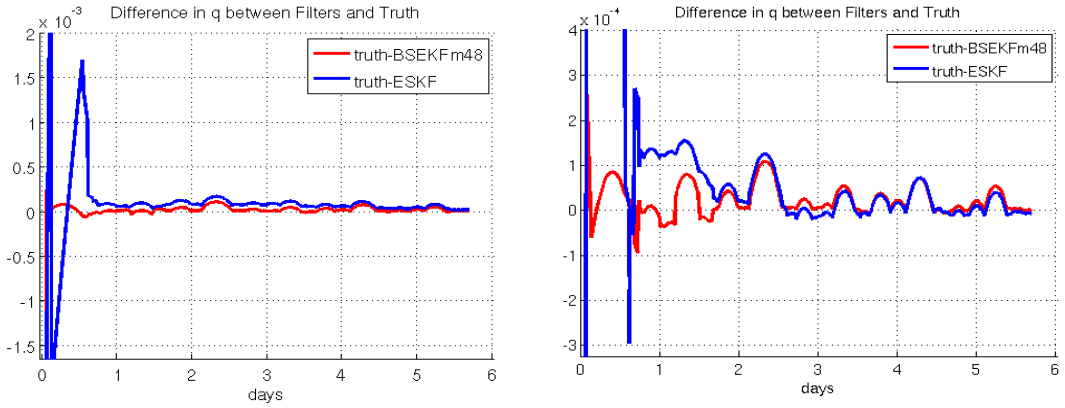

Figure 4.7 LEO mean q state variable for cases 1 and 2

Figure 4.8 shows the mean longitude element over the 6 day span. Again, the BSESKF converged more quickly and reached a more accurate steady state value than the ESKF.

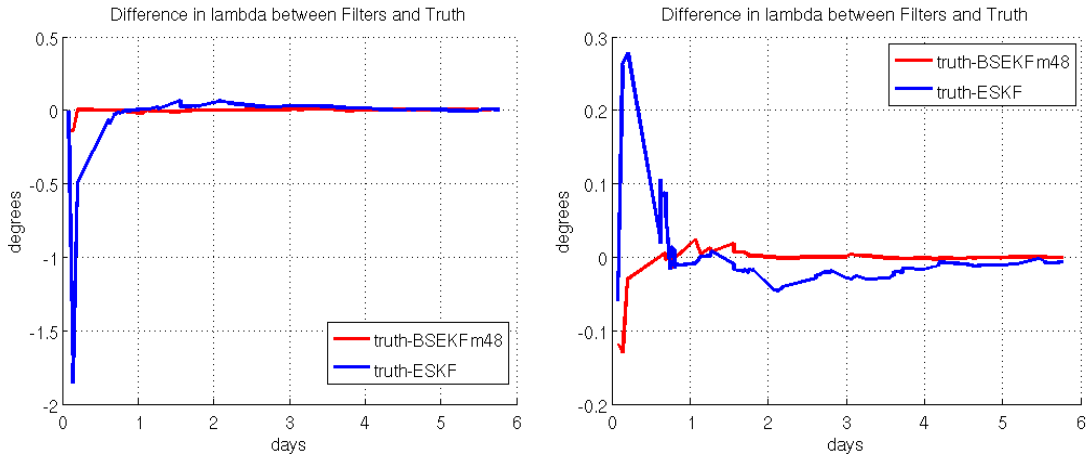

Figure 4.8 LEO mean λ state variable for cases 1 and 2

The coefficient of drag solution for each day is shown in Table 4.6. Although the ESKF began with larger errors in the drag parameter, it eventually produced smaller errors than the BSESKF. The BSESKF initially produced smaller errors, but the error remained relatively constant over the six day span.

Table 4.6 ESKF and BSESKF Drag Coefficient Solutions for LEO Case 1

Day	ESKF value	Diff. from truth	BSESKF value	Diff. from truth
1	3.282	1.282	2.058	0.058
2	2.314	0.314	2.056	0.056
3	2.020	0.020	2.057	0.057
4	1.981	0.019	2.057	0.057
5	1.986	0.014	2.058	0.058
6	1.994	0.006	2.058	0.058

Figures 4.9-4.14 show the test case results for the GEO cases. The semimajor axis differences in Figure 4.9 show that the BSESKF produced smaller errors both in the initial transient period and throughout the ten day span. This result was mirrored in the other equinoctial orbital elements shown in Figures 4.10-4.14 also. Overall, these LEO and GEO cases indicated that the BSESKF with an m-buffer of 24-48 past measurements

was able to estimate orbital elements with higher accuracy than the ESKF. In every case for each orbital element, the BSESKF exhibited equal or superior accuracy to the ESKF. In addition, the BSESKF was able to converge to an accurate estimate more quickly than the ESKF. The BSESKF accuracy did require a higher computational cost, however. The additional computational cost of the square-root information filter along with the several iterations per observation often needed for the BSESKF to converge meant that the BSESKF required about ten times as much computation time as the ESKF.

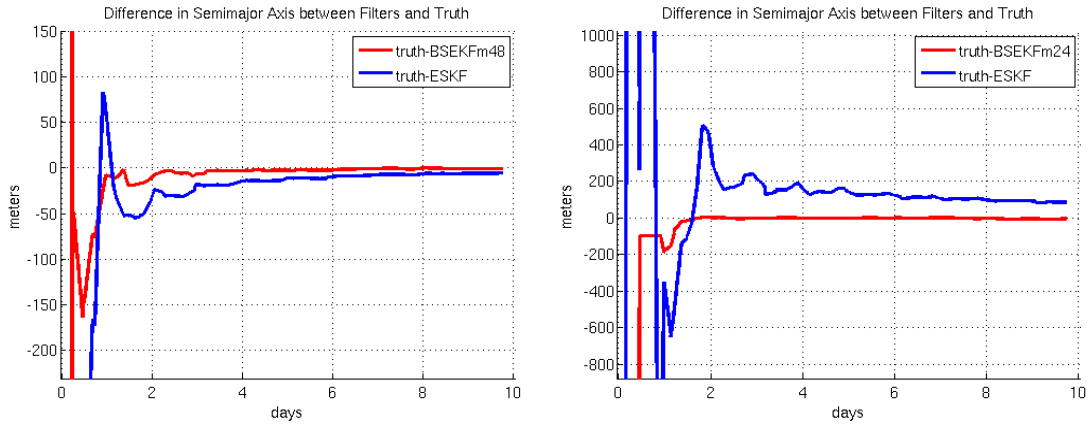

Figure 4.9 GEO mean semimajor axis state variable for cases 3 and 4

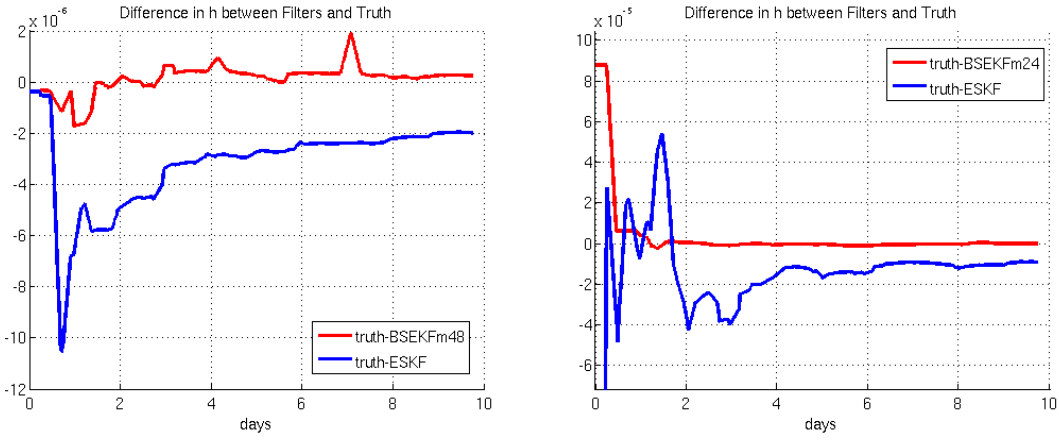

Figure 4.10 GEO mean h state variable for cases 3 and 4

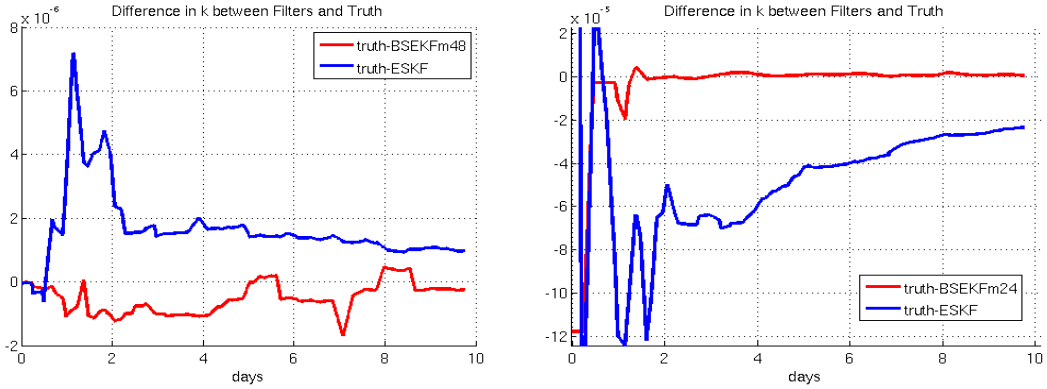

Figure 4.11 GEO mean k state variable for cases 3 and 4

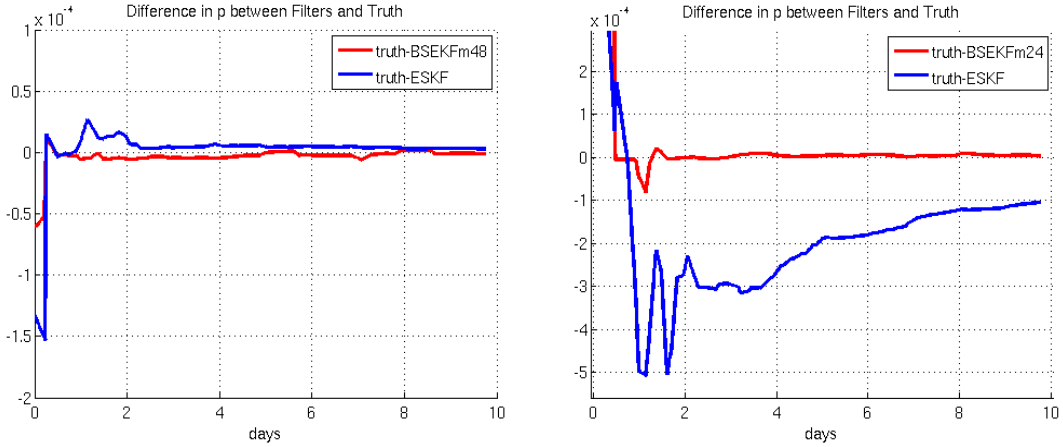

Figure 4.12 GEO mean p state variable for cases 3 and 4

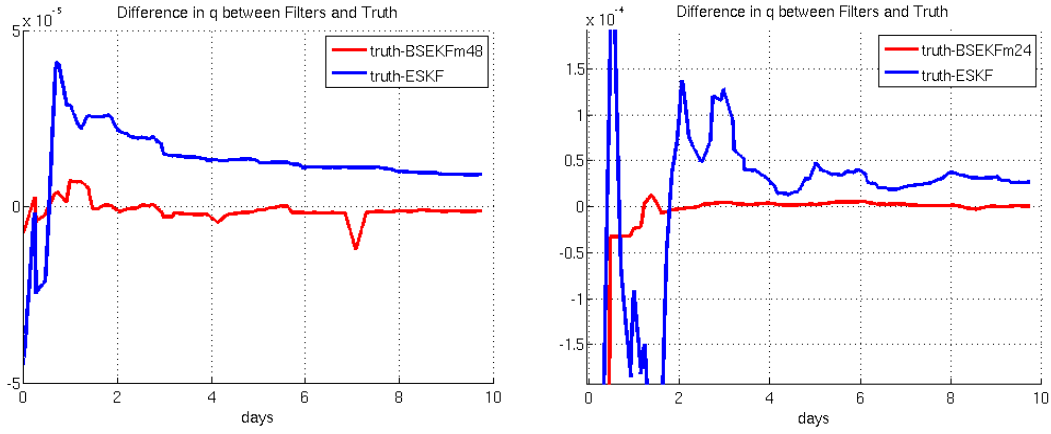

Figure 4.13 GEO mean q state variable for cases 3 and 4

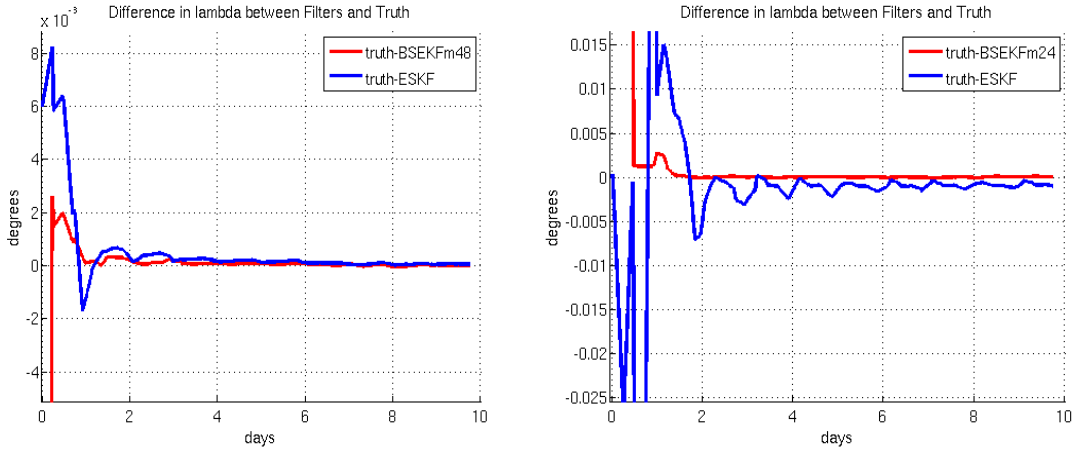

Figure 4.14 GEO mean λ state variable for cases 3 and 4

The role of the m-buffer can be illustrated by showing a case in which several m-buffer sizes were attempted. Figure 4.15 shows an example of the BSESKF behavior in the mean h equinoctial element when varying the m-buffer size. Mark Psiaki didn't indicate an m-buffer upper limit beyond which accuracy degrades. Therefore, one would expect that increasing the m-buffer size would always result in estimates with higher accuracy than smaller m-buffer sizes. However, the BSESKF with m=48 was less accurate overall than the BSESKF with m=24 or m14. In this case, it was likely that the inaccuracies due to interpolating state vectors and state transition matrices at past observation times outside the intended interpolation range was adversely affecting the accuracy of the BSESKF estimates. If this was the case, the inaccuracy for large m-buffer sizes would be due to the interface between the BSESKF and the DSST propagator rather than with the BSESKF or DSST alone.

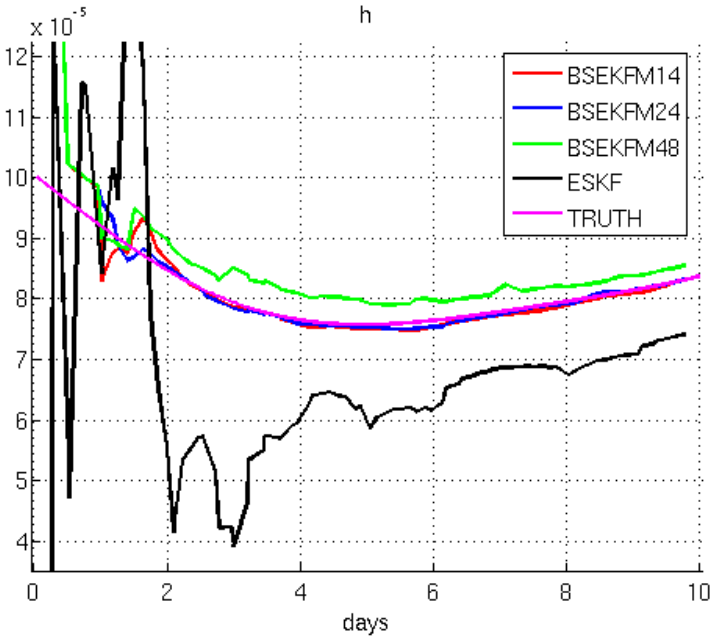

Figure 4.15 GEO mean h equinoctial element for case 4

[This page intentionally left blank]

Chapter 5 Software for Optimal Orbit Transfer Modeling

There are challenges in space surveillance analysis in predicting trajectories for satellites influenced by continuous thrust. A related challenge is in using orbit determination to refine trajectories with observations for such satellites. Incorporating accurate continuous thrust models based on optimal trajectory analysis is one way to address these challenges. Of course, the assumption that actual satellites use optimal thrust plans is perhaps not always valid, but this assumption reduces the search space for thrust plans. In addition, satellite operators are strongly influenced by the need to conserve fuel and so a time/fuel optimal thrust plan is perhaps the most probable thrust plan that can be assumed.

Because of the assumption that satellite operators use optimal thrust plans, the optimization problems and solution methods described in section 2.2 have been implemented in software to provide tools for generating optimal thrust plans. The software environment is a PC running the Linux operating system. Versions of the software were initially written in Matlab® and were later written in FORTRAN to take advantage of its higher performance. The Intel® FORTRAN Compiler version 9.1 was used to compile the FORTRAN code. The Intel® FORTRAN compiler was chosen because it has been used to compile the R&D GTDS source code. The software was designed to be given an initial orbit and destination orbit with either a given total transfer time or a given constant thrust acceleration magnitude. If given a total transfer time, the necessary thrust acceleration magnitude can be solved. If the thrust acceleration magnitude is known, the total transfer time can be solved. The thrust plan generated by

the optimal thrust plan tool can then be used in orbit determination as an additional force model. This will hopefully aid in more accurately predicting and refining orbits using orbit determination methods. This second part of the task is markedly different from the first and so the software was written in two parts. The first part is a standalone module that produces the optimal thrust plans. The second part was implemented as a force model in the R&D Goddard Trajectory Determination System (R&D GTDS) because the tool already had the software infrastructure for orbit prediction and orbit determination with observations. The original intention was to apply the thrust force model in GTDS to both the Cowell and DSST propagators and to ESKF, BSESKF and DC estimators. For this thesis, the GTDS thrust force model was completely implemented only with the Cowell propagator and the DC estimator. Future work will complete the implementation to allow the thrust force model to be used with the DSST propagator and the ESKF and BSESKF estimators.

The standalone tool is given the initial and final orbits. It uses numerical integration of the equations of motion and the quasi-Newton gradient search described in sections 2.2.2.3 and 2.2.2.4 to solve for an optimal thrust plan and two-body trajectory. Perturbations such as J_2 affect thrust plans over time spans of several days. However, implementing perturbations was left as future work. Separate modules were written for the averaged and exact equations of motion. This allows crude guesses for the necessary Lagrange multipliers to be refined first by the averaged equation module and then solved precisely by the exact equation module. The ultimate product of the exact equation module is a file containing time vs. acceleration vector directions and magnitude values. This thrust acceleration file format is described in Appendix A.

The thrust acceleration file produced by the exact equation standalone optimization module is read by the GTDS orbit determination software within a newly implemented continuous thrust acceleration force model module. This module can be used in either orbit prediction or in orbit determination to evaluate the accuracy of the produced thrust plan with real data.

5.1 Standalone Trajectory Optimization Software

The software written to calculate optimal thrust plans from an initial orbit to a final orbit was written first in Matlab™ for ease of implementation and then in FORTRAN to improve the performance. The code described here is the FORTRAN code. Both sets of code are very similar. Essentially, the language is the only difference. Because of the difficulty in guessing initial values for the Lagrange multipliers for the exact equation trajectory optimization code, the averaged equation trajectory code was implemented because it is more robust (57). Crude guesses for the averaged Lagrange multipliers can be used with the averaged equation code and the refined solution for the Lagrange multipliers can be used as initial guesses in subsequent executions of the exact equation code. Sections 5.1.1 and 5.1.2 detail the subroutines written to implement the trajectory optimization algorithm described in sections 2.2.2.3 and 2.2.2.4.

5.1.1 Exact Equation Trajectory Optimization Code

The subroutines described in this section comprise the exact two-body plus thrust equation of motion trajectory optimization standalone software. The source code for these subroutines is listed in Appendix E. The program flow for the exact equation standalone trajectory optimization code is shown in Figure 5.1.

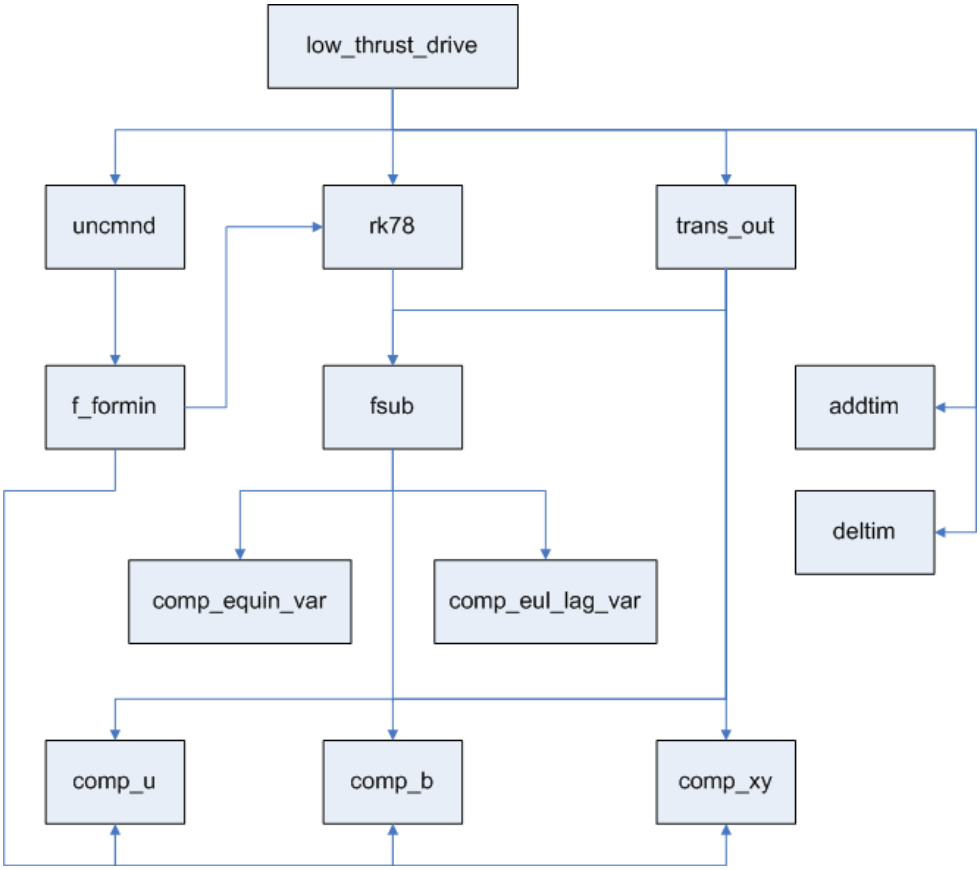

Figure 5.1 Exact Equation Trajectory Optimization Program Flow

LOW_THRUST_DRIVE is the driver subroutine for the software. It collects the initial and final Keplerian orbits, converts those to equinoctial orbit elements, calls the

UNCMND subroutine to execute the quasi-Newton iterative search to solve for the initial Lagrange multipliers. Once UNCMND is complete, the RK78 subroutine is used to integrate the variational equations of motion and the variational equations for the solved initial Lagrange multipliers from the initial to the final time. Finally, the trajectory is printed and the thrust plan file meant for GTDS input is written.

UNCMND is the subroutine provided by reference (58). This subroutine executes the quasi-Newton search described in Section 2.2.2.3. It calls the F_FORMIN subroutine to compute the equinoctial elements and Lagrange multipliers at the final time given the elements and multipliers at the initial time.

RK78 is the subroutine that executes the 7th order Runge-Kutta-Fehlberg integration. This subroutine was written at NASA JPL and is documented in NASA Technical Report TR R-287 (85). This subroutine is used to integrate both the equinoctial variational equations of motion and the variational equations for the Lagrange multipliers.

FSUB is the subroutine that is called by the FK78 subroutine to supply the equinoctial element and Lagrange multiplier derivatives with respect to time, i.e. rates. FSUB calls the COMP_XY, COMP_B, and COMP_U subroutines to calculate the auxiliary quantities, the 6x3 BL matrix and the normalized thrust acceleration vector. FSUB then executes the COMP_EQUIN_VAR and COMP_EUL_LAG_VAR

subroutines which compute the rates for the equinoctial variation equations and the rates for the Lagrange multipliers, respectively.

TRANS_OUT in the context of the exact equation code transforms the equinoctial elements into Keplerian elements and calls COMP_XY, COMP_M, COMP_U and FSUB to compute the Hamiltonian, thrust vector, and the yaw and pitch angles. The thrust vector is also transformed to inertial Cartesian coordinates to be compatible with GTDS for the thrust plan file. This transformation is described in detail in Appendix C. The quantities are returned to the calling subroutine in an array intended to be written as output.

F_FORMIN computes the equinoctial elements and Lagrange multipliers at the final time given the elements and multipliers at the initial time. F_FORMIN also computes the sum of the squares of the differences of the computed final orbital element conditions from the desired orbital element conditions. F_FORMIN uses the RK78 subroutine to perform the integration of the equinoctial orbital elements and the Lagrange multipliers.

COMP_EQUIN_VAR computes the derivatives of the equinoctial orbital elements with respect to time, i.e. element rates. This is done by multiplying the constant thrust acceleration magnitude by the product of the B^L matrix and the normalized thrust acceleration vector. This equation is shown in Section 2.2.2.2 (Equation 2.82), and the B^L matrix is shown in Appendix B.

COMP_EUL_LAG_VAR computes the derivatives of the Lagrange multipliers with respect to time, i.e. multiplier rates. This is done by multiplying the partial derivatives of the B^L matrix with respect to the equinoctial elements, the normalized thrust acceleration vector, the thrust acceleration magnitude and the current values of the orbital elements. The equations for this are shown in Section 2.2.2.2 (Equation 2.87) and the partials of the B^L matrix are shown in Appendix B.

COMP_B is the subroutine that computes the 6x3 BL matrix and its partial derivatives with respect to the equinoctial elements. The equations for this subroutine can be found in the Appendix of reference (5).

COMP_U computes the normalized thrust acceleration vector given the 6x3 B^L matrix and the vector of current Lagrange multipliers.

COMP_XY calculates auxiliary quantities based on the current equinoctial orbital elements.

DELTIM is a GTDS subroutine that was borrowed for this tool in order to assist in computing the calendar date given the initial date and a time duration. It is used along with the ADDTIM GTDS subroutine for this purpose.

ADDTIM is a GTDS subroutine borrowed for this tool in order to assist in computing the calendar date given the initial date and time duration. It is used along with the DELTIM GTDS subroutine for this purpose.

5.1.2 Averaged Equation Trajectory Optimization Code

The subroutines in this section comprise the averaged two-body equation of motion trajectory optimization standalone software. The source code for these subroutines is listed in Appendix F. The program flow for the averaged equation trajectory optimization code is shown in Figure 5.2.

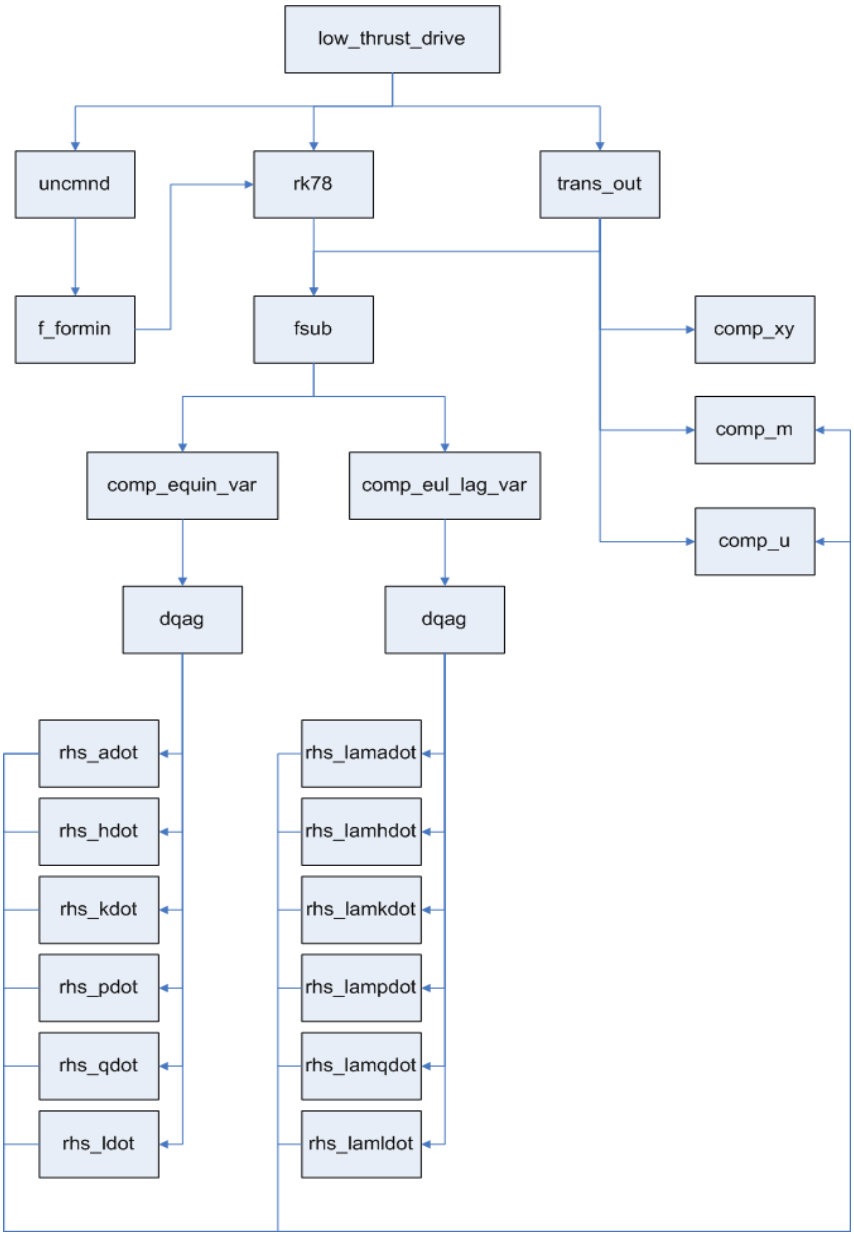

Figure 5.2 Averaged Equation Trajectory Optimization Program Flow

LOW_THRUST_DRIVE is the driver subroutine for the software. It collects the initial and final Keplerian orbits, converts those to equinoctial orbits, calls the UNCMND subroutine to execute the quasi-Newton search to solve for the initial Lagrange multipliers, and calls the RK78 subroutine to integrate the variational equations of motion

and the variational equations for the Lagrange multipliers from the initial to final time. Finally, the trajectory is printed.

UNCMND is the subroutine provided by the reference (58). This subroutine executes the quasi-Newton search described in Section 2.2.2.3. It calls the F_FORMIN subroutine to compute the equinoctial elements and Lagrange multipliers at the final time given the elements and multipliers at the initial time.

RK78 is the subroutine that executes the 7th order Runge-Kutta-Fehlberg integration. This subroutine was written at NASA JPL and is documented in NASA Technical Report TR R-287. This subroutine is used to integrate both the equinoctial variational equations of motion and the variational equations for the Lagrange multipliers.

FSUB is the subroutine that is called by the FK78 subroutine to supply the equinoctial element and Lagrange multiplier derivatives with respect to time, i.e. rates. FSUB executes the COMP_EQUIN_VAR and COMP_EUL_LAG_VAR subroutines which compute the rates for the equinoctial variation equations and the rates for the Lagrange multipliers, respectively.

TRANS_OUT in the context of the averaged equation code transforms the equinoctial elements into Keplerian elements and calls COMP_XY, COMP_M, COMP_U and FSUB to compute the Hamiltonian, thrust vector, and the yaw and pitch

angles. These quantities are returned to the calling subroutine in an array intended to be written as output.

F_FORMIN computes the equinoctial elements and Lagrange multipliers at the final time given the elements and multipliers at the initial time. F_FORMIN also computes the sum of the squares of the differences of the computed final orbital element conditions from the desired orbital element conditions. F_FORMIN uses the RK78 subroutine to perform the integration of the equinoctial orbital elements and the Lagrange multipliers.

COMP_EQUIN_VAR computes the derivatives of the equinoctial orbital elements with respect to time, i.e. element rates. Because the averaged equations of motion are used here, the DQAG subroutine is used to compute the element rates using a Gauss-Kronrod numerical quadrature.

COMP_EUL_LAG_VAR computes the derivatives of the Lagrange multipliers with respect to time, i.e. multiplier rates. The averaged equations for the multiplier rates are computed using the DQAG subroutine which performs numerical quadrature using the Gauss-Kronrod method.

DQAG uses a Gauss-Kronrod method to compute the definite integrals shown in section 2.2.2.4. DQAG is used in conjunction with the subroutines, RHS_ADOT, RHS_HDOT, RHS_KDOT, RHS_PDOT, RHS_QDOT, RHS_LDOT,

RHS_LAMADOT, RHS_LAMHDOT, RHS_LAMKDOT, RHS_LAMPDOT, RHS_LAMQDOT, and RHS_LAMLDOT. These subroutines compute the equinoctial element rates and the Lagrange multiplier rates given the current equinoctial elements and Lagrange multiplier values. DQAG is part of QUADPACK and was downloaded from http://www.netlib.org. QUADPACK is freely available software for numerical integration. DQAG was written by R. Piessens, K. U. Leuven, and E. De Doncker.

COMP_M is the subroutine that computes the 6x3 M matrix and its partial derivatives with respect to the equinoctial elements. The equations for this subroutine can be found in the Appendix of reference (3). According to Jean Kechichian, there is one small error in the partials in equation (A96). The term reading c_F - h should read c_F - k. This correction was also made in the COMP_M subroutine code.

COMP_U computes the normalized thrust acceleration vector given the 6x3 M matrix and the vector of current Lagrange multipliers.

COMP_XY calculates auxiliary quantities based on the current equinoctial orbital elements.

RHS_ADOT computes the semimajor axis rate of change using the current equinoctial elements and the COMP_M and COMP_U subroutines.

RHS_HDOT computes the equinoctial h element using the current equinoctial elements and the COMP_M and COMP_U subroutines.

RHS_KDOT computes the equinoctial k element using the current equinoctial elements and the COMP_M and COMP_U subroutines.

RHS_PDOT computes the equinoctial p element using the current equinoctial elements and the COMP_M and COMP_U subroutines.

RHS_QDOT computes the equinoctial q element using the current equinoctial elements and the COMP_M and COMP_U subroutines.

RHS_LDOT computes the equinoctial lambda element using the current equinoctial elements and the COMP_M and COMP_U subroutines.

RHS_LAMADOT computes the Lagrange multiplier associated with the semimajor axis using the COMP_M and COMP_U subroutines.

RHS_LAMHDOT computes the Lagrange multiplier associated with the equinoctial h element using the COMP_M and COMP_U subroutines.

RHS_LAMKDOT computes the Lagrange multiplier associated with the equinoctial k element using the COMP_M and COMP_U subroutines.

RHS_LAMPDOT computes the Lagrange multiplier associated with the equinoctial p element using the COMP_M and COMP_U subroutines.

RHS_LAMQDOT computes the Lagrange multiplier associated with the equinoctial q element using the COMP_M and COMP_U subroutines.

RHS_LAMLDOT computes the Lagrange multiplier associated with the equinoctial lambda (mean longitude) element using the COMP_M and COMP_U subroutines.

5.2 GTDS Continuous Thrust Implementation

5.2.1 New subroutines

The following new subroutines were written and included with GTDS to implement the thrust plan input.

THRSTTBL reads the thrust input from a file, interpolates the acceleration vectors and returns the thrust acceleration vector at the requested time.

The THRSTTBL.CMN common block contains an on/off switch for the thrust plan input as well as valid input start and end dates for the thrust plan input.

THRSTTBLCRD reads one thrust table input record from the FORTRAN file unit numbered 115. This subroutine is called by the THRSTTBL subroutine.

THR_RDNUMR reads the numeric fields on a thrust plan input record. This subroutine is called by the THRSTTBLCRD subroutine.

There were also some subroutines that were imported from the book, "Numerical Methods and Software," by Kahaner, Moler and Nash (58). These subroutines performed the interpolation necessary to compute acceleration vectors that are requested for times that fall between records provided by the thrust plan input file.

PCHEZ computes derivatives needed for the PCHEV subroutine. PCHEZ computes derivatives for spline or cubic Hermite interpolation.

PCHEV evaluates a function and first derivative of a piecewise cubic Hermite or spline function at an array of points. The function array and derivative array are provided as input and are assumed to be previously computed by the PCHEZ subroutine.

XERROR is a subroutine for handling and/or printing diagnostic messages generated by numeric subroutines in the Kahaner, Moler, and Nash text (58).

5.2.2 Modified subroutines

The GTDS source code needed modifications to read in thrust plans and apply the thrust acceleration vectors necessary for such plans. Because thrust plans could conceivably come from many sources, it was decided that a text file would be the mode of input. This would allow the optimal thrust plans generated by any source to be used as input in GTDS as a thrust force model. This text format is described in Appendix A which describes the GTDS input keywords introduced as part of this work. The THRSTTBL GTDS input keyword introduced to instruct GTDS to read from the thrust plan file input is described in Appendix A also.

The thrust plan input file defines the thrust acceleration vector at only the time points printed in the file. However, the GTDS orbit prediction execution requires thrust acceleration vectors at the numerical integration time points of its choosing. Therefore,

interpolators were used within the new continuous thrust module to linearly interpolate the thrust plan acceleration vectors at the time points requested by the orbit prediction software in GTDS.

There were several GTDS source code files that required modification to implement this file-based thrust plan input. The following subroutines were existing source code files in GTDS, but were modified for this task.

FILESBD is a block data initialization subroutine that identifies each file used by GTDS during its execution. The thrust input file was identified with unit 115.

SETDAF is the subroutine that opens all files used by GTDS. The thrust plan input file open statement was added to this source file.

SHUTDAF is the subroutine that closes all files used by GTDS upon termination of the program. The thrust plan input file was included in this closing sequence.

SETOG1 interprets all orbit generator optional keywords that come after the "DRAG" keyword in the keyword table. This subroutine is an extension of the SETORB subroutine. SETOG1 was modified to include the THRSTTBL keyword interpretation code.

SETORB reads and interprets orbit generator (EPHEM) optional keyword cards. The THRSTTBL keyword text was added to the keyword array.

SWITCHBD is a block data initialization subroutine that identifies many switches or options in GTDS. A thrust table input on/off switch variable was added.

ACCEL computes two-body and free-flight perturbative accelerations acting on a spacecraft at a given time and state. A call to the new THRSTTBL subroutine was added.

GQFUN computes the integrands of the integration for the average integration of the equation of motion. A call to the new THRSTTBL subroutine was added. GQFUN is used in the DSST propagator. Modification of the GQFUN subroutine partially implements the thrust force model for DSST. Further modifications to other GTDS DSST subroutines to complete the implementation of the thrust force model for DSST are left as future work.

5.2.3 GTDS modification summary

The program flow for GTDS differs depending on which ephemeris generator or propagator is chosen. All propagator subroutines are called from the ORBIT subroutine. The Cowell propagator typically uses a fixed-step size integrator that integrates equations of motion formulated with position and velocity as the variables. Figure 5.3 shows the program flow for the thrust acceleration force model starting from the ORBITC

subroutine. ORBITC is called by the ORBIT subroutine if the Cowell propagator is chosen.

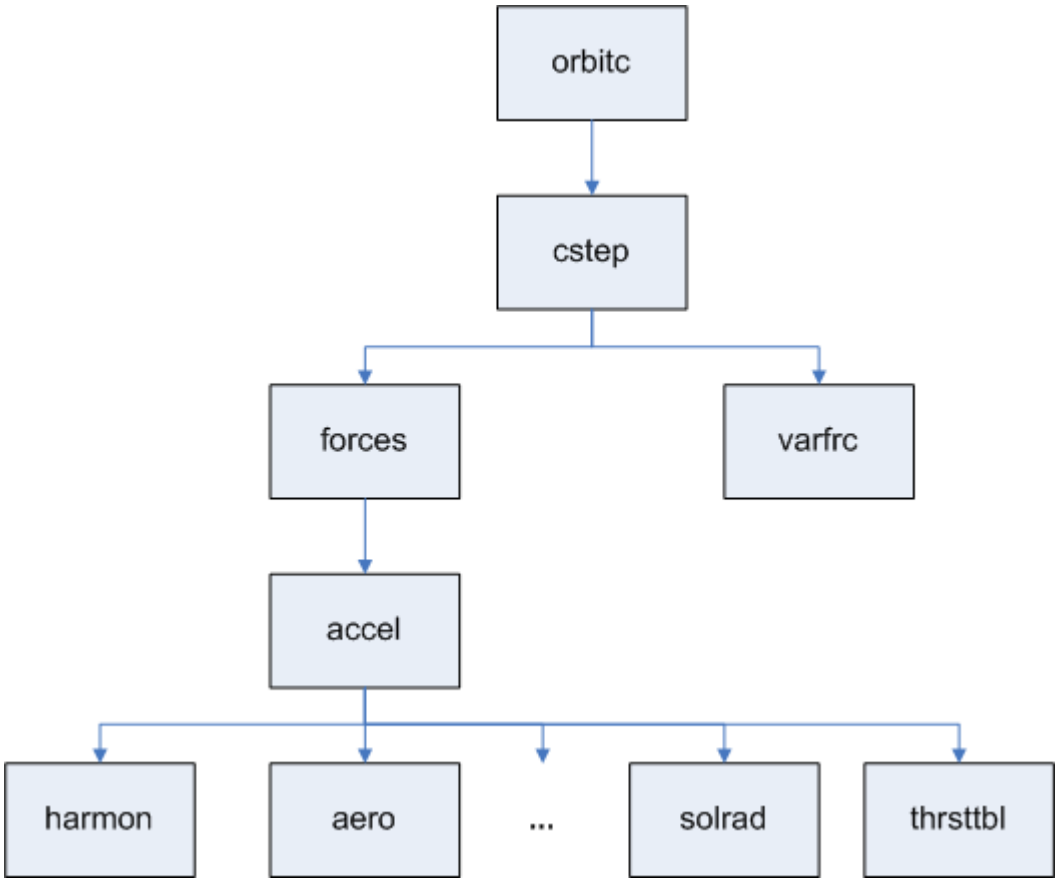

Figure 5.3 Cowell Program Flow for Thrust Acceleration File Input

The Draper Semianalytic Satellite Theory (DSST) is also called from the ORBIT subroutine. DSST integrates in mean equinoctial elements and adds short period motion using Fourier series. The driver is the ORBITV subroutine. Figure 5.4 shows the program flow for the thrust acceleration force model.

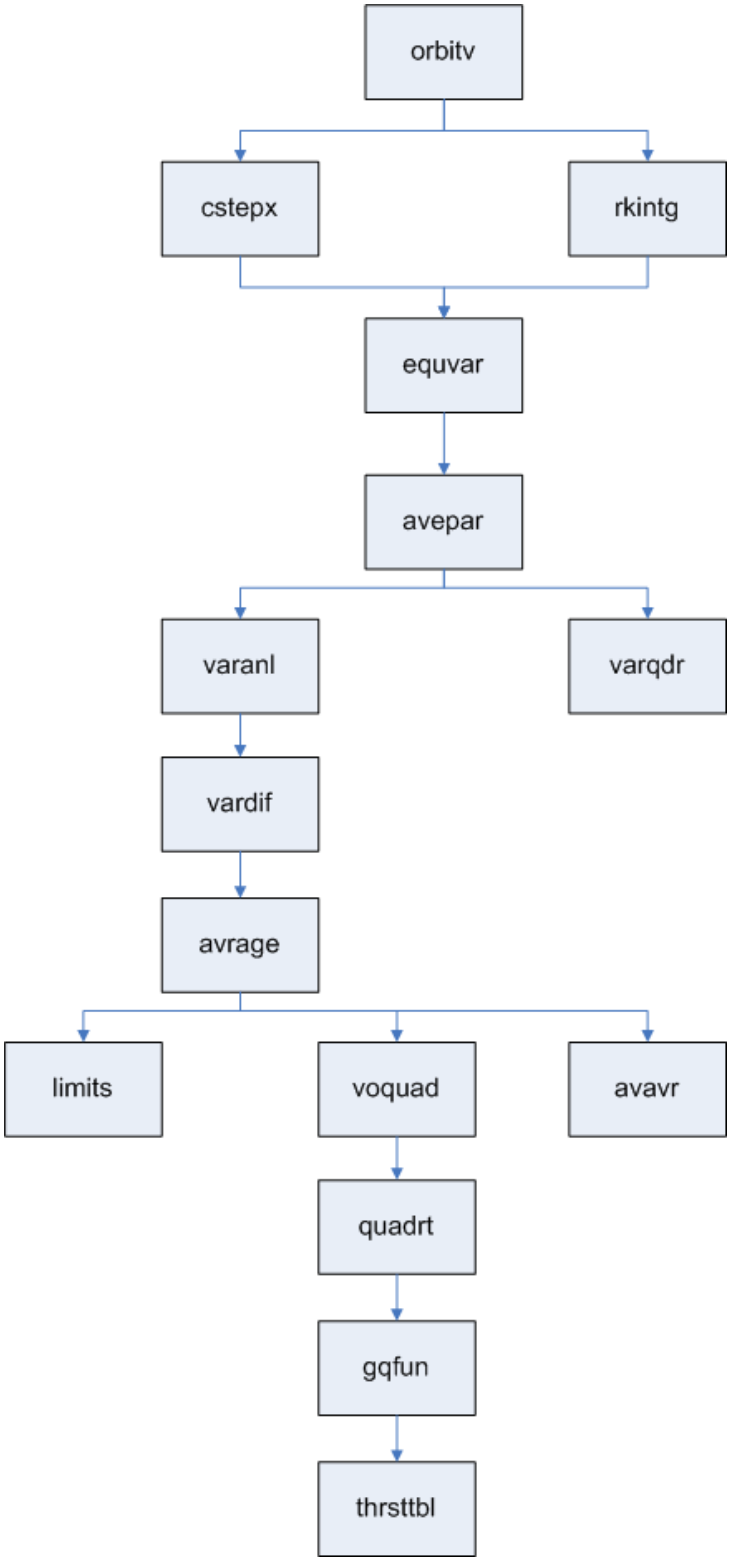

Figure 5.4 DSST Program Flow for Thrust Acceleration File Input

5.3 Verifying Test Case Results

The test case provided by Jean Kechichian in references (3) and (4) was run against the standalone thrust plan tool that was implemented for this thesis. In addition, the thrust plan generated by the standalone tool was used as input for the GTDS orbit prediction tool. The test case was run against the standalone and GTDS force model software to verify the correct implementation of the two-body equations of motion, the quasi-Newton optimal thrust plan search algorithm, and the new GTDS continuous thrust force module.

The test case run against the standalone module was an orbit transfer case between an initial LEO orbit and a final GEO orbit. The initial and final orbits are shown in Table 5.1. This shows the initial and final orbits as well as the final orbit achieved by the quasi-Newton search algorithm. The achieved orbit is very close to the desired final orbit indicating that the quasi-Newton search algorithm is able to precisely solve the two-point boundary value problem.

Table 5.1 Standalone Tool Initial and Final Orbit Achieved

Orbit	a, (km)	e	i, (deg)	Ω, (deg)	ω, (deg)	M, (deg)
Initial	7000	0	28.5	0	0	-130.333164
Final	42000	0.001	1	0	0	Free
Achieved	42000.0052	0.0009987	0.99982	-0.000193	0.055	46.192579

The final orbit did not include a mean anomaly. Rather, this orbital parameter was free for the quasi-Newton search to determine the optimal value. In addition, the total transfer

time was found to be 58089.9 seconds. The Hamiltonian was 1.004 upon completion. The solved-for Lagrange multipliers are shown in Table 5.2.

Table 5.2 Solved Initial Lagrange Multipliers for LEO to GEO Case

Lagrange Multiplier	Solution Value
$\left(\lambda_a^L\right)_0$ (s/km)	0.467522877173E+01
$\left(\lambda_h^L\right)_0$ (sec)	0.541341369629E+03
$\left(\lambda_k^L\right)_0$ (sec)	-0.920270214844E+04
$\left(\lambda_p^L\right)_0$ (sec)	0.177801189423E+02
$\left(\lambda_q^L\right)_0$ (sec)	-0.225845585937E+05
$\left(\lambda_L^L\right)_0$ (rad)	-0.647890140182E-08

The results were reasonably close to the results achieved by Jean Kechichian in reference (5). Kechichian's results are shown in Tables 5.3 and 5.4.

Table 5.3 Initial and Final Orbit Achieved by Jean Kechichian

Orbit	*a*, (km)	*e*	*i*, (deg)	Ω, (deg)	ω, (deg)	*M,* (deg)
Initial	7000	0	28.5	0	0	-130.333164
Final	42000	0.001	1	0	0	Free
Achieved	41999.9929	0.0009983	0.999797	0.000326	359.995148	46.169264

Kechichian solved for a total transfer time of 58089.9 seconds. The Lagrange multipliers solved by Jean Kechichian are shown in Table 5.4.

Table 5.4 Kechichian's Solved Initial Lagrange Multipliers for LEO to GEO Case

Lagrange Multiplier	Solution Value
$\left(\lambda_a^L\right)_0$ (s/km)	0.4675229762E+01
$\left(\lambda_h^L\right)_0$ (sec)	0.5413413947E+03
$\left(\lambda_k^L\right)_0$ (sec)	-0.9202702084E+04
$\left(\lambda_p^L\right)_0$ (sec)	0.1778011878E+02
$\left(\lambda_q^L\right)_0$ (sec)	-0.2258455855E+05
$\left(\lambda_L^L\right)_0$ (rad)	Not shown in paper

Because the results from the standalone code so closely match those of Jean Kechichian, it was surmised that the integrated equations of motion and the quasi-Newton algorithm were implemented correctly in the standalone software.

Another way to check the results is to compare the orbital element and thrust acceleration vector histories during the transfer. Selected element histories and the thrust pitch and yaw angles are shown in the following figures. Dr. Kechichian's results were taken from reference (4). Figure 5.5 shows the semimajor axis and eccentricity element time histories during the transfer from the initial orbit to the final orbit. There are small differences in the plots. The eccentricity oscillation during the initial part of the transfer has a slightly different character in the two plots. This is explained by the fact that the plots from Jean Kechichian's paper were generated with a slight error in a partial

derivative expression used in the equations of motion. This fact was learned both from Kechichian's paper, reference (7), and from personal communication with Dr. Kechichian (86). Figure 5.6 shows the thrust pitch and yaw angles for the optimal LEO to GEO thrust plan. The large pitch and yaw angle changes near the end of the transfer reflect the large eccentricity and inclination changes that are undergone near the end of the transfer.

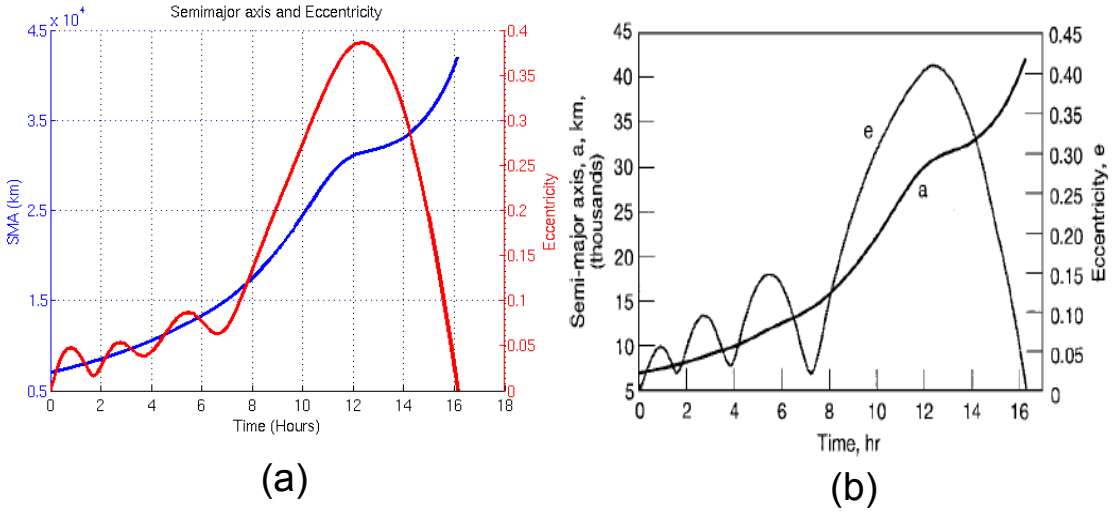

Figure 5.5 Semimajor Axis and Eccentricity Transfer Time History (a), Kechichian's result (b)

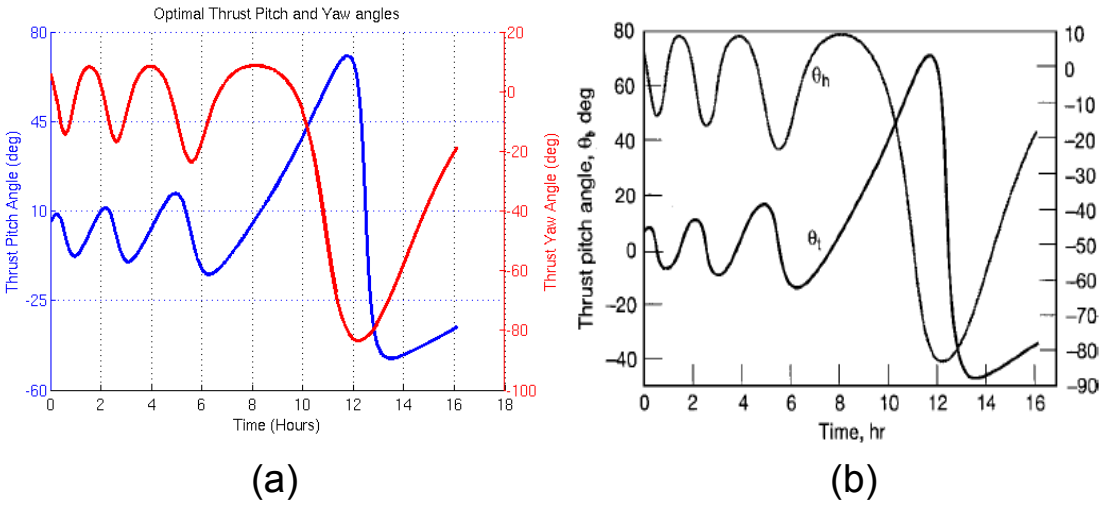

Figure 5.6 Thrust Pitch and Yaw Transfer Time History (a), Kechichian's result (b)

Figures 5.7 (a) and (b) have the inclination history in common. The inclination follows a similar trend in both figures, but the inclination from Kechichian's paper is somewhat different because the abscissa is the semimajor axis and not time. Nevertheless, both plots show the inclination is correct at the initial and final boundary conditions, 28.5 and 1.0 degrees, respectively.

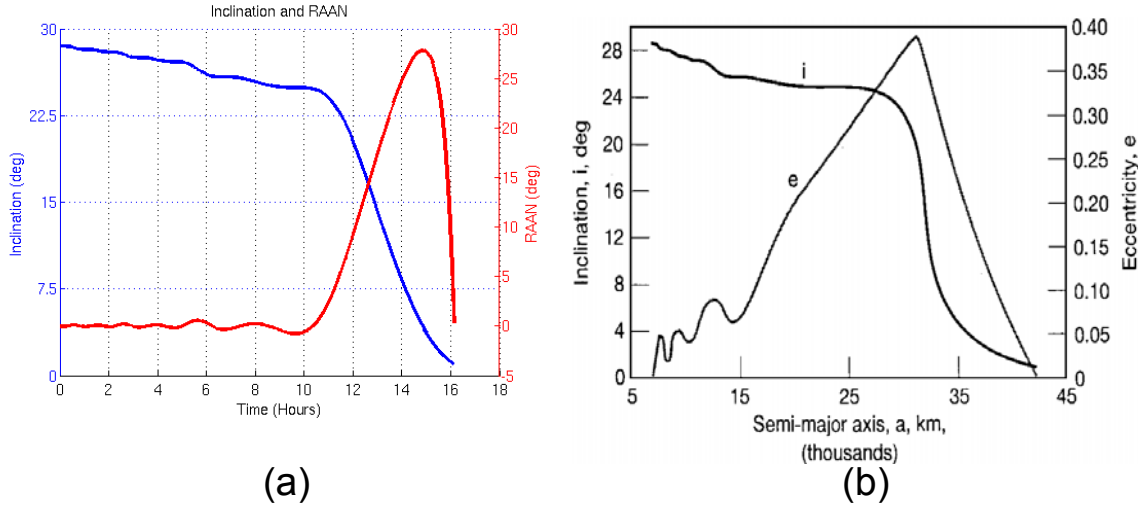

Figure 5.7 Inclination and RAAN Time History (a),
Kechichian's Inclination, Semimajor Axis and Eccentricity History (b)

The implementation of the averaged equation of motion standalone trajectory optimization code was also tested against Kechichian's results. The averaged equation of motion tool is useful because it is more robust than the exact equation of motion code in its ability to use initial Lagrange multiplier values with large errors (57). The refined initial Lagrange multiplier values solved by the averaged tool can then be used to initialize the exact code.

The initial and final orbits for the averaged case were the same as for the exact case. These are shown in Table 5.5 along with the final orbit achieved using the quasi-Newton search.

Table 5.5 Averaged Standalone Tool Initial and Final Orbit Achieved

Orbit	*a*, (km)	*e*	*i*, (deg)	Ω, (deg)	ω, (deg)	*M,* (deg)
Initial	7000	0	28.5	0	0	-130.333164
Final	42000	0.001	1	0	0	Free
Achieved	42000.00	0.001000	0.999999	0.0000002	0.005	-130.328514

These results are close to the results achieved by Kechichian in his paper (4). The quasi-Newton search algorithm is able to closely match the final conditions by searching for the initial averaged Lagrange multipliers. The total transfer time was found to be 56732.57 seconds which compares closely with Jean Kechichian's result of 56734.56 seconds. The Hamiltonian was constant over the transfer span with a value of 1.000005 which shows that the necessary condition of optimality for the minimum time and fuel transfer was essentially met. When the Hamiltonian is not an explicit function of time, it is constant over the trajectory transfer (49). Jean Kechichian's published value for the Hamiltonian is 1.000000007. The averaged initial Lagrange multipliers solved by the averaged standalone tool are shown in Table 5.6.

Table 5.6 Solved Initial Averaged Lagrange Multipliers for LEO to GEO Case

Lagrange Multiplier	Solution Value
$\left(\tilde{\lambda}_a\right)_0$ (s/km)	0.507914542146E+01
$\left(\tilde{\lambda}_h\right)_0$ (sec)	-0.197268742508E-02
$\left(\tilde{\lambda}_k\right)_0$ (sec)	0.425340801731E+02
$\left(\tilde{\lambda}_p\right)_0$ (sec)	-0.463304720272E-04
$\left(\tilde{\lambda}_q\right)_0$ (sec)	-0.792521641184E+05
$\left(\tilde{\lambda}_\lambda\right)_0$ (rad)	0.949939541356E-03

Kechichian's results include slight errors mentioned in reference (7). Therefore, Kechichian's solution for the averaged initial Lagrange multipliers does not exactly match the solution from the averaged standalone trajectory optimization tool. Kechichian's results are shown in Table 5.7 for comparison.

Table 5.7 Kechichian's Initial Avg. Lagrange Multipliers for LEO to GEO Case

Lagrange Multiplier	Solution Value
$\left(\tilde{\lambda}_a\right)_0$ (s/km)	0.5159779497E+01
$\left(\tilde{\lambda}_h\right)_0$ (sec)	-0.1448979417E-06
$\left(\tilde{\lambda}_k\right)_0$ (sec)	0.4342792320E+02
$\left(\tilde{\lambda}_p\right)_0$ (sec)	-0.1398718238E-07
$\left(\tilde{\lambda}_q\right)_0$ (sec)	-0.8360354382E+05
$\left(\tilde{\lambda}_\lambda\right)_0$ (rad)	0.0

The averaged implementation is more robust than the exact code because the averaged equations of motion do not include short-period oscillatory motion which complicates the job of the quasi-Newton search algorithm. The quasi-Newton search relies on finite differencing for computation of an approximate Jacobian matrix. The Jacobian matrix is used to find the best search direction for each iteration of the search algorithm. This finite differencing approximation is less robust for the exact equations of motion than it is for the smoother, averaged equations of motion. The smoother behavior of the averaged equations of motion is illustrated in Figure 5.8 which shows the Lagrange multiplier associated with the semimajor axis, λ_a, for both the averaged and exact equations of motion.

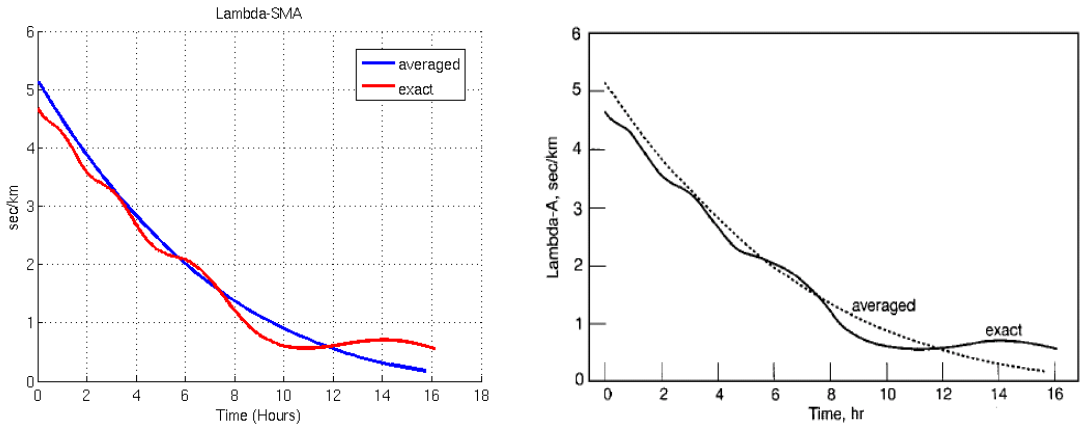

Figure 5.8 Lagrange Multiplier for SMA in Averaged and Exact Cases (a), Kechichian's results (b)

Figure 5.8 also shows the close level of agreement in the behavior of the averaged elements and Lagrange multipliers over time. This agreement provides good qualitative evidence that the equations of motion have been implemented correctly. The slight errors

in Kechichian's original work are not large enough to significantly affect the results shown in Figure 5.8. Figure 5.8.a was generated using the corrected equations of motion while Figure 5.8.b was generated by Jean Kechichian with the slight error.

Another aspect of the testing done for this LEO to GEO case involved using the thrust plan file generated by the exact standalone code to perform an orbit prediction in the Cowell orbit propagator in GTDS. This test exercised the new continuous thrust acceleration force model module in GTDS. Figures 5.9 – 5.11 show the results of this test. The orbit prediction done by GTDS used J_2 gravity terms so the final orbit achieved by the thrust plan does not exactly match the final orbit used in generating the thrust plan. However, the results are close. The thrust plan software could be modified to increase the optimal thrust plan accuracy by implementing J_2, J_3, J_4, third-body gravity, and other force models in the optimal thrust plan standalone software. For orbital transfers that take many days to execute, these perturbations significantly affect the optimal thrust plan required. Therefore, implementing these perturbations is important. However, for this thesis, such implementations are left as future work.

Figures 5.9-5.11 show that GTDS is able to correctly interpret the thrust plans and is able to reproduce the LEO to GEO orbit transfer with the associated orbital plane change. Also, once the thrust plan terminates, GTDS is able to continue on with just the natural force modeling.

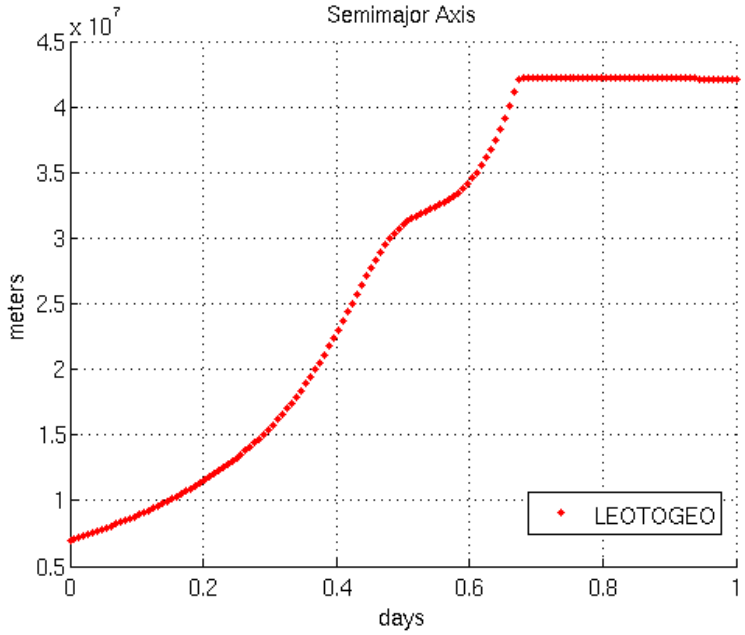

Figure 5.9 GTDS Cowell Ephemeris Generation of Semimajor Axis History

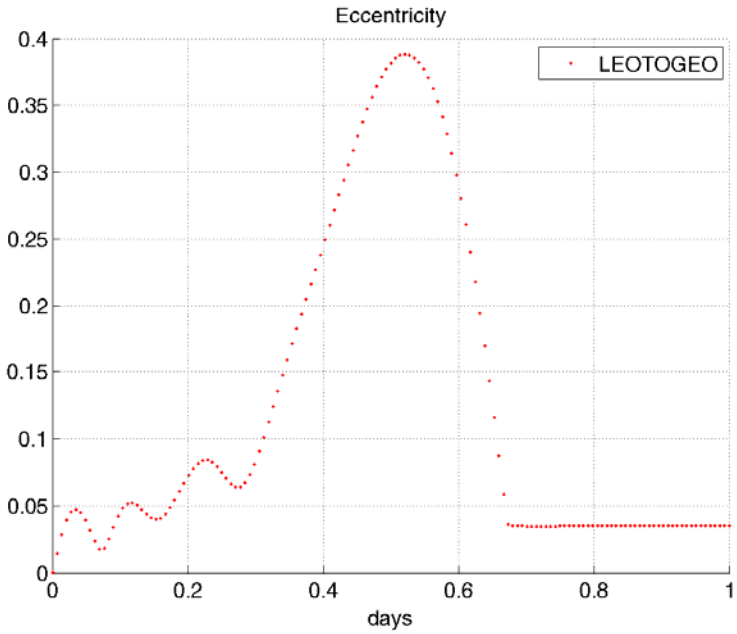

Figure 5.10 GTDS Cowell Ephemeris Generation of Eccentricity History

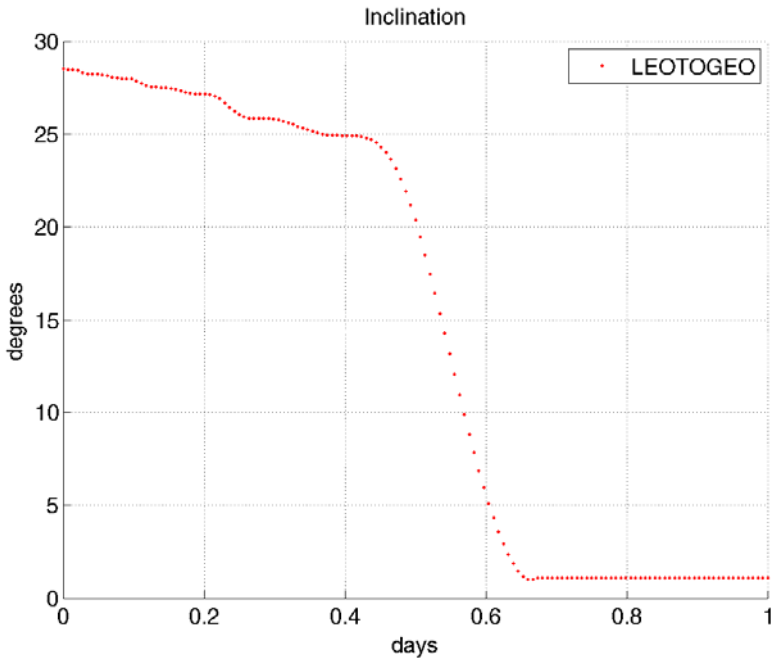

Figure 5.11 GTDS Cowell Ephemeris Generation of Inclination History

5.4 Real Data Test Case Results

5.4.1 ARTEMIS Satellite Background

The ARTEMIS telecommunication satellite was launched from Kourou, French Guiana on July 12, 2001 onboard an Ariane 510 Rocket. ARTEMIS was intended for a geostationary orbit. ARTEMIS is an ESA spacecraft that includes ion propulsion systems intended for North/South, i.e. orbital inclination, station keeping control.

During the course of the launch, the Ariane upper stage malfunctioned and injected ARTEMIS well short of its intended orbit. Although the satellite was launched with surplus bi-propellant, this system would not allow useful operational capability after boosting because of the large amount of fuel needed. The ARTEMIS team consisted of personnel from ESA, Alenia Spazio, and EADS. Working from the TELESPAZIO center in Fucino, Italy, a plan was developed to boost the ARTEMIS satellite to a useful geosynchronous orbit (GEO) while still allowing for a long useful satellite life (87). This plan first called for the bi-propellant thrusters to be used to raise the satellite's orbit outside the Van-Allen radiation belts. Then, the onboard ion propulsion systems would be used to perform a gradual orbit raising to GEO. Figure 5.12 depicts the overall plan developed by the ARTEMIS team.

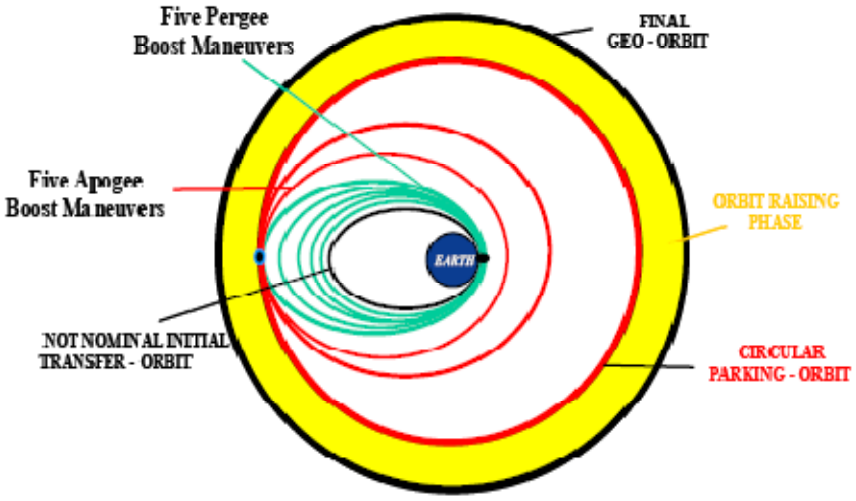

Figure 5.12 Maneuver Strategy for ARTEMIS Salvage Mission (47), (88)

The ion propulsion technology (IPP) on ARTEMIS consists of two Radio frequency Ion Thruster Assemblies (RITAs) and two Kaufmann ion type Electron Bombardment Ion Thruster Assemblies (EITAs) (47). The RITA thrusters use radio frequency radiation to ionize the Xenon atoms while the EITA thrusters are very similar to the gridded ion engines described in section 2.2.1. Both thrusters use grid technology to accelerate Xenon ions. The level of thrust when using both EITAs is 27 mN and 21 mN when using both RITAs. Using all four thrusters, or for that matter, any combination of thrusters on both the top and bottom sides of the spacecraft at once to perform orbit raising was deemed inefficient. The Isp would be reduced from greater than 3000 seconds to about 2300 seconds. This is because the thrusters are permanently canted with respect to the spacecraft North/South Z-axis. In addition, thermal constraints dictated that only one thruster per platform, i.e. top or bottom thruster array, could be used for the orbit raising operation. According to ARTEMIS recovery mission information obtained

by email from Leonardo Mazzini (88), The ion thruster orbit raising was accomplished in four phases. The following list summarizes these phases.

1. August 2001 – January 2002: Inclination control strategy using RITA1 or RITA2
2. February 2002 – April 2002: Nominal strategy using RITA1 and RITA2 (or EITA2)
3. April 2002 – July 2002: Thrust steering strategy using RITA1 and RITA2
4. August 2002 – January 2003: Back-up strategy using RITA2 only

Figure 5.13a shows the location of the IPP ion thrusters on the nominally zenith facing side of the ARTEMIS spacecraft. Figure 5.13b shows the nadir facing side of the ARTEMIS spacecraft. Figure 5.14 shows the spacecraft orientation during a single platform thruster firing. The depiction of the spacecraft orientation in Figure 5.14 is not the final, intended operational attitude, but was used during the ion thrust orbit raising.

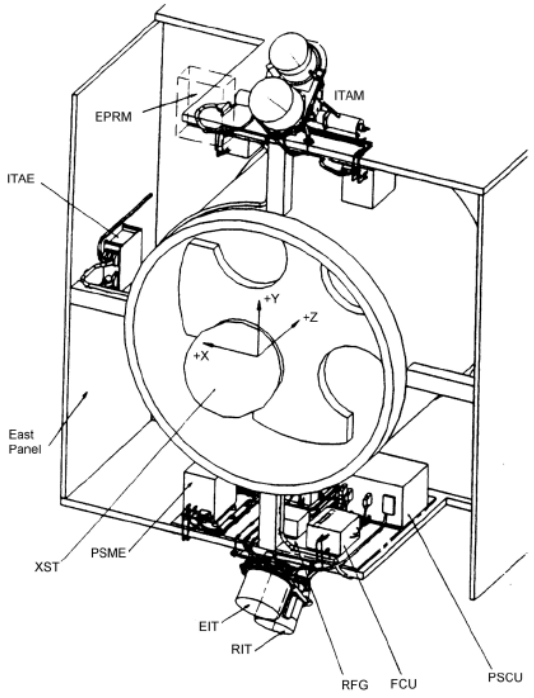

Figure 5.13a Ion Thruster Locations on the ARTEMIS Satellite (47)

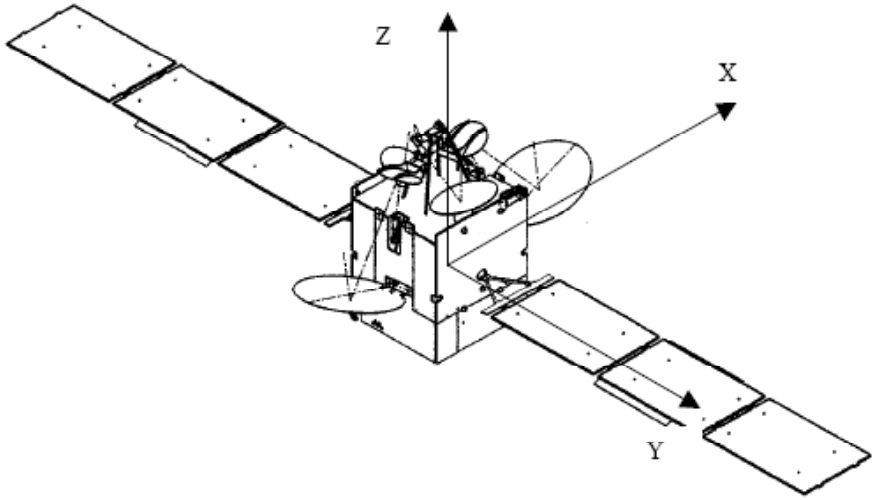

Figure 5.13b Spacecraft Axis in Orbit Reference System (88)

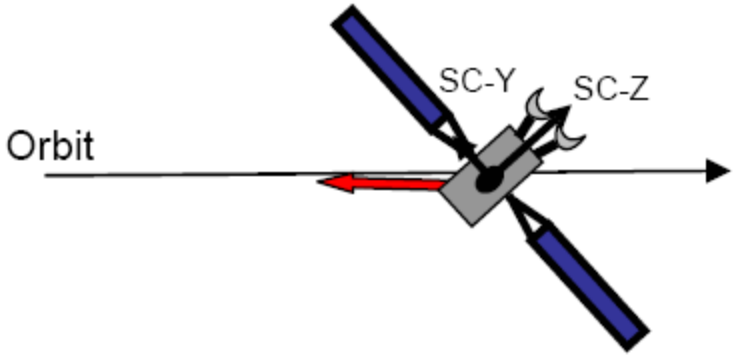

Figure 5.14 Spacecraft Orientation and Thrust Vector for Single Thruster Firing (87)

5.4.2 ARTEMIS Satellite Data and Test Case Methodology

The orbit raising operations for ARTEMIS started on April 4, 2002 and continued until final GEO orbit insertion on January 31st, 2003. The orbital elements at the end of orbit raising operations are shown in Table 5.8. The reference frame for these elements is not known. They were obtained from reference (89).

Table 5.8 Final Orbital Elements after all Ion Orbit Raising and Subsequent Chemical Burns

Epoch	January 31, 2003 20:00:00
Semimajor axis (km)	42169.731266
Eccentricity	0.000076
Inclination (deg)	1.565893
RAAN (deg)	112.524590
Arg. of Perigee (deg)	160.782369
True Anomaly (deg)	178.530958
East Longitude (deg)	21.216887

An Air Force Space Command Form1 document has been filed for the use of Two-Line orbital elements (TLEs) and U.S. Air Force Space Surveillance Network (AFSSN) observations in this research. A TLE is a text format that represents an orbital element set with an assumed orbital dynamic method, i.e. SGP4, and from which Earth centered inertial (ECI) vectors in the True Equator Mean Equinox of Epoch (TEME) reference frame can be obtained. Reference (90) contains detailed information about TLEs.

The TLEs were used to derive initial and final ARTEMIS satellite orbits at various times during the satellite's orbit raising phase. The time duration between the initial and final orbits was chosen so that the orbital states between the initial and final orbits appeared continuous. It is more straightforward to test a single continuous transfer rather than several discontinuous ones. Transfer spans on the order of ten days were chosen to allow for sufficient observations during the span because these spans would later serve as the basis for a least-squares orbit fit, i.e. GTDS differential correction (32). The estimator used for GTDS differential correction is described in section 2.3.1.1.8.

The initial and final orbits and a guess of the thrust acceleration magnitude were first passed to the averaged equation optimal thrust planning software. The initial guesses for the Lagrange multipliers were set to unity. Because the averaged equation code is robust, it was able to solve for refined values of the initial Lagrange multipliers and for an optimal transfer time. The refined Lagrange multiplier values, the guess for the thrust acceleration magnitude, the optimal transfer time, and the initial and final orbits were then passed to the exact equation optimal thrust planning software. This software generated an exact optimal thrust plan using two-body plus thrust orbital dynamics. The resulting thrust plan was used in GTDS differential corrections (DCs) with a fitspan that included all TLEs or AFSSN observations with observation times between the initial and final orbits. This GTDS DC was then evaluated for fit quality. This evaluation consisted of checking observation residuals for expected Gaussian means and variances, counting the number of observations edited due to the 3σ criteria, checking the chi-squared (91)

statistic, and examining the covariance of the final orbit estimate. The process flow for solving for the optimal continuous transfer is shown in Figure 5.15.

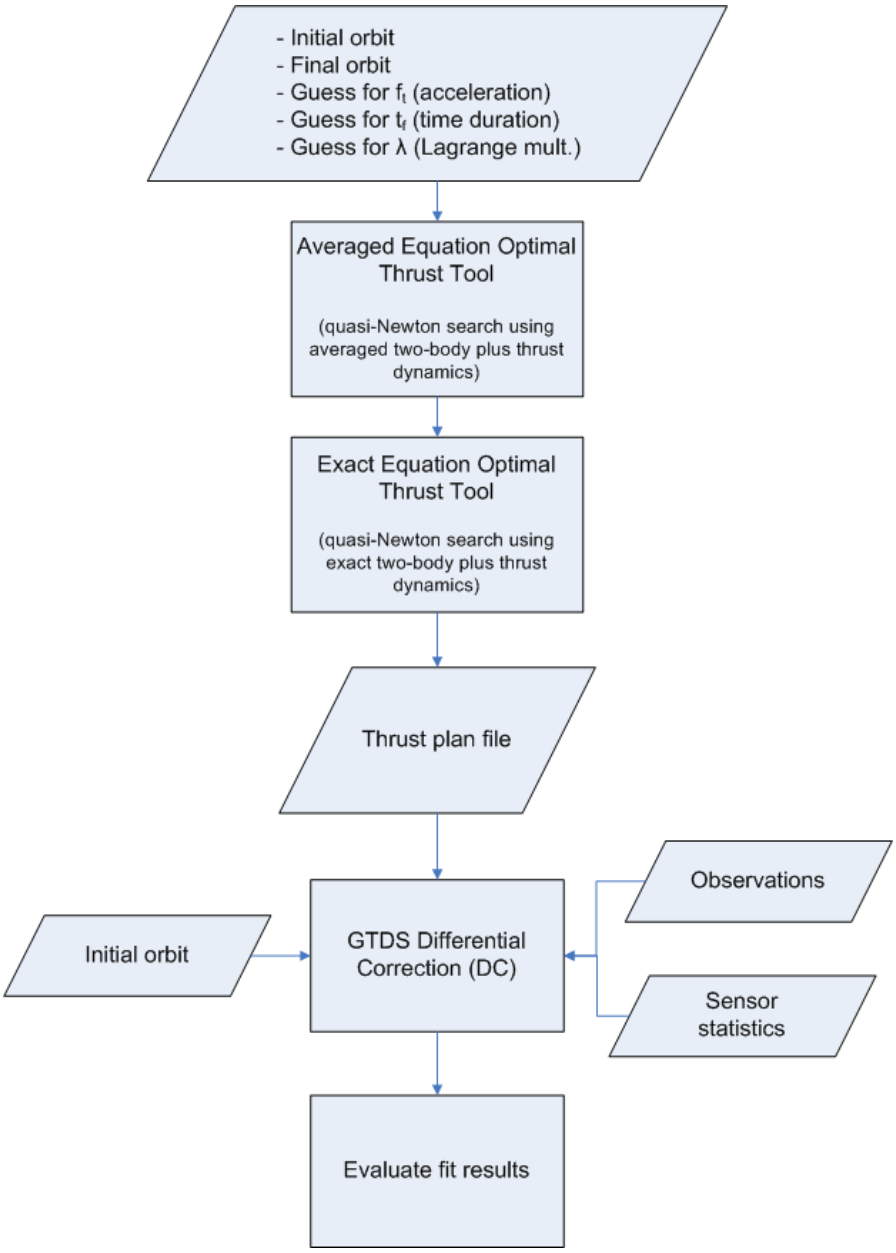

Figure 5.15 Thrust Plan Generation and Force Model Process Flow

Because the force exerted by the thrusters was not exactly known, the thrust acceleration was adjusted while the exact equation thrust planning software was run in

several iterations to affect the transfer time. These iterations continued until the transfer time solved by the thrust planning software was closely matched to the transfer time known from the epoch times of the endpoint TLEs.

Once optimal thrust plans were generated, they were used in the new GTDS file input thrust force model in orbit determination runs to evaluate whether the additional modeling yields any improvement in orbit determination accuracy. For these test cases, the GTDS Cowell Differential Correction (DC) subprogram was executed with the new thrust force model, and the resulting GTDS output file was parsed to collect observation residual information and overall orbit fit statistics. The natural forces modeled by GTDS during the DC included 12^{th} degree and 12^{th} order geopotential spherical harmonics based on coefficients from the JGM-2 geopotential model. Lunar and solar point mass gravity was modeled as was solar radiation pressure. The reflectivity coefficient for solar radiation pressure was not a solve-for parameter. This was to avoid any aliasing that might occur between the reflectivity coefficient and any thrust acceleration applied. Earth polar motion was modeled. Drag was not modeled because ARTEMIS was at a very high altitude essentially unaffected by atmospheric drag during the thrust transfer. Because GTDS includes an array of perturbation models while the thrust plan software does not currently include any perturbations, it was expected that the generated optimal thrust plans would not exactly reproduce the desired trajectory when modeled in GTDS. Including significant perturbations such as J_2 and lunar and solar gravity is a desired future enhancement to the thrust modeling software.

The observations used in some of the DC runs consisted of TLEs that were converted to their osculating counterparts using the SGP4 orbit propagator within GTDS. The SGP4 propagator was implemented in GTDS in 1988 by Darrell Herriges for work toward his Master's Thesis at MIT (18). The resulting osculating orbital elements were used as Cartesian position/velocity vector observations in the GTDS DC subprogram. This capability is sometimes referred to as *Precise Conversion of Elements*.

Air Force Space Surveillance Network (AFSSN) observations taken on ARTEMIS during its orbit raising were also used with the GTDS DC program to evaluate the usefulness of the thrust plans as acceleration models. These observations consisted of ground-based radar and optical observations. The radar observations consisted of topocentric range, azimuth, elevation and doppler measurements of the ARTEMIS satellite, and the optical observations consisted of right ascension and declination measurements of the satellite against the star background. The observations were taken by several different radar and optical sensors which were all part of the AFSSN in 2002 and 2003.

The semimajor axis, eccentricity, and inclination from the ARTEMIS TLEs are plotted in Figures 5.16 – 5.18. The TLEs are double averaged elements. Therefore the plots show only the secular motion of ARTEMIS from August, 2001 until May, 2003. These plots show how the continuous ion thrust affected the satellite orbit during its ion thrusting, orbit raising phase. The thrust strategies used to operate ARTEMIS during this phase included the following (88):

1. From August 2001 to January 2002: Inclination control strategy using RITA1 or RITA2
2. From February 2002 to April 002: Nominal strategy using RITA1 and RITA2 (or EITA2)
3. From April 2002 to July 2002: Thrust steering strategy using RITA1 and RITA2
4. From August 2002 to January 2003: Back-up strategy using RITA2

The thrust strategies are marked and labeled in Figures 5.16 – 5.18. The test cases used for this thesis are also marked and labeled as Case 1, Case 2, and Case 3 in Figures 5.16 – 5.18.

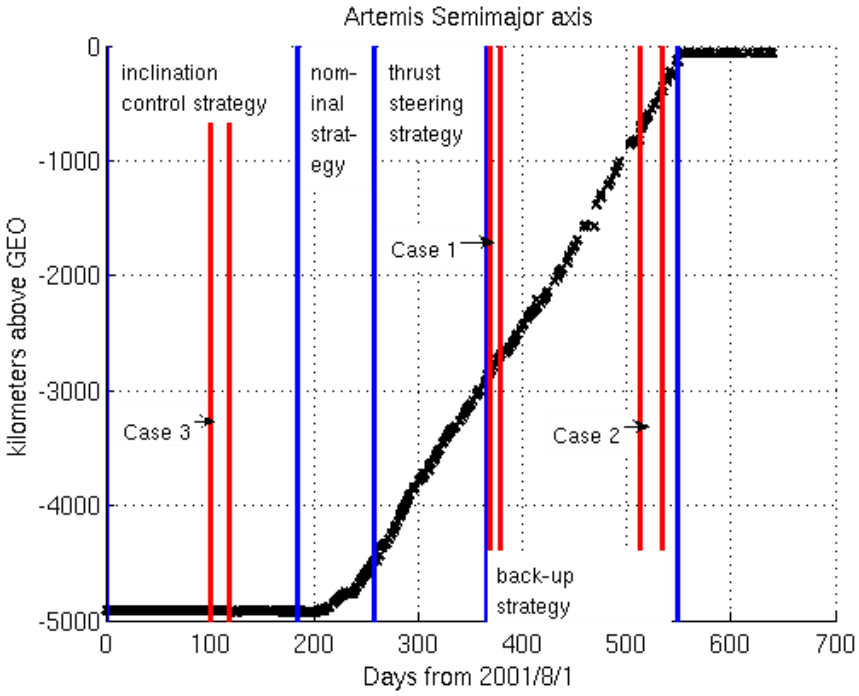

Figure 5.16 ARTEMIS Semimajor axis during Ion Thrusting

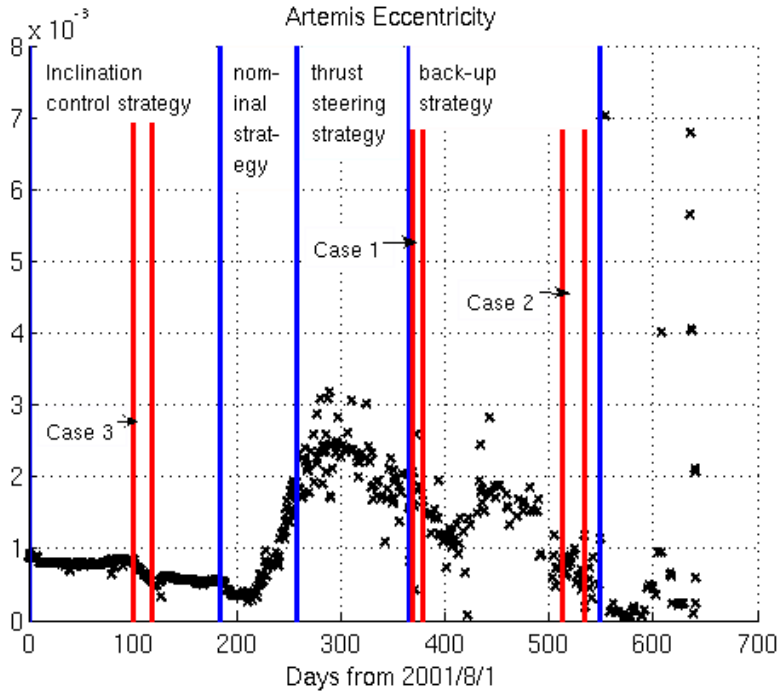

Figure 5.17 ARTEMIS Eccentricity during Ion Thrusting

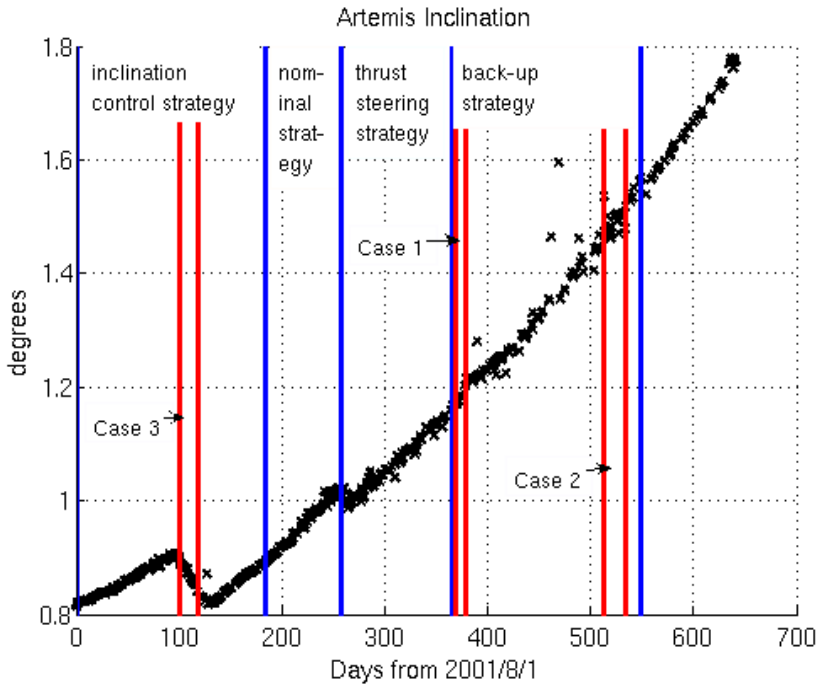

Figure 5.18 ARTEMIS Inclination during Ion Thrusting

Figure 5.16 shows the semimajor axis of ARTEMIS during the orbit raising maneuvers. The change to the semimajor axis is clearly linear, but some discontinuities during the transfer are apparent. This confirms text in reference (87) that mentions some drifting periods during the orbit raising. Some of these drifting periods lasted a few days. During some of the control strategies, the eccentricity history shown in Figure 5.17 is much noisier than the semimajor axis history. This is not unexpected because the TLEs were generated with orbit prediction models that didn't include continuous thrust force modeling. The inclination trend in Figure 5.18 shows that aside from maneuvers around day 100 and day 250, ARTEMIS exhibits natural evolution of the orbit plane due to lunar and solar gravitational perturbations. In references (87) and (88), the authors note that the inclination was actively controlled during the first three strategies used in the ion thrust orbit raising, i.e. the *inclination control strategy,* the *nominal strategy* and the *thrust steering strategy.* During the *back-up strategy* the inclination was not actively controlled. These comments in references (87) and (88) are reinforced by the inclination changes seen around days 100 and 250 in Figure 5.18. However, these appear to be the only times during which the inclination was reduced.

The test cases assembled for this thesis take place during the *inclination control strategy* and the *back-up strategy.* Test cases 1 and 2 as marked in Figures 5.16 -5.18 fall within the *back-up strategy* while test case 3 takes place in the *inclination control strategy.* With these three test cases, sampling of control strategies both with and without inclination control was accomplished. Sections 5.4.3 and 5.4.4 describe the test cases and results in detail.

5.4.3 ARTEMIS Orbit Determination Test Cases 1 and 2

The first ARTEMIS test case included AFSSN TLEs and observations recorded between the dates August 4, 2002 22:03:23 and August 14, 2002 19:12:00 UTC. These dates correspond with the epochs of the TLEs that bound the span. The second ARTEMIS test case included AFSSN TLEs and observations recorded between the dates, December 27th, 2002 12:21:23.7096 and January 16th, 2003 21:36:00 UTC. As in test case 1, the TLE epochs also bounded the span. Test case 1 is about 10 days long and test case 2 is about 20 days. Both test cases take place during the 4th phase of the ARTEMIS orbit raising. During this *back-up strategy*, only the RITA2 thruster was used (88), (88). The August and December-January 2002 test cases were chosen because there were many TLEs and AFSSN observations available during this 4th phase. It was also one of the longest phases of the orbit raising and so there were many multi-day time spans that could be used for developing test cases. All available TLEs during these time spans were converted to osculating Keplerian element sets and position/velocity vectors in the Mean Equator, Mean Equinox (MEME) of 1950 reference frame. The Keplerian elements were used to serve as inputs to the optimal thrust planning software while the position/velocity vectors were later used as observations in GTDS differential correction runs. The first and last resulting Keplerian element sets were used as the initial and final orbits in the optimal thrust planning software. These initial and final orbits are shown in Table 5.9 and Table 5.10.

Table 5.9 Initial and Final Orbits for ARTEMIS Case 1

Orbit	*a*, (km)	*e*	*i*, (deg)	Ω, (deg)	ω, (deg)	*M,* (deg)
Initial	39382.9722	0.00200685	1.435685	115.95324	297.51728	211.6003815
Final	39537.7077	0.00162154	1.435685	115.95324	297.51728	Free
Achieved	39537.7070	0.00162154	1.435683	115.95326	297.51717	191.3186515

Table 5.10 Initial and Final Orbits for ARTEMIS Case 2

Orbit	*a*, (km)	*e*	*i*, (deg)	Ω, (deg)	ω, (deg)	*M,* (deg)
Initial	41532.10828	0.902982×10^{-3}	1.7341511	109.06473	39.263634	354.061463
Final	41840.20862	0.633538×10^{-3}	1.7341511	109.06473	39.263634	Free
Achieved	41840.20700	0.633538×10^{-3}	1.7341511	109.06475	39.263634	279.485451

In test cases 1 and 2, the Keplerian elements for the final orbit were modified from the converted TLE at the final time because it was assumed that the inclination, the right ascension of the ascending node, and the argument of perigee were only experiencing drift according to natural perturbations during the *back-up control strategy* interval in the orbit raising. These orbital parameters were not intended to be changed through application of the ion thrusters. According to reference (87), the inclination was only affected by natural drift during the time span for these test cases. This doesn't mean, of course, that these elements didn't change as a result of the thrusting. However, to avoid calculating optimal thrust plans that duplicated natural perturbations affecting the inclination, RAAN and ARP, only the semimajor axis and eccentricity parameters were allowed to change from the initial to final orbits. This is reflected in Tables 5.9 and

5.10. As shown in Figures 5.16 and 5.17, the semimajor axis and eccentricity were significantly changed by the thrust application during this time span in August, 2002.

Tables 5.9 and 5.10 also show the final orbits achieved by the exact equation thrust planning software for cases 1 and 2, respectively. To calculate thrust plans that matched the final orbits this precisely, the averaged equation thrust planning software was first used to calculate a set of Lagrange multipliers starting from initial guesses of unity. The resulting averaged Lagrange multipliers for ARTEMIS case 1 are shown in Table 5.11.

Table 5.11 Initial Time Averaged Lagrange Multipliers for ARTEMIS Case 1

Lagrange Multiplier	Solution Value
$\left(\tilde{\lambda}_a\right)_0$ (s/km)	0.507334808835E+04
$\left(\tilde{\lambda}_h\right)_0$ (sec)	-0.999690395153E+08
$\left(\tilde{\lambda}_k\right)_0$ (sec)	-0.743508661166E+08
$\left(\tilde{\lambda}_p\right)_0$ (sec)	0.135424135144E+08
$\left(\tilde{\lambda}_q\right)_0$ (sec)	0.649226125343E+07
$\left(\tilde{\lambda}_\lambda\right)_0$ (rad)	-0.975118613527E-01

The resulting averaged Lagrange multipliers for ARTEMIS case 2 are shown in Table 5.12.

Table 5.12 Initial Time Averaged Lagrange Multipliers for ARTEMIS Case 2

Lagrange Multiplier	Solution Value
$\left(\tilde{\lambda}_a\right)_0$ (s/km)	0.570932635372E+04
$\left(\tilde{\lambda}_h\right)_0$ (sec)	-0.344890855691E+08
$\left(\tilde{\lambda}_k\right)_0$ (sec)	0.559045469505E+08
$\left(\tilde{\lambda}_p\right)_0$ (sec)	0.198345266291E+02
$\left(\tilde{\lambda}_q\right)_0$ (sec)	0.287833885967E+03
$\left(\tilde{\lambda}_\lambda\right)_0$ (rad)	0.764058716901E-02

The averaged equation code calculated a thrust plan with a constant Hamiltonian value equal to 1.000042 for ARTEMIS case 1 and a value of 1.000000 for ARTEMIS case 2. These Hamiltonian values are both very close to the value of exactly one, and this agreement indicates that the necessary conditions for optimality in ARTEMIS cases 1 and 2 have been met. Reference (47) indicates that the ion thrusters onboard ARTEMIS can produce between 21 mN and 27 mN of thrust. With a spacecraft mass of 3100 kg (47), this yields a thrust acceleration of between 6.77×10^{-9} and 8.71×10^{-9} km/s^2. As a first guess, 7.75×10^{-9} km/s^2 was chosen as the thrust acceleration for the averaged equation software run. For ARTEMIS case 1, the resulting transfer time from the averaged equation software was 830,823.3 seconds. This is less than the transfer time of 853,717.0 seconds known from the initial and final TLE orbits. However, because the averaged equation thrust planning software is simply intended to be used as a robust tool from which reasonable initial guesses for the Lagrange multipliers can be obtained, the difference between the known and solved values of the transfer time was deemed

sufficiently small. For ARTEMIS case 2, the resulting transfer time from the averaged equation software was 1,767,002.2 seconds. This is reasonably close to the known transfer time of 1,761,276.3 seconds. Adjusting the constant acceleration thrust level to obtain agreement with the transfer time was later done using the exact equation thrust planning software. It should be noted that in order to achieve the final desired orbit to a high degree of precision using the averaged equation thrust planning tool, adjustment of the weights in the cost function for the quasi-Newton search was done by hand in several iterations. These weights are depicted in equation (2.101).

The exact equation thrust planning tool was initialized with the initial guesses for the Lagrange multipliers and transfer time solved for by the averaged equation planning software. In order to produce a trajectory that closely matched the known transfer time of 853,717.0 seconds for ARTEMIS case 1 and 1,7612,76.3 seconds for ARTEMIS case 2, the exact equation software was iterated while the constant thrust acceleration was adjusted. As each iteration converged on a new set of values for the Lagrange multipliers and transfer time, the thrust acceleration was adjusted using the assumption that an excessively long transfer time indicated that the constant thrust acceleration was too small. Conversely, a short transfer time indicated that the constant thrust acceleration was too large. For ARTEMIS case 1, these iterations eventually resulted in an optimal thrust trajectory with a transfer time of 853,721.0 seconds. This is too long by 4 seconds, but is relatively close to the desired value. The constant value of the thrust acceleration used for this trajectory was 7.5628×10^{-9}. This is close to the initial guess of 7.5×10^{-9}

and corresponds to a thrust force of 23.44468 mN for a 3100 kg spacecraft. The solutions for the initial Lagrange multipliers for ARTEMIS case 1 are shown in Table 5.13.

Table 5.13 Initial Time Exact Lagrange Multipliers for ARTEMIS Case 1

Lagrange Multiplier	Solution Value
$\left(\lambda_a^L\right)_0$ (s/km)	0.326213810986E+04
$\left(\lambda_h^L\right)_0$ (sec)	-0.671777405244E+08
$\left(\lambda_k^L\right)_0$ (sec)	-0.478560629212E+08
$\left(\lambda_p^L\right)_0$ (sec)	0.157221510209E+05
$\left(\lambda_q^L\right)_0$ (sec)	-0.128834477535E+05
$\left(\lambda_L^L\right)_0$ (rad)	0.584382550521E+03

The Hamiltonian for the converged solution was equal to 1.0000052. This indicates a solution in which the necessary condition for optimality is met with a high degree of numerical precision. The final orbit achieved is shown in Table 5.9 and also shows that the thrust plan matches the final desired orbit with a high degree of numerical precision.

For ARTEMIS case 2, the end result was an optimal thrust trajectory with a transfer time of 1,761,200.1 seconds. This differs from the known transfer time by only 76.1 seconds. The known transfer time is the time difference between the epoch times of the initial and final orbits of the transfer. The constant value of the thrust acceleration used for this trajectory was 6.5113×10^{-9}. This is less than the solution value of 7.5628×10^{-9} obtained in ARTEMIS test case 1. ARTEMIS test case 1 occurs near the beginning of the *back-up control strategy* while test case 2 occurs near the end. Perhaps in the

several months of thrusting that occurred between the time spans of the two test cases, the thrust level on ARTEMIS was reduced. From open literature sources documenting the ARTEMIS orbit raising, i.e. references (87) and (47), it is not clear why this occurred. 6.5113×10^{-9} corresponds to a thrust force of 20.185 mN for a 3100 kg spacecraft. This assumes that there was no mass difference in the spacecraft between case 1 and 2. Because Xenon fuel was being used, the mass must have decreased. However, this does not account for the reduced thrust acceleration from test case 1 to test case 2. In fact, the thrust acceleration due to the ion engine should increase over time as fuel is being spent and the spacecraft mass decreases.

The solutions for the initial, exact equation Lagrange multipliers for ARTEMIS case 2 are shown in Table 5.14.

Table 5.14 Initial Exact Lagrange Multipliers for ARTEMIS Case 2

Lagrange Multiplier	Solution Value
$\left(\lambda_a^L\right)_0$ (s/km)	0.561982107075E+04
$\left(\lambda_h^L\right)_0$ (sec)	-0.178751941273E+08
$\left(\lambda_k^L\right)_0$ (sec)	0.513808871168E+08
$\left(\lambda_p^L\right)_0$ (sec)	-0.154227994948E+04
$\left(\lambda_q^L\right)_0$ (sec)	-0.157509957534E+04
$\left(\lambda_L^L\right)_0$ (rad)	0.313251259293E+04

The Hamiltonian for the converged solution for ARTEMIS case 2 was equal to 0.9999985. As in case 1, this Hamiltonian solution indicates that the necessary condition

for optimality is met with a high degree of numerical precision. The final orbit achieved for case 2 is shown in Table 5.10. The thrust plan matches the final desired orbit with a high degree of numerical precision.

Figures 5.19-5.21 show selected orbital element histories and the pitch and yaw thrust plan over the transfer trajectory for ARTEMIS case 1. The thrust plan was generated with the exact equation thrust planning software which uses only two-body motion and thrust acceleration dynamics.

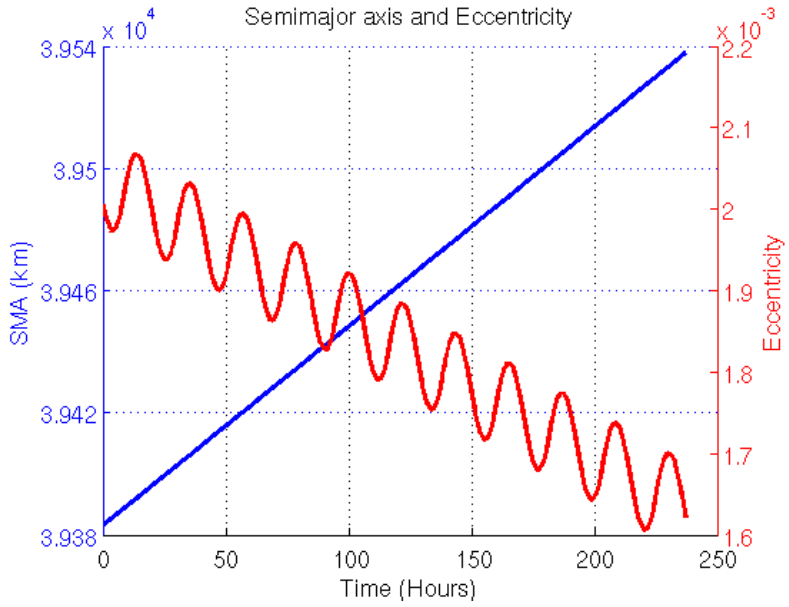

Figure 5.19 Semimajor axis and Eccentricity for ARTEMIS Optimal Thrust Plan Case 1

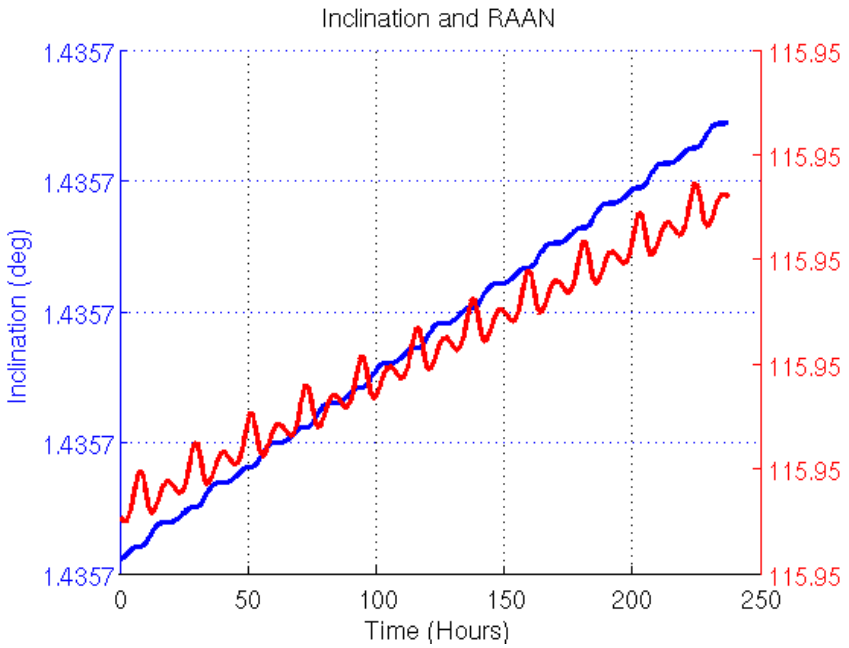

Figure 5.20 Inclination and RAAN for ARTEMIS Optimal Thrust Plan Case 1

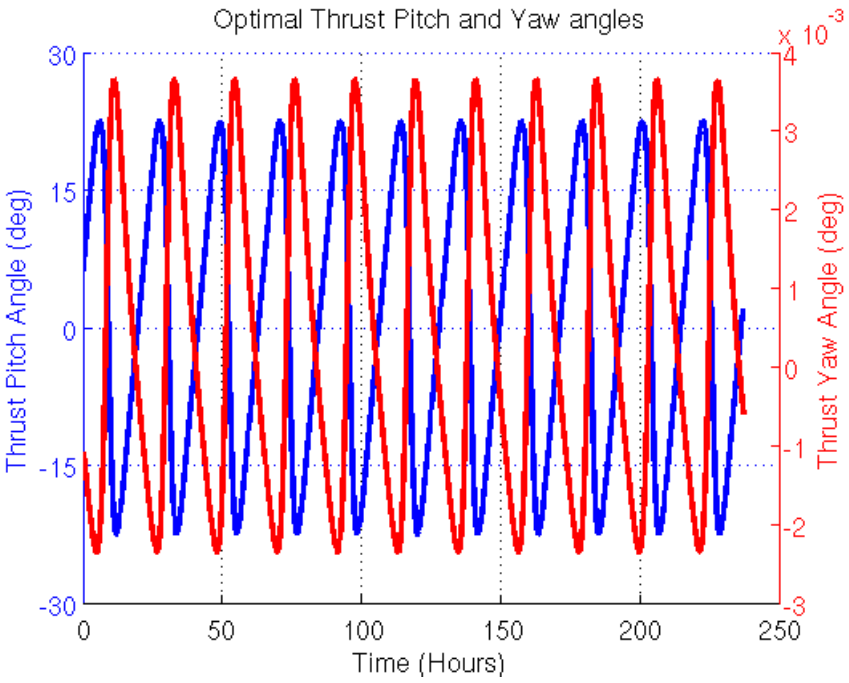

Figure 5.21 Pitch and Yaw Thrust Angles for Optimal Thrust Plan Case 1

Figure 5.19 shows that the semimajor axis undergoes an apparently linear increase from the initial boundary condition to the final boundary condition. The eccentricity

follows an oscillatory path during the transfer from the initial to final boundary condition. From Figure 5.20, it is clear that the inclination and RAAN change over time during the transfer, but the change is so small that it is less than the precision shown in the ordinate axis. This was the desired behavior because the inclination and RAAN were not supposed to change significantly as a result of thrust during this transfer. Figue 5.21 shows that the yaw angles used in controlling the trajectory are very small in relation to the pitch angles used. This relates to the very small changes due to thrust in the orbital plane compared to the larger changes due to thrust in the orbit's semimajor axis and eccentricity. The oscillations in the eccentricity seem to be caused by the varying pitch angle during the course of the thrust plan. Perturbations such as solar radiation pressure were not modeled in computing this thrust plan. Therefore, the changes in eccentricity must be due to the thrust acceleration.

Figures 5.22-5.24 show selected orbital element histories and thrust pitch and yaw directions over the transfer trajectory for ARTEMIS case 2.

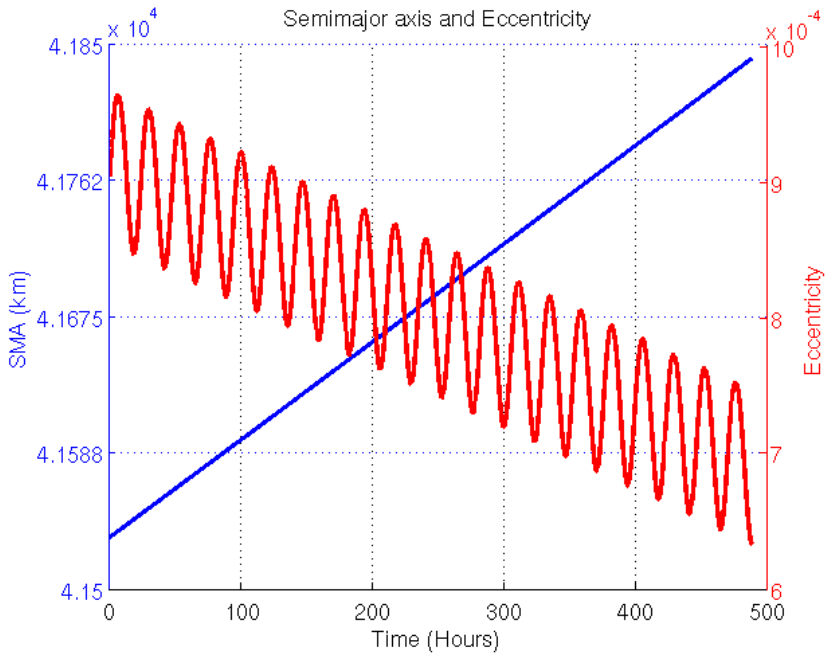

Figure 5.22 Semimajor axis and Eccentricity for ARTEMIS Optimal Thrust Plan Case 2

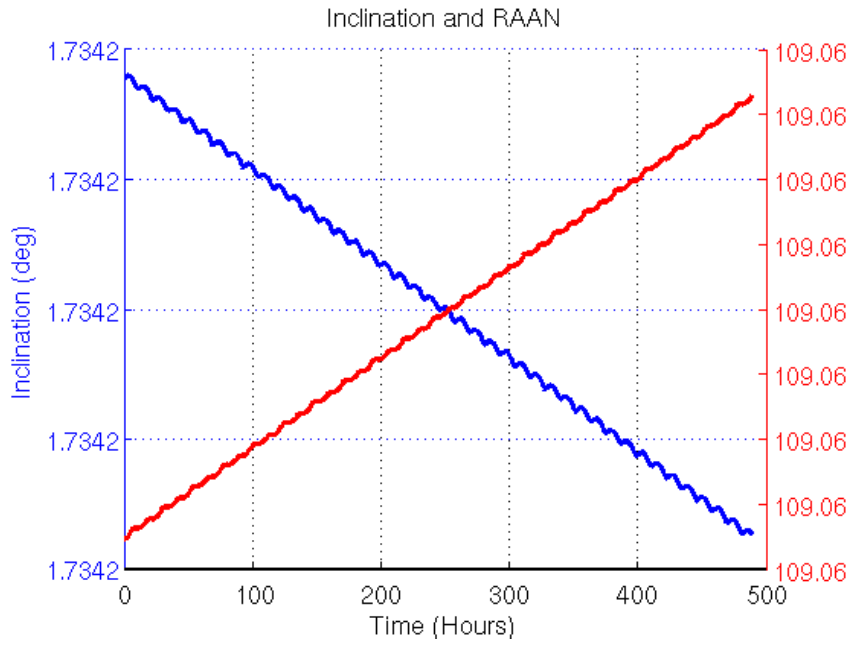

Figure 5.23 Inclination and RAAN for ARTEMIS Optimal Thrust Plan Case 2

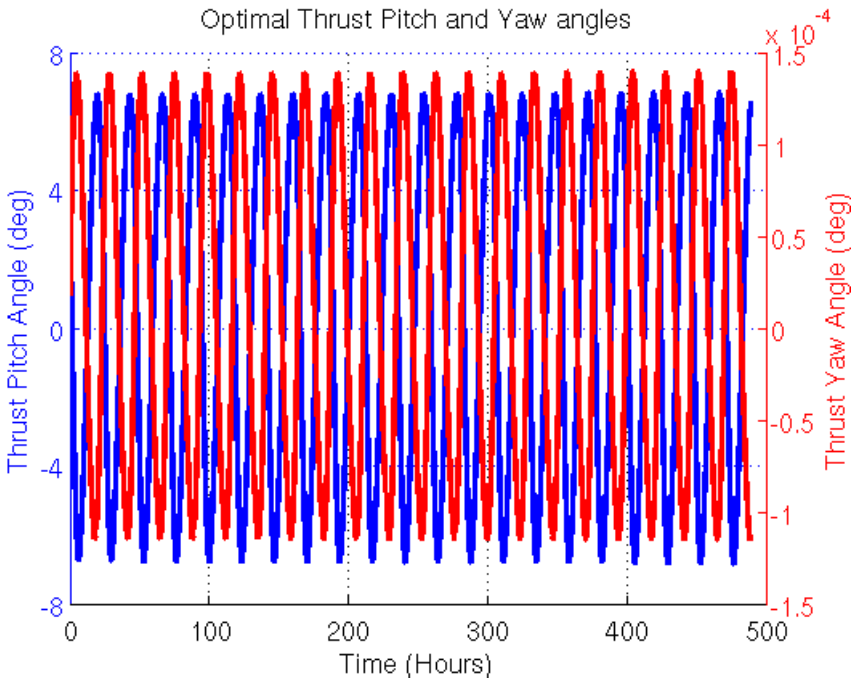

Figure 5.24 Pitch and Yaw Thrust Angles for Optimal Thrust Plan Case 2

Figure 5.22 shows that in case2, as in case 1, the semimajor axis undergoes a roughly linear increase from the initial orbit to the final orbit. In case 2, the eccentricity also follows an oscillatory path during the transfer. Cases 1 and 2 differ in the small inclination and RAAN changes shown in Figures 5.20 and 5.23, respectively. For case 1, the inclination and RAAN both increase while the inclination decreases over the transfer in case 2. However, as expected, the inclination and RAAN change little over the time of the transfer. The yaw angles used in controlling the trajectory are again very small in relation to the pitch angles used. The pitch and yaw angle oscillations shown in Figure 5.24 for case 2 are about three times less than the amplitudes of the pitch and yaw angle oscillations in Figure 5.21 for case 1. It seems the optimal thrust planning software chooses similar plans for cases 1 and 2, but smaller amplitudes in the thrust control angle oscillations are required for the longer transfer.

As mentioned before, inertial, osculating MEME of 1950 position/velocity vectors converted from TLEs were used as observations in GTDS Cowell differential corrections (DCs). In addition, AFSSN observations were used in a separate set of GTDS DCs. Specifically, the GTDS DC program used an iterative Bayes' Least Squares estimator. The force models used with the Cowell propagator for these DCs included 12x12 geopotential coefficients from the JGM-2 geopotential model, lunar and solar gravitational perturbations, solar radiation pressure and Earth polar motion. The solve-for vector was the Cartesian position and velocity in the MEME of 1950 reference frame. Starting from a reasonable a-priori orbital estimate, this nonlinear estimator should find the orbit that best fits a given set of observations in a least-squares sense. The DCs, i.e. fits, were done in order to evaluate whether optimal thrust plans provide any modeling improvement for the ARTEMIS satellite orbit during the August 4-14, 2002 and the December 27, 2002 – January 16, 2003 test case time spans. For both the TLE position/velocity vectors and the AFSSN radar and optical observations, two GTDS differential correction (DC) runs were executed. The first run for each set of observations did not use the optimal thrust plan generated by the exact equation optimal thrust planning software. The second run did use the optimal thrust plan. These GTDS DC runs were then compared in terms of the TLE and AFSSN observation residual statistics. The inertial, Cartesian position/velocity vector residuals, i.e. differences between observed and computed observations, for the TLE based observations in ARTEMIS case 1 are shown in Figures 5.25-5.27. The residuals obtained when the optimal thrust plan is used are shown as red crosses. The residuals obtained when no

thrust plan model is used are shown as blue circles. The Cartesian (X,Y,Z) position and velocity residuals are closer to zero-mean and have smaller variances when the thrust plan is integrated with the GTDS differential corrections versus when the plan is ignored. The improvement in residual statistics is clear evidence that the optimal thrust plan is improving the accuracy of the orbit modeling for ARTEMIS test case 1.

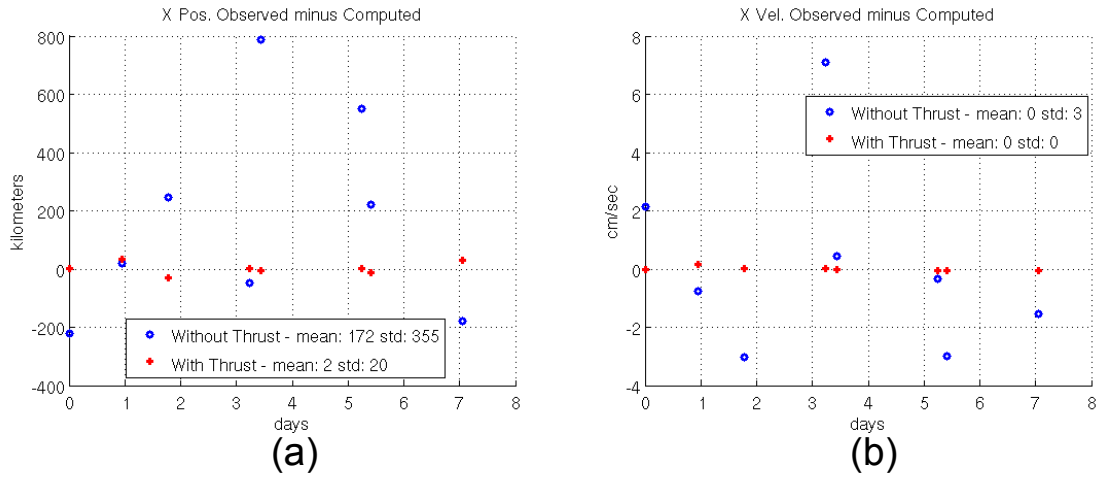

Figure 5.25 Case 1 Residuals of TLE based ECI osculating position (a) and velocity (b) vectors (X-direction)

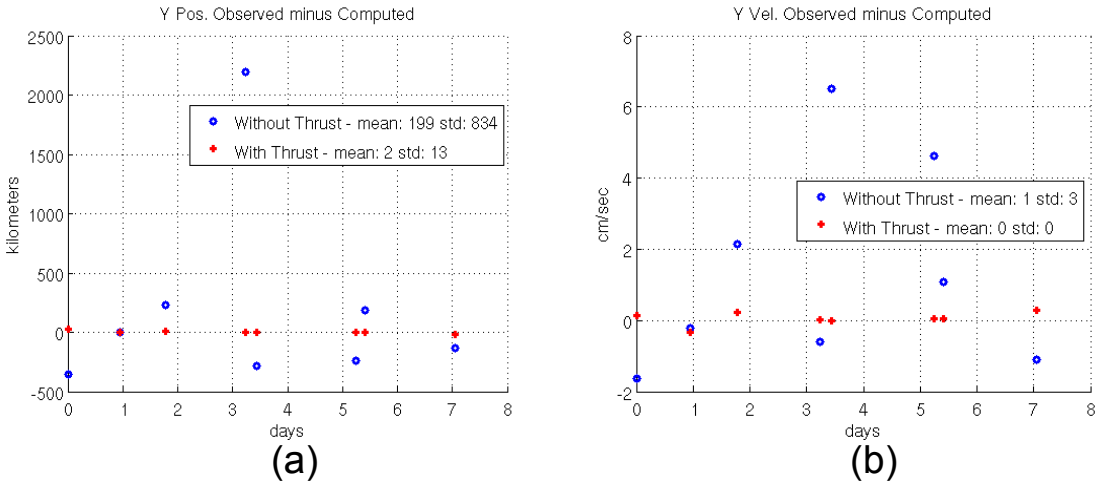

Figure 5.26 Case 1 Residuals of TLE based ECI osculating position (a) and velocity (b) vectors (Y-direction)

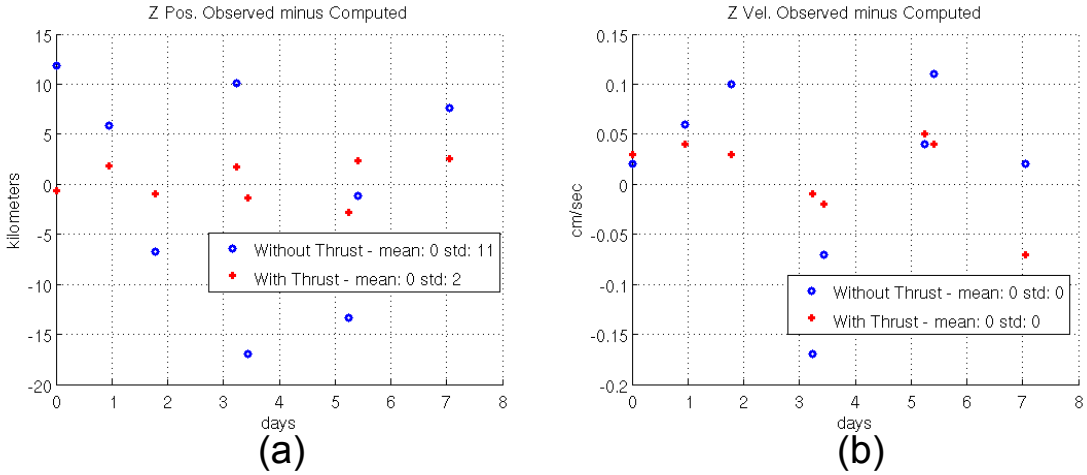

Figure 5.27 Case 1 Residuals of TLE based ECI osculating position (a) and velocity (b) vectors (Z-direction)

The improvement in modeling accuracy is also apparent when AFSSN radar and optical observations are used in a differential correction (DC) with the optimal thrust plan. The residual statistics when the optimal thrust plan is ignored in the DC for ARTEMIS case 1 are shown in Table 5.15.

Table 5.15 DC Residual Statistics for Case 1 when Thrust Plan is Ignored

Type	DEC (arcs)	RA (arcs)	TDEC (arcs)	TRA (arcs)	Azimuth (arcs)	Elevation (arcs)	Range (m)	Doppler (cm/s)
Total No.	10	10	73	73	70	70	70	70
No. Accepted	0	0	6	15	20	50	0	0
Mean Residual	0.0	0.0	-1.3E5	-5.6E4	-2.9E5	2.9E4	0.0	0.0
Std. Dev.	0.0	0.0	5.6	6.2E4	4.3E5	7.4E4	0.0	0.0

The case 1 DC fit to the AFSSN observations when ignoring the thrust plan is very poor. Over the fitspan, 446 observations are available (Total No.), but only 91, 26%, are

included in the fit (No. Accepted). The 74% of observations that were not included were rejected because the residuals for those observations surpassed the 3σ threshold. This means that most of the residuals were more than three standard deviations away from the mean value. The combined weighted RMS for all residuals was 2910. This is also known as the chi-squared statistic and is a measure of *goodness of fit*. If the residuals match the expected variances for the measurements, the weighted RMS should roughly equal 1.0. A value of 2910 clearly indicates that the best fit achieved using the Bayes' Least Squares estimator is still a poor fit of the observations. The covariance of the solve-for vector can also be examined for this fit. The standard deviation in the position/velocity solve-for vectors is on the order of 1.0×10^{68}. This nonsensical covariance indicates very large uncertainty in the solve-for vector values. The GTDS Cowell DC fit results indeed show very poor agreement with the AFSSN observations.

If the optimal thrust plan is used in the orbit prediction during the differential correction (DC), the AFSSN residual statistics in Table 5.16 are obtained for ARTEMIS case 1.

Table 5.16 DC Residual Statistics for Case 1 when Modeled with Thrust Plan

Type	DEC (arcs)	RA (arcs)	TDEC (arcs)	TRA (arcs)	Azimuth (arcs)	Elevation (arcs)	Range (m)	Doppler (cm/s)
Total No.	10	10	73	73	70	70	70	70
No. Accepted	10	10	62	48	60	70	45	70
Mean Residual	55.6	230	38.1	48.9	-119.6	-48.6	-5.02	-4.92
Std. Dev.	2.11	0.82	49.7	126.7	206.2	184.8	57.17	100.9

Residual statistics displayed in Table 5.16 show that satellite motion modeling which includes the optimal thrust plan results in a better fit of the residuals. However, the residual statistics are not at the level expected for the given observations. Radar range measurements, for example, are typically near zero-mean and can have a variance as small as 5-20 meters. Radar angular measurements, i.e. Azimuth and Elevation, should have residual statistics on the order of 20-30 arcseconds. Optical measurements, i.e. RA and DEC, should have residual statistics on the order of 10-15 arcseconds. The larger than expected residual statistics indicate that the optimal thrust plan is not precisely equal to the thrust plan actually executed by ARTEMIS. Many more observations were included in this fit than in the previous fit for case 1. Out of 446 observations, 375 or 84% were included. This means that the fit is including many more observations than the fit done without thrust modeling. The combined weighted RMS of all residuals is 7.08. This is still not an ideal fit because 7.08 is much larger than 1.0. However, this is much better than 2910. The standard deviations of the Cartesian position solve-for values are between 0.12 and 1.2 km. The covariance of the solve-for state in this fit indicates much more certainty of the satellite's position than in the fit which ignored thrust acceleration.

The SSN radar range and range rate (doppler) measurement residuals are plotted in Figures 5.28a and 5.28b. These residuals were the result when the optimal thrust plan was included in the GTDS Cowell DC fit.

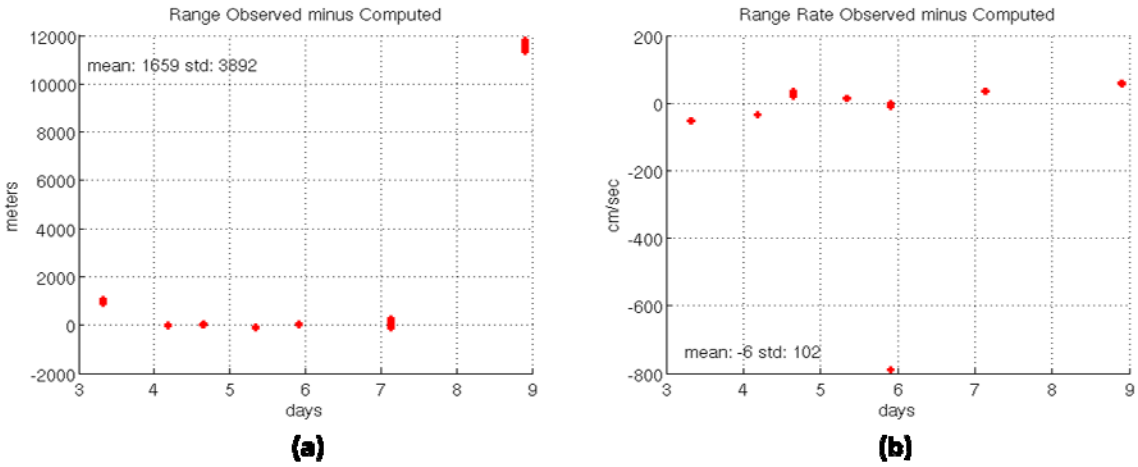

Figure 5.28 Range (a) and Range Rate (b) Residuals for ARTEMIS Case 1

The mean and standard deviation of the range and range rate measurement residuals displayed in Figure 5.28 differ from those in Table 5.16 because the statistics in Table 5.16 only include residuals that were accepted according to the 3σ criterion. The statistics in Figure 5.28 include all measurement residuals regardless of acceptance according to the 3σ criterion. The range residuals should not be much larger than about 20 meters given the radar measurements included in the DC fit. The larger than expected range residuals indicate some mismodeling in the GTDS DC fit. The optimal thrust plan seems to represent the actual ARTEMIS thrust strategy with some degree of inaccuracy. From Figure 5.28, it appears as though range measurements near the end of the fitspan are not fitting as well as observations in the beginning and middle of the fitspan. This could indicate that the thrust modeling is more inaccurate near the end of the fitspan. The range rate measurements appear to fit reasonably well except for a number of measurements near the end of day 5 in the fitspan. This could indicate an inaccurate radar track or inaccurate thrust modeling at that point in the fitspan.

Figure 5.29 shows the radar azimuth and elevation measurement residuals for ARTEMIS case 1 when the optimal thrust plan is included in the GTDS Cowell DC fit.

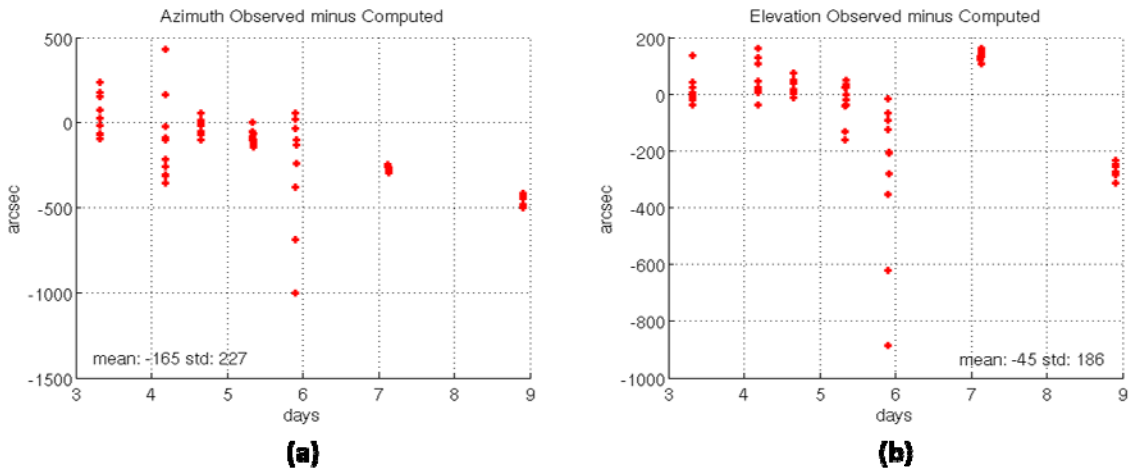

Figure 5.29 Azimuth (a) and Elevation (b) Residuals for ARTEMIS Case 1

In Figure 5.29 as in Figure 5.28, the mean and standard deviation statistics of the measurement residuals differ from those in Table 5.16 because Figure 5.29 includes all measurement residuals regardless of whether they were included in the fit according to the 3σ criterion. In Figure 5.29, azimuth and elevation residuals are much larger than the 20 arcsecond residuals one would expect given the radar measurements used in the GTDS DC fit. This indicates the optimal thrust plan used for this fit is inaccurate to some degree.

The optical right ascension and declination measurement residuals from the GTDS Cowell DC fit for ARTEMIS case 1 are shown in Figure 5.30. Here, the optimal thrust plan was included in the fit.

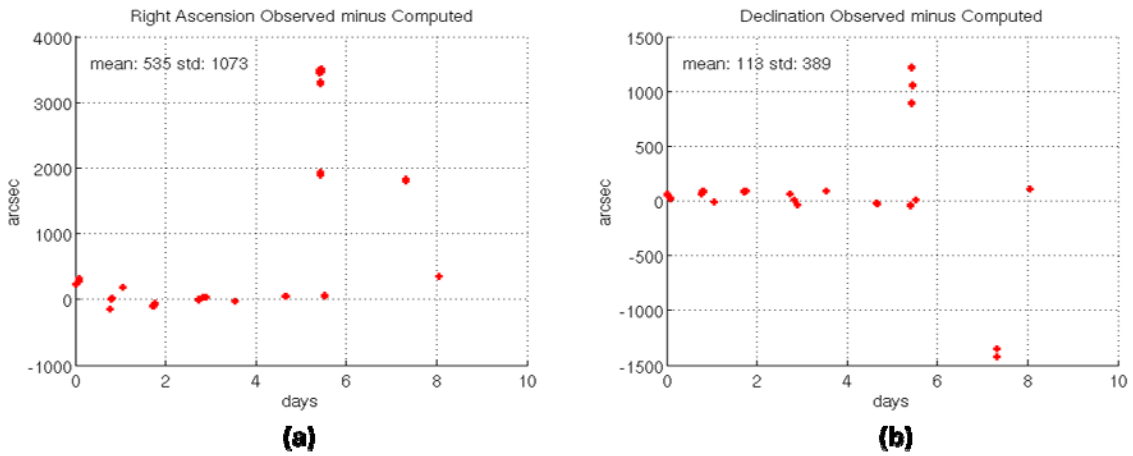

Figure 5.30 Right Ascension (a) and Declination (b) Residuals for ARTEMIS Case 1

In Figure 5.30, as in Figures 5.28 and 5.29, all residuals are included regardless of the 3σ criterion used in the GTDS DC. These residual mean and standard deviation statistics therefore differ from those in Table 5.16. The right ascension and declination residuals are not expected to be much larger than 15 arcseconds given the sensors supplying the observations. However, Figure 5.30 shows that the residuals are significantly larger than expected. Both the right ascension and declination residuals increase significantly after day 5 in the fitspan. This could mean that the thrust plan is more inaccurate in the latter half of the 10 day span than in the first half. This was also hinted at in the range residuals shown in Figure 5.28. It is clear that systematic errors in the thrust modeling prevent measurement residuals that exhibit only sensor measurement noise characteristics. Despite these noted imperfections, the lessening of the TLE derived position/velocity vector residuals demonstrated in Figures 5.25 – 5.27 shows that the optimal thrust plan does significantly improve thrust motion modeling for ARTEMIS.

ARTEMIS test case 2 shows improvement that is similar to the improvement demonstrated in case 1 when thrust acceleration is modeled with an optimal thrust plan. Figures 5.31-5.33 show the Cartesian, osculating position and velocity residuals when fit with and without optimal thrust modeling.

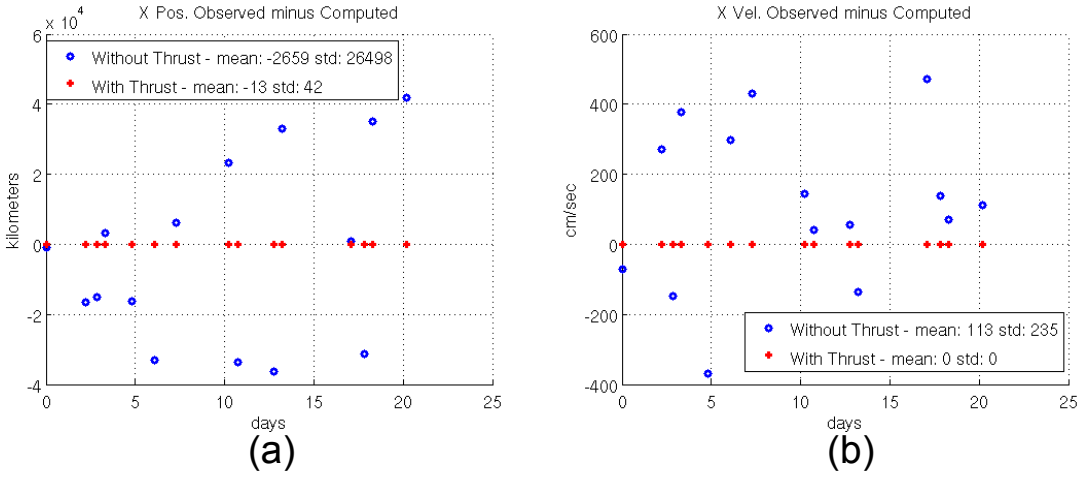

Figure 5.31 Case 2 Residuals of TLE based ECI osculating position (a) and velocity (b) vectors (X-direction)

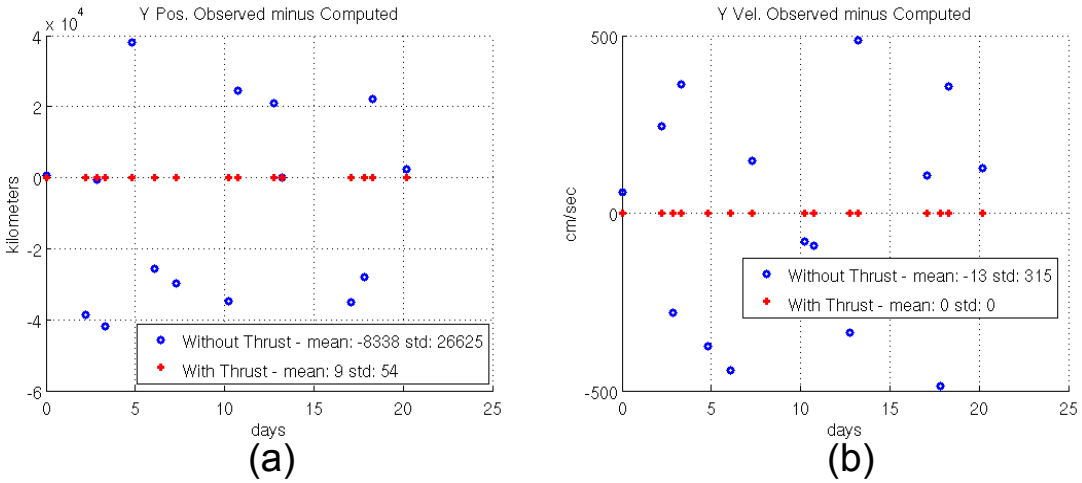

Figure 5.32 Case 2 Residuals of TLE based ECI osculating position (a) and velocity (b) vectors (Y-direction)

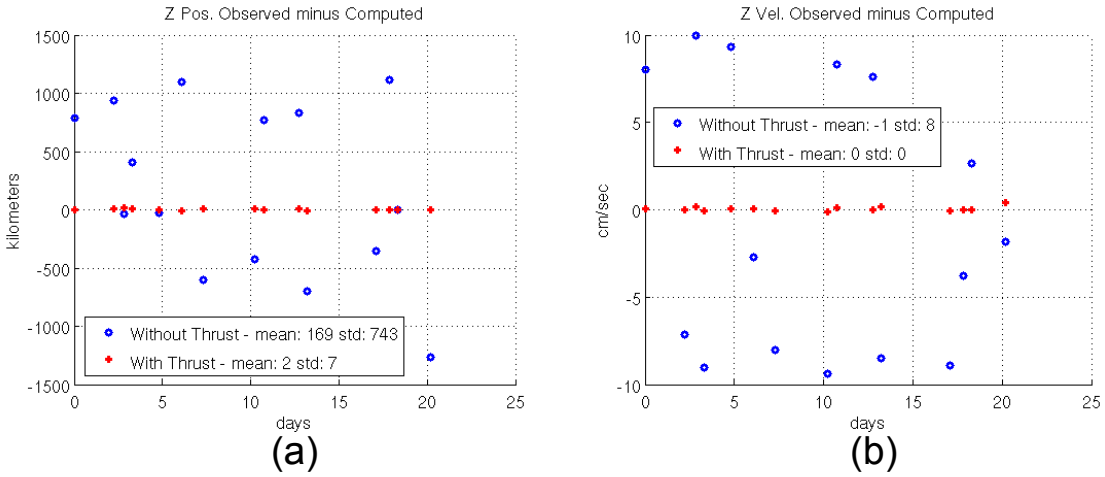

Figure 5.33 Case 2 Residuals of TLE based ECI osculating position (a) and velocity (b) vectors (Z-direction)

In Figures 5.31-5.33, the improvement in the ARTEMIS case 2 position/velocity residuals is more dramatic than for the case 1 residuals. Case 2 spans roughly 20 days while case 1 spans about 10 days. Therefore, attempting to fit the TLE-based position/velocity vectors is more difficult in case 2 because of the larger displacement of the orbit due to the continuous thrusting. Results from ARTEMIS test case 2 also demonstrate significant improvement in modeling the satellite's artificial motion when using the optimal thrust plan.

When fitting AFSSN observations without thrust modeling in case 2, the GTDS Cowell DC software is unable to converge. The DC rejects all observations and three consecutive iterations in the nonlinear estimator diverge rather than converge. It is apparently very difficult to fit observations of ARTEMIS over the 20 day span when no thrust modeling is attempted. When the optimal thrust plan for case 2 is applied, the

GTDS DC program is able to converge and produces the observation residual statistics shown in Table 5.17.

Table 5.17 DC Residual Statistics for Case 2 when Modeled with Thrust Plan

Type	DEC (arcs)	RA (arcs)	TDEC (arcs)	TRA (arcs)	Azimuth (arcs)	Elevation (arcs)	Range (m)	Doppler (cm/s)
Total No.	29	29	32	32	183	183	183	180
No. Accepted	29	24	31	28	181	183	58	180
Mean Residual	68.23	-158.1	37.01	142.9	3.862	38.13	-4.74	0.318
Std. Dev.	25.81	282.8	32.61	138.5	316.8	210.8	80.23	48.35

The DC accepts most of the observations. Out of 851 observations, 714, or 83%, are accepted by the 3σ criterion. The combined weighted RMS statistic is 14.74 for the fit. This is significantly larger than the ideal value of 1.0. Also, the individual measurement residual statistics shown in Table 5.17 are larger than expected. Both facts point to remaining modeling errors for the ARTEMIS trajectory over this 20 day span. The optimal thrust plans generated by the thrust plan software are not yet accurate enough to allow orbit fits that produce residuals that exhibit only measurement noise. Systematic errors caused by the inaccurate thrust plans prevent such high precision orbit fits. The covariance of the position estimate shows variances on the order of between 1.0 and 1.7 km. When modeling the thrust plan, the uncertainty in the satellite's position is therefore only slightly higher for case 2 than it is for case 1.

Figure 5.34 shows the radar range and range rate measurement residuals for ARTEMIS case 2 when the optimal thrust plan was included in the GTDS DC fit. The

mean and standard deviation of the residuals shown in Figures 5.34-5.36 differ from those in Table 5.17 because in Table 5.17, only residuals accepted in the fit according to the 3σ criterion are included. In Figures 5.34-5.36 all residual are included.

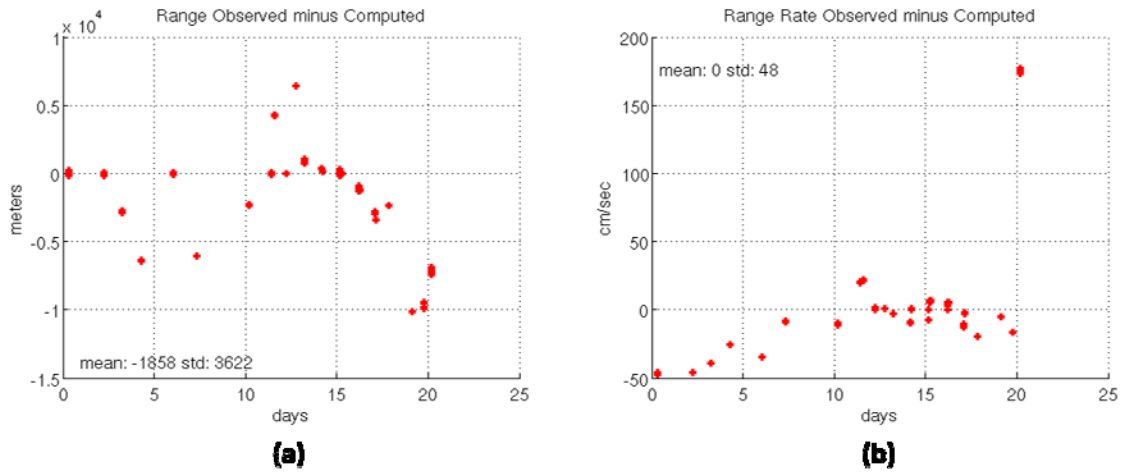

Figure 5.34 Range (a) and Range Rate (b) Residuals for ARTEMIS Case 2

As in Figure 5.30 for case 1, the range residuals are larger than expected. The range residuals for the radar sensors used in the GTDS Cowell DC fit would not be much larger than 20 meters if the ARTEMIS motion modeling was near perfect. As in case 1, the larger than expected range residuals for case 2 indicate that inaccuracies in the thrust modeling are indeed present.

Figure 5.35 shows the radar azimuth and elevation measurements for ARTEMIS case 2 when optimal thrust modeling is used in the GTDS Cowell DC.

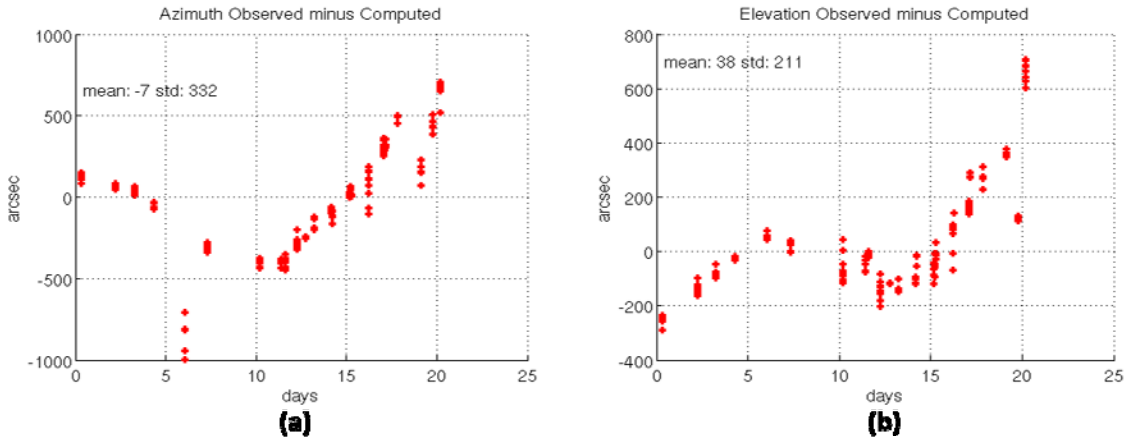

Figure 5.35 Azimuth (a) and Elevation (b) Residuals for ARTEMIS Case 2

The expected residuals for the azimuth and elevation angle radar observations would be on the order of 20 arcseconds if the ARTEMIS thrust acceleration and natural motion modeling was near perfect. Because the azimuth and elevation angle residuals are significantly larger than expected, it is likely the thrust modeling is inaccurate to some degree. There is also a trend in the residuals that indicates periodic errors in the modeling.

Figure 5.36 shows the optical right ascension and declination residuals for ARTEMIS case 2 when fit in a GTDS Cowell DC. The optimal thrust plan was used in the fit.

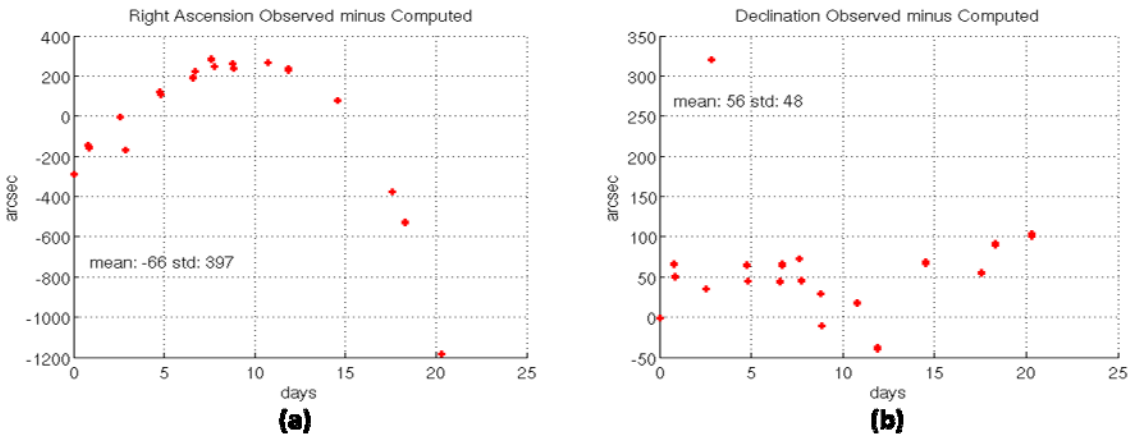

Figure 5.36 Right Ascension (a) and Declination (b) Residuals for ARTEMIS Case 2

The expected residuals would be on the order of 15 arcseconds if near perfect modeling of the ARTEMIS thrust was accomplished. The right ascension and declination residuals shown in Figure 5.36 show that near-perfect modeling was indeed not achieved. The right ascension residuals show a parabolic pattern over the 20 day span. This indicates that the along-track motion of ARTEMIS is not being modeled accurately. The declination residuals show a bias of around 50 arcseconds. This indicates that the ARTEMIS cross-track motion is not being modeled accurately.

5.4.4 ARTEMIS Orbit Determination Case 3

The third ARTEMIS test case was chosen to coincide with the *inclination control strategy*. The test case span was specifically chosen to coincide with the obvious change in inclination around day 100 as displayed in Figure 5.18. This third test case included only AFSSN TLEs. AFSSN radar and optical observations could not be obtained in time for this thesis. The test case start and end times were November 9, 2001 10:20:58.513

UTC and November 28, 2001 03:09:02.650 UTC, respectively. This span is about 18 days in duration. As in test cases 1 and 2, all available TLEs during these time spans were converted to osculating Keplerian element sets and position/velocity vectors in the Mean Equator, Mean Equinox (MEME) of 1950 reference frame. The Keplerian elements were used to serve as inputs to the optimal thrust planning software while the position/velocity vectors were later used as observations in GTDS differential correction runs. The first and last resulting Keplerian element sets were used as the initial and final orbits in the optimal thrust planning software. These initial and final orbits are shown in Table 5.18.

Table 5.18 Initial and Final Orbits for ARTEMIS Case 3

Orbit	a, (km)	e	i, (deg)	Ω, (deg)	ω, (deg)	M, (deg)
Initial	37303.61026	0.0007899398	1.0670144	135.6513	81.55172	238.70057
Final	37300.27217	0.0005757628	0.9876237	135.6513	81.55172	Free
Achieved	37300.27431	0.0005757811	0.9876415	135.6502	81.55504	88.196088

In test case 3, the Keplerian elements for the final orbit were modified from the converted TLE at the final time because the right ascension of the ascending node, the argument of perigee, and the mean anomaly were not actively controlled during the *inclination control strategy*. However, the semimajor axis, eccentricity, and inclination were deliberately changed according to the thrust control strategy executed by the ARTEMIS operators. Figures 5.17-5.18 show that the eccentricity and inclination have a decreasing trend during the case 3 interval. These trends seem to be isolated to the span approximately marked by the case 3 boundaries. This implies that the case 3 interval approximates a time period when the ARTEMIS operators actively controlled the

eccentricity and inclination. It seems as though the operators didn't control the eccentricity and inclination during the times immediately surrounding the case 3 interval. The semimajor axis also displays a decrease of around two kilometers during the case 3 interval, but that change is too small to be seen in Figure 5.16.

Table 5.18 shows the final orbit achieved by the exact equation thrust planning software for cases 3. This orbit nearly matches the desired final orbit. However, the agreement is not as good as that obtained in ARTEMIS cases 1 and 2. In ARTEMIS case 3, the exact equation thrust planning software was not able to minimize the cost function to a high degree of precision. This may be due to test case 3 having more periodicity in the orbital elements and Lagrange multipliers over the transfer time. There are better algorithms for solving the cost minimization than the quasi-Newton unconstrained minimization method used in this work and exploration of these more advanced methods should be addressed in future work.

To calculate the thrust plan that matched the final orbit with reasonable precision, the averaged equation thrust planning software was first used to calculate a set of averaged Lagrange multipliers starting from initial guesses of unity. The resulting averaged Lagrange multipliers for ARTEMIS case 3 are shown in Table 5.19.

Table 5.19 Initial Time Averaged Lagrange Multipliers for ARTEMIS Case 3

Lagrange Multiplier	Solution Value
$\left(\tilde{\lambda}_a\right)_0$ (s/km)	-0.772868765055E+02
$\left(\tilde{\lambda}_h\right)_0$ (sec)	0.227840011762E+07
$\left(\tilde{\lambda}_k\right)_0$ (sec)	0.311020912332E+07
$\left(\tilde{\lambda}_p\right)_0$ (sec)	-0.161366219567E+10
$\left(\tilde{\lambda}_q\right)_0$ (sec)	0.168962077600E+10
$\left(\tilde{\lambda}_\lambda\right)_0$ (rad)	-0.489428314171E-01

The averaged equation code calculated a thrust plan with a constant Hamiltonian value equal to 1.000000 for ARTEMIS case 3. The Hamiltonian value is very close the value of exactly one, and this agreement indicates that the necessary condition for optimality in ARTEMIS case 3 has been met. Reference (47) indicates that the ion thrusters onboard ARTEMIS can produce between 21 mN and 27 mN of thrust. With a spacecraft mass of 3100 kg (47), this yields a thrust acceleration of between 6.77×10^{-9} and 8.71×10^{-9} km/s^2. Because the inclination control strategy discussed in reference (88) involved about 5 hours of thrusting per day rather than continuous thrusting, a reduced guess for the thrust acceleration was used in this case. By iterating the averaged equation thrust planning software to achieve a computed transfer time approximately equal to the known transfer time, a constant thrust acceleration of 4.4 km/s^2 was converged upon. This thrust acceleration resulted in a transfer time of 1,619,696 seconds. This is greater than the transfer time of 1,615,684 seconds known from the

initial and final TLE orbits. However, because the averaged equation thrust planning software is simply intended to be used as a robust tool from which reasonable initial guesses for the Lagrange multipliers can be obtained, the difference between the known and solved values of the transfer time was deemed sufficiently small. It should be noted that in order to achieve the final desired orbit to a high degree of precision using the averaged equation thrust planning tool, adjustment of the weights in the cost function for the quasi-Newton search was done by hand in several iterations. These weights are depicted in equation (2.101).

The exact equation thrust planning tool was initialized with the initial guesses for the Lagrange multipliers, the constant thrust acceleration value, and the transfer time solved for by the averaged equation planning software. In order to produce a trajectory that closely matched the known transfer time of 1,615,684 seconds for ARTEMIS case 3, the exact equation software was iterated while the constant thrust acceleration was adjusted. As each iteration converged on a new set of values for the Lagrange multipliers and transfer time, the thrust acceleration was adjusted using the assumption that an excessively long transfer time indicated that the constant thrust acceleration was too small. Conversely, a short transfer time indicated that the constant thrust acceleration was too large. For ARTEMIS case 3, these iterations eventually resulted in an optimal thrust trajectory with a transfer time of 1,619,069 seconds. This is too long by 3385 seconds. However, the exact equation thrust planning software had difficulty finding cost function minima when the thrust acceleration was adjusted. This fact necessitated the use of this thrust plan even though the solved-for transfer time did not match the known

transfer time very precisely. The constant value of the thrust acceleration used for this trajectory was 4.4 x 10^{-9}. This is the same value solved for by the averaged equation software and was reused because the iterations using the exact equation software had difficulty finding a cost function minimum for other values of the thrust acceleration. 4.4 x 10^{-9} corresponds to a thrust force of 13.64 mN for a 3100 kg spacecraft. The exact equation solutions for the initial Lagrange multipliers for ARTEMIS case 3 are shown in Table 5.13.

Table 5.20 Initial Time Exact Lagrange Multipliers for ARTEMIS Case 3

Lagrange Multiplier	Solution Value
$\left(\lambda_a^L\right)_0$ (s/km)	-0.728573542528E+02
$\left(\lambda_h^L\right)_0$ (sec)	0.301891114262E+07
$\left(\lambda_k^L\right)_0$ (sec)	0.579883263045E+07
$\left(\lambda_p^L\right)_0$ (sec)	0.105599550006E+10
$\left(\lambda_q^L\right)_0$ (sec)	0.108651386979E+10
$\left(\lambda_L^L\right)_0$ (rad)	0.252160525941E+04

The Hamiltonian for the converged solution was equal to 1.0000705. This indicates a solution in which the necessary condition for optimality is met with a high degree of numerical precision. The final orbit achieved is shown in Table 5.18.

Figures 5.37-5.39 show selected orbital element histories and the pitch and yaw thrust plan over the transfer trajectory for ARTEMIS case 3. As for cases 1 and 2, the

thrust plan for test case 3 was generated with the exact equation thrust planning software using only two-body motion and thrust acceleration dynamics.

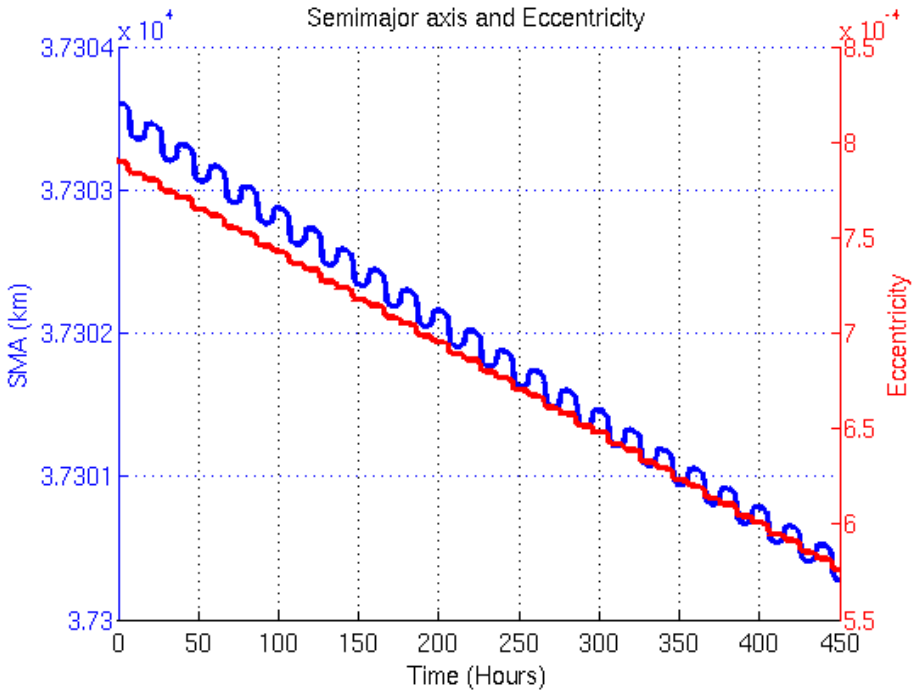

Figure 5.37 Semimajor axis and Eccentricity for ARTEMIS Optimal Thrust Plan Case 3

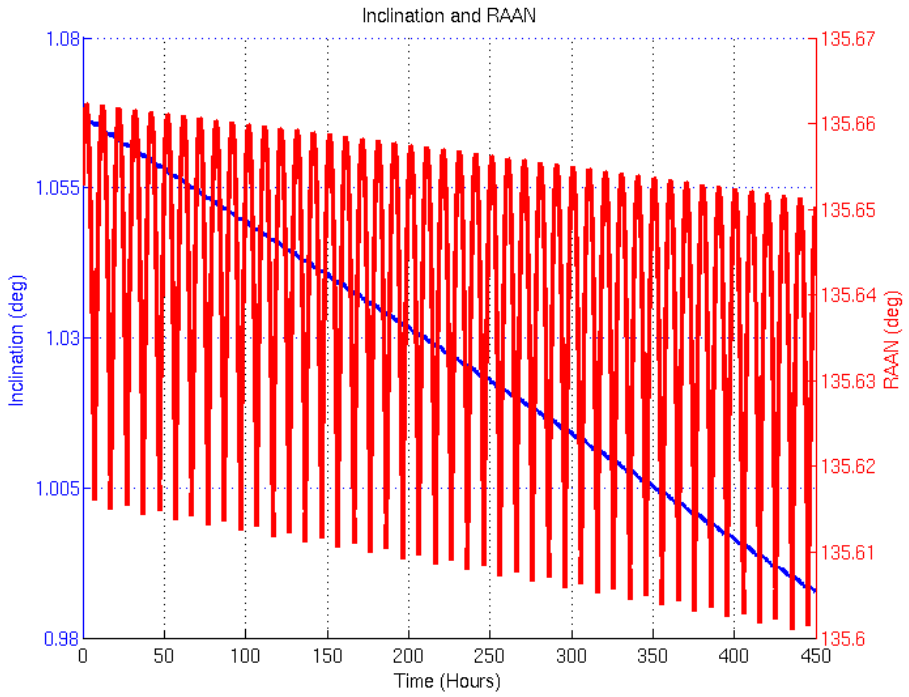

Figure 5.38 Inclination and RAAN for ARTEMIS Optimal Thrust Plan Case 3

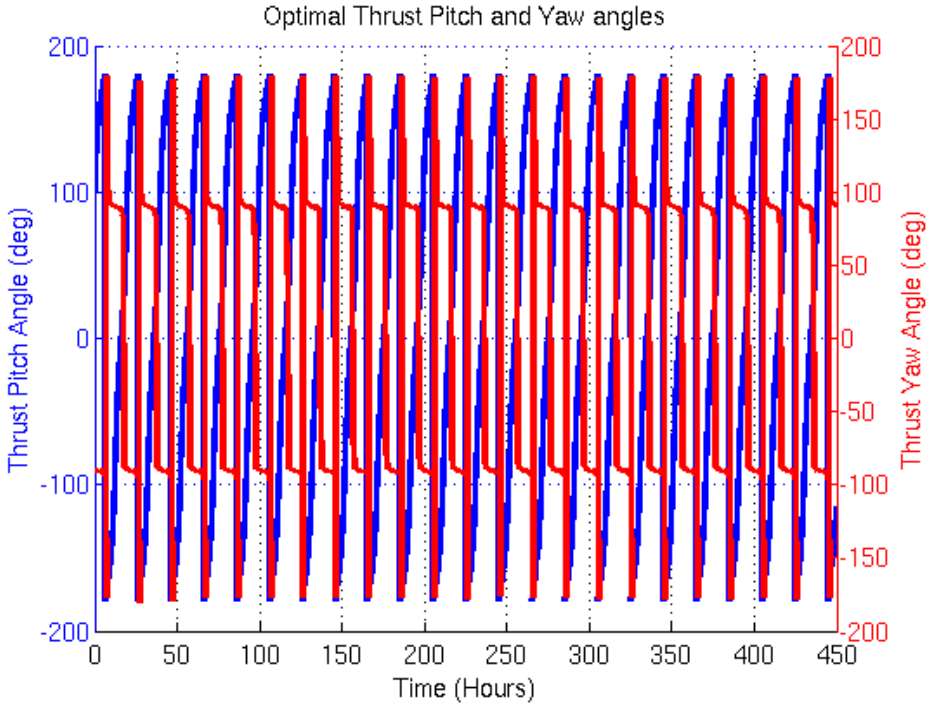

Figure 5.39 Pitch and Yaw Thrust Angles for Optimal Thrust Plan Case 3

Figure 5.37 shows that the semimajor axis decreases from the initial boundary condition to the final boundary condition. This decrease is not linear as in cases 1 and 2. Rather it has periodic behavior caused by the thrust plan. The eccentricity also decreases and follows an oscillatory path during the transfer from the initial to final boundary condition. The semimajor axis and eccentricity oscillations are due to the varying pitch angle of thrust during the transfer. Figure 5.38 shows the inclination and right ascension of the ascending node during the transfer plan. The right ascension trend is oscillatory while the inclination trend is linear. Figure 5.39 shows that the pitch and yaw angles both vary a great deal during each day of the thrust plan. The pitch angle rotates about 360 degrees in relation to the spacecraft each day. This ultimately achieves the semimajor axis and eccentricity changes required to get from the initial to the final orbit. The yaw angle pauses for several hours at 90 degrees and -90 degrees, i.e. normal to the orbital plane. This behavior is not surprising because during the *inclination control strategy,* a yaw angle of thrust that is normal to the orbit plane for several hours a day was described in reference (88). This yaw angle behavior in the thrust plan produced for ARTEMIS case 3 differs from the behavior in the plans created for ARTEMIS cases 1 and 2 because inclination control was used in the test case 3 interval. Inclination control was not attempted during the *back-up control strategy* according to reference (88).

As in ARTEMIS cases 1 and 2, inertial, osculating MEME of 1950 position/velocity vectors converted from TLEs were used as observations in GTDS Cowell differential corrections (DCs). AFSSN radar and optical observations were not applied in test case 3 because they were not available at the time of the writing of this

thesis. The GTDS DC program used an iterative Bayes' Least Squares estimator. The force models used with the Cowell propagator for these DCs were the same as for cases 1 and 2 and included 12x12 geopotential coefficients from the JGM-2 geopotential model, lunar and solar gravitational perturbations, solar radiation pressure and Earth polar motion. The solve-for vector was the Cartesian position and velocity in the MEME of 1950 reference frame. Starting from a reasonable a-priori orbital estimate, this nonlinear estimator should find the orbit that best fits a given set of observations in a least-squares sense. For test case 3, the DCs, i.e. fits, were done in order to evaluate whether optimal thrust plans provide any modeling improvement for the ARTEMIS satellite orbit during the time span from November 9th, 2001 to November 28th, 2001. As in cases 1 and 2, the TLE position/velocity vectors were applied in two GTDS differential correction (DC) runs. The first run for each set of observations did not use the optimal thrust plan generated by the exact equation optimal thrust planning software. The second run did use the optimal thrust plan. These GTDS DC runs were then compared in terms of the position/velocity residual statistics. The inertial, Cartesian position/velocity vector residuals, i.e. differences between observed and computed observations, for the TLE based observations in ARTEMIS case 3 are shown in Figures 5.40-5.42. The residuals obtained when the optimal thrust plan is used are shown as red crosses. The residuals obtained when no thrust plan model is used are shown as blue circles.

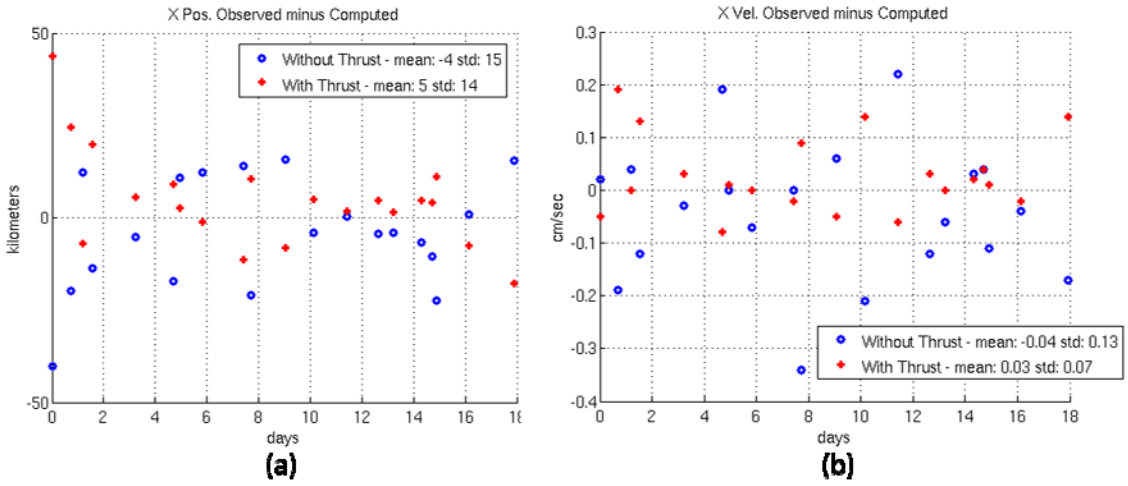

Figure 5.40 Case 3 Residuals of TLE based ECI osculating position (a) and velocity (b) vectors (X-direction)

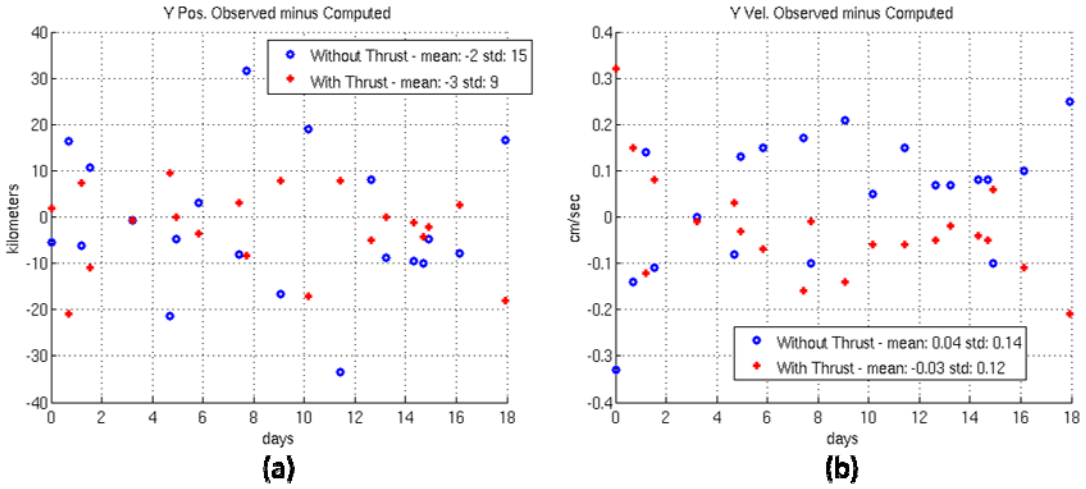

Figure 5.41 Case 3 Residuals of TLE based ECI osculating position (a) and velocity (b) vectors (Y-direction)

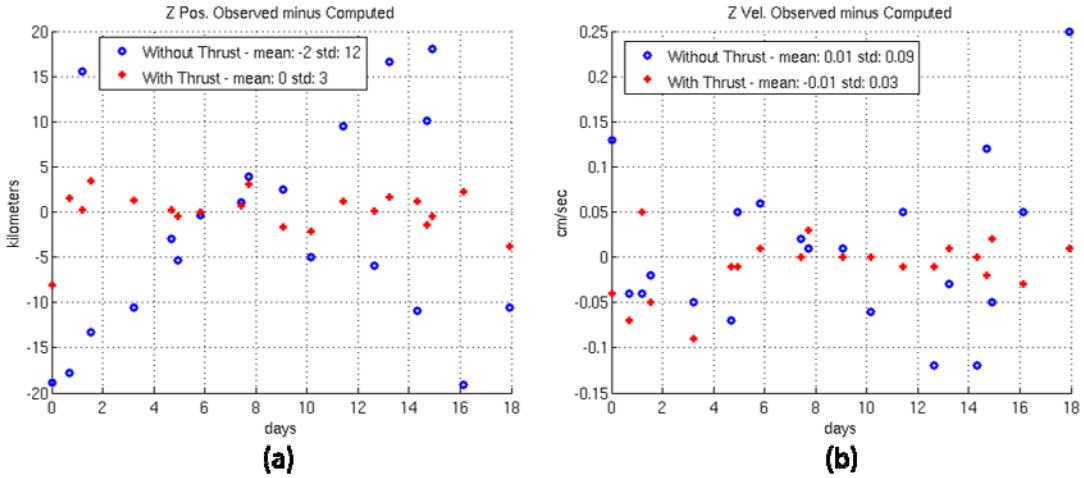

Figure 5.42 Case 3 Residuals of TLE based ECI osculating position (a) and velocity (b) vectors (Z-direction)

Figure 5.42 shows that the ECI, Cartesian position/velocity residuals in the Z direction are closer to zero-mean and have smaller variances when the thrust plan is integrated with the GTDS differential corrections versus when the plan is ignored. This is expected because the Z ECI directional axis most reflects satellite motion out of the orbital plane, and the optimal thrust plan was constructed to model the satellite's orbital plane change during the thrust transfer. The X direction and Y direction ECI residuals in Figures 5.40 and 5.41 show marginal improvement when applying the optimal thrust plan to orbit determination. This is perhaps because the optimal thrust plan did not perform the transfer within the known transfer time and did not model the transfer in discontinuous pieces as ARTEMIS actually performed the transfer. However, the improvement in residual statistics for the Z-axis clearly indicates that the optimal thrust plan improved the accuracy of the orbit modeling for ARTEMIS test case 3.

Overall, ARTEMIS test cases 1, 2 and 3 provide strong evidence that assuming an optimal thrust plan based on two-body dynamics can provide significant improvement in agreement with TLE and AFSSN observation data. Remaining modeling errors persist, however. These remaining modeling errors are apparent when examining AFSSN radar and optical measurement residuals. Future work to include J_2, lunar and solar gravity, and other significant perturbations in the optimal thrust planning software may allow for more accurate orbit determination. It is probable that the ARTEMIS operators used thrust plans that did consider J_2 and other perturbations. If that is the case, modeling these perturbations in the optimal thrust planning software should provide benefits for orbit determination and prediction of the ARTEMIS satellite during its orbit raising.

Iteration of the orbit determination and optimal thrust plan solutions should also be considered. Because the initial and final orbits used as input to the optimal thrust planning software are based on TLEs in these ARTEMIS test cases, they contain errors on the order of several kilometers. If an iteration can be executed whereby a DC orbit solution, i.e. initial and final orbits from the fitspan of the DC, can be used as input to the optimal thrust planning software, and new optimal thrust plans can be used in a subsequent DC, refinement of the satellite's trajectory could be made. This iterative process would essentially require an outer processing loop for the process depicted in Figure 5.15. However, it is unclear that such a super-iteration of this procedure would result in more accurate orbit determination without implementation of perturbations in the optimal thrust planning software.

Chapter 6 Conclusions and Future Work

6.1 Summary and Conclusions

The goals set out for this thesis included applying a modern filter/smoother to the problem of orbit determination for satellites that operate with continuous, low-thrust propulsion. The goal of applying modern filters and smoothers was set to determine whether estimation accuracy could be improved. Specifically, this thesis work was done in response to previous work in detecting satellite maneuvers (1). In this previous work, several detection algorithms for impulsive, chemical maneuvers were developed and evaluated. In the end, reference (1) presents an algorithm that uses an adaptive Extended Kalman Filter to detect maneuvers. The algorithm also attempts to verify the maneuvers and produce accurate post-maneuver orbital estimates using short-span differential corrections. The authors found that the adaptive Extended Semianalytic Kalman Filter (ESKF) alone provided insufficient accuracy immediately after satellite maneuvers occurred, and the ESKF required much time to recover accurate orbital state estimates.

For this thesis, a modern filter/smoother, the Backward Smoothing Extended Kalman Filter (BSEKF) (2), was chosen to attempt to remedy the accuracy issues found in reference (1) with the ESKF. The BSESKF was implemented in the Goddard Trajectory Determination System (GTDS) and coupled to the Draper Semianalytic Satellite Theory (DSST) to provide system dynamics. The BSESKF was tested with simulated observations generated from a nominal orbit for LEO and GEO test cases. For the LEO and GEO cases tested, the BSESKF provided more accurate orbital state

estimates than an existing ESKF implementation, and it required less time and fewer observations to obtain the accurate orbital state estimates. Increasing the number of observations over which the BSESKF filters and smoothes, i.e. the m-buffer, usually improved the accuracy and convergence of the BSESKF estimates. However, in some cases, increasing the m-buffer to sizes that include observations more than three integration time steps from the latest observation time reduced the accuracy of the BSESKF estimates. Preliminary testing indicates that with large m-buffer sizes, there is the potential to use interpolators outside of their valid ranges. Further software modifications could allow the interpolators to be reinitialized for past observations. Despite this software issue, the BSESKF estimates are a significant improvement over the ESKF estimates in the cases exercised.

Modeling continuous, low-thrust orbit transfers was also a goal for this thesis and significant progress has been made. Software tools to generate optimal thrust plans have been written and these plans have been used in GTDS for both orbit prediction and determination. The work of Jean Kechichian was particularly useful in developing these thrust planning tools. The ARTEMIS satellite orbit raising was used as a test case for determining how well orbit determination could be improved when optimal thrust plans were used to model continuous thrust orbit transfers. For the ARTEMIS case, optimal thrust plans were generated for three different time spans during the orbit raising. Orbit determination using GTDS was then performed for those spans. When the optimal thrust plan for the transfer was used to model the thrust acceleration, the orbit determination observation residuals varied much less than when the thrust acceleration was not

modeled. In some cases, this variance improved by an order of magnitude. The mean errors in the residuals were also much closer to zero when modeling spacecraft thrust with optimal thrust plans vs. no thrust plans. Particularly when using Air Force Space Surveillance Network (AFSSN) observations, neglecting spacecraft thrust didn't allow non-linear least squares differential orbit corrections to converge. When differential corrections included optimal thrust plan modeling, convergence was achieved. Results demonstrated in this thesis were achieved using position/velocity observations derived from Two Line Elements (TLEs) and using radar and optical observations from the AFSSN.

Although the observation residual behavior was significantly improved when using optimal thrust plans, the measurement residuals associated with AFSSN observations were still characterized by larger means and variances than were expected given the sensors and observation types used. This indicates that systematic orbit modeling errors were being projected into the observation residual space. These systematic errors are likely from a combination of sources. These error sources include: inaccuracy in the initial and final ARTEMIS orbital elements used to generate the optimal thrust plans, neglect of perturbative forces such as J_2 when calculating the optimal thrust plans, and the assumption that the ARTEMIS actually used an optimal thrust plan when executing its orbit raising. There also may have been times within the modeled spans during which the ARTEMIS thrusters were turned off. Such discontinuities would not have been modeled in the optimal, constant thrust plans generated with the thrust planning software.

6.2 Future Work

Because of the favorable results obtained in testing the BSESKF with simulated observations, future work should examine the behavior of the BSESKF when given real observations. This might first include applying it to real observation cases in which very accurate truth orbits could be used such as in the cases of satellites tracked by laser ranging stations or GPS satellites. If these tests prove successful, the BSEKF could be tested with sparser data or data from the Air Force Space Surveillance Network (AFSSN). Ultimately, the BSEKF could prove useful enough to replace the ESKF used for maneuver detection at MIT Lincoln Laboratory (1). This system uses the ESKF to detect off-nominal observation residuals and uses a series of subsequent tests to detect maneuvers in GEO satellites. It currently has to use short-span differential correction (DC) after maneuvers to provide accurate post-maneuver orbital estimates. Perhaps the BSESKF can replace the clumsier ESKF and DC approach.

Another application of the BSESKF could be estimating atmospheric density corrections. Others have worked on this problem and have applied other filter/smoothers to orbit determination and density correction estimation (92), (93). These references develop a Colored Noise Algorithm and an SVD decomposition estimation algorithm, respectively. It would be useful to compare the BSESKF to these estimators. It may also be worthwhile to use concepts from these estimators to improve convergence properties and treatments of measurement and process noise in the BSESKF. The sigma points smoother developed by Mark Psiaki (82) is an alternative to the BSEKF. It uses a

version of the UKF with a smoother and compares well with the BSEKF while being more computationally efficient (82).

The capabilities of the maneuver detection and modeling system depicted in Figure 1.1 are not yet fully realized with the tools that have developed for this thesis. Figure 6.1 shows helpful existing software, software developed for this thesis, and notional future software that would be useful in developing such a system.

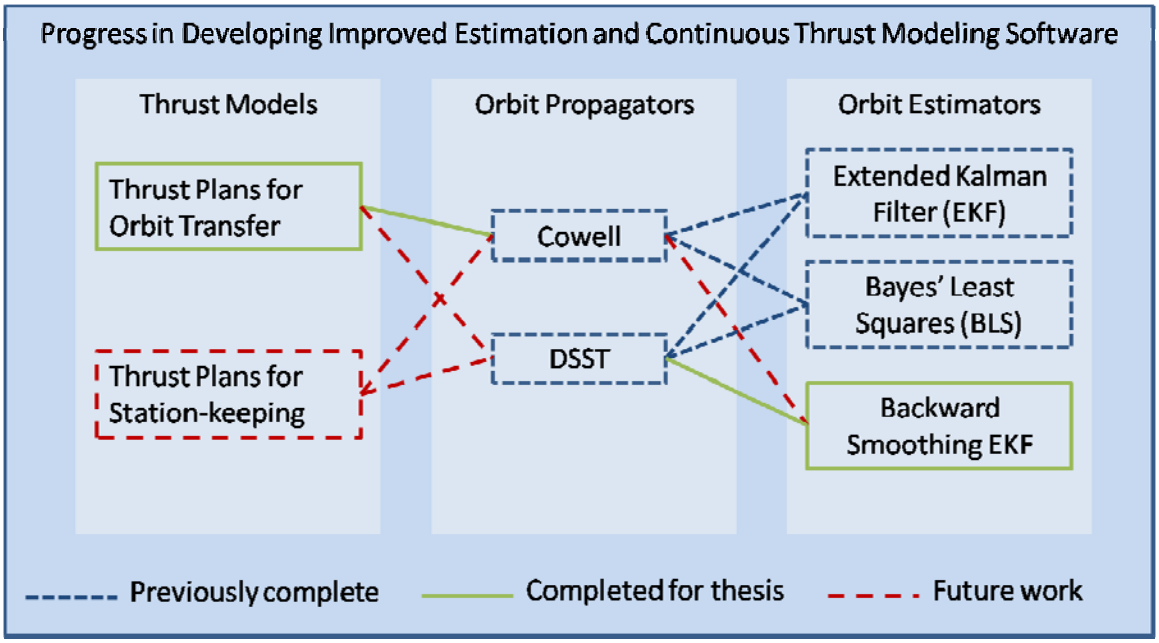

Figure 6.1 Progress in Modeling, Prediction and Estimation Tools for Improved Satellite Thrust Treatment

Although the thrust acceleration force model that has been implemented in GTDS has been successfully used with the Cowell orbit propagator, further software work must be done to use thrust plans with the DSST propagator. Some testing toward this end has been done, but further software issues remain. This implementation gap is depicted in Figure 6.1.

Although the BSEKF has also been coupled to the Cowell propagator in GTDS, testing currently indicates poorer convergence than with the existing GTDS EKF. The Cowell BSEKF also shows much poorer estimation performance than the DSST BSESKF. The software interface between the Cowell propagator and the BSEKF estimator requires more analysis and development for this capability to be realized. This gap in implentation is shown in Figure 6.1. Also, as discussed in section 4.4 of Chapter 4, allowing interpolator reinitialization in the BSESKF-DSST interface should improve BSESKF estimate accuracy with large m-buffer sizes.

The optimal thrust planning software would be made more useful with some enhancements. Modeling for Earth's J_2, J_3 and J_4 zonal harmonics should be included. Lunar and solar gravitational perturbations should also be included. References (7) and (8) would be good starting points for this effort. Perturbations such as these non-spherical harmonics and third-body effects alter long-duration thrust transfers significantly. Other useful enhancements could allow for bounded, variable thrust or bounded variable specific impulse. These enhancements might benefit efforts to model GEO satellite station keeping. The current thrust modeling software does a poor job of this because it assumes constant continuous thrust, whereas GEO satellites with EP operate thrusters discontinuously throughout each orbit. Capability to model station-keeping is future work as indicated in Figure 6.1.

In order to model thrust transfers that pass through critical inclination orbits, it would be important to include modeling for the odd zonal harmonics such as J_5, J_7 and J_9. Several satellite constellations that make use of critical inclination orbits are in use or have been proposed. It is probably only a matter of time before low-thrust EP satellites will occupy such orbits. Modeling optimal thrust trajectories for these satellites is likely to be worthwhile.

As more satellites with low-thrust EP are launched, the bookkeeping aspect of recording and retrieving the types of thrust trajectories these satellites use will become more important. It would be helpful to maintain a database of satellites with EP and thrust plans they have used. This is similar to the *launch folder* concept already used in space surveillance.

Another way to enhance the optimal thrust planning software would be to use a more robust minimization algorithm. Currently, the UNCMIN (58) quasi-Newton search algorithm used in the thrust planning software requires manual adjustment of weighting factors in the cost function. This adjustment requires a "human-in-the-loop." In communications with Jean Kechichian, it is clear that more convenient and robust minimization algorithms exist (94). One such algorithm is the DONLP2 algorithm written by Peter Spellucci (95). This algorithm performs constrained minimization and may be suitable for the trajectory optimization problems presented in this thesis.

In the process of implementing the exact and averaged optimal thrust planning software, a 7^{th} order Runge-Kutta-Fehlberg (RKF) integrator was found to provide the required accuracy while a 4^{th} order RKF integrator did not. Because a 4^{th} order RKF integrator is used in the GTDS DSST implementation, it is worth investigating whether a 7^{th} order RKF integrator would improve DSST predication accuracy.

Chapter 7 Appendices

Appendix A New GTDS Keywords

BSEKFSET Keyword

- Card Format (A8,3I3,3G21.14)
- Applicable programs: BSEKF subdeck BSEKFOPT
- Detailed Format:

Columns	Format	Description
1-8	A8	BSEKFSET – keyword to set options for running the BSEKF
9-11	I3	1 – Use the value in columns 18-38 as the m-buffer[6] size 0 – Use the default value for the m-buffer (24)
12-14	I3	1 – Use the value in columns 39-59 as the iteration tolerance[7] 0 – Use the default value for the iteration tolerance ($1x10^{-9}$)
15 – 17	I3	1 – Use the value in columns 60-80 to set the number of iterations allowed by the Gauss-Newton iteration[8] 0 – Use the default value for maximum iterations (20)
18-38	G21.14	If 1I3 is 1, this is the number of measurements to store in the m-buffer
39-59	G21.14	If 2I3 is 1, the iteration tolerance
60-80	G 12.14	If 3I3 is 1, the maximum number of iterations per measurement that are allowed.

[6] The m-buffer is the memory structure containing all measurements before the current one. The m-buffer is filtered and smoothed according to the BSEKF algorithm developed by Dr. Mark Psiaki at Cornell and implemented in GTDS. The m-buffer should be set large enough so that filtering/smoothing over the interval will allow nonlinear aspects in the system dynamics to be seen. A rule of thumb is to set the m-buffer to as many observations as it takes for an Extended Kalman Filter to converge.

[7] The iteration tolerance is the maximum difference between the previously calculated linearized cost function value and the current cost function value in order to determine convergence of the Gauss-Newton iterations within the BSEKF algorithm.

[8] The max number of iterations is set to prevent unnecessary iterations that result in insignificant gains in accuracy. These unnecessary iterations can significantly degrade performance of the BSEKF if this value is set too large. The default value is usually best.

[9]GTDS THRSTTBL Keyword

- Card Format (A8,3I3,2G21.14)
- Use with thrust acceleration vector input file
- Use in the OGOPT subdeck
- Detailed Format:

Columns	Format	Description
1-8	A8	THRSTTBL
9-11	I3	on/off switch 0 – Off 1 – On
12-17	n/a	blank (not used)
18-38	G21.14	Thrust vector start time (UTC) (YYYMMDDHHMISS.ssss) YYY – years from 1900 MM – month DD – day of month HH – hour MI – minute SS – second (integer part) .ssss – seconds (fractional part)
39-59	G21.14	Thrust vector stop time (UTC) (YYYMMDDHHMISS.ssss) YYY – years from 1900 MM – month DD – day of month HH – hour MI – minute SS – second (integer part) .ssss – seconds (fractional part)

[9] This card activates the thrust acceleration vector file input. It can be used to turn the feature on or off and it can be used to limit the times during which the thrust vector file is used. The first and second real fields specify the start and end times during which attempts to read the thrust acceleration file will be made.

[10]GTDS Thrust Acceleration Vector Input File (UNIT 116)

- Card Format (A8,3I3,5G21.14)
- Use with THRSTTBL keyword
- Detailed Format:

Columns	Format	Description
1-8	A8	Blank characters (not used)
9-11	I3	Coordinate frame 1 – Inertial Cartesian reference frame 2 – Body-fixed Cartesian ref. frame (not implemented)
12-17	n/a	blank (not used)
18-38	G21.14	Thrust vector time (UTC) (YYYMMDDHHMISS.ssss) YYY – years from 1900 MM – month DD – day of month HH – hour MI – minute SS – second (integer part) .ssss – seconds (fractional part)
39-59	G21.14	unit acceleration vector in X direction
60-80	G21.14	unit acceleration vector in Y direction
81-101	G21.14	unit acceleration vector in Z direction
102-122	G21.14	magnitude of acceleration vector in km/s^2

[10] The first line of the file must read "THRCARD" and the last line of the file must read "END"

[This page intentionally left blank]

Appendix B B^L Matrix and Partial Derivatives

The B^L matrix is used in equation (2.82) in section 2.2.2.2. It contains the elements that are multiplied by the thrust vector to calculate the partials of the equinoctial elements with respect to time. In addition, the partials of the B^L matrix with respect to the equinoctial elements are required to form the adjoint equations that are used to calculate the partials of the Lagrange multipliers with respect to time. Both sets of time derivatives are needed to integrate the satellite motion when it is taking an optimal thrust trajectory.

The B^L matrix is derived starting from the Gaussian VOP equations in Keplerian elements, $\{a, e, i, \Omega, \omega, M\}$. The VOP equations are then transformed to a set of equations in the equinoctial elements with the true longitude as the fast variable, $\{a, h, k, p, q, L\}$. To illustrate one of these transformations, a derivation of the variational equation for the semimajor axis is shown using equations (B1) through (B8). To begin, start with the Gaussian form of the semimajor axis VOP equation for two-body motion plus a perturbing force in terms of the Keplerian elements (11), (5):

$$\frac{da}{dt} = \dot{a} = \left(\frac{2a^2}{\acute{h}}\right)\left[e\sin(\theta^*)f_r + \left(\frac{\acute{p}}{r}\right)f_\theta\right] \tag{B1}$$

The f_r and f_θ directional indicators are the acceleration magnitude, f_t, times the perturbing force unit vector, $\hat{u}$, in the *r* and θ directions, i.e. $f_r = f_t u_r$ and $f_\theta = f_t u_\theta$.

The angular momentum magnitude, the orbital parameter, and the true anomaly are given by the following identities, respectively (5):

$$\acute{h} = \sqrt{[\mu a(1-e^2)]} = na^2\sqrt{(1-h^2-k^2)} \tag{B2}$$

$$\acute{p} = a(1-e^2) \tag{B3}$$

$$\theta^* = L - \omega - \Omega \tag{B4}$$

The variable L is the true longitude, ω is the argument of perigee, and Ω is the right ascension of the ascending node. The following identities for the radial distance from the central body to the orbiting body, r, the sine of the true anomaly, θ^*, and the quantity, $(\acute{p}/r)$, are used to write the variational equation for semimajor axis in terms of the equinoctial elements and true longitude (5):

$$r = \frac{a(1-h^2-k^2)}{(1+\mathrm{h}\sin L + k\cos L)} \tag{B5}$$

$$\sin(\theta^*) = \frac{(k\sin L - h\cos L)}{\sqrt{(h^2+k^2)}} \tag{B6}$$

$$\frac{\acute{p}}{r} = (1 + h\sin L + k\cos L) \tag{B7}$$

The variational equation of the semimajor axis can be ultimately written by substituting equations (B2), (B5), (B6), and (B7) into equation (B1) to obtain (5):

$$\dot{a} = \frac{2}{n\sqrt{(1-h^2-k^2)}}\{[k\sin L - h\cos L]f_r + [1 + h\sin L + k\cos L]f_\theta\} \qquad \text{(B8)}$$

Using equation (2.82a), the equation for r in equation (B5), and the definition for G in equation (B9), one can see that $B_{11}^L = 2n^{-1}G^{-1}(ks_L - hc_L)$ and $B_{12}^L = 2n^{-1}ar^{-1}G$. The partials of B^L with respect to the equinoctial elements $\{a, h, k, p, q, L\}$ can then be derived in a straightforward manner. For more complete details, reference (5) provides all of the identities required.

The complete set of B^L matrix elements and partial derivatives are all taken from reference (5). In personal communication with Jean Kechichian (86), some typos were corrected and the corrections are reflected in this appendix. Specifically, equations (B81) and (B90) reflect these corrections.

$$G = \sqrt{1 - h^2 - k^2} \qquad \text{(B9)}$$

$$K = 1 + p^2 + q^2 \qquad \text{(B10)}$$

$$\frac{\partial n}{\partial a} = \frac{-3n}{2a} \qquad \text{(B11)}$$

$$\frac{\partial r}{\partial a} = \frac{r}{a} \qquad \text{(B12)}$$

$$\frac{\partial r}{\partial h} = -\frac{r}{a(1-h^2-k^2)}(2ah + rs_L) \qquad \text{(B13)}$$

$$\frac{\partial r}{\partial k} = -\frac{r}{a\left(1-h^2-k^2\right)}(2ah + rc_L) \tag{B14}$$

$$\frac{\partial r}{\partial L} = -\frac{r^2(hc_L - ks_L)}{a(1-h^2-k^2)} \tag{B15}$$

$$B_{11}^L = 2n^{-1}G^{-1}(ks_L - hc_L) \tag{B16}$$

$$\frac{\partial B_{11}^L}{\partial a} = -2n^{-2}\frac{\partial n}{\partial a}G^{-1}(ks_L - hc_L) \tag{B17}$$

$$\frac{\partial B_{11}^L}{\partial h} = 2n^{-1}hG^{-3}(ks_L - hc_L) - 2n^{-1}G^{-1}c_L \tag{B18}$$

$$\frac{\partial B_{11}^L}{\partial k} = 2n^{-1}kG^{-3}(ks_L - hc_L) + 2n^{-1}G^{-1}s_L \tag{B19}$$

$$\frac{\partial B_{11}^L}{\partial p} = \frac{\partial B_{11}^L}{\partial q} = 0 \tag{B20}$$

$$\frac{\partial B_{11}^L}{\partial L} = 2n^{-1}G^{-1}(kc_L + hs_L) \tag{B21}$$

$$B_{12}^L = 2n^{-1}ar^{-1}G \tag{B22}$$

$$\frac{\partial B_{12}^L}{\partial a} = -2n^{-2}ar^{-1}\frac{\partial n}{\partial a}G \tag{B23}$$

$$\frac{\partial B_{12}^L}{\partial h} = -2n^{-1}ar^{-2}\frac{\partial r}{\partial h}G - 2n^{-1}ar^{-1}hG^{-1} \tag{B24}$$

$$\frac{\partial B_{12}^L}{\partial k} = -2n^{-1}ar^{-2}\frac{\partial r}{\partial k}G - 2n^{-1}ar^{-1}kG^{-1} \tag{B25}$$

$$\frac{\partial B_{12}^L}{\partial p} = \frac{\partial B_{12}^L}{\partial q} = 0 \tag{B26}$$

$$\frac{\partial B_{12}^L}{\partial L} = -2n^{-1}ar^{-2}G\frac{\partial r}{\partial L} \tag{B27}$$

$$B_{13}^{L} = \frac{\partial B_{13}^{L}}{\partial a} = \frac{\partial B_{13}^{L}}{\partial h} = \frac{\partial B_{13}^{L}}{\partial k} = \frac{\partial B_{13}^{L}}{\partial p} = \frac{\partial B_{13}^{L}}{\partial q} = \frac{\partial B_{13}^{L}}{\partial L} = 0 \tag{B28}$$

$$B_{21}^{L} = -n^{-1}a^{-1}Gc_{L} \tag{B29}$$

$$\frac{\partial B_{21}^{L}}{\partial a} = -2^{-1}n^{-1}a^{-2}Gc_{L} \tag{B30}$$

$$\frac{\partial B_{21}^{L}}{\partial h} = n^{-1}a^{-1}hG^{-1}c_{L} \tag{B31}$$

$$\frac{\partial B_{21}^{L}}{\partial k} = n^{-1}a^{-1}kG^{-1}c_{L} \tag{B32}$$

$$\frac{\partial B_{21}^{L}}{\partial p} = \frac{\partial B_{21}^{L}}{\partial q} = 0 \tag{B33}$$

$$\frac{\partial B_{21}^{L}}{\partial L} = n^{-1}a^{-1}Gs_{L} \tag{B34}$$

$$B_{22}^{L} = n^{-1}a^{-2}rG^{-1}(h + s_{L}) + n^{-1}a^{-1}Gs_{L} \tag{B35}$$

$$\frac{\partial B_{22}^{L}}{\partial a} = 2^{-1}n^{-1}a^{-3}rG^{-1}(h + s_{L}) + 2^{-1}n^{-1}a^{-2}Gs_{L} \tag{B36}$$

$$\frac{\partial B_{22}^{L}}{\partial h} = n^{-1}a^{-2}G^{-1}(h + s_{L})\left(\frac{\partial r}{\partial h} + rhG^{-2}\right) + n^{-1}a^{-2}rG^{-1} - n^{-1}a^{-1}hs_{L}G^{-1} \tag{B37}$$

$$\frac{\partial B_{22}^{L}}{\partial k} = n^{-1}a^{-2}G^{-1}(h + s_{L})\left(\frac{\partial r}{\partial k} + rkG^{-2}\right) - n^{-1}a^{-1}ks_{L}G^{-1} \tag{B38}$$

$$\frac{B_{22}^{L}}{\partial p} = \frac{B_{22}^{L}}{\partial q} = 0 \tag{B39}$$

$$\frac{B_{22}^{L}}{\partial L} = n^{-1}a^{-2}(h + s_{L})G^{-1}\frac{\partial r}{\partial L} + n^{-1}a^{-2}rc_{L}G^{-1} + n^{-1}a^{-1}c_{L}G \tag{B40}$$

$$B_{23}^{L} = -n^{-1}a^{-2}rG^{-1}k(pc_{L} - qs_{L}) \tag{B41}$$

$$\frac{\partial B_{23}^{L}}{\partial a} = -2^{-1} n^{-1} a^{-3} r G^{-1} k (pc_L - qs_L) \tag{B42}$$

$$\frac{\partial B_{23}^{L}}{\partial h} = -n^{-1} a^{-2} G^{-1} k (pc_L - qs_L) \left(\frac{\partial r}{\partial h} + hrG^{-2} \right) \tag{B43}$$

$$\frac{\partial B_{23}^{L}}{\partial k} = -n^{-1} a^{-2} G^{-1} k (pc_L - qs_L) \left(\frac{\partial r}{\partial k} + krG^{-2} \right) - n^{-1} a^{-2} r G^{-1} (pc_L - qs_L) \tag{B44}$$

$$\frac{\partial B_{23}^{L}}{\partial p} = -n^{-1} a^{-2} r G^{-1} k c_L \tag{B45}$$

$$\frac{\partial B_{23}^{L}}{\partial q} = n^{-1} a^{-2} r G^{-1} k s_L \tag{B46}$$

$$\frac{\partial B_{23}^{L}}{\partial L} = -n^{-1} a^{-2} G^{-1} k (pc_L - qs_L) \frac{\partial r}{\partial L} + n^{-1} a^{-2} r G^{-1} k (ps_L + qc_L) \tag{B47}$$

$$B_{31}^{L} = n^{-1} a^{-1} G s_L \tag{B48}$$

$$\frac{\partial B_{31}^{L}}{\partial a} = 2^{-1} n^{-1} a^{-2} G s_L \tag{B49}$$

$$\frac{\partial B_{31}^{L}}{\partial h} = -n^{-1} a^{-1} G^{-1} h s_L \tag{B50}$$

$$\frac{\partial B_{31}^{L}}{\partial k} = -n^{-1} a^{-1} G^{-1} k s_L \tag{B51}$$

$$\frac{B_{31}^{L}}{\partial p} = \frac{B_{31}^{L}}{\partial q} = 0 \tag{B52}$$

$$\frac{\partial B_{31}^{L}}{\partial L} = n^{-1} a^{-1} G c_L \tag{B53}$$

$$B_{32}^{L} = n^{-1} a^{-2} r G^{-1} (k + c_L) + n^{-1} a^{-1} G c_L \tag{B54}$$

$$\frac{\partial B_{32}^{L}}{\partial a} = 2^{-1} n^{-1} a^{-3} r G^{-1} (k + c_L) + 2^{-1} n^{-1} a^{-2} G c_L \tag{B55}$$

$$\frac{\partial B_{32}^{L}}{\partial h} = n^{-1}a^{-2}G^{-1}(k+c_L)\left(\frac{\partial r}{\partial h} + hrG^{-2}\right) - n^{-1}a^{-1}hG^{-1}c_L \tag{B56}$$

$$\frac{\partial B_{32}^{L}}{\partial k} = n^{-1}a^{-2}G^{-1}(k+c_L)\left(\frac{\partial r}{\partial k} + krG^{-2}\right) - n^{-1}a^{-1}kG^{-1}c_L + n^{-1}a^{-2}rG^{-1} \tag{B57}$$

$$\frac{\partial B_{32}^{L}}{\partial p} = \frac{\partial B_{32}^{L}}{\partial q} = 0 \tag{B58}$$

$$\frac{\partial B_{32}^{L}}{\partial L} = n^{-1}a^{-2}G^{-1}(k+c_L)\frac{\partial r}{\partial L} - n^{-1}a^{-2}rG^{-1}s_L - n^{-1}a^{-1}Gs_L \tag{B59}$$

$$B_{33}^{L} = n^{-1}a^{-2}rG^{-1}h(pc_L - qs_L) \tag{B60}$$

$$\frac{\partial B_{33}^{L}}{\partial a} = 2^{-1}n^{-1}a^{-3}rG^{-1}h(pc_L - qs_L) \tag{B61}$$

$$\frac{\partial B_{33}^{L}}{\partial h} = n^{-1}a^{-2}G^{-1}h(pc_L - qs_L)\left(\frac{\partial r}{\partial h} + hrG^{-2}\right) + n^{-1}a^{-2}rG^{-1}(pc_L - qs_L) \tag{B62}$$

$$\frac{\partial B_{33}^{L}}{\partial k} = n^{-1}a^{-2}G^{-1}h(pc_L - qs_L)\left(\frac{\partial r}{\partial k} + krG^{-2}\right) \tag{B63}$$

$$\frac{\partial B_{33}^{L}}{\partial p} = n^{-1}a^{-2}rG^{-1}hc_L \tag{B64}$$

$$\frac{\partial B_{33}^{L}}{\partial q} = -n^{-1}a^{-2}rG^{-1}hs_L \tag{B65}$$

$$\frac{\partial B_{33}^{L}}{\partial L} = n^{-1}a^{-2}G^{-1}h(pc_L - qs_L)\frac{\partial r}{\partial L} - n^{-1}a^{-2}rG^{-1}h(ps_L - qc_L) \tag{B66}$$

$$B_{43}^{L} = 2^{-1}n^{-1}a^{-2}rG^{-1}Ks_L \tag{B67}$$

$$\frac{\partial B_{43}^{L}}{\partial a} = 4^{-1}n^{-1}a^{-3}rG^{-1}Ks_L \tag{B68}$$

$$\frac{\partial B_{43}^{L}}{\partial h} = 2^{-1}n^{-1}a^{-2}G^{-1}Ks_L\left(\frac{\partial r}{\partial h} + hrG^{-2}\right) \tag{B69}$$

$$\frac{\partial B_{43}^{L}}{\partial k} = 2^{-1} n^{-1} a^{-2} G^{-1} Ks_{L} \left(\frac{\partial r}{\partial k} + krG^{-2} \right) \tag{B70}$$

$$\frac{\partial B_{43}^{L}}{\partial p} = n^{-1} a^{-2} rG^{-1} ps_{L} \tag{B71}$$

$$\frac{\partial B_{43}^{L}}{\partial q} = n^{-1} a^{-2} rG^{-1} qs_{L} \tag{B72}$$

$$\frac{\partial B_{43}^{L}}{\partial L} = 2^{-1} n^{-1} a^{-2} G^{-1} Ks_{L} \frac{\partial r}{\partial L} + 2^{-1} n^{-1} a^{-2} rG^{-1} Kc_{L} \tag{B73}$$

$$B_{53}^{L} = 2^{-1} n^{-1} a^{-2} rG^{-1} Kc_{L} \tag{B74}$$

$$\frac{\partial B_{53}^{L}}{\partial a} = 4^{-1} n^{-1} a^{-3} rG^{-1} Kc_{L} \tag{B75}$$

$$\frac{\partial B_{53}^{L}}{\partial h} = 2^{-1} n^{-1} a^{-2} G^{-1} Kc_{L} \left(\frac{\partial r}{\partial h} + hrG^{-2} \right) \tag{B76}$$

$$\frac{\partial B_{53}^{L}}{\partial k} = 2^{-1} n^{-1} a^{-2} G^{-1} Kc_{L} \left(\frac{\partial r}{\partial k} + krG^{-2} \right) \tag{B77}$$

$$\frac{\partial B_{53}^{L}}{\partial p} = n^{-1} a^{-2} rG^{-1} pc_{L} \tag{B78}$$

$$\frac{\partial B_{53}^{L}}{\partial q} = n^{-1} a^{-2} rG^{-1} qc_{L} \tag{B79}$$

$$\frac{\partial B_{53}^{L}}{\partial L} = 2^{-1} n^{-1} a^{-2} G^{-1} Kc_{L} \frac{\partial r}{\partial L} - 2^{-1} n^{-1} a^{-2} rG^{-1} Ks_{L} \tag{B80}$$

$$B_{63}^{L} = n^{-1} a^{-2} rG^{-1} (qs_{L} - pc_{L}) \tag{B81}$$

$$\frac{\partial B_{63}^{L}}{\partial a} = 2^{-1} n^{-1} a^{-3} rG^{-1} (qs_{L} - pc_{L}) \tag{B82}$$

$$\frac{\partial B_{63}^{L}}{\partial h} = n^{-1} a^{-2} (qs_{L} - pc_{L}) G^{-1} \left(\frac{\partial r}{\partial h} + rhG^{-2} \right) \tag{B83}$$

$$\frac{\partial B_{63}^{L}}{\partial k} = n^{-1}a^{-2}(qs_L - pc_L)G^{-1}\left(\frac{\partial r}{\partial k} + rkG^{-2}\right) \tag{B84}$$

$$\frac{\partial B_{63}^{L}}{\partial p} = -n^{-1}a^{-2}rG^{-1}c_L \tag{B85}$$

$$\frac{\partial B_{63}^{L}}{\partial p} = n^{-1}a^{-2}rG^{-1}s_L \tag{B86}$$

$$\frac{\partial B_{63}^{L}}{\partial L} = n^{-1}a^{-2}(qs_L - pc_L)G^{-1}\frac{\partial r}{\partial L} + n^{-1}a^{-2}r(qc_L + ps_L)G^{-1} \tag{B87}$$

This concludes the listing of the B^L matrix and its partials with respect to the equinoctial orbital elements. The following equations are the partials of the last term in equation (2.82a) with respect to the equinoctial orbital elements.

$$\frac{\partial}{\partial a}\left[\frac{na^2(1-h^2-k^2)^{\frac{1}{2}}}{r^2}\right] = -\frac{3}{2}nar^{-2}G \tag{B88}$$

$$\frac{\partial}{\partial h}\left[\frac{na^2(1-h^2-k^2)^{\frac{1}{2}}}{r^2}\right] = -2na^2r^{-3}G\frac{\partial r}{\partial h} - na^2r^{-2}hG^{-1} \tag{B89}$$

$$\frac{\partial}{\partial k}\left[\frac{na^2(1-h^2-k^2)^{\frac{1}{2}}}{r^2}\right] = -2na^2r^{-3}G\frac{\partial r}{\partial k} - na^2r^{-2}kG^{-1} \tag{B90}$$

$$\frac{\partial}{\partial L}\left[\frac{na^2(1-h^2-k^2)^{\frac{1}{2}}}{r^2}\right] = -2na^2r^{-3}G\frac{\partial r}{\partial L} \tag{B91}$$

[This page intentionally left blank]

Appendix C Thrust Plan Coordinate Systems

When defining the thrust acceleration plan file, it was necessary to evaluate which reference frame would be appropriate. The GTDS force model framework provides the satellite position and velocity in Cartesian coordinates. The reference frame of the coordinates is also the reference frame in which GTDS integrates and can be one of several user-selected reference frames. These reference frames include MEME J2000, MEME B1950, and TETE True of Date. Because, for now, the thrust plan file only represents an optimal trajectory from the perspective of two-body motion, subtleties in the reference frame do not come into play when applying a two-body thrust plan to orbit prediction within the GTDS framework. However, if Earth J2 and third-body modeling are incorporated into the optimal thrust plan standalone code, the thrust plan file will have to be carefully synchronized with the reference frame used in GTDS for subsequent application of the thrust plan for orbit prediction and orbit determination.

The coordinate system of the thrust acceleration vectors produced by the trajectory optimization code in Section 2.2.2.2, i.e. $\hat{\mathbf{u}} = \{u_r, u_\theta, u_h\}$, is the rotating Euler-Hill polar frame. Because of the relative ease of transforming thrust acceleration vectors represented in Euler-Hill rotating polar coordinates (6) to Cartesian position and velocity coordinates in an inertial frame, it was decided that the exact equation standalone trajectory optimization code should transform its thrust vector from rotating polar coordinates to an inertial frame compatible with the GTDS force model framework.

The Euler-Hill rotating polar coordinate frame is defined in terms of the *r, θ,* and *h* axes. The *r*-axis is a unit vector in the direction of the central body to the satellite. The *θ* axis is a unit vector perpendicular to the *r*-axis and in the orbit plane. The *h*-axis is formed by taking the cross product of the *r* and *θ* axes and so maintains an orthogonal right handed coordinate system. The *h*-axis is also coincident with the satellite's orbital angular momentum vector. The yaw and pitch angles used in attitude dynamics can be easily computed from the unit thrust vector in terms of these polar coordinates (6).

$$\theta_{pitch} = \tan^{-1}\left(\frac{u_r}{u_\theta}\right) \tag{C1}$$

$$\theta_{yaw} = \tan^{-1}\left(\frac{u_h}{u_\theta}\right) \tag{C2}$$

Figure C.1 depicts the pitch (C1) and yaw (C2) angles with respect to the satellite.

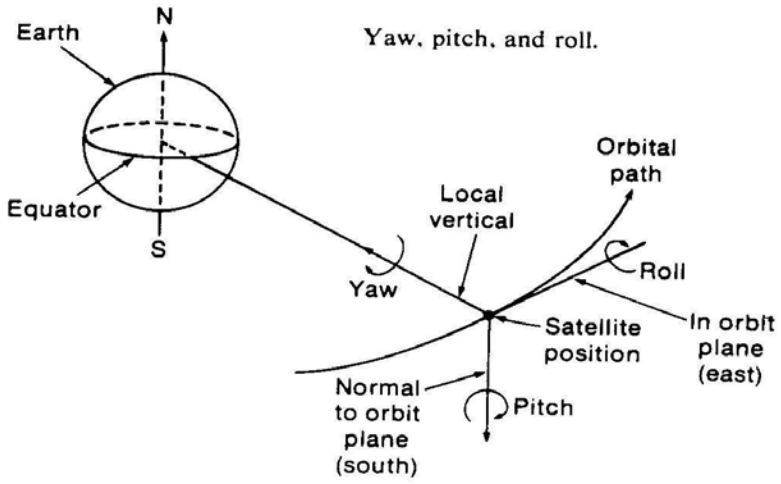

Figure C.1 Satellite Thrust Pitch and Yaw Angles (96)

Transforming the rotating polar coordinates to the equinoctial frame involves the following transformation matrix in terms of the equinoctial f, g, w and polar r, θ, h unit vectors (6):

$$\begin{pmatrix} \hat{f} \\ \hat{g} \\ \hat{w} \end{pmatrix} = \begin{pmatrix} X_1 r & -Y_1 r & 0 \\ Y_1 r & X_1 r & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \hat{r} \\ \hat{\theta} \\ \hat{h} \end{pmatrix} \tag{C3}$$

This transformation matrix will convert a vector represented in the Euler-Hill rotating polar frame to a vector in the equinoctial coordinate frame. The quantity r is the scalar distance from the central body to the satellite and the X_1 and Y_1 scalar quantities are defined by (6):

$$\mathbf{r} = X_1 \hat{f} + Y_1 \hat{g} \tag{C4}$$

$$X_1 = r\cos L = a\left[(1-h^2\beta)\cos F + hk\beta \sin F - k\right] \tag{C5}$$

$$Y_1 = r\sin L = a\left[hk\beta\cos F + (1-k^2\beta)\sin F - h\right] \tag{C6}$$

Here, L is the true longitude, and F is the eccentric longitude. The eccentric longitude can be found using the mean longitude, λ, from the satellite's equinoctial element set, $\mathbf{z} = \{a, h, k, p, q, \lambda\}$. Using Kepler's equation written in terms of eccentric longitude, F, we have (6):

$$\lambda = F - k\sin F + h\cos F \tag{C7}$$

Equation (C7) can be iterated with a starting guess of F to solve for F accurately. The β quantity in equations (C5) and (C6) is also defined in terms of the equinoctial elements (6):

$$\beta = \frac{1}{1+\sqrt{(1-h^2-k^2)}} \tag{C8}$$

With the thrust vector now transformed to the equinoctial frame, i.e. $\hat{\mathbf{u}} = \{u_f, u_g, u_w\}$, one can use a well known transformation to transform the thrust vector to the inertial, Cartesian, x, y, z, frame. First, compute the inclination, I, and right ascension of the ascending node, Ω, from the known equinoctial elements:

$$i = 2\tan^{-1}\left(\sqrt{p^2+q^2}\right) \tag{C9}$$

$$\Omega = \tan^{-1}\left(p/q\right) \tag{C10}$$

The following transformation then takes a vector in the equinoctial frame to one in the inertial, Cartesian frame (14):

$$\begin{pmatrix}\hat{x}\\ \hat{y}\\ \hat{z}\end{pmatrix} = \begin{pmatrix}\cos^2(\Omega)+I\cos(i)\sin^2(\Omega) & \cos(\Omega)\sin(-\Omega)\{1-I\cos(i)\} & -I\sin(i)\sin(\Omega)\\ I\cos(\Omega)\sin(-\Omega)\{1-I\cos(-i)\} & I\sin^2(\Omega)+\cos(i)\cos^2(\Omega) & \sin(-i)\cos(\Omega)\\ \sin(i)\sin(\Omega) & \sin(i)\cos(\Omega) & \cos(i)\end{pmatrix}\begin{pmatrix}\hat{f}\\ \hat{g}\\ \hat{w}\end{pmatrix} \tag{C11}$$

This transformation is used in the TRANS_OUT subroutine in the exact equation trajectory optimization software to transform the thrust acceleration vector before writing it to the thrust plan file for GTDS.

[This page intentionally left blank]

Appendix D Executing the Optimal Thrust Planning Software

Executing both the averaged equation and exact equation optimal thrust planning software requires a FORTRAN compiler. For this thesis, the Intel FORTRAN compiler® version 9.1 was used. The thrust planning software does not yet allow for runtime input. Instead, the initial orbit, final orbit, thrust acceleration magnitude and the guesses for the final time and for the initial 6x1 vector of Lagrange multipliers are hard-coded in the low_thrust_drive.for file. Any modification of the input parameters requires changing the low_thrust_drive.for file and recompiling the source code to obtain a new executable file. Eventually, it is the intent of the author to incorporate the thrust planning software within the GTDS framework as a subprogram. In this way, the GTDS input keyword processor can be used to provide input values to the optimal thrust planning algorithm. Figure D.1 shows the lines of source code in the low_thrust_drive.for source code file that can be modified to calculate thrust plans for alternative cases. The exact equation and averaged equation thrust planning software each include a separate version of low_thrust_drive.for. The set of input in Figure D.1 was used to execute ARTEMIS case #1.

```
C
C     Guess for the final time (seconds)
C
      tf0 = 853716.998592
C
C     Set the constant acceleration (km/s^2)
C
      ft = 7.5628E-9
C
C     Set the initial Keplerian elements
C
```

```
      sma0 = 39382.97217
      ecc0 = 0.2006846493E-2
      inc0 = 1.43568 * pi/180.0
      ran0 = 115.95324 * pi/180.0
      arp0 = 297.5172826 * pi/180.0
      mea0 = 211.6003815 * pi/180.0
C
C     Set the final Keplerian elements
C
      smaF = 39537.70766
      eccF = 0.1621537776E-2
      incF = 1.43568 * pi/180
      ranF = 115.95324 * pi/180
      arpF = 297.5172826 * pi/180
      meaF = 212.7786305 * pi/180
C
C     Set the initial guesses for the Lagrange multipliers
C
      lam_vect(1) = 0.326239885662E+04
      lam_vect(2) = -0.671795199693E+08
      lam_vect(3) = -0.478598052526E+08
      lam_vect(4) = 0.158973787328E+05
      lam_vect(5) = -0.129598180966E+05
      lam_vect(6) = 0.583925385401E+03
```

Figure D.1 Source Code Input for ARTEMIS Case #1

The following initialization in Figure D.2 shows how the weights array was set for the ARTEMIS case #1. Often, changing the weights is necessary in order for the UNCMND algorithm to obtain a solution that closely matches the final orbit conditions.

```
C
C     Set the weights for each outer loop optimization
C     iteration
C
      weights(1,1) = 1.0E3
      weights(2,1) = 1.0E12
      weights(3,1) = 1.0E12
      weights(4,1) = 1.0E11
      weights(5,1) = 1.0E11
```

```
weights(6,1) = 1.0
weights(7,1) = 1.0E6

weights(1,2) = 1.0E4
weights(2,2) = 1.0E13
weights(3,2) = 1.0E13
weights(4,2) = 1.0E12
weights(5,2) = 1.0E12
weights(6,2) = 1.0
weights(7,2) = 1.0E6

weights(1,3) = 1.0E3
weights(2,3) = 1.0E12
weights(3,3) = 1.0E12
weights(4,3) = 1.0E13
weights(5,3) = 1.0E13
weights(6,3) = 1.0
weights(7,3) = 1.0E6

weights(1,4) = 1.0E4
weights(2,4) = 1.0E13
weights(3,4) = 1.0E13
weights(4,4) = 1.0E14
weights(5,4) = 1.0E14
weights(6,4) = 1.0
weights(7,4) = 1.0E6

weights(1,5) = 1.0E4
weights(2,5) = 1.0E12
weights(3,5) = 1.0E12
weights(4,5) = 1.0E14
weights(5,5) = 1.0E14
weights(6,5) = 1.0
weights(7,5) = 1.0E6
```

Figure D.2 Source Code Weight Input for ARTEMIS Case #1

[This page intentionally left blank]

Appendix E Source code for the Exact Equation Optimal Thrust Planning Software

The exact equation optimal thrust planning software is described in Chapter 5 section 5.1.1. This appendix contains the source code corresponding to section 5.1.1. Only the source code written by the author is included. Other open source subroutines such as the UNCMND, DQAG, and RK78 subroutines are not included. Sources for those subroutines can be found in the References section or by contacting the author. Some of the subroutines in this appendix have the same names as subroutines in appendix F. The subroutines with identical names are different for the exact equation code than they are for the averaged equation code. Each set of software is in a separate directory space.

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: low_thrust_drive.for  (for exact equation software)
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
C.....................................................................
C. ROUTINE: LOW_THRUST_DRIVE
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: this the driver subroutine for the exact eq. software.
C.    It collects the initial and final Keplerian orbits,
C.    converts those to equinoctial orbits,
C.    calls the UNCMND subroutine to execute the quasi-Newton
C.    search to solve for the initial
C.    Lagrange multipliers.  Once UNCMND
C.    is complete, the RK78 subroutine is used to integrate the
C.    variational equations of motion and
C.    the variational equations for the solved initial Lagrange
C.    multipliers from the initial to the
```

```
C.    final time.  Finally, the trajectory is printed and the
C.    thrust plan file meant for GTDS input is written.
C.
C.
C. CALLING SEQUENCE:
C.          This is a main program and has no calling parameters.  However,
C.          Several of the initial variable values can be modified to solve
C.          optimal thrust trajectory problems.  Among these are:
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER   I/O                   DESCRIPTION
C.          ------  ---  ------------------------------------------
C.          tf0     I    the guess for the final time in seconds.
C.          ft      I    the constrant thrust acceleration km/second squared
C.          sma0    I    the semimajor axis for the initial orbit
C.          ecc0    I    the eccentricity of the initial orbit
C.          inc0    I    the inclination of the initial orbit
C.          ran0    I    the RAAN of the initial orbit
C.          arp0    I    the arg. of perigee of the initial orbit
C.          mea0    I    the mean anomaly of the initial orbit
C.          smaF    I    the semimajor axis for the final orbit
C.          eccF    I    the eccentricity of the final orbit
C.          incF    I    the inclination of the final orbit
C.          ranF    I    the RAAN of the final orbit
C.          arpF    I    the arg. of perigee of the final orbit
C.          meaF    I    the mean anomaly of the final orbit
C.          lam_vect I   the 6x1 vector of initial Lagrange multipliers
C.
C.
C. ROUTINES REQUIRED: UNCMND, RK78, TRANS_OUT, DELTIM, ADDTIM
C.
C.....................................................................
C
C.
C
C
C***************** DECLARATIONS ***************************************
C
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      PARAMETER (N_INT=12, N_MIN=7,
     *           LWORK=N_MIN*(N_MIN+10))
      DOUBLE PRECISION  Y(N_INT), TOL
      DOUBLE PRECISION  T, DT, TDIFF

      DOUBLE PRECISION mu, ft, pi
      DOUBLE PRECISION sma0, ecc0, inc0, ran0, arp0, mea0
      DOUBLE PRECISION tf0, tf
      DOUBLE PRECISION smaF, eccF, incF, ranF, arpF, meaF
      DOUBLE PRECISION z0_vect(6), zF_vect(6), lam_vect(7)
      DOUBLE PRECISION z0, weights(7,20)
      DOUBLE PRECISION XDUM(N_INT), F1(N_INT), F2(N_INT), F3(N_INT)
      DOUBLE PRECISION F4(N_INT), F5(N_INT), F6(N_INT), F7(N_INT)
      DOUBLE PRECISION Y_OUT(13), Y_FIN_DIFF(6)
      DOUBLE PRECISION x(N_MIN), x0(N_MIN)
      DOUBLE PRECISION WORK(LWORK), F, EXTDAT, WEIGHT
      DOUBLE PRECISION eccA0, L0, t0_sec, DT_SEC, DT_HMS, DT_YMD
      DOUBLE PRECISION DELTA_T, t0_JUL, SECJUL, DAYJUL
      DOUBLE PRECISION Beta0, F0, sF0, cF0, r0, cL0, sL0
      DOUBLE PRECISION BetaF, FF, sFF, cFF, rF, cLF, sLF

      INTEGER   I, J
      INTEGER   MAX_ITER, N_INT, N_MIN, LWORK
      INTEGER   IERROR, iter
      INTEGER   t0_year, t0_month, t0_day, t0_hour, t0_min
      INTEGER   IT_YMD, IT_HM, It0_JUL

      EXTERNAL  FSUB,F_FORMIN
```

```
      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /WEIGHT/ weights, iter
C
C     Set PI
C
      pi = 3.141592653589793
C
C     Set the integration error tolerance
C
      TOL = 1.0E-10
C
C     Set the date for the initial time
C
      t0_year  = 2002
      t0_month = 8
      t0_day   = 4

      t0_hour  = 22
      t0_min   = 3
      t0_sec   = 23
C
C     Guess for the final time (seconds)
C
      tf0 = 853716.998592
C
C     Set the Earth gravity constant (km^3/s^2)
C
      mu = 3.986004418E5
C
C     Set the constant acceleration (km/s^2)
C
      ft = 7.5628E-9
C
C     Set the maximum iterations for the outer optimization loop
C
      MAX_ITER = 5
C
C     Set the initial Keplerian elements
C
      sma0 = 39382.97217
      ecc0 = 0.2006846493E-2
      inc0 = 1.43568 * pi/180.0
      ran0 = 115.95324 * pi/180.0
      arp0 = 297.5172826 * pi/180.0
      mea0 = 211.6003815 * pi/180.0
C
C     Set the final Keplerian elements
C
      smaF = 39537.70766
      eccF = 0.1621537776E-2
      incF = 1.43568 * pi/180
      ranF = 115.95324 * pi/180
      arpF = 297.5172826 * pi/180
      meaF = 212.7786305 * pi/180
C
C     Compute the initial and final eccentric anomaly
C
      eccA0 = SOLVE_ECC_ANOMALY(mea0,ecc0)
      eccAF = SOLVE_ECC_ANOMALY(meaF,eccF)
C
C     Compute the initial equinoctial elements
C
      z0_vect(1) = sma0
      z0_vect(2) = ecc0 * DSIN(arp0 + ran0)
      z0_vect(3) = ecc0 * DCOS(arp0 + ran0)
      z0_vect(4) = DTAN(inc0/2)*DSIN(ran0)
      z0_vect(5) = DTAN(inc0/2)*DCOS(ran0)
C
C     Compute the initial true longitude.
C
      Beta0 = 1.0/(1.0 + DSQRT(1.0-(z0_vect(2)**2.0)-(z0_vect(3))**2.0))
```

```
      F0 = eccA0 + ran0 + arp0

      sF0 = DSIN(F0)
      cF0 = DCOS(F0)

      r0 = sma0*(1.0 - z0_vect(3)*cF0 - z0_vect(2)*sF0)

      cL0 = (sma0/r0)*((1.0-Beta0*(z0_vect(2)**2.0))*cF0 +
     &      z0_vect(2)*z0_vect(3)*Beta0*sF0 - z0_vect(3))

      sL0 = (sma0/r0)*(z0_vect(2)*z0_vect(3)*Beta0*cF0 +
     &      (1.0-Beta0*(z0_vect(3)**2.0))*sF0 - z0_vect(2))

      L0 = DATAN2(sL0,cL0)

      z0_vect(6) = L0
C
C     Compute the final equinoctial elements
C
      zF_vect(1) = smaF
      zF_vect(2) = eccF * DSIN(arpF + ranF)
      zF_vect(3) = eccF * DCOS(arpF + ranF)
      zF_vect(4) = DTAN(incF/2)*DSIN(ranF)
      zF_vect(5) = DTAN(incF/2)*DCOS(ranF)
C
C     Compute the final true longitude
C
      BetaF = 1.0/(1.0 + DSQRT(1.0-(z0_vect(2)**2.0)-(z0_vect(3))**2.0))
      FF = eccAF + ranF + arpF

      sFF = DSIN(FF)
      cFF = DCOS(FF)

      rF = smaF*(1.0 - zF_vect(3)*cFF - zF_vect(2)*sFF)

      cLF = (smaF/rF)*((1.0-BetaF*(zF_vect(2)**2.0))*cFF +
     &      zF_vect(2)*zF_vect(3)*BetaF*sFF - zF_vect(3))

      sLF = (smaF/rF)*(zF_vect(2)*zF_vect(3)*BetaF*cFF +
     &      (1.0-BetaF*(zF_vect(3)**2.0))*sFF - zF_vect(2))

      LF = DATAN2(sLF,cLF)

      zF_vect(6) = LF
C
C     Set the initial guesses for the Lagrange multipliers
C
      lam_vect(1) = 0.326239885662E+04
      lam_vect(2) = -0.671795199693E+08
      lam_vect(3) = -0.478598052526E+08
      lam_vect(4) = 0.158973787328E+05
      lam_vect(5) = -0.129598180966E+05
      lam_vect(6) = 0.583925385401E+03

      lam_vect(7) = tf0

      DO J=1,7
         x0(J) = lam_vect(J)
      END DO
C
C     Combine all initial conditions into Y array
C
      DO I=1,6
         Y(I) = z0_vect(I)
      END DO
      DO I=1,6
         Y(I+6) = lam_vect(I)
      END DO
C
C     Set initial time point
C
```

```
      T = 0.0
      DT = 600.0
C
C     Set the weights for each outer loop optimization iteration
C
      weights(1,1) = 1.0E3
      weights(2,1) = 1.0E12
      weights(3,1) = 1.0E12
      weights(4,1) = 1.0E11
      weights(5,1) = 1.0E11
      weights(6,1) = 1.0
      weights(7,1) = 1.0E6

      weights(1,2) = 1.0E4
      weights(2,2) = 1.0E13
      weights(3,2) = 1.0E13
      weights(4,2) = 1.0E12
      weights(5,2) = 1.0E12
      weights(6,2) = 1.0
      weights(7,2) = 1.0E6

      weights(1,3) = 1.0E3
      weights(2,3) = 1.0E12
      weights(3,3) = 1.0E12
      weights(4,3) = 1.0E13
      weights(5,3) = 1.0E13
      weights(6,3) = 1.0
      weights(7,3) = 1.0E6

      weights(1,4) = 1.0E4
      weights(2,4) = 1.0E13
      weights(3,4) = 1.0E13
      weights(4,4) = 1.0E14
      weights(5,4) = 1.0E14
      weights(6,4) = 1.0
      weights(7,4) = 1.0E6

      weights(1,5) = 1.0E4
      weights(2,5) = 1.0E12
      weights(3,5) = 1.0E12
      weights(4,5) = 1.0E14
      weights(5,5) = 1.0E14
      weights(6,5) = 1.0
      weights(7,5) = 1.0E6
C
C     If we are not using the optimizer, assign x
C
      DO I=1,7
         x(I) = x0(I)
      END DO
C
C     Set up the outer optimization loop
C
      DO iter=1,MAX_ITER
C
C        Call the unconstrained minimization subroutine
C
         CALL UNCMND (N_MIN, x0, F_FORMIN, x, F, IERROR, WORK, LWORK)
C
C        Print out the results of the minimization (the Lagrange multipliers and final
time)
C
         WRITE (*,*) 'Results of opt: Lagrange mult and final time'
         WRITE (*,'(7E24.12)') (x(I), I=1,7)
C
C        Copy the output back to the input.
C
         DO J=1,7
            x0(J) = x(J)
         END DO
      END DO
```

```
C
C     Open the file needed to store the output meant for GTDS
C
      OPEN (UNIT = 115, FORM = 'FORMATTED', ACCESS = 'SEQUENTIAL',
     1      FILE = 'optimal_traj.thr',
     2      STATUS = 'UNKNOWN')

      WRITE(115,1001) 'THRCARD '
C
C     Assign results of optimization to input for integration
C     and printout of the final trajectory
C
      DO I=1,6
         Y(I) = z0_vect(I)
      END DO
      Y(7)  = x(1)
      Y(8)  = x(2)
      Y(9)  = x(3)
      Y(10) = x(4)
      Y(11) = x(5)
      Y(12) = x(6)
      tf    = x(7)
C
C     Print out the results of the minimization (the Lagrange multipliers and final time)
C
      WRITE (*,*) 'Results of optimization Lagrange mult and final time'
      WRITE (*,'(7E24.12)') (x(I), I=1,7)
C
C     Output the final trajectory result of the optimization
C
      DO WHILE (T .LE. tf)
C
C        Integrate.
C
         CALL RK78 (IFLAG,N_INT,T,DT,Y,TOL,
     &              XDUM,F1,F2,F3,F4,F5,F6,F7,FSUB)
C
C        Are we finished?  If so, exit the loop.
C
         IF (T .EQ. tf) THEN
C
C           Write the output at this time step.
C
            CALL TRANS_OUT(Y, Y_OUT, ft, mu)
C
C           Compare the desired elements with the final elements achieved.
C
            Y_FIN_DIFF(1) = Y_OUT(1) - smaF
            Y_FIN_DIFF(2) = Y_OUT(2) - eccF
            Y_FIN_DIFF(3) = (Y_OUT(3) - incF)*180.0/pi
            Y_FIN_DIFF(4) = (Y_OUT(4) - ranF)*180.0/pi
            Y_FIN_DIFF(5) = (Y_OUT(5) - arpF)*180.0/pi
            Y_FIN_DIFF(6) = (Y_OUT(6) - meaF)*180.0/pi

            WRITE (*,*) 'Final element differences'
            WRITE (*,'(6E24.12)') Y_FIN_DIFF(1),Y_FIN_DIFF(2),
     &          Y_FIN_DIFF(3),Y_FIN_DIFF(4),Y_FIN_DIFF(5),Y_FIN_DIFF(6)

            WRITE (*,*) 'Final elements'
            WRITE (*,'(6E24.12)') Y_OUT(1),Y_OUT(2),
     &          Y_OUT(3),Y_OUT(4),Y_OUT(5),Y_OUT(6)
C
C
C        We are not yet finished, find the time yet to integrate.
C        If that time is less than the next time step, reduce the
C        next time step to equal the time left to integrate.
C
         ELSE
            TDIFF = tf - T
            IF (TDIFF .LT. DT) THEN
               DT = TDIFF
```

```
            END IF
         END IF
C
C        Write the output at this time step.
C
         CALL TRANS_OUT(Y, Y_OUT, ft, mu)

         WRITE (*,'(I9,11E24.12)') 111111111,
     *            T,Y_OUT(1),Y_OUT(2),Y_OUT(3),Y_OUT(4),Y_OUT(5),
     *              Y_OUT(6),Y_OUT(7),Y_OUT(12),Y_OUT(13)
C
C        Convert the time in seconds to YYYMMDDHHMMSS.sss format.
C
C        First, convert the initial t0 date to a Julian date
C
         DT_YMD = (t0_year - 1900)*1E4 + t0_month*1E2 + t0_day
         DT_HMS = (t0_hour)*1E4 + t0_min*1E2 + t0_sec

         CALL DELTIM(1,DT_YMD,DT_HMS,t0_JUL,t0_JUL,DELTA_T)
C
C        Add 2430000 to get the Julian Date from the Modified Julian Date
C
         t0_JUL = t0_JUL + 2430000
C
C        Now, find the Julian and Gregorian date of the final time
C
         DAYJUL  = DINT(t0_JUL)
         SECJUL  = (t0_JUL - DAYJUL)*86400.0

         CALL ADDTIM(DT_YMD,DT_HMS,DAYJUL,SECJUL,T,0.0001)

         IT_YMD = DT_YMD
         IT_HM  = DT_HMS/100.0
         DT_SEC = DMOD(DT_HMS,100.0)
C
C        Write the output to the obs file intended for GTDS
C
         IF (DT_SEC .LT. 10.0) THEN
            WRITE(115,1004) '        ',1,0,0,
     &                      IT_YMD,IT_HM,0,DT_SEC,Y_OUT(8),Y_OUT(9),
     &                      Y_OUT(10),ft
         ELSE
            WRITE(115,1003) '        ',1,0,0,
     &                        IT_YMD,IT_HM,DT_SEC,Y_OUT(8),Y_OUT(9),
     &                        Y_OUT(10),ft
         END IF
C
C        We have finished, exit the loop
C
         IF (T .EQ. tf) THEN
            EXIT
         END IF

       END DO
C
C      Finish writing to the GTDS thrust file and close it.
C
       WRITE(115,1002) 'END     '
       CLOSE(UNIT=115,STATUS='KEEP')

  1001 FORMAT(1A8)
  1002 FORMAT(1A3)
  1003 FORMAT(1A8,3I3,3X,1I7.7,1I4.4,1F7.4,4F21.14)
  1004 FORMAT(1A8,3I3,3X,1I7.7,1I4.4,1I1.1,1F6.4,4F21.14)
       END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: fsub.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      SUBROUTINE FSUB (T,Y,YDOT)
C
C.....................................................................
C. ROUTINE: FSUB
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: This the subroutine that is called by the FK78 subroutine to supply
C.    the equinoctial element and Lagrange multiplier derivatives with respect to
C.    time, i.e. rates.  FSUB calls the COMP_XY, COMP_B, and COMP_U subroutines to
C.    calculate the auxiliary quantities, the 6x3 BL matrix and the normalized thrust
C.    acceleration vector.  FSUB then executes the COMP_EQUIN_VAR and COMP_EUL_LAG_VAR
C.    subroutines which compute the rates for the equinoctial variation equations and
C.    the rates for the Lagrange multipliers, respectively.
C.
C.
C. CALLING SEQUENCE:
C.          FSUB(T,Y,YDOT)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER   I/O                    DESCRIPTION
C.          ------  ---  ---------------------------------------------
C.          T       I    The current time in seconds from time zero
C.          Y       I    The input vector of current equinoctial orbital
C.                       elements in elements 1-6 and the vector of
C.                       current lagrange multipliers in elements 7-12.
C.          YDOT    O    The output vector of equinoctial element rates in
C.                       elements 1-6 and the output vector of lagrange
C.                       multipliers in elements 7-12.
C.
C. ROUTINES REQUIRED: COMP_XY, COMP_U, COMP_B, COMP_EQUIN_VAR,
C.                    COMP_EUL_LAG_VAR
C.
C.....................................................................
C
C.
C
C
C***************** DECLARATIONS ***************************************
C
C
C
C       Routine for evaluating right hand sides of equations.
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      INTEGER N

      DOUBLE PRECISION T, Y(*), YDOT(*)
      DOUBLE PRECISION mu, ft, u_mag
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION z0_vect(6), zF_vect(6), EXTDAT
```

```
      DOUBLE PRECISION nm, cL, sL, G, r, K1
      DOUBLE PRECISION B(6,3), dBda(6,3), dBdh(6,3), dBdk(6,3)
      DOUBLE PRECISION dBdp(6,3), dBdq(6,3), dBdL(6,3)
      DOUBLE PRECISION u(3), dadt, dhdt, dkdt, dpdt, dqdt, dLdt
      DOUBLE PRECISION dlamadt, dlamhdt, dlamkdt
      DOUBLE PRECISION dlampdt, dlamqdt, dlamLdt

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
C
C     Assign the input arrays
C
      z_vect(1) = Y(1)
      z_vect(2) = Y(2)
      z_vect(3) = Y(3)
      z_vect(4) = Y(4)
      z_vect(5) = Y(5)
      z_vect(6) = Y(6)

      lam_vect(1) = Y(7)
      lam_vect(2) = Y(8)
      lam_vect(3) = Y(9)
      lam_vect(4) = Y(10)
      lam_vect(5) = Y(11)
      lam_vect(6) = Y(12)
C
C     Compute some parameters needed later
C
      CALL COMP_XY(z_vect,mu,nm,cL,sL,G,r,K1)
C
C     Compute the B matrix of equinoctial partials wrt rdot,
C     the partials of B wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_B(z_vect,nm,cL,sL,G,r,K1,
     &            B,dBda,dBdh,dBdk,dBdp,dBdq,dBdL)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,B,u,u_mag)
C
C     Compute the right hand side of the equinoctial element variational equations
C
      CALL COMP_EQUIN_VAR(B,u,ft,nm,z_vect(1),z_vect(2),z_vect(3),r,
     &                    dadt,dhdt,dkdt,dpdt,dqdt,dLdt)
C
C     Compute the right hand side of the Lagrange multiplier variational equations
C
      CALL COMP_EUL_LAG_VAR(z_vect,lam_vect,u,nm,ft,G,r,
     &                      dBda,dBdh,dBdk,dBdp,dBdq,dBdL,
     &                      dlamadt,dlamhdt,dlamkdt,dlampdt,
     &                      dlamqdt,dlamLdt)
C
C     Assign the output rates
C
      YDOT(1) = dadt
      YDOT(2) = dhdt
      YDOT(3) = dkdt
      YDOT(4) = dpdt
      YDOT(5) = dqdt
      YDOT(6) = dLdt
      YDOT(7) = dlamadt
      YDOT(8) = dlamhdt
      YDOT(9) = dlamkdt
      YDOT(10) = dlampdt
      YDOT(11) = dlamqdt
      YDOT(12) = dlamLdt
      RETURN
      END
```

```
C -------------------------------------------------------------------------
C
C   FILE NAME: F_FORMIN.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------
C
      SUBROUTINE F_FORMIN(N, X, F)
C
C.....................................................................
C. ROUTINE: F_FORMIN
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the equinoctial elements and Lagrange multipliers
C.    at the final time given the elements and multipliers at the initial
C.    time.  F_FORMIN also computes the sum of the squares of the differences
C.    of the computed final orbital element conditions from the desired orbital
C.    element conditions.  F_FORMIN uses the RK78 subroutine to perform the
C.    integration of the equinoctial orbital elements and the Lagrange multipliers.
C.    F_FORMIN is called by UNCMND to perform unconstrained minimization of the
C.    F cost function defined in F_FORMIN.
C.
C.
C. CALLING SEQUENCE:
C          CALL F_FORMIN(N, X, F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ------------------------------------
C.          N          I    Number of parameters to vary in search
C.                          for minimum.  In this case it is the 6
C.                          Lagrange multipliers plus the final time
C.                          for a total of 7.
C.          X          I    vector of lagrange multipliers and tf
C.          F          O    The value of the cost function given X
C.
C.
C. ROUTINES REQUIRED:  RK78, COMP_XY, COMP_B, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      PARAMETER(N_INT=12,N_MIN=7)
      INTEGER N_MIN, I, N_INT, IFLAG
      INTEGER MS, NROOT, MINT, LW, IW, LIW, iter
      DOUBLE PRECISION X(N_MIN), F, pi
      DOUBLE PRECISION mu,ft,aF,hF,kF,pF,qF,LF,ecc,inc
      DOUBLE PRECISION z0_vect(6), zF_vect(6)
      DOUBLE PRECISION a,h,k,p,q,L,lam(6),z_vect(6)
      DOUBLE PRECISION nm,cL,sL,G,r,K1
      DOUBLE PRECISION B(6,3),dBda(6,3),dBdh(6,3),dBdk(6,3)
      DOUBLE PRECISION dBdp(6,3),dBdq(6,3),dBdL(6,3)
      DOUBLE PRECISION B_u(6), lam_B_u, u(3), u_mag
```

```
      DOUBLE PRECISION wgt(7)
      DOUBLE PRECISION tf, Y(N_INT+3)
      DOUBLE PRECISION T, DT, TOL, TDIFF
      DOUBLE PRECISION weights(7,20)
      DOUBLE PRECISION EXTDAT, WEIGHT
      DOUBLE PRECISION XDUM(N_INT), F1(N_INT), F2(N_INT), F3(N_INT)
      DOUBLE PRECISION F4(N_INT), F5(N_INT), F6(N_INT), F7(N_INT)

      EXTERNAL  FSUB,GFUN

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /WEIGHT/ weights, iter
C
C     Set PI
C
      pi = 3.141592653589793
C
C     Initialize values needed by the integrator
C
C
C     Set the integration error tolerance
C
      TOL = 1.0E-10
C
C     Set initial time point
C
      T = 0.0
C
C     Copy the final time guess.
C
      tf = X(7)
C
C     Set the initial guess for integration.
C
      Y(1) = z0_vect(1)
      Y(2) = z0_vect(2)
      Y(3) = z0_vect(3)
      Y(4) = z0_vect(4)
      Y(5) = z0_vect(5)
      Y(6) = z0_vect(6)
      Y(7) = X(1)
      Y(8) = X(2)
      Y(9) = X(3)
      Y(10) = X(4)
      Y(11) = X(5)
      Y(12) = X(6)
C
C     Copy the final elements
C
      aF = zF_vect(1)
      hF = zF_vect(2)
      kF = zF_vect(3)
      pF = zF_vect(4)
      qF = zF_vect(5)
      LF = zF_vect(6)
C
C     We want the orbital and Lagrange multiplier
C     values only at the final time
C
      DT = 600.0
C
C     Start the integration loop
C
      DO WHILE (T .LE. tf)
C
C        Integrate.
C
         CALL RK78 (IFLAG,N_INT,T,DT,Y,TOL,
     &              XDUM,F1,F2,F3,F4,F5,F6,F7,FSUB)
C
C        Are we finished?  If so, exit the loop.
```

```
C
         IF (T .EQ. tf) THEN
            EXIT
C
C        We are not yet finished, find the time yet to integrate.
C        If that time is less than the next time step, reduce the
C        next time step to equal the time left to integrate.
C
         ELSE
            TDIFF = tf - T
            IF (TDIFF .LT. DT) THEN
               DT = TDIFF
            END IF
         END IF

       END DO
C
C     If the integrator was happy, compute the function value
C
      IF (T .EQ. tf) THEN

         a = Y(1)
         h = Y(2)
         k = Y(3)
         p = Y(4)
         q = Y(5)
         L = Y(6)

         lam(1) = Y(7)
         lam(2) = Y(8)
         lam(3) = Y(9)
         lam(4) = Y(10)
         lam(5) = Y(11)
         lam(6) = Y(12)
C
C        Compute the Hamiltonion at the final time
C
         z_vect(1) = a
         z_vect(2) = h
         z_vect(3) = k
         z_vect(4) = p
         z_vect(5) = q
         z_vect(6) = L
C
C        Compute some parameters needed later
C
         CALL COMP_XY(z_vect,mu,nm,cL,sL,G,r,K1)
C
C        Compute the B matrix of equinoctial partials wrt rdot,
C        the partials of B wrt the equinoctial elements, and
C        auxiliary partial derivatives.
C
         CALL COMP_B(z_vect,nm,cL,sL,G,r,K1,
     &               B,dBda,dBdh,dBdk,dBdp,dBdq,dBdL)
C
C        Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
         CALL COMP_U(lam,B,u,u_mag)
C
C        Calculate the Hamiltonian
C
         B_u(1) = B(1,1)*u(1)+B(1,2)*u(2)+B(1,3)*u(3)
         B_u(2) = B(2,1)*u(1)+B(2,2)*u(2)+B(2,3)*u(3)
         B_u(3) = B(3,1)*u(1)+B(3,2)*u(2)+B(3,3)*u(3)
         B_u(4) = B(4,1)*u(1)+B(4,2)*u(2)+B(4,3)*u(3)
         B_u(5) = B(5,1)*u(1)+B(5,2)*u(2)+B(5,3)*u(3)
         B_u(6) = B(6,1)*u(1)+B(6,2)*u(2)+B(6,3)*u(3)

         lam_B_u = lam(1)*B_u(1)+lam(2)*B_u(2)+lam(3)*B_u(3)+
     &             lam(4)*B_u(4)+lam(5)*B_u(5)+lam(6)*B_u(6)
```

```
          Ham = ft*lam_B_u + lam(6)*(a**2.0)*nm*
     &         ((1.0-h**2.0-k**2.0)**(1.0/2.0))/(r**2.0)
C
C         Assign the weights
C
          DO I=1,7
             wgt(I) = weights(I,iter)
          END DO
C
C         This cost function is for Kechichian's LEO to GEO case
C
          F = wgt(1)*(a - aF)**2.0 + wgt(2)*(h - hF)**2.0 +
     &        wgt(3)*(k - kF)**2.0 + wgt(4)*(p - pF)**2.0 +
     &        wgt(5)*(q - qF)**2.0 + wgt(6)*(lam(6) - 0.0)**2.0 +
     &        wgt(7)*(Ham - 1.0)**2.0

          ecc = (h**2.0 + k**2.0)**(1.0/2.0)
          inc = 2.0*DATAN2(DSQRT(p**2.0 + q**2.0),1.0)*180.0/pi

          WRITE (*,*) 'F_FORMIN output'

          WRITE (*,'(I3,6E16.7)')
     &          iter,F,Ham,tf,a,ecc,inc
          WRITE (*,'(I3,7E14.5)')
     &          iter,
     &          wgt(1)*((a - aF)**2.0),
     &          wgt(2)*((h - hF)**2.0),
     &          wgt(3)*((k - kF)**2.0),
     &          wgt(4)*((p - pF)**2.0),
     &          wgt(5)*((q - qF)**2.0),
     &          wgt(6)*((lam(6) - 0.0)**2.0),
     &          wgt(7)*((Ham - 1.0)**2.0)

C
C      If the integrator was unhappy, print a message and return
C
       ELSE
          WRITE (*,*) 'Error in integrating from initial to final time.'
          F = 0.0
          RETURN
       END IF

       RETURN
       END
```

```
C -----------------------------------------------------------------------
C
C   FILE NAME: COMP_B.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -----------------------------------------------------------------------
C
      SUBROUTINE COMP_B(z_vect,n,cL,sL,G,r,K1,
     &                  B,dBda,dBdh,dBdk,dBdp,dBdq,dBdL)
C
C.....................................................................
C. ROUTINE: COMP_B
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the 6x3 BL matrix and its partial derivatives
C.     with respect to the equinoctial elements.  The equations for
C.     this subroutine can be found in the Appendix of [Kechichian, J. A.,
C.     Trajectory Optimization Using Nonsingular Orbital Elements and True
C.     Longitude.  Journal of Guidance, Control and DYnamics.  Vol. 20, No. 5,
C.     Sept-Oct. 1997].
C.
C.
C. CALLING SEQUENCE:
C          COMP_B(z_vect,n,cL,sL,G,r,K1,B,dBda,dBdh,dBdk,dBdp,dBdq,dBdL)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER   I/O                    DESCRIPTION
C.          ------  ---  ------------------------------------------
C.          z_vect  I    The input equinoctial elements
C.          n       I    The mean motion
C.          cL      I    The cosine of the true longitude
C.          sL      I    The sine of the true longitude
C.          G       I    An auxiliary parameter dependent on h and k
C.                       equinoctial elements
C.          r       I    The current radial distance from the center of
C.                       the central body to the satellite
C.          K1      I    An auxiliary orbital parameter based on p and q
C.          B       O    The 6x3 output matrix containing the partial
C.                       derivatives of the equinoctial elements wrt rdot.
C.          dBda    O    The 6x3 output matrix of partials of B wrt a
C.          dBdh    O    The 6x3 output matrix of partials of B wrt h
C.          dBdk    O    The 6x3 output matrix of partials of B wrt k
C.          dBdp    O    The 6x3 output matrix of partials of B wrt p
C.          dBdq    O    The 6x3 output matrix of partials of B wrt q
C.          dBdL    O    The 6x3 output matrix of partials of B wrt true long.
C.
C.
C. ROUTINES REQUIRED:  NONE
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      DOUBLE PRECISION z_vect(6),n,cL,sL,G,r,K1
      DOUBLE PRECISION a,h,k,p,q,L
      DOUBLE PRECISION B(6,3),dBda(6,3),dBdh(6,3),dBdk(6,3)
```

```
      DOUBLE PRECISION dBdp(6,3),dBdq(6,3),dBdL(6,3)
      DOUBLE PRECISION drda,drdh,drdk,drdL,dnda
C
C     These partial derivatives taken from Kechichian:
C        "Trajectory Optimization Using Nonsingular
C         Orbital Elements and True Longitude."

      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      L = z_vect(6)
C
C     Auxiliary partials
C
      drda = r/a
      drdh = -r*(2.0*a*h + r*sL)/(a*(1.0 - h**2.0 - k**2.0))
      drdk = -r*(2.0*a*k + r*cL)/(a*(1.0 - h**2.0 - k**2.0))
      drdL = -(r**2.0*(h*cL - k*sL))/(a*(1.0 - h**2.0 - k**2.0))
      dnda = -3.0*n/(2.0*a)
C
C     Partials of a wrt rdot
C
      B(1,1) = 2.0*(n**-1.0)*(G**-1.0)*(k*sL - h*cL)
      B(1,2) = 2.0*(n**-1.0)*a*(r**-1.0)*G
      B(1,3) = 0
C
C     Partials of h wrt rdot
C
      B(2,1) = -(n**-1.0)*(a**-1.0)*G*cL
      B(2,2) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*(h + sL) +
     &         (n**-1.0)*(a**-1.0)*G*sL
      B(2,3) = -(n**-1.0)*(a**-2.0)*r*(G**-1.0)*k*(p*cL - q*sL)
C
C     Partials of k wrt rdot
C
      B(3,1) = (n**-1)*(a**-1.0)*G*sL
      B(3,2) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*(k + cL) +
     &         (n**-1.0)*(a**-1.0)*G*cL
      B(3,3) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*h*(p*cL - q*sL)
C
C     Partials of p wrt rdot
C
      B(4,1) = 0
      B(4,2) = 0
      B(4,3) = (2.0**-1.0)*(n**-1.0)*(a**-2.0)*r*(G**-1.0)*K1*sL
C
C     Partials of q wrt rdot
C
      B(5,1) = 0
      B(5,2) = 0
      B(5,3) = (2.0**-1.0)*(n**-1.0)*(a**-2.0)*r*(G**-1.0)*K1*cL
C
C     Partials of L wrt rdot
C
      B(6,1) = 0
      B(6,2) = 0
      B(6,3) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*(q*sL - p*cL)
C
C     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C     These next partials are partials of the M matrix wrt elements
C
C     Partials of B wrt h
C
      dBdh(1,1) = 2.0*(n**-1.0)*h*(G**-3.0)*(k*sL - h*cL) -
     &            2.0*(n**-1.0)*(G**-1.0)*cL
      dBdh(1,2) = -2.0*(1.0/n)*a*(1.0/(r**2.0))*drdh*G -
     &            2.0*(1.0/n)*a*(1.0/r)*h*(1.0/G)
      dBdh(1,3) = 0.0
```

```
       dBdh(2,1) = (n**-1.0)*(a**-1.0)*h*(G**-1.0)*cL

       dBdh(2,2) = (n**-1.0)*(a**-2.0)*(G**-1.0)*(h + sL)*
      &            (drdh + r*h*(G**-2.0)) +
      &            (n**-1.0)*(a**-2.0)*r*(G**-1.0) -
      &            (n**-1.0)*(a**-1.0)*h*sL*(G**-1.0)

       dBdh(2,3) = -(n**-1.0)*(a**-2.0)*(G**-1.0)*k*(p*cL - q*sL)*(drdh +
      &            h*r*(G**-2.0))

       dBdh(3,1) = -(n**-1.0)*(a**-1.0)*(G**-1.0)*h*sL

       dBdh(3,2) = (n**-1.0)*(a**-2.0)*(G**-1.0)*(k + cL)*(drdh +
      &            h*r*(G**-2.0)) -
      &            (n**-1.0)*(a**-1.0)*h*(G**-1.0)*cL

       dBdh(3,3) = (n**-1.0)*(a**-2.0)*(G**-1.0)*h*(p*cL - q*sL)*(drdh +
      &            h*r*(G**-2.0)) +
      &            (n**-1.0)*(a**-2.0)*r*(G**-1.0)*(p*cL - q*sL)

       dBdh(4,1) = 0.0
       dBdh(4,2) = 0.0

       dBdh(4,3) = (2.0**-1.0)*(n**-1.0)*(a**-2.0)*(G**-1.0)*K1*sL*(drdh+
      &            h*r*(G**-2.0))

       dBdh(5,1) = 0.0
       dBdh(5,2) = 0.0

       dBdh(5,3) = (2.0**-1.0)*(n**-1.0)*(a**-2.0)*(G**-1.0)*K1*cL*(drdh+
      &            h*r*(G**-2.0))

       dBdh(6,1) = 0.0

       dBdh(6,2) = 0.0

       dBdh(6,3) = (n**-1.0)*(a**-2.0)*(q*sL - p*cL)*(G**-1.0)*(drdh +
      &            r*h*(G**-2.0))
C
C      The partials of B wrt k
C
       dBdk(1,1) = 2.0*(n**-1.0)*k*(G**-3.0)*(k*sL - h*cL) +
      &            2.0*(n**-1.0)*(G**-1.0)*sL

       dBdk(1,2) = -2.0*(1.0/n)*a*(1.0/(r**2.0))*G*drdk -
      &            2.0*(1.0/n)*a*(1.0/r)*k*(1.0/G)

       dBdk(1,3) = 0.0

       dBdk(2,1) = (n**-1.0)*(a**-1.0)*k*(G**-1.0)*cL

       dBdk(2,2) = (n**-1.0)*(a**-2.0)*(G**-1.0)*(h + sL)*
      &            (drdk + r*k*(G**-2.0)) -
      &            (n**-1.0)*(a**-1.0)*k*sL*(G**-1.0)

       dBdk(2,3) = -(n**-1.0)*(a**-2.0)*(G**-1.0)*k*(p*cL-q*sL)*
      &            (drdk+k*r*(G**-2.0)) -
      &            (n**-1.0)*(a**-2.0)*r*(G**-1.0)*(p*cL - q*sL)

       dBdk(3,1) = -(n**-1.0)*(a**-1.0)*(G**-1.0)*k*sL

       dBdk(3,2) = (n**-1.0)*(a**-2.0)*(G**-1.0)*(k + cL)*
      &            (drdk+k*r*(G**-2.0)) -
      &            (n**-1.0)*(a**-1.0)*k*(G**-1.0)*cL +
      &            (n**-1.0)*(a**-2.0)*r*(G**-1.0)

       dBdk(3,3) = (n**-1.0)*(a**-2.0)*(G**-1.0)*h*(p*cL - q*sL)*(drdk +
      &            k*r*(G**-2.0))

       dBdk(4,1) = 0.0
       dBdk(4,2) = 0.0
```

```
      dBdk(4,3) = (2.0**-1.0)*(n**-1.0)*(a**-2.0)*(G**-1.0)*K1*sL*(drdk+
     &            k*r*(G**-2.0))

      dBdk(5,1) = 0.0
      dBdk(5,2) = 0.0

      dBdk(5,3) = (2.0**-1.0)*(n**-1.0)*(a**-2.0)*(G**-1.0)*K1*cL*(drdk+
     &            k*r*(G**-2.0))

      dBdk(6,1) = 0.0
      dBdk(6,2) = 0.0

      dBdk(6,3) = (n**-1.0)*(a**-2.0)*(q*sL - p*cL)*(G**-1.0)*(drdk +
     &            r*k*(G**-2.0))
C
C     The partials of B wrt p
C
      dBdp(1,1) = 0.0
      dBdp(1,2) = 0.0
      dBdp(1,3) = 0.0

      dBdp(2,1) = 0.0
      dBdp(2,2) = 0.0
      dBdp(2,3) = -(n**-1.0)*(a**-2.0)*r*(G**-1.0)*k*cL

      dBdp(3,1) = 0.0
      dBdp(3,2) = 0.0
      dBdp(3,3) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*h*cL

      dBdp(4,1) = 0.0
      dBdp(4,2) = 0.0
      dBdp(4,3) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*p*sL

      dBdp(5,1) = 0.0
      dBdp(5,2) = 0.0
      dBdp(5,3) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*p*cL

      dBdp(6,1) = 0.0
      dBdp(6,2) = 0.0
      dBdp(6,3) = -(n**-1.0)*(a**-2.0)*r*(G**-1.0)*cL
C
C     Partials of B wrt q
C
      dBdq(1,1) = 0.0
      dBdq(1,2) = 0.0
      dBdq(1,3) = 0.0

      dBdq(2,1) = 0.0
      dBdq(2,2) = 0.0
      dBdq(2,3) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*k*sL

      dBdq(3,1) = 0.0
      dBdq(3,2) = 0.0
      dBdq(3,3) = -(n**-1.0)*(a**-2.0)*r*(G**-1.0)*h*sL

      dBdq(4,1) = 0.0
      dBdq(4,2) = 0.0
      dBdq(4,3) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*q*sL

      dBdq(5,1) = 0.0
      dBdq(5,2) = 0.0
      dBdq(5,3) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*q*cL

      dBdq(6,1) = 0.0
      dBdq(6,2) = 0.0
      dBdq(6,3) = (n**-1.0)*(a**-2.0)*r*(G**-1.0)*sL
C
C     The partials of B wrt a
C
      dBda(1,1) = -2.0*(n**-2.0)*dnda*(G**-1.0)*(k*sL - h*cL)
```

```
      dBda(1,2) = -2.0*(n**-2.0)*a*(r**-1.0)*dnda*G
      dBda(1,3) = 0.0

      dBda(2,1) = -(2.0**-1.0)*(n**-1.0)*(a**-2.0)*G*cL

      dBda(2,2) = (2.0**-1.0)*(n**-1.0)*(a**-3.0)*r*(G**-1.0)*(h + sL) +
     &            (2.0**-1.0)*(n**-1.0)*(a**-2.0)*G*sL

      dBda(2,3) = -(2.0**-1.0)*(n**-1.0)*(a**-3.0)*r*(G**-1.0)*k*(p*cL -
     &            q*sL)

      dBda(3,1) = (2.0**-1.0)*(n**-1.0)*(a**-2.0)*G*sL

      dBda(3,2) = (2.0**-1.0)*(n**-1.0)*(a**-3.0)*r*(G**-1.0)*(k + cL) +
     &            (2.0**-1.0)*(n**-1.0)*(a**-2.0)*G*cL

      dBda(3,3) = (2.0**-1.0)*(n**-1.0)*(a**-3.0)*r*(G**-1.0)*h*(p*cL -
     &            q*sL)

      dBda(4,1) = 0.0
      dBda(4,2) = 0.0
      dBda(4,3) = (4.0**-1.0)*(n**-1.0)*(a**-3.0)*r*(G**-1.0)*K1*sL

      dBda(5,1) = 0.0
      dBda(5,2) = 0.0
      dBda(5,3) = (4.0**-1.0)*(n**-1.0)*(a**-3.0)*r*(G**-1.0)*K1*cL

      dBda(6,1) = 0.0
      dBda(6,2) = 0.0
      dBda(6,3) = (2.0**-1.0)*(n**-1.0)*(a**-3.0)*r*(G**-1.0)*(q*sL -
     &            p*cL)
C
C     Partials of B wrt L
C
      dBdL(1,1) = 2.0*(n**-1.0)*(G**-1.0)*(k*cL + h*sL)
      dBdL(1,2) = -2.0*(1.0/n)*a*(1.0/(r**2.0))*G*drdL
      dBdL(1,3) = 0.0

      dBdL(2,1) = (1.0/n)*(1.0/a)*G*sL
      dBdL(2,2) = (1.0/n)*(1.0/(a**2.0))*(h + sL)*(1.0/G)*drdL +
     &            (1.0/n)*(1.0/(a**2.0))*r*cL*(1.0/G) +
     &            (1.0/n)*(1.0/a)*cL*G
      dBdL(2,3) = -(n**-1.0)*(a**-2.0)*(G**-1.0)*k*(p*cL - q*sL)*drdL +
     &            (n**-1.0)*(a**-2.0)*r*(G**-1.0)*k*(p*sL + q*cL)

      dBdL(3,1) = (1.0/n)*(1.0/a)*G*cL
      dBdL(3,2) = (1.0/n)*(1.0/(a**2.0))*(1.0/G)*(k + cL)*drdL -
     &            (1.0/n)*(1.0/(a**2.0))*r*(1.0/G)*sL -
     &            (1.0/n)*(1.0/a)*G*sL
      dBdL(3,3) = (n**-1.0)*(a**-2.0)*(G**-1.0)*h*(p*cL - q*sL)*drdL -
     &            (n**-1.0)*(a**-2.0)*r*(G**-1.0)*h*(p*sL + q*cL)

      dBdL(4,1) = 0.0
      dBdL(4,2) = 0.0
      dBdL(4,3) = (2.0**-1.0)*(n**-1.0)*(a**-2.0)*(G**-1.0)*K1*sL*drdL +
     &            (2.0**-1.0)*(n**-1.0)*(a**-2.0)*r*(G**-1.0)*K1*cL

      dBdL(5,1) = 0.0
      dBdL(5,2) = 0.0
      dBdL(5,3) = (2.0**-1.0)*(n**-1.0)*(a**-2.0)*(G**-1.0)*K1*cL*drdL -
     &            (2.0**-1.0)*(n**-1.0)*(a**-2.0)*r*(G**-1.0)*K1*sL

      dBdL(6,1) = 0.0
      dBdL(6,2) = 0.0
      dBdL(6,3) = (1.0/n)*(1.0/(a**2.0))*(q*sL - p*cL)*(1.0/G)*drdL +
     &            (1.0/n)*(1.0/(a**2.0))*r*(q*cL + p*sL)*(1.0/G)


      RETURN
      END
```

```
C ------------------------------------------------------------------------
C
C   FILE NAME: COMP_EQUIN_VAR.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------
C
      SUBROUTINE COMP_EQUIN_VAR(B,u,ft,n,a,h,k,r,
     &                          dadt,dhdt,dkdt,dpdt,dqdt,dLdt)
C
C.....................................................................
C. ROUTINE: COMP_EQUIN_VAR
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the derivatives of the equinoctial orbital elements
C.       with respect to time, i.e. element rates.  This is done by multiplying
C.       the constant thrust acceleration magnitude by the product of the BL
C.       matrix and the normalized thrust acceleration vector.
C.
C.
C. CALLING SEQUENCE:
C          COMP_EQUIN_VAR(B,u,ft,n,a,h,k,r,dadt,dhdt,dkdt,dpdt,dqdt,dLdt)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER   I/O                   DESCRIPTION
C.          ------  ---  ---------------------------------------------
C.          B       I    The 6x3 partial derivative matrix
C.          u       I    The 3x1 thrust vector
C.          ft      I    The thrust acceleration magnitude
C.          n       I    The mean motion
C.          a       I    The semimajor axis
C.          h       I    The h equinoctial element
C.          k       I    The k equinoctial element
C.          r       I    The radial distance between the central
C.                       body center and the satellite
C.          dadt    O    The output time derivative of the semimajor axis
C.          dhdt    O    The output time derivative of h
C.          dkdt    O    The output time derivative of k
C.          dpdt    O    The output time derivative of p
C.          dqdt    O    The output time derivative of q
C.          dLdt    O    The output time derivative of true long.
C.
C.
C. ROUTINES REQUIRED:  NONE
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      DOUBLE PRECISION B(6,3),u(3),ft,n,a,h,k,r
      DOUBLE PRECISION dadt,dhdt,dkdt,dpdt,dqdt,dLdt

      dadt = ft*(B(1,1)*u(1) + B(1,2)*u(2) + B(1,3)*u(3))

      dhdt = ft*(B(2,1)*u(1) + B(2,2)*u(2) + B(2,3)*u(3))
```

```
      dkdt = ft*(B(3,1)*u(1) + B(3,2)*u(2) + B(3,3)*u(3))

      dpdt = ft*(B(4,1)*u(1) + B(4,2)*u(2) + B(4,3)*u(3))

      dqdt = ft*(B(5,1)*u(1) + B(5,2)*u(2) + B(5,3)*u(3))

      dLdt = ft*(B(6,1)*u(1) + B(6,2)*u(2) + B(6,3)*u(3)) +
     &       (n*a**2.0*((1.0-h**2.0-k**2.0)**(1.0/2.0))/r**2.0)

      RETURN
      END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: COMP_EUL_LAG_VAR.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      SUBROUTINE COMP_EUL_LAG_VAR(z_vect,lam_vect,u,n,ft,G,r,
     &                        dBda,dBdh,dBdk,dBdp,dBdq,dBdL,
     &                        dlamadt,dlamhdt,dlamkdt,dlampdt,
     &                        dlamqdt,dlamLdt)
C
C.....................................................................
C. ROUTINE: COMP_EUL_LAG_VAR
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the derivatives of the Lagrange multipliers
C       with respect to time, i.e. multiplier rates.  This is done
C       by multiplying the partial derivatives of the BL matrix with
C       respect to the equinoctial elements, the normalized thrust
C       acceleration vector, the thrust acceleration magnitude and
C       the current values of the orbital elements.
C.
C.
C. CALLING SEQUENCE:
C          CALL COMP_EUL_LAG_VAR(z_vect,lam_vect,u,n,ft,G,r,
C                                dBda,dBdh,dBdk,dBdp,dBdq,dBdL,
C                                dlamadt,dlamhdt,dlamkdt,dlampdt,
C                                dlamqdt,dlamLdt
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER   I/O                   DESCRIPTION
C.          ------  ---  ---------------------------------------------
C.          z_vect   I   the 6x1 vector of equinoctial elements
C.          lam_vect I   the 6x1 vector of lagrange multipliers
C.          u        I   the 3x1 thrust vector
C.          n        I   mean motion
C.          ft       I   thrust acceleration magnitude
C.          G        I   auxiliary orbital element
C.          r        I   radial distance from central body to sat
C.          dBda     I   partials of B matrix wrt a
C.          dBdh     I   partials of B matrix wrt h
C.          dBdk     I   partials of B matrix wrt k
C.          dBdp     I   partials of B matrix wrt p
C.          dBdq     I   partials of B matrix wrt q
C.          dBdL     I   partials of B matrix wrt true long.
C.          dlamadt  O   partials of Lagrange mult for a wrt time
C.          dlamhdt  O   partials of Lagrange mult for h wrt time
C.          dlamkdt  O   partials of Lagrange mult for k wrt time
C.          dlampdt  O   partials of Lagrange mult for p wrt time
C.          dlamqdt  O   partials of Lagrange mult for q wrt time
C.          dlamLdt  O   partials of Lagrange mult for L wrt time
C.
C.
C. ROUTINES REQUIRED:  NONE
C.
C.....................................................................
C
```

```
C***************** DECLARATIONS ************************************
C
C
C
      DOUBLE PRECISION z_vect(6),lam_vect(6),u(3),n,ft,G,r
      DOUBLE PRECISION dBda(6,3),dBdh(6,3),dBdk(6,3),dBdp(6,3)
      DOUBLE PRECISION dBdq(6,3),dBdL(6,3)
      DOUBLE PRECISION dlamadt,dlamhdt,dlamkdt,dlampdt
      DOUBLE PRECISION dlamqdt,dlamLdt
      DOUBLE PRECISION cL,sL,a,h,k,L,drdh,drdk,drdL
      DOUBLE PRECISION lama,lamh,lamk,lamp,lamq,lamL

      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      L = z_vect(6)

      cL = cos(L)
      sL = sin(L)

      drdh = -r*(2.0*a*h + r*sL)/(a*(1.0 - h**2.0 - k**2.0))
      drdk = -r*(2.0*a*k + r*cL)/(a*(1.0 - h**2.0 - k**2.0))
      drdL = -((r**2.0)*(h*cL - k*sL))/(a*(1.0 - h**2.0 - k**2.0))

      lama = lam_vect(1)
      lamh = lam_vect(2)
      lamk = lam_vect(3)
      lamp = lam_vect(4)
      lamq = lam_vect(5)
      lamL = lam_vect(6)

      dlamadt = ft*(-lama*(dBda(1,1)*u(1)+dBda(1,2)*u(2)+dBda(1,3)*u(3))
     &       + -lamh*(dBda(2,1)*u(1)+dBda(2,2)*u(2)+dBda(2,3)*u(3))
     &       + -lamk*(dBda(3,1)*u(1)+dBda(3,2)*u(2)+dBda(3,3)*u(3))
     &       + -lamp*(dBda(4,1)*u(1)+dBda(4,2)*u(2)+dBda(4,3)*u(3))
     &       + -lamq*(dBda(5,1)*u(1)+dBda(5,2)*u(2)+dBda(5,3)*u(3))
     &       + -lamL*(dBda(6,1)*u(1)+dBda(6,2)*u(2)+dBda(6,3)*u(3)))
     &       - lamL*(-(3.0/2.0)*n*a*(r**-2.0)*G)

      dlamhdt = ft*(-lama*(dBdh(1,1)*u(1)+dBdh(1,2)*u(2)+dBdh(1,3)*u(3))
     &       + -lamh*(dBdh(2,1)*u(1)+dBdh(2,2)*u(2)+dBdh(2,3)*u(3))
     &       + -lamk*(dBdh(3,1)*u(1)+dBdh(3,2)*u(2)+dBdh(3,3)*u(3))
     &       + -lamp*(dBdh(4,1)*u(1)+dBdh(4,2)*u(2)+dBdh(4,3)*u(3))
     &       + -lamq*(dBdh(5,1)*u(1)+dBdh(5,2)*u(2)+dBdh(5,3)*u(3))
     &       + -lamL*(dBdh(6,1)*u(1)+dBdh(6,2)*u(2)+dBdh(6,3)*u(3)))
     &       - lamL*(-2.0*n*(a**2.0)*(r**-3.0)*G*drdh -
     &              n*(a**2.0)*(r**-2.0)*h*(G**-1.0))

      dlamkdt = ft*(-lama*(dBdk(1,1)*u(1)+dBdk(1,2)*u(2)+dBdk(1,3)*u(3))
     &       + -lamh*(dBdk(2,1)*u(1)+dBdk(2,2)*u(2)+dBdk(2,3)*u(3))
     &       + -lamk*(dBdk(3,1)*u(1)+dBdk(3,2)*u(2)+dBdk(3,3)*u(3))
     &       + -lamp*(dBdk(4,1)*u(1)+dBdk(4,2)*u(2)+dBdk(4,3)*u(3))
     &       + -lamq*(dBdk(5,1)*u(1)+dBdk(5,2)*u(2)+dBdk(5,3)*u(3))
     &       + -lamL*(dBdk(6,1)*u(1)+dBdk(6,2)*u(2)+dBdk(6,3)*u(3)))
     &       - lamL*(-2.0*n*(a**2.0)*(r**-3.0)*G*drdk -
     &              n*(a**2.0)*(r**-2.0)*k*(G**-1.0))

      dlampdt = ft*(-lama*(dBdp(1,1)*u(1)+dBdp(1,2)*u(2)+dBdp(1,3)*u(3))
     &       + -lamh*(dBdp(2,1)*u(1)+dBdp(2,2)*u(2)+dBdp(2,3)*u(3))
     &       + -lamk*(dBdp(3,1)*u(1)+dBdp(3,2)*u(2)+dBdp(3,3)*u(3))
     &       + -lamp*(dBdp(4,1)*u(1)+dBdp(4,2)*u(2)+dBdp(4,3)*u(3))
     &       + -lamq*(dBdp(5,1)*u(1)+dBdp(5,2)*u(2)+dBdp(5,3)*u(3))
     &       + -lamL*(dBdp(6,1)*u(1)+dBdp(6,2)*u(2)+dBdp(6,3)*u(3)))

      dlamqdt = ft*(-lama*(dBdq(1,1)*u(1)+dBdq(1,2)*u(2)+dBdq(1,3)*u(3))
     &       + -lamh*(dBdq(2,1)*u(1)+dBdq(2,2)*u(2)+dBdq(2,3)*u(3))
     &       + -lamk*(dBdq(3,1)*u(1)+dBdq(3,2)*u(2)+dBdq(3,3)*u(3))
     &       + -lamp*(dBdq(4,1)*u(1)+dBdq(4,2)*u(2)+dBdq(4,3)*u(3))
     &       + -lamq*(dBdq(5,1)*u(1)+dBdq(5,2)*u(2)+dBdq(5,3)*u(3))
     &       + -lamL*(dBdq(6,1)*u(1)+dBdq(6,2)*u(2)+dBdq(6,3)*u(3)))
```

```
 dlamLdt = ft*(-lama*(dBdL(1,1)*u(1)+dBdL(1,2)*u(2)+dBdL(1,3)*u(3))
&       + -lamh*(dBdL(2,1)*u(1)+dBdL(2,2)*u(2)+dBdL(2,3)*u(3))
&       + -lamk*(dBdL(3,1)*u(1)+dBdL(3,2)*u(2)+dBdL(3,3)*u(3))
&       + -lamp*(dBdL(4,1)*u(1)+dBdL(4,2)*u(2)+dBdL(4,3)*u(3))
&       + -lamq*(dBdL(5,1)*u(1)+dBdL(5,2)*u(2)+dBdL(5,3)*u(3))
&       + -lamL*(dBdL(6,1)*u(1)+dBdL(6,2)*u(2)+dBdL(6,3)*u(3)))
&       - lamL*(-2.0*n*(a**2.0)*(r**-3.0)*G*drdL)

 RETURN
 END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: COMP_U.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      SUBROUTINE COMP_U(lam_vect,B,u,u_norm)
C
C.....................................................................
C. ROUTINE: COMP_U
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: computes the normalized thrust acceleration vector
C.     given the 6x3 BL matrix and the vector of current Lagrange multipliers.
C.
C.
C. CALLING SEQUENCE:
C          CALL COMP_U(lam_vect,B,u,u_norm)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER     I/O                   DESCRIPTION
C.          --------  ---  ------------------------------------------
C.          lam_vect  I    6x1 input vector of lagrange multipliers
C.          B         I    6x3 input matrix of partial derivatives
C.          u         O    3x1 normalized output thrust vector
C.          u_norm    O    scalar magnitude of thrust acceleration
C.
C. ROUTINES REQUIRED:  NONE
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      DOUBLE PRECISION lam_vect(6), B(6,3), u(3)
      DOUBLE PRECISION u_unnorm(3), u_norm

      u_unnorm(1) = lam_vect(1)*B(1,1) +
     &              lam_vect(2)*B(2,1) +
     &              lam_vect(3)*B(3,1) +
     &              lam_vect(4)*B(4,1) +
     &              lam_vect(5)*B(5,1) +
     &              lam_vect(6)*B(6,1)

      u_unnorm(2) = lam_vect(1)*B(1,2) +
     &              lam_vect(2)*B(2,2) +
     &              lam_vect(3)*B(3,2) +
     &              lam_vect(4)*B(4,2) +
     &              lam_vect(5)*B(5,2) +
     &              lam_vect(6)*B(6,2)

      u_unnorm(3) = lam_vect(1)*B(1,3) +
     &              lam_vect(2)*B(2,3) +
     &              lam_vect(3)*B(3,3) +
     &              lam_vect(4)*B(4,3) +
     &              lam_vect(5)*B(5,3) +
```

```
&              lam_vect(6)*B(6,3)

 u_norm = DSQRT(u_unnorm(1)**2.0 +
&               u_unnorm(2)**2.0 +
&               u_unnorm(3)**2.0)

 u(1) = (1.0/u_norm)*u_unnorm(1)
 u(2) = (1.0/u_norm)*u_unnorm(2)
 u(3) = (1.0/u_norm)*u_unnorm(3)

 RETURN
 END
```

```
C -------------------------------------------------------------------------
C
C   FILE NAME: COMP_XY.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------
C
      SUBROUTINE COMP_XY(z_vect,mu,n,cL,sL,G,r,K1)
C
C.....................................................................
C. ROUTINE: COMP_XY
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Calculates auxiliary quantities based on
C.    the current equinoctial orbital elements.
C.
C.
C. CALLING SEQUENCE:
C          CALL COMP_XY(z_vect,mu,n,cL,sL,G,r,K1)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER     I/O                   DESCRIPTION
C.          --------  ---  ------------------------------------------
C.          z_vect    I    the 6x1 vector of equinoctial elements
C.          mu        I    the central body gravitational constant
C.          n         O    mean motion
C.          cL        O    cosine of the true longitude
C.          sL        O    sine of the true longitude
C.          G         O    auxiliary parameter based on h,k
C.          K1        O    auxiliary parameter based on p,q
C.
C. ROUTINES REQUIRED:  NONE
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      DOUBLE PRECISION z_vect(6), mu, n, cL, sL
      DOUBLE PRECISION a,h,k,p,q,L,G,r,K1,Beta

      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      L = z_vect(6)
      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0 + G)
      cL = DCOS(L)
      sL = DSIN(L)
      n = DSQRT(mu)*(a**(-3.0/2.0))
      r = a*(1 - h**2.0 - k**2.0)/(1.0 + h*sL + k*cL)
      K1 = 1.0 + p**2.0 + q**2.0

      RETURN
      END
```

```
C -------------------------------------------------------------------------
C
C   FILE NAME: TRANS_OUT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------
C
      SUBROUTINE TRANS_OUT(Y_IN, Y_OUT, ft, mu)
C
C.....................................................................
C. ROUTINE: TRANS_OUT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Transforms the equinoctial elements into Keplerian
C.     elements and calls COMP_XY, COMP_M, COMP_U and FSUB to compute
C.     the Hamiltonian, thrust vector, and the yaw and pitch angles.
C.     These quantities are returned to the calling subroutine in an
C.     array intended to be written as output.
C.
C.
C. CALLING SEQUENCE:
C          CALL TRANS_OUT(Y_IN, Y_OUT, ft, mu)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ------------------------------------
C.          Y_IN       I    12x1 vector of equinoctial elements and
C.                          lagrange multipliers
C.          Y_OUT      O    13x1 vector of desired output quantities
C.          ft         I    thrust acceleration magnitude
C.          mu         I    central body gravitational constant
C.
C.
C. ROUTINES REQUIRED:  COMP_XY, COMP_U, COMP_B
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      DOUBLE PRECISION Y_IN(12), Y_OUT(13), ft, mu
      DOUBLE PRECISION a,h,k,p,q,L,lam(6),z_vect(6)
      DOUBLE PRECISION ecc,inc,ran,arp,mea
      DOUBLE PRECISION u(3),Ham,u_r,u_t,u_h
      DOUBLE PRECISION B(6,3),dBda(6,3),dBdh(6,3)
      DOUBLE PRECISION dBdk(6,3),dBdp(6,3),dBdq(6,3)
      DOUBLE PRECISION dBdL(6,3), B_u(6), lam_B_u, u_mag
      DOUBLE PRECISION nm,cL,sL,G,r,K1,theta_t,theta_h
      DOUBLE PRECISION u_eci(3), ROTL(3,3), ROTRI(3,3), FGW(3)

      a = Y_IN(1)
      h = Y_IN(2)
      k = Y_IN(3)
      p = Y_IN(4)
      q = Y_IN(5)
      L = Y_IN(6)
```

```
      z_vect(1) = Y_IN(1)
      z_vect(2) = Y_IN(2)
      z_vect(3) = Y_IN(3)
      z_vect(4) = Y_IN(4)
      z_vect(5) = Y_IN(5)
      z_vect(6) = Y_IN(6)

      lam(1) = Y_IN(7)
      lam(2) = Y_IN(8)
      lam(3) = Y_IN(9)
      lam(4) = Y_IN(10)
      lam(5) = Y_IN(11)
      lam(6) = Y_IN(12)

      ecc = (h**2.0 + k**2.0)**(1.0/2.0)
      inc = 2.0*DATAN2((p**2.0 + q**2.0)**(1.0/2.0),1.0)
      ran = DATAN2(p,q)
      arp = DATAN2(h,k) - DATAN2(p,q)
      mea = L - DATAN2(h,k)

      CALL COMP_XY(z_vect,mu,nm,cL,sL,G,r,K1)

      CALL COMP_B(z_vect,nm,cL,sL,G,r,K1,
     &            B,dBda,dBdh,dBdk,dBdp,dBdq,dBdL)

      CALL COMP_U(lam,B,u,u_mag)

      B_u(1) = B(1,1)*u(1)+B(1,2)*u(2)+B(1,3)*u(3)
      B_u(2) = B(2,1)*u(1)+B(2,2)*u(2)+B(2,3)*u(3)
      B_u(3) = B(3,1)*u(1)+B(3,2)*u(2)+B(3,3)*u(3)
      B_u(4) = B(4,1)*u(1)+B(4,2)*u(2)+B(4,3)*u(3)
      B_u(5) = B(5,1)*u(1)+B(5,2)*u(2)+B(5,3)*u(3)
      B_u(6) = B(6,1)*u(1)+B(6,2)*u(2)+B(6,3)*u(3)

      lam_B_u = lam(1)*B_u(1)+lam(2)*B_u(2)+lam(3)*B_u(3)+
     &          lam(4)*B_u(4)+lam(5)*B_u(5)+lam(6)*B_u(6)

      Ham = ft*lam_B_u + lam(6)*(a**2.0)*nm*
     &      ((1.0-h**2.0-k**2.0)**(1.0/2.0))/(r**2.0)

      u_r = u(1)
      u_t = u(2)
      u_h = u(3)
C
C     Compute the pitch and yaw angles
C
      theta_t = DATAN2(u_r,u_t)
      theta_h = DATAN2(u_h,u_t)
C
C     Rotate the acceleration into the equinoctial frame
C
      ROTL(1,1) = cL
      ROTL(1,2) = -sL
      ROTL(1,3) = 0.0
      ROTL(2,1) = sL
      ROTL(2,2) = cL
      ROTL(2,3) = 0.0
      ROTL(3,1) = 0.0
      ROTL(3,2) = 0.0
      ROTL(3,3) = 1.0

      FGW(1) = ROTL(1,1)*u_r+ROTL(1,2)*u_t+ROTL(1,3)*u_h
      FGW(2) = ROTL(2,1)*u_r+ROTL(2,2)*u_t+ROTL(2,3)*u_h
      FGW(3) = ROTL(3,1)*u_r+ROTL(3,2)*u_t+ROTL(3,3)*u_h
C
C     Now rotate the equinoctial acceleration to the inertial cartesian frame
C
      ROTRI(1,1) = (DCOS(-ran))**2.0 + DCOS(-inc)*((DSIN(-ran))**2.0)
      ROTRI(1,2) = DCOS(-ran)*DSIN(-ran)*(1-DCOS(-inc))
      ROTRI(1,3) = -DSIN(-inc)*DSIN(-ran)
      ROTRI(2,1) = DCOS(-ran)*DSIN(-ran)*(1-DCOS(-inc))
```

```
      ROTRI(2,2) = (DSIN(-ran))**2.0 + DCOS(-inc)*((DCOS(-ran))**2.0)
      ROTRI(2,3) = DSIN(-inc)*DCOS(-ran)
      ROTRI(3,1) = DSIN(-inc)*DSIN(-ran)
      ROTRI(3,2) = -DSIN(-inc)*DCOS(-ran)
      ROTRI(3,3) = DCOS(-inc)
C
C     Multiply the rotation matrix by the equinoctial
C     vector to get the ECI vector
C
      u_eci(1) = ROTRI(1,1)*FGW(1)+ROTRI(1,2)*FGW(2)+ROTRI(1,3)*FGW(3)
      u_eci(2) = ROTRI(2,1)*FGW(1)+ROTRI(2,2)*FGW(2)+ROTRI(2,3)*FGW(3)
      u_eci(3) = ROTRI(3,1)*FGW(1)+ROTRI(3,2)*FGW(2)+ROTRI(3,3)*FGW(3)
C
C     Assemble the output
C
      Y_OUT(1) = a
      Y_OUT(2) = ecc
      Y_OUT(3) = inc
      Y_OUT(4) = ran
      Y_OUT(5) = arp
      Y_OUT(6) = mea
      Y_OUT(7) = Ham
      Y_OUT(8) = u_eci(1)
      Y_OUT(9) = u_eci(2)
      Y_OUT(10) = u_eci(3)
      Y_OUT(11) = u_mag
      Y_OUT(12) = theta_t
      Y_OUT(13) = theta_h

      RETURN
      END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: SOLVE_ECC_ANOMALY.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION SOLVE_ECC_ANOMALY
     &                          (mean_anomaly_in, ecc_in)
C
C.....................................................................
C. ROUTINE: SOLVE_ECC_ANOMALY
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Calculates the eccentric anomaly based on the mean anomaly and
C.          eccentricity.
C.
C.
C. CALLING SEQUENCE:
C          ECC_ANOM = SOLVE_ECC_ANOMALY(mean_anomaly_in, ecc_in)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER           I/O                  DESCRIPTION
C.          ---------------  ---  ------------------------------------
C.          mean_anomaly_in  I    mean anomaly (radians)
C.          ecc_in           I    eccentricity
C.
C.
C. ROUTINES REQUIRED:  NONE
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      IMPLICIT NONE
      INTEGER max_iter, iter
      DOUBLE PRECISION mean_anomaly_in, ecc_in, epsilon
      DOUBLE PRECISION ecc_anomaly, ecc_anomaly_new
      DOUBLE PRECISION ecc_anomaly_diff
      epsilon = 1.0E-20
      max_iter = 50
      ecc_anomaly = epsilon + 1.0
      ecc_anomaly_new = 0.0
      iter = 0
      DO WHILE (DABS(ecc_anomaly - ecc_anomaly_new) .GT.
     &          epsilon .AND. iter .LE. max_iter)

         ecc_anomaly = ecc_anomaly_new
         ecc_anomaly_new = mean_anomaly_in + ecc_in * DSIN(ecc_anomaly)
         iter = iter+1

      END DO
      ecc_anomaly_diff = DABS(ecc_anomaly - ecc_anomaly_new)
      SOLVE_ECC_ANOMALY = ecc_anomaly_new
      RETURN
      END
```

Appendix F Source code for the Averaged Equation Optimal Thrust Planning Software

The exact equation optimal thrust planning software is described in Chapter 5 section 5.1.2. This appendix contains the source code corresponding to section 5.1.2. Only the source code written by the author is included. Other open source subroutines such as the UNCMND, DQAG, and RK78 subroutines are not included. Sources for those subroutines can be found in the References section or by contacting the author.

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: low_thrust_drive.for   (for averaged equation software)
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
C.....................................................................
C. ROUTINE: LOW_THRUST_DRIVE
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE:
C.
C.    This the driver subroutine for the averaged eq. software.
C.    It collects the initial and final Keplerian orbits, converts
C.    those to equinoctial orbits, calls the UNCMND subroutine to
C.    execute the quasi-Newton search to solve for the initial Lagrange
C.    multipliers, and calls the RK78 subroutine to integrate the
C.    variational equations of motion and the variational equations
C.    for the Lagrange multipliers from the initial to final time.
C.    Finally, the trajectory is printed.
C.
C. CALLING SEQUENCE:
C.          This is a main program and has no calling parameters.  However,
C.          Several of the initial variable values can be modified to solve
C.          averaged optimal thrust trajectory problems.  Among these are:
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER   I/O                   DESCRIPTION
C.          ------  ---  ------------------------------------------
C.          tf0     I    the guess for the final time in seconds.
C.          ft      I    the constrant thrust acceleration km/second squared
C.          sma0    I    the semimajor axis for the initial orbit
C.          ecc0    I    the eccentricity of the initial orbit
C.          inc0    I    the inclination of the initial orbit
```

```
C.          ran0    I    the RAAN of the initial orbit
C.          arp0    I    the arg. of perigee of the initial orbit
C.          mea0    I    the mean anomaly of the initial orbit
C.          smaF    I    the semimajor axis for the final orbit
C.          eccF    I    the eccentricity of the final orbit
C.          incF    I    the inclination of the final orbit
C.          ranF    I    the RAAN of the final orbit
C.          arpF    I    the arg. of perigee of the final orbit
C.          meaF    I    the mean anomaly of the final orbit
C.          lam_vect I   the 6x1 vector of initial Lagrange multipliers
C.
C.
C. ROUTINES REQUIRED: UNCMND, RK78, TRANS_OUT, DELTIM, ADDTIM
C.
C.....................................................................
C
C
C***************** DECLARATIONS ***************************************
C
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      PARAMETER (N_INT=12, N_MIN=7,
     *           LWORK=N_MIN*(N_MIN+10))
      DOUBLE PRECISION  Y(N_INT), TOL
      DOUBLE PRECISION  T, DT, TDIFF

      DOUBLE PRECISION mu, ft, pi
      DOUBLE PRECISION sma0, ecc0, inc0, ran0, arp0, mea0
      DOUBLE PRECISION tf0, tf, LAST_PRINT
      DOUBLE PRECISION smaF, eccF, incF, ranF, arpF, meaF
      DOUBLE PRECISION z0_vect(6), zF_vect(6), lam_vect(7)
      DOUBLE PRECISION z0, weights(7,20)
      DOUBLE PRECISION XDUM(N_INT), F1(N_INT), F2(N_INT), F3(N_INT)
      DOUBLE PRECISION F4(N_INT), F5(N_INT), F6(N_INT), F7(N_INT)
      DOUBLE PRECISION Y_OUT(10), Y_FIN_DIFF(6)
      DOUBLE PRECISION x(N_MIN), x0(N_MIN)
      DOUBLE PRECISION WORK(LWORK), F, EXTDAT, WEIGHT

      INTEGER   I, J, IFLAG
      INTEGER   MAX_ITER, N_INT, N_MIN, LWORK
      INTEGER   IERROR, iter
      EXTERNAL  FSUB,F_FORMIN

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /WEIGHT/ weights, iter
C
C     Set PI
C
      pi = 3.141592653589793
C
C     Set the integration error tolerance
C
      TOL = 1.0E6
C
C     Guess for the final time
C
C     tf0 = 1761276.2904
C
      tf0 = 1761276.2904
C
C     Set the Earth gravity constant (km^3/days^2)
C
      mu = 398600.4418
C
C     Set the constant acceleration (km/days^2)
C
      ft = 6.5E-9
C
C     Set the maximum iterations for the outer optimization loop
C
```

```
      MAX_ITER = 5
C
C     Set the initial Keplerian elements
C
      sma0 = 41532.10828
      ecc0 = 0.9029821557E-3
      inc0 = 1.734151104 * pi/180.0
      ran0 = 109.0647352 * pi/180.0
      arp0 = 39.26363404 * pi/180.0
      mea0 = 354.0614630 * pi/180.0
C
C     Set the final Keplerian elements
C
      smaF = 41840.20862
      eccF = 0.6335377066E-3
      incF = 1.734151104 * pi/180
      ranF = 109.0647352 * pi/180
      arpF = 39.26363404 * pi/180
      meaF = 217.6227012 * pi/180
C
C     Compute the initial equinoctial elements
C
      z0_vect(1) = sma0
      z0_vect(2) = ecc0 * sin(arp0 + ran0)
      z0_vect(3) = ecc0 * cos(arp0 + ran0)
      z0_vect(4) = tan(inc0/2)*sin(ran0)
      z0_vect(5) = tan(inc0/2)*cos(ran0)
      z0_vect(6) = mea0 + arp0 + ran0
C
C     Compute the final equinoctial elements
C
      zF_vect(1) = smaF
      zF_vect(2) = eccF * sin(arpF + ranF)
      zF_vect(3) = eccF * cos(arpF + ranF)
      zF_vect(4) = tan(incF/2)*sin(ranF)
      zF_vect(5) = tan(incF/2)*cos(ranF)
      zF_vect(6) = meaF + arpF + ranF
C
C     Initial guesses for the Lagrange multipliers
C
      lam_vect(1) = 0.570932641360E+04
      lam_vect(2) = -0.344890990023E+08
      lam_vect(3) = 0.559045506405E+08
      lam_vect(4) = 0.198345700960E+02
      lam_vect(5) = 0.287829595367E+03
      lam_vect(6) = 0.764058554639E-02

      lam_vect(7) = tf0

      DO J=1,7
         x0(J) = lam_vect(J)
      END DO
C
C     Combine all initial conditions into Y array
C
      DO I=1,6
         Y(I) = z0_vect(I)
      END DO
      DO I=1,6
         Y(I+6) = lam_vect(I)
      END DO
C
C     Set initial eccentric longitude
C
      T = 0.0
      DT = 0.1
      LAST_PRINT = T
C
C     Set the weights for each outer loop optimization iteration
C
      weights(1,1) = 1.0
```

```
      weights(2,1) = 1.0E12
      weights(3,1) = 1.0E12
      weights(4,1) = 1.0E9
      weights(5,1) = 1.0E9
      weights(6,1) = 1.0
      weights(7,1) = 1.0E6

      weights(1,2) = 1.0/10.0
      weights(2,2) = 1.0E11
      weights(3,2) = 1.0E11
      weights(4,2) = 1.0E10
      weights(5,2) = 1.0E10
      weights(6,2) = 1.0
      weights(7,2) = 1.0E6

      weights(1,3) = 1.0
      weights(2,3) = 1.0E12
      weights(3,3) = 1.0E12
      weights(4,3) = 1.0E11
      weights(5,3) = 1.0E11
      weights(6,3) = 1.0
      weights(7,3) = 1.0E6

      weights(1,4) = 1.0E1
      weights(2,4) = 1.0E13
      weights(3,4) = 1.0E13
      weights(4,4) = 1.0E12
      weights(5,4) = 1.0E11
      weights(6,4) = 1.0
      weights(7,4) = 1.0E6

      weights(1,5) = 1.0E2
      weights(2,5) = 1.0E12
      weights(3,5) = 1.0E12
      weights(4,5) = 1.0E11
      weights(5,5) = 1.0E11
      weights(6,5) = 1.0
      weights(7,5) = 1.0E6
C
C     Save the initial values in x in case we are skipping the UNCMND
C
      DO I=1,7
         x(I) = lam_vect(I)
      END DO
C
C     Set up the outer optimization loop
C
      DO iter=1,MAX_ITER
C
C        Call the unconstrained minimization subroutine
C
         CALL UNCMND (N_MIN, x0, F_FORMIN, x, F, IERROR, WORK, LWORK)
C
C        Print out the results of the minimization (the Lagrange multipliers and final
time)
C
         WRITE (*,*) 'Results of opt: Lagrange mult and final time'
         WRITE (*,'(7E24.12)') (x(I), I=1,7)
C
C        Copy the the output back to the input
C
         DO J=1,7
            x0(J) = x(J)
         END DO
      END DO
C
C     Assign results of optimization to input for integration
C     and printout of the final trajectory
C
      DO I=1,6
         Y(I) = z0_vect(I)
```

```
      END DO
      Y(7)  = x(1)
      Y(8)  = x(2)
      Y(9)  = x(3)
      Y(10) = x(4)
      Y(11) = x(5)
      Y(12) = x(6)
      tf    = x(7)
C
C     Print out the results of the minimization (the Lagrange multipliers and final time)
C
      WRITE (*,*) 'Results of optimization Lagrange mult and final time'
      WRITE (*,'(7E24.12)') (x(I), I=1,7)
C
C     Output the final trajectory result of the optimization
C
      DO WHILE (T .LE. tf)
C
C        Integrate.
C
         CALL RK78 (IFLAG,N_INT,T,DT,Y,TOL,
     &              XDUM,F1,F2,F3,F4,F5,F6,F7,FSUB)
C
C        Are we finished?  If so, exit the loop.
C
         IF (T .EQ. tf) THEN
C
C           Write the output at this time step.
C
            CALL TRANS_OUT(Y, Y_OUT, ft, mu)
C
C           Compare the desired elements with the final elements achieved.
C
            Y_FIN_DIFF(1) = Y_OUT(1) - smaF
            Y_FIN_DIFF(2) = Y_OUT(2) - eccF
            Y_FIN_DIFF(3) = (Y_OUT(3) - incF)*180.0/pi
            Y_FIN_DIFF(4) = (Y_OUT(4) - ranF)*180.0/pi
            Y_FIN_DIFF(5) = (Y_OUT(5) - arpF)*180.0/pi
            Y_FIN_DIFF(6) = (Y_OUT(6) - meaF)*180.0/pi

            WRITE (*,*) 'Final element differences'
            WRITE (*,'(6E24.12)') Y_FIN_DIFF(1),Y_FIN_DIFF(2),
     &          Y_FIN_DIFF(3),Y_FIN_DIFF(4),Y_FIN_DIFF(5),Y_FIN_DIFF(6)
C
C
C        We are not yet finished, find the time yet to integrate.
C        If that time is less than the next time step, reduce the
C        next time step to equal the time left to integrate.
C
         ELSE
            TDIFF = tf - T
            IF (TDIFF .LT. DT) THEN
               DT = TDIFF
            END IF
         END IF
C
C        Write the output at this time step if enough time has passed.
C
         IF (T - LAST_PRINT .GE. 1.0E-20) THEN

            CALL TRANS_OUT(Y, Y_OUT, ft, mu)

            WRITE (*,'(I9,12E24.12)') 111111111,
     *              T,Y_OUT(1),Y_OUT(2),Y_OUT(3),Y_OUT(4),Y_OUT(5),
     *                Y_OUT(6),Y_OUT(7),Y_OUT(8),Y_OUT(9),Y_OUT(10)

            LAST_PRINT = T
         END IF

         IF (T .EQ. tf) THEN
            EXIT
```

```
   END IF
END DO
END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: fsub.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      SUBROUTINE FSUB (T,Y,YDOT)
C
C.....................................................................
C. ROUTINE: FSUB
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: FSUB is the subroutine that is called by
C.   the FK78 subroutine to supply the equinoctial element and
C.   Lagrange multiplier derivatives with respect to time, i.e. rates.
C.   FSUB executes the COMP_EQUIN_VAR and COMP_EUL_LAG_VAR subroutines
C.   which compute the rates for the equinoctial variation equations and the
C.   rates for the Lagrange multipliers, respectively.
C.
C.
C. CALLING SEQUENCE:
C.          FSUB(T,Y,YDOT)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER   I/O                   DESCRIPTION
C.          ------  ---  ---------------------------------------------
C.          T       I    The current time in seconds from time zero
C.          Y       I    The input vector of current equinoctial orbital
C.                       elements in elements 1-6 and the vector of
C.                       current lagrange multipliers in elements 7-12.
C.          YDOT    O    The output vector of equinoctial element rates in
C.                       elements 1-6 and the output vector of lagrange
C.                       multipliers in elements 7-12.
C.
C. ROUTINES REQUIRED: COMP_EQUIN_VAR,
C.                    COMP_EUL_LAG_VAR
C.
C.....................................................................
C
C.
C
C
C***************** DECLARATIONS ***************************************
C
C
C       Routine for evaluating right hand sides of equations.
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      INTEGER N

      DOUBLE PRECISION T, Y(*), YDOT(*)
      DOUBLE PRECISION mu, ft
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION z0_vect(6), zF_vect(6), EXTDAT
      DOUBLE PRECISION nm, cF, sF, G, r, K1, X1, Xdot1
      DOUBLE PRECISION Y1, Ydot1
```

```
      DOUBLE PRECISION M(6,3), dMda(6,3), dMdh(6,3), dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3), dMdq(6,3), dMdl(6,3)
      DOUBLE PRECISION u(3), dadt, dhdt, dkdt, dpdt, dqdt, dldt
      DOUBLE PRECISION dlamadt, dlamhdt, dlamkdt
      DOUBLE PRECISION dlampdt, dlamqdt, dlamldt

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
C
C     Assign the input arrays
C
      z_vect(1) = Y(1)
      z_vect(2) = Y(2)
      z_vect(3) = Y(3)
      z_vect(4) = Y(4)
      z_vect(5) = Y(5)
      z_vect(6) = Y(6)

      lam_vect(1) = Y(7)
      lam_vect(2) = Y(8)
      lam_vect(3) = Y(9)
      lam_vect(4) = Y(10)
      lam_vect(5) = Y(11)
      lam_vect(6) = Y(12)
C
C     Compute the right hand side of the equinoctial element variational equations
C
      CALL COMP_EQUIN_VAR(z_vect,lam_vect,
     &                    dadt,dhdt,dkdt,dpdt,dqdt,dldt)
C
C     Compute the right hand side of the Lagrange multiplier variational equations
C
      CALL COMP_EUL_LAG_VAR(z_vect,lam_vect,
     &                      dlamadt,dlamhdt,dlamkdt,dlampdt,
     &                      dlamqdt,dlamldt)
C
C     Assign the output rates
C
      YDOT(1) = dadt
      YDOT(2) = dhdt
      YDOT(3) = dkdt
      YDOT(4) = dpdt
      YDOT(5) = dqdt
      YDOT(6) = dldt
      YDOT(7) = dlamadt
      YDOT(8) = dlamhdt
      YDOT(9) = dlamkdt
      YDOT(10) = dlampdt
      YDOT(11) = dlamqdt
      YDOT(12) = dlamldt
      RETURN
      END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: F_FORMIN.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      SUBROUTINE F_FORMIN(N, X, F_OUT)
C
C.....................................................................
C. ROUTINE: F_FORMIN
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the equinoctial elements and Lagrange multipliers
C.    at the final time given the elements and multipliers at the initial
C.    time.  F_FORMIN also computes the sum of the squares of the differences
C.    of the computed final orbital element conditions from the desired orbital
C.    element conditions.  F_FORMIN uses the RK78 subroutine to perform the
C.    integration of the equinoctial orbital elements and the Lagrange multipliers.
C.    F_FORMIN is called by UNCMND to perform unconstrained minimization of the
C.    F cost function defined in F_FORMIN.
C.
C.
C. CALLING SEQUENCE:
C          CALL F_FORMIN(N, X, F_OUT)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ------------------------------------
C.          N          I    Number of parameters to vary in search
C.                          for minimum.  In this case it is the 6
C.                          Lagrange multipliers plus the final time
C.                          for a total of 7.
C.          X          I    vector of lagrange multipliers and tf
C.          F_OUT      O    The value of the cost function given X
C.
C. ROUTINES REQUIRED:  RK78, FSUB
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      PARAMETER(N_INT=12,N_MIN=7)
      INTEGER N_MIN, I, N_INT, IFLAG
      INTEGER MS, NROOT, MINT, LW, IW, LIW, iter
      DOUBLE PRECISION X(N_MIN), F_OUT
      DOUBLE PRECISION mu,ft,aF,hF,kF,pF,qF,lF,ecc,inc
      DOUBLE PRECISION z0_vect(6), zF_vect(6)
      DOUBLE PRECISION a,h,k,p,q,l,lam(6)
      DOUBLE PRECISION nm,cF,sF,G,r,K1
      DOUBLE PRECISION M(6,3),dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION u(3)
      DOUBLE PRECISION wgt(7)
```

```
      DOUBLE PRECISION tf, Y(N_INT+3)
      DOUBLE PRECISION T, DT, TOL, TDIFF
      DOUBLE PRECISION weights(7,20)
      DOUBLE PRECISION EXTDAT, WEIGHT
      DOUBLE PRECISION XDUM(N_INT), F1(N_INT), F2(N_INT), F3(N_INT)
      DOUBLE PRECISION F4(N_INT), F5(N_INT), F6(N_INT), F7(N_INT)
      DOUBLE PRECISION YDOT(12)

      EXTERNAL  FSUB,GFUN

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /WEIGHT/ weights, iter
C
C     Initialize values needed by the integrator
C
C
C     Set the integration error tolerance
C
      TOL = 1.0E6
C
C     Set initial time point
C
      T = 0.0
C
C     Copy the final time guess.
C
      tf = X(7)
C
C     Set the initial guess for integration.
C
      Y(1) = z0_vect(1)
      Y(2) = z0_vect(2)
      Y(3) = z0_vect(3)
      Y(4) = z0_vect(4)
      Y(5) = z0_vect(5)
      Y(6) = z0_vect(6)
      Y(7) = X(1)
      Y(8) = X(2)
      Y(9) = X(3)
      Y(10) = X(4)
      Y(11) = X(5)
      Y(12) = X(6)
C
C     Copy the final elements
C
      aF = zF_vect(1)
      hF = zF_vect(2)
      kF = zF_vect(3)
      pF = zF_vect(4)
      qF = zF_vect(5)
      lF = zF_vect(6)
C
C     We want the orbital and Lagrange multiplier
C     values only at the final time
C
      DT = 600.0
C
C     Return a large value for F_OUT if the time is out of bounds
C
      IF (tf .LE. 0.0) THEN
         F_OUT = 1.0E20
         RETURN
      END IF
C
C     Start the integration loop
C
      DO WHILE (T .LE. tf)
C
C        Integrate.
C
         CALL RK78 (IFLAG,N_INT,T,DT,Y,TOL,
```

```
     &              XDUM,F1,F2,F3,F4,F5,F6,F7,FSUB)
C
C        Are we finished?  If so, exit the loop.
C
         IF (T .EQ. tf) THEN
            EXIT
C
C        We are not yet finished, find the time yet to integrate.
C        If that time is less than the next time step, reduce the
C        next time step to equal the time left to integrate.
C
         ELSE
            TDIFF = tf - T
            IF (TDIFF .LT. DT) THEN
               DT = TDIFF
            END IF
         END IF

      END DO
C
C     If the integrator was happy, compute the function value
C
      IF (T .EQ. tf) THEN

         a = Y(1)
         h = Y(2)
         k = Y(3)
         p = Y(4)
         q = Y(5)
         l = Y(6)

         lam(1) = Y(7)
         lam(2) = Y(8)
         lam(3) = Y(9)
         lam(4) = Y(10)
         lam(5) = Y(11)
         lam(6) = Y(12)
C
C        Calculate the Hamiltonian
C
         CALL FSUB(T,Y,YDOT)

         Ham = lam(1)*YDOT(1)+lam(2)*YDOT(2)+lam(3)*YDOT(3)+
     &         lam(4)*YDOT(4)+lam(5)*YDOT(5)+lam(6)*YDOT(6)
C
C        Assign the weights
C
         DO I=1,7
            wgt(I) = weights(I,iter)
         END DO
C
C        This cost function is for Kechichian's LEO to GEO case
C
         F_OUT = wgt(1)*(a - aF)**2.0 + wgt(2)*(h - hF)**2.0 +
     &           wgt(3)*(k - kF)**2.0 + wgt(4)*(p - pF)**2.0 +
     &           wgt(5)*(q - qF)**2.0 +
     &           wgt(6)*(lam(6) - 0.0)**2.0 +
     &           wgt(7)*(Ham - 1.0)**2.0

         ecc = (h**2.0 + k**2.0)**(1.0/2.0)
         inc = 2.0*DATAN2((p**2.0 + q**2.0)**(1.0/2.0),1.0)

         WRITE (*,*) 'F_FORMIN output'

         WRITE (*,'(I3,6E16.7)')
     &         iter,F_OUT,Ham,tf,a,ecc,inc
         WRITE (*,'(I3,7E14.5)')
     &         iter,
     &         wgt(1)*((a - aF)**2.0),
     &         wgt(2)*((h - hF)**2.0),
     &         wgt(3)*((k - kF)**2.0),
```

```
     &         wgt(4)*((p - pF)**2.0),
     &         wgt(5)*((q - qF)**2.0),
     &         wgt(6)*((lam(6)- 0.0)**2.0),
     &         wgt(7)*((Ham - 1.0)**2.0)

C
C     If the integrator was unhappy, print a message and return
C
      ELSE
         WRITE (*,*) 'Error in integrating from initial to final time.'
         WRITE (*,*) 'integrated', T, 'seconds.'
         F_OUT = 0.0
         RETURN
      END IF

      RETURN
      END
```

```
C
C   FILE NAME: COMP_M.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      SUBROUTINE COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,n,cF,sF,G,Beta,r,K1,
     &                  M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C.....................................................................
C. ROUTINE: COMP_M
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the 6x3 M matrix and its partial
C.   derivatives with respect to the equinoctial elements.
C.   The equations for this subroutine can be found in the
C.   Appendix of [Kechichian, J. A., Optimal Low-Thrust Rendezvous
C.   Using Equinoctial Orbit Elements.  ACTA Astronautica. Vol. 38,
C.   No. 1, pp. 1-14, 1996].  According to Jean Kechichian, there is
C.   one small error in the partials in equation (A96).
C.   The term reading cF - h should read cF - k.
C.
C.
C. CALLING SEQUENCE:
C          CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,n,cF,sF,G,Beta,r,K1,
C                      M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ------------------------------------
C.          z_vect     I    The 6x1 vector of equinoctial elements
C.          X1         I    Cartesian X position magnitude
C.          Xdot1      I    Cartesian X velocity magnitude
C.          Y1         I    Cartesian Y position magnitude
C.          Ydot1      I    Cartesian Y velocity magnitude
C.          n          I    mean motion
C.          cF         I    cosine of eccentric longitude
C.          sF         I    sine of eccentric longitude
C.          G          I    auxiliary value based on h,k
C.          Beta       I    auxiliary value also based on h,k
C.          r          I    radial distance between sat & central body
C.          K1         I    auxiliary value based on p,q
C.          M          O    6x3 partial derivative matrix of equinoctial
C.                          elements wrt rdot
C.          dMda       O    6x3 partial derivative matrix of M wrt a
C.          dMdh       O    6x3 partial derivative matrix of M wrt h
C.          dMdk       O    6x3 partial derivative matrix of M wrt k
C.          dMdp       O    6x3 partial derivative matrix of M wrt p
C.          dMdq       O    6x3 partial derivative matrix of M wrt q
C.          dMdl       O    6x3 partial derivative matrix of M wrt mean long.
C.
C. ROUTINES REQUIRED:  NONE
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
```

```
C
      IMPLICIT NONE

      DOUBLE PRECISION z_vect(6),n,cF,sF,G,r,K1
      DOUBLE PRECISION a,h,k,p,q,l,X1,Xdot1,Y1,Ydot1,Beta
      DOUBLE PRECISION M(6,3),dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION drda,drdh,drdk,drdl,dnda
      DOUBLE PRECISION dX1dh,dX1dk,dY1dh,dY1dk
      DOUBLE PRECISION dXdot1dh,dXdot1dk,dYdot1dh,dYdot1dk
      DOUBLE PRECISION d2X1dhh,d2X1dkk,d2X1dhdk,d2X1dkdh
      DOUBLE PRECISION d2Y1dhh,d2Y1dkk,d2Y1dhdk,d2Y1dkdh
      DOUBLE PRECISION d2X1dadk,d2X1dadh,d2Y1dadk,d2Y1dadh
      DOUBLE PRECISION dX1dF,dY1dF,dXdot1dF,dYdot1dF,d2X1dFdh
      DOUBLE PRECISION d2X1dFdk,d2Y1dFdh,d2Y1dFdk,dXdot1da,dYdot1da
      DOUBLE PRECISION dX1da,dY1da
C
C     These partial derivatives taken from Kechichian:
C        "Trajectory Optimization Using Nonsingular
C         Orbital Elements and True Longitude."

      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)
C
C     Compute partials of X1, Y1, Xdot1, Ydot1 wrt h and k
C
      dX1dh = a*(-(h*cF-k*sF)*(Beta+((h**2.0)*(Beta**3.0))/(1.0-Beta))-
     &        (a/r)*cF*(h*Beta-sF))
      dX1dk = -a*((h*cF-k*sF)*(h*k*(Beta**3.0))/(1.0-Beta) +
     &        1.0 + (a/r)*sF*(sF-h*Beta))
      dY1dh = a*((h*cF-k*sF)*(h*k*(Beta**3.0))/(1.0-Beta) -
     &        1.0 + (a/r)*cF*(k*Beta-cF))
      dY1dk = a*((h*cF-k*sF)*(Beta+((k**2.0)*(Beta**3.0))/(1.0-Beta)) +
     &        (a/r)*sF*(cF-k*Beta))

      dXdot1dh = (a/r)*Xdot1*(sF+(a/r)*cF*(k*sF - h*cF)) +
     &           ((n*(a**2.0))/r)*(h*Beta*sF + (k*cF + h*sF)*(Beta +
     &           ((h**2.0)*((Beta**3.0)))/(1-Beta)) +
     &           (a/r)*cF*(h*k*Beta*sF + (1-Beta*(h**2.0))*cF))

      dXdot1dk = -(a/r)*Xdot1*(-cF + (a/r)*sF*(k*sF - h*cF)) +
     &           ((n*(a**2.0))/r)*(((h*k*(Beta**3.0))/(1.0-Beta))*
     &           (k*cF+h*sF) + h*Beta*cF -
     &           (a/r)*sF*(h*k*Beta*sF + (1.0-Beta*(h**2.0))*cF))

      dYdot1dh = -(a/r)*Ydot1*(-sF - (a/r)*cF*(k*sF-h*cF)) +
     &           ((n*(a**2.0))/r)*(-((h*k*(Beta**3.0))/(1.0-Beta))*
     &           (k*cF+h*sF) - k*Beta*sF +
     &           (a/r)*cF*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF))

      dYdot1dk = -(a/r)*Ydot1*(-cF + (a/r)*sF*(k*sF - h*cF)) +
     &          ((n*(a**2.0))/r)*(-(Beta + ((k**2.0)*(Beta**3.0))/
     &          (1.0-Beta))*(k*cF+h*sF) - k*Beta*cF -
     &          (a/r)*sF*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF))

      d2X1dhh = a*(-(2.0*a/r)*cF*(Beta+((h**2.0)*(Beta**3.0))/
     &          (1-Beta)) -
     &          ((h*(Beta**3.0))/(1.0-Beta))*(h*cF-k*sF)*(3.0+(h**2.0)*
     &          (Beta**2.0)*(3.0-2.0*Beta)/((1.0-Beta)**2.0)) +
     &          (((a**2.0))/((r**2.0)))*cF*(h*Beta-sF)*(-sF +
     &          (a/r)*(h-sF)) - (((a**2.0))/((r**2.0)))*(cF**3.0))

      d2X1dkk = -a*(-(2.0*a/r)*sF*(h*k*(Beta**3.0))/(1.0-Beta) +
     &          (h*cF-k*sF)*(1.0+(((k**2.0)*(Beta**2.0)*(3.0-2.0*Beta))/
     &          ((1.0-Beta)**2.0)))*(h*(Beta**3.0))/(1.0-Beta) +
     &          ((a**2.0)/(r**2.0))*sF*(h*Beta-sF)*(-cF +
     &          (a/r)*(k-cF))+((a**2.0)/(r**2.0))*cF*(sF**2.0))
```

```
       d2X1dhdk = -a*((a/r)*cF*(h*k*(Beta**3.0))/(1.0-Beta) +
      &          (h*cF - k*sF)*(1.0+(h**2.0)*(Beta**2.0)*(3.0-2.0*Beta)/
      &          ((1.0-Beta)**2.0))*(k*(Beta**3.0))/(1.0-Beta) +
      &          (sF - h*Beta)*((a/r)*(sF**2.0 - h*sF) -
      &          cF**2.0)*((a**2.0)/(r**2.0)) -
      &          ((a**2.0)/(r**2.0))*sF*(cF**2.0) - (a/r)*sF*(Beta +
      &          (h**2.0)*(Beta**3.0)/(1.0-Beta)))

       d2X1dkdh = a*((a/r)*sF*(Beta+((h**2.0)*(Beta**3.0))/(1.0-Beta))-
      &          (h*cF-k*sF)*(1.0+(h**2.0)*(Beta**2.0)*(3.0-2.0*Beta)/
      &          ((1.0-Beta)**2.0))*(k*(Beta**3.0))/(1.0-Beta) +
      &          ((a**2.0)/(r**2.0))*((a/r)*(k*cF - cF**2.0) +
      &          sF**2.0)*(h*Beta - sF) -
      &          (a/r)*cF*(h*k*(Beta**3.0))/(1.0-Beta) +
      &          ((a**2.0)/(r**2.0))*(cF**2.0)*sF)

       d2Y1dhh = a*((2.0*a/r)*cF*(h*k*(Beta**3.0))/(1.0-Beta) +
      &         (h*cF-k*sF)*((k*(Beta**3.0))/(1.0-Beta))*(1.0 +
      &         (h**2.0)*(Beta**2.0)*(3.0-2.0*Beta)/((1.0-Beta)**2.0)) +
      &         ((a**2.0)/(r**2.0))*cF*(-(a/r)*(h-sF)+sF)*(k*Beta - cF)-
      &         ((a**2.0)/(r**2.0))*sF*(cF**2.0))

       d2Y1dkk = a*(-(2.0*a/r)*sF*(Beta+((k**2.0)*(Beta**3.0))/
      &         (1.0-Beta))+
      &         (h*cF - k*sF)*(3.0+((k**2.0)*(Beta**2.0)*(3.0-2.0*Beta))/
      &         ((1.0-Beta)**2.0))*(k*(Beta**3.0))/(1.0-Beta) +
      &         ((a**2.0)/(r**2.0))*sF*(-(a/r)*(k-cF) + cF)*(cF - k*Beta)
      &         - ((a**2.0)/(r**2.0))*(sF**3.0))

       d2Y1dhdk = a*((a/r)*cF*(Beta+((k**2.0)*(Beta**3.0))/(1.0-Beta))+
      &          (h*cF-k*sF)*((h*(Beta**3.0))/(1.0-Beta))*
      &          (1.0 + ((k**2.0)*(Beta**2.0)*(3.0-2.0*Beta))/
      &          ((1.0-Beta)**2.0)) -
      &          ((a**2.0)/(r**2.0))*((a/r)*sF*(h-sF) +
      &          cF**2.0)*(cF - k*Beta) +
      &          ((a**2.0)/(r**2.0))*cF*(sF**2.0) -
      &          (a/r)*sF*(h*k*(Beta**3.0))/(1.0-Beta))

       d2Y1dkdh = a*(-(a/r)*sF*(h*k*(Beta**3.0))/(1.0-Beta) +
      &          (h*cF-k*sF)*((h*(Beta**3.0))/(1.0-Beta))*
      &          (1.0+((k**2.0)*(Beta**2.0)*(3.0-2.0*Beta))/
      &          ((1.0-Beta)**2.0))-
      &          ((a**2.0)/(r**2.0))*((a/r)*cF*(k-cF) + sF**2.0)*
      &          (k*Beta - cF) +
      &          (a/r)*cF*(Beta + ((k**2.0)*(Beta**3.0))/(1.0-Beta)) +
      &          ((a**2.0)/(r**2.0))*cF*(sF**2.0))

       d2X1dadk = (1.0/a)*dX1dk
       d2X1dadh = (1.0/a)*dX1dh
       d2Y1dadk = (1.0/a)*dY1dk
       d2Y1dadh = (1.0/a)*dY1dh
C
C      Auxiliary partials
C
       dX1dF = a*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
       dY1dF = a*(-h*k*Beta*sF + (1.0-Beta*(k**2.0))*cF)

       dXdot1dF = -(a/r)*(k*sF - h*cF)*Xdot1 + (n*(a**2.0)/r)*
      &           (-h*k*Beta*sF - (1.0-Beta*(h**2.0))*cF)

       dYdot1dF = -(a/r)*(k*sF - h*cF)*Ydot1 + (n*(a**2.0)/r)*
      &           (-h*k*Beta*cF - (1.0-Beta*(k**2.0))*sF)

       d2X1dFdh = a*((h*sF+k*cF)*(Beta+((h**2.0)*(Beta**3.0))/
      &           (1.0-Beta))+
      &           ((a**2.0)/(r**2.0))*(h*Beta-sF)*(sF-h)+(a/r)*(cF**2.0))

       d2X1dFdk = -a*(-(h*sF + k*cF)*h*k*(Beta**3.0)/(1.0-Beta) +
      &           ((a**2.0)/(r**2.0))*(sF-h*Beta)*(cF-k)+(a/r)*sF*cF)
```

```
       d2Y1dFdh = a*(-(h*sF + k*cF)*h*k*(Beta**3.0)/(1.0-Beta) -
      &           ((a**2.0)/(r**2.0))*(k*Beta-cF)*(sF-h)+(a/r)*sF*cF)

       d2Y1dFdk = a*(-(h*sF+k*cF)*(Beta+(k**2.0)*(Beta**3.0)/(1.0-Beta))+
      &           ((a**2.0)/(r**2.0))*(cF-k*Beta)*(cF-k)-(a/r)*(sF**2.0))

       dXdot1da = -(n*a/(2.0*r))*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
       dYdot1da =  (n*a/(2.0*r))*(h*k*Beta*sF - (1.0-Beta*(k**2.0))*cF)

       dX1da = X1/a
       dY1da = Y1/a
C
C      Partials of a wrt rdot
C
       M(1,1) = 2.0*(a**-1.0)*(n**-2.0)*Xdot1
       M(1,2) = 2.0*(a**-1.0)*(n**-2.0)*Ydot1
       M(1,3) = 0.0
C
C      Partials of h wrt rdot
C
       M(2,1) = G*(n**-1.0)*(a**-2.0)*(dX1dk-h*Beta*Xdot1/n)
       M(2,2) = G*(n**-1.0)*(a**-2.0)*(dY1dk-h*Beta*Ydot1/n)
       M(2,3) = k*(q*Y1 - p*X1)*(n**-1.0)*(a**-2.0)*(G**-1.0)
C
C      Partials of k wrt rdot
C
       M(3,1) = -G*(n**-1.0)*(a**-2.0)*(dX1dh + k*Beta*Xdot1/n)
       M(3,2) = -G*(n**-1.0)*(a**-2.0)*(dY1dh + k*Beta*Ydot1/n)
       M(3,3) = -h*(q*Y1 - p*X1)*(n**-1.0)*(a**-2.0)*(G**-1.0)
C
C      Partials of p wrt rdot
C
       M(4,1) = 0.0
       M(4,2) = 0.0
       M(4,3) = K1*Y1*((n**-1.0)*(a**-2.0)*(G**-1.0))/2.0
C
C      Partials of q wrt rdot
C
       M(5,1) = 0.0
       M(5,2) = 0.0
       M(5,3) = K1*X1*((n**-1.0)*(a**-2.0)*(G**-1.0))/2.0
C
C      Partials of l wrt rdot
C
       M(6,1) = (n**-1.0)*(a**-2.0)*(-2.0*X1 +
      &         G*(h*Beta*dX1dh + k*Beta*dX1dk))
       M(6,2) = (n**-1.0)*(a**-2.0)*(-2.0*Y1 +
      &         G*(h*Beta*dY1dh + k*Beta*dY1dk))
       M(6,3) = (n**-1.0)*(a**-2.0)*(G**-1.0)*(q*Y1 - p*X1)
C
C      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C      % These next partials are partials of the M matrix wrt elements
C
C      Partials of M wrt h
C
       dMdh(1,1) = (2.0/(a*(n**2.0)))*dXdot1dh
       dMdh(1,2) = (2.0/(a*(n**2.0)))*dYdot1dh
       dMdh(1,3) = 0.0

       dMdh(2,1) = (-h/(G*n*(a**2.0)))*(dX1dk - (1.0/n)*h*Beta*Xdot1) +
      &            (G/(n*(a**2.0)))*(d2X1dhdk - (Xdot1/n)*(Beta +
      &            (h**2.0)*(Beta**3.0)/(1.0-Beta)) -
      &            h*Beta*dXdot1dh/n)

       dMdh(2,2) = (-h/(G*n*(a**2.0)))*(dY1dk - (1.0/n)*h*Beta*Ydot1) +
      &            (G/(n*(a**2.0)))*(d2Y1dhdk - (Ydot1/n)*(Beta +
      &            (h**2.0)*(Beta**3.0)/(1.0-Beta)) -
      &            h*Beta*dYdot1dh/n)

       dMdh(2,3) = ((1.0/(n*(a**2.0)))*h*k*(G**-3.0))*(q*Y1 - p*X1) +
      &            k*(q*dY1dh - p*dX1dh)/(n*(a**2.0)*G)
```

```
       dMdh(3,1) = (h/(n*(a**2.0)*G))*(dX1dh + k*Beta*Xdot1/n) -
      &            (G/(n*(a**2.0)))*(d2X1dhh + h*k*(Beta**3.0)*
      &            Xdot1/(n*(1.0-Beta)) +
      &            k*Beta*dXdot1dh/n)

       dMdh(3,2) = (h/(n*(a**2.0)*G))*(dY1dh + k*Beta*Ydot1/n) -
      &            (G/(n*(a**2.0)))*(d2Y1dhh + h*k*(Beta**3.0)*
      &            Ydot1/(n*(1.0-Beta)) +
      &            k*Beta*dYdot1dh/n)

       dMdh(3,3) = (-1.0/(n*(a**2.0)*G))*((q*Y1 - p*X1) +
      &            h*(q*dY1dh - p*dX1dh)) - ((h**2.0)*(q*Y1 - p*X1))/
      &            (n*(a**2.0)*(G**3.0))

       dMdh(4,1) = 0.0
       dMdh(4,2) = 0.0

       dMdh(4,3) = (K1/(2.0*n*(a**2.0)*G))*(dY1dh + h*Y1/(G**2.0))

       dMdh(5,1) = 0.0
       dMdh(5,2) = 0.0

       dMdh(5,3) = (K1/(2.0*n*(a**2.0)*G))*(dX1dh + h*X1/(G**2.0))

       dMdh(6,1) = (1.0/(n*(a**2.0)))*(-2.0*dX1dh - (h*Beta*(G**-1.0))*
      &            (h*dX1dh + k*dX1dk) +
      &            G*((Beta + (h**2.0)*(Beta**3.0)/(1.0-Beta))*dX1dh +
      &            h*k*(Beta**3.0)*dX1dk/(1.0-Beta) +
      &            Beta*(h*d2X1dhh + k*d2X1dhdk)))

       dMdh(6,2) = (1.0/(n*(a**2.0)))*(-2.0*dY1dh - (h*Beta*(G**-1.0))*
      &            (h*dY1dh + k*dY1dk) +
      &            G*((Beta + (h**2.0)*(Beta**3.0)/(1.0-Beta))*dY1dh +
      &            h*k*(Beta**3.0)*dY1dk/(1.0-Beta) +
      &            Beta*(h*d2Y1dhh + k*d2Y1dhdk)))

       dMdh(6,3) = ((G**-1.0)/(n*(a**2.0)))*((q*dY1dh - p*dX1dh) +
      &            h*(q*Y1 - p*X1)*(G**-2.0))
C
C      The partials of M wrt k
C
       dMdk(1,1) = (2.0/(a*(n**2.0)))*dXdot1dk

       dMdk(1,2) = (2.0/(a*(n**2.0)))*dYdot1dk

       dMdk(1,3) = 0.0

       dMdk(2,1) = (-k/(n*(a**2.0)*G))*(dX1dk - h*Beta*Xdot1/n) +
      &            (G/(n*(a**2.0)))*(d2X1dkk - h*k*(Beta**3.0)*Xdot1/
      &            (n*(1.0-Beta)) -
      &            h*Beta*dXdot1dk/n)

       dMdk(2,2) = (-k/(n*(a**2.0)*G))*(dY1dk - h*Beta*Ydot1/n) +
      &            (G/(n*(a**2.0)))*(d2Y1dkk - h*k*(Beta**3.0)*Ydot1/
      &            (n*(1.0-Beta)) -
      &            h*Beta*dYdot1dk/n)

       dMdk(2,3) = (q*Y1 - p*X1)/(n*(a**2.0)*G) +
      &            (1.0/(n*(a**2.0)*G))*(k*(q*dY1dk - p*dX1dk) +
      &            (k**2.0)*(q*Y1 - p*X1)/(G**2.0))

       dMdk(3,1) = (k/(n*(a**2.0)*G))*(dX1dh + k*Beta*Xdot1/n) -
      &            (G/(n*(a**2.0)))*(d2X1dkdh + (Beta + (k**2.0)*
      &            (Beta**3.0)/(1.0-Beta))*Xdot1/n +
      &            k*Beta*dXdot1dk/n)

       dMdk(3,2) = (k/(n*(a**2.0)*G))*(dY1dh + k*Beta*Ydot1/n) -
      &            (G/(n*(a**2.0)))*(d2Y1dkdh + (Beta + (k**2.0)*
      &            (Beta**3.0)/(1.0-Beta))*Ydot1/n +
      &            k*Beta*dYdot1dk/n)
```

```
       dMdk(3,3) = (-h/(n*(a**2.0)*G))*(q*dY1dk - p*dX1dk) -
      &            (h*k*(q*Y1 - p*X1)/(n*(a**2.0)*(G**3.0)))

       dMdk(4,1) = 0.0
       dMdk(4,2) = 0.0

       dMdk(4,3) = K1*dY1dk/(2.0*n*(a**2.0)*G) +
      &            k*K1*Y1/(2.0*n*(a**2.0)*(G**3.0))

       dMdk(5,1) = 0.0
       dMdk(5,2) = 0.0

       dMdk(5,3) = K1*dX1dk/(2*n*(a**2.0)*G) +
      &            k*K1*X1/(2*n*(a**2.0)*(G**3.0))

       dMdk(6,1) = (1.0/(n*(a**2.0)))*(-2.0*dX1dk - (k*Beta*(G**-1.0))*
      &            (h*dX1dh + k*dX1dk) +
      &            G*((Beta + (k**2.0)*(Beta**3.0)/(1.0-Beta))*dX1dk +
      &            h*k*(Beta**3.0)*dX1dh/(1.0-Beta) +
      &            Beta*(h*d2X1dkdh + k*d2X1dkk)))

       dMdk(6,2) = (1.0/(n*(a**2.0)))*(-2.0*dY1dk - (k*Beta*(G**-1.0))*
      &            (h*dY1dh + k*dY1dk) +
      &            G*((Beta + (k**2.0)*(Beta**3.0)/(1.0-Beta))*dY1dk +
      &            h*k*(Beta**3.0)*dY1dh/(1.0-Beta) +
      &            Beta*(h*d2Y1dkdh + k*d2Y1dkk)))

       dMdk(6,3) = ((G**-1.0)/(n*(a**2.0)))*((q*dY1dk - p*dX1dk) +
      &            k*(q*Y1-p*X1)*(G**-2.0))
C
C     The partials of M wrt p
C
       dMdp(2,3) = -k*X1/(n*(a**2.0)*G)
       dMdp(1,1) = 0.0
       dMdp(1,2) = 0.0
       dMdp(1,3) = 0.0

       dMdp(2,1) = 0.0
       dMdp(2,2) = 0.0

       dMdp(3,1) = 0.0
       dMdp(3,2) = 0.0
       dMdp(3,3) = h*X1/(n*(a**2.0)*G)

       dMdp(4,1) = 0.0
       dMdp(4,2) = 0.0
       dMdp(4,3) = p*Y1/(n*(a**2.0)*G)

       dMdp(5,1) = 0.0
       dMdp(5,2) = 0.0
       dMdp(5,3) = p*X1/(n*(a**2.0)*G)

       dMdp(6,1) = 0.0
       dMdp(6,2) = 0.0
       dMdp(6,3) = -X1/(n*(a**2.0)*G)
C
C     Partials of M wrt q
C
       dMdq(2,3) = k*Y1/(n*(a**2.0)*G)
       dMdq(1,1) = 0.0
       dMdq(1,2) = 0.0
       dMdq(1,3) = 0.0

       dMdq(2,1) = 0.0
       dMdq(2,2) = 0.0

       dMdq(3,1) = 0.0
       dMdq(3,2) = 0.0
       dMdq(3,3) = -h*Y1/(n*(a**2.0)*G)
```

```
      dMdq(4,1) = 0.0
      dMdq(4,2) = 0.0
      dMdq(4,3) = q*Y1/(n*(a**2.0)*G)

      dMdq(5,1) = 0.0
      dMdq(5,2) = 0.0
      dMdq(5,3) = q*X1/(n*(a**2.0)*G)

      dMdq(6,1) = 0.0
      dMdq(6,2) = 0.0
      dMdq(6,3) = Y1/(n*(a**2.0)*G)
C
C     The partials of M wrt a
C
      dMda(1,1) = 4.0*Xdot1/((n**2.0)*(a**2.0)) +
     &            2.0*dXdot1da/(a*(n**2.0))

      dMda(1,2) = 4.0*Ydot1/((n**2.0)*(a**2.0)) +
     &            2.0*dYdot1da/(a*(n**2.0))

      dMda(1,3) = 0.0

      dMda(2,1) = (G/(n*(a**2.0)))*(-dX1dk/(2.0*a) +
     &            d2X1dadk - h*Beta*Xdot1/(n*a) - h*Beta*dXdot1da/n)

      dMda(2,2) = (G/(n*(a**2.0)))*(-dY1dk/(2.0*a) +
     &            d2Y1dadk - h*Beta*Ydot1/(n*a) - h*Beta*dYdot1da/n)

      dMda(2,3) = (k/(n*(a**2.0)*G))*(-(1.0/(2.0*a))*(q*Y1 - p*X1) +
     &            q*dY1da - p*dX1da)

      dMda(3,1) = (-G/(n*(a**2.0)))*(-dX1dh/(2.0*a) + d2X1dadh +
     &            k*Beta*Xdot1/(n*a) + k*Beta*dXdot1da/n)

      dMda(3,2) = (-G/(n*(a**2.0)))*(-dY1dh/(2.0*a) + d2Y1dadh +
     &            k*Beta*Ydot1/(n*a) + k*Beta*dYdot1da/n)

      dMda(3,3) = (-h/(n*(a**2.0)*G))*(-(1.0/(2.0*a))*(q*Y1 - p*X1) +
     &            q*dY1da - p*dX1da)

      dMda(4,1) = 0.0
      dMda(4,2) = 0.0
      dMda(4,3) = (K1/(2.0*n*(a**2.0)*G))*(-(1.0/(2.0*a))*Y1 + dY1da)

      dMda(5,1) = 0.0
      dMda(5,2) = 0.0
      dMda(5,3) = (K1/(2.0*n*(a**2.0)*G))*(-(1.0/(2.0*a))*X1 + dX1da)

      dMda(6,1) = -M(6,1)/(2.0*a) + (1.0/(n*(a**2.0)))*
     &            (-2.0*dX1da + G*(h*Beta*d2X1dadh + k*Beta*d2X1dadk))

      dMda(6,2) = -M(6,2)/(2.0*a) + (1.0/(n*(a**2.0)))*
     &            (-2.0*dY1da + G*(h*Beta*d2Y1dadh + k*Beta*d2Y1dadk))

      dMda(6,3) = -M(6,3)/(2.0*a) + (1.0/(n*(a**2.0)))*
     &            (q*dY1da - p*dX1da)*(G**-1.0)
C
C     Partials of M wrt l
C
      dMdl(1,1) = (2.0/((n**2.0)*r))*dXdot1dF
      dMdl(1,2) = (2.0/((n**2.0)*r))*dYdot1dF
      dMdl(1,3) = 0.0

      dMdl(2,1) = (G/(n*a*r))*(d2X1dFdk - h*Beta*dXdot1dF/n)
      dMdl(2,2) = (G/(n*a*r))*(d2Y1dFdk - h*Beta*dYdot1dF/n)
      dMdl(2,3) = (1.0/(n*a*r*G))*(k*(q*dY1dF - p*dX1dF))

      dMdl(3,1) = -(G/(n*a*r))*(d2X1dFdh + k*Beta*dXdot1dF/n)
      dMdl(3,2) = -(G/(n*a*r))*(d2Y1dFdh + k*Beta*dYdot1dF/n)
      dMdl(3,3) = (1.0/(n*a*r*G))*(-h*(q*dY1dF - p*dX1dF))
```

```
 dMdl(4,1) = 0.0
 dMdl(4,2) = 0.0
 dMdl(4,3) = K1*dY1dF/(2.0*n*a*r*G)

 dMdl(5,1) = 0.0
 dMdl(5,2) = 0.0
 dMdl(5,3) = K1*dX1dF/(2.0*n*a*r*G)

 dMdl(6,1) = (1.0/(n*a*r))*(-2.0*dX1dF + G*(h*Beta*d2X1dFdh +
&            k*Beta*d2X1dFdk))

 dMdl(6,2) = (1.0/(n*a*r))*(-2.0*dY1dF + G*(h*Beta*d2Y1dFdh +
&            k*Beta*d2Y1dFdk))

 dMdl(6,3) = (1.0/(n*a*r*G))*(q*dY1dF - p*dX1dF)

 RETURN
 END
```

```
C -------------------------------------------------------------------------
C
C   FILE NAME: COMP_U.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------
C
      SUBROUTINE COMP_U(lam_vect,B,u)
C
C.....................................................................
C. ROUTINE: COMP_U
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: computes the normalized thrust acceleration vector
C.     given the 6x3 M matrix and the vector of current Lagrange multipliers.
C.
C.
C. CALLING SEQUENCE:
C          CALL COMP_U(lam_vect,B,u,u_norm)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER     I/O                   DESCRIPTION
C.          --------  ---  ------------------------------------------
C.          lam_vect  I    6x1 input vector of lagrange multipliers
C.          B         I    6x3 input matrix of partial derivatives
C.          u         O    3x1 normalized output thrust vector
C.          u_norm    O    scalar magnitude of thrust acceleration
C.
C. ROUTINES REQUIRED:  NONE
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      DOUBLE PRECISION lam_vect(6), B(6,3), u(3)
      DOUBLE PRECISION u_unnorm(3), u_norm

      u_unnorm(1) = lam_vect(1)*B(1,1) +
     &              lam_vect(2)*B(2,1) +
     &              lam_vect(3)*B(3,1) +
     &              lam_vect(4)*B(4,1) +
     &              lam_vect(5)*B(5,1) +
     &              lam_vect(6)*B(6,1)

      u_unnorm(2) = lam_vect(1)*B(1,2) +
     &              lam_vect(2)*B(2,2) +
     &              lam_vect(3)*B(3,2) +
     &              lam_vect(4)*B(4,2) +
     &              lam_vect(5)*B(5,2) +
     &              lam_vect(6)*B(6,2)

      u_unnorm(3) = lam_vect(1)*B(1,3) +
     &              lam_vect(2)*B(2,3) +
     &              lam_vect(3)*B(3,3) +
     &              lam_vect(4)*B(4,3) +
     &              lam_vect(5)*B(5,3) +
```

```
&              lam_vect(6)*B(6,3)

 u_norm = DSQRT(u_unnorm(1)**2.0 +
&               u_unnorm(2)**2.0 +
&               u_unnorm(3)**2.0)

 u(1) = (1.0/u_norm)*u_unnorm(1)
 u(2) = (1.0/u_norm)*u_unnorm(2)
 u(3) = (1.0/u_norm)*u_unnorm(3)

 RETURN
 END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: COMP_XY.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      SUBROUTINE COMP_XY(z_vect,mu,X1,Xdot1,Y1,Ydot1,nm,
     &                   cF,sF,G,Beta,r,K1)
C
C.....................................................................
C. ROUTINE: COMP_XY
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Calculates auxiliary quantities based on
C.    the current equinoctial orbital elements.
C.
C.
C. CALLING SEQUENCE:
C          CALL COMP_XY(z_vect,mu,X1,Xdot1,Y1,Ydot1,nm,
C                       cF,sF,G,Beta,r,K1)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER     I/O                   DESCRIPTION
C.          --------  ---  ------------------------------------------
C.          z_vect    I    the 6x1 vector of equinoctial elements
C.          mu        I    the central body gravitational constant
C.          X1        O    Cartesian X position magnitude
C.          Xdot1     O    Cartesian X velocity magnitude
C.          Y1        O    Cartesian Y position magnitude
C.          Ydot1     O    Cartesian Y velocity magnitude
C.          nm        O    mean motion
C.          cF        O    cosine of the eccentric longitude
C.          sF        O    sine of the eccentric longitude
C.          G         O    auxiliary parameter based on h,k
C.          Beta      O    auxiliary parameter based on h,k
C.          r         O    radial distance from center of primary mass
C.                         and satellite
C.          K1        O    auxiliary parameter based on p,q
C.
C. ROUTINES REQUIRED:  SOLVE_ECC_ANOMALY
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      IMPLICIT NONE

      DOUBLE PRECISION z_vect(6), mu, nm, cF, sF
      DOUBLE PRECISION a,h,k,p,q,l,G,r,K1,Beta
      DOUBLE PRECISION X1,Xdot1,Y1,Ydot1
      DOUBLE PRECISION arp, ran, E, F
      DOUBLE PRECISION SOLVE_ECC_ANOMALY

      a = z_vect(1)
      h = z_vect(2)
```

```
k = z_vect(3)
p = z_vect(4)
q = z_vect(5)
l = z_vect(6)

G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
Beta = 1.0/(1.0+G)

arp = DATAN2(h,k)-DATAN2(p,q)
ran = DATAN2(p,q)

E = SOLVE_ECC_ANOMALY(l-ran-arp,DSQRT(h**2.0+k**2.0))

F = E + DATAN2(h,k)

cF = DCOS(F)
sF = DSIN(F)

nm = DSQRT(mu)*(a**(-3.0/2.0))

r = a*(1.0 - k*cF - h*sF)

K1 = 1.0 + p**2.0 + q**2.0

X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
Ydot1 = (a**2.0)*nm*(1.0/r)*((1-Beta*(k**2.0))*cF - h*k*Beta*sF)

RETURN
END
```

```
C
C   FILE NAME: COMP_EQUIN_VAR.for  (averaged equation software)
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      SUBROUTINE COMP_EQUIN_VAR(z_vect,lam_vect,
     &                          dadt,dhdt,dkdt,dpdt,dqdt,dldt)
C
C.....................................................................
C. ROUTINE: COMP_EQUIN_VAR
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the derivatives of the equinoctial
C.     orbital elements with respect to time, i.e. element
C.     rates.  Because the averaged equations of motion are
C.     used here, the DQAG subroutine is used to compute the
C.     element rates using a Gauss-Kronrod numerical quadrature.
C.
C.
C. CALLING SEQUENCE:
C          CALL COMP_EQUIN_VAR(z_vect,lam_vect,
C                              dadt,dhdt,dkdt,dpdt,dqdt,dldt)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                   DESCRIPTION
C.          ---------  ---  -------------------------------------
C.          z_vect     I    6x1 vector of equinoctial elements
C.          lam_vect   I    6x1 vector of lagrange multipliers
C.          dadt       O    derivative of a wrt time
C.          dhdt       O    derivative of h wrt time
C.          dkdt       O    derivative of k wrt time
C.          dpdt       O    derivative of p wrt time
C.          dqdt       O    derivative of q wrt time
C.          dldt       O    derivative of mean long. wrt time
C.
C. ROUTINES REQUIRED:  DQAG
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      PARAMETER (limit = 50,lenw=limit*4)

      INTEGER key, neval, ier, limit, lenw, last
      INTEGER iwork(limit), I

      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION dadt,dhdt,dkdt,dpdt,dqdt,dldt,pi
      DOUBLE PRECISION a, b, epsabs, epsrel, result, abserr
      DOUBLE PRECISION work(lenw), FQUAD
      DOUBLE PRECISION RHS_ADOT, RHS_HDOT, RHS_KDOT, RHS_PDOT, RHS_QDOT
      DOUBLE PRECISION RHS_LDOT

      EXTERNAL RHS_ADOT, RHS_HDOT, RHS_KDOT, RHS_PDOT, RHS_QDOT
```

```
 EXTERNAL RHS_LDOT

 COMMON /FQUAD/ fquad_z_vect, fquad_lam_vect

 key = 3
 epsabs = 1.0E-6
 epsrel = 1.0E-6

 pi = 3.141592653589793

 DO I=1,6
    fquad_z_vect(I) = z_vect(I)
    fquad_lam_vect(I) = lam_vect(I)
 END DO

 CALL DQAG(RHS_ADOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dadt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_HDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dhdt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_KDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dkdt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_PDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dpdt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_QDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dqdt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_LDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dldt = (1.0/(2.0*pi))*result

 RETURN
 END
```

```
C
C   FILE NAME: COMP_EUL_LAG_VAR.for  (averaged equation software)
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      SUBROUTINE COMP_EUL_LAG_VAR(z_vect,lam_vect,
     &                            dlamadt,dlamhdt,dlamkdt,dlampdt,
     &                            dlamqdt,dlamldt)
C
C.....................................................................
C. ROUTINE: COMP_EUL_LAG_VAR
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the derivatives of the Lagrange
C.     multipliers with respect to time, i.e. multiplier
C.     rates.  The averaged equations for the multiplier
C.     rates are computed using the DQAG subroutine which
C.     performs numerical quadrature using the Gauss-Kronrod method.
C.
C.
C. CALLING SEQUENCE:
C          CALL COMP_EUL_LAG_VAR(z_vect,lam_vect,
C                                dlamadt,dlamhdt,dlamkdt,dlampdt,
C                                dlamqdt,dlamldt)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ------------------------------------
C.          z_vect     I    6x1 vector of equinoctial elements
C.          lam_vect   I    6x1 vector of lagrange mulipliers
C.          dlamadt    O    derivative of lagrange mult. for a wrt time
C.          dlamhdt    O    derivative of lagrange mult. for h wrt time
C.          dlamkdt    O    derivative of lagrange mult. for k wrt time
C.          dlampdt    O    derivative of lagrange mult. for p wrt time
C.          dlamqdt    O    derivative of lagrange mult. for q wrt time
C.          dlamldt    O    derivative of lagrange mult. for mean long. wrt time
C.
C. ROUTINES REQUIRED:  DQAG
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      PARAMETER (limit = 50,lenw=limit*4)

      INTEGER key, neval, ier, limit, lenw, last
      INTEGER iwork(limit), I

      DOUBLE PRECISION z_vect(6),lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION dlamadt,dlamhdt,dlamkdt,dlampdt
      DOUBLE PRECISION dlamqdt,dlamldt,pi
      DOUBLE PRECISION a, b, epsabs, epsrel, result, abserr
      DOUBLE PRECISION work(lenw), FQUAD
```

```
 EXTERNAL RHS_LAMADOT, RHS_LAMHDOT, RHS_LAMKDOT
 EXTERNAL RHS_LAMPDOT, RHS_LAMQDOT, RHS_LAMLDOT

 COMMON /FQUAD/ fquad_z_vect, fquad_lam_vect

 key = 3
 epsabs = 1.0E-6
 epsrel = 1.0E-6

 pi = 3.141592653589793

 DO I=1,6
    fquad_z_vect(I) = z_vect(I)
    fquad_lam_vect(I) = lam_vect(I)
 END DO

 CALL DQAG(RHS_LAMADOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dlamadt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_LAMHDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dlamhdt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_LAMKDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dlamkdt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_LAMPDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dlampdt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_LAMQDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dlamqdt = (1.0/(2.0*pi))*result

 CALL DQAG(RHS_LAMLDOT,-pi,pi,epsabs,epsrel,key,result,abserr,
&          neval,ier,limit,lenw,last,iwork,work)

 dlamldt = (1.0/(2.0*pi))*result

 RETURN
 END
```

```
C -------------------------------------------------------------------------------
C
C   FILE NAME: RHS_ADOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_ADOT(F)
C
C.....................................................................
C. ROUTINE: RHS_ADOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the semimajor axis rate of change using
C.          the current equinoctial elements and the COMP_M
C.          and COMP_U subroutines.
C.
C.
C. CALLING SEQUENCE:
C          ADOT = RHS_ADOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ----------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect, fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
         lam_vect(I) = fquad_lam_vect(I)
      END DO
```

```
      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)
C
C     Compute the result
C
      RHS_ADOT = ft*(M(1,1)*u(1) + M(1,2)*u(2) + M(1,3)*u(3))*
     &           (1.0-k*cF-h*sF)

      RETURN
      END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: RHS_HDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_HDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_HDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the equinoctial h element rate of change using
C.          the current equinoctial elements and the COMP_M
C.          and COMP_U subroutines.
C.
C.
C. CALLING SEQUENCE:
C          HDOT = RHS_HDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ----------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect, fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
         lam_vect(I) = fquad_lam_vect(I)
      END DO
```

```
      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)
C
C     Compute the result
C
      RHS_HDOT = ft*(M(2,1)*u(1) + M(2,2)*u(2) + M(2,3)*u(3))*
     &           (1.0-k*cF-h*sF)

      RETURN
      END
```

```
C -------------------------------------------------------------------------------
C
C   FILE NAME: RHS_KDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_KDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_KDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the equinoctial k element rate of change using
C.          the current equinoctial elements and the COMP_M
C.          and COMP_U subroutines.
C.
C.
C. CALLING SEQUENCE:
C          KDOT = RHS_KDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ----------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect, fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
         lam_vect(I) = fquad_lam_vect(I)
      END DO
```

```
      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)
C
C     Compute the result
C
      RHS_KDOT = ft*(M(3,1)*u(1) + M(3,2)*u(2) + M(3,3)*u(3))*
     &           (1.0-k*cF-h*sF)

      RETURN
      END
```

```
C -------------------------------------------------------------------------------
C
C   FILE NAME: RHS_PDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_PDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_PDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the equinoctial p element rate of change using
C.          the current equinoctial elements and the COMP_M
C.          and COMP_U subroutines.
C.
C.
C. CALLING SEQUENCE:
C          PDOT = RHS_PDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ----------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect, fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
         lam_vect(I) = fquad_lam_vect(I)
      END DO
```

```
      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)
C
C     Compute the result
C
      RHS_PDOT = ft*(M(4,1)*u(1) + M(4,2)*u(2) + M(4,3)*u(3))*
     &           (1.0-k*cF-h*sF)

      RETURN
      END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: RHS_QDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_QDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_QDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the equinoctial q element rate of change using
C.          the current equinoctial elements and the COMP_M
C.          and COMP_U subroutines.
C.
C.
C. CALLING SEQUENCE:
C          QDOT = RHS_QDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ----------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect, fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
         lam_vect(I) = fquad_lam_vect(I)
      END DO
```

```
      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)
C
C     Compute the result
C
      RHS_QDOT = ft*(M(5,1)*u(1) + M(5,2)*u(2) + M(5,3)*u(3))*
     &           (1.0-k*cF-h*sF)

      RETURN
      END
```

```
C -------------------------------------------------------------------------------
C
C   FILE NAME: RHS_LDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_LDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_LDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the mean longitude element rate of change using
C.          the current equinoctial elements and the COMP_M
C.          and COMP_U subroutines.
C.
C.
C. CALLING SEQUENCE:
C          LDOT = RHS_LDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ----------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect, fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
         lam_vect(I) = fquad_lam_vect(I)
      END DO
```

```
      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)
C
C     Compute the result
C
      RHS_LDOT = ft*(M(6,1)*u(1) + M(6,2)*u(2) + M(6,3)*u(3))*
     &           (1.0-k*cF-h*sF)

      RETURN
      END
```

```
C -------------------------------------------------------------------------------
C
C   FILE NAME: RHS_LAMADOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_LAMADOT(F)
C
C.....................................................................
C. ROUTINE: RHS_LAMADOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the Lagrange multiplier rate associated with
C.          the semimajor axis using the COMP_M and COMP_U subroutines.
C.
C.
C. CALLING SEQUENCE:
C          LAMADOT = RHS_LAMADOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ------------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION lama,lamh,lamk,lamp,lamq,laml
      DOUBLE PRECISION dnda
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect, fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
         lam_vect(I) = fquad_lam_vect(I)
```

```
      END DO

      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      lama = lam_vect(1)
      lamh = lam_vect(2)
      lamk = lam_vect(3)
      lamp = lam_vect(4)
      lamq = lam_vect(5)
      laml = lam_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)

      dnda = -3.0*nm/(2.0*a);

      RHS_LAMADOT =
     &  ft*(-lama*(dMda(1,1)*u(1) + dMda(1,2)*u(2) + dMda(1,3)*u(3)) +
     &      -lamh*(dMda(2,1)*u(1) + dMda(2,2)*u(2) + dMda(2,3)*u(3)) +
     &      -lamk*(dMda(3,1)*u(1) + dMda(3,2)*u(2) + dMda(3,3)*u(3)) +
     &      -lamp*(dMda(4,1)*u(1) + dMda(4,2)*u(2) + dMda(4,3)*u(3)) +
     &      -lamq*(dMda(5,1)*u(1) + dMda(5,2)*u(2) + dMda(5,3)*u(3)) +
     &      -laml*(dMda(6,1)*u(1) + dMda(6,2)*u(2) + dMda(6,3)*u(3)))*
     &     (1.0-k*cF-h*sF) +
     &  -laml*dnda*(1.0-k*cF-h*sF)

      RETURN
      END
```

```
C ------------------------------------------------------------------------------
C
C   FILE NAME: RHS_LAMHDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C ------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_LAMHDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_LAMHDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the Lagrange multiplier rate associated with
C.          the h equinoctial element using the COMP_M and COMP_U
C.          subroutines.
C.
C.
C. CALLING SEQUENCE:
C          LAMHDOT = RHS_LAMHDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ------------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION lama,lamh,lamk,lamp,lamq,laml
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION dnda
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect,fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
```

```
         lam_vect(I) = fquad_lam_vect(I)
      END DO

      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      lama = lam_vect(1)
      lamh = lam_vect(2)
      lamk = lam_vect(3)
      lamp = lam_vect(4)
      lamq = lam_vect(5)
      laml = lam_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)

      dnda = -3.0*nm/(2.0*a);

      RHS_LAMHDOT =
     &  ft*(-lama*(dMdh(1,1)*u(1) + dMdh(1,2)*u(2) + dMdh(1,3)*u(3)) +
     &      -lamh*(dMdh(2,1)*u(1) + dMdh(2,2)*u(2) + dMdh(2,3)*u(3)) +
     &      -lamk*(dMdh(3,1)*u(1) + dMdh(3,2)*u(2) + dMdh(3,3)*u(3)) +
     &      -lamp*(dMdh(4,1)*u(1) + dMdh(4,2)*u(2) + dMdh(4,3)*u(3)) +
     &      -lamq*(dMdh(5,1)*u(1) + dMdh(5,2)*u(2) + dMdh(5,3)*u(3)) +
     &      -laml*(dMdh(6,1)*u(1) + dMdh(6,2)*u(2) + dMdh(6,3)*u(3)))*
     &     (1.0-k*cF-h*sF)

      RETURN
      END
```

```
C -------------------------------------------------------------------------------
C
C   FILE NAME: RHS_LAMKDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_LAMKDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_LAMKDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the Lagrange multiplier rate associated with
C.          the k equinoctial element using the COMP_M and COMP_U
C.          subroutines.
C.
C.
C. CALLING SEQUENCE:
C          LAMKDOT = RHS_LAMKDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ------------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION lama,lamh,lamk,lamp,lamq,laml
      DOUBLE PRECISION dnda
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect,fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
```

```
         lam_vect(I) = fquad_lam_vect(I)
       END DO

       a = z_vect(1)
       h = z_vect(2)
       k = z_vect(3)
       p = z_vect(4)
       q = z_vect(5)
       l = z_vect(6)

       lama = lam_vect(1)
       lamh = lam_vect(2)
       lamk = lam_vect(3)
       lamp = lam_vect(4)
       lamq = lam_vect(5)
       laml = lam_vect(6)

       G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
       Beta = 1.0/(1.0+G)

       nm = DSQRT(mu)*(a**(-3.0/2.0))

       r = a*(1.0 - k*cF - h*sF)

       K1 = 1.0 + p**2.0 + q**2.0

       X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
       Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

       Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
       Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C      Compute the M matrix of equinoctial partials wrt rdot,
C      the partials of M wrt the equinoctial elements, and
C      auxiliary partial derivatives.
C
       CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
      &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C      Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
       CALL COMP_U(lam_vect,M,u)

       dnda = -3.0*nm/(2.0*a);

       RHS_LAMKDOT =
      &  ft*(-lama*(dMdk(1,1)*u(1) + dMdk(1,2)*u(2) + dMdk(1,3)*u(3)) +
      &      -lamh*(dMdk(2,1)*u(1) + dMdk(2,2)*u(2) + dMdk(2,3)*u(3)) +
      &      -lamk*(dMdk(3,1)*u(1) + dMdk(3,2)*u(2) + dMdk(3,3)*u(3)) +
      &      -lamp*(dMdk(4,1)*u(1) + dMdk(4,2)*u(2) + dMdk(4,3)*u(3)) +
      &      -lamq*(dMdk(5,1)*u(1) + dMdk(5,2)*u(2) + dMdk(5,3)*u(3)) +
      &      -laml*(dMdk(6,1)*u(1) + dMdk(6,2)*u(2) + dMdk(6,3)*u(3)))*
      &     (1.0-k*cF-h*sF)

       RETURN
       END
```

```
C -------------------------------------------------------------------------------
C
C   FILE NAME: RHS_LAMPDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_LAMPDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_LAMPDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the Lagrange multiplier rate associated with
C.          the p equinoctial element using the COMP_M and COMP_U
C.          subroutines.
C.
C.
C. CALLING SEQUENCE:
C          LAMPDOT = RHS_LAMPDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ----------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION lama,lamh,lamk,lamp,lamq,laml
      DOUBLE PRECISION dnda
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect,fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
```

```
         lam_vect(I) = fquad_lam_vect(I)
      END DO

      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      lama = lam_vect(1)
      lamh = lam_vect(2)
      lamk = lam_vect(3)
      lamp = lam_vect(4)
      lamq = lam_vect(5)
      laml = lam_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)

      dnda = -3.0*nm/(2.0*a);

      RHS_LAMPDOT =
     &  ft*(-lama*(dMdp(1,1)*u(1) + dMdp(1,2)*u(2) + dMdp(1,3)*u(3)) +
     &      -lamh*(dMdp(2,1)*u(1) + dMdp(2,2)*u(2) + dMdp(2,3)*u(3)) +
     &      -lamk*(dMdp(3,1)*u(1) + dMdp(3,2)*u(2) + dMdp(3,3)*u(3)) +
     &      -lamp*(dMdp(4,1)*u(1) + dMdp(4,2)*u(2) + dMdp(4,3)*u(3)) +
     &      -lamq*(dMdp(5,1)*u(1) + dMdp(5,2)*u(2) + dMdp(5,3)*u(3)) +
     &      -laml*(dMdp(6,1)*u(1) + dMdp(6,2)*u(2) + dMdp(6,3)*u(3)))*
     &     (1.0-k*cF-h*sF)

      RETURN
      END
```

```
C -------------------------------------------------------------------------------
C
C   FILE NAME: RHS_LAMQDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_LAMQDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_LAMQDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the Lagrange multiplier rate associated with
C.          the q equinoctial element using the COMP_M and COMP_U
C.          subroutines.
C.
C.
C. CALLING SEQUENCE:
C          LAMQDOT = RHS_LAMQDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ----------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION lama,lamh,lamk,lamp,lamq,laml
      DOUBLE PRECISION dnda
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect,fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
         lam_vect(I) = fquad_lam_vect(I)
```

```
      END DO

      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      lama = lam_vect(1)
      lamh = lam_vect(2)
      lamk = lam_vect(3)
      lamp = lam_vect(4)
      lamq = lam_vect(5)
      laml = lam_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)

      dnda = -3.0*nm/(2.0*a);

      RHS_LAMQDOT =
     &  ft*(-lama*(dMdq(1,1)*u(1) + dMdq(1,2)*u(2) + dMdq(1,3)*u(3)) +
     &      -lamh*(dMdq(2,1)*u(1) + dMdq(2,2)*u(2) + dMdq(2,3)*u(3)) +
     &      -lamk*(dMdq(3,1)*u(1) + dMdq(3,2)*u(2) + dMdq(3,3)*u(3)) +
     &      -lamp*(dMdq(4,1)*u(1) + dMdq(4,2)*u(2) + dMdq(4,3)*u(3)) +
     &      -lamq*(dMdq(5,1)*u(1) + dMdq(5,2)*u(2) + dMdq(5,3)*u(3)) +
     &      -laml*(dMdq(6,1)*u(1) + dMdq(6,2)*u(2) + dMdq(6,3)*u(3)))*
     &     (1.0-k*cF-h*sF)

      RETURN
      END
```

```
C -------------------------------------------------------------------------------
C
C   FILE NAME: RHS_LAMLDOT.for
C
C     VERSION: 1.0
C
C     CREATED: 10/13/2007
C
C     Copyright Massachusetts Institute of Technology.  All rights reserved.
C
C -------------------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION RHS_LAMLDOT(F)
C
C.....................................................................
C. ROUTINE: RHS_LAMLDOT
C.
C.
C. VERSION: 1.0
C.
C.
C. PROGRAMMED BY:
C.          Z J. FOLCIK
C.
C.
C. PURPOSE: Computes the Lagrange multiplier rate associated with
C.          the mean longitude using the COMP_M and COMP_U
C.          subroutines.
C.
C.
C. CALLING SEQUENCE:
C          LAMLDOT = RHS_LAMLDOT(F)
C.
C. PARAMETER DESCRIPTION:
C.          PARAM-     1
C.           ETER      I/O                  DESCRIPTION
C.          ---------  ---  ----------------------------------
C.          F          I    Input value of eccentric longitude
C.
C. ROUTINES REQUIRED:  COMP_M, COMP_U
C.
C.....................................................................
C
C***************** DECLARATIONS ***************************************
C
C
      INTEGER I

      DOUBLE PRECISION F,sF,cF,a,h,k,p,q,l,G,Beta,nm,r,K1,X1,Y1
      DOUBLE PRECISION Ydot1,Xdot1,M(6,3),u(3)
      DOUBLE PRECISION dMda(6,3),dMdh(6,3),dMdk(6,3)
      DOUBLE PRECISION dMdp(6,3),dMdq(6,3),dMdl(6,3)
      DOUBLE PRECISION ft, mu, z0_vect(6), zF_vect(6)
      DOUBLE PRECISION z_vect(6), lam_vect(6)
      DOUBLE PRECISION fquad_z_vect(6), fquad_lam_vect(6)
      DOUBLE PRECISION lama,lamh,lamk,lamp,lamq,laml
      DOUBLE PRECISION dnda
      DOUBLE PRECISION EXTDAT, FQUAD

      COMMON /EXTDAT/ ft,mu,z0_vect,zF_vect
      COMMON /FQUAD/  fquad_z_vect,fquad_lam_vect
C
C     Replace the sF and cF because we are calculating the quadrature
C     from F = -pi to pi.
C
      sF = DSIN(F)
      cF = DCOS(F)

      DO I=1,6
         z_vect(I) = fquad_z_vect(I)
         lam_vect(I) = fquad_lam_vect(I)
```

```
      END DO

      a = z_vect(1)
      h = z_vect(2)
      k = z_vect(3)
      p = z_vect(4)
      q = z_vect(5)
      l = z_vect(6)

      lama = lam_vect(1)
      lamh = lam_vect(2)
      lamk = lam_vect(3)
      lamp = lam_vect(4)
      lamq = lam_vect(5)
      laml = lam_vect(6)

      G = (1.0 - h**2.0 - k**2.0)**(1.0/2.0)
      Beta = 1.0/(1.0+G)

      nm = DSQRT(mu)*(a**(-3.0/2.0))

      r = a*(1.0 - k*cF - h*sF)

      K1 = 1.0 + p**2.0 + q**2.0

      X1 = a*((1.0-Beta*(h**2.0))*cF + h*k*Beta*sF - k)
      Y1 = a*(h*k*Beta*cF + (1.0-Beta*(k**2.0))*sF - h)

      Xdot1 = (a**2.0)*nm*(1.0/r)*(h*k*Beta*cF - (1.0-Beta*(h**2.0))*sF)
      Ydot1 = (a**2.0)*nm*(1.0/r)*((1.0-Beta*(k**2.0))*cF - h*k*Beta*sF)
C
C     Compute the M matrix of equinoctial partials wrt rdot,
C     the partials of M wrt the equinoctial elements, and
C     auxiliary partial derivatives.
C
      CALL COMP_M(z_vect,X1,Xdot1,Y1,Ydot1,nm,cF,sF,G,Beta,r,K1,
     &            M,dMda,dMdh,dMdk,dMdp,dMdq,dMdl)
C
C     Compute the unit vector u, i.e. the components of the equinoctial f,g,w vector.
C
      CALL COMP_U(lam_vect,M,u)

      dnda = -3.0*nm/(2.0*a);

      RHS_LAMLDOT =
     &  ft*(-lama*(dMdl(1,1)*u(1) + dMdl(1,2)*u(2) + dMdl(1,3)*u(3)) +
     &      -lamh*(dMdl(2,1)*u(1) + dMdl(2,2)*u(2) + dMdl(2,3)*u(3)) +
     &      -lamk*(dMdl(3,1)*u(1) + dMdl(3,2)*u(2) + dMdl(3,3)*u(3)) +
     &      -lamp*(dMdl(4,1)*u(1) + dMdl(4,2)*u(2) + dMdl(4,3)*u(3)) +
     &      -lamq*(dMdl(5,1)*u(1) + dMdl(5,2)*u(2) + dMdl(5,3)*u(3)) +
     &      -laml*(dMdl(6,1)*u(1) + dMdl(6,2)*u(2) + dMdl(6,3)*u(3)))*
     &     (1.0-k*cF-h*sF)

      RETURN
      END
```

Chapter 8 References

1. *GEO Maneuver Detection for Space Situational Awareness AAS 07-285.* **Folcik, Z., Cefola, P. J., Abbot, R.,.** Mackinac Island, MI : AAS/AIAA, August 19-23, 2007. Astrodynamics 2007, Advances in the Astronautical Sciences Volumn 129.
2. *Backward Smoothing Extended Kalman Filter.* **Psiaki, M.** 5, s.l. : Journal of Guidance, Control and Dynamics, Sept-Oct 2005, Vol. 28.
3. *Optimal Low-Thrust Rendezvous Using Equinoctial Orbit Elements.* **Kechichian, J. A.** 1, pp. 1-14, s.l. : Acta Astronautica, 1996, Vol. 38.
4. *Optimal Low-Earth-Orbit-Geostationary-Earth-Orbit Intermediate Acceleration Orbit Transfer.* **Kechichian, J. A.,.** 4, s.l. : Journal of Guidance, Control and Dynamics, July-Aug, 1997, Vol. 20.
5. *Trajectory Optimization Using Nonsingular Orbital Elements and True Longitude.* **Kechichian, J. A.** 5, s.l. : Journal of Guidance, Control and Dynamics, Sept-Oct, 1997, Vol. 20.
6. *Mechanics of Trajectory Optimization Using Nonsingular Variational Equations in Polar Coordinates.* **Kechichian, J. A.** 4, s.l. : Journal of Guidance, Control and Dynamics, July-Aug 1997, Vol. 20.
7. *Minimum-Time Constant Acceleration Orbit Transfer with First-Order Oblateness Effect.* **Kechichian, J. A.** 4, s.l. : Journal of Guidance, Control and Dynamics, July-Aug 2000, Vol. 23.
8. *The Inclusion of the Higher Order J3 and J4 Zonal Harmonics in the Modeling of Optimal Low-Thrust Orbit Transfers.* **Kechichian, J. A.** Galveston, TX : Paper No. 08-198, Proceedings of the 2008 AAS/AIAA Space Flight Mechanics Meeting, Jan 27-31, 2008.
9. **Kechichian, J. A.** Optimal Low-Thrust Orbit Transfer (Chapter 14). [book auth.] V. A. Chobotov. *Orbital Mechanics, 2nd Ed.* Reston, VA : AIAA Education Series, 1996.
10. **Taylor, S. P.** Semianalytical Satellite Theory and Sequential Estimation. *Master's Thesis at the Massachusetts Institute of Technology.* Cambridge, MA : s.n., 1981.
11. **Battin, R.** *An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition.* Reston, VA : AIAA Education Series, 1999.
12. *The Motion of a Close Earth Satellite.* **Kozai, Y.** 1274 pp.367-377, s.l. : Astronomical Journal, 1959, Vol. 64.
13. *Solutions of the Problem of Artificial Satellite Theory without Drag.* **Brouwer, D.** 1274 pp. 378-397, s.l. : Astronomical Journal, 1959, Vol. 64.
14. **Vallado, D.** *Fundamentals of Astrodynamics and Applications, Second Edition.* El Segundo, CA and Dordrecht, the Netherlands : Microcosm Press, Kluwer Academic Publishers, 2001.
15. **McClain, W. D.** *A Semianalytic Artificial Satellite Theory - Volume 1.* s.l. : Copyright Waine D. McClain, 1992.
16. **Wiesel, W.** *Modern Astrodynamics.* Beavercreek, OH : Aphelion Press, 2003.
17. **Hoots, F. R., Roehrich, R. I.,.** *Models for Propagation of NORAD Element Sets - Spacetrack Report No. 3.* s.l. : Aerospace Defense Command, United States Air Force, 1980.

18. **Herriges, D. L.** NORAD General Perturbation Theories: An Independent Analysis. *Master's Thesis at the Massachusetts Institute of Technology.* Cambridge, MA : s.n., 1988.
19. *A Short Efficient Analytic Satellite Theory.* **Hoots, F. R.,.** Danvers, MA : Journal of Guidance paper No. AIAA 80-1659R, AIAA/AAS Astrodynamics Conference, Aug 11-13, 1980.
20. *Revisiting Space Track Report #3.* **Vallado, D.A., Crawford, P., Hujsak, R., Kelso, T.S.** AIAA/AAS Astrodynamics Specialist Conference Keystone, CO : AIAA 2006-6753, Aug 21-24, 2006.
21. **Lane, M., Hoots, F.** *General Perturbations Theories Derived froum the 1965 Lane Drag Theory - Spacetrack Report No. 2.* s.l. : Aerospace Defense Command, United States Air Force, 1979.
22. **Sridharan, R., Seniw, W.** *ANODE: An Analytic Orbit Determination System.* s.l. : Lincoln Laboratory Archives, Technical Note ESD-TR-80-75, June 24, 1980.
23. *Extension of the Naval Space Command Satellite Theory PPT2 to Include a General Tesseral M-Daily Model.* **Cefola, P. J., Fonte, D. J.** AIAA/AAS Astrodynamics Conference, San Diego, CA : AIAA Paper No. AIAA–96–3606–CP, July 29-31, 1996.
24. *A Portable Orbit Generator Using Semianalytic Satellite Theory.* **Early, L. W.** Williamsburg, VA : AIAA/AAS Astrodynamics Conference Paper No. 86-2164-CP, Aug 1986.
25. **King-Hele, Desmond.** *A Tapestry of Orbits.* Cambridge, UK and New York, NY, USA : Cambridge University Press, 1992.
26. **Fonte, D. J.** Implementing a 50x50 Gravity Field Model in an Orbit Determination System. *Master's Thesis at the Massachusetts Institute of Technology.* Cambridge, MA : s.n., 1993.
27. **Lyon, R. H.** Geosynchronous Orbit Determination Using Space Surveillance Network Observations and Improved Radiative Force Modeling. *Master's Thesis at the Massachusetts Institute of Technology.* Cambridge, MA : s.n., June 2004.
28. **Efroimsky, M.** *Implicit Gauge Symmetry Emerging in the N-Body Problem of Celestial Mechanics.* s.l. : http://arxiv.org/abs/astro-ph/0212245v9 visited March, 2008, revised July 8, 2003.
29. **Gurfil, Pini. Editor.** *Modern Astrodynamics.* Oxford, UK : Elsevier Academic Press, 2006.
30. *On the Matrizant of the Two-Body Problem.* **Broucke, R. A.** pp. 173-182, s.l. : Astronomy and Astrophysics, 1970, Vol. 6.
31. *On the Equinoctial Orbit Elements.* **Broucke, R.A., Cefola, P. J.** 3, pp. 303-310, s.l. : Celestial Mechanics and Dynamical Astronomy, 1972, Vol. 5.
32. *Goddard Trajectory Determination System (GTDS) Mathematical Theory - NASA's Operational GTDS Mathematical Specification Revision 1 - Contract NAS 5.31500. Task 213.* s.l. : Edited by Computer Sciences Corporation and NASA Goddard Spaceflight Center, July 1989.
33. *Problems Concerning the Perturbation Due to the Tesseral Harmonic Terms in the Nonspherical Gravitational Potential of the Earth.* **Ma Jian-bo, Liu Lin, Wang Xin.** pp. 235-244, s.l. : Chinese Astronomy and Astrophysics, 2002, Vol. 26.

34. *A Recursive Formulation of the Short-Periodic Perturbations in Equinoctial Variables.* **Cefola, P. J., McClain, W. D.** Palo Alto, CA : Pre-print 78-1383 at the AIAA/AAS Astrodynamics Conference, August, 1978.
35. **Green, A. J.** Orbit Determination and Prediction Processes for Low Altitude Satellites. *Ph.D. Thesis at the Massachusetts Institute of Technology.* Cambridge, MA : s.n., 1980.
36. **Collins, S. K.** Long Term Prediction of High Altitude Orbits. *Ph.D. Dissertation, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, CSDL-T-739.* March, 1981.
37. *Recent Advances in Satellite Propulsion and Associated Mission Benefits - AIAA Paper No. 2006-5306.* **Wilson, F.** San Diego, CA : AIAA International Communications Satellite System Conference, June 2006.
38. *Spacecraft Electric Propulsion - An Overview.* **Martinez-Sanchez, M., Pollard, J. E.** 5, pp. 688-699, s.l. : AIAA Journal of Propulsion and Power, 1998, Vol. 14.
39. *Pulsed Plasma Thruster.* **Burton, R. L., Turchi, P. J.** 5, s.l. : AIAA Journal of Propulsion and Power, 1998, Vol. 14.
40. *SMART-1 Mission Description and Development Status.* **Racca, G. D., et. al.** 14-15, pp. 1323-1337, s.l. : Planetary and Space Science, December 2002, Vol. 50.
41. **King, Brad.** Personal communication through email. Feb 25,2008.
42. *Controlling a Stationary Orbit Using Electric Propulsion.* **Anzel, B.** Garnisch-Partenkirchen, Federal Republic of Germany : 20th International Electric Propulsion Conference, October 1988.
43. **Douglas, T., Kelly, C., Grise, A.** *On-Orbit Stationkeeping with Ion Thrusters - Telesat Canada's BSS-702 Experience.* Ottawa, Ontario, K1B 5P4 : Telesat Canada, 2006.
44. *In-Flight Performance of the NSTAR Ion Propulsion System on the Deep Space One Mission.* **Polk, J. E., Kakuda, R. Y., Anderson, J. R., Brophy, J. R.** Big Sky, Montana : IEEE Aerospace Conference, March 2000.
45. Integrated Defense Systems. *Boeing Web site.* [Online] : http://www.boeing.com/defense-space/space/bss/factsheets/702/702fleet.html.
46. *SPT-100 Subsystem Qualification Status.* **Day, M., Maslennikov, N., Randolph, T., Rogers, W.** Lake Buena Vista, FL : 32nd AIAA, ASME, SAE, ASEE Joint Propulsion Conference, 1996.
47. *Orbit Raising with Ion Propulsion on ESA's ARTEMIS Satellite - Paper No. AIAA 2002-3672.* **Killinger, R., Kukies, R., Surauer, M., van Holtz, L., Tomasetto, A.** Indianapolis, IN : AIAA/ASME/SAE/ASEE Joint Propulsion Conference, July 7-10, 2002.
48. *Ion Propulsion Development in the United Kingdom.* **Fearn, D. G., Martin, A. R., Smith, P.** Monterey, CA : 29th AIAA, SAE, ASME, ASEE Joint Propulsion Conference and Exhibit, June 28-30, 1993.
49. **Bryson, A. E., Ho, Y.** *Applied Optimal Control : Optimization, Estimation and Control Revised Printing.* New York, NY : Hemisphere Publishing Corp., Halsted Press, 1975.
50. **Kirk, D. E.,.** *Optimal Control Theory : An Introduction.* Englewood Cliffs, NJ : Prentice-Hall, 1970.

51. **How, J.** Principles of Optimal Control. *MIT Course 16.323 accessible with MIT Open Courseware at http://ocw.mit.edu.* 2007.
52. **Kechichian, J. A.** Personal communication through email. April 1, 2008.
53. *A Linear Theory of Optimum Low-Thrust Rendezvous Trajectories.* **Gobetz, F. W.** 3, pp. 69-76, s.l. : Journal of the Astronautical Sciences, 1965, Vol. 12.
54. *Optimum Low-Thrust Rendezvous and Station Keeping.* **Edelbaum, T. N.** 7, pp. 1196-1201, s.l. : AIAA Journal, 1964, Vol. 2.
55. *Optimal Low Thrust Geocentric Transfer - Paper No. 73-1074.* **Edelbaum, T. N., Sackett, L. L., Malchow, H. L.** s.l. : AIAA Journal , Nov. 1973.
56. *Equinoctial Orbit Elements : Application to Artificial Satellite Orbits - AIAA Paper No. 72-937.* **Cefola, P. J.** Oct. 1973.
57. *The Treatment of the Earth Oblateness Effect in Trajectory Optimization in Equinoctial Coordinates.* **Kechichian, J. A.** 1, pp. 69-82, s.l. : Acta Astronautica, 1997, Vol. 40.
58. **Kahaner, D., Moler, C., Nash, S.** *Numerical Methods and Software.* Englewood Cliffs, NJ : Prentice-Hall, 1989.
59. *Stationkeeping the Hughes HS 702 Satellite with a Xenon Ion Propulsion System - Paper No. IAF-98-A.1.09.* **Anzel, B.** Melbourne, Australia : 49th International Astronautical Congress, 1998.
60. Radiocommunication Sector (ITU-R). *ITU website.* [Online] ITU, Place des Nations, CH-1211 Geneva Switzerland. http://www.itu.int/ITU-R.
61. **Escobal, P. R.** *Methods of Orbit Determination - Krieger Reprint Edition.* New York, NY : John Wiley, 1985.
62. **Yurasov, V. S., Nazarenko, A. I.** *Atmosphere Density Tracking, Density Corrections over the Last Three Years - Stage 2 Report prepared for Texas A&M University.* College Station, TX : s.n., Nov. 2007.
63. *Dynamic Orbit Determination using GPS Measurements from TOPEX/POSEIDON.* **Shutz, B. E., Tapley, B. D., Abusali, P. A. M., Rim, H. J.** 19, pp. 2179-2182, s.l. : American Geophysical Union, Geophysical Research Letters, 1994, Vol. 21.
64. *A New Approach to Linear Filtering and Predication Problems.* **Kalman, R. E.** pp. 35-45, s.l. : Transactions of the ASME - Journal of Basic Engineering, 1960, Vol. 82.
65. *A New Approach for Filtering Nonlinear Systems.* **Julier, S. J., Uhlmann, J. K.** Seattle, WA : Proceedings of the American Control Conference, 1995.
66. *A New Extension of the Kalman Filter to Nonlinear Systems.* **Julier, S. J., Uhlmann, J. K.** s.l. : International Symposium Aerospace/Defense Sensing, Simulation and Controls, 1997.
67. **Gelb, A. Editor.** *Applied Optimal Estimation.* Cambridge, MA : MIT Press, 1974.
68. **Wornell, G.** Algorithms for Estimation and Inference. *MIT Course 6.438 accessible through MIT Open Courseware http://ocw.mit.edu.* 2007-2008.
69. **Bar-Shalom, Li, Kirubarajan.** *Estimation with Applications to Tracking and Navigation.* New York, NY : John Wiley & Sons, Inc., 2001.
70. **Tapley, Shutz, Born.** *Statistical Orbit Determination.* s.l. : Elsevier Academic Press, 2004.
71. **Stengal, R. F.** *Optimal Control and Estimation.* New York, NY : Dover Publications, 1994.

72. **Dunham, J. B.** *R&D GTDS Filter Program Software Specifications and User's Guide.* s.l. : Computer Sciences Corporation report CSC/SD-79/6032, January 1980.
73. **Duda, R. O.** Pattern Recognition for HCI. [Online] 2001. http://www.engr.sjsu.edu/~knapp/HCIRODPR/PR_home.htm.
74. **Meditch, S.** *Stochastic Optimal Linear Estimation and Control.* New York, NY : McGraw-Hill, 1969.
75. *Solutions to the Linear Smoothing Problem.* **Rauch, H. E.** s.l. : IEEE Trans. Auto. Control, AC-8:371, 1963.
76. *Maximum Likelihood Estimates of Linear Dynamic Systems.* **Rauch, H. E., Tung, F., Striebel, C. T.** pp. 1445-1450, No. 8, s.l. : AIAA Journal, 1965, Vol. 3.
77. **Brown, R. G., Hwang, P. Y. C.** *Introduction to Random Signals and Applied Kalman Filtering.* New York, NY : John Wiley & Sons, Inc., 1997.
78. *A Survey of Discrete Kalman-Bucy Filtering with Unknown Noise Covariances - Paper No. 70-955.* **Weiss, I. M.** Santa Barbara, CA : AIAA Guidance, Control and Flight Mechanics Conference, Aug. 1970.
79. **Bierman, G. J.** *Factorization Methods for Discrete Sequential Estimation.* New York, NY : Academic Press, pp. 69-76, 115-122, and 214-217. 1977.
80. *The Unscented Kalman Filter for Nonlinear Estimation.* **Wan, E., van der Merwe, R.** s.l. : IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium AS-SPCC, 2000.
81. *A New Method for the Nonlinear Transformation of Means and Covariances in Filters and Estimators.* **Julier, S., Uhlmann, J., Durrant-Whyte, H. F.** 3, pp. 477-482, s.l. : IEEE Transactions on Automatic Control, 2000, Vols. AC-45.
82. *Derivation and Simulation Testing of a Sigma-Points Smoother.* **Psiaki, M.** 1, s.l. : Journal of Guidance, Control and Dynamics, Jan-Feb. 2007, Vol. 30.
83. **Herklotz, R. L.** Incorporation of Cross-Link Range Measurements in the Orbit Determination Process to Increase Satellite Constellation Autonomy. *Ph.D. Thesis at the Massachusetts Institute of Technology.* Cambridge, MA : s.n., 1987.
84. **Wagner, E.** Application of the Extended Semianalytic Kalman Filter to Synchronous Orbits. *Master's Thesis at the Massachusetts Institute of Technology.* Cambridge, MA : s.n., June 1983.
85. **Fehlberg, E.** *Technical Report TR R-287.* Huntsville, AL : NASA Marshall Space Flight Center, 1968.
86. **A., Kechichian. J.** Personal communication through email. Jan 21, 2008.
87. *Final Report on the ARTEMIS Salvage Mission using Electric Propulsion - Paper No. AIAA 2003-4546.* **Killinger, R., Kukies, R., Surauer, M., Saccoccia, G., Gray, H.** Huntsville, AL : 39th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, July 20-23, 2003.
88. **Mazzini, Leonardo.** Personal communication through email. March 31, 2008.
89. *Final Stage of ARTEMIS Orbit Raising.* **Notarantonio, A., Cambriles, A. P., Amorosi, L.** San Diego, CA : 2nd AIAA Unmanned Unlimited Systems, Technologies, and Operations - AIAA 2003-6602, 15-18 Sept. 2003.
90. **Kelso, T. S.** Celestrak. [Online] [Cited: April 1, 2008.] http://www.celestrak.com.
91. **Mann, P. S.** *Introductory Statistics 6th Edition.* Hoboken, NJ : John Wiley & Sons, 2007.

92. *Optimal Measurement Filtering and Motion Prediction Taking into Account the Atmospheric Perturbations.* **Nazarenko, A. I., Yurasov, V. S., Alfriend, K. T., Cefola, P. J.** Mackinac Island, MI : Astrodynamics 2007 - Advances in the Astronautical Sciences Volume 129 - Proceedings of the AAS/AIAA Astrodynamics Specialist Conference, Aug. 19-23, 2007.
93. *Singular Value Decomposition and Least Squares Orbit Determination.* **Boikov, V., Khutorovsky, Z. N., Alfriend, K. T.** Mackinac Island, MI : Astrodynamics 2007 - Advances in the Astronautical Sciences Volumn 129 - Proceedings of the AAS/AIAA Astrodynamics Specialist Conference, August 19-23, 2007.
94. **Kechichian, J. A.** Personal communication through email. Jan 21, 2008.
95. **Spellucci, P.** Nonlinear (local) Optimization State of the Art. *Accessible on the web at http://www.mathematik.tu-darmstadt.de:8080/ags/ag8/Mitglieder/spellucci/docs/stateoftheartnlo.ps.gz.* March, 2008.
96. **Thaiprayoon, R., Homsup, N.** Study on Effects of Yaw and Co-Location on Cross-Polarized Signal Variation. [Online] April 7, 2008. [Cited: April 7, 2008.] https://pindex.ku.ac.th/file_research/EE53.doc.