A Variable-Step Double-Integration Multi-Step Integrator

Matthew M. Berry

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Aerospace Engineering

Dr. Christopher Hall, Committee Chair
Dr. Liam Healy, Committee Member
Dr. Lee Johnson, Committee Member
Dr. Hanspeter Schaub, Committee Member
Dr. Craig Woolsey, Committee Member

April 16, 2004
Blacksburg, Virginia

Keywords: Numerical Integration, Orbit Propagation, Orbit Determination,
Variable-Step Integration

A Variable-Step Double-Integration Multi-Step Integrator

Matthew M. Berry

Abstract

A new method of numerical integration is presented here, the variable-step Störmer-Cowell method. The method uses error control to regulate the step size, so larger step sizes can be taken when possible, and is double-integration, so only one evaluation per step is necessary when integrating second-order differential equations. The method is not variable-order, because variable-order algorithms require a second evaluation.

The variable-step Störmer-Cowell method is designed for space surveillance applications, which require numerical integration methods to track orbiting objects accurately. Because of the large number of objects being processed, methods that can integrate the equations of motion as fast as possible while maintaining accuracy requirements are desired. The force model used for earth-orbiting objects is quite complex and computationally expensive, so methods that minimize the force model evaluations are needed.

The new method is compared to the fixed-step Gauss-Jackson method, as well as a method of analytic step regulation (s-integration), and the variable-step variable-order Shampine-Gordon integrator. Speed and accuracy tests of these methods indicate that the new method is comparable in speed and accuracy to s-integration in most cases, though the variable-step Störmer-Cowell method has an advantage over s-integration when drag is a significant factor. The new method is faster than the Shampine-Gordon integrator, because the Shampine-Gordon integrator uses two evaluations per step, and is biased toward keeping the step size constant. Tests indicate that both the new variable-step Störmer-Cowell method and s-integration have an advantage over the fixed-step Gauss-Jackson method for orbits with eccentricities greater than 0.15.

In Memory of Dr. Frederick H. Lutze, Jr.

Acknowledgments

This work could not have been completed without the help and support of many individuals. First, I would like to acknowledge my late advisor, Dr. Frederick Lutze. Dr. Lutze provided support and encouragement whenever it was needed, which was quite often. He is missed dearly by both his students and colleagues.

Next, I must thank Liam Healy, who gave direction to this work and provided constant assistance. Some of this work was originally presented in conference papers which he co-authored. Liam has acted as a mentor to me over the years, and I owe much of what I've accomplished to his support.

I would also like to thank Christopher Hall for taking over as my advisor and quickly coming up to speed on my work. The other members of my committee, Lee Johnson, Hanspeter Schaub and Craig Woolsey, have also provided assistance through lengthy discussion of this work.

This work was funded by the Naval Research Laboratory, and I thank Shannon Coffey for providing the funding. Alan Segerman supported this work through many lengthy discussions, and Keith Akins provided much needed technical support.

I would also like to thank members of the astrodynamics community for their interest in this work, and for providing insight and feedback at the AAS meetings. Paul Schumacher provided both direction and insight into the issues involved in this work. Felix Hoots provided insight into ephemeris compression techniques, and Paul Cefola and Bob Schutz provided references to relevant literature.

The support of my parents, family, and friends has also been invaluable over the years in my career as a student.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The United States has two space surveillance centers, Air Force Space Command and Naval Networks and Space Operations Command (NNSOC), that track objects in Earth orbit. These centers are currently tracking over 12,000 objects. Most of these objects are space debris and pose a risk to other spacecraft, including manned spacecraft, with which they may collide. The centers track these objects for several reasons, one of which is to predict when a collision may occur between one object and a manned or otherwise important spacecraft, known as an asset, so the collision can be avoided by moving the asset. The centers also track objects so that new objects can be detected when they are launched.

The centers track the objects by first taking observations of the objects from two main sources, the Space Surveillance Network (SSN), maintained by the Air Force, and the Naval Space Surveillance Network, known as the Fence. The SSN consists of radar and optical sensors that target known objects. Observations from the SSN are in the form of some combination of azimuth and elevation, range, and range rate, depending on the type of sensor used. The Fence is a non-targeted system, which has three transmitters and six non-colocated receivers. Radar waves originating from the transmitters are reflected by orbiting objects and detected by the receivers, resulting in directional observations in the form of direction cosines. The direction cosines give the objects' position in a topocentric frame [1]. Orbit determination is performed from the SSN and Fence observations through a differential correction process that performs a least squares fit on the observations to give updated orbital parameters. Orbit parameters are either in the form of orbital elements or position and velocity. In the differential correction, the initial conditions are propagated to each observation time, and the difference, or residual, between the actual observation and the propagated observation is found. The residuals are then used

to correct the orbit parameters ([2] pp. 682-702). After the orbit parameters of all the objects have been updated, the parameters are propagated forward several days and compared to a propagation of assets of concern to predict collisions.

Both the orbit determination process and the collision prediction process require an orbit propagator, which gives the state at some time before or after an epoch state. The propagator solves the equations of motion

$$\ddot{\boldsymbol{r}} = -\frac{\mu}{r^3}\boldsymbol{r} + \boldsymbol{P}, \tag{1.1}$$

where $\boldsymbol{r}$ is the inertial position vector, $\ddot{\boldsymbol{r}}$ is the acceleration vector, μ is the gravitational parameter, and $\boldsymbol{P}$ is the perturbation force per unit mass. The equations of motion can be solved by integrating either analytically or numerically. To solve the equations analytically, the perturbation force is truncated to an expression that can be analytically propagated. This procedure is known as general perturbations (GP). To solve the equations numerically, a full force model is used with a numerical integrator. This method is known as special perturbations (SP).

In the past, the space surveillance centers have primarily used GP propagators. Because GP propagators use an analytic solution, they are faster than SP propagators. However, because the perturbation force is truncated, the analytic solution is less accurate than is possible with numerical integration. With the launch of the International Space Station, the accuracy requirements for tracking space objects have increased to the point where GP propagators are no longer adequate. The space surveillance centers have therefore adopted special perturbations systems. However, using special perturbations greatly increases the computing capability needed by the centers. Moreover, an upgrade is planned for the Fence that will allow for the detection of smaller objects, which could increase the number of tracked objects by an order of magnitude. This upgrade will significantly increase the amount of computing capability required. Methods of numerical integration that require less computation time than the methods currently employed would reduce the burden.

Numerous methods exist to perform numerical integration, and many are commonly used in orbit propagation. Integrators fall into various categories, which are described in Chapter 2. Integrators are either single-step or multi-step, which refers to the number of points that are used when integrating to the next point. Integrators may either have a fixed or a variable step size. Multi-step integrators come in two forms, summed and non-summed, which refers to whether the integration is performed from epoch or step-by-step, and is described in Section 2.3.4. The summed form contains a summation term which allows for an integration directly from epoch. Finally, integrators can perform either single or double integration. Single-integration integrators find velocity given acceleration, and

position given velocity. Double-integration integrators compute position directly from acceleration, bypassing the intermediate velocity calculation.

Table 1.1 lists the features of several integrators commonly used in astrodynamics. These integrators are described in detail in Chapter 2. Each integrator has one of two possibilities in the four categories. In general, multi-step integrators are advantageous over single-step integrators, because multi-step integrators can take larger step sizes for a given accuracy requirement[3]; however, there are some disadvantages of multi-step integration discussed in Section 2.3. Variable-step integration is advantageous over fixed-step integration because variable-step integration handles elliptical orbits more efficiently. The benefits of variable-step integration are discussed in Section 2.5. Summed integrators are advantageous over non-summed integrators because the summed form reduces round-off error[4]. Double integration also reduces the round-off error compared to single integration, because position is found by only applying the integrator once instead of twice[4], [3]. Evidence to support these claims is presented in Section 4.1.

Table 1.1: Summary of Integration Methods

Method	Single / Multi	Fixed / Variable	Non-Summed / Summed	Single / Double
Runge-Kutta	Single	Fixed	NA	Single
Runge-Kutta-Fehlberg	Single	Variable	NA	Single
Adams	Multi	Fixed	Non-Summed	Single
Summed Adams	Multi	Fixed	Summed	Single
Shampine-Gordon	Multi	Variable	Non-Summed	Single
Störmer-Cowell	Multi	Fixed	Non-Summed	Double
Gauss-Jackson	Multi	Fixed	Summed	Double
Proposed	Multi	Variable	Non-Summed	Double

The proposed integrator is a variable-step double-integration multi-step integrator, using the non-summed form, essentially a variable-step form of the Störmer-Cowell integrator. This integrator has the best of each possible feature, except that it is non-summed. Issues regarding the development of a summed variable-step double-integration method are discussed in Chapter 6. The proposed integrator is derived by following the derivation of Shampine-Gordon, which is a variable-step single-integration multi-step integrator, and applying that derivation to double integration.

Numerically solving the equations of motion in the form of (1.1) is known as Cowell's formulation. An alternative formulation, known as Encke's formulation, involves only

integrating the perturbation force, and adding that result to the known solution of the two-body problem. All of the testing done in this work is on Cowell's formulation, though the integration methods discussed can be used in Encke's formulation as well.

Chapter 2 describes integrators commonly used in astrodynamics and focuses on the derivation of multi-step integrators, including Störmer-Cowell and Shampine-Gordon. Chapter 3 describes procedures for testing the accuracy of numerical integrators, which are used both to motivate the need for the proposed integrator in Section 4.1, and to test the integrator against existing methods in Chapter 5. Following the motivation in Section 4.1, the rest of Chapter 4 derives the variable-step double-integration multi-step integrator, and describes how the integrator is implemented. Chapter 5 compares the proposed integrator to existing methods, using both orbit propagation and orbit determination tests. Conclusions and suggestions for further study are presented in Chapter 6.

Chapter 2

Numerical Integrators

2.1 Introduction

Equations of the form

$$\frac{dy}{dt} = f(t, y), \tag{2.1}$$

in which the derivative of a dependent variable is a function of that variable, are known as ordinary differential equations (ODEs). In order to find a specific solution of a differential equation the initial conditions $(t_0, y(t_0))$ must be given. A given differential equation along with initial conditions is known as an initial value problem. The solution to an initial value problem, $y(t)$, can be found analytically for some differential equations. However, most differential equations must be solved numerically. Algorithms that numerically solve initial value problems are known as numerical integrators. Numerical integrators give an approximate solution at distinct values of t, known as mesh points. The numerical solution at a mesh point t_n is denoted y_n, and due to error in the numerical method is different from the exact solution, $y(t_n)$.

The differential equation in (2.1) is called a first-order differential equation because the equation is for the first derivative. Differential equations may have a higher order,

$$\frac{d^n y}{dt^n} = f(t, y, y', \ldots, y^{(n-1)}), \tag{2.2}$$

where n is the order of differential equation. Initial value problems involving higher order ODEs require initial conditions for each derivative through $y^{(n-1)}$. Though higher order ODEs may be rewritten as a system of first order equations and solved with a standard numerical integrator, some numerical integrators are specifically designed to handle higher order ODEs. Integrators that solve second-order ODEs are called double-integration

methods, while integrators that solve first-order ODEs are called single-integration methods.

2.2 Single-Step Integrators

Integrators that integrate forward to the next point y_{n+1}, using only information from the current point y_n, are known as single-step integrators, because only information from a single step is used. Multi-step integrators, which use information from several previous points $y_{n-i} \dots y_n$, are described in Section 2.3. Though this work focuses on multi-step integration, single-step integrators have a simpler form, so some of the basic concepts of numerical integration are more easily demonstrated using single-step integrators as an example.

2.2.1 Euler's Method

Euler's method is the most basic numerical integration algorithm. Given the conditions at some mesh point (t_n, y_n) for a differential equation (2.1), the slope can be computed,

$$\frac{dy}{dt} = f(t_n, y_n). \tag{2.3}$$

The slope is used to approximate the change in y to the next mesh point,

$$\begin{aligned} \frac{dy}{dt} \approx \frac{\Delta y}{\Delta t} &= f(t_n, y_n) \\ \Delta y &= f(t_n, y_n)\Delta t, \end{aligned} \tag{2.4}$$

so the next y value can be approximated,

$$y_{n+1} = y_n + \Delta y. \tag{2.5}$$

This method is repeated to give an algorithm for solving differential equations,

$$\begin{aligned} y_{n+1} &= y_n + f(t_n, y_n)\Delta t \\ t_{n+1} &= t_n + \Delta t. \end{aligned} \tag{2.6}$$

This algorithm is **Euler's method** for solving initial value problems. The algorithm is started with the initial conditions (t_0, y_0). The value of Δt is known as the **step size**.

The error of Euler's method can be found by examining a Taylor series,

$$y_{n+1} = y_n + f'(t_n)(t_{n+1} - t_n) + \frac{f''(t_n)}{2}(t_{n+1} - t_n)^2 + \ \ldots . \tag{2.7}$$

Using the step size $\Delta t = t_{n+1} - t_n$ in the Taylor series,

$$y_{n+1} = y_n + f'(t_n)\Delta t + \frac{1}{2}f''(t_n)\Delta t^2 + \ldots , \tag{2.8}$$

shows that the first two terms of the series,

$$y_{n+1} = y_n + f'(t_n)\Delta t, \tag{2.9}$$

are Euler's Method, so the additional terms,

$$\frac{1}{2}f''(t_n)\Delta t^2 + \ldots , \tag{2.10}$$

comprise the **truncation error** for Euler's method. The error is called truncation error because it represents terms that have been truncated from the Taylor series.

The truncation error is on the order of Δt^2, written $\mathrm{O}(\Delta t^2)$. This error is called **local error**, because it is the error at one step. Since the local error is second order, the method is locally correct to first order. Figure 2.1 shows how the local error at one step affects the next step. Because the numerical solution does not match the true solution, at the next step the method uses the wrong initial conditions, and so in effect attempts to solve a different, though related, problem. This build-up of local error is known as **global error**. The global error is of one order lower than the local error, so Euler's method has a first-order global error, and is globally correct to zeroth order. By convention, the order of an integrator is the order to which it is locally correct, or the order of its global error. Euler's method is called a first-order integrator.

The truncation error given in (2.10) is generally the largest source of integration error. However, additional error arises due to error in floating point arithmetic. Computers only store numbers to a certain precision, so some accuracy is lost every time a computation is performed. Error due to this source is called **round-off error** ([5], pp. 18-21). The total integration error is due to both truncation and round-off error. As the step size is reduced, the truncation error decreases; however, round-off error increases as the step size is reduced, because more calculations are performed. When using double-precision arithmetic, truncation error normally dominates over round-off error. The step size h where round-off error is the same order as truncation error is given by

$$h = \sqrt{\frac{2\delta}{M}}, \tag{2.11}$$

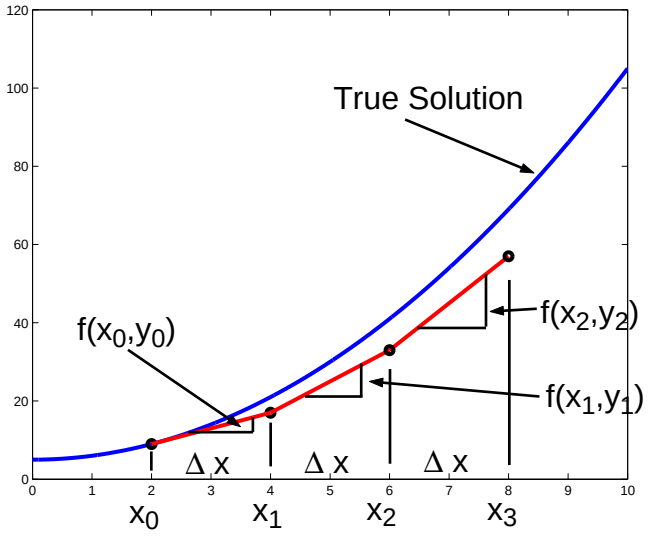


Figure 2.1: Local and Global Error for Euler's Method

where δ is the maximum round-off error in any step, and M is an upper bound on the second derivative, $|y''(t)| \leq M$ ([6], pp. 264-267). The value of δ depends on the value of machine epsilon, which is the precision to which the machine being used stores floating point numbers. The value of M depends on the nature of the initial value problem being solved.

2.2.2 Runge-Kutta

Runge-Kutta methods ([6], pp. 276-284) are a family of single-step integrators having the general form

$$y_{n+1} = y_n + \phi \Delta t, \tag{2.12}$$

where ϕ is a function of weighted slope estimates over the interval Δt, and has the form

$$\phi = a_1 k_1 + a_2 k_2 + \cdots + a_m k_m, \tag{2.13}$$

where k_i are slope estimates, m is the number of estimates, and a_i are weighting constants. The values of k_i are found by evaluating the function $f(t, y)$ at various points in the interval.

Heun's method ([6], p. 280) is an example of a second-order Runge-Kutta method. The slope at the beginning of the interval is averaged with the slope resulting from an Euler step to give a better estimate of the solution. The next mesh point y_{n+1} is found by the

algorithm

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + \Delta t, y_n + k_1 \Delta t), \\ a_1 &= 1/2, \\ a_2 &= 1/2, \\ y_{n+1} &= y_n + \left(\frac{1}{2} k_1 + \frac{1}{2} k_2 \right) \Delta t. \end{aligned}$$

The most widely used Runge-Kutta method is the "Classical Fourth Order" method. The algorithm uses a weighted average of four slope estimates ([6], p. 281),

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + \frac{1}{2}\Delta t, y_n + k_1 \frac{1}{2}\Delta t), \\ k_3 &= f(t_n + \frac{1}{2}\Delta t, y_n + k_2 \frac{1}{2}\Delta t), \\ k_4 &= f(t_n + \Delta t, y_n + k_3 \Delta t), \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta t. \end{aligned}$$

Because the fourth-order Runge-Kutta uses four slope estimates, the function $f(t, y)$ must be evaluated four times at each step. By contrast Heun's method requires two evaluations per step and Euler's method requires one. In space surveillance, the force model used is quite complex so these function evaluations are the biggest factor in the total computation time of orbit propagation. Results in Section 3.5 show that 90% of the total run-time of an integration is spent evaluating the force model.

For one integrator to be more efficient than another, it must have fewer total evaluations throughout the integration, while maintaining a specified accuracy. The total number of evaluations depends on two parameters: the number of evaluations per step, and the step size used. For instance, for the fourth-order Runge-Kutta to be more efficient than Heun's method, it must be able to take steps that are at least twice as large. In general, as single-step integrators such as Runge-Kutta increase in order, they become more accurate and can take larger steps to maintain a given accuracy. However, the number of evaluations per step increases with the order. On the other hand, multi-step integrators, described in the next section, have an advantage because the number of evaluations per step is not dependent on the order. Generally multi-step methods need only one or two evaluations per step.

2.3 Multi-Step Integrators

Multi-step integrators integrate forward to the next mesh point using function values at the current point as well as several previous mesh points. The set of the previous points used as well as the current point is called the set of **backpoints**. One disadvantage of multi-step integrators is that the initial set of backpoints must be found through some startup procedure, which complicates the method. Also, the function $f(t, y)$ must be continuous and smooth through the set of backpoints. If there are any discontinuities in f, for example going through eclipse when solar radiation pressure is considered, the integration must either be restarted, or modified to handle the discontinuity [7]. The studies performed here only consider continuous forces; however Chapter 6 discusses how variable-step multi-step integrators may be used to handle solar radiation pressure.

Multi-step integrators are known as predictor-corrector methods. The algorithms first predict a y value for the next mesh point and the function $f(t, y)$ is evaluated at the predicted point. That predicted function value is added to the set of backpoints, and a corrector formula is used with this revised set of backpoints to refine the predicted y value.

This procedure offers several variations on the implementation. First, when adding the predicted value to the set of backpoints, the farthest backpoint from the current point may or may not be dropped. If it is not dropped, the corrector is of higher order than the predictor because the corrector uses one more backpoint than the predictor. Otherwise, the predictor and corrector are of the same order. Also, a second evaluation may or may not be performed after the corrector. A second evaluation improves the accuracy of the method, because improved function values are used in the set of backpoints in the subsequent steps. However, the additional evaluation causes an increase in run-time. Methods that use one evaluation per step are known as Predict-Evaluate-Correct, or PEC, methods, and methods that perform a second evaluation are called PECE methods. Some implementations perform additional iterations of the corrector to meet some tolerance, and are called P(EC)^n or PE(CE)^n methods.

Multi-step integrators can be derived to be either fixed or variable step. The fixed-step implementations are considered here first; the variable-step derivations are found in Section 2.5 and Section 4.2. The methods can be derived in a variety of ways, but the fixed-step methods can be derived in a simplified form by using backward difference operators, which are described in the next section. The derivations of the Adams, summed-Adams, Störmer-Cowell, and Gauss-Jackson methods follow. These derivations follow the derivations presented by Maury and Segal [8], and the NORAD document known as TP008 [9].

2.3.1 Tables and Operators

Predictor-corrector integrators can be defined in terms of difference tables. For a fixed-step method, assume that the function $f(t, y)$ is known at a discrete set of equally-spaced t values, $\ldots$, $t_0 - 2h$, $t_0 - h$, t_0, $t_0 + h$, $t_0 + 2h$, $\ldots$, where h is the *step*. As shorthand, these values are referred to as $t_n = t_0 + nh$, so this set of five values is t_{-2}, t_{-1}, t_0, t_1, t_2. The corresponding function values are $f_n = f(t_n, y_n)$, where y_n is the numerical solution at the mesh point t_n (not the exact solution $y(t_n)$).

The differential equations are solved by taking differences of the function value. There are three kinds of differences, represented by different operators:

$$\begin{array}{lrcl} \text{forward difference} & \Delta f_i & = & f_{i+1} - f_i \\ \text{backward difference} & \nabla f_i & = & f_i - f_{i-1} \\ \text{central difference} & \delta f_i & = & f_{i+1/2} - f_{i-1/2}. \end{array}$$

Note that first central differences cannot be calculated directly, as the half-step points are not in the sequence of points, though the second central differences (see below) can be calculated. As previous authors ([3], [10]) have recognized, conversion between series in these operators is straightforward. The backward difference derivation is used here, because the majority of the time, when one is predicting or correcting, it is the most natural. Derivations using other difference operators give algebraically equivalent methods, though differences in their implementation may give slightly different results due to round-off error.

Not only can the differences of the function f be computed, the differences of the differences can be computed, and so on. For instance, the square of the backward difference operator is the operator applied to the first difference,

$$\begin{aligned} \nabla^2 f_i &= \nabla\nabla f_i = \nabla(f_i - f_{i-1}) \\ &= f_i - f_{i-1} - (f_{i-1} - f_{i-2}) = f_i - 2f_{i-1} + f_{i-2}. \end{aligned} \tag{2.14}$$

In general the n^{th} difference can be written in terms of the $(n-1)^{\text{th}}$ differences,

$$\nabla^n f_i = \nabla^{n-1} f_i - \nabla^{n-1} f_{i-1}. \tag{2.15}$$

The relationships of the backward differences are illustrated in Table 2.1. The arrows in the table point towards the difference; the upper component is always subtracted from the lower. For example, $\nabla^2 f_1 = \nabla f_1 - \nabla f_0$.

In addition to the three difference operators, there is also a displacement operator E. The displacement operator is defined as

$$E f_i = f_{i+1} \tag{2.16}$$

Table 2.1: Backward Difference Table

i	f_i	∇f_i	$\nabla^2 f_i$	$\nabla^3 f_i$
-3	f_{-3}	∇f_{-3}	$\nabla^2 f_{-3}$	$\nabla^3 f_{-3}$
-2	f_{-2}	∇f_{-2}	$\nabla^2 f_{-2}$	$\nabla^3 f_{-2}$
-1	f_{-1}	∇f_{-1}	$\nabla^2 f_{-1}$	$\nabla^3 f_{-1}$
0	f_0	∇f_0	$\nabla^2 f_0$	$\nabla^3 f_0$
1	f_1	∇f_1	$\nabla^2 f_1$	$\nabla^3 f_1$
2	f_2	∇f_2	$\nabla^2 f_2$	$\nabla^3 f_2$
3	f_3	∇f_3	$\nabla^2 f_3$	$\nabla^3 f_3$

Powers of this operator act as expected; e.g. $E^2 f_i = f_{i+2}$. The backward difference operator can be defined in terms of the displacement operator,

$$\nabla = 1 - E^{-1}, \tag{2.17}$$

and alternatively the displacement operator can be defined in terms of the backward difference operator,

$$E = \frac{1}{1 - \nabla}. \tag{2.18}$$

The summed integration formulas (see Section 2.3.4.), contain negative powers of the operators; e.g., ∇^{-2}. The backward difference table (Table 2.1) can be extended to the left to get negative powers of the backward difference operator, as in Table 2.2. Extending the differencing to the left gives $f_i = \nabla^{-1} f_i - \nabla^{-1} f_{i-1}$. This relation can be changed into a recursion formula that defines the inverse operator,

$$\nabla^{-1} f_i = f_i + \nabla^{-1} f_{i-1}. \tag{2.19}$$

Table 2.2: Inverse Backward Differences

i	$\nabla^{-2} f_i$	$\nabla^{-1} f_i$	f_i	∇f_i
-2	$C_2 - 2C_1 + f_0$	$C_1 - f_0 - f_{-1}$	f_{-2}	$f_{-2} - f_{-3}$
-1	$C_2 - C_1$	$C_1 - f_0$	f_{-1}	$f_{-1} - f_{-2}$
0	C_2	C_1	f_0	$f_0 - f_{-1}$
1	$C_2 + C_1 + f_1$	$C_1 + f_1$	f_1	$f_1 - f_0$
2	$C_2 + 2C_1 + 2f_1 + f_2$	$C_1 + f_1 + f_2$	f_2	$f_2 - f_1$

Note that the initial ($i = 0$) term in the sum is arbitrary; the difference relation holds no matter what this value is. This value is effectively an integration constant C_1. Thus, if the inverse operator is defined as a sum,

$$\nabla^{-1} f_i = \begin{cases} C_1 + \sum_{j=1}^{i} f_j, & \text{if } i \geq 1 \\ C_1 - \sum_{j=i+1}^{0} f_j, & \text{if } i \leq -1 \end{cases} \tag{2.20}$$

the difference relation (2.15) holds. The constant C_1 is discussed further in Section 2.3.9.

Powers of the summation ∇^{-1} are understood to be multiple sums, analogous to multiple differences. The presence of the operator ∇^{-2} in the Gauss-Jackson formulas sometimes gives rise to the name *second sum integration* formula. The second sum has an integration constant C_2, analogous to C_1.

2.3.2 Differentiation

Differentiation $D(\cdot) = d(\cdot)/dt$ is the operator that turns a function into its derivative function. The differentiation operator D can be represented in terms of E by noting the

following:

$$\begin{aligned} E^p f(t_0) &= f(t_0 + ph) \\ &= f(t_0) + phDf(t_0) + \frac{(ph)^2}{2!}D^2 f(t_0) + \frac{(ph)^3}{3!}D^3 f(t_0) + \dots \\ &= e^{phD} f(t_0), \end{aligned} \tag{2.21}$$

for any real number p, using a Taylor series expansion. The exponential of an operator is to be interpreted as its ordinary Taylor expansion,

$$e^t = 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \dots, \tag{2.22}$$

with powers of the operator well-defined. Thus we may identify the two operators

$$E^p = e^{phD}, \tag{2.23}$$

or taking the log

$$phD = p \log E. \tag{2.24}$$

This expression can be written in terms of the backward difference operator (2.18),

$$hD = \log E = -\log(1 - \nabla). \tag{2.25}$$

The integration methods can now be derived by taking the inverse of the differentiation operator. The Adams method is derived first, and the other methods are derived from it.

2.3.3 Adams Method

Indefinite integration is the operator inverse of differentiation,

$$\int (\cdot) dt = D^{-1}(\cdot), \tag{2.26}$$

so its action on an arbitrary function is to give the anti-derivative,

$$F(t) = D^{-1} f(t). \tag{2.27}$$

The single-integration operator D^{-1} is computed as the inverse of the differentiation operator (2.25),

$$D^{-1} = -\frac{h}{\log(1 - \nabla)}. \tag{2.28}$$

With this operator Adams integration can be developed. The focus here is satellite orbits, so the function f is replaced with acceleration $\ddot{\boldsymbol{r}}$.

The definite integral is computed by using the displacement operator on the indefinite integral,

$$(E^p - 1)D^{-1}\ddot{\boldsymbol{r}}_n = (E^p - 1)\dot{\boldsymbol{r}}_n = \int_{t_n}^{t_n+ph} \ddot{\boldsymbol{r}} dt. \tag{2.29}$$

For one timestep, p is 1. This integration corresponds to a *predictor* in a multi-step numerical integration, because the equation finds a value of $\dot{\boldsymbol{r}}$ at a future time. If $p = 1$, the predictor operator J can be written in terms of the backward difference operator, as

$$\begin{aligned} J &= \frac{1}{h}(E-1)D^{-1} = \frac{1}{h}\left[(1-\nabla)^{-1} - 1\right] D^{-1} = \frac{1}{h}\frac{\nabla}{(1-\nabla)}D^{-1} \\ &= -\frac{\nabla}{(1-\nabla)\log(1-\nabla)}. \end{aligned} \tag{2.30}$$

To integrate from $t_n - h$ to t_n, shift backward using the displacement operator (2.18),

$$E^{-1}J = (1-\nabla)J = -\frac{\nabla}{\log(1-\nabla)}. \tag{2.31}$$

This operation corresponds to the *corrector* in a multi-step numerical integration, because it assumes that the n^{th} value is already known and uses that value to recalculate a new, better, value. This operator has the simplest expansion so it is considered first.

The coefficients of the operators may be developed using a recursion relation [11]. For convenience in developing expansions, the corrector operator L is defined with its Taylor expansion using coefficients c_i as

$$L = E^{-1}J = -\frac{\nabla}{\log(1-\nabla)} = c_0 + c_1\nabla + c_2\nabla^2 + c_3\nabla^3 + \dots. \tag{2.32}$$

The expansion of L is developed recursively through the standard Taylor series of the logarithm,

$$\log(1-\nabla) = -\nabla - \frac{1}{2}\nabla^2 - \frac{1}{3}\nabla^3 - \frac{1}{4}\nabla^4 \dots, \tag{2.33}$$

which can be substituted in the definition of L,

$$L = -\frac{\nabla}{\log(1-\nabla)} = \frac{1}{1 + \frac{1}{2}\nabla + \frac{1}{3}\nabla^2 + \frac{1}{4}\nabla^3 + \dots} = \sum_{i=0}^{\infty} c_i \nabla^i, \tag{2.34}$$

and then expanded in the unknowns c_i,

$$\left(c_0 + c_1\nabla + c_2\nabla^2 + c_3\nabla^3 + \dots\right)\left(1 + \frac{1}{2}\nabla + \frac{1}{3}\nabla^2 + \frac{1}{4}\nabla^3 + \dots\right) = 1. \tag{2.35}$$

Expanding and grouping by powers of ∇,

$$c_0 + \left(\frac{1}{2}c_0 + c_1\right)\nabla + \left(\frac{1}{3}c_0 + \frac{1}{2}c_1 + c_2\right)\nabla^2 + \left(\frac{1}{4}c_0 + \frac{1}{3}c_1 + \frac{1}{2}c_2 + c_3\right)\nabla^3 + \ldots = 1, \quad (2.36)$$

allows for the development of a recursion relation for the coefficients

$$1 = c_0, \quad (2.37a)$$

$$0 = \frac{1}{2}c_0 + c_1, \quad (2.37b)$$

$$0 = \frac{1}{3}c_0 + \frac{1}{2}c_1 + c_2, \quad (2.37c)$$

or, in general,

$$c_n = -\sum_{i=0}^{n-1} \frac{c_i}{n+1-i} \quad (2.38)$$

for $n \geq 1$ with $c_0 = 1$. The first few coefficients are

i	0	1	2	3	4	5	6	7	8
c_i	1	$-\frac{1}{2}$	$-\frac{1}{12}$	$-\frac{1}{24}$	$-\frac{19}{720}$	$-\frac{3}{160}$	$-\frac{863}{60480}$	$-\frac{275}{24192}$	$-\frac{33953}{3628800}$

. (2.39)

With the coefficients c_i known, integration from $t_n - h$ to t_n may be written in terms of L,

$$\int_{t_n-h}^{t_n} \ddot{\boldsymbol{r}} dt = hE^{-1}J\ddot{\boldsymbol{r}}_n = hL\ddot{\boldsymbol{r}}_n. \quad (2.40)$$

A formula for the corrected value can be found by performing the integration and using the coefficients c_i,

$$\dot{\boldsymbol{r}}_n = \dot{\boldsymbol{r}}_{n-1} + h\left(1 - \frac{1}{2}\nabla - \frac{1}{12}\nabla^2 - \frac{1}{24}\nabla^3 - \frac{19}{720}\nabla^4 - \frac{3}{160}\nabla^5 + \ldots\right)\ddot{\boldsymbol{r}}_n. \quad (2.41)$$

This formula is the *Adams-Moulton corrector formula* in difference form. This form is called the difference form because it uses the differences ∇^i. An alternate form, called the ordinate form, in which the differences are re-written in terms of the function values is described in Section 2.3.8.

Returning to the predictor (2.30), the expansion of J can be computed by noting that

$$J = (1-\nabla)^{-1}L = (1 + \nabla + \nabla^2 + \ldots)(c_0 + c_1\nabla + c_2\nabla + \ldots) = \sum_{i=0}^{\infty} \gamma_i \nabla^i. \quad (2.42)$$

Each coefficient γ_i can be expressed as the sum of the coefficients c_k, $k \leq i$

$$\gamma_i = \sum_{k=0}^{i} c_k, \tag{2.43}$$

so they may be computed directly from (2.39),

i	0	1	2	3	4	5	6	7	8
γ_i	1	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$	$\frac{19087}{60480}$	$\frac{5257}{17280}$	$\frac{1070017}{3628800}$

. (2.44)

and the integral is then

$$\int_{t_n}^{t_n+h} \ddot{\boldsymbol{r}} dt = hJ\ddot{\boldsymbol{r}}_n, \tag{2.45}$$

or,

$$\dot{\boldsymbol{r}}_{n+1} = \dot{\boldsymbol{r}}_n + h\left(1 + \frac{1}{2}\nabla + \frac{5}{12}\nabla^2 + \frac{3}{8}\nabla^3 + \frac{251}{720}\nabla^4 + \frac{95}{288}\nabla^5 + \dots\right)\ddot{\boldsymbol{r}}_n. \tag{2.46}$$

This formula is the *Adams-Bashforth predictor* formula, in difference form, so called because when applied to t_n it integrates forward to t_{n+1}.

In practice, given m known accelerations

$$\ddot{\boldsymbol{r}}_{n-m},\ \ddot{\boldsymbol{r}}_{n-m+1}, \dots,\ \ddot{\boldsymbol{r}}_n, \tag{2.47}$$

the backward difference table (Table 2.1) for the derivatives can be computed through ∇^{m-1}, a predicted value for $\dot{\boldsymbol{r}}_{n+1}$ is made based on (2.46), then that value is corrected with (2.41).

2.3.4 Summed Adams Method

An alternative form of single integration, called the summed form, can be developed using the summation operator ∇^{-1}. The Adams-Moulton corrector formula (2.41) may be written so both velocity terms are on the left side,

$$\dot{\boldsymbol{r}}_n - \dot{\boldsymbol{r}}_{n-1} = h\left(1 - \frac{1}{2}\nabla - \frac{1}{12}\nabla^2 - \frac{1}{24}\nabla^3 - \frac{19}{720}\nabla^4 - \frac{3}{160}\nabla^5 + \dots\right)\ddot{\boldsymbol{r}}_n. \tag{2.48}$$

The difference of the two velocity terms may be written using the backward difference operator,

$$\nabla\dot{\boldsymbol{r}}_n = h\left(1 - \frac{1}{2}\nabla - \frac{1}{12}\nabla^2 - \frac{1}{24}\nabla^3 - \frac{19}{720}\nabla^4 - \frac{3}{160}\nabla^5 + \dots\right)\ddot{\boldsymbol{r}}_n. \tag{2.49}$$

The corrected value of the velocity can be found by multiplying both sides of (2.49) by ∇^{-1} [8],

$$\dot{\boldsymbol{r}}_n = h\left(\nabla^{-1} - \frac{1}{2} - \frac{1}{12}\nabla - \frac{1}{24}\nabla^2 - \frac{19}{720}\nabla^3 - \frac{3}{160}\nabla^4 + \dots\right)\ddot{\boldsymbol{r}}_n. \tag{2.50}$$

This expression is the summed Adams corrector formula. The summed form uses the same coefficients as the Adams-Moulton corrector, but they have been shifted by one place. The formula is called the summed form because of the presence of the summation operator.

A similar process can be performed on the Adams-Bashforth predictor formula (2.46),

$$\dot{\boldsymbol{r}}_{n+1} = h\left(\nabla^{-1} + \frac{1}{2} + \frac{5}{12}\nabla + \frac{3}{8}\nabla^2 + \frac{251}{720}\nabla^3 + \frac{95}{288}\nabla^4 + \dots\right)\ddot{\boldsymbol{r}}_n, \tag{2.51}$$

giving the summed Adams predictor. The summed form gives the predicted or corrected value by integrating directly from epoch, while the non-summed form integrates from point to point. According to Henrici ([4], p. 327) the non-summed form is preferred for reducing the propagation of roundoff error.

2.3.5 Störmer-Cowell

To find a corrector formula for position, the Adams-Moulton corrector (2.40) can be multiplied by the corrector operator,

$$hL\int_{t_n-h}^{t_n} \ddot{\boldsymbol{r}}dt = hL(\dot{\boldsymbol{r}}_n - \dot{\boldsymbol{r}}_{n-1}) = h^2L^2\ddot{\boldsymbol{r}}_n. \tag{2.52}$$

Performing the corrector operation on the velocity terms, $hL\dot{\boldsymbol{r}}_n = \boldsymbol{r}_n - \boldsymbol{r}_{n-1}$, gives an expression for position,

$$\boldsymbol{r}_n = 2\boldsymbol{r}_{n-1} + \boldsymbol{r}_{n-2} + h^2L^2\ddot{\boldsymbol{r}}_n. \tag{2.53}$$

To find the coefficients in the expansion of L^2,

$$L^2 = q_0 + q_1\nabla + q_2\nabla^2 + \dots, \tag{2.54}$$

in terms of the Adams coefficients c (2.38), note that (2.54) is the square of (2.32),

$$q_i = \sum_{k=0}^{i} c_k c_{i-k}; \tag{2.55}$$

so the first nine coefficients are

$$\begin{array}{c|ccccccccc} i & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline q_i & 1 & -1 & \frac{1}{12} & 0 & -\frac{1}{240} & -\frac{1}{240} & -\frac{221}{60480} & -\frac{19}{6048} & -\frac{9829}{3628800} \end{array}. \tag{2.56}$$

The corrector formula is given by using these coefficients in (2.53),

$$\boldsymbol{r}_n = 2\boldsymbol{r}_{n-1} + \boldsymbol{r}_{n-2} + h^2\left(1 - \nabla + \frac{1}{12}\nabla^2 - \frac{1}{240}\nabla^4 - \frac{1}{240}\nabla^5 + \dots\right)\ddot{\boldsymbol{r}}_n. \tag{2.57}$$

This formula is the Cowell corrector formula. It is a double-integration formula, since it computes the position given acceleration.

As with single integration, the predictor formula is found by shifting where the operator is applied, in other words multiplying both sides of (2.53) by the shift operator E,

$$\boldsymbol{r}_{n+1} = 2\boldsymbol{r}_n + \boldsymbol{r}_{n-1} + h^2 EL^2\ddot{\boldsymbol{r}}_n. \tag{2.58}$$

Writing the shift operator in terms of ∇, (2.18), and multiplying by L^2,

$$EL^2 = (1-\nabla)^{-1}L^2 = (1 + \nabla + \nabla^2 + \dots)(q_0 + q_1\nabla + q_2\nabla^2 + \dots) = \sum_{i=1}^{\infty}\lambda_i\nabla^i, \tag{2.59}$$

shows that the predictor coefficients λ_i can be written as a sum of the corrector coefficients, just as with single integration. The predictor coefficients can be computed from (2.56),

$$\begin{array}{c|ccccccccc} i & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline \lambda_i & 1 & 0 & \frac{1}{12} & \frac{1}{12} & \frac{19}{240} & \frac{3}{40} & \frac{863}{12096} & \frac{275}{4032} & \frac{33953}{518400} \end{array}. \tag{2.60}$$

Using these coefficients in (2.58),

$$\boldsymbol{r}_{n+1} = 2\boldsymbol{r}_n + \boldsymbol{r}_{n-1} + h^2\left(1 + \frac{1}{12}\nabla^2 + \frac{1}{12}\nabla^3 + \frac{19}{240}\nabla^4 + \frac{3}{40}\nabla^5 + \cdots\right)\ddot{\boldsymbol{r}}_n, \tag{2.61}$$

gives the Störmer predictor formula.

2.3.6 Gauss-Jackson

The Gauss-Jackson method is the summed form of the Störmer-Cowell method. According to Herrick ([3], p. 12), the method was named Gauss-Jackson because of its appearance in a 1924 paper by Jackson [12]. The Gauss-Jackson method is also referred to as the second-sum method.

The Gauss-Jackson corrector can be derived from the Cowell corrector by noting that the position terms in (2.57) may be combined using a second backward difference,

$$\nabla^2 \boldsymbol{r}_n = h^2 \left(1 - \nabla + \frac{1}{12}\nabla^2 - \frac{1}{240}\nabla^4 - \frac{1}{240}\nabla^5 - \frac{221}{60480}\nabla^6 + \dots\right) \ddot{\boldsymbol{r}}_n. \tag{2.62}$$

Multiplying both sides by the second sum term, ∇^{-2}, gives an equation for position [8],

$$\boldsymbol{r}_n = h^2 \nabla^{-2} \left(1 - \nabla + \frac{1}{12}\nabla^2 - \frac{1}{240}\nabla^4 - \frac{1}{240}\nabla^5 - \frac{221}{60480}\nabla^6 + \dots\right) \ddot{\boldsymbol{r}}_n. \tag{2.63}$$

Note that $(1 - \nabla)\ddot{\boldsymbol{r}}_n = E^{-1}\ddot{\boldsymbol{r}}_n = \ddot{\boldsymbol{r}}_{n-1}$, so that the first two terms can be simplified,

$$\boldsymbol{r}_n = h^2 \left[\nabla^{-2}\ddot{\boldsymbol{r}}_{n-1} + \left(\frac{1}{12} - \frac{1}{240}\nabla^2 - \frac{1}{240}\nabla^3 - \frac{221}{60480}\nabla^4 + \dots\right) \ddot{\boldsymbol{r}}_n\right]; \tag{2.64}$$

this expression is the *Gauss-Jackson corrector* formula.

A similar process on the Störmer predictor (2.61) gives a summed predictor formula,

$$\boldsymbol{r}_{n+1} = h^2 \left[\nabla^{-2}\ddot{\boldsymbol{r}}_n + \left(\frac{1}{12} + \frac{1}{12}\nabla + \frac{19}{240}\nabla^2 + \frac{3}{40}\nabla^3 + \dots\right) \ddot{\boldsymbol{r}}_n\right], \tag{2.65}$$

which is the *Gauss-Jackson predictor* formula. Note that for the predictor the second sum operator acts on $\ddot{\boldsymbol{r}}_n$, while for the corrector it acts on $\ddot{\boldsymbol{r}}_{n-1}$.

2.3.7 Startup Formulas

In order to calculate the differences ∇^i, the difference table 2.1 must be calculated, which depends on having values of acceleration at the backpoints. To calculate the N^{th} difference $N + 1$ points must be known. The initial value problem only gives the initial conditions at epoch, so a startup procedure is required to find position and velocity, and then acceleration, at the other backpoints. One possible method is to use a single-step integrator, such as Runge-Kutta, to find the values at the initial set of backpoints. Another method, described here, is to use an iterative method that takes an initial guess of the backpoints and refines them with corrector formulas. These corrector formulas, which correct a value using not only previous points, but also future known points, may be called **mid-corrector** formulas.

The initial set of guess values can be found by using the analytic two-body solution. For a method that uses the N^{th} difference, $N + 1$ total points are needed, so N points in addition to epoch must be found. To reduce the error that comes from the two-body solution, instead of propagating N steps forward, $N/2$ steps are taken both forward and

backward. These points are numbered using a symmetric index, from $-N/2$ to $N/2$, with epoch numbered 0. This system requires that N be even, as it is for the integrators in this study. If N were odd an additional point would be needed either before or after epoch.

After the set of guess values is found, each value, other than epoch, is corrected with a mid-corrector formula, except for the last value, which uses the corrector formula. The *computation point* refers to the point that is being corrected. The formulas are numbered using the letter j as an index with symmetric index numbers. So the mid-correctors are numbered $j = -N/2$ to $j = N/2 - 1$, and the corrector is numbered $j = N/2$. Similarly the predictor can be considered the $j = N/2 + 1$ formula.

The mid-correctors are found from the corrector by shifting backwards where the operator is applied, using the shift operator $E^{-1} = 1 - \nabla$. Using the corrector operator L as an example, the relation of operators to coefficient sets can be seen on the diagram for an eighth-order method:

$$\begin{array}{lccl}
 & j = -4 & (1-\nabla)^8 L & \\
 & j = -3 & (1-\nabla)^7 L & \\
\text{mid-correctors} & \vdots & \vdots & \uparrow \text{ apply } 1-\nabla \\
 & j = 2 & (1-\nabla)^2 L & \downarrow \text{ apply } (1-\nabla)^{-1} \\
 & j = 3 & (1-\nabla) L & \\
\text{corrector} & j = 4 & L & \\
\text{predictor} & j = 5 & \dfrac{L}{1-\nabla} &
\end{array}$$

The coefficients for each mid-corrector may be computed from the next one by application of the operator $1 - \nabla$. For an arbitrary power series in ∇ with coefficients d, multiply the series:

$$(d_0 + d_1\nabla + d_2\nabla^2 + \dots)(1 - \nabla) = d_0 + (d_1 - d_0)\nabla + (d_2 - d_1)\nabla^2 + \dots \tag{2.66}$$

In other words, the coefficients for a particular value of j are just the differences of coefficients of adjacent values of powers of ∇ for the next higher value of j. As shown in the derivation of the Adams and Störmer-Cowell predictors from their correctors, the coefficients are also the sums of all the coefficients of the next lower j, through the

particular power of ∇,

$$\begin{aligned}(m_0 + m_1\nabla + m_2\nabla^2 + \dots)\frac{1}{1-\nabla} \\ &= (m_0 + m_1\nabla + m_2\nabla^2 + \dots)\left(1 + \nabla + \nabla^2 + \nabla^3 + \dots\right) \\ &= \sum_k \left(\sum_{i=0}^{k} m_i\right)\nabla^k. \end{aligned} \quad (2.67)$$

All the coefficients taken together may be viewed as an array, with the rows indexed by j, and the columns by the backcount i. For an N^{th}-order integrator, there are $N+1$ terms in the series expansion of the operator for each j, and there are $N+2$ values of j (including the predictor); therefore, the full array of coefficients has $(N+1)(N+2)$ elements. For example, the eighth-order integrator has $9 \times 10 = 90$ coefficients.

Coefficient arrays for the integrators described above may be found using this procedure. The arrays for the summed Adams and Gauss-Jackson methods are shown here, since those methods are used as the control for the study in Chapter 5. The array for the summed Adams coefficients is called b', and the array for the Gauss-Jackson coefficients is called a'. These arrays have two subscripts: the first is the number of the formula j, and the second is the power of ∇ to which the coefficient corresponds. For instance, the summed Adams corrector (2.50) may be written with b',

$$\dot{\boldsymbol{r}}_n = h\left(\nabla^{-1}\ddot{\boldsymbol{r}}_n + \sum_{i=0}^{N} b'_{\frac{N}{2},i}\nabla^i\ddot{\boldsymbol{r}}_n\right). \quad (2.68)$$

The last mid-corrector formula ($j = \frac{N}{2} - 1$) is obtained by applying E^{-1} to the corrector formula,

$$\begin{aligned}\dot{\boldsymbol{r}}_{\frac{N}{2}-1} =& E^{-1}\dot{\boldsymbol{r}}_{\frac{N}{2}} \\ =& E^{-1}h\left[\nabla^{-1}\ddot{\boldsymbol{r}}_{\frac{N}{2}} + \left(-\frac{1}{2} - \frac{1}{12}\nabla - \frac{1}{24}\nabla^2 - \frac{19}{720}\nabla^3 - \frac{3}{160}\nabla^4 + \dots\right)\ddot{\boldsymbol{r}}_{\frac{N}{2}}\right] \\ =& h\left[\nabla^{-1}\ddot{\boldsymbol{r}}_{\frac{N}{2}-1} + \left(-\frac{1}{2} + \frac{5}{12}\nabla + \frac{1}{24}\nabla^2 + \frac{11}{720}\nabla^3 + \frac{11}{1440}\nabla^4 + \dots\right)\ddot{\boldsymbol{r}}_{\frac{N}{2}}\right], \end{aligned} \quad (2.69)$$

keeping in mind $E^{-1} = 1 - \nabla$ from (2.17). Using b' as the coefficients, in general the mid-corrector formulas can be written as

$$\dot{\boldsymbol{r}}_n = h\left(\nabla^{-1}\ddot{\boldsymbol{r}}_n + \sum_{i=0}^{N} b'_{ni}\nabla^i\ddot{\boldsymbol{r}}_{\frac{N}{2}},\right) \quad (2.70)$$

where $-\frac{N}{2} \leq n \leq \frac{N}{2} - 1$. The coefficients b'_{ji} for each value of j are obtained by taking differences of the coefficients for $j+1$. The mid-corrector coefficients are thus computed recursively,

$$b'_{ji} = \begin{cases} b'_{j+1,i} - b'_{j+1,i-1} & \text{if } i > 0 \\ b'_{j+1,i} & \text{if } i = 0 \end{cases} \tag{2.71}$$

for $-\frac{N}{2} \leq j \leq \frac{N}{2} - 1$.

The predictor formula (2.51), may also be written using b',

$$\dot{\boldsymbol{r}}_{n+1} = h \left(\nabla^{-1} \ddot{\boldsymbol{r}}_n + \sum_{i=0}^{N} b'_{\frac{N}{2}+1,i} \nabla^i \ddot{\boldsymbol{r}}_n \right). \tag{2.72}$$

The eighth-order coefficients b'_{ji} are presented in Table 2.3.

Table 2.3: Eighth-Order Summed-Adams Difference Coefficients, b'_{ji}

	i								
j	0	1	2	3	4	5	6	7	8
-4	$-\frac{1}{2}$	$\frac{47}{12}$	$-\frac{107}{8}$	$\frac{18701}{720}$	$-\frac{45083}{1440}$	$\frac{1445281}{60480}$	$-\frac{1354079}{120960}$	$\frac{10468447}{3628800}$	$-\frac{25713}{89600}$
-3	$-\frac{1}{2}$	$\frac{41}{12}$	$-\frac{239}{24}$	$\frac{11531}{720}$	$-\frac{22021}{1440}$	$\frac{520399}{60480}$	$-\frac{11603}{4480}$	$\frac{1070017}{3628800}$	$\frac{8183}{1036800}$
-2	$-\frac{1}{2}$	$\frac{35}{12}$	$-\frac{169}{24}$	$\frac{6461}{720}$	$-\frac{1011}{160}$	$\frac{138241}{60480}$	$-\frac{5257}{17280}$	$-\frac{33953}{3628800}$	$-\frac{425}{290304}$
-1	$-\frac{1}{2}$	$\frac{29}{12}$	$-\frac{37}{8}$	$\frac{3131}{720}$	$-\frac{2837}{1440}$	$\frac{19087}{60480}$	$\frac{275}{24192}$	$\frac{7297}{3628800}$	$\frac{7}{12800}$
0	$-\frac{1}{2}$	$\frac{23}{12}$	$-\frac{65}{24}$	$\frac{1181}{720}$	$-\frac{95}{288}$	$-\frac{863}{60480}$	$-\frac{13}{4480}$	$-\frac{3233}{3628800}$	$-\frac{2497}{7257600}$
1	$-\frac{1}{2}$	$\frac{17}{12}$	$-\frac{31}{24}$	$\frac{251}{720}$	$\frac{3}{160}$	$\frac{271}{60480}$	$\frac{191}{120960}$	$\frac{2497}{3628800}$	$\frac{2497}{7257600}$
2	$-\frac{1}{2}$	$\frac{11}{12}$	$-\frac{3}{8}$	$-\frac{19}{720}$	$-\frac{11}{1440}$	$-\frac{191}{60480}$	$-\frac{191}{120960}$	$-\frac{3233}{3628800}$	$-\frac{7}{12800}$
3	$-\frac{1}{2}$	$\frac{5}{12}$	$\frac{1}{24}$	$\frac{11}{720}$	$\frac{11}{1440}$	$\frac{271}{60480}$	$\frac{13}{4480}$	$\frac{7297}{3628800}$	$\frac{425}{290304}$
4	$-\frac{1}{2}$	$-\frac{1}{12}$	$-\frac{1}{24}$	$-\frac{19}{720}$	$-\frac{3}{160}$	$-\frac{863}{60480}$	$-\frac{275}{24192}$	$-\frac{33953}{3628800}$	$-\frac{8183}{1036800}$
5	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$	$\frac{19087}{60480}$	$\frac{5257}{17280}$	$\frac{1070017}{3628800}$	$\frac{25713}{89600}$

The Gauss-Jackson corrector (2.64) can be written using the coefficients a',

$$r_n = h^2 \left(\nabla^{-2} \ddot{\boldsymbol{r}}_{n-1} + \sum_{i=0}^{N} a'_{\frac{N}{2},i} \nabla^i \ddot{\boldsymbol{r}}_n \right), \tag{2.73}$$

as can the predictor,

$$r_{n+1} = h^2 \left(\nabla^{-2} \ddot{\boldsymbol{r}}_n + \sum_{i=0}^{N} a'_{\frac{N}{2}+1,i} \nabla^i \ddot{\boldsymbol{r}}_n \right). \tag{2.74}$$

Table 2.4: Eighth-Order Gauss-Jackson Difference Coefficients, a'_{ji}

	i								
j	0	1	2	3	4	5	6	7	8
-4	$\frac{1}{12}$	$-\frac{2}{3}$	$\frac{559}{240}$	$-\frac{371}{80}$	$\frac{347539}{60480}$	$-\frac{45601}{10080}$	$\frac{7965611}{3628800}$	$-\frac{427487}{725760}$	$\frac{3250433}{53222400}$
-3	$\frac{1}{12}$	$-\frac{7}{12}$	$\frac{419}{240}$	$-\frac{347}{120}$	$\frac{172651}{60480}$	$-\frac{20191}{12096}$	$\frac{1908311}{3628800}$	$-\frac{8183}{129600}$	$-\frac{330157}{159667200}$
-2	$\frac{1}{12}$	$-\frac{1}{2}$	$\frac{299}{240}$	$-\frac{79}{48}$	$\frac{73111}{60480}$	$-\frac{6961}{15120}$	$\frac{33953}{518400}$	$\frac{407}{172800}$	$\frac{45911}{159667200}$
-1	$\frac{1}{12}$	$-\frac{5}{12}$	$\frac{199}{240}$	$-\frac{49}{60}$	$\frac{23719}{60480}$	$-\frac{275}{4032}$	$-\frac{9829}{3628800}$	$-\frac{641}{1814400}$	$-\frac{3499}{53222400}$
0	$\frac{1}{12}$	$-\frac{1}{3}$	$\frac{119}{240}$	$-\frac{77}{240}$	$\frac{863}{12096}$	$\frac{19}{6048}$	$\frac{1571}{3628800}$	$\frac{289}{3628800}$	$\frac{317}{22809600}$
1	$\frac{1}{12}$	$-\frac{1}{4}$	$\frac{59}{240}$	$-\frac{3}{40}$	$-\frac{221}{60480}$	$-\frac{31}{60480}$	$-\frac{289}{3628800}$	0	$\frac{317}{22809600}$
2	$\frac{1}{12}$	$-\frac{1}{6}$	$\frac{19}{240}$	$\frac{1}{240}$	$\frac{31}{60480}$	0	$-\frac{289}{3628800}$	$-\frac{289}{3628800}$	$-\frac{3499}{53222400}$
3	$\frac{1}{12}$	$-\frac{1}{12}$	$-\frac{1}{240}$	0	$\frac{31}{60480}$	$\frac{31}{60480}$	$\frac{1571}{3628800}$	$\frac{641}{1814400}$	$\frac{45911}{159667200}$
4	$\frac{1}{12}$	0	$-\frac{1}{240}$	$-\frac{1}{240}$	$-\frac{221}{60480}$	$-\frac{19}{6048}$	$-\frac{9829}{3628800}$	$-\frac{407}{172800}$	$-\frac{330157}{159667200}$
5	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{19}{240}$	$\frac{3}{40}$	$\frac{863}{12096}$	$\frac{275}{4032}$	$\frac{33953}{518400}$	$\frac{8183}{129600}$	$\frac{3250433}{53222400}$

The mid-correctors can also be written with the a' coefficients,

$$r_n = h^2 \left(\nabla^{-2} \ddot{\boldsymbol{r}}_{n-1} + \sum_{i=0}^{N} a'_{ni} \nabla^i \ddot{\boldsymbol{r}}_{\frac{N}{2}} \right), \tag{2.75}$$

where $-\frac{N}{2} \leq n \leq \frac{N}{2} - 1$. The coefficients of the mid-correctors are computed as the difference (2.66) of the corrector coefficients; the entire set of coefficients are thus computed recursively,

$$a'_{ji} = \begin{cases} a'_{j+1,i} - a'_{j+1,i-1} & \text{if } i > 0 \\ a'_{j+1,i} & \text{if } i = 0 \end{cases} \tag{2.76}$$

for $-\frac{N}{2} \leq j \leq \frac{N}{2} - 1$. The eighth-order coefficients a'_{ji} are presented in Table 2.4.

Note that the sum introduced by the ∇^{-2} term in the mid-correctors or corrector is accumulated through the point *before* the point being corrected; this is given by the $\ddot{\boldsymbol{r}}_{n-1}$ term in (2.64).

2.3.8 Ordinate Forms

The formulas given above all involve multiple differences and sums. In order to compute these formulas to a particular order, all the elements of the appropriate difference

tables need to be computed. Calculating the differences can be time-consuming and is unnecessary because the formulas can be re-expressed in terms of the function values themselves.

As shown above in (2.14), powers of the difference operators can be reduced to linear combinations of the original function values. The backward difference operator raised to any power can be expressed in terms of the binomial coefficients of the function points

$$\nabla^i \ddot{\boldsymbol{r}}_n = \sum_{m=0}^{i} \binom{i}{m} (-1)^m \ddot{\boldsymbol{r}}_{n-m}, \tag{2.77}$$

for $i \geq 0$. A sum over arbitrary coefficients z'_i of the backward difference operator can thus be written in terms of the values of the function

$$\begin{aligned} \sum_{i=0}^{N} z'_i \nabla^i \ddot{\boldsymbol{r}}_n &= \sum_{i=0}^{N} z'_i \sum_{m=0}^{i} \binom{i}{m} (-1)^m \ddot{\boldsymbol{r}}_{n-m} \\ &= \sum_{m=0}^{N} (-1)^m \ddot{\boldsymbol{r}}_{n-m} \sum_{i=m}^{N} z'_i \binom{i}{m} \\ &= \sum_{m=0}^{N} z_{Nm} \ddot{\boldsymbol{r}}_{n-m}, \end{aligned} \tag{2.78}$$

with the *ordinate coefficients* defined as

$$z_{Nm} = (-1)^m \sum_{i=m}^{N} z'_i \binom{i}{m}. \tag{2.79}$$

Notice that these coefficients, unlike the difference coefficients z', depend on N, the order of the integration method. Here the *reference point* index m numbers the backpoints so the current point is $m = 0$ and each previous point has a higher positive value of m. The coefficients can also be indexed with the symmetric index form and the letter k, so that $k = N/2 - m$.

As an example, consider the summed Adams corrector (2.50). For a fourth-order method, $N = 4$, the difference coefficients z'_i and the ordinate coefficients z_{4m} are

m	0	1	2	3	4
z'_i	$-\frac{1}{2}$	$-\frac{1}{12}$	$-\frac{1}{24}$	$-\frac{19}{720}$	$-\frac{3}{160}$
z_{4m}	$-\frac{193}{288}$	$\frac{77}{240}$	$-\frac{7}{30}$	$\frac{73}{720}$	$-\frac{3}{160}$

. (2.80)

An alternative formulation includes the ∇^0 term $-\frac{1}{2}\ddot{\boldsymbol{r}}_n$ with the sum rather than with the coefficients. The example Adams-Moulton corrector (2.50) is now written

$$\dot{\boldsymbol{r}}_n = h\left[\nabla^{-1}\ddot{\boldsymbol{r}}_n - \frac{1}{2}\ddot{\boldsymbol{r}}_n + \left(-\frac{1}{12}\nabla - \frac{1}{24}\nabla^2 - \frac{19}{720}\nabla^3 - \frac{3}{160}\nabla^4 + \dots\right)\ddot{\boldsymbol{r}}_n\right]. \tag{2.81}$$

With this formulation the $m = 0$ coefficients are different from (2.80):

m	0	1	2	3	4
z'_i	0	$-\frac{1}{12}$	$-\frac{1}{24}$	$-\frac{19}{720}$	$-\frac{3}{160}$
z_{Nm}	$-\frac{49}{288}$	$\frac{77}{240}$	$-\frac{7}{30}$	$\frac{73}{720}$	$-\frac{3}{160}$

. (2.82)

All the single-integration corrector and mid-corrector coefficients are computed this way; however, the ∇^0 term is included in the coefficients for the predictor. The eighth-order summed Adams coefficients in ordinate form b_{jk} are shown in Table 2.5, and the Gauss-Jackson coefficients a_{jk} are shown in Table 2.6. For Gauss-Jackson, the ∇^0 term $\frac{1}{12}\ddot{\boldsymbol{r}}_n$, is included with the coefficients and not the sum ∇^{-2}. Although this alternate formulation is not necessary, it makes the programming easier, as described in the next section. Both the Gauss-Jackson and the summed Adams coefficients may be compared with [9], Table III (pages 25–26) and [11] Appendix C.3. Ordinate forms for the Adams, summed Adams, Störmer-Cowell, and Gauss-Jackson predictors and correctors for orders between 1 and 14 may also be found in [8].

In this section, the z' are generic difference coefficients such as a' or b', and the z are the corresponding ordinate coefficients, which have an extra subscript to indicate the order. The actual eighth-order ordinate coefficients a or b do not have an order subscript even though they are dependent on the order; the two subscripts are the point of integration j and the point of reference k.

2.3.9 Implementation

To find both position and velocity given a function for acceleration, either a single-integration method must be used once for velocity and the method used again for position, or a single-integration method must be used along with a double-integration method. Assuming that summed and non-summed forms are not used together, the four integrators derived above offer four implementations: two Adams, two summed Adams, Adams and Störmer-Cowell, or summed Adams and Gauss-Jackson integration. For convenience, Gauss-Jackson integration is understood to mean Gauss-Jackson and summed Adams integration, and Störmer-Cowell integration is understood to mean Störmer-Cowell and Adams integration. The summed Adams and Gauss-Jackson combination is used in the tests in Chapter 5. Because these integrators involve sum terms, their implementation is somewhat complicated, and so is described in detail in this section.

Table 2.5: Eighth-Order Summed-Adams Coefficients in Ordinate Form, b_{jk}

	k								
j	-4	-3	-2	-1	0	1	2	3	4
-4	$\frac{19087}{89600}$	$-\frac{427487}{725760}$	$\frac{3498217}{3628800}$	$-\frac{500327}{403200}$	$\frac{6467}{5670}$	$-\frac{2616161}{3628800}$	$\frac{24019}{80640}$	$-\frac{263077}{3628800}$	$\frac{8183}{1036800}$
-3	$\frac{8183}{1036800}$	$\frac{57251}{403200}$	$-\frac{1106377}{3628800}$	$\frac{218483}{725760}$	$-\frac{69}{280}$	$\frac{530177}{3628800}$	$-\frac{210359}{3628800}$	$\frac{5533}{403200}$	$-\frac{425}{290304}$
-2	$-\frac{425}{290304}$	$\frac{76453}{3628800}$	$\frac{5143}{57600}$	$-\frac{660127}{3628800}$	$\frac{661}{5670}$	$-\frac{4997}{80640}$	$\frac{83927}{3628800}$	$-\frac{19109}{3628800}$	$\frac{7}{12800}$
-1	$\frac{7}{12800}$	$-\frac{23173}{3628800}$	$\frac{29579}{725760}$	$\frac{2497}{57600}$	$-\frac{2563}{22680}$	$\frac{172993}{3628800}$	$-\frac{6463}{403200}$	$\frac{2497}{725760}$	$-\frac{2497}{7257600}$
0	$-\frac{2497}{7257600}$	$\frac{1469}{403200}$	$-\frac{68119}{3628800}$	$\frac{252769}{3628800}$	0	$-\frac{252769}{3628800}$	$\frac{68119}{3628800}$	$-\frac{1469}{403200}$	$\frac{2497}{7257600}$
1	$\frac{2497}{7257600}$	$-\frac{2497}{725760}$	$\frac{6463}{403200}$	$-\frac{172993}{3628800}$	$\frac{2563}{22680}$	$-\frac{2497}{57600}$	$-\frac{29579}{725760}$	$\frac{23173}{3628800}$	$-\frac{7}{12800}$
2	$-\frac{7}{12800}$	$\frac{19109}{3628800}$	$-\frac{83927}{3628800}$	$\frac{4997}{80640}$	$-\frac{661}{5670}$	$\frac{660127}{3628800}$	$-\frac{5143}{57600}$	$-\frac{76453}{3628800}$	$\frac{425}{290304}$
3	$\frac{425}{290304}$	$-\frac{5533}{403200}$	$\frac{210359}{3628800}$	$-\frac{530177}{3628800}$	$\frac{69}{280}$	$-\frac{218483}{725760}$	$\frac{1106377}{3628800}$	$-\frac{57251}{403200}$	$-\frac{8183}{1036800}$
4	$-\frac{8183}{1036800}$	$\frac{263077}{3628800}$	$-\frac{24019}{80640}$	$\frac{2616161}{3628800}$	$-\frac{6467}{5670}$	$\frac{500327}{403200}$	$-\frac{3498217}{3628800}$	$\frac{427487}{725760}$	$-\frac{19087}{89600}$
5	$\frac{25713}{89600}$	$-\frac{9401029}{3628800}$	$\frac{5393233}{518400}$	$-\frac{9839609}{403200}$	$\frac{167287}{4536}$	$-\frac{135352319}{3628800}$	$\frac{10219841}{403200}$	$-\frac{40987771}{3628800}$	$\frac{3288521}{1036800}$

Table 2.6: Eighth-Order Gauss-Jackson Coefficients in Ordinate Form, a_{jk}

	k								
j	-4	-3	-2	-1	0	1	2	3	4
-4	$\frac{3250433}{53222400}$	$\frac{572741}{5702400}$	$-\frac{8701681}{39916800}$	$\frac{4026311}{13305600}$	$-\frac{917039}{3193344}$	$\frac{7370669}{39916800}$	$-\frac{1025779}{13305600}$	$\frac{754331}{39916800}$	$-\frac{330157}{159667200}$
-3	$-\frac{330157}{159667200}$	$\frac{530113}{6652800}$	$\frac{518887}{19958400}$	$-\frac{27631}{623700}$	$\frac{44773}{1064448}$	$-\frac{531521}{19958400}$	$\frac{109343}{9979200}$	$-\frac{1261}{475200}$	$\frac{45911}{159667200}$
-2	$\frac{45911}{159667200}$	$-\frac{185839}{39916800}$	$\frac{171137}{1900800}$	$\frac{73643}{39916800}$	$-\frac{25775}{3193344}$	$\frac{77597}{13305600}$	$-\frac{98911}{39916800}$	$\frac{24173}{39916800}$	$-\frac{3499}{53222400}$
-1	$-\frac{3499}{53222400}$	$\frac{4387}{4989600}$	$-\frac{35039}{4989600}$	$\frac{90817}{950400}$	$-\frac{20561}{3193344}$	$\frac{2117}{9979200}$	$\frac{2059}{6652800}$	$-\frac{317}{2851200}$	$\frac{317}{22809600}$
0	$\frac{317}{22809600}$	$-\frac{2539}{13305600}$	$\frac{55067}{39916800}$	$-\frac{326911}{39916800}$	$\frac{14797}{152064}$	$-\frac{326911}{39916800}$	$\frac{55067}{39916800}$	$-\frac{2539}{13305600}$	$\frac{317}{22809600}$
1	$\frac{317}{22809600}$	$-\frac{317}{2851200}$	$\frac{2059}{6652800}$	$\frac{2117}{9979200}$	$-\frac{20561}{3193344}$	$\frac{90817}{950400}$	$-\frac{35039}{4989600}$	$\frac{4387}{4989600}$	$-\frac{3499}{53222400}$
2	$-\frac{3499}{53222400}$	$\frac{24173}{39916800}$	$-\frac{98911}{39916800}$	$\frac{77597}{13305600}$	$-\frac{25775}{3193344}$	$\frac{73643}{39916800}$	$\frac{171137}{1900800}$	$-\frac{185839}{39916800}$	$\frac{45911}{159667200}$
3	$\frac{45911}{159667200}$	$-\frac{1261}{475200}$	$\frac{109343}{9979200}$	$-\frac{531521}{19958400}$	$\frac{44773}{1064448}$	$-\frac{27631}{623700}$	$\frac{518887}{19958400}$	$\frac{530113}{6652800}$	$-\frac{330157}{159667200}$
4	$-\frac{330157}{159667200}$	$\frac{754331}{39916800}$	$-\frac{1025779}{13305600}$	$\frac{7370669}{39916800}$	$-\frac{917039}{3193344}$	$\frac{4026311}{13305600}$	$-\frac{8701681}{39916800}$	$\frac{572741}{5702400}$	$\frac{3250433}{53222400}$
5	$\frac{3250433}{53222400}$	$-\frac{11011481}{19958400}$	$\frac{6322573}{2851200}$	$-\frac{8660609}{1663200}$	$\frac{25162927}{3193344}$	$-\frac{159314453}{19958400}$	$\frac{18071351}{3326400}$	$-\frac{24115843}{9979200}$	$\frac{103798439}{159667200}$

Overview

In order to use the predictor-corrector formulas to integrate forward in time with an N^{th} order method, $N+1$ points must already be known. To preserve the order of the method, these $N+1$ points must have also been found using an N^{th} order method. Since initially only one point, epoch, is known, a startup procedure is necessary. If $N+1$ initial estimated points are given, the mid-corrector and corrector formulas can be used to refine them. By applying the mid-correctors iteratively until the points converge, the resulting points are accurate through N^{th} order.

For the eighth-order methods used in this study, a Taylor expansion of the two-body solution to fifth order (f and g series, [2] Eq. (4–68)) is used for the initial estimate, though a single-step integrator, such as Runge-Kutta could be used. The analytic solution generates eight positions and velocities in addition to epoch: four before epoch and four after epoch. The accelerations at all nine points are then found from the positions and velocities, and corrector and mid-corrector formulas are applied to the eight non-epoch accelerations, which generates corrected positions and velocities. These corrected positions and velocities are used to find corrected accelerations, and the cycle repeats, until the accelerations converge, i.e., on two successive iterations, the absolute difference is less than a specified tolerance. This process can be denoted SECECE…CE, where "S" is startup estimate, "E" is evaluate, and "C" is correct. This process is done for the eight initial points other than epoch.

Once the startup points have been corrected to satisfactory accuracy, the regular predictor-corrector process can start. First the predictor equation is used to find the position and velocity of the first point after startup, and the force model is evaluated to find the acceleration of that point. This predicted acceleration is then used in the corrector formula to find a corrected position and velocity. If only performing one evaluation per step, PEC, the process stops here and the next value is predicted. If multiple evaluations are allowed, the corrected position and velocity are compared to the predicted values, and if they do not match to some tolerance, another acceleration is evaluated, which is then used in the corrector. Note that this procedure is different from startup, where convergence of acceleration is checked. This cycle repeats until the position and velocity converge. This process gives a P(EC)^n implementation. In some implementations a maximum value of n may be specified.

Single Integration

The first step in startup of single integration is to use the mid-corrector formulas to correct the eight estimated velocities around epoch. In difference form, these formulas are (2.70); using the ordinate coefficients b, the formulas are

$$\dot{\boldsymbol{r}}_n = h\left(\nabla^{-1}\ddot{\boldsymbol{r}}_n - \frac{\ddot{\boldsymbol{r}}_n}{2} + \sum_{k=-4}^{4} b_{nk}\ddot{\boldsymbol{r}}_k\right), \tag{2.83}$$

with $-4 \le n \le 4$. Note the term $-\frac{\ddot{\boldsymbol{r}}_n}{2}$ is excluded from the coefficient sum in anticipation of its inclusion in the acceleration term, as in (2.81). The integration constant C_1 (see Section 2.3.1) may be determined by making sure that the initial conditions ($n = 0$)

$$\dot{\boldsymbol{r}}_0 = h\left(\nabla^{-1}\ddot{\boldsymbol{r}}_0 - \frac{\ddot{\boldsymbol{r}}_0}{2} + \sum_{k=-4}^{4} b_{0k}\ddot{\boldsymbol{r}}_k\right) \tag{2.84}$$

are satisfied; solving for $C_1 = \nabla^{-1}\ddot{\boldsymbol{r}}_0$,

$$C_1 = \frac{\dot{\boldsymbol{r}}_0}{h} - \sum_{k=-4}^{4} b_{0k}\ddot{\boldsymbol{r}}_k + \frac{\ddot{\boldsymbol{r}}_0}{2}. \tag{2.85}$$

As discussed in the previous section, the acceleration term may be included with the sum to make software implementation easier. Define the term $\boldsymbol{s}_n = \nabla^{-1}\ddot{\boldsymbol{r}}_n - \frac{1}{2}\ddot{\boldsymbol{r}}_n$, which replaces the first two terms in (2.83), and define a new integration constant $C_1' = C_1 - \frac{\ddot{\boldsymbol{r}}_0}{2}$. Then (2.83) may be rewritten

$$\dot{\boldsymbol{r}}_n = h\left(\boldsymbol{s}_n + \sum_{k=-4}^{4} b_{nk}\ddot{\boldsymbol{r}}_k\right), \tag{2.86}$$

where n ranges from -4 to 4, with

$$\boldsymbol{s}_n = \begin{cases} \boldsymbol{s}_{n-1} + \dfrac{\ddot{\boldsymbol{r}}_{n-1} + \ddot{\boldsymbol{r}}_n}{2} & \text{if } n > 0, \\ C_1' & \text{if } n = 0, \\ \boldsymbol{s}_{n+1} - \dfrac{\ddot{\boldsymbol{r}}_{n+1} + \ddot{\boldsymbol{r}}_n}{2} & \text{if } n < 0. \end{cases} \tag{2.87}$$

Table 2.7 shows the relationship between the summed accelerations and $\boldsymbol{s}_n$ near epoch. For a given value of n, notice the symmetry of the formulas in the last column; changing the sign of n merely changes the sign of the term added to C_1'. This formulation makes programming simpler and is the reason that the term $-\ddot{\boldsymbol{r}}_0/2$ is moved from the coefficients

Table 2.7: Relationship Between Summation Term and $\boldsymbol{s}_n$

n	$\ddot{\boldsymbol{r}}_n$	$\nabla^{-1}\ddot{\boldsymbol{r}}_n$	$\boldsymbol{s}_n = \nabla^{-1}\ddot{\boldsymbol{r}}_n - \frac{1}{2}\ddot{\boldsymbol{r}}_n$
-2	$\ddot{\boldsymbol{r}}_{-2}$	$C_1 - \ddot{\boldsymbol{r}}_0 - \ddot{\boldsymbol{r}}_{-1}$	$C_1' - \frac{1}{2}\ddot{\boldsymbol{r}}_0 - \ddot{\boldsymbol{r}}_{-1} - \frac{1}{2}\ddot{\boldsymbol{r}}_{-2}$
-1	$\ddot{\boldsymbol{r}}_{-1}$	$C_1 - \ddot{\boldsymbol{r}}_0$	$C_1' - \frac{1}{2}\ddot{\boldsymbol{r}}_0 - \frac{1}{2}\ddot{\boldsymbol{r}}_{-1}$
0	$\ddot{\boldsymbol{r}}_0$	C_1	C_1'
1	$\ddot{\boldsymbol{r}}_1$	$C_1 + \ddot{\boldsymbol{r}}_1$	$C_1' + \frac{1}{2}\ddot{\boldsymbol{r}}_0 + \frac{1}{2}\ddot{\boldsymbol{r}}_1$
2	$\ddot{\boldsymbol{r}}_2$	$C_1 + \ddot{\boldsymbol{r}}_1 + \ddot{\boldsymbol{r}}_2$	$C_1' + \frac{1}{2}\ddot{\boldsymbol{r}}_0 + \ddot{\boldsymbol{r}}_1 + \frac{1}{2}\ddot{\boldsymbol{r}}_2$

to the sum. The $\boldsymbol{s}_n$ are computed successively in order to compute the startup velocities. While the formulas are correct for $n = 0$, no correction is performed on epoch.

Once the startup points have been corrected, integration may proceed forward with application of the predictor, followed by one or more applications of the corrector. The predictor finds the velocity $\dot{\boldsymbol{r}}_{n+1}$ from the coefficients in row 5 of the b array, and the nine most recent accelerations $\ddot{\boldsymbol{r}}_{n-8} \ldots \ddot{\boldsymbol{r}}_n$. The formula also includes a ∇^{-1} term which is the sum of all the accelerations since epoch and C_1,

$$\dot{\boldsymbol{r}}_{n+1} = h\left(\nabla^{-1}\ddot{\boldsymbol{r}}_n + \sum_{k=-4}^{4} b_{5k}\ddot{\boldsymbol{r}}_{n+k-4}\right), \tag{2.88}$$

with $n \geq 4$. This expression differs from the difference form given in (2.72). For programming, the predictor can be written using $\boldsymbol{s}_n$,

$$\dot{\boldsymbol{r}}_{n+1} = h\left(\boldsymbol{s}_n + \frac{\ddot{\boldsymbol{r}}_n}{2} + \sum_{k=-4}^{4} b_{5k}\ddot{\boldsymbol{r}}_{n+k-4}\right). \tag{2.89}$$

The Adams-Moulton corrector formula in difference form (2.81), for any point $n > 4$ after the startup, can be rewritten using the eighth-order ordinate coefficients b. These coefficients are applied to the eight previous accelerations and the current predicted acceleration,

$$\dot{\boldsymbol{r}}_n = h\left(\nabla^{-1}\ddot{\boldsymbol{r}}_n - \frac{\ddot{\boldsymbol{r}}_n}{2} + \sum_{k=-4}^{4} b_{4k}\ddot{\boldsymbol{r}}_{n+k-4}\right), \tag{2.90}$$

with $n \geq 4$. Note that (2.90) is the same as (2.83) for $n = 4$. The corrector can be written using $\boldsymbol{s}_n$,

$$\dot{\boldsymbol{r}}_n = h\left(\boldsymbol{s}_n + \sum_{k=-4}^{4} b_{4k}\ddot{\boldsymbol{r}}_{n+k-4}\right). \tag{2.91}$$

Depending on the implementation, this equation may be applied through several evaluate and correct (EC) iterations, but only the last acceleration, $k = 4$, changes, so the summation for the first eight need only be done once, then saved through the iterations.

Second Integration

The mid-correctors, predictor, and corrector for the second integration ordinate form are respectively

$$\boldsymbol{r}_n = h^2 \left(\nabla^{-2} \ddot{\boldsymbol{r}}_{n-1} + \sum_{k=-4}^{4} a_{nk} \ddot{\boldsymbol{r}}_k \right) \qquad -4 \le n \le 4\,, \tag{2.92}$$

$$\boldsymbol{r}_{n+1} = h^2 \left(\nabla^{-2} \ddot{\boldsymbol{r}}_n + \sum_{k=-4}^{4} a_{5k} \ddot{\boldsymbol{r}}_{n+k-4} \right) \qquad n \ge 4\,, \tag{2.93}$$

$$\boldsymbol{r}_n = h^2 \left(\nabla^{-2} \ddot{\boldsymbol{r}}_{n-1} + \sum_{k=-4}^{4} a_{4k} \ddot{\boldsymbol{r}}_{n+k-4} \right) \qquad n \ge 4\,, \tag{2.94}$$

corresponding to the difference form (2.75), (2.74), (2.73). To find the integration constant for second integration, C_2, use $n = 0$ in (2.92),

$$\boldsymbol{r}_0 = h^2 \left(\nabla^{-2} \ddot{\boldsymbol{r}}_{-1} + \sum_{k=-4}^{4} a_{0k} \ddot{\boldsymbol{r}}_k \right). \tag{2.95}$$

Table 2.2 shows that the second sum on the point immediately before epoch is the difference of the two constants of integration,

$$\nabla^{-2} \ddot{\boldsymbol{r}}_{-1} = C_2 - C_1, \tag{2.96}$$

which then allows us to solve for C_2, because C_1 is known (2.85),

$$C_2 = \frac{\boldsymbol{r}_0}{h^2} - \sum_{k=-4}^{4} a_{0k} \ddot{\boldsymbol{r}}_k + C_1. \tag{2.97}$$

For software implementation, a term $\boldsymbol{S}_n = \nabla^{-2} \ddot{\boldsymbol{r}}_{n-1}$ is defined recursively

$$\boldsymbol{S}_n = \begin{cases} \boldsymbol{S}_{n-1} + \boldsymbol{s}_{n-1} + \dfrac{\ddot{\boldsymbol{r}}_{n-1}}{2} & \text{if } n > 0, \\ C_2 - C_1 & \text{if } n = 0, \\ \boldsymbol{S}_{n+1} - \boldsymbol{s}_{n+1} + \dfrac{\ddot{\boldsymbol{r}}_{n+1}}{2} & \text{if } n < 0. \end{cases} \tag{2.98}$$

The mid-correctors, predictor, and corrector can now be written in terms of $\boldsymbol{S}_n$,

$$\boldsymbol{r}_n = h^2 \left(\boldsymbol{S}_n + \sum_{k=-4}^{4} a_{nk} \ddot{\boldsymbol{r}}_k, \right) \qquad -4 \leq n \leq 4, \tag{2.99}$$

$$\boldsymbol{r}_{n+1} = h^2 \left(\boldsymbol{S}_{n+1} + \sum_{k=-4}^{4} a_{5k} \ddot{\boldsymbol{r}}_{n+k-4} \right) \qquad n \geq 4, \tag{2.100}$$

$$\boldsymbol{r}_n = h^2 \left(\boldsymbol{S}_n + \sum_{k=-4}^{4} a_{4k} \ddot{\boldsymbol{r}}_{n+k-4} \right) \qquad n \geq 4. \tag{2.101}$$

As with first integration, the summation in the corrector may be split, because the first eight accelerations do not change during the EC iteration.

Procedure

A step by step procedure for the operation of the integrator can now be written:

Startup

1. Use f and g series to calculate 8 positions and velocities surrounding epoch.
2. Evaluate 9 accelerations from these positions and velocities, and those of epoch.
3. While the accelerations have not converged:
   (a) Calculate s_0 (2.87) and S_0 (2.98).
   (b) For each point $n = -4 \ldots 4, n \neq 0$:
      i. Calculate s_n (2.87) and S_n (2.98).
      ii. Calculate $\sum_{k=-4}^{4} b_{nk} \ddot{\boldsymbol{r}}_k$.
      iii. Calculate $\sum_{k=-4}^{4} a_{nk} \ddot{\boldsymbol{r}}_k$.
      iv. Calculate $\dot{\boldsymbol{r}}_n$ (2.86) and $\boldsymbol{r}_n$ (2.99).
      v. Evaluate the acceleration $\ddot{\boldsymbol{r}}_n$ using the appropriate force model.
   (c) Test convergence of the accelerations.

Predict

4. Calculate S_{n+1} (2.98).
5. Calculate $\sum_{k=-4}^{4} b_{5k}\ddot{\boldsymbol{r}}_{n+k-4}$.
6. Calculate $\sum_{k=-4}^{4} a_{5k}\ddot{\boldsymbol{r}}_{n+k-4}$.
7. Calculate $\dot{\boldsymbol{r}}_{n+1}$ (2.89) and $\boldsymbol{r}_{n+1}$ (2.100).

Evaluate — Correct

8. Evaluate the acceleration $\ddot{\boldsymbol{r}}_{n+1}$.
9. Increment n.
10. While $\dot{\boldsymbol{r}}_n$ and $\boldsymbol{r}_n$ have not converged, and the maximum number of corrector iterations is not exceeded:
    (a) Calculate s_n (2.87).
    (b) If first iteration:
        i. Calculate $\sum_{k=-4}^{3} b_{4k}\ddot{\boldsymbol{r}}_{n+k-4}$.
        ii. Calculate $\sum_{k=-4}^{3} a_{4k}\ddot{\boldsymbol{r}}_{n+k-4}$.
    (c) Calculate $b_{44}\ddot{\boldsymbol{r}}_n$ and $a_{44}\ddot{\boldsymbol{r}}_n$.
    (d) Calculate $\dot{\boldsymbol{r}}_n$ (2.91) and $\boldsymbol{r}_n$ (2.101).
    (e) Test convergence of $\dot{\boldsymbol{r}}_n$ and $\boldsymbol{r}_n$; if not converged, and this is not the last allowable corrector iteration, evaluate $\ddot{\boldsymbol{r}}_n$.
11. Go to 4.

Similar procedures are used for implementing two Adams, two summed Adams, or Adams with Störmer-Cowell.

2.4 s-integration

For elliptical orbits, fixed-step integrators are less efficient than variable-step methods, because many steps have to be taken at apogee in order to get the desired accuracy at perigee. One way of increasing the efficiency of numerically integrating elliptical orbits is to change the independent variable using a generalized Sundman transformation [13]. Integrating with the new independent variable is known as s-integration, while conventional integration with time as the independent variable is known as t-integration. Use of s-integration can be considered an analytic step regulation, because the step size in time is changed by an analytic transformation.

2.4.1 Transformation Equation

Sundman [14] and Levi-Civita [15], in attempting to solve the restricted problem of three bodies, introduced the transformation of the independent variable

$$dt = cr\,ds, \tag{2.102}$$

with c constant for the two-body orbit, because this transformation regularizes, and in fact linearizes, the equations of motion. Later investigators raised r to different powers in the transformation,

$$dt = cr^n\,ds, \tag{2.103}$$

known as the *generalized Sundman transformation* (Szebehely and Bond [16] generalized even more, allowing an arbitrary function of r).

In some cases the new independent variable s can be written in terms of an *orbit angle*. An orbit angle is any angle θ considered a function of true anomaly $\theta(\nu)$ that has the following properties:

- At perigee, the value of the angle is the same as true anomaly: $\theta(\nu) = \nu = 2\pi m$ for any integer m.
- At apogee, the value of the angle is the same as true anomaly: $\theta(\nu) = \nu = \pi m$ for any odd integer m.
- The angle increases monotonically with true anomaly, $\theta(\nu_2) > \theta(\nu_1)$ if $\nu_2 > \nu_1$.
- There is symmetry about the major axis: $\theta(\nu) = -\theta(-\nu)$.

Examples include the mean, eccentric, and true anomalies. Simply applying the transformation to s does not assure that s is then an orbit angle; c must be picked so that the appropriate boundary conditions are satisfied.

For each of the possible values of $n \geq 1$, there is a corresponding angle ([9], p. 100; [17]; [18], p. 19; [2], p. 484):

- $n = 1$ or $dt = cr\,ds$. The angle s is the eccentric anomaly if c is chosen so that s is an orbit angle, $c = \sqrt{a/\mu}$. This case is Sundman's original transformation, or the Kustaanheimo-Stiefel transformation [18].
- $n = 2$ or $dt = cr^2 ds$. The angle s is the true anomaly if c is chosen so that s is an orbit angle, $c = [\mu a(1 - e^2)]^{-1/2}$.
- $n = 3/2$ or $dt = cr^{3/2} ds$. In this case s is known as the intermediate anomaly (see below) when $c = 1/\sqrt{\mu}$ [17], though s does not have any meaning as an angle in the orbit. A value of c so that s is an orbit angle is found in [13].

Ferrer and Sein-Echaluce [19] showed that only $n = 1$ and $n = 2$ linearize the Kepler problem; the latter only with regularization. Palmer et al. [20] studied the use of an $n = 1$ transformation for integration with a Bulirsch-Stoer integrator and compared results using positions obtained from a GPS-equipped satellite. Merson [21] introduced the idea of using the value $n = 3/2$ in the generalized Sundman transformation, with an intent not of regularization *per se*, but of maximizing computational efficiency; see also [22]. He gave an analysis showing that this value of n equally distributes the integration error around a full orbit, even if the eccentricity is high. One may conclude that this method is preferred for numerical integration. He also gave accuracy and timing results for $n = 3/2$, compared to other integrators. Nacozy [17], expressed s with $n = 3/2$ in terms of an elliptic integral of the true anomaly, and dubbed this angle the *intermediate anomaly*. His choice of constant $c = 1/\sqrt{\mu}$ made s dimensionless but did not result in an orbit angle.

There are two ways to approach a numerical integration implementation using these transformations. The step size at perigee can be fixed through the transformation; in this case, the total number of steps varies depending on the transformation. Expressing the transformations as orbit angles, Figure 2.2 illustrates this approach with an approximately fixed perigee step of $\Delta\nu \approx 1.0$ radian. Alternatively, one can fix the total number of steps on an orbit, and allow the step sizes to vary. Figure 2.3 illustrates this approach with sixteen points equally distributed in the same four orbit angles. The figures show an orbit with 0.75 eccentricity. Since perturbations affect the orbit, they also affect the

transformations. However, the steps in these figures assume a Kepler problem in order to understand the general characteristics of s-integration.

Figure 2.2(a) demonstrates the inefficiency of t-integration for elliptical orbits. In order to get the desired step at perigee, there are many integration points close together at apogee. These steps are closer than needed to maintain the desired accuracy. Integrating with the intermediate anomaly, shown in Figure 2.2(c), gives the same step size at perigee with only 10 steps over the orbit, in contrast to 58 steps with t-integration. Figure 2.3 shows how the integration points are spread throughout the orbit using the different orbit angles. With t-integration, Figure 2.3(a), most of the steps are at apogee, where they are not needed. With eccentric anomaly, Figure 2.3(b), the points seem to be spread evenly about the orbit. With the intermediate anomaly, Figure 2.3(c), there are more steps toward perigee. Because the perturbations due to atmospheric drag and geopotential are greater at perigee, integrating with the intermediate anomaly has an advantage over the eccentric anomaly. Figure 2.3(d) shows that integrating with the true anomaly puts more steps towards perigee than integrating with the intermediate anomaly. The steps at apogee may be too far apart with the true anomaly to maintain accuracy.

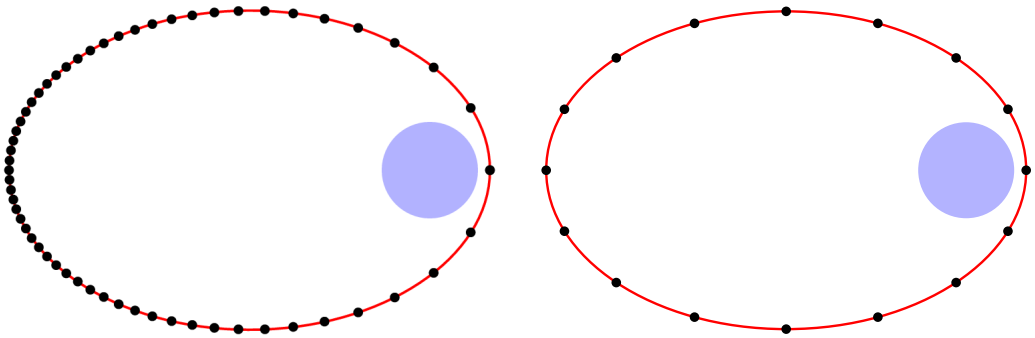
2.4.2 Implementation of the Transformation

One of the integration methods tested in Chapter 5 is s-integration. The s-integration is the Merson / Nacozy form discussed in the previous section with $n = 3/2$ and $c = 1/\sqrt{\mu}$, so s is not an orbit angle. The s-integration is performed using the eighth-order Gauss-Jackson integrator. The implementation of t-integration with Gauss-Jackson is described in Section 2.3.9. Integration in s uses the same code, but adds the necessary transformation from t to s space; the transformation back into to time requires the integration of an additional, seventh, differential equation (2.103).

A step size in time is first selected. For s-integration the step size must be converted into s-space, by using the derivative (2.103) as an approximation of the discrete step. The conversion is performed using the distance from earth center to the satellite at perigee r_p,

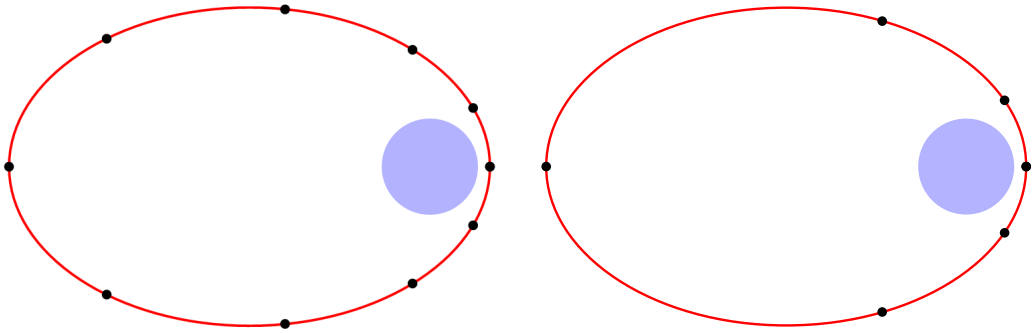
$$\Delta s = \sqrt{\mu}\, r_p^{-\frac{3}{2}}\, \Delta t. \tag{2.104}$$

Since the distance at perigee is used in the conversion, the step at perigee in true anomaly is approximately the same for s-integration (Figure 2.2(c)) as it is in t-integration (Figure 2.2(a)). In the tests in Chapter 5 step sizes for s-integration are referred to in units of time; these are actually the step sizes at perigee which must be converted to s-steps with (2.104).



(a) Equal Mean Anomaly (n=0) with 58 Steps

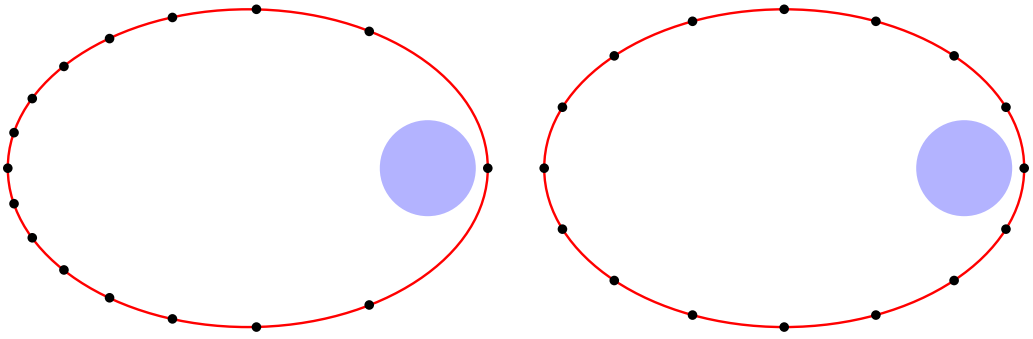
(b) Equal Eccentric Anomaly (n=1) with 16 Steps



(c) Equal Intermediate Anomaly (n=3/2) with 10 Steps

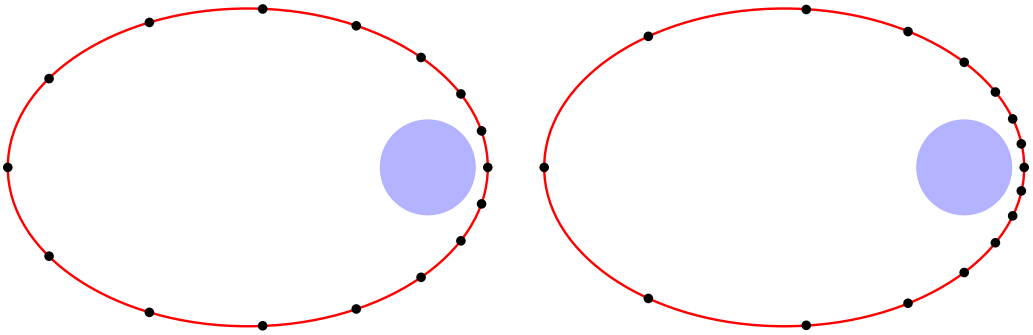
(d) Equal True Anomaly (n=2) with 6 Steps

Figure 2.2: Points Separated by Equal Values of Various Orbit Angles, with Equal Steps at Perigee



(a) Equal Mean Anomaly (n=0); Step Size at Perigee Is $\Delta\nu = 1.97$ Radians

(b) Equal Eccentric Anomaly (n=1); Step Size at Perigee Is $\Delta\nu = 0.97$ Radians



(c) Equal Intermediate Anomaly (n=3/2); Step Size at Perigee Is $\Delta\nu = 0.60$ Radians

(d) Equal True Anomaly (n=2); Step Size at Perigee Is $\Delta\nu = 0.39$ Radians

Figure 2.3: Points Separated by Equal Values of Various Orbit Angles, with 16 Steps in each Orbit

After the step size is determined the integrator enters its initialization phase, which for t-integration is described in Section 2.3.9. Because the f and g series equations used in the initialization depend on the time between the points, and for s-integration the points must be equally spaced in s, a conversion must be made from s to time. To be exact, the time should be found by integrating (2.103); however, accuracy is not as critical in this first phase of the initialization routine, so the time is approximated by holding the epoch distance constant,

$$\Delta t_{fg} = \frac{r_0^{\frac{3}{2}}}{\sqrt{\mu}} \Delta s. \tag{2.105}$$

After the positions and velocities are calculated by the f and g series, the force model is evaluated to find the accelerations. These accelerations must then be converted to s-space by changing t derivatives to s derivatives.

For any value of n and c, the derivatives with respect to s are computed using the derivatives with respect to t via the relation

$$\frac{d}{dt} = c^{-1} r^{-n} \frac{d}{ds}. \tag{2.106}$$

Differentiation with respect to s is indicated with a prime ($'$), while differentiation with respect to t is indicated by a dot ($\dot{}$). The velocity is converted by application of (2.106),

$$\dot{\boldsymbol{r}} = c^{-1} r^{-n} \boldsymbol{r}'. \tag{2.107}$$

The acceleration can then also be transformed,

$$\ddot{\boldsymbol{r}} = -c^{-3} c' r^{-2n} \boldsymbol{r}' - n c^{-2} r^{-2n-1} r' \boldsymbol{r}' + c^{-2} r^{-2n} \boldsymbol{r}'', \tag{2.108}$$

where $c' = dc/ds$ may be non-zero if perturbations are present (say, if c depends on a or e). In the present case $c = 1/\sqrt{\mu}$, and $n = 3/2$, so (2.108) can be solved for $\boldsymbol{r}''$,

$$\boldsymbol{r}'' = \frac{1}{\mu} \left(\frac{3}{2} r^2 \dot{r} \dot{\boldsymbol{r}} + r^3 \ddot{\boldsymbol{r}} \right). \tag{2.109}$$

This equation involves the derivative of the (scalar) magnitude r which is easily calculated,

$$\dot{r} = \frac{d\sqrt{\boldsymbol{r} \cdot \boldsymbol{r}}}{dt} = \frac{\boldsymbol{r} \cdot \dot{\boldsymbol{r}}}{r}, \tag{2.110}$$

so that the second derivative may be rewritten,

$$\boldsymbol{r}'' = \frac{1}{\mu} \left(\frac{3}{2} r (\boldsymbol{r} \cdot \dot{\boldsymbol{r}}) \dot{\boldsymbol{r}} + r^3 \ddot{\boldsymbol{r}} \right). \tag{2.111}$$

These second derivatives are then integrated using the Gauss-Jackson and summed Adams mid-corrector formulas to find the position and velocity at each of the 8 points surrounding epoch. The integration does not give velocity directly; it gives $r' = dr/ds$. So a conversion must be made to find velocity,

$$\dot{\boldsymbol{r}} = \frac{\sqrt{\mu} r'}{r^{\frac{3}{2}}}. \tag{2.112}$$

Before the force model can be re-evaluated, the time at each point must be found. The time is found by integrating the transformation (2.103),

$$t' = \frac{1}{\sqrt{\mu}} r^{\frac{3}{2}}, \tag{2.113}$$

using the summed Adams mid-corrector formulas. With the time known the forces are evaluated to compute refined estimates of the accelerations, these accelerations are converted into s-space second derivatives, and the integration is performed again to obtain positions, velocities, and times at the points. This process repeats until the accelerations between two iterations converges to a prescribed tolerance. The initialization procedure for s integration may thus be summarized:

Initialization for s-integration

1. Convert t step size to s step using the perigee distance, (2.104).
2. Use f and g series to calculate 8 positions and velocities surrounding epoch, holding the epoch distance constant to find the time, (2.105).
3. Evaluate 9 accelerations from the positions and velocities, including epoch.
4. Convert the accelerations into s derivatives, (2.111).
5. While the s second derivatives have not converged:
   (a) For each point $n = -4 \ldots 4, n \neq 0$:
      i. Calculate $\boldsymbol{r}_n$ and $\boldsymbol{r}'_n$, using Gauss-Jackson and summed Adams mid-corrector formulas (2.86), (2.99).
      ii. Convert $\boldsymbol{r}'_n$ to $\dot{\boldsymbol{r}}_n$, (2.112).
      iii. Calculate the time at point n by integrating (2.113) with the summed Adams mid-corrector formulas.
      iv. Evaluate the acceleration $\ddot{\boldsymbol{r}}_n$ using the appropriate force model.
      v. Convert $\ddot{\boldsymbol{r}}_n$ to $\boldsymbol{r}''_n$, (2.111).

(b) Test convergence of $\boldsymbol{r}''_n$.

The initialization process is followed by a predictor-corrector cycle, which for t-integration is described in Section 2.3.9. This cycle is modified for s-integration. When the independent variable is s, the predictor and corrector give dr/ds, not velocity, so the velocity must be found using (2.112). After the position and velocity are found the time at the new point must be found by solving (2.113) using the summed Adams predictor or corrector formula. After the forces are evaluated, the accelerations must be converted to second s derivatives using (2.111). Thus the predictor-corrector modified for s-integration continues from steps 1-5 above as follows:

Predict (s-integration)

6. Calculate $\boldsymbol{r}_{n+1}$ and $\boldsymbol{r}'_{n+1}$, using Gauss-Jackson and summed Adams predictor formulas,(2.100),(2.89).
7. Convert $\boldsymbol{r}'_{n+1}$ to $\dot{\boldsymbol{r}}_{n+1}$, (2.112).
8. Calculate the time at point $n+1$ by integrating (2.113) with the summed Adams predictor formula.

Evaluate — Correct (s-integration)

10. Evaluate the acceleration $\ddot{\boldsymbol{r}}_{n+1}$.
11. Convert $\ddot{\boldsymbol{r}}_{n+1}$ to $\boldsymbol{r}''_{n+1}$, (2.111).
12. Increment n.
13. While $\boldsymbol{r}_n$ and $\dot{\boldsymbol{r}}_n$ have not converged, and the maximum number of corrector iterations has not been reached:
    (a) Calculate $\boldsymbol{r}_n$ and $\boldsymbol{r}'_n$, using Gauss-Jackson and summed Adams corrector formulas,(2.101),(2.91).
    (b) Convert $\boldsymbol{r}'_n$ to $\dot{\boldsymbol{r}}_n$, (2.112).
    (c) Calculate the time at point n by integrating (2.113) with the summed Adams corrector formula.
    (d) Test convergence of $\boldsymbol{r}_n$ and $\dot{\boldsymbol{r}}_n$; if not converged, and this is not the last allowable corrector iteration, evaluate $\ddot{\boldsymbol{r}}_n$, and convert to $\boldsymbol{r}''_n$.
14. Predict next time step (go to 6).

One disadvantage of s-integration is that the integration of (2.113) subjects time to integration error, which can significantly contribute to in-track error. Another disadvantage of s-integration is that there is still no control over the local error. Though s-integration provides analytic step regulation by varying the steps through the orbit as shown in Figure 2.2, it is still a fixed-step method. The steps are equally spaced by s, instead of equally spaced by time, t. Variable-step methods, described in the next section, regulate the step size by controlling the local error.

2.5 Variable-Step Integration

2.5.1 Introduction

True variable-step integration methods change the step size at each step to meet some tolerance for the local error. The local error is estimated at each step, and the step size is adjusted so that the estimated error at the next step will be approximately equal to the tolerance. The local error estimate is made by comparing two integrations made of different orders. For instance, the Runge-Kutta-Fehlberg method ([6], pp. 286-292) estimates the local error by comparing the result of fourth-order and fifth-order Runge-Kutta formulas. For multi-step integrators, local error estimates can be made by using predictor and corrector formulas of different order.

The multi-step methods derived above in terms of the backward difference operator do not lend themselves to variable-step integration. Because the backward differences require a set of backpoints that are equally spaced by the independent variable, a new set of backpoints is required if the step size is changed. Creating this new set of backpoints effectively means the integration must be restarted, which reduces the effectiveness of the variable-step method. For an eighth-order integrator, the startup procedure described in Section 2.3.7 requires eight evaluations for each iteration of the startup cycle. If, say, there are typically three iterations of the startup cycle, then a restart costs 24 evaluations. Since an increase of step size when moving towards apogee means there also has to be a decrease on the other side, increasing the step size costs 48 evaluations total. So if the step size is doubled, 48 steps must be taken with the new step size just to break even.

Krogh [23] presented several algorithms for changing the step size that do not involve a full restart. Some of these methods involve interpolating to find solution values at the new set of backpoints and then performing evaluations only once, or simply interpolating the function values to the new set of backpoints so no new evaluations are required. Gear presented an integrator using interpolation to find the new set of function values

[5]. Though interpolation does save function evaluations, error is introduced through the interpolation [23].

Another method suggested by Krogh is to use integration formulas based on interpolating polynomials in divided difference form so that the backpoints are not required to be equally spaced. Krogh indicated that this method gives the most flexibility, however it has the disadvantage that the coefficients must be calculated at every step. Krogh pointed out that the disadvantage of computing coefficients is minor if the calculation of the function f is much more costly than the calculation of the coefficients. In our case the cost of computing the coefficients is nearly insignificant compared to the cost of the function evaluation, so this method is likely the best for implementing a variable-step algorithm.

Krogh presented algorithms for integrating differential equations using divided differences in [24]. His method applies to differential equations of any order, so single-integration as well as double-integration and higher can be performed. However the algorithms for solving differential equations of order greater than one involve terms containing the derivatives. For instance, the double-integration formula involves a velocity term. The integrator presented in Chapter 4 does not have such a velocity term. Shampine and Gordon [25] also used divided differences to develop a single-integration variable-step method. Their work followed the work of Krogh and also included discussion of how to change the step size to control local error. The derivation of the Shampine-Gordon method is described in the next section, and is the starting point of the derivation of the variable-step Störmer-Cowell method presented in Chapter 4.

2.5.2 Shampine-Gordon

The Shampine-Gordon integrator follows a predict, evaluate, correct, evaluate (PECE) implementation, so there are two evaluations at every step. The corrector is one order higher than the predictor, which allows for a local error estimate at each step. The step size and order are adjusted at every step based on the local error estimate. The derivation of Shampine-Gordon was originally presented in [25], and follows the work of Krogh [24]. The derivation is repeated here because it is the starting point of the derivation of the variable-step Störmer-Cowell integrator. The derivation is based on the concept of divided differences.

Divided Differences

The Shampine-Gordon integrator is derived by integrating a polynomial that interpolates through the function values at the backpoints. This polynomial is written in divided difference form so that the backpoints do not have to be equally spaced.

The $(k-1)^{\text{th}}$ degree interpolating polynomial $\boldsymbol{P}_{k,n}(t)$ passes through the k function values $\boldsymbol{f}_{n-k+1} \cdots \boldsymbol{f}_n$ if

$$\boldsymbol{P}_{k,n}(t_{n+1-i}) = \boldsymbol{f}_{n+1-i}, \quad i = 1 \ldots k. \tag{2.114}$$

Here $\boldsymbol{f}_i = \boldsymbol{f}(t_i, \ddot{\boldsymbol{r}}_i, \dot{\boldsymbol{r}}_i)$, where $\boldsymbol{f}$ is the right-hand side function in the differential equation for acceleration

$$\ddot{\boldsymbol{r}} = \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}}). \tag{2.115}$$

In divided difference form, the polynomial is written

$$\begin{aligned} \boldsymbol{P}_{k,n}(t) = \boldsymbol{f}[t_n] + (t - t_n)\boldsymbol{f}[t_n, t_{n-1}] + (t - t_n)(t - t_{n-1})\boldsymbol{f}[t_n, t_{n-1}, t_{n-2}] + \cdots \\ + (t - t_n)(t - t_{n-1}) \cdots (t - t_{n-k+2})\boldsymbol{f}[t_n, t_{n-1}, \ldots, t_{n-k+1}], \end{aligned} \tag{2.116}$$

where $\boldsymbol{f}[t_n, \ldots, t_{n-i+1}]$ is the i^{th} divided difference of $\boldsymbol{f}_n$. The divided differences are calculated through a recursive relation,

$$\begin{aligned} \boldsymbol{f}[t_n] &= \boldsymbol{f}_n, \\ \boldsymbol{f}[t_n, t_{n-1}] &= \frac{\boldsymbol{f}_n - \boldsymbol{f}_{n-1}}{t_n - t_{n-1}}, \\ \boldsymbol{f}[t_n, \ldots, t_{n-i}] &= \frac{\boldsymbol{f}[t_n, \ldots, t_{n-i+1}] - \boldsymbol{f}[t_{n-1}, \ldots, t_{n-i}]}{t_n - t_{n-i}}. \end{aligned} \tag{2.117}$$

As an example, consider the divided difference table in Table 2.8. Each divided difference is calculated by subtracting the two values to the left and dividing by the total span in t that the difference covers.

For the example in Table 2.8, the polynomial $\boldsymbol{P}_{4,4}$ that passes through the values is found from (2.116),

$$\boldsymbol{P}_{4,4} = 7 + (t-7)(-2/3) + (t-7)(t-4)(-14/12) + (t-7)(t-4)(t-3)(-11/36). \tag{2.118}$$

The example can also be used to illustrate how a new point can be added to an existing polynomial, which is used in the derivation of the corrector. The derivation of the corrector involves a polynomial that passes through the predicted value as well as all the same values as the polynomial used by the predictor. When $n = 3$, the polynomial $\boldsymbol{P}_{3,3}$ is known,

$$\boldsymbol{P}_{3,3} = 9 + (t - 4)(4) + (t - 4)(t - 3)(2/3). \tag{2.119}$$

Table 2.8: Example Divided Difference Table

n	t_n	$\boldsymbol{f}[t_n]$	$\boldsymbol{f}[t_n, t_{n-1}]$	$\boldsymbol{f}[t_n, t_{n-1}, t_{n-2}]$	$\boldsymbol{f}[t_n, \ldots, t_{n-3}]$
1	1	1			
2	3	5	2		
3	4	9	4	2/3	
4	7	7	−2/3	−14/12	−11/36

The polynomial $\boldsymbol{P}_{4,4}$ can be found from $\boldsymbol{P}_{3,3}$ by adding one more term at $t = 1$ that includes the new difference,

$$\boldsymbol{P}_{4,4} = 9 + (t-4)(4) + (t-4)(t-3)(2/3) + (t-4)(t-3)(t-1)(-11/36). \quad (2.120)$$

The two forms of $\boldsymbol{P}_{4,4}$, (2.118) and (2.120), are equivalent.

Predictor

The Shampine-Gordon predictor finds the predicted value of velocity, $\dot{\boldsymbol{r}}^p_{n+1}$, at point $(n+1)$ from the value at n, $\dot{\boldsymbol{r}}_n$, by integrating the interpolating polynomial $\boldsymbol{P}_{k,n}$ defined in (2.116),

$$\dot{\boldsymbol{r}}^p_{n+1} = \dot{\boldsymbol{r}}_n + \int_{t_n}^{t_{n+1}} \boldsymbol{P}_{k,n}(t)\, dt. \quad (2.121)$$

The superscript p is used to indicate the value is predicted. In order to develop an effective algorithm to integrate $\boldsymbol{P}_{k,n}(t)$, the terms are rewritten. First, the independent variable t is replaced by τ which measures the fraction of the current interval covered,

$$\tau = \frac{t - t_n}{h_{n+1}}, \quad (2.122)$$

with $h_n = t_n - t_{n-1}$ the separation between adjacent values of t. Second, the divided differences are modified so that they are equivalent to backward differences when the

step size is constant. The modified divided differences, $\boldsymbol{\phi}$, are defined

$$\begin{aligned}\boldsymbol{\phi}_1(n) &= \boldsymbol{f}[t_n] = \boldsymbol{f}_n = \ddot{\boldsymbol{r}}_n,\\ \boldsymbol{\phi}_i(n) &= \psi_1(n)\psi_2(n)\cdots\psi_{i-1}(n)\boldsymbol{f}[t_n, t_{n-1}, \ldots, t_{n-i+1}] \qquad i > 1,\end{aligned} \tag{2.123}$$

where $\psi_i(n)$ is the sum of the i steps leading up to point n,

$$\psi_i(n) = h_n + h_{n-1} + \cdots + h_{n+1-i}. \tag{2.124}$$

With a constant step size h, $\psi_i(n) = ih$, and the modified divided differences reduce to backward differences,

$$\boldsymbol{\phi}_i(n) = \nabla^{i-1}\boldsymbol{f}_n = \nabla^{i-1}\ddot{\boldsymbol{r}}_n, \tag{2.125}$$

if $h = h_n = h_{n-1} = \cdots = h_{n+1-i}$.

The first term of the polynomial $\boldsymbol{P}_{k,n}$, (2.116), is simply $\boldsymbol{f}[t_n]$. For $i > 1$, the i^{th} term of $\boldsymbol{P}_{k,n}(t)$,

$$(t - t_n)(t - t_{n-1})\cdots(t - t_{n-i+2})\boldsymbol{f}[t_n, t_{n-1}, \ldots, t_{n-i+1}] \tag{2.126}$$

can be written in terms of τ and $\boldsymbol{\phi}_i(n)$,

$$(\tau h_{n+1})(\tau h_{n+1} + h_n)\cdots(\tau h_{n+1} + h_n + \cdots + h_{n-i+3})\frac{\boldsymbol{\phi}_i(n)}{\psi_1(n)\psi_2(n)\ldots\psi_{i-1}(n)}. \tag{2.127}$$

Next, the term is multiplied and divided by $\psi_1(n+1)$ through $\psi_{i-1}(n+1)$, which allows the differences to be easily computed from one step to the next,

$$\begin{aligned}\left(\frac{\tau h_{n+1}}{\psi_1(n+1)}\right)\left(\frac{\tau h_{n+1} + h_n}{\psi_2(n+1)}\right)\cdots\left(\frac{\tau h_{n+1} + h_n + \cdots + h_{n-i+3}}{\psi_{i-1}(n+1)}\right)&\\ \times \frac{\psi_1(n+1)\psi_2(n+1)\cdots\psi_{i-1}(n+1)}{\psi_i(n)\psi_2(n)\cdots\psi_{i-1}(n)}\boldsymbol{\phi}_i(n).&\end{aligned} \tag{2.128}$$

This expression is simplified by introducing $\boldsymbol{\phi}^*$, defined

$$\boldsymbol{\phi}^*_i(n) = \beta_i(n+1)\boldsymbol{\phi}_i(n), \tag{2.129}$$

where

$$\begin{aligned}\beta_1(n+1) &= 1,\\ \beta_i(n+1) &= \frac{\psi_1(n+1)\psi_2(n+1)\cdots\psi_{i-1}(n+1)}{\psi_1(n)\psi_2(n)\cdots\psi_{i-1}(n)} \qquad i > 1,\end{aligned} \tag{2.130}$$

and by condensing the sums of h in the numerators into ψ,

$$\left(\frac{\tau h_{n+1}}{\psi_1(n+1)}\right)\left(\frac{\tau h_{n+1} + \psi_1(n)}{\psi_2(n+1)}\right)\cdots\left(\frac{\tau h_{n+1} + \psi_{i-2}(n)}{\psi_{i-1}(n+1)}\right)\boldsymbol{\phi}^*_i(n). \tag{2.131}$$

Noting that the $i = 1$ term of $\boldsymbol{P}_{k,n}(t)$ is $\boldsymbol{f}[t_n] = \boldsymbol{\phi}_1(t_n)$, the i^{th} term (2.126) can be written,

$$c_{i,n}(\tau)\boldsymbol{\phi}_i^*(n), \tag{2.132}$$

where $c_{i,n}(\tau)$ is defined

$$c_{i,n}(\tau) = \begin{cases} 1 & i = 1, \\ \left(\dfrac{\tau h_{n+1}}{\psi_1(n+1)}\right) = \tau & i = 2, \\ \left(\dfrac{\tau h_{n+1}}{\psi_1(n+1)}\right)\left(\dfrac{\tau h_{n+1} + \psi_1(n)}{\psi_2(n+1)}\right)\cdots\left(\dfrac{\tau h_{n+1} + \psi_{i-2}(n)}{\psi_{i-1}(n+1)}\right) & i \geq 3. \end{cases} \tag{2.133}$$

The expression for $c_{i,n}(\tau)$ is simplified through a recursion formula,

$$c_{i,n}(\tau) = \left(\frac{\tau h_{n+1} + \psi_{i-2}(n)}{\psi_{i-1}(n+1)}\right) c_{i-1,n}(\tau), \tag{2.134}$$

when $i \geq 3$. This expression is further simplified by defining $\alpha_i(n+1)$,

$$\alpha_i(n+1) = \frac{h_{n+1}}{\psi_i(n+1)}, \tag{2.135}$$

so that $c_{i,n}(\tau)$, (2.133), can be written

$$c_{i,n}(\tau) = \begin{cases} 1 & i = 1, \\ \tau & i = 2, \\ \left(\alpha_{i-1}(n+1)\tau + \dfrac{\psi_{i-2}(n)}{\psi_{i-1}(n+1)}\right) c_{i-1,n}(\tau) & i \geq 3. \end{cases} \tag{2.136}$$

The interpolating polynomial $\boldsymbol{P}_{k,n}(t)$ can now be written as a summation,

$$\boldsymbol{P}_{k,n}(t) = \sum_{i=1}^{k} c_{i,n}(\tau)\boldsymbol{\phi}_i^*(n). \tag{2.137}$$

Returning to the original problem of finding the predicted velocity, the polynomial in (2.121) can be replaced with (2.137),

$$\dot{\boldsymbol{r}}_{n+1}^p = \dot{\boldsymbol{r}}_n + \int_{t_n}^{t_{n+1}} \sum_{i=1}^{k} c_{i,n}(\tau)\boldsymbol{\phi}_i^*(n)\, dt. \tag{2.138}$$

Since the $\boldsymbol{\phi}_i^*(n)$ are constants, they can be pulled out of the integration,

$$\dot{\boldsymbol{r}}_{n+1}^p = \dot{\boldsymbol{r}}_n + \sum_{i=1}^{k} \boldsymbol{\phi}_i^*(n) \int_{t_n}^{t_{n+1}} c_{i,n}(\tau)\, dt. \tag{2.139}$$

The integration variable can be changed to τ by (2.122), noting that $\tau = 0$ when $t = t_n$, $\tau = 1$ when $t = t_{n+1}$, and $dt = h_{n+1}\, d\tau$,

$$\dot{\boldsymbol{r}}^p_{n+1} = \dot{\boldsymbol{r}}_n + h_{n+1} \sum_{i=1}^{k} \boldsymbol{\phi}^*_i(n) \int_0^1 c_{i,n}(\tau)\, d\tau. \tag{2.140}$$

To solve the problem, the integral of $c_{i,n}(\tau)$ is needed. Focusing on $i \geq 3$, the integral of $c_{i,n}$ to an arbitrary point τ can be written using (2.136),

$$\int_0^\tau c_{i,n}(\tau_0)\, d\tau_0 = \int_0^\tau \left(\alpha_{i-1}(n+1)\tau_0 + \frac{\psi_{i-2}(n)}{\psi_{i-1}(n+1)} \right) c_{i-1,n}(\tau_0)\, d\tau_0. \tag{2.141}$$

This integral can be solved using integration by parts,

$$\int u\, dv = uv - \int v\, du, \tag{2.142}$$

where in this case

$$u = \left(\alpha_{i-1}(n+1)\tau_0 + \frac{\psi_{i-2}(n)}{\psi_{i-1}(n+1)} \right), \tag{2.143}$$

and

$$dv = c_{i-1,n}(\tau_0)\, d\tau_0, \tag{2.144}$$

which gives,

$$\begin{aligned} \int_0^\tau c_{i,n}(\tau_0)\, d\tau_0 = &\left(\alpha_{i-1}(n+1)\tau + \frac{\psi_{i-2}(n)}{\psi_{i-1}(n+1)} \right) \int_0^\tau c_{i-1,n}(\tau_0)\, d\tau_0 \\ &- \alpha_{i-1}(n+1) \int_0^{\tau_0} \int_0^{\tau_1} c_{i-1,n}(\tau_0)\, d\tau_0\, d\tau_1. \end{aligned} \tag{2.145}$$

Though a double integral has been introduced, it is on a $c_{i-1,n}$ term. Since the integrals of the $c_{1,n}$ and $c_{2,n}$ terms can be found easily, a recursive formula for the integral of $c_{i,n}$ can be found in terms of multiple integrals of lower i values of c.

The recursive formula is found by first introducing notation for multiple integrals of $c_{i,n}(\tau)$,

$$c^{(-q)}_{i,n}(\tau) = \int_0^\tau \int_0^{\tau_{q-1}} \int_0^{\tau_{q-2}} \cdots \int_0^{\tau_0} c_{i,n}(\tau_0)\, d\tau_0\, d\tau_1 \ldots d\tau_{q-1}. \tag{2.146}$$

Under this notation (2.145) may be written

$$c^{(-1)}_{i,n}(\tau) = \left(\alpha_{i-1}(n+1)\tau + \frac{\psi_{i-2}(n)}{\psi_{i-1}(n+1)} \right) c^{(-1)}_{i-1,n}(\tau) - \alpha_{i-1}(n+1) c^{(-2)}_{i-1,n}(\tau). \tag{2.147}$$

The general case is

$$c_{i,n}^{(-q)}(\tau) = \left(\alpha_{i-1}(n+1)\tau + \frac{\psi_{i-2}(n)}{\psi_{i-1}(n+1)}\right) c_{i-1,n}^{(-q)}(\tau) - q\alpha_{i-1}(n+1)c_{i-1,n}^{(-q-1)}(\tau). \quad (2.148)$$

To find the predicted value, (2.140) indicates that $c_{i,n}^{(-1)}(1)$ is needed. A variable $g_{i,q}$ is introduced to simplify the process of finding these values,

$$g_{i,q} = (q-1)!\, c_{i,n}^{(-q)}(1). \quad (2.149)$$

Substituting in the formula for $c_{i,n}^{(-q)}$ given in (2.147),

$$\begin{aligned} g_{i,q} &= \left(\alpha_{i-1}(n+1) + \frac{\psi_{i-2}(n)}{\psi_{i-1}(n+1)}\right)(q-1)!\, c_{i-1,n}^{(-q)}(1) - \alpha_{i-1}(n+1)q!\, c_{i-1,n}^{(-q-1)}(1) \\ &= \left(\frac{h_{n+1}}{\psi_{i-1}(n+1)} + \frac{\psi_{i-2}(n)}{\psi_{i-1}(n+1)}\right) g_{i-1,q} - \alpha_{i-1}(n+1)g_{i-1,q+1}, \end{aligned} \quad (2.150)$$

where $\alpha_{i-1}(n+1)$ has been replaced with (2.135) on the second line. This formula is further simplified by noting that $h_{n+1} + \psi_{i-2}(n) = \psi_{i-1}(n+1)$,

$$g_{i,q} = g_{i-1,q} - \alpha_{i-1}(n+1)g_{i-1,q+1}. \quad (2.151)$$

This equation holds when $i \geq 3$.

To write a recursive formula for g, the special cases of $i = 1$ and $i = 2$ must be considered. When $i = 1$, $c_{1,n} = 1$, so the integral is

$$g_{1,q} = (q-1)!\, c_{1,n}^{(-q)}(1) = (q-1)! \int_0^1 \int_0^{\tau_{q-1}} \cdots \int_0^{\tau_1} d\tau_0\, d\tau_1 \ldots d\tau_{q-1} = \frac{(q-1)!}{q!} = \frac{1}{q}. \quad (2.152)$$

When $i = 2$, $c_{2,n} = s$, so the integral is

$$g_{2,q} = (q-1)! \int_0^1 \int_0^{\tau_{q-1}} \cdots \int_0^{\tau_1} \tau\, d\tau_0\, d\tau_1 \ldots d\tau_{q-1} = \frac{(q-1)!}{(q+1)!} = \frac{1}{q(q+1)}. \quad (2.153)$$

Now a recursive formula for the coefficients $g_{i,q}$ is available,

$$g_{i,q} = \begin{cases} \dfrac{1}{q} & i = 1, \\ \dfrac{1}{q(q+1)} & i = 2, \\ g_{i-1,q} - \alpha_{i-1}(n+1)g_{i-1,q+1} & i \geq 3, \end{cases} \quad (2.154)$$

and the predictor formula (2.121), (2.140), can be written,

$$\dot{\boldsymbol{r}}^p_{n+1} = \dot{\boldsymbol{r}}_n + h_{n+1} \sum_{i=1}^{k} g_{i,1} \boldsymbol{\phi}^*_i(n). \tag{2.155}$$

This formula is the Shampine-Gordon predictor. The predictor is followed by an evaluation, and then the corrector. For a fixed step, $\alpha_i(n+1) = 1/i$, so the coefficients g may be calculated ([25], p. 83). Those coefficients are in Table 2.9. The first row of coefficients, $g_{i,1}$, in Table 2.9 matches the Adams-Bashforth coefficients γ_{i-1} (2.44).

Table 2.9: Coefficients g_{iq} for Constant Step Size

		1	2	3	4	5
	1	1	1/2	5/12	3/8	251/720
	2	1/2	1/6	1/8	19/180	
q	3	1/3	1/12	7/120		
	4	1/4	1/20			
	5	1/5				

Corrector

After the predicted value $\dot{\boldsymbol{r}}^p_{n+1}$ is found, the function f is evaluated at the predicted point, giving a predicted acceleration value, $\ddot{\boldsymbol{r}}^p_{n+1}$. The corrector then uses an interpolating polynomial that is one degree higher than $\boldsymbol{P}_{k,n}(t)$, interpolating through all the same points as $\boldsymbol{P}_{k,n}(t)$ plus the new point $\ddot{\boldsymbol{r}}^p_{n+1}$. This new polynomial $\boldsymbol{P}^*_{k+1,n}(t)$ can be written in terms of $\boldsymbol{P}_{k,n}(t)$ (see Section 2.5.2),

$$\boldsymbol{P}^*_{k+1,n}(t) = \boldsymbol{P}_{k,n}(t) + c_{k+1,n}(\tau) \boldsymbol{\phi}^p_{k+1}(n+1). \tag{2.156}$$

The new modified divided difference, $\boldsymbol{\phi}^p_{k+1}(n+1)$, is calculated from the previous modified divided differences, $\boldsymbol{\phi}_i(n)$, and the new acceleration $\ddot{\boldsymbol{r}}^p_{n+1}$. From the definition of divided differences, (2.117), and the definition of modified divided differences, (2.123), the relation from $\boldsymbol{\phi}^p_i(n+1)$ to $\boldsymbol{\phi}_i(n)$ can be found,

$$\boldsymbol{\phi}^p_i(n+1) = \boldsymbol{\phi}_{i-1}(n+1) - \frac{\psi_1(n+1)\cdots\psi_{i-1}(n+1)}{\psi_1(n)\cdots\psi_{i-1}(n)} \boldsymbol{\phi}_{i-1}(n), \tag{2.157}$$

where $\boldsymbol{\phi}_1^p(n+1) = \ddot{\boldsymbol{r}}_{n+1}^p$. This relation is simplified by the definition of $\boldsymbol{\phi}_i^*(n)$, (2.129),

$$\begin{aligned}\boldsymbol{\phi}_1^p(n+1) &= \ddot{\boldsymbol{r}}_{n+1}^p,\\ \boldsymbol{\phi}_i^p(n+1) &= \boldsymbol{\phi}_{i-1}^p(n+1) - \boldsymbol{\phi}_{i-1}^*(n+1).\end{aligned} \tag{2.158}$$

This relation motivates why β is introduced into the derivation of the predictor.

The corrected velocity at point $(n+1)$, $\dot{\boldsymbol{r}}_{n+1}$, may be found by integrating $\boldsymbol{P}_{k+1,n}^*(t)$,

$$\dot{\boldsymbol{r}}_{n+1} = \dot{\boldsymbol{r}}_n + \int_{t_n}^{t_{n+1}} \left[\boldsymbol{P}_{k,n}(t) + c_{k+1,n}(\tau)\boldsymbol{\phi}_{k+1}^p(n+1)\right]\, dt. \tag{2.159}$$

Combining the terms already known to comprise $\dot{\boldsymbol{r}}_{n+1}^p$, and changing the integration variable to τ, the expression is

$$\dot{\boldsymbol{r}}_{n+1} = \dot{\boldsymbol{r}}_{n+1}^p + h_{n+1}\int_0^1 c_{k+1,n}(\tau)\boldsymbol{\phi}_{k+1}^p(n+1)\, d\tau. \tag{2.160}$$

The integral term may be replaced with $g_{k+1,1}$,

$$\dot{\boldsymbol{r}}_{n+1} = \dot{\boldsymbol{r}}_{n+1}^p + h_{n+1}g_{k+1,1}\boldsymbol{\phi}_{k+1}^p(n+1), \tag{2.161}$$

which is the Shampine-Gordon corrector formula.

After the corrected value is found, another function evaluation is performed, to find $\ddot{\boldsymbol{r}}_{n+1}$. A new set of differences, $\boldsymbol{\phi}_i(n+1)$ are found from a relation analogous to (2.158),

$$\begin{aligned}\boldsymbol{\phi}_1(n+1) &= \ddot{\boldsymbol{r}}_{n+1},\\ \boldsymbol{\phi}_i(n+1) &= \boldsymbol{\phi}_{i-1}(n+1) - \boldsymbol{\phi}_{i-1}^*(n+1).\end{aligned} \tag{2.162}$$

These differences are used by the predictor in the next step.

Interpolation

In general, integration steps do not correspond with output times requested by the user. Though the step size could be controlled to hit output times exactly, this method would usually involve taking steps that are smaller than necessary, which costs run-time. Instead, an interpolation method is used which does not require any additional evaluations. To find a requested value of velocity $\dot{\boldsymbol{r}}_{\text{out}}$ at time t_{out} between t_n and t_{n+1}, a k^{th} degree interpolating polynomial is used which passes through $k+1$ known points,

$$\boldsymbol{P}_{k+1,n+1}(t_{n+2-i}) = \boldsymbol{f}_{n+2-i}, \qquad i = 1, 2 \ldots k+1. \tag{2.163}$$

In divided difference form, the i^{th} term of $P_{k+1,n+1}$ is

$$(t - t_{n+1})(t - x_n) \cdots (t - t_{n-i+3}) \boldsymbol{f}[t_{n+1}, t_n, \ldots, t_{n-i+2}], \tag{2.164}$$

for $i > 1$. This expression can be simplified by first defining the step size h_I between the output time and the last integration time,

$$h_I = t_{\text{out}} - t_{n+1}, \tag{2.165}$$

and changing variables from t to the fraction of a step τ,

$$\tau = \frac{t - t_{n+1}}{h_I}. \tag{2.166}$$

The i^{th} term (2.164) can now be written

$$\frac{\tau h_I}{\psi_1(n+1)} \frac{\tau h_I + \psi_1(n+1)}{\psi_2(n+1)} \cdots \frac{\tau h_I + \psi_{i-2}(n+1)}{\psi_{i-1}(n+1)} \boldsymbol{\phi}_i(n+1). \tag{2.167}$$

The expression is further simplified by defining $c^I_{i,n+1}(\tau)$,

$$c^I_{i,n+1}(\tau) = \begin{cases} 1 & i = 1, \\ \Gamma_{i-1}(\tau) c^I_{i-1,n+1}(\tau) & i \geq 2, \end{cases} \tag{2.168}$$

where

$$\Gamma_i(\tau) = \begin{cases} \dfrac{\tau h_I}{\psi_1(n_1)} & i = 1, \\ \dfrac{\tau h_I + \psi_{i-1}(n+1)}{\psi_i(n+1)} & i \geq 2. \end{cases} \tag{2.169}$$

Now the term (2.164) is simply $c^I_{i,n+1}(\tau)\boldsymbol{\phi}_i(n+1)$.

The value of the velocity at the requested time is found by integrating the interpolating polynomial,

$$\dot{\boldsymbol{r}}_{\text{out}} = \dot{\boldsymbol{r}}_{n+1} + \int_{t_{n+1}}^{t_{\text{out}}} \boldsymbol{P}_{k+1,n+1}(t)\, dt. \tag{2.170}$$

Using the notation $c^I_{i,n+1}(\tau)$, this expression is

$$\dot{\boldsymbol{r}}_{\text{out}} = \dot{\boldsymbol{r}}_{n+1} + h_I \sum_{i=1}^{k+1} \boldsymbol{\phi}_i(n+1) \int_0^1 c^I_{i,n+1}(\tau)\, d\tau. \tag{2.171}$$

The integration of $c^I_{i,n+1}(\tau)$ is found by integrating by parts,

$$\begin{aligned} \int_0^\tau c^I_{i,n+1}(\tau_0)\, d\tau_0 = {} & \Gamma_{i-1}(\tau) \int_0^\tau c^I_{i-1,n+1}(\tau_0)\, d\tau_0 \\ & - \int_0^\tau \int_0^{\tau_1} \frac{h_I}{\psi_{i-1}(n+1)} c^I_{i-1,n+1}(\tau_0)\, d\tau_0\, \tau_1. \end{aligned} \tag{2.172}$$

In general, the formula for multiple integrals of c^I is

$$c_{i,n+1}^{I(-q)}(\tau) = \Gamma_{i-1}(\tau)c_{i-1,n+1}^{I(-q)}(\tau) - \frac{qh_I}{\psi_{i-1}(n+1)}c_{i-1,n+1}^{I(-q-1)}(\tau), \tag{2.173}$$

for $i > 1$.

Interpolation coefficients g^I are defined which are similar to the integration coefficients g,

$$g_{i,q}^I = (q-1)!\, c_{i,n+1}^{I(-q)}(1). \tag{2.174}$$

A recursive formula is available for these coefficients by using (2.173),

$$g_{i,q}^I = \begin{cases} \dfrac{1}{q} & i = 1, \\ \Gamma_{i-1}(1)g_{i-1,q}^I - \dfrac{h_I}{\psi_{i-1}(n+1)}g_{i-1}^I q + 1 & i \geq 2. \end{cases} \tag{2.175}$$

A formula for the requested velocity value is now available in terms of these coefficients,

$$\dot{\boldsymbol{r}}_{\text{out}} = \dot{\boldsymbol{r}}_{n+1} + h_I \sum_{i=1}^{k+1} g_{i,1}^I \boldsymbol{\phi}_i(n+1). \tag{2.176}$$

This expression is the Shampine-Gordon interpolation formula. The formula requires the extra calculation of the g^I coefficients.

Step-Size Control

The step size is controlled by keeping the local error at each step below a user-defined tolerance, ϵ. The tolerance is defined in terms of an absolute tolerance, ϵ_{abs}, and a relative tolerance, ϵ_{rel}. The local error for each component L being integrated must be lower than the absolute tolerance and the relative tolerance times the value of the component,

$$le_L \leq \epsilon_{\text{rel}}\dot{\boldsymbol{r}}_L + \epsilon_{\text{abs}} = \epsilon, \tag{2.177}$$

where $\dot{\boldsymbol{r}}_L$ represents the L^{th} component of the velocity vector.

The local error is estimated by subtracting from the corrected value $\dot{\boldsymbol{r}}_{n+1}$ a value given by a lower order corrector $\dot{\boldsymbol{r}}_{n+1}(k)$,

$$\boldsymbol{le}_{\boldsymbol{L}n+1}(k) \approx \dot{\boldsymbol{r}}_{n+1} - \dot{\boldsymbol{r}}_{n+1}(k). \tag{2.178}$$

The value of $\dot{\boldsymbol{r}}_{n+1}(k)$ is obtained by integrating a $(k-1)^{\text{th}}$ degree interpolating polynomial $\boldsymbol{P}_{k,n}^*$ that passes through the predicted point $\ddot{\boldsymbol{r}}_{n+1}^p$. This polynomial can be written by adding a term to the polynomial $\boldsymbol{P}_{k,n}$,

$$\boldsymbol{P}_{k,n}^*(t) = \boldsymbol{P}_{k,n}(t) + c_{k,n}(\tau)\boldsymbol{\phi}_{k+1}^p(n+1). \tag{2.179}$$

Using this polynomial in place of $\boldsymbol{P}^*_{k+1,n}$ in (2.159) leads to an expression for $\dot{\boldsymbol{r}}_{n+1}(k)$,

$$\dot{\boldsymbol{r}}_{n+1}(k) = \dot{\boldsymbol{r}}^p_{n+1} + h_{n+1} g_{k,1} \boldsymbol{\phi}^p_{k+1}(n+1). \tag{2.180}$$

The local error, (2.178), is estimated by subtracting (2.180) from (2.161),

$$\boldsymbol{le}_{n+1}(k) \approx h_{n+1}(g_{k+1,1} - g_{k,1}) \boldsymbol{\phi}^p_{k+1}(n+1). \tag{2.181}$$

At each step, this local error estimate is compared to the tolerance for each component. The integration method checks that

$$\sqrt{\sum_{L=1}^{3} \left(\frac{le_L}{\mathrm{WT_S}(L)} \right)^2} \leq \mathrm{EPS}, \tag{2.182}$$

where EPS is the maximum of the relative and absolute tolerances,

$$\mathrm{EPS} = \max(\epsilon_{\mathrm{rel}}, \epsilon_{\mathrm{abs}}), \tag{2.183}$$

and $\mathrm{WT_S}(L)$ is a weight for each component,

$$\mathrm{WT_S}(L) = |\dot{\boldsymbol{r}}_L| \frac{\epsilon_{\mathrm{rel}}}{\mathrm{EPS}} + \frac{\epsilon_{\mathrm{abs}}}{\mathrm{EPS}}. \tag{2.184}$$

The subscript S refers to single integration. Double integration equivalents are given in Section 4.2.4. Using (2.182) guarantees that (2.177) holds for every component ([25], pp. 175-179). If the local error is above the tolerance, the step fails, and is tried again with half the step size, $h_{n+1} = 0.5 h_{n+1\ (\mathrm{failed})}$ ([25], p. 117). Note that this error estimate is made with $\boldsymbol{\phi}^p_{k+1}(n+1)$, before the second evaluation is performed. Therefore, if a step fails, the second evaluation does not need to be performed.

A further consideration is made if a step fails after multiple tries. If a step fails after halving and trying again three times, the integration restarts as a first-order method. The method assumes that the failures are a sign of a discontinuity, so the method restarts as first order to handle the discontinuity correctly ([25], pp. 117-118).

If the step succeeds, the next step size, h_{n+2}, is chosen as a multiple of the current step size, $h_{n+2} = \rho h_{n+1}$, to keep the local error at the next step,

$$\boldsymbol{le}_{n+2}(k) \approx h_{n+2}(g_{k+1,1} - g_{k,1}) \boldsymbol{\phi}^p_{k+1}(n+2), \tag{2.185}$$

as close as possible to the tolerance. The modified divided difference at the next step,

$$\boldsymbol{\phi}^*_{k+1}(n+2) = \psi_1(n+2) \cdots \psi_k(n+2) f^p[t_{n+2}, \ldots, t_{n+2-k}], \tag{2.186}$$

is unknown. However, it may be approximated from $\boldsymbol{\phi}_{k+1}(n+1)$ if the divided differences are assumed to be slowly varying ([25], p. 111).

Because the step size h_{n+2} appears in the values of $\psi_i(n+2)$, and is needed to calculate the coefficients $g_{k+1,1}$ and $g_{k,1}$, there is no easy way to solve (2.181) for a value of h_{n+2} that meets the tolerance. Instead, a value of h_{n+2} is found that meets the tolerance if all the preceding steps were also taken with h_{n+2}. Using this assumption, and the assumption that the divided differences are slowly varying, the modified divided difference at $(n+2)$ is approximated,

$$\begin{aligned}\boldsymbol{\phi}^p_{k+1}(n+2) &\approx (\rho h_{n+1})(2\rho h_{n+1})\cdots(k\rho h_{n+1})f^p[t_{n+2},\ldots,t_{n+2-k}]\\ &\approx \rho^k\sigma_{k+1}(n+1)\boldsymbol{\phi}^p_{k+1}(n+1),\end{aligned} \tag{2.187}$$

where $\sigma_i(n+1)$ is defined,

$$\begin{aligned}\sigma_1(n+1) &= 1,\\ \sigma_i(n+1) &= \frac{h_{n+1}\cdot 2h_{n+1}\cdots(i-1)h_{n+1}}{\psi_1(n+1)\cdot\psi_2(n+1)\cdots\psi_{i-1}(n+1)} \qquad i>1.\end{aligned} \tag{2.188}$$

When the step size is constant, the coefficients $g_{k+1,1}$ and $g_{k,1}$ become the fixed-step Adams-Bashforth predictor coefficients, γ_k and γ_{k-1}, found in Section 2.3.3. The local error, (2.185), using $h_{n+2}=\rho h_{n+1}$ can now be approximated,

$$\boldsymbol{le}_{n+2}(k) = \rho^{k+1}h_{n+1}\gamma^*_k\sigma_{k+1}(n+1)\boldsymbol{\phi}^p_{k+1}(n+1), \tag{2.189}$$

where γ^*_k is the difference between the constant step size coefficients, $\gamma^*_k=\gamma_k-\gamma_{k-1}$. The value of $\sigma_{k+1}(n+1)$ can be found through a recursive formula,

$$\begin{aligned}\sigma_1(n+1) &= 1,\\ \sigma_i(n+1) &= (i-1)\alpha_{i-1}(n+1)\sigma_{i-1}(n+1) \qquad i>1.\end{aligned} \tag{2.190}$$

To solve for the value of ρ to get the next step size, $h_{n+2}=\rho h_{n+1}$, the Shampine-Gordon integrator calculates the approximated error ($\mathrm{ERK_S}$) that would be made if the previous steps had been taken with h_{n+1},

$$\mathrm{ERK_S} = |h_{n+1}\gamma^*_k\sigma_{k+1}(n+1)|\sqrt{\sum_{L=1}^{3}\left(\frac{\phi^p_{L\,k+1}(n+1)}{\mathrm{WT_S}(L)}\right)^2}. \tag{2.191}$$

From (2.189), the approximated error at the next step with a step size of ρh_{n+1} is $\rho^{k+1}\mathrm{ERK_S}$, so the optimal value of ρ to satisfy the tolerance ϵ can be found,

$$\rho = \left(\frac{\mathrm{EPS}}{\mathrm{ERK_S}}\right)^{\frac{1}{k+1}}. \tag{2.192}$$

However, because of the assumptions made in this derivation, the integrator uses a so-called "chicken factor" [26] of 0.5, giving a more conservative value of ρ,

$$\rho = \left(\frac{0.5\text{EPS}}{\text{ERK}_S} \right)^{\frac{1}{k+1}} . \tag{2.193}$$

Shampine-Gordon places further restrictions on changes to the step size ([25], pp. 115-118). If the calculated value of ρ is greater than 2, the step size is only doubled. The step size is not changed at all if the calculated value of ρ is between 0.9 and 2. And if ρ is less than 0.5, the step size is cut in half. These restrictions serve two purposes. First, bounding changes in the step size between 0.5 and 2 keeps the method stable. Second, not changing the step size when ρ is between 0.9 and 2 keeps the step size constant for a significant number of steps. Shampine and Gordon sought to keep a constant step size as much as possible. With a constant step size, the coefficients g do not have to be calculated, because they are simply the known Adams-Bashforth coefficients. Not needing to calculate the coefficients provides a reduction in overhead when the step size is constant. Also, keeping a constant step size allows the order of the method to be increased, which is described in the following section.

Variable Order

The Shampine-Gordon method also controls the local error by adjusting the order of the method ([25], pp. 112-115). Local error estimates, similar to (2.191), are made for $(k-1)$ and $(k-2)$, and the order is reduced from k to $(k-1)$ if these estimates indicate that reducing the order would give less error. These estimates rely on $\boldsymbol{\phi}_k^p(n+1)$ and $\boldsymbol{\phi}_{k-1}^p(n+1)$, so they are made before the second evaluation is performed. Therefore, the order can be reduced if the step fails. Reduction in order is considered before the step size is changed, and the step size is allowed to change even if the order is changed on the same step.

Increases in order are considered after the step size regulation. The order is increased only when the step size has been kept constant. A local error estimate is made for $(k+1)$, and the order is increased if the estimate indicates the error would be reduced. The local error estimate for $(k+1)$ uses the value $\boldsymbol{\phi}_{k+2}(n+1)$ ([25], p. 113), so the second evaluation must be performed before increasing the order can be considered.

The variable-order algorithm allows Shampine-Gordon to be a self-starting method. The method is started as first order ($k = 1$), so only the initial conditions are required. The order is then increased at every step until either a step fails, the order selection algorithms indicate to lower the order, or the maximum order of 12 is reached ([25], pp 118-120). The

method is started as first order, so the initial step size is chosen accordingly. Consider a method of zeroth order, where $\dot{\boldsymbol{r}}_1 = \dot{\boldsymbol{r}}_0$. The error of this method can be found from a Taylor series approximation,

$$\dot{\boldsymbol{r}}(t_1) = \dot{\boldsymbol{r}}(t_0) + h\ddot{\boldsymbol{r}}(t_0) + \mathrm{O}(h^2), \tag{2.194}$$

so the local error is

$$\dot{\boldsymbol{r}}(t_1) - \dot{\boldsymbol{r}}_1 = h\ddot{\boldsymbol{r}}(t_0). \tag{2.195}$$

Assume that the error of the first-order method is h times the error of the zeroth-order method, so keeping the local error within ϵ requires

$$\epsilon \approx |le_1| \approx h^2|\ddot{\boldsymbol{r}}_0|. \tag{2.196}$$

Solving for the step size gives,

$$h \approx \sqrt{\frac{\epsilon}{\ddot{\boldsymbol{r}}_0}}. \tag{2.197}$$

To be conservative, one quarter of this value is used. Accounting for the multiple components of $\boldsymbol{f}$, and the relative tolerance, the initial step size is

$$h_1 = \frac{1}{4}\sqrt{\frac{\mathrm{EPS}}{\sqrt{\sum_{L=1}^{3}\left(\frac{\ddot{\boldsymbol{r}}_{L\,0}}{\mathrm{WT_S}(L)}\right)^2}}}. \tag{2.198}$$

In case the initial accelerations are near zero, an upper bound is placed on h, equal to the size of the step to the first requested output point. In addition, a lower bound is placed on the initial step size, equal to $4\epsilon_m t_0$, where ϵ_m is machine epsilon, defined as the smallest positive number where $1 + \epsilon_m > 1$ on the machine. Step sizes lower than this bound cause a significant amount of round-off error.

Implementation

In Chapter 5 the Shampine-Gordon integrator is compared to other methods. In those tests, the implementation follows the code published by Shampine and Gordon ([25], pp. 186-231).

2.6 Summary

The derivations of four fixed-step multi-step integrators, the Adams, summed Adams, Störmer-Cowell, and Gauss-Jackson integrators, as well as the variable-step multi-step

Shampine-Gordon integrator, have been presented in this chapter. These derivations are needed to understand the derivation of the variable-step double-integration method derived in Chapter 4. Also, comparing these methods to one another motivates the need for the variable-step double-integration method, and the benefits of that method can be seen by comparing it to the methods presented in this chapter. The comparisons to motivate the need for the variable-step double-integration method are presented in Section 4.1, and the comparisons of that method to other integrators are presented in Chapter 5. Before any comparisons can be performed, a discussion of how to measure integration accuracy is required, which is presented in the next chapter.

Chapter 3

Testing Integrators

3.1 Introduction

When comparing integration methods to one another, two things must be considered: accuracy and speed. To compare the methods, accuracy tests are first performed for various types of orbits. The accuracy tests are used to tune the integrators to give equivalent accuracy, and the integrators are then tested for speed. Since the speed tests are made for equivalent accuracy, this testing procedure identifies the most efficient integration method for each type of orbit.

While speed testing is simply a matter of measuring computation time, accuracy testing is a more complicated issue. Although assessment of integration accuracy is a long-standing problem, much of the focus of the available literature is on the three-body problem in astronomy, particularly over long periods of time. Because of the chaotic nature of such systems, accuracy is desired to assess characteristics of chaotic regions in time and space. In astrodynamics, the concern is with the problem of orbiting a planet with geopotential, atmospheric drag, solar radiation and other perturbations, and often for a relatively modest number of orbits. Accuracy is desired here for precise knowledge of spacecraft position and orbit over a short period of time. Although the problems are similar, the needs are sufficiently different that the integrator, including order and step size, may be chosen differently. Nevertheless, the techniques developed by astronomers to assess integrators may be used to develop a means of assessing integrators for astrodynamics.

Attempts at characterization of integration error frequently stop with two-body integration because of the ability to determine absolute accuracy. Perturbations have a significant effect on integration accuracy that is not apparent from a two-body study, as

is shown below.

An N^{th} order system of ordinary differential equations, with initial conditions given at $t = t_0$, can be written in the general form [27]

$$\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y}), \qquad \boldsymbol{y}(t_0) = \boldsymbol{y_0}, \tag{3.1}$$

where $\boldsymbol{y}$ and $\boldsymbol{f}$ are vectors of N functions and $\boldsymbol{y_0}$ is a vector of N constants. In astrodynamics $\boldsymbol{y}$ consists of three position functions and three velocity functions, and t is time. If some numerical integration algorithm is used to solve (3.1), the algorithm generates an approximate solution $\tilde{\boldsymbol{y}}$. After n steps the accumulated error is

$$\boldsymbol{\xi_n} = \boldsymbol{y}(t_n) - \tilde{\boldsymbol{y}}_{\boldsymbol{n}}, \tag{3.2}$$

where $\boldsymbol{y}(t_n)$ is the exact solution, $\tilde{\boldsymbol{y}}_{\boldsymbol{n}}$ is the computed solution at t_n, $t_n = t_0 + nh$, and h is the step size. The error can be written in the form [27]

$$\boldsymbol{\xi}_n = [\boldsymbol{y}(t_n) - \boldsymbol{y_n}] + [\boldsymbol{y_n} - \tilde{\boldsymbol{y}}_{\boldsymbol{n}}], \tag{3.3}$$

where the first difference is the *truncation error*, and the second difference is the *round-off error* [6]. Truncation error exists because the numerical integration algorithm has been truncated at some (locally correct) order, p. Truncation error is dependent on the step size h and the order p, and decreases as the step size is decreased. Round-off error exists because computers only keep track of numbers to a finite number of significant digits, and some error is introduced during every calculation. Round-off error increases as the step size is decreased, because more computations are performed.

Various techniques exist to estimate the error, $\boldsymbol{\xi}$, of a numerical integrator. Each technique has different strengths and weaknesses. In this chapter, six such techniques are discussed, and one is chosen to use in the comparisons made in Chapters 4 and 5. All the techniques involve the numerical comparison of two prospective orbits. This comparison is done through the computation of the error ratio, which provides a figure of relative merit between two integrated orbits. One of the orbits is produced by the integrator being tested; the other, the reference orbit, covers the same time period and initial conditions and may be produced by either integration or analytic computation.

3.2 Error Ratio

In a study of integration methods, Merson used an error ratio defined in terms of the RMS error of the integration as a measure of integration accuracy [21]. First define

position errors as

$$\Delta r = |r_{\text{computed}} - r_{\text{reference}}|. \tag{3.4}$$

The RMS position error can be calculated,

$$\Delta r_{\text{RMS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\Delta r_i)^2}. \tag{3.5}$$

The RMS position error is normalized by the apogee distance and the number of orbits to find the position error ratio,

$$\rho_r = \frac{\Delta r_{\text{RMS}}}{r_A N_{\text{orbits}}}. \tag{3.6}$$

In [21], Merson uses a position error ratio to test integrators. His error ratio was only normalized by the apogee distance; the number of orbits is added here to make error ratios of orbits of different eccentricities more comparable. Also, a velocity ratio test is added here because velocity and position are often integrated with different numerical integrators, and it may be useful to estimate the error in orbital elements, which depend directly on the velocity. The velocity error ratio is the RMS velocity error normalized by the number of orbits and the perigee speed,

$$\rho_v = \frac{\Delta v_{\text{RMS}}}{v_P N_{\text{orbits}}}. \tag{3.7}$$

Both position and velocity error ratios are computed for each of the testing techniques described in the next section.

3.3 Testing Techniques

Six testing techniques are described in this section, the two-body test, the step-size halving test, comparing with a higher-order integrator, use of integral invariants, and Zadunaisky's technique. The use of these techniques is demonstrated by testing them on three test-case orbits and two integration methods, which are explained in the next section.

3.3.1 Test Cases

If an integrator is expected to perform satisfactorily over a wide variety of orbits, some representative sample of these orbits is needed as a test set of initial conditions. In

particular, forces that stress an integrator should be included to give a sense of the worst case. For the case of space surveillance catalog maintenance, circular orbits near the earth and at geosynchronous altitude represent the bulk of the catalog; the addition of a high-eccentricity elliptical orbit with perigee dipping well into the atmosphere gives a case that stresses the integrator.

Three test cases are considered, a low earth orbit (LEO), a highly elliptical orbit (HEO), and a geosynchronous orbit (GEO). The initial conditions of these test cases are shown in Table 3.1. The test cases all have an initial epoch of 1999-10-01 00:00:00 UT, and a ballistic coefficient of $0.01\,{}^{\mathrm{m}^2}/_{\mathrm{kg}}$.

Table 3.1: Test Case Initial Conditions

Test Case	Perigee Height (km)	e	i (°)
LEO	300	0.0	40
HEO	200	0.75	40
GEO	35786	0.0	0.01

The purpose here is to compare accuracy assessment techniques, so in addition to test orbits, test integrators are needed on which to try the various techniques. Two integrators commonly used in astrodynamics are used in the tests. The first is a fourth-order Runge-Kutta, as described in Section 2.2.2. The Runge-Kutta integrator is a single-step, single-integration, fixed-step integrator. The step sizes used for the Runge-Kutta integrator are five seconds for the LEO and HEO cases, and one minute for the GEO case. The second integrator is an eighth-order Gauss-Jackson integrator, using the implementation described in Section 2.3.9. The corrector is applied only once at each integration step, giving a predict, evaluate, correct cycle. For the Gauss-Jackson integrator, the step sizes are 30 seconds for the LEO and HEO cases, and 20 minutes for the GEO case. The testing techniques show that these step sizes give reasonably accurate results.

The tests are performed using the Special-K software suite ([28]), developed by the Naval Research Laboratory, which is used operationally by Naval Network and Space Operations Command (NNSOC, formerly Naval Space Command). Operationally, the software uses the Gauss-Jackson integrator. These tests are performed with a research version of the software, which has been modified to allow different integrators to be used.

3.3.2 Two-Body Test

If the force is a simple two-body (Kepler) force, an exact solution is available for comparison. The advantage of this technique is that the error is then known exactly, but the disadvantage is that the force may not be realistic. This test does not necessarily indicate how well the integrator handles perturbations. Orbits typically integrated for space surveillance may cover a time period of several days or more. During that time, the integrated effects of the perturbations cause a substantial deviation from the two-body solution. Integrators that handle the two-body force well, but not one or more of the perturbations, will appear accurate with the two-body test though they are not accurate in real problems. However, the two-body test is useful for evaluating other testing techniques. The other techniques should give the same results as the two-body test when only the two-body force is used in those techniques.

Fox [29] performed extensive tests on integrators using the two-body test. Part of the purpose of his study was to assess accuracy in light of the execution time, so he put "dead weight" into the force evaluation — computations that do nothing but soak up processor time — in an attempt to simulate the evaluation-dominated execution time of integrations with realistic force models. Montenbruck [30] also used the two-body force to assess integrators.

Table 3.2 shows position and velocity error ratios with the two-body test for both Runge-Kutta and Gauss-Jackson integrators. The position and velocity error ratios are found using (3.6) and (3.7), respectively. Ephemeris generated by the numerical integrators over a three-day time span is used for r_{computed} and v_{computed}, and $r_{\text{reference}}$ and $v_{\text{reference}}$ are the values given by the exact analytic solution. Though the cases use different step sizes, the ephemeris is generated at one minute intervals, so that N in (3.5) is 4321. For the GEO case with the Gauss-Jackson integrator, where the step size is 20 minutes, the intermediate points are found using a 5^{th} order interpolator. Table 3.3 gives the maximum position error over three days for each test case. A comparison of Table 3.2 to Table 3.3 demonstrates how an error ratio corresponds to the maximum position error, which may be of interest.

Table 3.2 shows that the Gauss-Jackson integrator is more accurate in every case, except in the velocity for the GEO case. Gauss-Jackson is roughly four orders of magnitude more accurate than Runge-Kutta for the low earth orbit, roughly one order of magnitude more accurate for the eccentric orbit, and of comparable accuracy for the geosynchronous orbit. Both integrators show the least accuracy in position with the eccentric orbit, though the Gauss-Jackson has the least accuracy in velocity with the geosynchronous orbit.

With the exact error of the integrators for the two-body problem known, it is possible to

Table 3.2: Two-Body Test Results

Test Case	Runge-Kutta		Gauss-Jackson	
	Position	Velocity	Position	Velocity
LEO	2.05×10^{-10}	2.05×10^{-10}	1.21×10^{-14}	1.19×10^{-14}
HEO	2.49×10^{-10}	5.15×10^{-10}	1.03×10^{-11}	2.26×10^{-11}
GEO	3.27×10^{-11}	3.25×10^{-11}	8.98×10^{-12}	8.58×10^{-11}

Table 3.3: Two-Body Position Error (mm)

Test Case	Runge-Kutta	Gauss-Jackson
LEO	133	0.00616
HEO	286	15.0
GEO	7.21	2.61

get an indication of error not measured in the other techniques. The techniques described in the subsequent sections are each tested two ways. In the first test only the two-body force is considered, so that the error measured by the technique can be compared to the known error. This comparison gives an indication of how well the techniques measure integration accuracy. In the second test a perturbation model is used. The perturbation forces are 36×36 WGS-84 geopotential, the Jacchia 70 drag model [31], and lunar and solar forces. Note that all of these perturbations are continuous forces. To simplify this study, discontinuous forces such as solar radiation pressure are not considered.

3.3.3 Step-Size Halving

For the step-size halving test, the reference integration is produced with the same integrator but with the step size cut in half. Because the truncation error is related to the step size, this technique can give a good estimate of truncation error, and even an estimate of the order of the integration method, provided the step size is large enough that truncation error dominates the total error. If the step size is too small, round-off error dominates over truncation error. The step-size halving test does not give reliable results in regions where round-off error dominates. However, the test can be used to identify the region where round-off error dominates. If the results of the step-size halving test appear worse as the step size is decreased, round-off error is dominating the total error.

Tables 3.4 and 3.5 show error ratios using the step-size halving test, for the two-body force and for the full force model, respectively. The error ratios with the two-body force

are the same order of magnitude as the true error ratios in Table 3.2, and even match to one significant digit, except for the LEO case with Gauss-Jackson. These results show that step-size halving gives a reasonable measure of integration error.

Table 3.4: Two-Body Step-Size Halving Results

Test Case	Runge-Kutta		Gauss-Jackson	
	Position	Velocity	Position	Velocity
LEO	1.96×10^{-10}	1.96×10^{-10}	6.70×10^{-14}	6.70×10^{-14}
HEO	2.34×10^{-10}	4.85×10^{-10}	1.04×10^{-11}	2.29×10^{-11}
GEO	3.07×10^{-11}	3.05×10^{-11}	8.97×10^{-12}	8.62×10^{-11}

Table 3.5: Perturbed Step-Size Halving Results

Test Case	Runge-Kutta		Gauss-Jackson	
	Position	Velocity	Position	Velocity
LEO	8.68×10^{-10}	8.70×10^{-10}	3.09×10^{-9}	3.10×10^{-9}
HEO	2.48×10^{-10}	5.07×10^{-10}	1.05×10^{-8}	2.24×10^{-8}
GEO	3.07×10^{-11}	3.05×10^{-11}	9.14×10^{-12}	8.62×10^{-11}

A comparison of Table 3.4 to Table 3.5 shows that the numerical integrators perform worse with perturbations than with only the two-body force for the low-earth and eccentric cases, but nearly the same for the geosynchronous case. This result may be because the geosynchronous orbit is not subject to drag, and the presence of drag causes an increase in integration error. This result shows that the two-body test described above does not capture all of the error of an integrator. The effect of perturbations on the low-perigee cases is more noticeable with the Gauss-Jackson integrator. The drag model used in these tests, Jacchia 70, involves a table look-up to find the atmospheric density. The table look-up is performed with a linear interpolator, giving density values that are continuous but not smooth. Because the density is not smooth the drag force is also non-smooth. Since multi-step methods involve the use of backpoints, non-smooth forces have a greater effect on multi-step methods than on single-step methods such as Runge-Kutta.

A more formal error analysis is also possible with step-size halving. In theory the global error is on the order of the step size raised to the power of the order, p, of the numerical integrator [32],

$$y - \tilde{y} = \xi \approx Ch^p, \tag{3.8}$$

where y is the actual solution, $\tilde{y}$ is the numerical solution, and C is some constant that depends on the numerical integrator. An estimate of ξ can be found by comparing results

with half the step size. Considering the numerical solution, $\tilde{y}$, to be a function of the step size h, the error with half the step size can be found,

$$[y - \tilde{y}(h/2)] \approx \frac{Ch^p}{2^p} \approx \frac{[y - \tilde{y}(h)]}{2^p}. \tag{3.9}$$

This equation can be solved for the error,

$$\begin{aligned} 2^p[y - \tilde{y}(h/2)] &\approx [y - \tilde{y}(h)] \\ (2^p - 1)[y - \tilde{y}(h/2)] &\approx \tilde{y}(h/2) - \tilde{y}(h) \\ \xi(h/2) &\approx \frac{\tilde{y}(h/2) - \tilde{y}(h)}{2^p - 1} \end{aligned} \tag{3.10}$$

In order for (3.10) to be valid, (3.8) must hold true. This condition can be checked by forming the quotient

$$\frac{[y - \tilde{y}(h/2)] - [y - \tilde{y}(h/4)]}{[y - \tilde{y}(h)] - [y - \tilde{y}(h/2)]} = \frac{\tilde{y}(h/4) - \tilde{y}(h/2)}{\tilde{y}(h/2) - \tilde{y}(h)} \approx \frac{Ch^p/2^p - Ch^p/4^p}{Ch^p - Ch^p/2^p} = \frac{1}{2^p}. \tag{3.11}$$

The quotients should approach 2^{-p} as the step size decreases. But when round-off becomes a factor, the quotients drift away from the theoretical value. As long as the quotients indicate that (3.8) is valid, round-off error is not a major concern and (3.10) can be used. Equations (3.10) and (3.11) have been written in scalar form; in practice they can be applied to each component of the vector state $\boldsymbol{x}$.

The step-size halving test does give a good indication of error, and can be used to measure truncation error. The test is relatively easy to perform with fixed-step integrators, for which the step size is controlled by the user. However, variable-step methods, in which the step size is controlled internally by the integrator, cannot use this test. Because some of the integrators tested in this work are variable step, the step-size halving technique cannot be used.

3.3.4 Comparison with High-Order Integrator

The comparison integration may be a higher-order high-accuracy integrator. The advantage of this technique is that perturbations can be tested. However, this technique relies on the assumption that the higher-order integrator is correct, or more correct, than the integrator being tested. This assumption is not necessarily true. For instance, the higher-order method might be using a step size that is too large to give adequate accuracy. Tables 3.6 and 3.7 show error ratios comparing the two test integrators to a 14^{th}-order Gauss-Jackson, with the two-body force and full perturbation forces, respectively. The step size used in the 14^{th}-order Gauss-Jackson are 15 seconds for cases 1

and 2, and 1 minute for case 3. The 14^{th} order Gauss-Jackson integrator uses a P(EC)^n implementation, with a maximum number of corrector cycles of six (see Section 2.3.9). The tolerance used in the corrector cycles is 1×10^{-12}.

Table 3.6: Two-Body High-Order Results

Test Case	Runge-Kutta		Gauss-Jackson	
	Position	Velocity	Position	Velocity
LEO	2.05×10^{-10}	2.05×10^{-10}	1.81×10^{-14}	1.81×10^{-14}
HEO	2.49×10^{-10}	5.16×10^{-10}	1.04×10^{-11}	2.29×10^{-11}
GEO	3.28×10^{-11}	3.25×10^{-11}	8.99×10^{-12}	8.58×10^{-11}

Table 3.7: Perturbed High-Order Results

Test Case	Runge-Kutta		Gauss-Jackson	
	Position	Velocity	Position	Velocity
LEO	3.17×10^{-9}	3.17×10^{-9}	3.09×10^{-9}	3.09×10^{-9}
HEO	8.95×10^{-9}	1.96×10^{-8}	1.05×10^{-8}	2.26×10^{-8}
GEO	3.27×10^{-11}	3.25×10^{-11}	8.77×10^{-12}	8.58×10^{-11}

A comparison of Table 3.6 to Table 3.2 shows that the high-order test results are the same order of magnitude as the exact two-body results in every case. For the Runge-Kutta integrator, the test matches the true results to at least two significant figures in each case. For the Gauss-Jackson integrator, the test matches the true results to two significant figures for the eccentric case, and the geosynchronous case. A comparison of these results to Table 3.4 shows that the higher-order test does a better job of measuring the two-body error than the step-size halving test for the Runge-Kutta integrator, as well as the low-earth case with the Gauss-Jackson method. The higher-order test and the step-size halving test give equivalent results for the other two cases. As a reference, the results of the 14^{th}-order Gauss-Jackson method itself in the two-body test are given in Table 3.8. The higher-order test gives the best results when the error of the method being tested is significantly greater than the method used as the reference.

A comparison of Table 3.7 to Table 3.5 shows that the step-sizing halving results and the high-order results with perturbations are the same order of magnitude in all cases except the LEO and HEO cases with Runge-Kutta. The higher-order test indicates more error than the step-size halving test for these two cases. The Gauss-Jackson results match to at least one significant figure in each case, and the Runge-Kutta results match to one significant figure in the GEO case. The Gauss-Jackson results match closely between

Table 3.8: Two-Body Test of 14^{th}-Order Gauss-Jackson

Test Case	Error Ratio	
	Position	Velocity
LEO	8.84×10^{-15}	8.85×10^{-15}
HEO	1.37×10^{-13}	2.96×10^{-13}
GEO	1.42×10^{-14}	1.39×10^{-14}

the step-size halving test and the high-order test because in both tests, the reference integrator is a Gauss-Jackson integrator with a low step size. Though the accuracy of the 14^{th}-order Gauss-Jackson method itself cannot be measured with perturbations, using the step-size halving test on the method gives an indication of its accuracy. Results of the step-size halving test for the 14^{th}-order method are shown in Table 3.9.

Table 3.9: Step-Size Halving Test of 14^{th}-Order Gauss-Jackson

test #	Error Ratio	
	Position	Velocity
LEO	1.58×10^{-9}	1.58×10^{-9}
HEO	8.12×10^{-9}	1.78×10^{-8}
GEO	5.79×10^{-14}	5.58×10^{-14}

3.3.5 Reverse Test

In this technique the test and reference orbits come from the same integrator. The test orbit is produced by integrating forward from the initial conditions, and the reference orbit is produced by integrating backward to the original starting time using the final state of the first integration as initial conditions. These two integrations should be identical, and any difference between them is due to integration error. Using this technique to measure integration error is advantageous because it is a relatively simple procedure to perform. A disadvantage with this technique is that it does not measure any reversible integration error. Any error that is an odd function of the step size is canceled when the sign of the step changes on the reverse integration. The reverse test has been used extensively for integration accuracy checks, including recently in the N-body problem [33].

Tables 3.10 and 3.11 show results of the reverse test with a two-body force and full

perturbation forces, respectively. Again the tests are over a three-day interval with ephemeris generated at one-minute increments. A comparison of Table 3.10 to Table 3.2 shows that the reverse test fails to capture a significant portion of the error; in the GEO case for the Runge-Kutta integrator, only about a tenth of the error is captured. This results may be an example of the weakness of the reverse test pointed out by Zadunaisky [34], who demonstrated for the three-body problem that the reverse test will certify that a second-order Adams-Moulton multi-step method is perfectly accurate because of the symmetry of the equations used under time reversal. A comparison of Table 3.11 to Tables 3.5 and 3.7 shows that the reverse test is also inconsistent with the other techniques in the presence of perturbations.

Table 3.10: Two-Body Reverse Test Results

Test Case	Runge-Kutta		Gauss-Jackson	
	Position	Velocity	Position	Velocity
LEO	2.27×10^{-10}	2.27×10^{-10}	6.15×10^{-14}	6.15×10^{-14}
HEO	5.14×10^{-11}	1.09×10^{-10}	2.21×10^{-11}	4.68×10^{-11}
GEO	3.58×10^{-12}	3.59×10^{-12}	2.11×10^{-11}	2.11×10^{-11}

Table 3.11: Perturbed Reverse Test Results

Test Case	Runge-Kutta		Gauss-Jackson	
	Position	Velocity	Position	Velocity
LEO	1.97×10^{-10}	1.98×10^{-10}	3.00×10^{-9}	3.01×10^{-9}
HEO	4.67×10^{-11}	1.03×10^{-10}	1.99×10^{-11}	1.10×10^{-10}
GEO	3.51×10^{-12}	3.52×10^{-12}	2.20×10^{-11}	2.20×10^{-11}

3.3.6 Integral Invariants

Another frequently-used technique for integration accuracy assessment is to check the invariance of integral invariants such as energy, which is easy to perform. This technique has been used recently for integrators applied in the N-body problem [33]. The main drawback is that this test does not capture all errors. For example, energy invariance does not measure in-track errors, because an orbit shifted in time has the same energy. In general, the more forces present that break a particular symmetry, the fewer conserved quantities and thus the fewer quantities that can be checked; the presence of drag means

that energy is no longer conserved at all. Huang and Innanen [35] showed that this accuracy check is not exact and reliable and suggested a revised technique. However, even their revised technique cannot measure integration error when drag is present. Because drag is one of the perturbation forces under consideration in this test, no results can be given for this technique.

3.3.7 Zadunaisky's Test

Zadunaisky, in [36], [27], [37], and [34], suggested a technique for measuring numerical integration error. This technique is based on the construction of an analytical function near the solution to the actual problem, then constructing differential equations for which this function is an exact solution. The method has two desirable properties. First, there is an exact solution, because the problem is constructed to match the original solution. Second, the solution is realistic, because it is close in some sense to the solution of the real problem. The technique is like the two-body test in providing an absolute reference for error computation, but has a behavior that mimics real forces.

First a set of polynomials $\boldsymbol{P}(t)$ are determined that represent the components of $\tilde{\boldsymbol{y}}$, the numerical solution of (3.1). A *pseudo-system* of equations is constructed from these polynomials, for which $\boldsymbol{P}(t)$ is the exact solution,

$$\dot{\boldsymbol{z}} = \boldsymbol{f}(t, \boldsymbol{z}) + \boldsymbol{D}(t), \tag{3.12}$$

where

$$\boldsymbol{D}(t) = \dot{\boldsymbol{P}}(t) - \boldsymbol{f}(t, \boldsymbol{P}(t)), \tag{3.13}$$

and where (3.12) has the same initial conditions as the polynomial,

$$\boldsymbol{z}(t_0) = \boldsymbol{P}(t_0). \tag{3.14}$$

When the pseudo-system is integrated with the same numerical integrator used to create $\tilde{\boldsymbol{y}}$, the errors of the pseudo-system,

$$\boldsymbol{\xi} = \tilde{\boldsymbol{z}} - \boldsymbol{P}(t), \tag{3.15}$$

may be a good approximation of the error in the original problem.

In [36], Zadunaisky gave conditions for $\boldsymbol{P}(t)$ under which this technique gives a good approximation of the original error,

$$\left.\begin{matrix} ||\tilde{\boldsymbol{y}} - \boldsymbol{P}(t)|| \\ ||\tilde{\boldsymbol{y}}^{p+1} - \boldsymbol{P}^{p+1}(t)|| \end{matrix}\right\} \leq O(h^2), \tag{3.16}$$

where p is the order of the numerical integrator, and $\tilde{\boldsymbol{y}}^{p+1}$ are the numerical approximations for the $(p+1)^{\text{th}}$ derivatives of $\boldsymbol{y}$. These conditions are based on error propagation theory, and are meant to ensure that the asymptotic behavior of the errors accumulated in the numerical integration of both the original system and the pseudo-system are the same. In [34], Zadunaisky gives a different condition, which is that $\boldsymbol{D}(t)$ must not be larger than either the local truncation error or the local round-off error.

In [37] and [34], Zadunaisky found the polynomial $\boldsymbol{P}(t)$, needed to form the pseudo-system, using Newton's interpolation formula with backward divided differences. Because an $\tilde{N}^{\text{th}}$ degree polynomial is needed to interpolate $\tilde{N}+1$ points, the original interval is divided into subintervals to avoid the problems involved with interpolating data to a high degree polynomial. After integrating the original problem for $\tilde{N}$ steps, where $\tilde{N} \leq 10$, he applies Newton's formula to obtain polynomials $\boldsymbol{P}(t)$ of $\tilde{N}^{\text{th}}$ degree. Each polynomial $P_i(t)$ interpolates one of the components of position or velocity through the $\tilde{N}+1$ points spanned by the $\tilde{N}$ steps. These polynomials are used to construct the pseudo-system, (3.12), and the error estimate is obtained. This process is then repeated using the last point of the previous set as the first point on the next set, and using $\tilde{\boldsymbol{y}}_{\tilde{N}+1}$ and $\tilde{\boldsymbol{z}}_{\tilde{N}+1}$ as initial conditions in the original system and pseudo-system, respectively. Using this method, the solution to the pseudo-system over the entire interval, $\boldsymbol{z}(t)$, is a function of successive $\tilde{N}^{\text{th}}$ degree polynomials that match up the subintervals of $\tilde{N}+1$ points. Therefore $\boldsymbol{z}(t)$ is continuous over the entire interval, but its derivative is discontinuous at the last point of each subinterval. Zadunaisky claimed that these discontinuities are irrelevant to the validity of the technique.

To implement Zadunaisky's technique for a given test case, ephemeris $\tilde{\boldsymbol{y}}(t)$ is first generated by numerically integrating the test case. The ephemeris should be generated at time increments equal to the step size h of the numerical integrator. This ephemeris is then used to find coefficients of the polynomials $\boldsymbol{P}(t)$ at each subinterval. The polynomials are of the form

$$P_i(\tilde{t}) = a_0 + a_1\tilde{t} + a_2\tilde{t}^2 + ... + a_{\tilde{N}}\tilde{t}^{\tilde{N}}, \tag{3.17}$$

where $\tilde{t}$ is the time since the beginning of the subinterval, and the subscript i refers to the component of the state $\tilde{\boldsymbol{y}} = [r_x\, r_y\, r_z\, v_x\, v_y\, v_z]^T$ that the polynomial fits. To make the polynomial exactly match the ephemeris at each of the $\tilde{N}+1$ points on the subinterval, the coefficients $a_0 ... a_{\tilde{N}}$ are found by solving the system

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \tilde{t}_1 & \tilde{t}_1^2 & \dots & \tilde{t}_1^{\tilde{N}} \\ 1 & \tilde{t}_2 & \tilde{t}_2^2 & \dots & \tilde{t}_2^{\tilde{N}} \\ \vdots & & & & \\ 1 & \tilde{t}_{\tilde{N}} & \tilde{t}_{\tilde{N}}^2 & \dots & \tilde{t}_{\tilde{N}}^{\tilde{N}} \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{\tilde{N}} \end{Bmatrix} = \begin{Bmatrix} \tilde{y}_i(t_0) \\ \tilde{y}_i(t_1) \\ \tilde{y}_i(t_2) \\ \vdots \\ \tilde{y}_i(t_{\tilde{N}}) \end{Bmatrix}, \tag{3.18}$$

where the subscripts on $\tilde{t}$ and t refer to the points on the subinterval, and $\tilde{t}_0 = 0$. This system can be solved for the coefficients by inverting the first matrix and multiplying it by the vector on the right-hand side. Though this procedure must be repeated for each of the six components and for each subinterval, the matrix inversion only needs to be performed once, because the matrix only depends on the order $\tilde{N}$, and the time step h. This matrix is always non-singular, so the inversion can be performed.

After the coefficients are found, the same numerical integrator is used to generate another set of ephemeris, but with the force model modified so that the pseudo-system (3.12) is being integrated. Normally, the force model returns an acceleration based on position, velocity, and time, $\ddot{\boldsymbol{r}} = \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}})$. Instead, the coefficients for the appropriate subinterval are used to find the values of $\boldsymbol{P}$ and $\dot{\boldsymbol{P}}$, and the force model returns

$$\ddot{\boldsymbol{r}} = \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}}) + \dot{\boldsymbol{P}}(t) - \boldsymbol{f}(t, \boldsymbol{P}(t)). \tag{3.19}$$

Note that $\boldsymbol{P}$ is a six-component vector consisting of both position and velocity, though $\ddot{\boldsymbol{r}}$ is only a three-component vector. Therefore, the vector $\dot{\boldsymbol{P}}$ used in (3.19) is a three-component vector consisting of the first derivative of the velocity polynomials. This formulation makes (3.19) have a different form than (3.12), but this change makes the technique easier to implement.

There are two practical considerations to make when generating and using the coefficients in the modified force model. First, the ephemeris from which the coefficients are generated must be in the same coordinate system in which the integration is performed. Second, conversions may be necessary if the units of the ephemeris, and of the time $\tilde{t}$ used in the polynomial equations, are different from the units used by the integrator.

The ephemeris generated in the second integration, $\tilde{\boldsymbol{z}}$, are compared to the original ephemeris, $\tilde{\boldsymbol{y}}$, to determine an error ratio. For the GEO case, with the Runge-Kutta integrator using only the two-body force, the position error ratio is 2.51×10^{-7} with a 9^{th} order polynomial, $\tilde{N} = 9$. Table 3.2 shows that the actual error ratio is 3.27×10^{-11}, so the technique produces a result four orders of magnitude too high. Examining the first 90 minutes of data, the technique gives an error ratio that is two orders of magnitude too high. When $\tilde{N} = 6$, the error ratio given by the technique is 1.90×10^{-9}, two orders of magnitude too high. However, the error ratio over 90 minutes is the correct order of magnitude. With $\tilde{N} = 5$, the error ratio is 5.96×10^{-10}, one order of magnitude too high. Over the 90 minute time span, the technique gives an error ratio that is an order of magnitude too low. These results highlight the difficulty in choosing an appropriate degree polynomial.

Because the reliability of Zadunaisky's technique depends on how well the polynomial fits the ephemeris, an improvement in the method of determining the polynomials is

suggested to improve the results. In addition to matching the values of the ephemeris, the derivatives are also matched at the end-points of each subinterval. So for a subinterval consisting of $\tilde{N}-1$ points, an $\tilde{N}^{\text{th}}$ degree polynomial is found by modifying (3.18),

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \tilde{t}_1 & \tilde{t}_1^2 & \dots & \tilde{t}_1^{\tilde{N}} \\ . & & & & \\ . & & & & \\ 1 & \tilde{t}_{\tilde{N}-2} & \tilde{t}_{\tilde{N}-2}^2 & \dots & \tilde{t}_{\tilde{N}-2}^{\tilde{N}} \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 1 & 2\tilde{t}_{\tilde{N}-2} & \dots & \tilde{N}\tilde{t}_{\tilde{N}-2}^{\tilde{N}-1} \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ . \\ . \\ a_{\tilde{N}-2} \\ a_{\tilde{N}-1} \\ a_{\tilde{N}} \end{Bmatrix} = \begin{Bmatrix} \tilde{x}_i(t_0) \\ \tilde{x}_i(t_1) \\ \\ . \\ . \\ \tilde{x}_i(t_{\tilde{N}-2}) \\ \dot{\tilde{x}}_i(t_0) \\ \dot{\tilde{x}}_i(t_{\tilde{N}-2}) \end{Bmatrix}, \tag{3.20}$$

where $\dot{\tilde{x}}_i(t)$ is the velocity given in the ephemeris for $i = 1 \dots 3$, and the acceleration given by the same force model used to generate the ephemeris for $i = 4 \dots 6$.

Tables 3.12 and 3.13 show error ratios given by Zadunaisky's technique with the two-body force and the full force model, respectively. The polynomials used in the method match each point in the subinterval, and the derivatives of the polynomial match the force model at the end points of each subinterval, as described above. For the Runge-Kutta integrator a 5^{th} order polynomial is used, $\tilde{N} = 5$, and for the Gauss-Jackson a 3^{rd} order polynomial is used, $\tilde{N} = 3$. These polynomials have been found to give the best results. With higher-order polynomials, the technique gives error ratios that are too high, compared to the two-body test, and with lower order polynomials the error ratios are too low. In [37], Zadunaisky uses $\tilde{N} \geq p$, which is followed here for the Runge-Kutta integrator but not for the Gauss-Jackson integrator.

Table 3.12: Two-Body Zadunaisky Results

Test Case	Runge-Kutta		Gauss-Jackson	
	Position	Velocity	Position	Velocity
LEO	3.07×10^{-10}	3.07×10^{-10}	1.32×10^{-14}	1.32×10^{-14}
HEO	3.38×10^{-9}	7.27×10^{-9}	7.78×10^{-14}	1.71×10^{-13}
GEO	3.54×10^{-11}	3.54×10^{-11}	3.66×10^{-15}	3.34×10^{-15}

A comparison of Table 3.12 to the exact two-body error in Table 3.2 shows that the technique matches the true error ratios in order of magnitude for the LEO and GEO cases for Runge-Kutta, and the LEO case for Gauss-Jackson. For the eccentric orbit with Runge-Kutta the technique gives an error ratio that is an order of magnitude too high compared to the actual two-body error. In [34] Zadunaisky suggests a variant of

Table 3.13: Perturbed Zadunaisky Results

Test Case	Runge-Kutta		Gauss-Jackson	
	Position	Velocity	Position	Velocity
LEO	3.78×10^{-10}	3.78×10^{-10}	1.11×10^{-13}	1.11×10^{-13}
HEO	3.43×10^{-9}	7.36×10^{-9}	2.84×10^{-14}	6.21×10^{-14}
GEO	3.10×10^{-11}	3.09×10^{-11}	1.18×10^{-14}	1.27×10^{-14}

his technique that gives improved results for eccentric orbits, but that improvement has not been implemented here. For Gauss-Jackson the technique gives an error ratio that is three orders of magnitude too low for the HEO and GEO cases compared to the actual two-body error. A comparison of the perturbed results in Table 3.13 to the results from the step-size halving and high-order test in Tables 3.5 and 3.7 shows that the Runge-Kutta results with Zadunaisky's technique match the results from the other techniques, at least in order of magnitude. However the Gauss-Jackson results are several orders of magnitude lower for Zadunaisky's technique for all cases.

Note that for the method of choosing polynomials described above, the minimum order polynomial is $\tilde{N} = 3$, because that involves a subinterval of two points. The fact that the best results were found for Gauss-Jackson with this lowest order polynomial, and that this order violates Zadunaisky's criteria of $\tilde{N} \geq p$, indicates that this method of determining polynomials may not be appropriate for Gauss-Jackson, and may explain why the error ratios do not match the error ratios from the other tests.

Another technique for finding a polynomial to fit the ephemeris is available from ephemeris compression techniques. Ephemeris compression is a way to transmit satellite ephemeris without needing to send out a full ephemeris file, and without the receiver needing a full propagator. When space surveillance was done entirely with general perturbations, the space surveillance centers could simply send out initial conditions to users, and the users could generate ephemeris on-site with the same analytic propagator. With the centers now using special perturbations, the users would be required to have SP capability on-site if the centers were to only send out initial conditions. Special perturbations software is both complex and computationally expensive, so not all users have this capability. On the other hand, having the centers send out ephemeris generated at small intervals to all the users places an additional burden on the centers. Ephemeris compression offers a compromise, in which the centers send out coefficients from a fit of the ephemeris, and the users can then reconstruct the ephemeris.

Hoots and Segerman [38], presented a method of ephemeris compression called the Hybrid

Ephemeris Compression Model (HECM). The compression model consists of three stages. Given position ephemeris equally spaced in time, the method first converts the ephemeris to mean elements at each time step. These mean elements are then fit to a power series in time using a Chebyshev fit. This fit gives an approximation to the secular effects of perturbations on the elements. The second part of the compression models the periodic J_2 perturbation. The final part of the compression uses a Fourier series to model the remaining periodic effects of perturbations. The Fourier model was first presented by Segerman and Coffey[39]. At each time step, the power series representing the mean elements is evaluated, and the resulting elements are converted to osculating elements by accounting for the J_2 periodic effects. The difference between this estimated state and the original ephemeris is called residual ephemeris, which is modeled by the Fourier series. The residual ephemeris represents perturbation effects that are known to have two predominate periods: one orbit and one day. Therefore, a set of Fourier coefficients is found for each orbit, with the period in the Fourier series being the period of the argument of latitude. To further compress the total number of coefficients needed in the compression, a second Fourier fit is done on the Fourier coefficients themselves. This second fit uses one day as the period, so a set of coefficients is found for each day.

To reconstruct the ephemeris, the coefficients from the Chebyshev fit are used to compute mean elements at each time, and those mean elements are converted to osculating elements by accounting for the J_2 effect. This procedure gives an estimated state to which the residual ephemeris must be added. To reconstruct the residual ephemeris, the one-day Fourier coefficients are used to reconstruct the one-orbit Fourier coefficients. The one-orbit coefficients are then used to find the residual ephemeris at each time step. However, the Fourier series does not give a reliable fit near the endpoints of the orbit due to the Gibb's phenomenon, which causes error in Fourier fits near endpoints when the data is not exactly periodic. Instead of using the Fourier series at the endpoints, a quintic interpolating polynomial is used that passes through three points each on adjacent orbits. If the points in the region of an orbit endpoint are numbered 1 through 10, so that the orbit endpoint occurs between points 5 and 6, then the interpolating polynomial passes through the points reconstructed by the Fourier series at points 1, 2, 3, 8, 9, and 10. Using this interpolating polynomial prevents a discontinuity that would occur if only the Fourier series were used. To provide smoothness between the Fourier series and the polynomial, a cosine smoothing function is used which keeps the first and second derivative continuous, but not the third derivative. The smoothing function is used between points 1 and 2 and between points 9 and 10. For the first orbit, a prior orbit is unavailable so this technique cannot be used. Instead, a quartic polynomial is used that passes through the known epoch value as well as the values from the Fourier series. If the points closest to epoch are numbered 1 through 6, with epoch numbered

1, the polynomial passes through epoch at point 1, and the Fourier reconstruction at points 3 through 4. Again, a smoothing function is used between points 5 and 6. No consideration is made for the last orbit.

Ephemeris compression can be used in Zadunaisky's technique by taking two derivatives of the compression expressions for position. To simplify the process, the one-day period Fourier fit is not performed. This second fit is not needed because minimizing the number of coefficients is not an issue here. Also, the J_2 periodic correction is not made, so that the derivatives of this correction are not needed. Instead, the J_2 periodic effects are handled by the Fourier fit, by increasing the number of terms in the series. In [38], Hoots and Segerman used a maximum of 91 coefficients per day. Here, a maximum of 601 coefficients per day is used. The derivatives of the ephemeris compression expressions are given in Appendix A.

The results for the the LEO and HEO cases with the Runge-Kutta integrator with perturbations are shown in Table 3.14. These results, where a five-second step size is used, match the results from the step-size halving and higher-order test and indicate that using ephemeris compression with Zadunaisky's method works in theory. However, the results with the GEO case, using a one-minute step size, show the flaws of this method. The GEO case gives a position error ratio of 2.5×10^{-2} with perturbations, which is nine orders of magnitude too large. An investigation of the integration step-by-step shows that the integration error occurs when the first smoothing function is used. The smoothing function lasts one minute, which is the same as the integration step size. Integrating the same case with a five second step size does give reasonable results. This result indicates that the problem is stiff in this region, so only small step sizes can be used. The second derivative changes rapidly between the interpolating polynomial and the Fourier series, and the smoothing function is adequate to account for the change only when small steps are used. With Gauss-Jackson, the technique does not give reasonable results even with a five second step, most likely because of the discontinuity in the 3rd derivative of the smoothing function.

Table 3.14: Runge-Kutta Ephemeris-Compression Zadunaisky Results

Test Case	Position	Velocity
1	2.03×10^{-9}	2.04×10^{-9}
2	3.74×10^{-9}	8.27×10^{-9}

Using only the power series part of the fit, which is smooth, the method gives results that are two orders of magnitude too low. The power series does not fit the data well enough to give adequate results. Specifically, the power series does not model any of the

periodic effects, which contribute to integration error. So, use of ephemeris compression in Zadunaisky's technique at this time is limited to single-step methods with small step sizes. If a method of fitting the ephemeris with an expression that is more smooth becomes available, it is likely to give good results with Zadunaisky's technique.

With a function of time that closely fits the ephemeris available, one might try to integrate the second derivative of that function by itself, instead of as part of Zadunaisky's technique, and use the error of that integration as a measure of the error in the original problem. Table 3.15 shows results for the LEO and HEO cases with perturbations integrating only the compression expression, compared to the reconstruction of the ephemeris. These results are somewhat lower than the higher-order results for position, and significantly lower for velocity. Since the expression being integrated is only a function of time, the integration error at one step does not affect the next step. So this method only measures local error; global error does not build up. Position has a larger error than velocity because position is found by integrating velocity, so the velocity error does affect the position error.

Table 3.15: Integration of Ephemeris-Compression Derivative Results

Test Case	Position	Velocity
1	1.23×10^{-9}	8.24×10^{-12}
2	6.74×10^{-10}	5.91×10^{-11}

3.3.8 Summary

Because perturbations, including atmospheric drag, affect integration error, the two-body test and the integral invariant test can not be used to measure integration accuracy. However the two-body test has proved useful in assessing the effectiveness of the other techniques. In particular the two-body test has shown that the reverse test does not give a good measure of integration error. Though Zadunaisky's technique in theory gives a measure of integration error compared to an exact solution, the method has proven too difficult to implement. No methods are currently available to find a polynomial that is both sufficiently smooth and sufficiently represents the orbital motion. On the other hand, the step-size halving test and the higher-order test both give comparable results and measure the two-body error reasonably well. Because some of the integration methods used in subsequent tests are variable-step, the step-size halving test cannot be implemented. Therefore, the higher-order test is used. Though the higher-order test does not give an exact measure of integration error, it does give a reasonable approximation of

the error. Since the main purpose of the comparisons in Chapter 5 is to measure the speed of integration methods that have "about the same" accuracy, the higher-order test will suffice. Because the 14^{th}-order Gauss-Jackson method used in the higher-order test uses a relatively small step-size and has multiple evaluations per step, it has a significantly longer run-time than the methods being tested. So the test effectively measures how much accuracy is lost by using the faster methods.

3.4 Speed Tests

To measure integration speed, the integration methods are first adjusted to give equivalent accuracy. The step size, or other parameters such as integration tolerance, are adjusted to give an error ratio, with perturbations, of approximately 1×10^{-9} in a higher-order test covering 3 days. The computation time is then found in an integration covering 30 days with perturbations. Computation time is measured as the user time on an SGI Octane.

3.5 Evaluations

When comparing the computation time of integrators, the main factor determining the execution time is the number of force model evaluations performed. To demonstrate the effect that perturbations have on run-time, the highly elliptic case is integrated with the Gauss-Jackson integrator with a 30-second step size for 30 days with and without perturbations. The integration with perturbations takes 19.0 seconds, and the integration without perturbations takes 1.94 seconds of user time. So 90% of the run-time is spent evaluating the perturbations. These run-times include not only the integration but also other overhead associated with creating the output ephemeris file. With the Runge-Kutta method, which uses a smaller step size so the overhead is less significant, 93% of the run-time is spent evaluating the perturbations for this test case. The other test cases have similar percentages.

Because evaluations take up such a large percentage of run-time, the number of evaluations in an integration can be used as an estimate of speed, without actually performing speed tests. For instance, when comparing two integration methods to one another, if one is known to have twice as many evaluations, because it uses half the step size or has twice as many evaluations per step, then it can be assumed to have about twice the run-time of the other method. Furthermore, if two integrations have the same number

of evaluations, then they can be assumed to have equivalent run-times, and the method that is more accurate can be considered the more efficient method.

As an example, consider a test of the Gauss-Jackson method with various step sizes and various orders, and also varying the number of evaluations per step. Table 3.16 shows position error ratios for such a study for the low earth test case, and Table 3.17 shows results for the highly elliptical test case. The tables show results for both one evaluation per step (PEC), and two evaluations per step (PECEC). Results in between the horizontal lines have equivalent run-times. The * in the tables denotes that the integration has become unstable. It is obvious when instability has occurred, because the eccentricity of the integrated ephemeris grows to the point where the orbit becomes hyperbolic. High order multi-step integrators are known to be susceptible to instability at large step sizes ([40], pp. 139, 146).

Table 3.16: Gauss-Jackson Error Ratios for LEO Case

Step Size (sec)	Evals/ Step	Order 6	8	10	12	14
480	2	1.7×10^{-3}	2.9×10^{-3}	*	*	*
240	1	4.1×10^{-4}	*	*	*	*
240	2	4.9×10^{-5}	2.8×10^{-5}	3.0×10^{-5}	*	*
120	1	3.5×10^{-7}	5.2×10^{-7}	*	*	*
120	2	1.2×10^{-6}	1.1×10^{-6}	1.0×10^{-6}	1.3×10^{-6}	5.0×10^{-7}
60	1	3.7×10^{-8}	2.8×10^{-8}	1.1×10^{-8}	*	*
60	2	4.0×10^{-8}	2.9×10^{-8}	2.2×10^{-8}	1.6×10^{-8}	1.6×10^{-8}
30	1	2.9×10^{-9}	3.1×10^{-9}	3.2×10^{-9}	3.2×10^{-9}	*
30	2	2.9×10^{-9}	3.1×10^{-9}	3.2×10^{-9}	3.2×10^{-9}	3.2×10^{-9}
15	1	4.9×10^{-12}	2.6×10^{-12}	1.0×10^{-12}	2.3×10^{-12}	7.6×10^{-12}

The results in Table 3.16 and 3.17 indicate that cutting the step size in half gives a greater accuracy improvement than adding a second evaluation. However, for the higher-order methods, the second evaluation is necessary to maintain stability at larger step sizes. The greatest speed advantage is gained by using one evaluation per step with a lower-order method, to avoid instability. The tests presented in Chapter 5 involve the eighth-order Gauss-Jackson, with one evaluation per step. The step sizes used in the tests are sufficiently small to avoid instability.

These results support a claim made by Herrick [3] that the Gauss-Jackson integrator can run without the corrector. Though the corrector is applied once in a PEC implementation, without a second evaluation the results of the correction are not used in subsequent

Table 3.17: Gauss-Jackson Error Ratios for HEO Case

Step Size (sec)	Evals/ Step	Order 6	8	10	12	14
480	2	1.0×10^{-1}	1.1×10^{-1}	*	*	*
240	1	3.5×10^{-3}	1.7×10^{-2}	9.4×10^{-3}	*	*
240	2	4.6×10^{-3}	2.7×10^{-3}	6.1×10^{-4}	1.3×10^{-3}	3.4×10^{-5}
120	1	2.7×10^{-5}	2.0×10^{-5}	2.0×10^{-5}	6.0×10^{-5}	1.1×10^{-2}
120	2	1.2×10^{-5}	8.9×10^{-6}	1.5×10^{-5}	1.5×10^{-5}	1.7×10^{-5}
60	1	5.7×10^{-7}	5.8×10^{-7}	1.5×10^{-6}	1.8×10^{-6}	2.9×10^{-5}
60	2	7.3×10^{-7}	7.9×10^{-7}	7.2×10^{-7}	7.6×10^{-7}	5.1×10^{-7}
30	1	1.6×10^{-8}	1.1×10^{-8}	6.7×10^{-9}	8.3×10^{-9}	2.8×10^{-9}
30	2	1.7×10^{-8}	1.0×10^{-8}	7.5×10^{-9}	6.0×10^{-9}	5.6×10^{-9}
15	1	1.9×10^{-10}	9.2×10^{-11}	4.2×10^{-11}	4.6×10^{-11}	9.5×10^{-11}

steps. Therefore, PE and PEC implementations have the same stability properties, and have comparable results. Adding the corrector does improve the results somewhat over predictor-only implementations, so it is used here, because applying the corrector without a second evaluation has little effect on run-time.

With s-integration, the integration is always unstable with one evaluation per step (PEC) ([21], p. 184). A second evaluation (PECEC) provides stability, but the computation time associated with a second evaluation significantly limits the advantage of s-integration over t-integration discussed in Section 2.4. However, it is not necessary for the second evaluation to be a full evaluation to maintain stability. Because the difference between the predicted and corrected states is small, the difference in perturbation forces at the two states is small. Therefore, computation time can be saved by simply re-evaluating the two-body force during the second evaluation, and adding it to the perturbation force from the first evaluation. This second evaluation is called a **partial evaluation** ([41], p. 108) and denoted $\tilde{\text{E}}$, so the implementation is PEC$\tilde{\text{E}}$C.

Table 3.18 shows the maximum position difference over three days between using s-integration with the PECEC method and the PEC$\tilde{\text{E}}$C method for a variety of orbits. Each integration was performed with perturbations, with a 30-second step size at perigee. Though the PECEC method takes twice as long, it changes the result by less than 10 mm. The s-integration results in Chapter 5 are obtained using the PEC$\tilde{\text{E}}$C method.

Table 3.18: Effect of Partial Evaluation in s-integration.

Perigee Height (km)	e	Position Difference (mm)
300	0.25	1.96
300	0.50	2.12
300	0.75	2.30
500	0.25	1.91
500	0.50	2.17
500	0.75	2.27
1000	0.25	1.85
1000	0.50	2.05
1000	0.75	8.79

3.6 Summary

Several methods of testing the accuracy of integrators have been presented in this chapter. One of these methods, the higher-order test, has been chosen as the test to use for the testing in subsequent chapters. In addition, a method for testing the speed of integrators has been described, and used to illustrate the effect of evaluations on run-time. The effect of evaluations on stability has also been discussed. Further testing in the next chapter motivates the need for a variable-step double-integration multi-step integrator, which uses one evaluation per step. That method is also derived in the next chapter, and then in Chapter 5 it is compared to other integration methods.

Chapter 4

Variable-Step Störmer-Cowell Method

4.1 Motivation

The need for a variable-step double-integration method can be illustrated by comparing some of the integration methods described in Chapter 2. The benefit of variable-step integration is demonstrated by comparing the Shampine-Gordon method, and the s-integration method, to the Gauss-Jackson method. The benefit of double integration is shown by comparing the Adams method to the Störmer-Cowell method. Following this motivation the method is derived in Section 4.2, the implementation of the method is discussed in Section 4.3, and preliminary results are given in Section 4.4.

4.1.1 Variable Step

Variable-step integrators have an advantage over fixed-step integrators for elliptical orbits. The advantage of variable-step methods can be quantified using the speed testing procedure described in Section 3.4. The fixed-step Gauss-Jackson method is compared to the variable-step Shampine-Gordon method as well as s-integration, which uses analytic step regulation, in speed tests on test-case orbits of various eccentricities. The test cases all have a perigee height of 400 km, and an inclination of 40°. As in the testing in the previous chapter, the test cases have a ballistic coefficient of $0.01\,{}^{\mathrm{m}^2}/_{\mathrm{kg}}$, and an epoch of 1999-10-01 00:00:00 UT.

Table 4.1 shows the step sizes needed for the fixed-step method (t-integration) and s-

integration, and the relative tolerance needed by Shampine-Gordon, to achieve an error ratio of 1×10^{-9} in the higher-order test. An absolute tolerance of 1×10^{-31} is used for every case with the Shampine-Gordon integrator. The step size used for the 14^{th}-order Gauss-Jackson used as the reference in the higher-order test is chosen to give an error ratio of 1×10^{-10} or better in the step-size halving test. Table 4.1 also shows the time required to integrate 30 days with the step size or tolerance shown. Finally, the table shows a speed ratio, which is the time of the fixed-step method divided by the time of the variable-step method. The variable step methods have an advantage when the speed ratio is above one. Figure 4.1 shows a plot of the speed ratio against eccentricity for both variable-step methods.

Table 4.1: Speed Comparisons for Perigee Height of 400 km

	Step Size / Tolerance			Time for 30 Day Run (sec)			Speed Ratio to t	
e	t	s	SG	t	s	SG	s	SG
0	29	20	2×10^{-11}	20.6	32.1	89.9	0.64	0.23
0.05	36	34	7×10^{-11}	16.5	17.5	58.8	0.94	0.28
0.10	41	40	2×10^{-10}	14.5	13.9	42.4	1.0	0.34
0.15	39	36	1×10^{-10}	15.1	13.8	37.6	1.1	0.40
0.20	38	34	2×10^{-11}	15.3	13.2	40.1	1.2	0.38
0.25	38	34	7×10^{-11}	15.2	12.1	30.6	1.3	0.50
0.30	39	34	2×10^{-11}	14.7	11.0	32.8	1.3	0.45
0.40	35	32	1×10^{-11}	16.3	9.29	29.6	1.8	0.55
0.50	33	30	1×10^{-11}	17.2	7.69	23.8	2.2	0.72
0.55	35	25	7×10^{-12}	16.2	7.85	22.2	2.1	0.73
0.60	31	28	1×10^{-11}	18.3	6.03	18.2	3.0	1.0
0.65	31	27	2×10^{-11}	18.2	5.23	14.1	3.5	1.3
0.70	29	26	1×10^{-11}	19.4	4.41	13.0	4.4	1.5
0.80	26	23	1×10^{-11}	21.8	2.92	8.16	7.5	2.7
0.90	25	22	7×10^{-11}	22.5	1.24	2.89	18	7.8
0.95	25	22	2×10^{-10}	22.5	0.56	1.15	40	20

Figure 4.1 shows that s-integration is more efficient than fixed-step integration at an eccentricity of approximately 0.15, and that Shampine-Gordon is more efficient than Gauss-Jackson at an eccentricity of approximately 0.60. Test results at other perigee heights, given in Chapter 5, show that the eccentricity where the variable-step methods are more efficient is independent of the perigee height. The plot also shows that s-integration is always more efficient than Shampine-Gordon. This result is because s-integration only has one full evaluation and one partial evaluation, and Shampine-Gordon

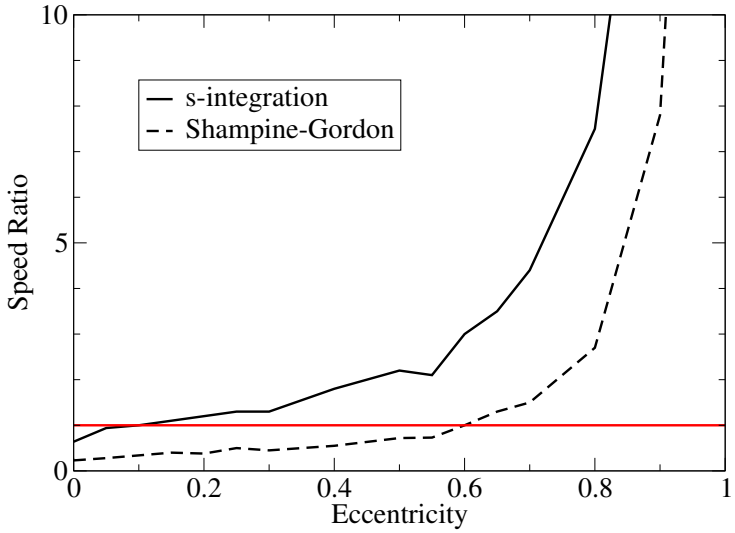


Figure 4.1: Speed Ratios to Fixed-Step Integration at 400 km Perigee

has two full evaluations. Though s-integration does provide a significant advantage for elliptical orbits, it still does not provide local error control, so there is no guarantee that it is meeting a specific accuracy requirement. Also, s-integration has the disadvantage of in-track error mentioned in Section 2.4.2. A variable-step integrator that only requires one evaluation per step would combine some advantages of both Shampine-Gordon and s-integration.

4.1.2 Double vs. Single Integration

Because two integrations are necessary to integrate a second-order differential equation, single integration has more round-off error than double integration. To examine the advantage of double integration over single integration, the double-integration Störmer-Cowell method is compared to the single-integration Adams method. These integrators are compared on test case orbits with varying eccentricity and perigee heights. Again, the test cases all have an inclination of 40°, a ballistic coefficient of $0.01\,{}^{\mathrm{m}^2}/_{\mathrm{kg}}$, and an epoch of 1999-10-01 00:00:00 UTC.

Table 4.2 gives error ratios over three days for the Störmer-Cowell and Adams integrators without perturbations. The table shows that the error ratio is always smaller with Störmer-Cowell. The reference value used to compute the error ratio in (3.4) is the analytic two-body solution. The integrators both use a 30 second step size in the tests, so have nearly identical run-times. The integrators are both set to use two evaluations

per step. The first evaluation is a full evaluation, and the second evaluation is a partial-evaluation.

Table 4.2: Error Ratios for Störmer-Cowell and Adams, Two Body

Height (km)	Eccentricity	Störmer-Cowell	Adams
300	0.00	2.47×10^{-13}	2.66×10^{-12}
300	0.25	3.05×10^{-12}	7.90×10^{-12}
300	0.50	1.28×10^{-11}	9.35×10^{-11}
300	0.75	4.01×10^{-11}	2.66×10^{-10}
500	0.00	3.49×10^{-13}	7.90×10^{-13}
500	0.25	2.87×10^{-12}	9.21×10^{-12}
500	0.50	7.94×10^{-12}	6.46×10^{-11}
500	0.75	2.21×10^{-11}	1.69×10^{-10}
1000	0.00	9.63×10^{-14}	4.78×10^{-12}
1000	0.25	3.53×10^{-13}	9.58×10^{-12}
1000	0.50	1.73×10^{-12}	2.40×10^{-11}
1000	0.75	9.70×10^{-12}	7.03×10^{-11}

Table 4.3 gives position error ratios over three days for the two integrators with perturbations. The table shows that the error ratio is always smaller with Störmer-Cowell, except for the 300 km and 500 km circular orbits. Again a step size of 30 seconds is used for both integrators, and two evaluations per step are performed, with the second one being a partial-evaluation. In the tests with perturbations the reference used in (3.4) is the 14^{th}-order Gauss-Jackson method. To ensure that the reference method is significantly more accurate than the methods being tested, a five-second step size is used with the 14^{th}-order method.

The results in Tables 4.2 and 4.3 show that Störmer-Cowell is generally more accurate than Adams for equivalent run-times. These results show the advantage of double integration over single integration. In the tests used to create these results, the integrators performed two evaluations per step. A further advantage of double integration is shown by removing the second evaluation. In a predict, evaluate, correct (PEC) implementation, the Adams method is unstable in all of the test cases, with and without perturbations. The Störmer-Cowell method is not unstable. Without the second evaluation Störmer-Cowell gives results that are only slightly less accurate than with two evaluations. This finding implies that double integration gives a stability advantage over single integration. Krogh ([24], p. 31) and Herrick ([3], p. 12) also pointed out that double integration can

Table 4.3: Error Ratios for Störmer-Cowell and Adams, Perturbations

Height (km)	Eccentricity	Störmer-Cowell	Adams
300	0.00	4.69×10^{-09}	3.45×10^{-09}
300	0.25	3.73×10^{-09}	7.31×10^{-09}
300	0.50	3.60×10^{-09}	1.31×10^{-08}
300	0.75	1.16×10^{-08}	4.04×10^{-08}
500	0.00	2.14×10^{-10}	1.11×10^{-10}
500	0.25	4.89×10^{-10}	1.41×10^{-09}
500	0.50	1.30×10^{-09}	3.82×10^{-09}
500	0.75	3.96×10^{-09}	1.13×10^{-08}
1000	0.00	3.33×10^{-12}	1.47×10^{-11}
1000	0.25	1.86×10^{-11}	5.49×10^{-11}
1000	0.50	5.65×10^{-11}	1.74×10^{-10}
1000	0.75	2.09×10^{-10}	6.27×10^{-10}

perform as well with one evaluation per step as single integration can with two evaluations per step. Krogh specifically pointed out the stability advantage of double integration in [42]. This stability advantage allows a variable-step double-integration method to require only one evaluation per step, giving it a significant advantage over Shampine-Gordon.

4.2 Derivation

To create an integrator with features that the motivation tests indicate would be useful, a variable-step double-integration method can be derived using the concepts in the derivation of Shampine-Gordon. However, the proposed implementation differs from Shampine-Gordon in several ways. Only one evaluation is performed per step, for a PEC implementation, which significantly reduces the run-time of the method. The method is not variable order, because a second evaluation would be necessary to estimate when the order should be increased. Finally, since the main goal is to reduce run-time, and because order increases that require a constant step are not being considered, fewer restrictions are placed on the factor ρ which changes the step size. The argument that keeping the step size constant reduces the overhead does not apply, because the force model used in orbit propagation is so expensive that the overhead associated with calculating the coefficients is insignificant.

4.2.1 Predictor

For satellite orbits, the governing second-order differential equation is

$$\ddot{\boldsymbol{r}} = \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}}). \tag{4.1}$$

To solve for position, $\boldsymbol{r}$, take the double integral of each side in (4.1),

$$\int_{t_n}^{t_{n+1}} \int_{t_n}^{\tilde{t}} \ddot{\boldsymbol{r}}(t)\, dt\, d\tilde{t} = \int_{t_n}^{t_{n+1}} \int_{t_n}^{\tilde{t}} \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}})\, dt\, d\tilde{t}. \tag{4.2}$$

Performing the integration on the left side of (4.2) gives

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n + h_{n+1}\dot{\boldsymbol{r}}_n + \int_{t_n}^{t_{n+1}} \int_{t_n}^{\tilde{t}} f(t, \boldsymbol{r}, \dot{\boldsymbol{r}})\, dt\, d\tilde{t}. \tag{4.3}$$

Proceeding with this form leads to the integration methods given by Krogh [24]. Lundberg also used this form, and called it the *general formulation* of double integration [41].

General Formulation

The general formulation predictor is found by replacing the function f with the interpolating polynomial $\boldsymbol{P}_{k,n}$,

$$\boldsymbol{r}_{n+1}^p = \boldsymbol{r}_n + h_{n+1}\dot{\boldsymbol{r}}_n + \int_{t_n}^{t_{n+1}} \int_{t_n}^{\tilde{t}} \boldsymbol{P}_{k,n}(t)\, dt\, d\tilde{t}. \tag{4.4}$$

Substituting (2.137) for $\boldsymbol{P}_{k,n}$, and changing the integration variable to τ, gives

$$\boldsymbol{r}_{n+1}^p = \boldsymbol{r}_n + h_{n+1}\dot{\boldsymbol{r}}_n + h_{n+1}^2 \sum_{i=1}^{k} \boldsymbol{\phi}_i^*(n) \int_0^1 \int_0^{\tilde{\tau}} c_{i,n}(\tau)\, d\tau\, d\tilde{\tau}. \tag{4.5}$$

The double integration of $c_{i,n}$ is given by the coefficients $g_{i,2}$ found for Shampine-Gordon, so the predictor simplifies,

$$\boldsymbol{r}_{n+1}^p = \boldsymbol{r}_n + h_{n+1}\dot{\boldsymbol{r}}_n + h_{n+1}^2 \sum_{i=1}^{k} g_{i,2}\boldsymbol{\phi}_i^*(n). \tag{4.6}$$

This expression is the predictor formula for the general formulation of double integration.

Variable-Step Störmer-Cowell

The general formulation contains a velocity term, $\dot{\boldsymbol{r}}_n$, which must be removed for a truly double integration formula. To remove the velocity term, take a step backward ([4], p. 290),

$$\int_{t_n}^{t_{n-1}} \int_{t_n}^{\tilde{t}} \ddot{\boldsymbol{r}}(t)\, dt\, d\tilde{t} = \int_{t_n}^{t_{n-1}} \int_{t_n}^{\tilde{t}} \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}})\, dt\, d\tilde{t}, \tag{4.7}$$

which leads to the relation

$$-\boldsymbol{r}_n = -\boldsymbol{r}_{n-1} - h_n \dot{\boldsymbol{r}}_n + \int_{t_n}^{t_{n-1}} \int_{t_n}^{\tilde{t}} \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}})\, dt\, d\tilde{t}. \tag{4.8}$$

The first derivative term is removed by adding (h_{n+1}/h_n) times (4.8) to (4.3),

$$\begin{aligned} \boldsymbol{r}_{n+1} = \left(1 + \frac{h_{n+1}}{h_n}\right) \boldsymbol{r}_n - \frac{h_{n+1}}{h_n} \boldsymbol{r}_{n-1} + \int_{t_n}^{t_{n+1}} \int_{t_n}^{\tilde{t}} \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}})\, dt\, d\tilde{t} \\ + \frac{h_{n+1}}{h_n} \int_{t_n}^{t_{n-1}} \int_{t_n}^{\tilde{t}} \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}})\, dt\, d\tilde{t}. \end{aligned} \tag{4.9}$$

This formulation represents the variable-step form of the Störmer-Cowell method. Again, the interpolating polynomial $\boldsymbol{P}_{k,n}$ is used in place of $\boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}})$ to give an expression for the predicted value of position, $\boldsymbol{r}^p_{n+1}$,

$$\begin{aligned} \boldsymbol{r}^p_{n+1} = \left(1 + \frac{h_{n+1}}{h_n}\right) \boldsymbol{r}_n - \frac{h_{n+1}}{h_n} \boldsymbol{r}_{n-1} + \int_{t_n}^{t_{n+1}} \int_{t_n}^{\tilde{t}} \boldsymbol{P}_{k,n}(t)\, dt\, d\tilde{t} \\ + \frac{h_{n+1}}{h_n} \int_{t_n}^{t_{n-1}} \int_{t_n}^{\tilde{t}} \boldsymbol{P}_{k,n}(t)\, dt\, d\tilde{t}. \end{aligned} \tag{4.10}$$

Substituting (2.137) for $\boldsymbol{P}_{k,n}$ gives

$$\begin{aligned} \boldsymbol{r}^p_{n+1} = \left(1 + \frac{h_{n+1}}{h_n}\right) \boldsymbol{r}_n - \frac{h_{n+1}}{h_n} \boldsymbol{r}_{n-1} + \int_{t_n}^{t_{n+1}} \int_{t_n}^{\tilde{t}} \sum_{i=1}^{k} c_{i,n}(\tau) \boldsymbol{\phi}^*_i(n)\, dt\, d\tilde{t} \\ + \frac{h_{n+1}}{h_n} \int_{t_n}^{t_{n-1}} \int_{t_n}^{\tilde{t}} \sum_{i=1}^{k} c_{i,n}(\tau) \boldsymbol{\phi}^*_i(n)\, dt\, d\tilde{t}. \end{aligned} \tag{4.11}$$

The integration variable can be changed to τ by noting that $\tau = -h_n/h_{n+1}$ when $t = t_{n-1}$, for the upper limit of the second integration term,

$$\begin{aligned} \boldsymbol{r}^p_{n+1} = \left(1 + \frac{h_{n+1}}{h_n}\right) \boldsymbol{r}_n - \frac{h_{n+1}}{h_n} \boldsymbol{r}_{n-1} + h^2_{n+1} \sum_{i=1}^{k} \boldsymbol{\phi}^*_i(n) \int_0^1 \int_0^{\tilde{\tau}} c_{i,n}(\tau)\, d\tau\, d\tilde{\tau} \\ + \frac{h^3_{n+1}}{h_n} \sum_{i=1}^{k} \boldsymbol{\phi}^*_i(n) \int_0^{\frac{-h_n}{h_{n+1}}} \int_0^{\tilde{\tau}} c_{i,n}(\tau)\, d\tau\, d\tilde{\tau}. \end{aligned} \tag{4.12}$$

This expression can be written with the simplified notation $c_{i,n}^{(-q)}$ using (2.146),

$$\begin{aligned}\boldsymbol{r}_{n+1}^{p} = &\left(1+\frac{h_{n+1}}{h_n}\right)\boldsymbol{r}_n - \frac{h_{n+1}}{h_n}\boldsymbol{r}_{n-1} + h_{n+1}^2\sum_{i=1}^{k}\boldsymbol{\phi}_i^*(n)c_{i,n}^{(-2)}(1)\\ &+\frac{h_{n+1}^3}{h_n}\sum_{i=1}^{k}\boldsymbol{\phi}_i^*(n)c_{i,n}^{(-2)}\left(\frac{-h_n}{h_{n+1}}\right).\end{aligned} \tag{4.13}$$

A new set of coefficients, $g'_{i,q}$, is needed to find the second integration term. Define $g'_{i,q}$ as

$$g'_{i,q} = (q-1)!\,c_{i,n}^{(-q)}\left(\frac{-h_n}{h_{n+1}}\right). \tag{4.14}$$

Using (2.148), for $i \geq 3$, (4.14) is

$$\begin{aligned}g'_{i,q} = &\left(\alpha_{i-1}(n+1)\frac{-h_n}{h_{n+1}} + \frac{\psi_{i-2}(n)}{\psi_{i-1}(n+1)}\right)(q-1)!\,c_{i-1,n}^{(-q)}\left(\frac{-h_n}{h_{n+1}}\right)\\ &-\alpha_{i-1}(n+1)q!\,c_{i-1,n}^{(-q-1)}\left(\frac{-h_n}{h_{n+1}}\right).\end{aligned} \tag{4.15}$$

Using the definition of α, (2.135), and noting that $-h_n + \psi_{i-2}(n) = \psi_{i-3}(n-1)$, the expression simplifies,

$$g'_{i,q} = \frac{\psi_{i-3}(n-1)}{\psi_{i-1}(n+1)}g'_{i-1,q} - \alpha_{i-1}(n+1)g'_{i-1,q+1}, \tag{4.16}$$

for $i \geq 3$. To implement a recursive formula for g', the expressions for $i = 1$ and $i = 2$ are needed. When $i = 1$, change the limit of integration in (2.152),

$$g'_{1,q} = (q-1)!\int_0^{\frac{-h_n}{h_{n+1}}}\cdots\int_0^{\tau_1} d\tau_0\ldots d\tau_{q-1} = \frac{(q-1)!}{q!}\left(\frac{-h_n}{h_{n+1}}\right)^q = \frac{1}{q}\left(\frac{-h_n}{h_{n+1}}\right)^q. \tag{4.17}$$

For $i = 2$, change the integration limit in (2.153),

$$\begin{aligned}g'_{2,q} &= (q-1)!\int_0^{\frac{-h_n}{h_{n+1}}}\cdots\int_0^{\tau_1}\tau\,d\tau_0\ldots d\tau_{q-1} = \frac{(q-1)!}{(q+1)!}\left(\frac{-h_n}{h_{n+1}}\right)^{q+1}\\ &= \frac{1}{q(q+1)}\left(\frac{-h_n}{h_{n+1}}\right)^{q+1}.\end{aligned} \tag{4.18}$$

Now a recursive formula for $g'_{i,q}$ is available, similar to (2.154) for $g_{i,q}$,

$$g'_{i,q} = \begin{cases}\dfrac{1}{q}\left(\dfrac{-h_n}{h_{n+1}}\right)^q & i = 1,\\ \dfrac{1}{q(q+1)}\left(\dfrac{-h_n}{h_{n+1}}\right)^{q+1} & i = 2,\\ \dfrac{\psi_{i-3}(n-1)}{\psi_{i-1}(n+1)}g'_{i-1,q} - \alpha_{i-1}(n+1)g'_{i-1,q+1} & i \geq 3.\end{cases} \tag{4.19}$$

Note that for $i = 3$, $\psi_{i-3}(n-1) = 0$. When the step size is constant, $\psi_i(n) = ih$ and $\alpha_i(n+1) = 1/i$, so the formula reduces to

$$g'_{i,q} = \begin{cases} \dfrac{1}{q}(-1)^q & i = 1, \\ \dfrac{1}{q(q+1)}(-1)^{q+1} & i = 2, \\ \dfrac{i-3}{i-1} g'_{i-1,q} - \dfrac{1}{i} g'_{i-1,q+1} & i \geq 3. \end{cases} \tag{4.20}$$

The coefficients g'_{iq} for a constant step are shown in Table 4.4. Adding the second row of the table, $g'_{i,2}$, to the coefficients $g_{i,2}$ given in Table 2.9, gives the Störmer-Cowell coefficients λ_{i-1} (2.60).

Table 4.4: Coefficients g'_{iq} for Constant Step Size

		1	2	3	4	5
	1	-1	1/2	1/12	1/24	19/720
	2	1/2	-1/6	-1/24	-1/45	
q	3	-1/3	1/12	1/40		
	4	1/4	-1/20			
	5	-1/5				

Using the coefficients g and g', the double-integration predictor formula (4.13) can be written

$$\boldsymbol{r}^p_{n+1} = \left(1 + \frac{h_{n+1}}{h_n}\right)\boldsymbol{r}_n - \frac{h_{n+1}}{h_n}\boldsymbol{r}_{n-1} + h^2_{n+1}\sum_{i=1}^{k} g_{i,2}\boldsymbol{\phi}^*_i(n) + \frac{h^3_{n+1}}{h_n}\sum_{i=1}^{k} g'_{i,2}\boldsymbol{\phi}^*_i(n), \tag{4.21}$$

or, combining the terms,

$$\boldsymbol{r}^p_{n+1} = \left(1 + \frac{h_{n+1}}{h_n}\right)\boldsymbol{r}_n - \frac{h_{n+1}}{h_n}\boldsymbol{r}_{n-1} + h^2_{n+1}\sum_{i=1}^{k}\left(g_{i,2} + \frac{h_{n+1}}{h_n}g'_{i,2}\right)\boldsymbol{\phi}^*_i(n). \tag{4.22}$$

This expression is the predictor formula for the variable-step Störmer-Cowell method.

4.2.2 Corrector

As in single integration, the corrector uses the interpolating polynomial $\boldsymbol{P}^*_{k+1,n}$, given in (2.156). The corrected value at point $(n+1)$, $\boldsymbol{r}_{n+1}$, is found by replacing $\boldsymbol{P}_{k,n}$ with

$\boldsymbol{P}^*_{k+1,n}$ in (4.10),

$$\boldsymbol{r}_{n+1} = \left(1 + \frac{h_{n+1}}{h_n}\right)\boldsymbol{r}_n - \frac{h_{n+1}}{h_n}\boldsymbol{r}_{n-1} + \int_{t_n}^{t_{n+1}}\int_{t_n}^{\tilde{t}} \left[\boldsymbol{P}_{k,n}(t) + c_{k+1,n}(\tau)\boldsymbol{\phi}^p_{k+1}(n+1)\right] dt\, d\tilde{t}$$
$$+ \frac{h_{n+1}}{h_n}\int_{t_n}^{t_{n-1}}\int_{t_n}^{\tilde{t}} \left[\boldsymbol{P}_{k,n}(t) + c_{k+1,n}(\tau)\boldsymbol{\phi}^p_{k+1}(n+1)\right] dt\, d\tilde{t}. \tag{4.23}$$

This expression is simplified by combining the terms known to be $\boldsymbol{r}^p_{n+1}$ and by changing the integration variable to τ,

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}^p_{n+1} + h^2_{n+1}\int_0^1\int_0^{\tilde{\tau}} c_{k+1,n}(\tau)\boldsymbol{\phi}^p_{k+1}(n+1)\, d\tau\, d\tilde{\tau}$$
$$+ \frac{h^3_{n+1}}{h_n}\int_0^{\frac{-h_n}{h_{n+1}}}\int_0^{\tilde{\tau}} c_{k+1,n}(\tau)\boldsymbol{\phi}^p_{k+1}(n+1)\, d\tau\, d\tilde{\tau}. \tag{4.24}$$

The integrals can be written in terms of the coefficients g and g',

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}^p_{n+1} + h^2_{n+1}\left(g_{k+1,2} + \frac{h_{n+1}}{h_n}g'_{k+1,2}\right)\boldsymbol{\phi}^p_{k+1}(n+1), \tag{4.25}$$

giving a corrector formula for the variable-step Störmer-Cowell method. To reduce runtime, a second evaluation is not performed, so only a PEC implementation is used. Results show this implementation to be stable. Without the second evaluation the differences used by the predictor in the next step are simply $\boldsymbol{\phi}_i(n+1) = \boldsymbol{\phi}^p_i(n+1)$.

4.2.3 Interpolation

To interpolate to a requested point, an integration is performed from the last integration point to the requested point, similar to (2.170), but with the additional backwards integration to remove the velocity term,

$$\boldsymbol{r}_{\text{out}} = \left(1 + \frac{h_I}{h_{n+1}}\right)\boldsymbol{r}_{n+1} - \frac{h_I}{h_{n+1}}\boldsymbol{r}_n + \int_{t_{n+1}}^{t_{\text{out}}}\int_{t_{n+1}}^{\tilde{t}} \boldsymbol{P}_{k+1,n+1}(t)\, dt\, d\tilde{t}$$
$$+ \frac{h_I}{h_{n+1}}\int_{t_{n+1}}^{t_n}\int_{t_{n+1}}^{\tilde{t}} \boldsymbol{P}_{k+1,n+1}(t)\, dt\, d\tilde{t}. \tag{4.26}$$

The expression is simplified using the notation $c^{I(-q)}_{i,n+1}$, given in (2.173),

$$\boldsymbol{r}_{\text{out}} = \left(1 + \frac{h_I}{h_{n+1}}\right)\boldsymbol{r}_{n+1} - \frac{h_I}{h_{n+1}}\boldsymbol{r}_n + h^2_I\sum_{i=1}^{k+1}\boldsymbol{\phi}_i(n+1)c^{I(-2)}_{i,n+1}(1)$$
$$+ \frac{h^3_I}{h_{n+1}}\sum_{i=1}^{k+1}\boldsymbol{\phi}_i(n+1)c^{I(-2)}_{i,n+1}\left(\frac{-h_{n+1}}{h_I}\right). \tag{4.27}$$

The coefficients $g^I_{2,i}$ can be used for the first integration term. Another set of coefficients, $g^{I'}$ is needed for the second term. These coefficients are defined

$$g^{I'}_{i,q} = (q-1)!\, c^{I(-q)}_{i,n+1}\left(\frac{-h_{n+1}}{h_I}\right). \tag{4.28}$$

Following (2.173), a recursive formula for these coefficients is available,

$$g^{I'}_{i,q} = \begin{cases} \dfrac{1}{q}\left(\dfrac{-h_{n+1}}{h_I}\right) & i = 1, \\ \Gamma_{i-1}\left(\dfrac{-h_{n+1}}{h_I}\right) g^{I'}_{i-1,q} - \dfrac{h_I}{\psi_{i-1}(n+1)} g^{I'}_{i-1,q+1} & i \geq 2. \end{cases} \tag{4.29}$$

The value of Γ in this expression can be simplified following (2.169),

$$\Gamma_i\left(\frac{-h_{n+1}}{h_I}\right) = \begin{cases} -1 & i = 1, \\ \dfrac{\psi_{i-2}(n)}{\psi_i(n+1)} & i \geq 2. \end{cases} \tag{4.30}$$

Note that for $i = 2$, $\psi_{i-2}(n+1) = 0$. With these coefficients defined, an expression for $\boldsymbol{r}_{\text{out}}$ is available,

$$\boldsymbol{r}_{\text{out}} = \left(1 + \frac{h_I}{h_{n+1}}\right)\boldsymbol{r}_{n+1} - \frac{h_I}{h_{n+1}}\boldsymbol{r}_n + h_I^2 \sum_{i=1}^{k+1}\left(g^I_{i,2} + \frac{h_I}{h_{n+1}} g^{I'}_{i,2}\right)\boldsymbol{\phi}_i(n+1). \tag{4.31}$$

This expression is the interpolation formula for double integration.

4.2.4 Step-Size Control

The step size is controlled by estimating the local position error at each step, similarly to (2.178) for velocity. For double integration, the value of $\boldsymbol{r}_{n+1}(k)$ is found by replacing $\boldsymbol{P}^*_{k+1,n}(t)$ with $\boldsymbol{P}^*_{k,n}(t)$, (2.179), in (4.23),

$$\begin{aligned} \boldsymbol{r}_{n+1}(k) = \boldsymbol{r}^p_{n+1} &+ h^2_{n+1}\int_0^1 \int_0^{\tilde{\tau}} c_{k,n}(\tau)\boldsymbol{\phi}^p_{k+1}(n+1)\, d\tau\, d\tilde{\tau} \\ &+ \frac{h^3_{n+1}}{h_n}\int_0^{\frac{-h_n}{h_{n+1}}} \int_0^{\tilde{\tau}} c_{k+1,n}(\tau)\boldsymbol{\phi}^p_k(n+1)\, d\tau\, d\tilde{\tau}. \end{aligned} \tag{4.32}$$

The integrals may be written in terms of the coefficients g and g',

$$\boldsymbol{r}_{n+1}(k) = \boldsymbol{r}^p_{n+1} + h^2_{n+1}\left(g_{k,2} + \frac{h_{n+1}}{h_n} g'_{k,2}\right)\boldsymbol{\phi}^p_{k+1}(n+1). \tag{4.33}$$

The local error vector is estimated by subtracting (4.33) from (4.25),

$$\boldsymbol{le}_{n+1}(k) \approx h_{n+1}^2 \left(g_{k+1,2} - g_{k,2} + \frac{h_{n+1}}{h_n}(g'_{k+1,2} - g'_{k,2}) \right) \boldsymbol{\phi}_{k+1}^p(n+1). \tag{4.34}$$

The method determines if a step fails based on this local estimate; the step fails if

$$\sqrt{\sum_{L=1}^{3} \left(\frac{le_L}{\mathrm{WT}_\mathrm{D}(L)} \right)^2} \leq \mathrm{EPS}, \tag{4.35}$$

where $\mathrm{WT}_\mathrm{D}(L)$ is a weight for double integration,

$$\mathrm{WT}_\mathrm{D}(L) = |\boldsymbol{r}_L| \frac{\epsilon_\mathrm{rel}}{\mathrm{EPS}} + \frac{\epsilon_\mathrm{abs}}{\mathrm{EPS}}, \tag{4.36}$$

similar to (2.191) for single integration. If the step fails the step is tried again with half the step size. Like the Shampine-Gordon integrator, the method restarts as first order if a step fails after three tries. This restart is needed with double integration, because if the step size becomes too small compared to the previous step, the term $(-h_n/h_{n+1})^q$ in the g' coefficients (4.19) becomes large, contributing to numerical error.

To choose the step size at the next step, $h_{n+2} = \rho h_{n+1}$, the local error at that step is approximated, analogous to (2.185),

$$\boldsymbol{le}_{n+2}(k) \approx h_{n+2}^2 \left(g_{k+1,2} - g_{k,2} + \frac{h_{n+2}}{h_{n+1}}(g'_{k+1,2} - g'_{k,2}) \right) \boldsymbol{\phi}_{k+1}^p(n+2). \tag{4.37}$$

As with single integration, this expression is approximated assuming that the differences are slowly varying and the previous steps were also taken at ρh_{n+1}, analogous to (2.189),

$$\boldsymbol{le}_{n+2}(k) \approx \rho^2 h_{n+1}^2 (\lambda_k - \lambda_{k-1}) \rho^k \sigma_{k+1}(n+1) \boldsymbol{\phi}_{k+1}^p(n+1), \tag{4.38}$$

where λ_i are the Störmer-Cowell predictor coefficients given in Section 2.3.5. Introducing $\lambda_k^* = \lambda_k - \lambda_{k-1}$ simplifies the expression,

$$\boldsymbol{le}_{n+2}(k) \approx \rho^{k+2} h_{n+1}^2 \lambda_k^* \sigma_{k+1}(n+1) \boldsymbol{\phi}_{k+1}^p(n+1). \tag{4.39}$$

To calculate the value of ρ, the error using a step size of h_{n+1} is found, as in (2.191),

$$\mathrm{ERK}_\mathrm{D} = |h_{n+1}^2 \lambda_k^* \sigma_{k+1}(n+1)| \sqrt{\sum_{L=1}^{3} \left(\frac{\phi_{L\,k+1}^p(n+1)}{\mathrm{WT}_\mathrm{D}(L)} \right)^2}. \tag{4.40}$$

Using a chicken factor of 0.5, the factor ρ for the next step is found similarly to (2.193),

$$\rho = \left(\frac{0.5\epsilon}{\mathrm{ERK}_\mathrm{D}} \right)^{\frac{1}{k+2}}. \tag{4.41}$$

For stability, the calculated value of ρ is bounded between 0.5 and 2. However, no other restrictions are placed on r, so that unlike Shampine-Gordon step-size increases between 1 and 2 can be made. This technique allows the step size to increase as soon as possible, which reduces overall run-time. Because of the expensive force model it is better to make a small increase in the step size and recompute the coefficients than to leave the step size constant to avoid recomputing the coefficients.

4.2.5 Initialization

To start the method, a variable-order implementation is used. The method starts as first-order, and at each step the order is increased, and the step size is doubled. During this startup phase a second evaluation is performed for stability. The order is increased until nine backpoints are used, at which point the startup phase ends and the normal predict, evaluate, correct cycle begins. Nine backpoints are used in the normal procedure because this corresponds to an eighth-order multi-step method. The testing in Section 3.5 showed the eighth-order Gauss-Jackson method to have good stability characteristics with one evaluation per step, even with relatively large step sizes.

The initial step size is chosen by a method similar to the method used by the Shampine-Gordon method. For double integration, the equivalent of (2.198), is

$$h_1 = \frac{1}{4}\sqrt{\frac{\text{EPS}}{\sqrt{\sum_{L=1}^{3}\left(\frac{\dot{\boldsymbol{r}}_{L\,0}}{\text{WT}_\text{D}(L)}\right)^2}}}. \tag{4.42}$$

Again, this step size is bounded by $4\epsilon_m t_0$, and the size of the step to the first output point. The error in the first step is a lower bound on the error of the entire method, so additional procedures are used to find the best value of the initial step. If the first step fails the accuracy check, the step size is reduced by one half and the method tries again. If the step does not fail, the step size is doubled and the initial step is tried again. The doubling is repeated on each successful try until the step fails, and then the step size is reduced back to the last successful value. This procedure biases the initial step toward a large value. Initial steps that are too small can cause a significant amount of round-off error, which this procedure avoids.

At the first step, when $n = 0$, only the initial values of position and velocity are known. Therefore, the variable-step Störmer-Cowell formula for position, (4.22), cannot be used, because $\boldsymbol{r}_{-1}$ would be needed. Instead the general formulation, (4.6), is used for the first

step. At the first step $k = 1$, so the predictor can be written,

$$\boldsymbol{r}_1^p = \boldsymbol{r}_0 + h_1\dot{\boldsymbol{r}}_0 + \frac{1}{2}h_1^2\ddot{\boldsymbol{r}}_0, \tag{4.43}$$

and the corrector is

$$\boldsymbol{r}_1 = \boldsymbol{r}_1^p + \frac{1}{6}h_1^2\boldsymbol{\phi_2}(1). \tag{4.44}$$

Though the general formulation is undesirable because the presence of the velocity term contributes to round-off error, at the first step the velocity is an initial condition, and so by definition contains no round-off error. Since the general formulation is used at the first step, a different equation is needed to check the accuracy for position,

$$\frac{1}{3}h_1^2\sqrt{\sum_{L=1}^{3}\left(\frac{\phi_{L2}(2)}{\mathrm{WT_D}(L)}\right)} < \mathrm{EPS}. \tag{4.45}$$

The accuracy check for velocity at the first step is given by (2.182).

4.3 Implementation

The double-integration variable-step integrator is implemented with a Shampine-Gordon style single-integration integrator to solve the differential equation $\ddot{\boldsymbol{r}}(t) = \boldsymbol{f}(t, \boldsymbol{r}, \dot{\boldsymbol{r}})$. The double-integration integrator is used to find $\boldsymbol{r}$ and the single-integration integrator is used to find $\dot{\boldsymbol{r}}$. The integrators are implemented together to use the same step size, which is the smaller of the two step sizes given by their respective step-size control algorithms. The single-integration integrator differs from Shampine-Gordon in that the order of the method remains fixed, the step size is allowed to change by a factor between 0.9 and 2, and only one evaluation is performed per step, just as with the double-integration integrator.

The method must first be started as first order. For the first step the general formulation is used. The procedure for this step is:

1. The initial step size for double integration, (4.42), and single integration, (2.198), are calculated.
2. The step size is set to the lower of the two values, and bounded between $4\epsilon_m t_0$ and the size of the step to the first output point.
3. The predicted position, (4.43), and velocity, (2.155), are calculated.

4. The function is evaluated at the predicted point.
5. The new differences $\boldsymbol{\phi_i(1)}$ are calculated (2.162).
6. The corrected values of position, (4.44), and velocity, (2.161), are found.
7. The error is estimated for $\boldsymbol{r}_1$, (4.45), and $\dot{\boldsymbol{r}}_1$, (2.182).
8. If the error is below the tolerance, the step size is halved, and the procedure returns to step 3.
9. If the error is above the tolerance, and no previous tries have failed, the step size is doubled, and the procedure returns to step 3.
10. The function is re-evaluated at the corrected point, and the differences are recomputed.
11. The number of backpoints is incremented to $k = 2$.
12. The value of ρ is set to 2.

After the first step has been taken, the variable-step Störmer-Cowell is used. At each step $(n + 1)$,

13. The new step size is calculated, $h_{n+1} = \rho h_n$.
14. The values of $\psi_i(n + 1)$, $\psi_i(n)$, and $\psi_i(n - 1)$ are calculated, (2.124).
15. The values of $\alpha_i(n + 1)$ are calculated, (2.135).
16. The coefficients are calculated, g, (2.154), and g', (4.19).
17. The values of β_i are calculated, (2.130), and the values of $\boldsymbol{\phi}_i^*$, (2.129).
18. The predicted values $\boldsymbol{r}_{n+1}^p$, (4.22), and $\dot{\boldsymbol{r}}_{n+1}^p$, (2.155), are found.
19. The function is evaluated at the predicted point.
20. The new differences $\boldsymbol{\phi}_i(n + 1)$ are calculated, (2.162).
21. The corrected values of $\boldsymbol{r}_{n+1}$, (4.25), and $\dot{\boldsymbol{r}}_{n+1}$, (2.161), are found.
22. The error is estimated for $\boldsymbol{r}$, (4.34), and $\dot{\boldsymbol{r}}$, (2.181).

23. If either error estimate is above the tolerance (4.35), (2.182), the step fails, the differences are reset, ρ is set to 0.5, and the procedure returns to step 13. If there are three consecutive failures, the method restarts at step 1.

24. If the number of backpoints k is equal to the maximum of 9:

    (a) The value of ERK is calculated for double integration, (4.40), and single integration, (2.191).

    (b) The factor ρ recommended for double integration, (4.41), and single integration, (2.193) is calculated.

    (c) The factor ρ is set to the lower of the two recommended values, and bounded between 0.5 and 2.

    Otherwise the method is still in the startup phase:

    (a) A second evaluation at the corrected point is performed, and the differences are recalculated, (2.162).

    (b) The number of backpoints k is incremented.

    (c) The factor ρ is set to 2 to double the step size.

25. The step n is incremented and the procedure returns to step 13.

4.4 Results

Two separate implementations show the integrator to be effective. The first implementation is in Matlab. The Matlab implementation is used to integrate the second order differential equation $y'' = -y$, with initial conditions $y(0) = 0$, $y'(0) = 1$, over $0 \leq t \leq 10\pi$. The exact solution of this problem is $y(t) = \sin(t)$. Because this problem has no dependence on y', only the double-integration method has been implemented. An absolute tolerance of $\epsilon_{\text{abs}} = 1 \times 10^{-14}$ is used in the integration. Figure 4.2 shows a plot of the numerical solution, the step sizes used, and the total error at each step, $|y_n - \sin(t_n)|$.

Figure 4.2 shows that the step size fluctuates periodically between approximately 0.1 and 0.15. There is a slight offset between the peaks of the solution and the peaks of the step-size curve. The error curve also behaves periodically, with peaks of the error correlating to peaks of the solution. The error grows as the integration progresses, which is expected. The maximum error is 2.68×10^{-12}. The values plotted at in Figure 4.2 are the actual integration points, so no interpolation is used. Figure 4.3 shows the solution,

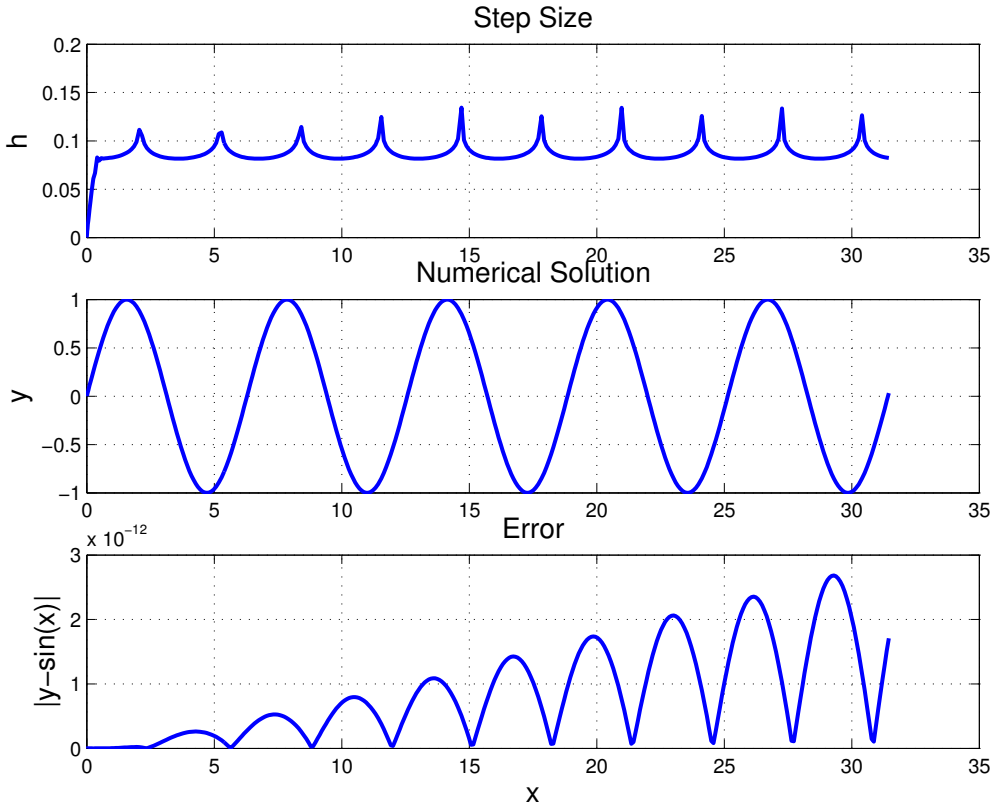


Figure 4.2: Results of Integrating $y'' = -y$

and error, of interpolated values in increments of t of 0.1. The error for these interpolated values is on the same order as the error of the integration itself. The Matlab code used to create Figures 4.2 and 4.3 is given in Appendix B.

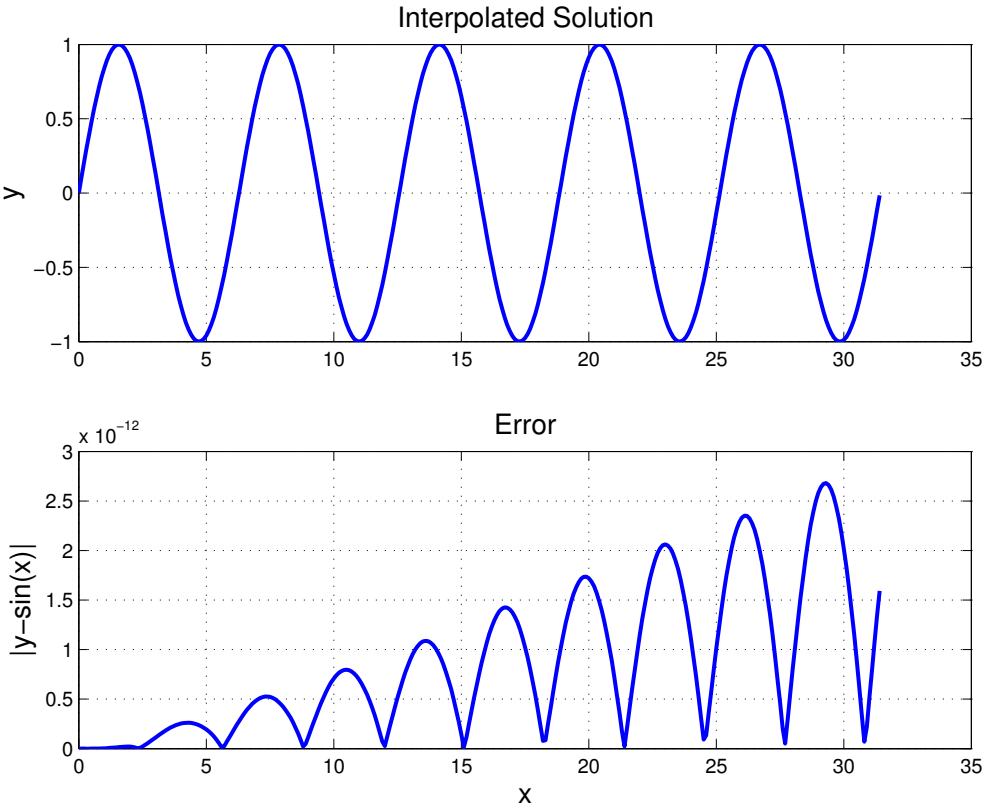


Figure 4.3: Interpolation Results for Integrating $y'' = -y$

The integration method has also been implemented in Fortran to test using the method for orbit propagation. The full implementation of both the variable-step double and single-integration integrators described in the Section 4.3 has been implemented into Special-K [28]. Sample Fortran code is given in Appendix C. Table 4.2 shows error ratios in a two-body test of various test case orbits. A relative tolerance of $\epsilon_r = 1 \times 10^{-12}$ and an absolute tolerance of $\epsilon_a = 1 \times 10^{-13}$ are used in the tests. Canonical units are used in the integration, so the position and velocity are on the order of one. Using an absolute tolerance an order of magnitude lower than the relative tolerance assures that the relative tolerance usually dominates the weighting factors given by (2.184) and (4.36). However, when one of the components of position or velocity approaches zero, the absolute tolerance prevents the values of ERK_S, (2.191), and ERK_D, (4.40), from becoming too large. With a smaller absolute tolerance steps would unnecessarily fail when components approach

zero, costing run-time.

Table 4.5: Variable-Step Double-Integration Results for the Two-Body Problem

Height (km)	Eccentricity	Error Ratio
300	0.00	3.18×10^{-10}
300	0.25	4.90×10^{-11}
300	0.50	1.80×10^{-10}
300	0.75	1.85×10^{-10}
500	0.00	3.46×10^{-10}
500	0.25	2.59×10^{-10}
500	0.50	6.68×10^{-11}
500	0.75	1.94×10^{-10}
1000	0.00	2.39×10^{-10}
1000	0.25	1.69×10^{-10}
1000	0.50	2.12×10^{-10}
1000	0.75	8.90×10^{-11}

These preliminary results show the integrator to be effective. Additional testing comparing the method to other integration methods is shown in the next chapter.

Chapter 5

Comparisons

5.1 Introduction

Many studies have compared the use of different methods to integrate satellite orbits. Merson [21] tested different integration methods on a variety of types of orbits, and found that the eighth-order Gauss-Jackson method, with only the predictor, to be most efficient for circular orbits. Merson recommended the Gauss-Jackson method with s-integration for highly elliptical orbits. However, Merson's report was prepared in 1974, before variable-step multi-step methods such as Shampine-Gordon were available.

Fox [29] compared the fixed-step Gauss-Jackson method, Gauss-Jackson with s-integration, a variable-step variable-order Adams method, and several variable-step single-step methods for orbits with various eccentricities. Fox used the two-body problem in his tests so an exact measure of error was available. To simulate the effects of complex perturbations on computation time, Fox added unnecessary calculations to the evaluation to waste time. Like Merson, Fox found the Gauss-Jackson method to be best for low eccentricities. Fox found that for elliptical orbits with complicated force models s-integration is the best method, because it requires fewer evaluations than the single-step methods. The main drawback of Fox's tests is that the effect of perturbations on integration error was not considered.

More recently, Montenbruck [30] performed a study comparing several single-step and multi-step methods, with both fixed and variable steps. Again, his study did not include perturbations. His tests indicated that some of the more recently developed single-step methods are competitive with multi-step methods in some cases.

A set of tests performed by Lundberg [41] compared integrators similar to the methods

considered here. Lundberg's study compared eight integrators:

- The general formulation in backward difference form (fixed step, multi-step, double integration)
- The Gauss-Jackson method in backward difference form (fixed step, multi-step, double integration)
- The Gauss-Jackson method in ordinate from (fixed step, multi-step, double integration)
- The variable-order Shampine-Gordon method (variable step, multi-step, single integration)
- The variable-order Krogh integrator (variable step, multi-step, double integration)
- The non-summed Adams method (fixed step, multi-step, single integration)
- A $7(8)^{\text{th}}$-order Runge-Kutta-Fehlberg integrator (variable step, single step, single integration)
- A $7(8)^{\text{th}}$-order Runge-Kutta-Nyström integrator (variable step, single step, double integration)

The multi-step integrators all used a PECE implementation in the tests. Lundberg compared the methods on five sets of differential equations that have exact periodic solutions: an harmonic oscillator, the circular problem of two bodies, the elliptical problem of two bodies, the Euler rigid-body problem, and the restricted three-body problem. The circular and elliptical problems of two bodies involved two bodies of equal mass orbiting one another, so are not the same as the two-body problem for earth-orbiting satellites. Lundberg also compared the integrators for a near circular earth-orbiting satellite with and without perturbations. For the case without perturbations he used the analytic solution as the reference, and for the case with perturbations the reference was an integration performed with the $7(8)^{\text{th}}$-order Runge-Kutta-Fehlberg with a higher precision and lower tolerance than the method used in the tests. This testing procedure is similar in nature to the higher-order test. The case with perturbations considered only the spherical 11^{th}-degree geopotential effects.

Lundberg's study examined both the number of function evaluations and the total computation time, so he was able to draw conclusions about the overhead cost of the integrators. For instance, integrators that have fewer evaluations but longer run-times than another

method, when the evaluation is not expensive, have a greater overhead cost. Lundberg concluded that the variable-step multi-step methods have a higher overhead cost than the variable-step single-step methods. However, both the fixed and variable-step multi-step methods were more efficient than the single-step methods in terms of function evaluations. In the elliptical problem, the fixed-step multi-step methods require less overhead than the variable-step methods, but require more evaluations than the variable-step multi-step methods. However, the fixed-step multi-step methods still require fewer evaluations than the variable-step single-step methods to achieve a certain accuracy. In comparing the double integration methods to the single integration methods, Lundberg found the double integration methods to be more accurate for a given step size and order. Lundberg also concluded that there was no significant difference between the difference and ordinate forms of the Gauss-Jackson method.

Though Lundberg's tests demonstrate many of the differences between these methods, the tests do not reveal the best integrator to use for satellite orbits with a full force model in all cases. Lundberg found the fixed-step methods to be more efficient for satellite orbits, but he only examined circular orbits. His study did not reveal the benefit of using variable-step methods for elliptical satellite orbits. Also, Lundberg's tests did not include drag, which has a significant effect on integration error.

In this Chapter, tests are performed comparing the variable-step Shampine-Gordon and Störmer-Cowell methods, as well as s-integration, to the fixed-step Gauss-Jackson method. Orbit propagation tests are performed at various eccentricities to show where the variable-step methods have an advantage over the fixed step method. Though Merson and Fox both recommended variable-step methods for elliptical orbits and fixed-step methods for circular orbits, they do not indicate when to switch between the two methods. The tests are also performed at different perigee heights to show the effect that drag has on the results. Following the orbit propagation tests, orbit determination tests are performed to indicate how using the variable-step methods for orbits where they are appropriate can speed up the catalog maintenance process.

5.2 Orbit Propagation

Figures 5.1 – 5.4 show plots of speed ratio against eccentricity for perigee heights of 300 km, 400 km, 500 km, and 1000 km, respectively. Plots for s-integration, Shampine-Gordon, and the variable-step Störmer-Cowell method are all shown on each figure. The horizontal line on the plots represent a speed ratio of one, above which the variable-step methods are more efficient than the fixed-step methods. Tables 5.1 – 5.4 show the times

from the 30-day propagation used to compute the speed ratios. The speed ratio is the time of the fixed-step Gauss-Jackson divided by the time of the variable-step methods. The 300 km circular orbit decays in less than 30 days, so the times shown for that case are the times to integrate 20 days. The tables also show the step sizes and tolerances needed for the methods to give a position error ratio, given by (3.6), of 1×10^{-9} against the 14^{th}-order Gauss-Jackson method. To create the reference values, the 14^{th}-order is set to give an error ratio of 1×10^{-10} in the step-size halving test.

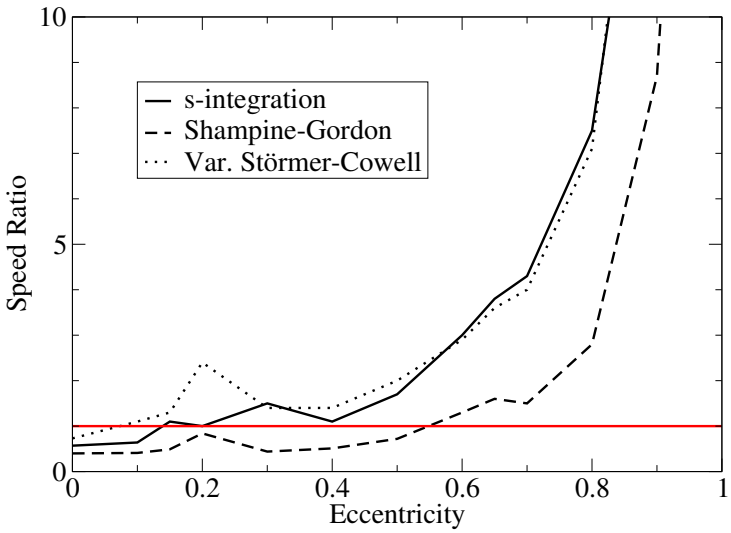


Figure 5.1: Speed Ratios to t-integration at 300 km Perigee

Figure 5.1 shows that the variable-step Störmer-Cowell method has an advantage over s-integration for 300 km perigee orbits with eccentricities below 0.55. This result indicates that the variable-step Störmer-Cowell method handles the high-drag cases better than s-integration. Controlling the step size by local-error control allows the method to use appropriately small steps near perigee where drag is a factor, and still use large enough steps at apogee to decrease run-time. With analytic step regulation, s-integration cannot account for the fact that there is drag near perigee. For s-integration to give accurate results near perigee at the low altitude, it must use a step that is smaller than necessary at apogee. So this test case demonstrates the advantage of local error control over analytic step regulation. The results at the higher perigee heights indicate that s-integration does vary the step size appropriately when drag is a less significant factor. Figure 2.3(c) shows that s-integration does take smaller steps, in arc-length, at perigee than at apogee. The comparison between the variable-step Störmer-Cowell method and s-integration indicates that these smaller steps are appropriate to account for the increased geopotential perturbation forces near perigee, but are not small enough to handle high drag cases.

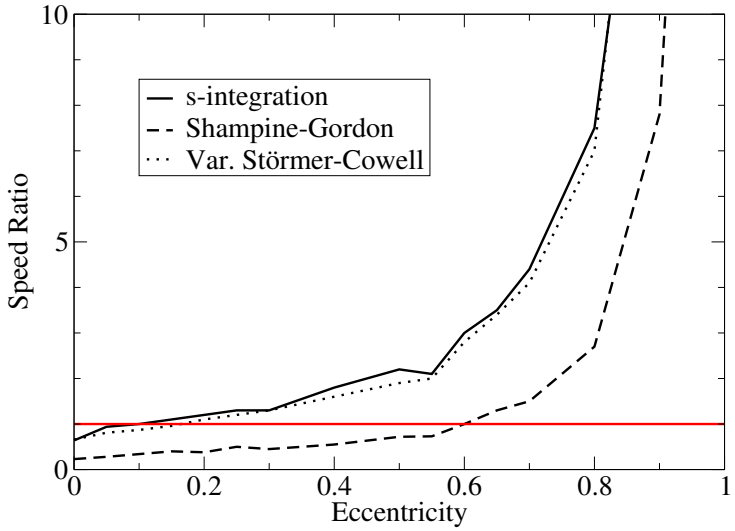


Figure 5.2: Speed Ratios to t-integration at 400 km Perigee

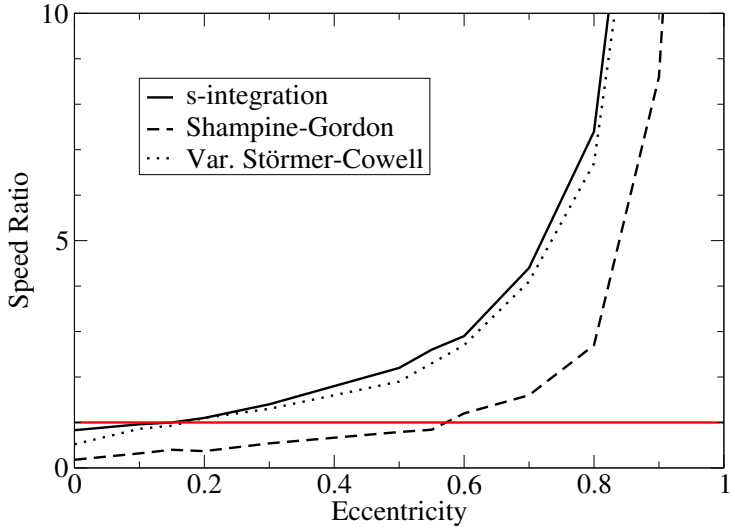


Figure 5.3: Speed Ratios to t-integration at 500 km Perigee

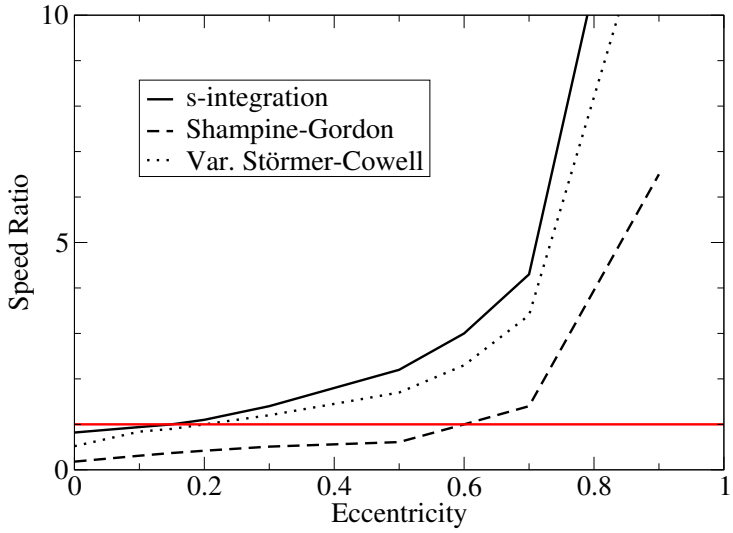


Figure 5.4: Speed Ratios to t-integration at 1000 km Perigee

Table 5.1: Comparisons for Perigee Height of 300 km

	Step Size / Tolerance				Time for 30 Day Run (sec)			
e	t	s	SG	vSC	t	s	SG	vSC
0	12	7	5×10^{-12}	3×10^{-12}	33.3	58.7	82.6	45.5
0.10	26	15	4×10^{-11}	5×10^{-11}	22.9	35.6	55.3	20.6
0.15	26	23	4×10^{-11}	5×10^{-11}	22.7	21.4	46.0	17.3
0.20	14	11	1×10^{-11}	2×10^{-11}	41.3	39.5	49.2	16.9
0.30	32	30	1×10^{-11}	4×10^{-11}	18.0	12.2	40.6	13.0
0.40	35	20	1×10^{-11}	2.5×10^{-11}	16.3	14.5	31.9	11.6
0.50	31	21	1×10^{-11}	3×10^{-11}	18.4	10.8	25.7	9.32
0.60	28	25	7×10^{-11}	6×10^{-11}	20.3	6.71	15.2	6.91
0.65	26	24	4×10^{-11}	4.5×10^{-11}	22.0	5.86	13.8	6.10
0.70	28	24	1×10^{-11}	6×10^{-11}	20.2	4.74	13.5	4.99
0.80	26	23	2×10^{-11}	8×10^{-11}	21.8	2.92	7.76	3.09
0.90	22	18	1×10^{-10}	1×10^{-10}	25.6	1.53	2.93	1.37
0.95	23	18	2×10^{-10}	1.5×10^{-10}	24.4	0.65	1.21	0.62

Table 5.2: Comparisons for Perigee Height of 400 km

	Step Size / Tolerance				Time for 30 Day Run (sec)			
e	t	s	SG	vSC	t	s	SG	vSC
0	29	20	2×10^{-11}	2×10^{-12}	20.6	32.1	89.9	31.4
0.05	36	34	7×10^{-11}	8×10^{-11}	16.5	17.5	58.8	20.4
0.10	41	40	2×10^{-10}	2×10^{-10}	14.5	13.9	42.4	16.7
0.15	39	36	1×10^{-10}	6×10^{-11}	15.1	13.8	37.6	15.8
0.20	38	34	2×10^{-11}	7×10^{-11}	15.3	13.2	40.1	13.7
0.25	38	34	7×10^{-11}	5×10^{-11}	15.2	12.1	30.6	12.7
0.30	39	34	2×10^{-11}	6×10^{-11}	14.7	11.0	32.8	11.2
0.40	35	32	1×10^{-11}	4×10^{-11}	16.3	9.29	29.6	10.3
0.50	33	30	1×10^{-11}	3×10^{-11}	17.2	7.69	23.8	8.83
0.55	35	25	7×10^{-12}	2×10^{-11}	16.2	7.85	22.2	8.20
0.60	31	28	1×10^{-11}	6.5×10^{-11}	18.3	6.03	18.2	6.47
0.65	31	27	2×10^{-11}	8×10^{-11}	18.2	5.23	14.1	5.42
0.70	29	26	1×10^{-11}	6×10^{-11}	19.4	4.41	13.0	4.75
0.80	26	23	1×10^{-11}	8×10^{-11}	21.8	2.92	8.16	3.13
0.90	25	22	7×10^{-11}	1.5×10^{-10}	22.5	1.24	2.89	1.21
0.95	25	22	2×10^{-10}	1×10^{-10}	22.5	0.56	1.15	0.54

Table 5.3: Comparisons for Perigee Height of 500 km

	Step Size / Tolerance				Time for 30 Day Run (sec)			
e	t	s	SG	vSC	t	s	SG	vSC
0	45	40	5×10^{-11}	1×10^{-10}	13.3	16.1	75.0	25.8
0.10	44	39	8×10^{-11}	1.5×10^{-10}	13.7	14.2	43.3	15.9
0.15	42	37	1×10^{-10}	5×10^{-11}	14.0	13.4	35.2	15.1
0.20	42	37	2×10^{-11}	1×10^{-10}	13.8	12.3	37.6	12.7
0.30	40	35	6×10^{-11}	7.5×10^{-11}	14.4	10.6	26.8	10.9
0.50	35	32	2×10^{-11}	2.5×10^{-11}	16.2	7.23	20.6	8.61
0.55	34	31	1×10^{-11}	5×10^{-11}	16.7	6.49	19.9	7.24
0.60	34	30	4×10^{-11}	6.5×10^{-11}	16.7	5.71	14.5	6.25
0.70	31	28	2×10^{-11}	1×10^{-10}	18.2	4.12	11.4	4.42
0.80	29	26	2×10^{-11}	9×10^{-11}	19.4	2.61	7.16	2.89
0.90	26	24	2×10^{-10}	1×10^{-10}	21.7	1.15	2.51	1.27
0.95	27	24	5×10^{-10}	8×10^{-11}	20.9	0.53	1.05	0.59

Table 5.4: Comparisons for Perigee Height of 1000 km

	Step Size / Tolerance				Time for 30 Day Run (sec)			
e	t	s	SG	vSC	t	s	SG	vSC
0	68	60	1×10^{-10}	2×10^{-10}	8.83	10.8	49.1	16.9
0.10	65	56	1×10^{-10}	2×10^{-10}	9.19	9.78	29.9	11.0
0.15	62	55	1×10^{-10}	1×10^{-10}	9.40	9.04	25.7	10.4
0.20	60	53	1×10^{-10}	1.5×10^{-10}	9.65	8.57	23.1	9.33
0.30	57	50	1×10^{-10}	1×10^{-10}	10.1	7.48	19.8	8.41
0.50	51	46	1×10^{-11}	6×10^{-11}	11.2	5.08	18.3	6.59
0.60	48	44	5×10^{-11}	7×10^{-11}	11.8	3.93	11.7	5.21
0.70	46	41	3×10^{-11}	1.5×10^{-10}	12.3	2.88	8.98	3.63
0.90	38	34	1×10^{-10}	1×10^{-10}	14.9	0.87	2.30	1.14

Figures 5.2 – 5.4 show that at higher perigees the variable-step Störmer-Cowell method and s-integration have roughly the same speed, though the variable-step Störmer-Cowell method is somewhat slower. The additional overhead associated with calculating the integration coefficients may account for the difference between the run-times of the two methods. The variable-step Störmer-Cowell method and s-integration are always faster than the Shampine-Gordon integrator. The Shampine-Gordon method requires two evaluations per step, while the other two methods use only one full evaluation per step, so the Shampine-Gordon method should have twice the run-time as the other methods. However, the results shown in Tables 5.1 – 5.4 indicate that the Shampine-Gordon integrator has more than double the run-time as the other variable-step methods. The Shampine-Gordon method is biased toward keeping the step size constant, only increasing it when the step size can be doubled. This limitation accounts for the additional run-time of the Shampine-Gordon method beyond what is expected from the additional evaluation.

Figures 5.1 – 5.4 show that s-integration has an advantage over t-integration for orbits with eccentricities over approximately 0.15, and the Shampine-Gordon integrator is more efficient than the Gauss-Jackson method with t-integration for eccentricities over approximately 0.60. The figures indicate that these results are independent of perigee height. The variable-step Störmer-Cowell method also has an advantage over the Gauss-Jackson method for eccentricities over approximately 0.15 for perigee heights of 400 km and higher. For 300 km perigee height orbits, the variable-step Störmer-Cowell method has an advantage for eccentricities over approximately 0.10.

5.3 Orbit Determination

Orbit determination testing is performed on a test set of catalogued satellites for 1999-09-29. There are 8003 objects in the catalog, of which 1000 are randomly selected for testing. The goal of the test is to find the improvement in computation time by using the variable-step methods where the orbit propagation tests show they are more efficient, as well as to validate that the variable step methods give comparable results to the fixed-step Gauss-Jackson.

The initial vectors from the catalog are fit using differential correction with a fitspan between 1.5 and 10 days. The fitspan, which is the time span from which observations are used in the fit, is determined by operational algorithms and depends on the mean motion and rate of change of mean motion. The fit includes observations up to 1999-10-01 00:00:00, going back through the length of the fitspan. Before the fit, the initial vector is propagated forward to the time of the last observation. The fit solves for position and velocity at the time of the last observation, and also solves for the ballistic coefficient when the perigee height is below 1200 km.

The fit is performed with a batch least-squares differential correction process. The differential correction process propagates the epoch state, at the time of the last observation, backwards to the time of each observation. At each observation time a residual is calculated, which is the difference between the actual observation and the observation computed from the propagation. These residuals are then used to correct the value of the state at epoch,

$$\delta \boldsymbol{x} = \left(A^{\mathsf{T}} W A\right)^{-1} A^{\mathsf{T}} W \tilde{\boldsymbol{b}}, \tag{5.1}$$

where $\delta \boldsymbol{x}$ is the correction to the epoch state vector, $\tilde{\boldsymbol{b}}$ are the residuals, A is a matrix of partial derivatives, and W is a weighting matrix. The A matrix contains the partial derivatives of the observations with respect to the components of the epoch state vector. The W matrix weights observations based on how well the sensors are known to perform. After the state is updated the process is repeated with residuals given by the updated state. This process repeats through several iterations until the epoch state converges. At each step the weighted RMS is calculated,

$$RMS = \sqrt{\frac{\tilde{\boldsymbol{b}}^{\mathsf{T}} W \tilde{\boldsymbol{b}}}{N-1}}, \tag{5.2}$$

where N is the number of observations. The process is converged when the percent change in weighted RMS from one iteration to the next is less than 1%, or if the change in weighted RMS is less than 1×10^{-5}. The weighted RMS is a unitless value, because the W matrix normalizes the residuals.

To get a baseline for computation time in the tests, the time is found to fit the 1000 test objects using Gauss-Jackson with t-integration. The test is performed on a 450 MHz Pentium II machine running Linux. The total user time is 11.2 hours. Of the 1000 objects, 913 update, while 87 do not update because they fail some criteria, such as having a final RMS that is too large, or not having enough observations for the object.

To test s-integration, the objects with eccentricities above 0.15 are fit with Gauss-Jackson using both t-integration and s-integration. The time using t-integration is 2.28 hrs, and the time using s-integration is 0.65 hrs. So s-integration is 3.5 times faster than t-integration for processing only the objects with eccentricities above 0.15. Operational algorithms are used to choose the step sizes for the methods, with both t-integration and s-integration having the same step at perigee. Objects with eccentricities over 0.25 use a step size of 30 seconds, and low-earth objects with lower eccentricities use a step size of 1 minute. Objects at higher altitudes use larger steps. There are 136 objects with eccentricities above 0.15, of which 102 update and 34 do not. Comparing the final states given by t- and s-integration, 71 of the objects have a final position difference of less than 1 m. The remaining 31 objects are shown in Table 5.5, which gives the final position difference in meters, and the difference in final weighted RMS. A negative value for RMS difference in the table indicates that the s-integration has a lower final weighted RMS.

The objects with the largest position difference in Table 5.5, 19622 and 21589, have a lower weighted RMS with s-integration, indicating that s-integration gives a better fit. The object with the next highest position difference, 3827, converged on a different iteration in the differential correction, and accepted a different number of observations. Though this position difference is relatively large, it is still within the accuracy of the observations and the force model. The remaining position differences are relatively minor, and well within the accuracy of the observations.

Using s-integration to perform the fits saves 1.63 hrs over t-integration. In the entire set of 1000 objects, if t-integration is used to fit the objects with eccentricities below 0.15 and s-integration is used to fit objects with eccentricities above 0.15, the total computation time would be 9.57 hrs, which is a 14.6% savings over using only t-integration.

To test the variable-step Störmer-Cowell integrator, the method is also used to fit objects with eccentricities over 0.15. A relative tolerance of 2×10^{-11} is used in the fits. One of the objects that did not update under t-integration or s-integration, object 21538, is removed from the set for the variable-step Störmer-Cowell method, because the variable-step algorithm uses a step size that is prohibitively small for the object. Under t-integration and s-integration the integrations become unstable for this object, generating an error that causes the differential correction to stop, so the object is not

Table 5.5: Orbit Determination Differences for t- vs. s-integration

Satellite Number	Position Difference (m)	RMS Difference (s-int − t-int)
3827	112.778	0.0007
10960	2.70625	0.0000
13970	4.65988	-0.0002
19622	378.819	-0.0912
19884	1.06457	0.0000
19994	7.05259	-0.0007
19998	1.13009	-0.0004
21589	264.48	-0.0555
21591	4.22768	0.0008
21709	9.7989	0.0000
22020	13.9651	0.0000
22098	18.3051	0.0000
22238	7.41498	0.0000
22633	10.5869	0.0000
22997	1.71619	0.0007
23229	5.68604	0.0000
23332	15.019	0.0000
23403	7.89305	0.0000
23430	9.05374	0.0000
23460	15.5898	0.0000
23523	7.00024	0.0000
23616	4.74221	-0.0001
23950	3.42605	0.0000
24211	1.82044	-0.0001
24293	10.1841	0.0000
24655	14.9491	0.0000
24764	17.3145	-0.0001
25503	4.20061	0.0000
25542	18.3253	0.0000
25552	2.91824	0.0000
25805	2.00252	0.0000

updated. Instead of becoming unstable, the variable-step method finds a step size small enough to perform the integration, but the step size is on the order of one millisecond, so the object can not update in a reasonable amount of time. This problem is not necessarily a flaw in the method, since the other methods cannot handle this object either. However, additional error handling to prevent situations like this one from occurring are necessary before the variable-step Störmer-Cowell method can be used operationally.

Of the remaining 135 objects with eccentricities over 0.15, 105 of the objects update with the variable-step Störmer-Cowell method, while 30 do not. Three objects that failed to update with t-integration and s-integration, 14136, 18719 and 25539, are able to update with the variable-step method. Characteristics of these objects, and their reason for failing in t-integration, are show in Table 5.6. Two of the objects have perigee heights below 300 km, where drag is significant. The other object has a higher perigee height of 380 km, but with ballistic coefficient $0.43^{\mathrm{m}^2}/_{\mathrm{kg}}$ can still be considered a high-drag case. These results indicate that the variable-step method is able to handle drag better than the other methods. With local error control the variable-step method can take smaller steps when drag is a factor. Of the 102 objects that update under both t-integration and the variable-step Störmer-Cowell method, 85 have final position differences under 1 m. The remaining 17 objects are shown in Table 5.7. A negative value for RMS difference indicates the fit is better with the variable-step method.

Table 5.6: Objects Updated by var. Störmer-Cowell But Not t-integration

Satellite	Reason for Failing in t-integration	e	Perigee Height (km)	Ballistic Coefficient ($^{\mathrm{m}^2}/_{\mathrm{kg}}$)
14136	Final B-term too large	0.72	272	0.0018
18719	Final RMS too large	0.51	161	0.045
25539	Final B-term too large	0.72	382	0.43

The object with the largest position difference, 23950, has a larger final weighted RMS with the variable-step method. This object has only 25 observations within its fitspan. The differential correction converges on the 16^{th} iteration for this object under the variable-step method. With t-integration the differential correction rejects one observation after the 15^{th} iteration, and continues for six more iterations considering only 24 observations. So the difference between the fits with t-integration and with the variable-step Störmer-Cowell method for this object can be accounted for by the low number of observations for this object. All of the remaining objects with position differences greater than 10 m have lower final RMS values with the variable-step Störmer-Cowell method.

The variable-step Störmer-Cowell method takes 0.63 hours to process the objects, so it

Table 5.7: Orbit Determination Differences for t-integration vs. var. Störmer-Cowell

Satellite Number	Position Difference (m)	RMS Difference (vSC − t-int)
5977	5.07898	0.0001
8195	1.4376	0.0000
9911	1.11593	0.0000
10946	1.01612	0.0011
10960	3.03786	0.0002
11007	231.851	-0.0407
14131	52.4079	-0.0317
19622	380.928	-0.0909
19807	1.56643	0.0000
19994	10.3783	-0.0011
21589	259.809	-0.0545
21590	30.3005	-0.2210
22020	1.48802	0.0000
22997	2.87322	0.0010
23177	1.20855	-0.0054
23824	115.035	-0.0896
23950	466.252	0.6853

is 1.65 hours faster than t-integration. The method is 3.6 times faster than the Gauss-Jackson method with t-integration for processing only the objects with eccentricities above 0.15. Using the the variable-step Störmer-Cowell method for objects over 0.15 eccentricity and Gauss-Jackson with t-integration for the remaining objects would take 9.55 hours to process all 1000 objects, a 14.7% savings over only using t-integration for all of the objects. This savings is comparable to the savings with s-integration.

To test Shampine-Gordon, the objects with eccentricities above 0.6 are fit with both Gauss-Jackson using t-integration and Shampine-Gordon. A relative tolerance of 1×10^{-11} is used for the Shampine-Gordon integrator in the fits. Gauss-Jackson takes 1.64 hrs to process the 87 objects, while Shampine-Gordon takes 0.85 hrs. So Shampine-Gordon is 1.9 times faster then Gauss-Jackson for these objects. The Gauss-Jackson method updates 67 of the objects, while the Shampine-Gordon integrator updates 66 of the objects. The Shampine-Gordon method fails to update object 10946 because the final RMS is too large, though the object updates with Gauss-Jackson. The final position differences after the fit between the two integrators is less than 1 m for 55 of the objects, the remaining 15 are shown in Table 5.8. A negative RMS difference in the table indicates that the weighted RMS is lower with Shampine-Gordon.

Table 5.8: Orbit Determination Differences for t-integration vs. Shampine-Gordon

Satellite Number	Position Difference (m)	RMS Difference (SG − t-int)
8195	2.15318	-0.0001
9911	2.47439	0.0000
12992	1.57563	0.0007
13999	1.28456	-0.0002
17078	1.56477	0.0000
19622	1.87689	0.0480
19884	1.3129	-0.0003
20649	1.2846	-0.0001
21589	264.477	-0.0577
22020	1.01124	0.0000
22068	1.65849	0.0002
22633	1.79357	-0.0001
22997	2.53152	0.0009
23824	115.065	-0.0877
24655	36.3626	0.5819

Again the objects with the largest position differences, 21589 and 23824, have lower weighted RMS values with Shampine-Gordon. The remaining objects have position differences that are relatively small, and well within the accuracy of the observations.

Using Shampine-Gordon on eccentricities over 0.60 saves 0.79 hrs over t-integration. If the entire set of 1000 objects is fit with Gauss-Jackson using t-integration for eccentricities below 0.60 and with Shampine-Gordon for eccentricities above 0.60, the total computation time would be 10.41 hrs, which is a 7.0% savings over using only Gauss-Jackson with t-integration.

5.4 Summary

Two sets of tests are presented in this chapter. The orbit propagation tests compare the speed of variable-step and fixed-step integration methods with the integrators tuned to give equivalent accuracy. Those tests show that s-integration and the variable-step Störmer-Cowell method have comparable results in most cases, and that both methods have an advantage over the fixed-step Gauss-Jackson method for objects with eccentricities greater than 0.15. The variable-step Störmer-Cowell method has an advantage over s-integration for low-perigee orbits where drag is significant. The Shampine-Gordon integrator is slower than s-integration and the variable-step Störmer-Cowell, and has an advantage over the fixed-step method for eccentricities greater than 0.60.

Orbit determination tests also show the advantage of variable-step integration. The tests are designed to show the speed advantage that would be gained by using the variable-step methods for elliptical orbits and fixed-step methods for near-circular orbits, compared to using fixed-step methods for all objects. The tests show that using the variable-step Störmer-Cowell method for objects with eccentricities above 0.15, and the fixed-step Gauss-Jackson for the other objects gives a 14.7% speed advantage over using the fixed-step Gauss-Jackson for all objects. Using s-integration for objects with eccentricities greater than 0.15 has a comparable advantage, 14.6%. Using the Shampine-Gordon integrator for objects with eccentricities above 0.60 has an advantage of 7.0% over using the fixed-step method for all objects. The tests show that in most cases using a different integrator to perform the orbit determination does not have a significant effect on the final result. However, the variable-step Störmer-Cowell method is able to update more objects, because the local error control allows the method to take small steps when needed. Other cases where the different methods produce relatively large final position differences can be attributed to the accuracy and sparseness of the observations.

Chapter 6

Conclusions And Suggestions for Future Work

6.1 Summary

A new method of numerical integration, the variable-step Störmer-Cowell method, is derived. A study of currently available integration methods indicates that this new method has features desirable for elliptical orbits, and for orbits experiencing significant atmospheric drag. The method is variable-step with error control, so larger step sizes can be taken when possible, and the method is double-integration, so only one evaluation per step is necessary. The method uses a variable-order implementation for initialization, so it is self-starting. However, the method is not variable-order beyond the initialization phase, because variable-order algorithms require a second evaluation, which is considered too time-consuming for the application intended. This method is designed for space surveillance applications, which require numerical integrators that can efficiently handle a complex force model for a variety of orbit types. The best integrator for a given orbit is the one that has the fastest run-time while maintaining a given accuracy requirement.

In addition to deriving numerical integration methods, techniques for assessing the accuracy of numerical integrators are discussed. The two-body test gives an exact measure of error when perturbations are not considered. This exact measure of error is useful to evaluate the other techniques, by testing them without perturbations. However, when these other techniques are used with perturbations, they show a significantly larger error than the two-body test when drag is a factor. The step-size halving test and the higher-order test give results that are consistent with one another, and match the two-body error well. The reverse test gives results that are inconsistent, and previous authors

have shown it to be unreliable. Zadunaisky's technique also gives inconsistent results. Zadunaisky's technique requires a fit of the ephemeris that is both accurate and smooth, which is difficult for earth-orbiting satellites over several days.

Several of the integrators are compared to one another in speed and accuracy tests. These tests show the advantage the variable-step methods have over fixed-step methods for elliptical orbits. The variable-step Störmer-Cowell method is compared to two other variable-step methods in these tests, s-integration and the Shampine-Gordon integrator. Orbit propagation tests in which the integrators are set to give equivalent accuracy show that s-integration and the variable-step Störmer-Cowell method are faster than the Shampine-Gordon integrator, because Shampine-Gordon has an additional evaluation per step, and because Shampine-Gordon is biased toward keeping a constant step size. The variable-step Störmer-Cowell method is somewhat slower than s-integration in most cases, which indicates that the analytical step regulation that s-integration uses is giving appropriate step sizes. However, the variable-step Störmer-Cowell method has an advantage over s-integration at lower perigees, indicating that s-integration needs step sizes at apogee that are too small in order to have small enough steps at perigee to maintain accuracy when drag is a significant factor. The tests show that s-integration has an advantage over t-integration above an eccentricity of approximately 0.15, and the Shampine-Gordon integrator is more efficient than t-integration for eccentricities over approximately 0.60. These results are independent of perigee height. The variable-step Störmer-Cowell method also has an advantage over t-integration for eccentricities over approximately 0.15 for perigee heights of 400 km and higher. For 300 km perigee height orbits, the variable-step Störmer-Cowell method has an advantage for eccentricities over approximately 0.10.

Orbit determination tests show that s-integration and Shampine-Gordon give comparable results to t-integration when used at the eccentricities specified by the orbit propagation tests. A time improvement of 14.6% and 14.7% are achieved by using s-integration and the variable-step Störmer-Cowell method, respectively, at eccentricities above 0.15, and a time improvement of 7.0% is achieved by using Shampine-Gordon at eccentricities above 0.60. The variable-step Störmer-Cowell method is also able to update more objects than t-integration and s-integration, indicating that it may handle drag better than the other methods.

6.2 Recommendations for Future Study

The results suggest that s-integration works well except when drag is a significant factor. To give larger steps at apogee while maintaining accuracy at perigee with drag present, a different value of n is needed in the general Sundman transformation. While Merson, Nacozy, and others found that $n = 1.5$ works best in the transformation, they only investigated values of $n = 1, 1.5$, and 2. A study of other values of n is warranted to find the value that gives the best speed advantage, which may depend on perigee height and other factors. The variable-step Störmer-Cowell method, or a similar method, could be used in such a study. If a variable-step method is used with s as the independent variable, then the variable-step algorithm should never want to change the step size. The best value of n can be found by varying n and examining the step-size changing factor ρ over the integration. The value of n that keeps ρ closest to one is best. The results here indicate that a value close to $n = 1.5$ is most likely ideal for higher perigees, but a different value is needed at lower perigees.

The results from Zadunaisky's test using ephemeris compression with the Runge-Kutta integrator with a small step size demonstrate that Zadunaisky's test can give reliable results. However, the lack of smoothness in the fit used prevents the test from working well with larger step sizes or with multi-step integrators. To overcome this problem, a method of fitting ephemeris is needed which is both accurate and smooth. Some modification of the Hybrid Ephemeris Compression Model used here may give such a result. The smoothing functions between consecutive orbits would need to be modified to cover a longer timespan, and to keep more derivatives continuous.

This study did not consider discontinuous forces, such as solar radiation pressure or thrust. Multi-step integrators are derived assuming that the forces are continuous among the backpoints, so any discontinuity causes integration error. One way to avoid this error is to restart the integrator at the point of the discontinuity. Because the method derived here is variable step and self-starting, the method could be used to handle discontinuities by stepping directly to the point of the discontinuity and then restarting as a first-order method. However, for solar radiation pressure these restarts would be required twice an orbit, as the object moves in and out of eclipse. Such frequent restarts would have a significant effect on run-time. Woodburn [7], Lundberg [43], [44], and Lundberg et. al. [45] have discussed ways to handle solar radiation pressure without the need to restart. Instead, following a shadow crossing the backpoints are modified to account for the discontinuity. When the object crosses into eclipse, the solar radiation pressure force is subtracted from the backpoints, and when the object comes out of eclipse the force is added to backpoints. This method could be used with the integrator derived here to

handle solar radiation pressure. A similar method could be used to handle thrust.

The variable-step double-integration multi-step integrator is derived here because it has the best features in several categories of numerical integrators. However, the method is non-summed, though the summed form reduces round-off error in multi-step methods. Creating a summed variable-step method is complicated because the summation term assumes that the step size is constant going back to epoch. Though Krogh describes how to recompute the sums when the step size is doubled or halved in [23], restricting the variable-step algorithm to doubling and halving reduces the benefit of variable-step integration. A method for recomputing the sums when the step size is changed by an arbitrary factor is required for an effective summed variable-step method.

Appendix A

Ephemeris Compression Equations

The Hybrid Ephemeris Compression Model [38], gives position by adding the position given from the mean element fit to the residual ephemeris given by the Fourier fit. The position in Cartesian coordinates, x, y, and z, may be written

$$\begin{aligned} x &= x' + \delta x, \\ y &= y' + \delta y, \\ z &= z' + \delta z, \end{aligned} \tag{A.1}$$

where x', y', and z' are from the mean element fit and δx, δy, and δz are from the Fourier fit.

A.1 Mean Element Fit

The position can be found from the mean elements given by the fit. For the purposes of this study, these elements are not corrected for the J_2 periodic effect. The classical elements are given by the coefficients of the fit,

$$\begin{aligned} n &= c_{10} + c_{11}t + c_{12}t^2 + c_{13}t^3, \\ e &= c_{20} + c_{21}t + c_{22}t^2 + c_{23}t^3, \\ i &= c_{30} + c_{31}t + c_{32}t^2 + c_{33}t^3, \\ \omega &= c_{40} + c_{41}t + c_{42}t^2 + c_{43}t^3, \\ \Omega &= c_{50} + c_{51}t + c_{52}t^2 + c_{53}t^3, \\ M &= c_{60} + c_{61}t + c_{62}t^2 + c_{63}t^3, \end{aligned} \tag{A.2}$$

where the c are coefficients from the Chebyshev fit, t is time, and n is the mean motion, e is eccentricity, i is inclination, ω is the argument of perigee, Ω is the right ascension of the ascending node, and M is the mean anomaly.

The derivatives in time of these expressions are straightforward, the first derivatives are

$$\begin{aligned}
\dot{n} &= c_{11} + c_{12}t + c_{13}t^2, \\
\dot{e} &= c_{21} + c_{22}t + c_{23}t^2, \\
\dot{i} &= c_{31} + c_{32}t + c_{33}t^2, \\
\dot{\omega} &= c_{41} + c_{42}t + c_{43}t^2, \\
\dot{\Omega} &= c_{51} + c_{52}t + c_{53}t^2, \\
\dot{M} &= c_{61} + c_{62}t + c_{63}t^2,
\end{aligned} \tag{A.3}$$

and the second derivatives are

$$\begin{aligned}
\ddot{n} &= c_{12} + c_{13}t, \\
\ddot{e} &= c_{22} + c_{23}t, \\
\ddot{i} &= c_{32} + c_{33}t, \\
\ddot{\omega} &= c_{42} + c_{43}t, \\
\ddot{\Omega} &= c_{52} + c_{53}t, \\
\ddot{M} &= c_{62} + c_{63}t.
\end{aligned} \tag{A.4}$$

To find the position from the elements, define the quantities,

$$\begin{aligned}
l_1 &= \cos\Omega\cos\omega - \sin\Omega\sin\omega\cos i, \\
m_1 &= \sin\Omega\cos\omega + \cos\Omega\sin\omega\cos i, \\
n_1 &= \sin\omega\sin i, \\
l_2 &= -\cos\Omega\sin\omega - \sin\Omega\cos\omega\cos i, \\
m_2 &= -\sin\Omega\sin\omega + \cos\Omega\cos\omega\cos i, \\
n_2 &= \cos\omega\sin i.
\end{aligned} \tag{A.5}$$

These quantities represent elements of a rotation matrix from the satellite's orbital frame to an earth centered inertial frame ([46], p. 82). The position can be found in terms of these rotational quantities,

$$\begin{aligned}
x' &= al_1\cos E + bl_2\sin E - ael_1, \\
y' &= am_1\cos E + bm_2\sin E - aem_1, \\
z' &= an_1\cos E + bn_2\sin E - aen_1,
\end{aligned} \tag{A.6}$$

where a is the semi-major axis,

$$a = \sqrt[3]{\frac{\mu}{n^2}}, \tag{A.7}$$

b is the semi-minor axis,

$$b = a\sqrt{1-e^2}, \tag{A.8}$$

and E is the eccentric anomaly. The eccentric anomaly is found by solving Kepler's equation,

$$M = E - e\sin E. \tag{A.9}$$

The first derivative of position is found by taking a time derivative of (A.6),

$$\begin{aligned}
\dot{x}' =& \dot{a}l_1\cos E + a\dot{l_1}\cos E - al_1\dot{E}\sin E + \dot{b}l_2\sin E + b\dot{l_2}\sin E + bl_2\dot{E}\cos E \\
&- \dot{a}el_1 - a\dot{e}l_1 - ael_1^{\cdot}, \\
\dot{y}' =& \dot{a}m_1\cos E + a\dot{m_1}\cos E - am_1\dot{E}\sin E + \dot{b}m_2\sin E + b\dot{m_2}\ \sin E \\
&+ bm_2\dot{E}\cos E - \dot{a}em_1 - a\dot{e}m_1 - aem_1^{\cdot}, \\
\dot{z}' =& \dot{a}n_1\cos E + a\dot{n_1}\cos E - an_1\dot{E}\sin E + \dot{b}n_2\sin E + b\dot{n_2}\sin E + bn_2\dot{E}\cos E \\
&- \dot{a}en_1 - a\dot{e}n_1 - aen_1^{\cdot}.
\end{aligned} \tag{A.10}$$

The derivatives of the rotational quantities are found by taking the derivative of (A.5),

$$\begin{aligned}
\dot{l_1} =& -\dot{\Omega}m_1 + \dot{\omega}l_2 + \dot{i}n_1\sin\Omega, \\
\dot{m_1} =& \dot{\Omega}l_1 + \dot{\omega}m_2 - \dot{i}n_1\cos\Omega, \\
\dot{n_1} =& \dot{\omega}n_2 + \dot{i}\sin\omega\cos i, \\
\dot{l_2} =& -\dot{\Omega}m_2 - \dot{\omega}l_1 + \dot{i}n_2\sin\Omega, \\
\dot{m_2} =& \dot{\Omega}l_2 - \dot{\omega}m_1 - \dot{i}n_2\cos\Omega, \\
\dot{n_2} =& -\dot{\omega}n_1 + \dot{i}\cos\omega\cos i.
\end{aligned} \tag{A.11}$$

The derivative of the semi-major axis is found from (A.7),

$$\dot{a} = -\frac{2}{3}\dot{n}\sqrt[3]{\frac{\mu}{n^5}}, \tag{A.12}$$

and the derivative of the semi-minor axis is found from (A.8),

$$\dot{b} = \dot{a}\sqrt{1-e^2} - \frac{ae\dot{e}}{\sqrt{1-e^2}}. \tag{A.13}$$

The derivative of the eccentric anomaly is found by taking a derivative of Kepler's equation, (A.9),

$$\dot{E} = \frac{\dot{M} + \dot{e}\sin E}{1 - e\cos E}. \tag{A.14}$$

The second derivative of position is found by taking a time derivative of (A.10),

$$\begin{aligned}
\ddot{x}' =& \ddot{a}l_1 \cos E + 2\dot{a}\dot{l}_1 \cos E - 2\dot{a}l_1\dot{E} \sin E + a\ddot{l}_1 \cos E - 2a\dot{l}_1\dot{E} \sin E - al_1\ddot{E} \sin E \\
& - al_1\dot{E}^2 \cos E + \ddot{b}l_2 \sin E + 2\dot{b}\dot{l}_2 \sin E + 2\dot{b}l_2\dot{E} \cos E + b\ddot{l}_2 \sin E + 2b\dot{l}_2\dot{E} \cos E \\
& + bl_2\ddot{E} \cos E - bl_2\dot{E}^2 \sin E - \ddot{a}el_1 - 2\dot{a}\dot{e}l_1 - 2\dot{a}e\dot{l}_1 - a\ddot{e}l_1 - 2a\dot{e}\dot{l}_1 - ae\ddot{l}_1, \\
\ddot{y}' =& \ddot{a}m_1 \cos E + 2\dot{a}\dot{m}_1 \cos E - 2\dot{a}m_1\dot{E} \sin E + a\ddot{m}_1 \cos E - 2a\dot{m}_1\dot{E} \sin E - am_1\ddot{E} \sin E \\
& - am_1\dot{E}^2 \cos E + \ddot{b}m_2 \sin E + 2\dot{b}\dot{m}_2 \sin E + 2\dot{b}m_2\dot{E} \cos E + b\ddot{m}_2 \sin E + 2b\dot{m}_2\dot{E} \cos E \\
& + bm_2\ddot{E} \cos E - bm_2\dot{E}^2 \sin E - \ddot{a}em_1 - 2\dot{a}\dot{e}m_1 - 2\dot{a}e\dot{m}_1 - a\ddot{e}m_1 - 2a\dot{e}\dot{m}_1 - ae\ddot{m}_1, \\
\ddot{z}' =& \ddot{a}n_1 \cos E + 2\dot{a}\dot{n}_1 \cos E - 2\dot{a}n_1\dot{E} \sin E + a\ddot{n}_1 \cos E - 2a\dot{n}_1\dot{E} \sin E - an_1\ddot{E} \sin E \\
& - an_1\dot{E}^2 \cos E + \ddot{b}n_2 \sin E + 2\dot{b}\dot{n}_2 \sin E + 2\dot{b}n_2\dot{E} \cos E + b\ddot{n}_2 \sin E + 2b\dot{n}_2\dot{E} \cos E \\
& + bn_2\ddot{E} \cos E - bn_2\dot{E}^2 \sin E - \ddot{a}en_1 - 2\dot{a}\dot{e}n_1 - 2\dot{a}e\dot{n}_1 - a\ddot{e}n_1 - 2a\dot{e}\dot{n}_1 - ae\ddot{n}_1.
\end{aligned} \tag{A.15}$$

The second derivative of the rotational quantities is found from (A.11),

$$\begin{aligned}
\ddot{l}_1 =& -\ddot{\Omega}m_1 - \dot{\Omega}\dot{m}_1 + \ddot{\omega}l_2 + \dot{\omega}\dot{l}_2 + \ddot{i}n_1 \sin\Omega + \dot{i}\dot{n}_1 \sin\Omega + \dot{i}\dot{\Omega}n_1 \cos\Omega, \\
\ddot{m}_1 =& \ddot{\Omega}l_1 + \dot{\Omega}\dot{l}_1 + \ddot{\omega}m_2 + \dot{\omega}\dot{m}_2 - \ddot{i}n_1 \cos\Omega - \dot{i}\dot{n}_1 \cos\Omega + \dot{i}\dot{\Omega}n_1 \sin\Omega, \\
\ddot{n}_1 =& \ddot{\omega}n_2 + \dot{\omega}\dot{n}_2 + \ddot{i} \sin\omega \cos i + \dot{i}\dot{\omega} \cos\omega \cos i - \dot{i}^2 n_1, \\
\ddot{l}_2 =& -\ddot{\Omega}m_2 - \dot{\Omega}\dot{m}_2 - \ddot{\omega}l_1 - \dot{\omega}\dot{l}_1 + \ddot{i}n_2 \sin\Omega + \dot{i}\dot{n}_2 \sin\Omega + \dot{i}\dot{\Omega}n_2 \cos\Omega, \\
\ddot{m}_2 =& \ddot{\Omega}l_2 + \dot{\Omega}\dot{l}_2 - \ddot{\omega}m_1 - \dot{\omega}\dot{m}_1 - \ddot{i}n_2 \cos\Omega - \dot{i}\dot{n}_2 \cos\Omega + \dot{i}\dot{\Omega}n_2 \sin\Omega, \\
\ddot{n}_2 =& -\ddot{\omega}n_1 - \dot{\omega}\dot{n}_1 + \ddot{i} \cos\omega \cos i - \dot{i}\dot{\omega} \sin\omega \cos i - \dot{i}^2 n_2.
\end{aligned} \tag{A.16}$$

The second derivative of the semi-major axis is found from (A.12),

$$\ddot{a} = -\frac{2}{3}\ddot{n}\sqrt[3]{\frac{\mu}{n^5}} + \frac{10}{9}\dot{n}^2\sqrt[3]{\frac{\mu}{n^8}}, \tag{A.17}$$

and the second derivative of the semi-minor axis is found from (A.13),

$$\ddot{b} = \ddot{a}\sqrt{1-e^2} - \frac{2\dot{a}e\dot{e} - a\dot{e}^2 - ae\ddot{e}}{\sqrt{1-e^2}} - \frac{a\dot{e}^2}{\sqrt{(1-e^2)^3}}. \tag{A.18}$$

The second derivative of the eccentric anomaly is found from (A.14),

$$\ddot{E} = \frac{(1 - e\cos E)(\ddot{M} + \ddot{e}\sin E + \dot{e}\dot{E}\cos E) - (\dot{M} + \dot{e}\sin E)(-\dot{e}\cos E + e\dot{E}\sin E)}{(1 - e\cos E)^2}. \tag{A.19}$$

With the coefficients c from the Chebyshev fit given, the elements and the derivatives of the elements can be found, from (A.2), (A.3), and (A.4). From the elements, the related quantities given in (A.5) can be found, as well as the semi-major and semi-minor axes, the eccentric anomaly, and their derivatives. Finally, the Cartesian position, x', y', and z', and their derivatives can be found. These values must then be added to the values of the residual ephemeris from the Fourier fit.

A.2 Fourier Fit

The equations for constructing the residual ephemeris are found in [39]. There are six separate formulas, for the Fourier series, the quintic interpolating polynomial, the quartic interpolating polynomial, and the three smoothing functions.

The Fourier form of the residual vector ([39], p. 346) is

$$\boldsymbol{x}_{\mathrm{f}}(t) = \boldsymbol{a}_{0N} + \sum_{k=1}^{\infty} \left(\boldsymbol{a}_{kN} \cos\left[\frac{2\pi k(t - t_{\min_N})}{T_1}\right] + \boldsymbol{b}_{kN} \sin\left[\frac{2\pi k(t - t_{\min_N})}{T_1}\right] \right), \quad t_{\min_N} < t < t_{\max_N}, \tag{A.20}$$

where the residual vector $\boldsymbol{x}$ represents $[\delta x \ \delta y \ \delta z]^T$, N refers to the orbit number, T_1 is the period of the orbit, and $\boldsymbol{a}$ and $\boldsymbol{b}$ are coefficients from the Fourier fit. Differentiating,

$$\dot{\boldsymbol{x}}_{\mathrm{f}}(t) = \sum_{k=1}^{\infty} \frac{2\pi k}{T_1} \left(-\boldsymbol{a}_{kn} \sin\left[\frac{2\pi k(t - t_{\min_n})}{T_1}\right] + \boldsymbol{b}_{kn} \cos\left[\frac{2\pi k(t - t_{\min_n})}{T_1}\right] \right), \quad t_{\min_n} < t < t_{\max_n}. \tag{A.21}$$

Differentiating again,

$$\ddot{\boldsymbol{x}}_{\mathrm{f}}(t) = -\sum_{k=1}^{\infty} \left(\frac{2\pi k}{T_1}\right)^2 \left(\boldsymbol{a}_{kn} \cos\left[\frac{2\pi k(t - t_{\min_n})}{T_1}\right] + \boldsymbol{b}_{kn} \sin\left[\frac{2\pi k(t - t_{\min_n})}{T_1}\right] \right), \quad t_{\min_n} < t < t_{\max_n}. \tag{A.22}$$

The quintic interpolating polynomial used between consecutive orbits ([39], p. 352) is

$$\boldsymbol{x}_{\mathrm{p}}(t) = \sum_{\substack{j=1 \\ j \notin [4,7]}}^{10} \boldsymbol{x}_j \prod_{\substack{i=1 \\ i \notin [4,7] \\ i \neq j}}^{10} \frac{t - t_i}{t_j - t_i}, \qquad t_1 \leq t \leq t_{10}. \tag{A.23}$$

Differentiating,

$$\dot{\boldsymbol{x}}_{\mathrm{p}}(t) = \sum_{\substack{j=1 \\ j \notin [4,7]}}^{10} \boldsymbol{x}_j \sum_{\substack{k=1 \\ k \notin [4,7] \\ k \neq j}}^{10} \frac{1}{t_j - t_k} \prod_{\substack{i=1 \\ i \notin [4,7] \\ i \neq j,k}}^{10} \frac{t - t_i}{t_j - t_i}, \qquad t_1 \leq t \leq t_{10}. \tag{A.24}$$

Differentiating again,

$$\ddot{\boldsymbol{x}}_{\mathrm{p}}(t) = \sum_{\substack{j=1 \\ j \notin [4,7]}}^{10} \boldsymbol{x}_j \sum_{\substack{k=1 \\ k \notin [4,7] \\ k \neq j}}^{10} \frac{1}{t_j - t_k} \sum_{\substack{\ell=1 \\ \ell \notin [4,7] \\ \ell \neq j,k}}^{10} \frac{1}{t_j - t_\ell} \prod_{\substack{i=1 \\ i \notin [4,7] \\ i \neq j,k,\ell}}^{10} \frac{t - t_i}{t_j - t_i}, \qquad t_1 \leq t \leq t_{10}. \tag{A.25}$$

The first smoothing function between the Fourier series at the end of an orbit and the interpolating polynomial ([39], p. 353) is

$$\boldsymbol{x}_{1-2}(t) = \boldsymbol{x}_{\mathrm{f}}(t) + \frac{1}{2}\left[\boldsymbol{x}_{\mathrm{p}}(t) - \boldsymbol{x}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t-t_1}{t_2-t_1}\right), \qquad t_1 \le t \le t_2, \tag{A.26}$$

Differentiating,

$$\begin{aligned}\dot{\boldsymbol{x}}_{1-2}(t) = \dot{\boldsymbol{x}}_{\mathrm{f}}(t) &+ \frac{1}{2}\left[\dot{\boldsymbol{x}}_{\mathrm{p}}(t) - \dot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t-t_1}{t_2-t_1}\right)\\ &+ \frac{\pi\left[\boldsymbol{x}_{\mathrm{p}}(t) - \boldsymbol{x}_{\mathrm{f}}(t)\right]}{2(t_2-t_1)}\sin\pi\frac{t-t_1}{t_2-t_1}, \qquad t_1 \le t \le t_2.\end{aligned} \tag{A.27}$$

Differentiating again,

$$\begin{aligned}\ddot{\boldsymbol{x}}_{1-2}(t) = \ddot{\boldsymbol{x}}_{\mathrm{f}}(t) &+ \frac{1}{2}\left[\ddot{\boldsymbol{x}}_{\mathrm{p}}(t) - \ddot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t-t_1}{t_2-t_1}\right)\\ &+ \frac{\pi\left[\dot{\boldsymbol{x}}_{\mathrm{p}}(t) - \dot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]}{(t_2-t_1)}\sin\pi\frac{t-t_1}{t_2-t_1}\\ &+ \frac{\pi^2\left[\boldsymbol{x}_{\mathrm{p}}(t) - \boldsymbol{x}_{\mathrm{f}}(t)\right]}{2(t_2-t_1)^2}\cos\pi\frac{t-t_1}{t_2-t_1}, \qquad t_1 \le t \le t_2.\end{aligned} \tag{A.28}$$

The second smoothing function between the interpolating polynomial and the Fourier series at the beginning of the orbit ([39], p. 353) is

$$\begin{aligned}\boldsymbol{x}_{9-10}(t) = \boldsymbol{x}_{\mathrm{f}}(t) &+ \frac{1}{2}\left[\boldsymbol{x}_{\mathrm{p}}(t) - \boldsymbol{x}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t_{10}-t}{t_{10}-t_9}\right),\\ &t_9 \le t \le t_{10},\end{aligned} \tag{A.29}$$

Differentiating,

$$\begin{aligned}\dot{\boldsymbol{x}}_{9-10}(t) = \dot{\boldsymbol{x}}_{\mathrm{f}}(t) &+ \frac{1}{2}\left[\dot{\boldsymbol{x}}_{\mathrm{p}}(t) - \dot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t_{10}-t}{t_{10}-t_9}\right)\\ &- \frac{\pi\left[\boldsymbol{x}_{\mathrm{p}}(t) - \boldsymbol{x}_{\mathrm{f}}(t)\right]}{2(t_{10}-t_9)}\sin\pi\frac{t_{10}-t}{t_{10}-t_9}, \qquad t_9 \le t \le t_{10}.\end{aligned} \tag{A.30}$$

Differentiating again,

$$\begin{aligned}\ddot{\boldsymbol{x}}_{9-10}(t) = \ddot{\boldsymbol{x}}_{\mathrm{f}}(t) &+ \frac{1}{2}\left[\ddot{\boldsymbol{x}}_{\mathrm{p}}(t) - \ddot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t_{10}-t}{t_{10}-t_9}\right)\\ &- \frac{\pi\left[\dot{\boldsymbol{x}}_{\mathrm{p}}(t) - \dot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]}{(t_{10}-t_9)}\sin\pi\frac{t_{10}-t}{t_{10}-t_9}\\ &+ \frac{\pi^2\left[\boldsymbol{x}_{\mathrm{p}}(t) - \boldsymbol{x}_{\mathrm{f}}(t)\right]}{2(t_{10}-t_9)^2}\cos\pi\frac{t_{10}-t}{t_{10}-t_9}, \qquad t_9 \le t \le t_{10}.\end{aligned} \tag{A.31}$$

The quartic polynomial near epoch ([39], p. 353) is

$$\boldsymbol{x}_{\mathrm{p}}(t) = \sum_{\substack{j=1\\ j\neq 2}}^{6} \boldsymbol{x}_j \prod_{\substack{i=1\\ i\neq 2 \text{ or } j}}^{6} \frac{t - t_i}{t_j - t_i}, \qquad t_1 \leq t \leq t_6. \tag{A.32}$$

Differentiating,

$$\dot{\boldsymbol{x}}_{\mathrm{p}}(t) = \sum_{\substack{j=1\\ j\neq 2}}^{6} \boldsymbol{x}_j \sum_{\substack{k=1\\ k\neq 2 \text{ or } j}}^{6} \frac{1}{t_j - t_k} \prod_{\substack{i=1\\ i\neq 2,j,k}}^{6} \frac{t - t_i}{t_j - t_i}, \qquad t_1 \leq t \leq t_6. \tag{A.33}$$

Differentiating again,

$$\ddot{\boldsymbol{x}}_{\mathrm{p}}(t) = \sum_{\substack{j=1\\ j\neq 2}}^{6} \boldsymbol{x}_j \sum_{\substack{k=1\\ k\neq 2 \text{ or } j}}^{6} \frac{1}{t_j - t_k} \sum_{\substack{\ell=1\\ \ell\neq 2,j,k}}^{6} \frac{1}{t_j - t_\ell} \prod_{\substack{i=1\\ i\neq 2,j,k,\ell}}^{6} \frac{t - t_i}{t_j - t_i}, \qquad t_1 \leq t \leq t_6. \tag{A.34}$$

The smoothing function between the quartic polynomial and the Fourier series of the first orbit ([39], p. 354) is

$$\boldsymbol{x}_{5-6}(t) = \boldsymbol{x}_{\mathrm{f}}(t) + \frac{1}{2}\left[\boldsymbol{x}_{\mathrm{p}}(t) - \boldsymbol{x}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t_6 - t}{t_6 - t_5}\right), \qquad t_5 \leq t \leq t_6, \tag{A.35}$$

Differentiating,

$$\begin{aligned}\dot{\boldsymbol{x}}_{5-6}(t) = \dot{\boldsymbol{x}}_{\mathrm{f}}(t) &+ \frac{1}{2}\left[\dot{\boldsymbol{x}}_{\mathrm{p}}(t) - \dot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t_6 - t}{t_6 - t_5}\right)\\ &- \frac{\pi\left[\boldsymbol{x}_{\mathrm{p}}(t) - \boldsymbol{x}_{\mathrm{f}}(t)\right]}{2(t_6 - t_5)}\sin\pi\frac{t_6 - t}{t_6 - t_5}, \qquad t_5 \leq t \leq t_6.\end{aligned} \tag{A.36}$$

Differentiating again,

$$\begin{aligned}\ddot{\boldsymbol{x}}_{5-6}(t) = \ddot{\boldsymbol{x}}_{\mathrm{f}}(t) &+ \frac{1}{2}\left[\ddot{\boldsymbol{x}}_{\mathrm{p}}(t) - \ddot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t_6 - t}{t_6 - t_5}\right)\\ &- \frac{\pi\left[\dot{\boldsymbol{x}}_{\mathrm{p}}(t) - \dot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]}{(t_6 - t_5)}\sin\pi\frac{t_6 - t}{t_6 - t_5}\\ &+ \frac{\pi^2\left[\boldsymbol{x}_{\mathrm{p}}(t) - \boldsymbol{x}_{\mathrm{f}}(t)\right]}{2(t_6 - t_5)^2}\cos\pi\frac{t_6 - t}{t_6 - t_5}, \qquad t_5 \leq t \leq t_6.\end{aligned} \tag{A.37}$$

Because the interpolating polynomials pass through the values of the Fourier series, the smoothing functions give continuous first and second derivatives. However, the third

derivative is not continuous. As an example, consider the derivative of (A.37),

$$\begin{aligned}\dddot{\boldsymbol{x}}_{5-6}(t) = \dddot{\boldsymbol{x}}_{\mathrm{f}}(t) + \frac{1}{2}\left[\dddot{\boldsymbol{x}}_{\mathrm{p}}(t) - \dddot{\boldsymbol{x}}_{\mathrm{f}}(t)\right]\left(1 - \cos\pi\frac{t_6 - t}{t_6 - t_5}\right) \\ - \frac{3\pi[\ddot{\boldsymbol{x}}_{\mathrm{p}} - \ddot{\boldsymbol{x}}_{\mathrm{f}}]}{2(t_6 - t_5)}\sin\pi\frac{t_6 - t}{t_6 - t_5} \\ + \frac{\pi^2[\dot{\boldsymbol{x}}_{\mathrm{p}} - \dot{\boldsymbol{x}}_{\mathrm{f}}]}{2(t_6 - t_5)^2}\cos\pi\frac{t_6 - t}{t_6 - t_5} \\ + \frac{\pi^3[\boldsymbol{x}_{\mathrm{p}} - \boldsymbol{x}_{\mathrm{f}}]}{2(t_6 - t_5)^3}\sin\pi\frac{t_6 - t}{t_6 - t_5}\end{aligned} \tag{A.38}$$

Because the first derivatives of the interpolating polynomial and the Fourier series are not equal, the third derivative of the smoothing function does not match the values of the third derivative of the polynomial and Fourier series at the endpoints.

Appendix B

Matlab Code

```
% Matt Berry
% This code uses a variable-step double-integration multi-step
% integrator (variable-step Stormer-Cowell), to integrate
% scalar 2nd order differential equations, y'' = f(x,y,y').
% The function dbrhs returns f(x,y,y').  The initial
% conditions here are for integrating a sine wave,
% with dbrhs returning -y.
%
% Only the double-integration code is used in this script,
% but the corresponding single-integration code is also shown,
% in some cases commented-out. Only an absolute tolerance is
% used in this code, so there are no weighting factors.
%
% This script produces a graph of integration points and
% interpolation points.

clear

% Stormer-Cowell coefficients, lamda
d_coeffs = [1 0 1/12 1/12 19/240 3/40 863/12096 275/4032 ...
      33953/518400 8183/129600];

% first order coefficients, gamma
s_coeffs = [1 1/2 5/12 3/8 251/720 95/288 19087/60480 ...
      5257/17280 1070017/3628800 25713/89600];
```

```
% set tolerance
tol = 1e-14;

% set maximum number of backpoints
kmax = 9;
% index on gamma, lamda starts at zero in derivation, 1 here.
for i=1:kmax
   lamdastar(i) = d_coeffs(i+1) - d_coeffs(i);
   gammastar(i) = s_coeffs(i+1) - s_coeffs(i);
end

% initialize, define f, dx, initial conditions
x0 = 0;
y0 = 0;
yp0 = 1;
xf = 10*pi;
n = 1;
k = 1;


ydp(1) = dbrhs(x0,[y0 yp0]);
x(1) = x0;
y(1) = y0;
yp(1) = yp0;

dx = 0.25 * sqrt(tol/abs(yp(1)));
% Set the initial step size to dx, to prevent a divide by zero
% message. This value isn't actually used for anything.
h(1) = dx;

% start difference table
diff(1,1) = ydp;

% use initial step for first step
r = 1;

% now we can integrate
```

```
nout = 1;
% ask for evenly space values
for xout=x0:(xf-x0)/1000:xf

   % integrate to, or past, the requested point
   while x(n) < xout
      dx = r*dx;
      x(n+1) = x(n) + dx;
      h(n+1) = dx;
      ratio = h(n+1) / h(n);
      invrat = 1/ratio;

      % calculate coefficients
      % get alpha
      for i=1:k
         if i==1
            psi(1) = h(n+1);
         else
            psi(i) = psi(i-1) + h(n-i+2);
         end
         alpha(i) = h(n+1) / psi(i);
      end
      psinm1(1) = 0;
      % this i is really i+1, since psinm1(0) is needed
      for i=2:k-1
         psinm1(i) = psinm1(i-1) + h(n+1-i);
      end
      for i=1:k+1
         for q=1:k+3-i
            if i == 1
               g(q,i) = 1/q;
               gp(q,i) = 1/q * (-invrat)^q;
            elseif i == 2
               g(q,i) = 1/q/(q+1);
               gp(q,i) = 1/q/(q+1) * (-invrat)^(q+1);
            else
               g(q,i) = g(q,i-1) - alpha(i-1) * g(q+1,i-1);
               % index is changed in psinm1
```

```
            gp(q,i) = psinm1(i-2) / psi(i-1) * gp(q,i-1) ...
               - alpha(i-1) * gp(q+1,i-1);
         end
      end
   end

   % need psi(n)
   psin(1) = h(n);
   for i=2:k-1
      psin(i) = psin(i-1) + h(n+1-i);
   end

   sum2=0;
   sum1=0;
   for m=1:k
%beta(m) = psi_1(n+1)...psi_{m-1}(n+1) / psi_1(n)...psi_{m-1}(n)
      if m == 1
         beta(1) = 1;
      else
         beta(m) = beta(m-1) * psi(m-1) / psin(m-1);
      end
      % phi* = beta * phi
      diff(k,m) = diff(k,m) * beta(m);
      sum2 = sum2 + ( g(2,m) + ratio * gp(2,m) ) * diff(k,m);
      sum1 = sum1 + g(1,m)*diff(k,m);
   end
   % predict
   yp(n+1) = yp(n) + dx * sum1;
   if k>1
      y(n+1) = (1+ratio)*y(n) - ratio*y(n-1) + dx^2 * sum2;
   else
      % use general formulation on first point only
      y(n+1) = y(n) + dx*yp(n) + dx^2 * g(2,1)*diff(1,1);
   end

   % evaluate, and get new phi
   if k < kmax
      % append this value to difference array
```

```
   npt = k+1;
else
   % cycle this value into difference array
   diff(1:k-1,:) = diff(2:k,:);
   npt = k;
end
diff(npt,1) = dbrhs(x(n+1),[y(n+1) yp(n+1)]);
for m=2:k+1
   diff(npt,m) = diff(npt,m-1) - diff(npt-1,m-1);
end

% corrector
if k > 1
   y(n+1) = y(n+1) + dx^2 * ( g(2,k+1) + ratio*gp(2,k+1) ) ...
      * diff(npt,k+1);
else
   % general formulation
   y(n+1) = y(n+1) + dx^2 * g(2,2)*diff(2,2);
end
yp(n+1) = yp(n+1) + dx * g(1,k+1) * diff(npt,k+1);

% error estimate
if k>1
   errd = abs( h(n+1)^2 * ( g(2,k+1) - ...
      g(2,k) + ratio*( gp(2,k+1)-gp(2,k) ) ) ...
      * diff(k,k+1) );
else
   errd = abs( h(n+1)^2 * ( g(2,k+1) - g(2,k) ) * diff(k,k+1) );
end
errs = abs( h(n+1) * ( g(1,k+1) - g(1,k) ) * diff(k,k+1) );

if errd > tol% | errs > tol
   % reject this step, put everything back and start over
   r = 0.5;
   if k == kmax
      diff(2:k,:) = diff(1:k-1,:);
   end
   for m=1:k
```

```
      diff(k,m) = diff(k,m) / beta(m);
   end
   fprintf('Step size too big, x= %f, h= %f\n',x(n+1),h(n+1));
   continue
end

if k == kmax
   % normal procedure, find next step size

   sigma = 1;
   for i=2:k+1
      sigma = (i-1) * alpha(i-1) * sigma;
   end
   erks = abs( h(n+1) * gammastar(k) * sigma * diff(k,k+1) );
   erkd = abs( h(n+1)^2 * lamdastar(k) * sigma * diff(k,k+1) );

   rs = (tol/4/erks) ^ (1/(k+1));
   rd = (tol/4/erkd) ^ (1/(k+2));
   %    r = min(rs,rd);
   r = rd;
   % bound r between 0.5 and 2
   if r > 2
      r = 2;
   elseif r < 0.5
      r = 0.5
   end
   % set last used value of k, for interpolator
   lk = k;
   n = n+1;

else
   % do second eval, add increment k, double step size

   % evaluate again
   diff(k+1,1) = dbrhs(x(n+1),[y(n+1) yp(n+1)]);
   for m=2:k+1
      diff(k+1,m) = diff(k+1,m-1) - diff(k,m-1);
   end
```

```
      % increment k, but save value of k used for this
      % step for interpolator
      lk = k;
      k = k+1;
      r = 2;
      n = n+1;
   end

end % end of integration while loop

if xout < x(n)

   % now integrated past requested point, find requested point
   xouts(nout) = xout;
   % get interp coefficients
   hI = xout - x(n);
   ratio = hI/h(n);
   invrat = 1/ratio;
   gamma1(1) = hI/psi(1);
   for i=2:lk
      gamma1(i) = (hI + psi(i-1) ) / psi(i);
   end
   gammap(1) = -1;
   gammap(2) = 0;
   for i=3:lk
      gammap(i) = psin(i-2) / psi(i);
   end
   for i=1:lk+1
      for q=1:lk+3-i
         if i == 1
            gint(q,i) = 1/q;
            gpint(q,i) = 1/q * (-invrat)^q;
         else
            gint(q,i) = gamma1(i-1) * gint(q,i-1) - ...
               hI / psi(i-1) * gint(q+1,i-1);
            gpint(q,i) = gammap(i-1) * gpint(q,i-1) - ...
               hI / psi(i-1) * gpint(q+1,i-1);
```

```
            end
         end
      end
      sum1 = 0;
      sum2 = 0;
      for i=1:lk+1
         sum1 = sum1 + gint(1,i) * diff(npt,i);
         sum2 = sum2 + (gint(2,i) + ratio*gpint(2,i)) * diff(npt,i);
      end
      yout(nout) = (1+ratio)*y(n) - ratio*y(n-1) + hI^2*sum2;
      ypout(nout) = y(n) + hI*sum1;
      nout = nout+1;

   else

      % integrated exactly to requested point
      xouts(nout) = x(n);
      yout(nout) = y(n);
      nout = nout + 1;

   end

end % end of for loop

% plot integration points, step size, and error
subplot 311
plot(x,h,'linewidth',2)
ylabel('\fontsize{14}h')
title('\fontsize{14}Step Size')
grid
subplot 312
plot(x,y,'linewidth',2)
grid
ylabel('\fontsize{14}y')
title('\fontsize{14}Numerical Solution')
subplot 313
plot(x,abs(y-sin(x)),'linewidth',2)
title('\fontsize{14}Error')
```

```
xlabel('\fontsize{14}x')
ylabel('\fontsize{14}|y-sin(x)|')
grid

% plot interpolation points and error
figure
subplot 211
plot(xouts,yout,'linewidth',2)
ylabel('\fontsize{14}y')
title('\fontsize{14}Interpolated Solution')
grid
subplot 212
plot(xouts,abs(yout-sin(xouts)),'linewidth',2)
title('\fontsize{14}Error')
ylabel('\fontsize{14}|y-sin(x)|')
xlabel('\fontsize{14}x')
grid
```

Appendix C

Fortran Code

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine varstormcow(rqtime,currtime,rel,abse,pos,vel,
     >     reset,delt)

      implicit none

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c     variable-step Stormer-Cowell integration routine
c     written by:  Matt Berry

c     Subroutine varstormcow performs one step of the variable-step
c     Stormer-Cowell algorithm.  If the integration step takes the
c     current time beyond the request time, the routine will
c     interpolate to the request time.  The routine changes the value
c     of current time, position, and velocity during the call.  The
c     routine sets the value of delt, which is the amount the
c     current time changed.
c
c     Call with reset = 1 to start the integrator.  The routine will
c     normally return with reset = 0.  If the routine returns
c     reset = 1, the method must restart because a step has failed
c     after 3 tries.
```

```
c     The algorithms used in this code are explained in
c     "A Variable-Step Double-Integration Multi-Step Integrator" by
c     Matthew M. Berry.  PhD Dissertation, Virginia Polytechnic
c     Institute and State University, Blacksburg, VA. April 2004.

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     subroutines used:
c       DERIV(pos,vel,currtime,accel)
c     this routine should return acceleration given position,
c     velocity, and time.
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     call line variables

      double precision, intent(in) :: rqtime ! requested time
      double precision, intent(inout) :: currtime ! current time
      double precision, intent(in) :: rel      ! relative tolerance
      double precision, intent(in) :: abse     ! absolute tolerance
      double precision, intent(inout) :: pos(3) ! position vector
      double precision, intent(inout) :: vel(3)   ! velocity vector
      double precision, intent(out) :: delt ! change in currtime
                                !                during call

c     note: the units of rqtime, currtime, pos, vel, and accel must
c     all be compatible

      integer reset             ! 1 if restarting

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     paramaters
      integer kmax              ! maximum value of k
      parameter (kmax = 9)

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     local variables
      double precision stepsize ! step size
      double precision stepsize2 ! step size squared
```

```
double precision accel(3) ! acceleration vector
double precision diff(3,kmax+1) ! modified divided differences
double precision diffsave(3,kmax) ! diff from last step
double precision r ! amount to change the step by on next step
double precision steps(kmax) ! step size history
double precision psi(kmax) ! sums of steps
double precision alpha(kmax) ! ratio of current step to psi
double precision psinm1(0:kmax-2) ! sums of steps,
                          !          starting 2 steps back
double precision psin(kmax-1) ! sums of steps,
                          !        starting 1 step back
double precision g(kmax+2,kmax+1) ! integration coefficients
double precision gp(kmax+2,kmax+1)! integration coefficients
double precision beta(kmax) ! ratio of phi's
double precision pos2back(3) ! position from 2 steps ago
double precision lastpos(3) ! position from last step
double precision lastvel(3) ! velocity from last step
double precision newdiff(3,kmax+1) ! differences after
                          !             evaluation
double precision savecurrtime ! last value of current time
double precision errd     ! error estimate on position
double precision errs     ! error estimate on velocity
double precision erkd     ! position error estimate
double precision erks     ! velocity error estimate
double precision terrd    ! temp value for errd
double precision terrs    ! temp value for errs
double precision rs       ! r value for single integration
double precision rd       ! r value for double integration
double precision sigma    ! parameter used in error estimate
double precision savestep ! saved step size of
                          !   furthest backpoint
double precision ratio    ! ratio of latest step sizes
double precision invrat   ! inverse of ratio
double precision sum1     ! integration sum for velocity
double precision sum2     ! integration sum for position

double precision gammastars(kmax) ! difference of single
                          !          fixed-step coefficients
```

```
double precision gammastard(kmax) ! difference of double
                          !          fixed-step coefficients

double precision eps      ! tolerance
double precision releps   ! relative error / eps
double precision abseps   ! absolute error / eps
double precision macheps  ! machine epsilon
double precision wts(3)   ! error weighting for velocity
double precision wtd(3)   ! error weighting for position
double precision h1s      ! initial step for single integration
double precision h1d      ! initial step for double integration

double precision intpos(3) ! last position integrated to
double precision intvel(3) ! last velocity integrated to
double precision inttime  ! last time integrated to
                          !   inttime is the same as currtime
                          !   unless the last step had an
                          !   interpolation

double precision hI       ! interpolation step
double precision gint(kmax+2,kmax+1) ! interp. coefficients
double precision gpint(kmax+2,kmax+1) ! interp. coefficients
double precision gamma1(kmax) ! gamma associated with gint
double precision gammap(kmax) ! gamma associated with gpint

integer intdir            ! integration direction
                          !   1 = forward
                          !  -1 = backward

integer q                 ! coefficient index
integer i,m               ! loop control
integer k                 ! number of backpoints
integer lk                ! k in last integration step,
                          !   for interpolator

integer fail              ! number of consecutive failures
logical again             ! true if repeating initial step
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     variables needed for the next step
      save r,diff,stepsize,steps,k,lk,eps, intpos, inttime, intvel,
     >     lastpos, lastvel, psi, psin, releps, abseps, fail, intdir

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     data statements
      data gammastars /-0.5d0, -0.08333333333333333d0,
     >     -0.04166666666666667d0, -0.02638888888888891d0,
     >     -0.01874999999999999d0, -0.014269179894179 86d0,
     >     -0.01136739417989419d0, -0.009356536596119958d0,
     >     -0.007892554012345676d0/

      data gammastard /-1.d0, 0.08333333333333333d0, 0.0d0,
     >     -0.004166666666666666d0, -0.004166666666666666d0,
     >     -0.003654100529100521d0, -0.003141534391534404d0,
     >     -0.002708608906525564d0, -0.002355324074074072d0/

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c     save the current time
      savecurrtime = currtime


      if (reset == 1) then
c     initialization

c     set EPS, releps, abseps
         eps = dmax1(rel,abse)
         releps = rel / eps
         abseps = abse / eps

c     get machine epsilon
         macheps = epsilon(1.d0)

         k = 1
         reset = 0
```

```
c     get initial value of acceleration
         call DERIV(pos,vel,currtime,accel)
         terrs = 0.0d0
         terrd = 0.0d0
         do i = 1,3
            lastpos(i) = pos(i)
            lastvel(i) = vel(i)
            diff(i,1) = accel(i)
            wts(i) = abs(vel(i)) * releps + abseps
            wtd(i) = abs(pos(i)) * releps + abseps
            terrs = terrs + (accel(i) / wts(i))**2
            terrd = terrd + (vel(i) / wtd(i))**2
         enddo

c     get initial step estimate for single and double integration
         h1s = 0.25d0 * sqrt(eps / sqrt(terrs))
         h1d = 0.25d0 * sqrt(eps / sqrt(terrd))

c     get initial step size
         stepsize = min(h1s,h1d,abs(rqtime-currtime))
         stepsize = max(stepsize,4*macheps * currtime)

c     set step size in the right direction
         if ( rqtime >= currtime ) then
            intdir = 1
         else
            intdir = -1
         endif
         stepsize = sign(stepsize,dble(intdir))

         fail = 0
         again = .true.

         do while (again .and. fail < 10)

            again = .false.

            stepsize2 = stepsize * stepsize
```

```
            steps(1) = stepsize

c     do first step with general formulation
c     predict
            do i=1,3
               vel(i) = lastvel(i) + stepsize * diff(i,1)
               pos(i) = lastpos(i) + stepsize * lastvel(i) +
     >              stepsize2 * diff(i,1) / 2.0d0
            enddo
c     advance time
            currtime = currtime + stepsize
c     evaluate
            call DERIV(pos,vel,currtime,accel)
            terrs = 0.0d0
            terrd = 0.0d0
            do i=1,3
               newdiff(i,1) = accel(i)
               newdiff(i,2) = newdiff(i,1) - diff(i,1)
c     correct
               vel(i) = vel(i) + stepsize * newdiff(i,2) / 2.0d0
               pos(i) = pos(i) + stepsize2 * newdiff(i,2) / 6.0d0

c     get error estimate
               wts(i) = abs(lastvel(i)) * releps + abseps
               wtd(i) = abs(lastpos(i)) * releps + abseps
               terrs = terrs + (newdiff(i,k+1) / wts(i))**2
               terrd = terrd + (newdiff(i,k+1) / wtd(i))**2
            enddo

            terrs = sqrt(terrs)
            terrd = sqrt(terrd)

            errd = abs( stepsize2 * ( 1.d0/3.d0 ) ) * terrd
            errs = abs( stepsize * ( 1.d0/2.d0 ) ) * terrs

            if (errd > eps .or. errs > eps) then
c     set everthing back and try again with half step
               again = .true.
```

```
               fail = fail + 1
               currtime = currtime - stepsize
               stepsize = 0.5d0 * stepsize
            elseif(fail == 0) then
c     step succeeded, try again with a larger step,
c     to find maximum initial step
               again = .true.
               currtime = currtime - stepsize
               stepsize = 2.0d0 * stepsize
            endif

         enddo

c     end of initial step loop
         if (again) then
            print*,'Unable to take first step after 10 tries, giving up'
            stop
         endif

c     reset fail count
         fail = 0

c     re-evaluate
         call DERIV(pos,vel,currtime,accel)
c     get differences
         do i=1,3
            newdiff(i,1) = accel(i)
            newdiff(i,2) = newdiff(i,1) - diff(i,1)
            do m=1,2
               diff(i,m) = newdiff(i,m)
            enddo
         enddo

c     set integration time and state to current values
         inttime = currtime
         do i=1,3
            intpos(i) = pos(i)
            intvel(i) = vel(i)
```

```
         enddo

c     increment order, double step size on next step
         lk = 1
         k = 2
         r = 2.0d0

         return

      endif                     ! end of initialization


c     take a step if requested time is past integration time
      if ( (rqtime > inttime .and. intdir == 1) .or.
     >     (rqtime < inttime .and. intdir == -1) )
     >     then

         DO I = 1,3
            pos2back(i) = lastpos(i)
            lastpos(i) = intpos(i)
            lastvel(i) = intvel(i)
         END DO

c     change the step size
         stepsize = stepsize * r
         stepsize2 = stepsize * stepsize
         if (lk == kmax) then
c     If already using maximum value of k, cycle this value into
c     steps array. Otherwise, it gets appended to the end.
            savestep = steps(1)
            do i=1,k-1
               steps(i) = steps(i+1)
            enddo
         endif
         steps(k) = stepsize

c     compute ratio of current step size to last
         ratio = stepsize / steps(k-1)
```

```
         invrat = 1.d0/ratio

c     calculate alpha and psi(n+1)
         do i = 1,k
            if (i > 1) then
               psi(i) = psi(i-1) + steps(k+1-i)
            else
               psi(1) = stepsize
            endif
            alpha(i) = stepsize / psi(i)
         enddo

c     get psi(n-1) (need 0 through k-2)
         psinm1(0) = 0.d0
         do i=1,k-2
            psinm1(i) = psinm1(i-1) + steps(k-1-i)
         enddo

c     get psi(n)
         psin(1) = steps(k-1)
         do i=2,k-1
            psin(i) = psin(i-1) + steps(k-i)
         enddo

c     calculate coefficients

         do i = 1,k+1
            do q = 1,k+3-i
               if (i==1) then
                  g(q,1) = 1.d0 / dble(q)
                  gp(q,1) = g(q,1) * (-invrat)**q
               elseif (i==2) then
                  g(q,2) = 1.d0 / dble(q) / ( dble(q) + 1.d0 )
                  gp(q,2) = g(q,2) * (-invrat)**(q+1)
               else
                  g(q,i) = g(q,i-1) - alpha(i-1) * g(q+1,i-1)
                  gp(q,i) = psinm1(i-3) / psi(i-1) * gp(q,i-1) -
     >                 alpha(i-1) * gp(q+1,i-1)
```

```
               endif
            enddo
         enddo

c     calculate beta
         beta(1) = 1.d0
         do m = 2,k
            beta(m) = beta(m-1) * psi(m-1) / psin(m-1)
         enddo

c     now integrate
         do i=1,3
            sum2 = 0.d0
            sum1 = 0.d0
            do m = 1,k
c     save differences before multiplying by beta
               diffsave(i,m) = diff(i,m)
c     calculate phi* = phi*beta
               diff(i,m) = beta(m) * diff(i,m)
c     calculate sums for integration
               sum2 = sum2 + ( g(2,m) + ratio * gp(2,m) ) * diff(i,m)
               sum1 = sum1 + g(1,m) * diff(i,m)
            enddo

c     predict velocity and position
            pos(i) = (1.d0 + ratio) * lastpos(i) - ratio*pos2back(i)
     >           + stepsize2 * sum2
            vel(i) = lastvel(i) + stepsize * sum1

         enddo

c     update the time
         currtime = inttime + stepsize

c     evaluate force model
         call DERIV(pos,vel,currtime,accel)

         terrs = 0.0d0
```

```
         terrd = 0.0d0

         do i=1,3
c     get new differences
               newdiff(i,1) = accel(i)
               do m=2,k+1
                  newdiff(i,m) = newdiff(i,m-1) - diff(i,m-1)
               enddo

c     corrector
               pos(i) = pos(i) +
     >              stepsize2 * ( g(2,k+1) + ratio*gp(2,k+1) )
     >              * newdiff(i,k+1)
               vel(i) = vel(i) +
     >              stepsize * g(1,k+1) * newdiff(i,k+1)

c     get error estimate
               wts(i) = abs(lastvel(i)) * releps + abseps
               wtd(i) = abs(lastpos(i)) * releps + abseps
               terrs = terrs + (newdiff(i,k+1) / wts(i))**2
               terrd = terrd + (newdiff(i,k+1) / wtd(i))**2
            enddo

            terrs = sqrt(terrs)
            terrd = sqrt(terrd)

            errd = abs( stepsize2 * ( g(2,k+1) - g(2,k) +
     >           ratio * ( gp(2,k+1) - gp(2,k) ) ) ) * terrd
            errs = abs( stepsize * ( g(1,k+1) - g(1,k) ) ) * terrs

            if (errd > eps .or. errs > eps) then
c     step failed,
c     reset everything and use half this step next time
               fail = fail + 1
               r = 0.5d0
               if (lk == kmax) then
                  do m = k,2,-1
                     steps(m) = steps(m-1)
```

```
               enddo
               steps(1) = savestep
            endif
            do i=1,3
               pos(i) = lastpos(i)
               vel(i) = lastvel(i)
               lastpos(i) = pos2back(i)
               do m = 1,k
                  diff(i,m) = diffsave(i,m)
               enddo
            enddo
            currtime = savecurrtime
            delt = 0.d0

c     if 3rd failure, start over
            if (fail >= 3) then
               reset = 1
            endif

c     exit routine
            return
         endif

c     step succeded, reset fail count
         fail = 0

         if (k == kmax) then
c     no longer in variable-order mode
c     calculate step for next time

            sigma = 1.0d0
            do i = 2,10
               sigma = (i-1) * alpha(i-1) * sigma
            enddo

            erks = abs( stepsize * gammastars(k) * sigma ) * terrs
            erkd = abs( stepsize2 * gammastard(k) * sigma ) * terrd
```

```
c     compute r for single and double integration
            rs = (eps/2.d0/erks) ** (1.d0/dble(k+1))
            rd = (eps/2.d0/erkd) ** (1.d0/dble(k+2))

c     use smallest r
            r = min(rs,rd)

c     bound r
            if (r > 2.d0) r = 2.d0
            if (r < 0.5d0) r = 0.5d0

c     set last used value of k
            lk = k

         else
c     variable-order start-up mode
c     Evaluate again, increment k and double step

c     evaluate force model
            call DERIV(pos,vel,currtime,accel)

c     get new differences
            do i=1,3
               newdiff(i,1) = accel(i)
               do m=2,k+1
                  newdiff(i,m) = newdiff(i,m-1) - diff(i,m-1)
               enddo
            enddo

c     set last used value of k, in case an interpolation
c     is needed now
            lk = k
            k = k+1
            r = 2.0d0

         endif

c     Get ready for next step: set integration time and state to
```

```
c     current values, and copy latest differences from newdiff
c     to diff.
         inttime = currtime
         do i=1,3
            do m = 1,lk+1
               diff(i,m) = newdiff(i,m)
            enddo
            intpos(i) = pos(i)
            intvel(i) = vel(i)
         enddo

      endif

c     end of integration step, check to see if
c     interpolation is needed

      if ( (inttime > rqtime .and. intdir == 1) .or.
     >     (inttime < rqtime .and. intdir == -1) ) then
c     integrated too far, interpolate to request time

c     set interpolation step
         hI = rqtime - inttime
         ratio = hI/steps(lk)
         invrat = 1.0d0 / ratio

c     compute gamma values
         gamma1(1) = hI / psi(1)
         do i=2,lk
            gamma1(i) = (hI + psi(i-1)) / psi(i)
         enddo
         gammap(1) = -1.0d0
         gammap(2) = 0.0d0
         do i=3,lk
            gammap(i) = psin(i-2) / psi(i)
         enddo

c     calculate coefficients
         do i=1,lk+1
```

```
            do q=1,lk+3-i
               if (i == 1) then
                  gint(q,i) = 1.0d0 / q
                  gpint(q,i) = 1.0d0 / q * (-invrat)**q
               else
                  gint(q,i) = gamma1(i-1) * gint(q,i-1) - hI/psi(i-1)
     >                 * gint(q+1,i-1)
                  gpint(q,i) = gammap(i-1) * gpint(q,i-1) - hI/psi(i-1)
     >                 * gpint(q+1,i-1)
               endif
            enddo
         enddo

c     compute interpolation values
         do i=1,3
            sum1 = 0.0d0
            sum2 = 0.0d0
            do m=1,lk+1
               sum1 = sum1 + gint(1,m) * diff(i,m)
               sum2 = sum2 + (gint(2,m) + ratio*gpint(2,m))
     >              * diff(i,m)
            enddo

            vel(i) = intvel(i) + hI * sum1
            pos(i) = (1.0d0 + ratio) * intpos(i)
     >           - ratio * lastpos(i) + hI**2 * sum2
         enddo

c     set current time to the request time
         currtime = rqtime

      endif                     ! end of interpolation

c     set the change in time
      delt = currtime - savecurrtime

      return
      end
```

References

[1] J. H. Seago, M. A. Davis, A. E. Reed, E. D. Lydick, and P. W. Schumacher, "Results of naval space surveillance system calibration using satellite laser ranging," in *Proceedings of the 2001 AAS/AIAA Astrodynamics Specialists Conference*, (Quebec City), American Astronautical Society, January 2001. AAS 01–361. 1

[2] D. A. Vallado, *Fundamentals of Astrodynamics and Applications*. New York: McGraw-Hill, 1997. 2, 29, 36

[3] S. Herrick, *Astrodynamics: Orbit Correction, Perturbation Theory, Integration*, vol. 2. New York: Van Nostrand Reinhold Company, 1972. 3, 11, 19, 80, 86

[4] P. Henrici, *Discrete Variable Methods in Ordinary Differential Equations*. New York: John Wiley and Sons, 1962. 3, 18, 89

[5] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*. Englewood Cliffs, New Jersey: Prentice-Hall, 1971. 7, 44

[6] R. L. Burden and J. D. Faires, *Numerical Analysis*. New York: Brooks/Cole Publishing Company, sixth ed., 1997. 8, 9, 43, 61

[7] J. Woodburn, "Mitigation of the effects of eclipse boundary crossings on the numerical integration of orbit trajectories using an Encke type correction algorithm," in *AAS/AIAA Space Flight Mechanics Meeting, Santa Barbara, CA, 11–14 February 2001*, (AAS Publications Office, P. O. Box 28130, San Diego, CA 92198), AAS/AIAA, 2001. Paper AAS 01-223. 10, 119

[8] J. L. Maury and G. P. Segal, "Cowell type numerical integration as applied to satellite orbit computation," Tech. Rep. X-553-69-46, NASA, 1969. NTIS #N6926703. 10, 18, 20, 26

[9] NORAD, "Mathematical foundation for SCC astrodynamic theory," Tech. Rep. TP SCC 008, Headquarters North American Aerospace Defense Command, 1982. Ob-

tainable from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145, as AD #B081394. 10, 26, 36

[10] J. F. Frankena, "Störmer-Cowell: straight, summed and split. An overview," *Journal of Computational and Applied Mathematics*, vol. 62, pp. 129–154, 1995. 11

[11] R. N. Wallner, "Earth gravitational error budget initial report," Tech. Rep., Kaman Sciences Corporation, 1994. 15, 26

[12] J. Jackson, "Note on the numerical integration of $d^2x/dt^2 = f(x,t)$," in *Monthly Notes*, vol. 84, pp. 602–606, Royal Astronomy Society, 1924. 19

[13] M. Berry and L. Healy, "The generalized Sundman transformation for propagation of high-eccentricity elliptical orbits," in *Advances in Astronautics*, (San Diego, CA), American Astronautical Society, February 2002. AAS 02–109. 35, 36

[14] K. F. Sundman, "Mémoire sur le problème des trois corps," *Acta Mathematica*, vol. 36, pp. 105–179, 1912. 35

[15] T. Levi-Civita, "Sur la résolution qualitative du problème restreint des trois corps," *Acta Mathematica*, vol. 30, pp. 305–327, 1906. 35

[16] V. Szebehely and V. Bond, "Transformations of the perturbed two-body problem to unperturbed harmonic oscillators," *Celestial Mechanics*, vol. 30, no. 1, pp. 59–69, 1983. 35

[17] P. Nacozy, "The intermediate anomaly," *Celestial Mechanics*, vol. 16, pp. 309–313, 1977. 36

[18] E. L. Stiefel and G. Sheifele, *Linear and Regular Celestial Mechanics*, vol. 174 of *Die Grundlehren der mathematischen Wissenshaften*. New York: Springer-Verlag, 1971. 36

[19] S. Ferrer and M. K. Sein-Echaluce, "On the Szebehely-Bond equation. Generalized Sundman's transformation for the perturbed two-body problem," *Celestial Mechanics*, vol. 32, pp. 333–347, April 1984. 36

[20] P. L. Palmer, S. J. Aarseth, S. Mikkola, and Y. Hashida, "High precision integration methods for orbit propagation," *The Journal of the Astronautical Sciences*, vol. 46, pp. 329–342, October-December 1998. 36

[21] R. H. Merson, "Numerical integration of the differential equations of celestial mechanics," Tech. Rep. TR 74184, Royal Aircraft Establishment, Farnborough, Hants,

UK, January 1975. Defense Technical Information Center number AD B004645. 36, 61, 62, 81, 102

[22] C. E. Velez, "Notions of analytic vs. numerical stability as applied to the numerical calculation of orbits," *Celestial Mechanics*, vol. 10, pp. 405–422, 1974. 36

[23] F. T. Krogh, "Algorithms for changing the step size," *SIAM Journal on Numerical Analysis*, vol. 10, no. 5, pp. 949–965, 1973. 43, 44, 120

[24] F. T. Krogh, "Changing stepsize in the integration of differential equations using modified divided differences," in *Proceedings of the Conference on the Numerical Solution of Ordinary Differential Equations* (D. G. Bettis, ed.), vol. 362 of *Lecture Notes in Mathematics*, (New York), Springer-Verlag, 1974. 44, 86, 88

[25] L. F. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations*. San Francisco: W. H. Freeman and Company, 1975. 44, 51, 55, 56, 57, 58

[26] J. M. A. Danby, *Computing Applications to Differential Equations*. Reston, Virginia: Reston Publishing Company, 1985. QA 371.D258 1985. 57

[27] P. E. Zadunaisky, "On the accuracy in the numerical computation of orbits," in *Periodic Orbits, Stability and Resonances* (G. E. O. Giacaglia, ed.), (Dordrecht, Holland), pp. 216–227, D. Reidel Publishing Company, 1970. 61, 71

[28] H. L. Neal, S. L. Coffey, and S. Knowles, "Maintaining the space object catalog with special perturbations," in *Astrodynamics 1997 Part II* (F. Hoots, B. Kaufman, P. Cefola, and D. Spencer, eds.), vol. 97 of *Advances in the Astronautical Sciences*, (San Diego, CA), pp. 1349–1360, American Astronautical Society, August 1997. AAS 97–687. 63, 100

[29] K. Fox, "Numerical integration of the equations of motion of celestial mechanics," *Celestial Mechanics*, vol. 33, pp. 127–142, June 1984. 64, 102

[30] O. Montenbruck, "Numerical integration methods for orbital motion," *Celestial Mechanics and Dynamical Astronomy*, vol. 53, pp. 59–69, 1992. 64, 102

[31] L. G. Jacchia, "New static models of the thermosphere and exosphere with empirical temperature models," Tech. Rep. 313, Smithsonian Astrophysical Observatory, 1970. 65

[32] L. W. Johnson and R. D. Riess, *Numerical Analysis*. Reading, Mass.: Addison-Wesley, 1982. QA297.J63 1982. 66

[33] K. G. Hadjifotinou and M. Gousidou-Koutita, "Comparison of numerical methods for the integration of natural satellite systems," *Celestial Mechanics and Dynamical Astronomy*, vol. 70, no. 2, pp. 99–113, 1998. 69, 70

[34] P. E. Zadunaisky, "On the accuracy in the numerical solution of the n-body problem," *Celestial Mechanics*, vol. 20, pp. 209–230, 1979. 70, 71, 72, 74

[35] T.-Y. Huang and K. A. Innanen, "The accuracy check in numerical integration of dynamical systems," *Astronomical Journal*, vol. 88, pp. 870–876, June 1983. 71

[36] P. E. Zadunaisky, "A method for the estimation of errors propagated in the numerical solution of a system of ordinary differential equations," in *The Theory of Orbits in the Solar System and in Stellar Systems* (G. Contopoulos, ed.), (New York), pp. 281–287, International Astronomical Union, Academic Press, 1966. 71

[37] P. E. Zadunaisky, "On the estimation of errors propagated in the numerical integration of ordinary differential equations," *Numerische Mathematik*, vol. 27, no. 1, pp. 21–39, 1976. 71, 72, 74

[38] F. R. Hoots and A. Segerman, "Satellite ephemeris representation using hybrid compression," in *Proceedings of the 2002 AAS/AIAA Space Flight Mechanics Conference*, (San Antonio, TX), American Astronautical Society, January 2002. AAS 02–133. 75, 77, 121

[39] A. M. Segerman and S. L. Coffey, "Ephemeris compression using multiple fourier series," *J. Astro. Sci.*, vol. 46, no. 4, pp. 343–359, 1998. 76, 125, 126, 127

[40] O. Montenbruck and E. Gill, *Satellite Orbits*. New York: Springer, 2000. 80

[41] J. Lundberg, "Multistep integration formulas for the numerical integration of the satellite problem," Tech. Rep. IASOM TR 81-1, Center for Space Research, The University of Texas at Austin, Austin, TX, April 1981. 81, 88, 102

[42] F. T. Krogh, "A variable step variable order multistep method for the numerical solution of ordinary differential equations," *Information Processing; Proceedings of the IFIP Congress*, vol. 68, pp. 194–199, 1969. 87

[43] J. Lundberg, "Computational errors and their control in the determination of satellite orbits," Tech. Rep. CSR-85-3, Center for Space Research, The University of Texas at Austin, Austin, TX, March 1985. 119

[44] J. B. Lundberg, "Mitigation of satellite orbit errors resulting from the numerical integration across shadow boundaries," American Astronautical Society, 1996. AAS 96–408. 119

[45] J. Lundberg, M. Feulner, P. Abusali, and C. Ho, "Improving the numerical integration solution of satellite orbits in the presence of solar radiation pressure using modified back differences," American Astronautical Society, 1991. AAS 91–187. 119

[46] R. R. Bate, D. D. Mueller, and J. E. White, *Fundamentals of Astrodynamics*. New York: Dover Publications, 1971. 122

Vita

Matthew Muhlhauser Berry was born on October 14, 1977 in Baltimore, Maryland. After graduating from high school in Timonium, Maryland, he matriculated to Virginia Tech in August 1995. Matt completed his Bachelor of Science in Aerospace Engineering in May 2000, and his Master of Science in the same field in December 2002. Matt participated in the cooperative education program as an undergraduate and as a graduate student, employed at the Naval Research Laboratory in Washington, DC. Matt's work at NRL has been focused on research and development of space surveillance software used operationally by the Naval Network and Space Operations Command.

Go Hokies.